Redbooks Paper

Bill Seubert
Daniel Raisch

The Role of IBM System z in the Design of a Service-Oriented Architecture

Introduction

This IBM® Redpaper explores how an existing mainframe IT infrastructure and assets can work in the new on demand, service-oriented architecture (SOA) paradigm. IBM clients have many years worth of applications, some performing critical business functions, in their portfolios, and huge investments in mainframe computing. This paper explains how to exploit the features of IBM System z™ to transform existing applications to work in the SOA universe.

This paper does not attempt to define SOA in great detail. An overview of that and other related questions can be found at:

`http://www-128.ibm.com/developerworks/webservices/library/ws-soa-whitepaper`

There are a number of other good resources from which to obtain more detailed information about SOA and IBM's SOA strategy, as well. An especially interesting source is the IBM developerWorks® Web site at:

`http://www.ibm.com/developer`

There are links to SOA-specific resources and to IBM System z resources on the developerWorks home page. The developerWorks site also contains many whitepapers, tutorials, and software downloads that provide additional information related to SOA and the mainframe.

IBM SOA strategy and the SOA lifecycle

In late 2005, IBM announced a refined approach to implementing SOA, centering on what IBM calls the *SOA lifecycle*. The SOA lifecycle is the framework of IBM's SOA strategy. As Figure 1 on page 3 shows, the SOA lifecycle consists of four stages:

- **Model** - Use modeling tools to define the business process, at a business function level, and model the actual services that will be part of an assembled, composite application.
- **Assemble** - Assemble the individual services and write the code that is needed to implement the business rules for the application. Pre-existing services can be re-used, new services can be developed, or both.
- **Deploy** - Deploy the services to run-time environments, such as transaction management engines like WebSphere® Application Server, CICS®, IMS™, and so forth. Use integration components, primarily an enterprise service bus (ESB), to link together the various services needed for the composite application.
- **Manage** - Implement the management infrastructure for monitoring and managing the services and the service infrastructure. This includes not only IT management tools, but also business management and monitoring tools to measure actual business activities.

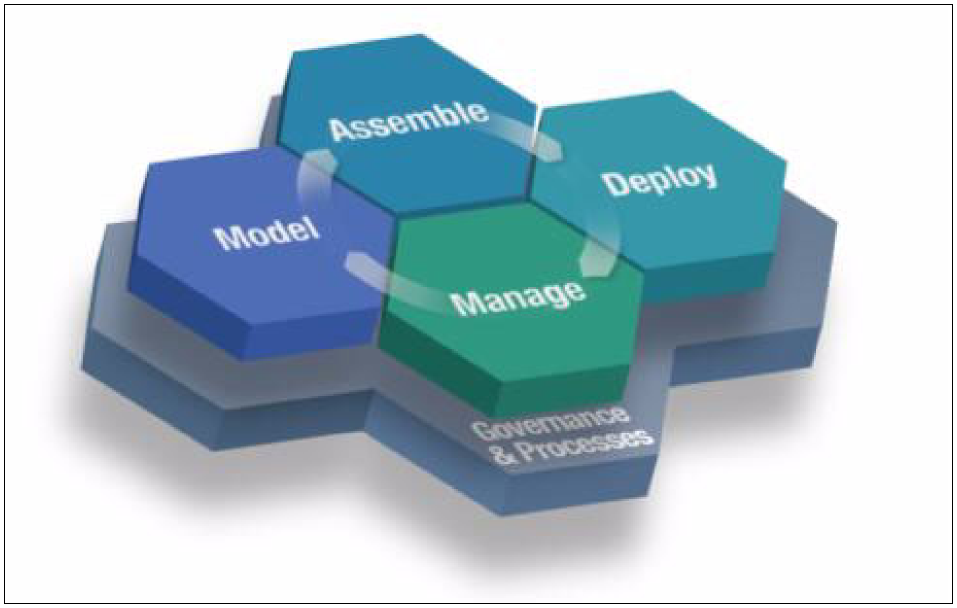


Figure 1 IBM's SOA Lifecycle

Since this is a lifecycle, it implies a *closed loop*, meaning that at the end of the cycle, the Manage stage feeds information back to the Model stage. Results from the run-time management tools can be fed into the modeling tools to provide feedback for refinement of the business processes that are being instantiated into services and composite applications.

Underlying the four phases of the lifecycle is the function of *governance*. Governance of the SOA implementation is critical to the integrity of the architecture. Governance can ensure the consistency of the service development strategy and adherence to the policies and procedures of the SOA implementation. This is often the area where enterprises stumble in their implementation. A "build it and they will come" approach to an SOA and the accompanying infrastructure is often a ticket to failure, unless there is constant monitoring, management, and enforcement of how the architecture is rolled out and used.

The SOA lifecycle is not an entirely new concept. Many application development models, including those for developing mainframe applications, have involved the same concepts. However, there are several aspects of SOA that make this particular lifecycle unique:

- SOA involves modeling of *business processes*, not just the modeling of the applications themselves. Developers are very familiar with the idea of object-oriented (OO) design and modeling. This involves the use of OO design tools, creation of UML models, and generation of application code from the models. In SOA, this technique is replicated at a higher level of abstraction. With SOA, the business process is modeled first, and the results of the model are fed to additional development tools in a fashion similar to that of UML models, only in this case the tools produce Business Process Execution Language (BPEL). The BPEL artifacts are then passed to other tooling that is designed to enable the construction of composite applications that consist of multiple, orchestrated services.

In the past, mainframe applications have been built using traditional processes such as the “waterfall” design method. Design techniques that include modeling at the business process level are closer to design methods such as the Rational® Unified Process®, which use design feedback loops with the end user. Business process modeling assists with that feedback process.

- SOA applications are assembled from multiple services that can be “wired” together using orchestration and process management tools. This is similar to how visual design tools such as VisualAge® Smalltalk were used several years ago for building object-oriented applications. However, with SOA, the assembly process is done using service components that are based on many different technologies, not just objects.

  Mainframe applications can be employed as services in the SOA lifecycle. This is discussed in more detail in “Integrating existing mainframe applications in an SOA” on page 27. Many mainframe customers use this application integration and service enablement process as their key “entry point” into SOA.

- In a full SOA implementation, the individual services are deployed to a run time, just like a “traditional” transactional system. However, the composite applications are represented in and “executed” from a flow engine, and that run-time engine takes care of calling the services in the proper sequence, based upon what is specified in the BPEL – the input to the flow engine. The advantage behind this construct is that the “rules” for the service flow can be modified more easily than in an application where the rules and workflow are coded into the application source. This is an important feature that helps enable the flexibility, reuse, and speed that are key attributes of the on demand model.

  In the past, many mainframe applications included process management within the application. Moving this flow management out of the application and into middleware components can make it easier to adapt to changes in business processes. Changes are made in the choreography tool rather than modifying existing COBOL source or other application program logic.

- While the logical linkage between the services is represented by the flows that execute under the control of a flow engine, the physical linkage between deployed services is typically implemented in an Enterprise Service Bus (ESB). The ESB is the abstraction layer that is designed to eliminate the point-to-point connectivity between specific services. The traditional message-oriented middleware implementation requires defined connections between endpoints, but, in an ESB, the endpoints are not explicitly hard-coded into the tools; they can be resolved at run time.

Enterprise application integration (EAI) has been a significant focus area over the last 5 to 10 years, and integration of existing mainframe applications has been the primary target. Many EAI principles and best practices still apply in an ESB-centered SOA, but the presence of standards such as messaging (WebSphere MQ, JMS) and SOAP have helped to make this job easier and faster than before.

- In previous application architectures (mainframe and other), monitoring is traditionally only done at the I/T level. Organizations are often concerned about the number of transactions per second being executed, what the end-user response time is, application server utilization, and so forth. In the SOA lifecycle, there is a keen interest in the business performance. For example, Bank A might be interested in how many new accounts were opened last week, or how many ATM withdrawals occurred, rather than what the average response time was for the ATM transactions. This kind of business information is fed into the business modeling tool to refine the business process. In fact, the business process modeling tools can often simulate a process model that will help to determine where improvements might be found. The input to the simulation tools can come from the actual run-time monitors.
- IBM’s sharp focus on governance is unique in this model. While architectural governance (including security) has always been an important facet of I/T and architectural management, it is particularly critical in SOA due to the loosely-coupled nature of services. In a world where applications are composed from services that are scattered throughout the enterprise and beyond, it is critical to have a firm grasp on the architecture and the implementation of services within it.[1]

  The mainframe environment has a long history of strong IT architecture and infrastructure governance and stringent application development practices. The focus on SOA governance is an extension of the same type of rigid policies that have succeeded in the traditional host-based world.

[1] More on SOA governance can be found at http://www-128.ibm.com/developerworks/webservices/library/ws-soa-govern/

More on reuse and governance

In the SOA lifecycle model, there are two elements that are crucial to SOA deployment success: *reuse* and *governance*. The construction of SOA-based applications through the assembly and choreography of existing services depends upon the *reuse* of assets, both those that are newly created and those harvested from existing mainframe (and other) environments. Reuse is one of the most important sources of SOA business value because it reduces development and testing time. It is closely tied to flexibility and agility since any new application to be deployed will primarily be the result of a new assembly of existing parts – a set of proven existing services choreographed to match a new business process. The key focus of reuse is to avoid building new services and incurring the associated cost of maintaining them.

Reuse must be supported by a governance infrastructure. There must be policies and best practices that define the development architecture that allows a service to be used by many applications, or to make it reusable. There should be a central repository where services are stored or cataloged, and mechanisms to make them accessible, similar to the topical listing in a phone book. Beyond this central repository, additional management requirements emerge, including versioning, tracking, and migration of services and their various releases. Mainframe shops generally have a good understanding of these concepts and requirements because a mainframe application development lifecycle has similar requirements, but with different scope and different technologies. Mainframe shops typically have a set of source code repositories and respective load module libraries, as well as other libraries for shared application code (such as error handling routines) and procedures and programs used by the IT department. Access to these repositories is controlled, and there are rules for code checkin and checkout, versioning, and other change management procedures. For mainframe shops, reuse and the controls that surround it are not new concepts.

As described here, there are many policies that must be defined in order to make an SOA infrastructure more robust. This set of rules, practices, and procedures is known as *governance*. Because SOA requires a corporate view of IT, it is essential that corporations define these rules and procedures. Governance is required along with technical SOA design considerations, and companies that are adopting SOA must develop a governance model that matches company business processes and their SOA deployment model.[2]

[2] For more details, see *SOA Compass*, IBM Press, ISBN 0131870025, Published October 25, 2005, Copyright 2006.

Choosing a development and design approach

One of the more interesting philosophical discussions in SOA is "Where do we start?" This is a particularly lively discussion when mainframe assets are involved, since few companies are willing to simply discard working applications that contain valuable intellectual capital. The general approaches to SOA analysis and design tend to center on three philosophies:

- **Top-down** - This is usually considered the "purist" approach to SOA. The top-down approach begins with analysis and understanding of the business environment and the services that make up the functions of the business. The business model is decomposed into the elementary components that represent business services. Those business services are eventually represented by IT services. This analysis is accompanied by documentation of the processes that link together the various business services.

  IBM's Component Business Modeling (CBM) can be followed to discover the business architecture. Once a model is developed, the strategically important parts of the business architecture can be identified and fed into the process of service modeling so IT services can be created from them.

  One drawback to the top-down method is that it may result in service decompositions that do not closely match the existing business applications. This can imply several things. For example, if the ideal model does not match existing assets, there may be a greater need for newly developed services. There is a fair chance that the match will be close—that existing assets *can* be used—but there may need to be more IT development work done to more invasively extract and reuse code.

- **Bottom-up** - The bottom-up approach involves exposing existing applications as services which are then used to create new, composite applications developed using techniques such as business modeling and development with process orchestration tools.

  When a bottom-up approach is used, concerns often emerge regarding the granularity of the services. With existing mainframe systems, an interesting situation often arises: older mainframe systems are frequently well-suited for SOA because the applications themselves already represent discrete business functions. A recent article in Enterprise Systems magazine stated:

  > *The irony is that host applications are probably better suited for exposure as part of an SOA than many applications based on more modern 4GL object-oriented languages, said Phil Murphy, a principal analyst with consultancy Forrester Research, in an interview last year. "When folks wrote screen-based transactions many months ago, they wrote it at a business function viewpoint: I add a customer, I add an order for that customer, I check backlogs for that customer, etc. So in many respects, those CICS screens of 15 years ago are better suited to service*

orientation than a lot of the newer, distributed code that's been written over the last several years, because of their affinity with a business function," he argued, adding: "What did the object-oriented guys do? They took those screens and they broke them down into a thousand different objects.[3]

So, at least at the user interface, the reuse of existing host applications is sometimes simpler than it would appear on the surface. Unfortunately, the underlying programming structure is not always as friendly to reuse as the application may appear at the surface. These applications may not match perfectly with the services that would be identified in an analysis exercise such as CBM. In this case, a compromise may be appropriate.

- **Meet-in-the-middle** - The "meet-in-the-middle" approach is a compromise that employs techniques from both top-down and bottom-up methodologies. It affirms that there is considerable value in existing assets that should be reused when appropriate, and also utilizes top-down service identification and decomposition techniques. A recent article on the IBM developerWorks Web site states:

  There are no green field projects in the real-world as legacy applications … always have to be taken into account. Therefore, a meet-in-the-middle approach is required, rather than pure, top-down or bottom-up process.

  The bottom-up approach tends to lead to poor business-service abstractions in case the design is dictated by the existing IT environment, rather than existing and future business needs. On the other hand, top-down processing might cause insufficient, non-functional requirement characteristics, and compromise other architecture quality factors (for example, performance problems due to lack of normalization in the domain model) as well as provide impedance mismatches on the service and component layer.[4]

  This summary shows the need to address SOA analysis and design from both the top and the bottom. In a mainframe environment, where there is a pre-existing inventory of mature, high-performance business applications, it only makes sense to try to reuse as much as possible.

  Figure 2 illustrates how top-down and bottom-up approaches relate. Business knowledge, issues, and architectures press from the top, and are represented using an analysis like CBM. The IT infrastructure, based on an on demand operating environment and SOA as an application architecture, provides the computing base for the modeling, development, deployment, and management of composite services that instantiate the processes from the business model.

[3] Source: http://esj.com/enterprise/article.aspx?EditorialsID=1457. Reprinted by permission of Enterprise Strategies (www.esj.com), a publication of 1105 Media LLC, Copyright 2006.
[4] Source: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/

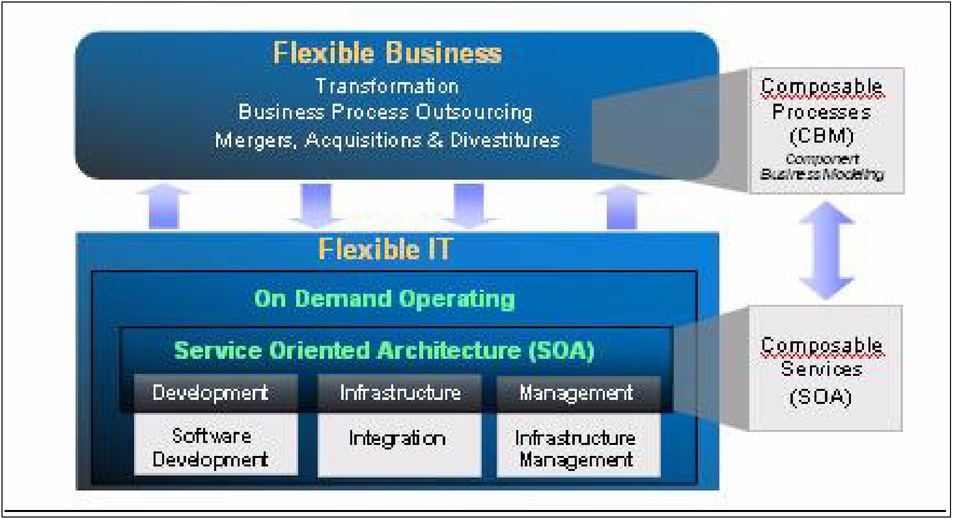

Flexible Business
Transformation
Business Process Outsourcing
Mergers, Acquisitions & Divestitures
Composable Processes (CBM)
Component Business Modeling
Flexible IT
On Demand Operating
Service Oriented Architecture (SOA)
Development
Infrastructure
Management
Software Development
Integration
Infrastructure Management
Composable Services (SOA)

Figure 2 Top-down/bottom-up relationship between business and IT

A further illustration of this is found in Figure 3, which shows how traditional architectures do not necessarily have the same degree of cooperation between the top and bottom.

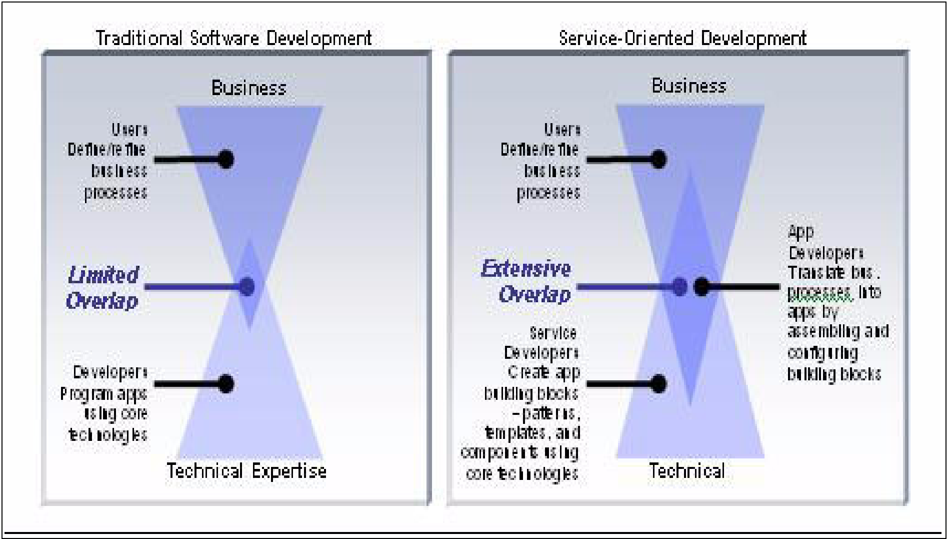

Traditional Software Development
Business
Users Define/refine business processes
Limited Overlap
Developers Program apps using core technologies
Technical Expertise
Service-Oriented Development
Business
Users Define/refine business processes
Extensive Overlap
App Developers Translate bus. processes into apps by assembling and configuring building blocks
Service Developers Create app building blocks - patterns, templates, and components using core technologies
Technical

Figure 3 Traditional versus SOA - top-down/bottom-up

Service-oriented architecture on IBM System z

The main focus of this paper is not to define service-oriented architecture or other terms related to it, but to clarify the role of the mainframe in a world centered on SOA. The natural inclination for many is to assume that SOA only involves distributed systems and technology that is less than five years old! However, this is far from the truth.

The 21st century mainframe can still trace its origins to the System/360™ of the mid-1960s. One of the key design points of the System/360 was to implement a computer architecture that would scale within the family and throughout a long upgrade lifecycle, without losing backwards compatibility. IBM's commitment to this level of compatibility has been maintained ever since. Because of this backward compatibility, many programs that were written decades ago will still run on current IBM mainframes. This has provided significant business value to our customers by allowing them to protect the investment in their applications. However, as the world has adopted new technologies and architectures, this has posed a problem: *how do we unleash the power of these business assets in the new computing architectures that have emerged in recent years?* This challenge began in the client/server days of the 1990s, and has continued into this decade. SOA is simply an extension of that challenge.

Standards in SOA

Fortunately, SOA possesses a key attribute that positions the mainframe as a full participant. The IBM developerWorks Web site refers to the following attribute of SOA:

The interface is defined in a neutral manner that should be independent of the hardware platform, the operating system, and the programming language in which the service is implemented.

Standards are critical to SOA, and specifically to the mainframe's participation in it. Through the standardization of interfaces between services, it is now possible for vendors (like IBM) to incorporate those standards into the transaction processing systems and database managers so existing applications can be integrated more easily into the SOA environment. There are a number of standards that enable this interaction, but several stand out:

- **Web services** - One of the most important standards to emerge in the last ten years is *SOAP*. SOAP defines a common XML format to describe service calls and return messages for invoked services. SOAP is a simple XML-based protocol to let applications exchange information over HTTP or messaging middleware.

  The emergence of SOAP as a standard transport has helped enable vendors to SOAP-enable their subsystems to make it easier for services to interact. The HTTP transport protocol is relatively ubiquitous, and as a result, SOAP messages can be passed between most computing platforms, providing the hardware and OS neutrality described in our SOA definition. Later in this paper, we present details about how the System z software has implemented SOAP and other standard interfaces to facilitate Web services calls to mainframe transactions and data.

- **XML** - XML is not a new technology; it has been in fairly wide use for more than five years now. It is central to the required interoperability between services in SOA. The self-describing nature of XML, along with the large number of utilities that process it, make it an ideal data representation format for information that flows in a service invocation. This pervasiveness extends to System z, where XML processing is possible in new technology server software such as WebSphere Application Server, and support for XML is also built into the traditional software systems, including CICS, IMS, and the COBOL and PL/I compilers.

- **Message-oriented middleware** - Messaging and queuing has been in existence for over ten years, providing a reliable, fast transport mechanism for passing data between applications. Many IBM customers have employed WebSphere MQ (formerly known as MQSeries®) as a transport to call remote functions on separate systems, just as SOAP does. In fact, WebSphere MQ can be used as the underlying transport for SOAP. IBM's WebSphere MQ (WMQ) is present in the business of many IBM customers world-wide, and this penetration has helped to make WMQ a de facto standard for implementing program-to-program communications. Some IBM customers have designed their entire SOA implementation around a WMQ transport. There is a key message here: *SOA is not just Web services!* A service-oriented architecture can be implemented without employing Web services, although this is uncommon. SOA simply must have a hardware- OS- and language-neutral transport to facilitate communication between services. In the case of IBM System z, WebSphere MQ is a key component of many customers' infrastructure on z/OS®, and there are bridges that permit the invocation of host transactions using its transport.

- **Java™ Database Connectivity (JDBC™) and Service Data Objects (SDO)** An emerging concept in SOA is *Information As A Service (IAAS)*. Besides accessing applications as services, composite applications may need to access data in a similar fashion. JDBC, along with the Structured Query Language (SQL), are key standard technologies for providing IAAS. Now any database management system can provide a JDBC driver interface to enable a Java-based SQL application to access information in a seamless, hardware- and platform-neutral fashion. Obviously, JDBC is inherently language-dependent – it requires Java! However, most database management systems now provide SOAP-based interfaces to the data resources. For example, technology now exists in DB2® Universal Database™ to access data in DB2 via Web services protocols.

  Service Data Object (SDO) is a newer technology than JDBC and is a key component of IBM's SOA programming model. An SDO *"define(s) a uniform paradigm of data graphs to access and manipulate data from heterogeneous sources, including relational databases, XML data sources, Web services, and enterprise information systems"*.[5] When using SDOs, a service developer connects to a *data access service* to access data, rather than using a language-specific construct such as JDBC. This provides abstraction between the data request and the data source.

The following sections focus on the specifics of the IBM strategy for SOA on the mainframe, and how the mainframe and z/OS can participate in and host the infrastructure for a service-oriented architecture implementation.

Design considerations for an SOA with IBM System z

There are many architectural principles involved in the design of an SOA infrastructure and applications that use that infrastructure. Some of these principles are not unique to environments with mainframes, but the presence of the mainframe can introduce certain special considerations. Some design principles that are of particular interest:

- **Existing IT standards** - This is the design principle that generally comes up first in architecture design sessions. These common statements: "We already use a particular product" "We are pursuing an open systems design for our systems" "We are trying to get off the mainframe" are the kinds of quasi-standards that are often encountered, in addition to more formalized standards documents. Existing IT standards may be dictated by the presence of zSeries® in the customer's environment. For example, if CICS is being used to host existing applications, then an infrastructure that permits easy integration of existing CICS business logic is usually desirable. "Placement of

[5] Source: IBM Systems Journal, http://www.research.ibm.com/journal/sj/444/ferguson.html

critical business data on an existing DB2 for z/OS system" is an IT standard that is often followed by mainframe customers.

- **Scalability** - Scalability is the ability to adjust the capacity of a system to absorb varying transaction rates, usually dynamically. Questions posed include "How can a particular system be scaled to take on additional work? Can this scalability occur dynamically, or does it require additional fixed resources (servers, storage, network, and so forth) to be added manually?" Often, scalability is referred to as *horizontal* (adding more capacity by adding additional systems or servers), or *vertical* (increasing capacity by growing the size of the existing platform).

  The IBM mainframe has a long heritage of scalability, both vertical and horizontal. The current generation of IBM System z9™, the System z9 EC, can scale from several hundred *millions of instructions per second*, or *MIPS*, to well over 17,000 MIPS, on one to fifty-four processor units, or *engines* in common terminology.

  Note: MIPS is only a very rough indicator of processor capacity. IBM has always stated that MIPS is a poor indicator of performance because different workloads consume processor resource at different rates. See `http://zjournalarchives.com/PDF/deitchoct.pdf` for an interesting article on the topic.

  If a System z customer requires more capacity than is available in a single mainframe, horizontal scaling may be accomplished by implementing a Parallel Sysplex®[6], which permits multiple z/OS systems to be clustered using a set of unique hardware and software technologies that provide horizontal scaling, full data-sharing between systems, and improved system availability by virtually eliminating planned and unplanned outages.

- **Availability** - No one wants their system to fail. *Availability* is a rather subjective concept, particularly in an SOA. What defines available? Is it availability to the end user? Availability of a particular infrastructure component? Does poor performance equate to unavailable? How good is good enough, with respect to availability?

  IBM mainframes have gained a stellar reputation over time by achieving very high levels of system and application availability, and by avoiding both planned and unplanned outages. The IBM System z9 is often referred to as having "five nines" availability (available 99.999% of the time, which translates to around 5 minutes of down time per year). This high availability is achieved through a variety of hardware and software features, including hardware component sparing, clustering via Parallel Sysplex, and

[6] Details on Parallel Sysplex can be found at
http://www-03.ibm.com/servers/eserver/zseries/pso/sysover.html

long-distance clustering with Geographically Dispersed Parallel Sysplex™.[7] Many distributed computing platforms claim to have mainframe-like features. However, the features of single system, intra-datacenter and inter-datacenter failover for System z are unique and unparalleled.

An SOA can deliver high availability by selection of high-availability run-time platforms, but the reduction of potential points-of-failure can deliver the same results. IBM System z helps facilitate the reduction of points-of-failure by enabling the architect to place more components on the mainframe platform, thereby helping to reduce the potential for failures on the network or on the system itself. Failover on a Parallel Sysplex is usually painless and seamless to the application.

- **Performance** - No one wants their transactions to be slow! Performance is one of the key considerations that drives design decisions: What components and software can deliver optimal performance? This is a much more interesting question in an SOA than in a traditional, single-tier mainframe application architecture since a multi-tier application has many more components that may pose performance issues.

  Several design principles may affect performance: proximity to data, reduction of network hops, fast computing platforms, reduction or optimization of I/O, and many others. The IBM System z delivers many of these high-performance features. By collocating components on the same mainframe or OS instance, or both, network activity can be reduced or eliminated. Marshalling/de-marshalling of objects in an RMI-style environment can be eliminated, thus removing the associated CPU and I/O overhead. The mainframe I/O subsystem is generally much more scalable and efficient than other platforms, helping to remove another potential overhead bottleneck for transactions. Middleware components such as CICS, IMS, and DB2 for z/OS have matured over decades and have been tuned to sustain very high transaction rates. Even if Linux® on IBM System z is a part of the architecture, proximity can be employed to improve performance by using a Hipersocket[8] connection between Linux and z/OS.

- **Workload management** - z/OS has long been known for its superior operating system functionality in the management of multiple, heterogeneous workloads. z/OS is designed to run—on a single OS instance—transaction processing (IMS, CICS, WebSphere), database management (DB2, IMS, ISV), batch, interactive work (TSO), systems management tools, portals, e-mail (Domino®), and many other types of work, simultaneously! The sophistication of the z/OS dispatching algorithms and the Workload Manager (WLM) make it possible for all of this work to run at the same time, without

[7] More details on Geographically Dispersed Parallel Sysplex (GDPS®) can be found at http://www-03.ibm.com/servers/eserver/zseries/gdps/

[8] For more information on Hipersockets: http://www-03.ibm.com/servers/eserver/zseries/networking/hipersockets.html

impacting other work running alongside it. Furthermore, when a Parallel Sysplex is implemented, work can be dispatched in the same manner across multiple z/OS instances in the sysplex. Hardware partitioning takes care of virtualizing CPU, memory, and peripheral channels so they can be shared across Logical Partitions (LPARs) on a single System z machine.

What does this mean? A z/OS system can be fully utilized to extract the full value of the machine. A zSeries/System z system is designed to run at 100% utilization, 24 hours a day with the workloads balanced in a way that preserves adequate performance based upon business objectives specified by the customer at the operating system level. This is true not only of z/OS, but also across z/VM® and Linux on System z as well, since virtualization happens at the hardware and the OS level.

The implications of mixed workloads and virtualization in SOA are interesting. As we asserted in the Performance topic, collocation of data and transactions is important for maintaining good levels of performance. Since z/OS can support multiple heterogeneous workloads, this collocation works well and should not pose performance problems. The various SOA infrastructure components can be run alongside the services themselves and the data being accessed by the services, helping to optimize performance and minimize points of failure. Leveraging the underlying qualities of service of the operating system by using the z/OS infrastructure components can help optimize these interrelationships.

- **Security** - There are several areas of security that are of concern in an SOA: authentication, authorization, and privacy. The mainframe security infrastructure already exists to support all of these through a synergy of the hardware and software. System z hardware contains a number of security features that help make the platform inherently secure. For example, the Storage Protect Key feature of the hardware and operating system makes it virtually impossible for one user or address space to overwrite another. All System z machines come with built-in, on-board encryption engines to accelerate functions like SSL. z/OS ships with built-in Public Key Infrastructure software so customers can build their own Certificate Authority. The Resource Access Control Facility (RACF®) is the standard security facility for z/OS; it supports authentication and authorization for all system components.
- **Ease of use** - The mainframe has a “green screen” reputation. For years the 3270 terminal interface has been the standard way to gain interactive access to mainframe applications. But, this is no longer the only way to access the mainframe. New development tool options, such as WebSphere Developer for zSeries, allow traditional application developers to build COBOL and PL/I applications using a workstation interactive development environment (IDE). Monitoring utilities such as the Tivoli® Omegamon suite provide an attractive GUI for performance and availability monitoring. And if existing 3270 user

interfaces to applications are inadequate, the Host Access Transformation Services (HATS) tool provides a very fast, simple way to wrap transactions with an attractive Web face. (See "Improve" on page 30 for more information about HATS).

The "New Face of z/OS"[9] initiative is intended to provide friendlier ways to access and administer the mainframe, and has already provided innovative ways to present documentation and training materials to new z/OS users and administrators.

- **Flexibility** - The mainframe will probably not be the only platform in an SOA implementation. Placing components on the IBM System z does not restrict a customer to that platform. There is a great deal of flexibility in placing components where they best fit, based upon the aforementioned characteristics. Use of standards in service creation and SOA infrastructure help to provide flexibility in placement of the services themselves, and of the components, such as a portal or an enterprise service bus.

  Most major middleware from IBM is supported on the Windows/Linux/UNIX® environment *and* on z/OS and Linux on IBM System z. And while there is somewhat limited support for the IMS and CICS transaction management APIs outside z/OS, there is the option to create transactions for both using Java, and to access those transactions using Web services. These features help to make the code more portable than it would be otherwise, and make it possible to swap out native CICS or IMS Web services with other alternatives, should that need arise.

- **Politics** - No one wants to admit that politics or ideology plays a role in technical decisions, but it does. Everyone harbors some natural preferences– customers, vendors, and consultants. And those preferences drive decisions.

  Mainframes are associated with many myths and urban legends, such as "the mainframe is too expensive," "the mainframe is just an old box for running batch jobs," "we can't find anyone to write COBOL," and more. The key to making an architectural decision for SOA with respect to the mainframe is: *Go beyond the myths and make decisions based upon business issues.* Among the questions you should be asking are: Where is the best platform to provide optimal performance/security/reliability, based upon the SLA for this application? What is the most cost-effective solution (looking beyond the cost of acquisition to the true total cost of ownership)?

  When decisions are backed with facts and good business cases, political influences are reduced, although not eliminated.

[9] Further information on the z/OS Ease of Use project may be found at: http://www-03.ibm.com/servers/eserver/zseries/zos/eou/

The three facets of IBM's SOA on System z strategy

When examining the various parts of the IBM software portfolio and strategy, three facets of the IBM software offerings stand out with respect to SOA:

- Deployment of new Web services to IBM transaction managers on System z
- Placement of SOA infrastructure components on System z
- Integration of existing z/OS transactions into the SOA as services

We now look at each of these in detail.

Deploying services to an IBM System z runtime server

One of the key strengths of the mainframe and z/OS is transaction processing. The ability of the mainframe to process large amounts of data simultaneously, with mixed workloads (thousands of transactions per second, as well as database managers, security managers, Web serving, and so forth, all on the same OS image), is well known in the IT industry. For more than 30 years, CICS and IMS have been hosting mission-critical applications for virtually all of the Fortune 500 companies and most major non-commercial enterprises. The mainframe is designed to deliver the best quality of service (QoS) in the industry for commercial transaction processing. Beginning in the late 1990s, Web application serving also took off on the z/OS platform, with the emergence of the WebSphere Application Server for z/OS. Now the mainframe is also a premier platform for serving Java 2 Platform, Enterprise Edition (J2EE™) compliant transactions.

Many services in an SOA are today's business transactions. A service may be akin to a subroutine, or it may be a consolidation of many different transactions (as in a composite SOA transaction) into a single orchestrated business process. The nature of *services* in SOA is the same as *transactions* – they are a key part of a business transaction and should be hosted in a transaction manager that provides a high level of QoS. Services that update critical business data have the same requirements to run that transactions have, with the same atomicity, consistency, isolation, and durability (ACID) attributes that any traditional transaction must possess.

Java transaction management

Since a service often must be transactional, it makes sense for transaction management middleware to host the service. In fact, most SOA implementations outside the mainframe realm do use transaction managers for this purpose. The WebSphere Application Server is IBM's primary transaction management container on distributed platforms for services written in the Java programming language. WebSphere is the market leader for J2EE-compliant application servers. On the mainframe, the WebSphere Application Server for z/OS provides that function. If a company wishes to build J2EE-compliant services with a Web services (SOAP) interface, WebSphere Application Server for z/OS fulfills that role. WebSphere Application Server for z/OS provides the same J2EE functions and APIs as distributed versions of WebSphere Application Server. All WebSphere Application Server products are now written to a common code base, and WebSphere Application Server for z/OS benefits from that. Over 90% of the WebSphere Application Server code is common across platforms, and the portion that is *not* common on z/OS actually provides the superior QoS for the WebSphere Application Server for z/OS product. It allows WebSphere Application Server for z/OS to exploit the underlying z/OS features, including: System Access Facility (SAF), the security interface to z/OS; Resource Recovery Services (RRS), the z/OS component that implements a two-phase syncpoint coordinator; and Workload Manager for z/OS (WLM), the operating system component that manages workloads on z/OS in a business-goal oriented manner.

Traditional transaction management

The WebSphere Application Server is not the only transaction manager on z/OS that can host services in an SOA. The two long-established transaction managers, Information Management System (IMS) and Customer Information Control System (CICS), are also able to serve transactions that cooperate directly in a service-oriented environment. IMS and CICS both have over thirty years of heritage in supporting high volume, high reliability transactional applications, and that environment is perfect for services today that require those same high qualities of service.

The traditional transaction managers support development in their native APIs, using traditional compiled programming languages, including COBOL, PL/I, C, and Assembler. However, both IMS and CICS also support the Java programming language for development of transaction programs and services. In IMS, this feature is known as the IMS JDBC Connector, although it can be used for more than just Java Database Connectivity (JDBC) connections. More details regarding this can be found at:

http://www-306.ibm.com/software/data/ims/imsjava/javapi.html

Applications can be written in the Java language that access IMS databases, using the JDBC protocol. Also, those Java applications can perform other IMS functions such as calling transaction commit and rollback services, communicating with IMS message queues, and calling IMS XML DB services.

An analogous API exists in CICS; it is known as the CICS Java API, or JCICS.

Like the IMS API, this feature allows a Java programmer to create an application which accesses CICS functions using the Java language and the JCICS interface. JCICS functionality encapsulates most CICS functions, such as the traditional commit/rollback functions, CICS terminal control, file control, and other key areas of CICS. For more information on JCICS, see:

`http://www.redbooks.ibm.com/abstracts/sg245275.html?Open`

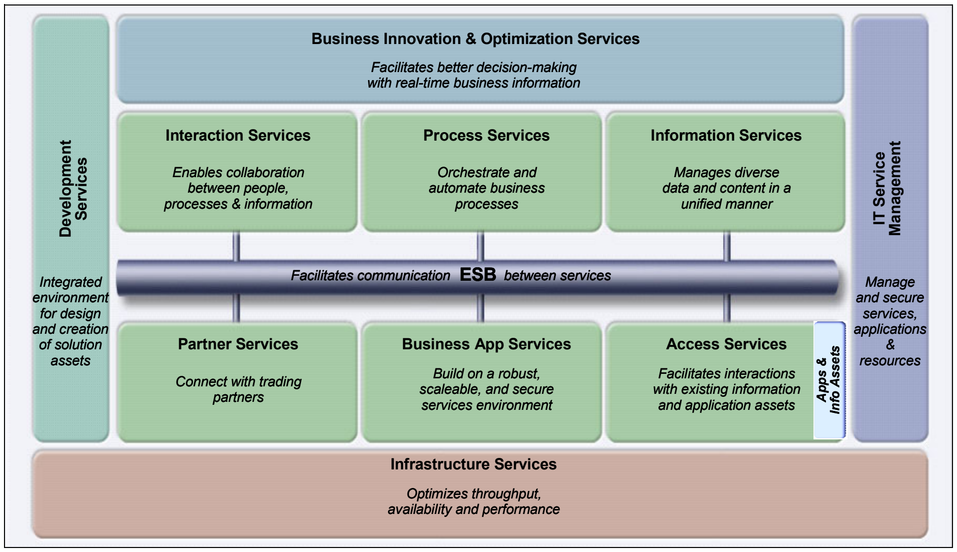
There are features within IMS and CICS to enable SOAP and other Web services protocols, independent of Java. In "Integrating existing mainframe applications in an SOA" on page 27 we examine how IMS, CICS, and DB2 transactions can be enabled to support SOA and Web services. The Java functions provide the ability for native transaction development inside these traditional systems. This is important in expanding the potential skill pool for developing transactions and services. Since many current college and university graduates have skills in Java development, support for Java in the subsystems helps to enlarge the potential CICS and IMS developer community. Skills in heritage programming languages are no longer necessary for building applications in these environments, enabling wider use of IMS and CICS in SOA.

Hosting the SOA infrastructure on System z

Service-oriented architecture, in its purest form, requires very little heavyweight infrastructure because the nature of service orientation simply dictates a logical structuring of application code that encapsulates business functions as services. The definition provided in the Appendix makes no reference to any kind of middleware or other supporting software. It simply refers to the standardization of the service interface and neutrality of platform. However, in large SOA implementations, it is not really practical to implement without tools to assist in the connectivity and the orchestration of the interaction between services. Also, other ancillary tools are necessary in most SOA implementations, including development and test tools, user interface tools such as portals, and monitoring and management utilities. This section describes why IBM System z is an ideal platform for running these tools and middleware products that are commonly used in the implementation of an SOA.

As organizations adopt SOA as the guiding architectural framework for development of enterprise applications, the newly deployed services quickly become business-critical components of the application infrastructure. This implies that services must be treated as such, and should be deployed on a robust, scalable, secure, high performance platform. Along with the services, the SOA infrastructure middleware and tools should also reside on such a platform. The mainframe, z/OS in particular, is the premier IBM computing environment for providing ultra-high qualities of service for enterprise applications.

What are the key SOA infrastructure components? The IBM SOA reference architecture, shown in Figure 4, defines the necessary building blocks to support an SOA environment.



Business Innovation & Optimization Services
Facilitates better decision-making with real-time business information
Development Services
Integrated environment for design and creation of solution assets
Interaction Services
Enables collaboration between people, processes & information
Process Services
Orchestrate and automate business processes
Information Services
Manages diverse data and content in a unified manner
IT Service Management
Manage and secure services, applications & resources
Facilitates communication ESB between services
Partner Services
Connect with trading partners
Business App Services
Build on a robust, scaleable, and secure services environment
Access Services
Facilitates interactions with existing information and application assets
Apps & Info Assets
Infrastructure Services
Optimizes throughput, availability and performance

Figure 4 The IBM SOA reference architecture

At the center of this reference architecture is the Enterprise Service Bus, or ESB. Surrounding the ESB are the key services needed to support the SOA runtime environment: interaction services, process services, information services, partner services, business application services, and access services. In the following sections we touch briefly on each of these to expand upon what they do to support the SOA. Along with those core services, the surrounding boxes describe the other functions that are necessary to support the services before, during, and after their deployment.

Infrastructure services

At the base of the SOA is the platform for deployment. This includes the deployment of hardware and software for the actual business services and service infrastructure. A production environment usually requires the highest degree of quality of service (QoS), and the infrastructure services component provides the QoS. Services at this layer can include operating system functions, security, and hardware functions. This requirement for a high level of service is where System z plays such an important role. Considering the mission critical nature of most production SOA and infrastructure implementations, the high degree of scalability, reliability, and security of the mainframe is key to providing robust infrastructure for the applications deployed in the SOA. For example, the z/OS Security Server, including RACF, is considered the premier security implementation on any platform[10]. Resource Recovery Services (RRS) provides a native z/OS-based transaction syncpoint manager for support of two-phase commit transactions that access multiple z/OS back-end systems.

Linux on System z also participates in this component. For customers that wish to use Linux as their primary run-time operating system, most other components of the SOA infrastructure can be hosted on a Linux on System z base and provide some of the same QoS advantages that are gained from running on z/OS.

Development services

The Model and Assemble phases of the SOA lifecycle make extensive use of the d*evelopment services* portion of the architecture. Development services include the tools that are used for the modeling and assembly of the business services. Modeling consists of the modeling of the business process and the modeling of the actual services and the business logic within them. The tools may be higher level tools suitable for business analysts and architects, or lower level tools such as those used for object-oriented development. The output from these tools consists of artifacts such as UML models, actual application source code in a variety of languages from Java to COBOL, and also markup languages including XML and Business Process Execution Language (BPEL).[11]

From the System z perspective, most code development and modeling tools don't run on the mainframe, unless you count ISPF and TSO. However, if the client wishes to build new services or reuse existing services already on the mainframe, tooling such as the WebSphere Integration Developer or the WebSphere Developer for zSeries are used to build and integrate services that will be deployed to the mainframe. For modeling, the development services tools available include WebSphere Business Modeler for modeling high-level business processes, and the Rational Software Architect for creating UML models of services and other components. The artifacts produced by these tools are deployed to the run-time servers that host and orchestrate the execution of the business services.

[10] See http://www-03.ibm.com/servers/eserver/zseries/zos/racf/ychooseracf.html for details.

[11] This is not intended to be a detailed description of technologies such as BPEL. For more information, see http://www-128.ibm.com/developerworks/library/ specification/ws-bpel/

IT service management

The SOA infrastructure components consist primarily of middleware software. Like traditional application architectures, an SOA implementation requires adequate levels of monitoring and management of the middleware and applications to ensure proper levels of system performance and security. The *IT service management* component in the SOA reference architecture includes those monitoring, management, and security tools. This is particularly important in an SOA-based system because the infrastructure required to support an SOA is significantly more complex than traditional mainframe transaction systems like CICS, which are relatively self-contained. The SOA has more "moving parts," making the monitoring and management tasks more complex. For example, a composite application in an SOA may span several computing platforms and networks, and may require security and transactional contexts to be carried from one server or operating system to another. Debugging errors and correcting performance problems in such an environment requires good monitoring and management tools.

Many tools in the Tivoli portfolio provide IT service management functionality. A few examples: for end-to-end performance monitoring, IBM Tivoli Composite Application Manager for SOA and the Tivoli Omegamon monitoring tools provide comprehensive monitoring of all the components in a multi-tier application that spans distributed and mainframe systems. For security across SOA components, Tivoli Access Manager and Tivoli Identity Manager provide functionality to give access control and authentication for SOA applications. IT service management encompasses far more than is listed here. More information on IT service management, including its relationship to the IT Infrastructure Library, can be found at:

http://www-306.ibm.com/software/tivoli/features/it-serv-mgmt/index.html

Business innovation and optimization services

While the IT infrastructure is monitored and managed through the IT service management components, what about the business services? An organization can ensure proper levels of system and application performance, but what about business performance? The *business innovation* and *optimization services* component provides the functionality to effectively monitor and record what is going on in the enterprise from a purely business perspective. How many <*widgets*> were sold in the last hour/day/month/year? How long is it taking to execute the process to service a customer? How many customers did a particular team serve yesterday? This component provides the tools necessary to monitor these kinds of business functions, report the results, *and* feed results to the modeling tools (used in the development services component) so processes can be modeled more accurately, simulations can be run, and modifications can be made to business processes to further improve the execution of the business.

The WebSphere Business Monitor provides a monitor that watches business services and accumulates and reports statistics about how the business functions themselves are performing. Again, this is not a question such as “How fast is this transaction executing?” Instead, it asks and answers “How many of these business functions are we performing and how efficiently are we doing them?”

Interaction services

The *interaction services* component provides the user interface to the SOA application. A key principle of SOA is the abstraction of application layers. In this case, the application’s user interface (UI) is exposed in a separate layer from the business logic. Interaction services are commonly thought of as the portal layer, since the UI for many SOA applications is provided by a portal, although this component is not mandatory. The portal not only provides abstraction for the UI, but it also provides a standard set of services to give the end-user a customized, personalized user experience. With new standards such as Web Services for Remote Portlets (WSRP)[12], portal applications (portlets) can be more easily bound to services to ease the integration between the UI and the business logic service.

WebSphere Portal Server is the primary component for providing interaction services. WebSphere Portal Server is available on either z/OS or Linux on System z (as well as on distributed platforms). As mentioned previously, in an SOA design, proximity to services, transactions, and data is a primary architectural design principle. Placing a portal on-platform, close to the services being invoked, reduces communication overhead and points of failure. However, over the last few years it has become commonplace to put user interface components, including fixed content HTTP servers and portal servers, on distributed platforms. WebSphere Portal Server provides both options, as it is platform agnostic.

Process services

A key attribute of an application in an SOA is that it is often a “composite application,” meaning one that is constructed from several discrete services, all connected via some sort of orchestration engine. The *process services* components provide the orchestration and workflow services that are required to meld multiple services into a single, composite business application. This

[12] See the WSRP Oasis standard for further information: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf

application may be a single transaction, or it may be a series of transactions joined together into a business process.

The WebSphere Process Server (WPS), and to a lesser extent, the WebSphere Message Broker (WMB), act as process choreography servers. WPS provides orchestration of multiple services, driven by a “script” expressed in Business Process Execution Language (BPEL). WPS is the “BPEL run time.” WMB executes “message flows” triggered by inbound messages, and can invoke multiple functions and services. WMB would be used to aggregate services into a single, short running transaction, whereas WPS can be used to run short or long running, workflow-oriented transactions.

z/OS and Linux on System z can be used to host the process services components on the mainframe. Placement of these components on z/OS or Linux on System z satisfies the architectural principle of proximity to data and transactions.

Information services

Data is everywhere. Over decades, enterprises have collected vast amounts of data in various repositories around the business. The information services component provides SOA-based access to the data repositories through techniques such as accessing stored procedures as Web services, providing standardized interfaces to non-relational data repositories, and other access mechanisms that return *Information As A Service* (IAAS). Since data or information itself isn’t “executable,” infrastructure is required to expose that data to applications that use SOA standards like SOAP.

Much of the critical heritage data in the enterprise lives on the mainframe. IBM provides a number of System z-based tools to expose IAAS, including the DB2 UDB for z/OS engine itself, and WebSphere Information Integrator, Classic Federation for z/OS (IICF). IICF provides an SQL interface to a number of heritage data formats, such as VSAM, IMS DB, and third-party databases including CA-Datacom, CA-IDMS, and Software AG’s Adabas.

Partner services

The *partner services* component is the interface to outside business partners. Exposing enterprise services to the outside world and invoking external services poses challenges to the integrity and security of the SOA. Services must be exposed in a manner that maintains the security and scalability of the service, since this makes the usage patterns for the service much less predictable and controllable than if they were being accessed exclusively in house. Historically, much of the partner interaction has been done through mechanisms such as EDI.

Partner services are provided on the mainframe via traditional channels such as the WebSphere Data Interchange, for EDI translation. External interfaces, such as those provided in partner services, are usually exposed via distributed systems that are dedicated to these kinds of functions. The WebSphere Partner Gateway products serve as the face to the outside world for services and can be hosted on a variety of distributed servers, including Windows®, UNIX, and Linux.

Business application services

Newly developed services reside in *business application services* components. These new services are generally deployed on servers such as IBM's WebSphere Application Server, which is a transaction manager for Java 2 Enterprise Edition (J2EE) applications. The WebSphere Application Server can be used for full-function applications (presentation, business logic, and data logic), or it can host business services that have the other functions abstracted to other portions of the reference architecture (see Interaction services and Information services).

WebSphere Application Server is available on both z/OS and Linux for System z on the mainframe. The Linux on System z version is identical to the WebSphere Application Server product that executes on distributed Windows and Linux platforms. However, the WebSphere Application Server for z/OS is slightly different. As mentioned in "Java transaction management" on page 17, WebSphere Application Server on z/OS exploits the underlying z/OS operating system functions, without sacrificing application portability, since there is a very high level (well over 90%) of cross-platform product compatibility.

Access services

Of particular interest to most mainframe customers is the *access services* component, which is the linkage to existing applications, both on and off the mainframe. The functionality here provides the connectivity to applications on IMS, CICS, SAP, PeopleSoft, and so forth, using various connector and adapter technologies.

The third facet of IBM's System z SOA strategy is integrating existing mainframe applications, and the access services component is directly related to this topic; detailed discussion is in the next section, "Enterprise service bus".

Enterprise service bus

While the *enterprise service bus* (ESB) is the last topic being presented, it is actually the critical item across all of the reference architecture. In Figure 4 on page 20, it is apparent that the ESB is literally at the center of the SOA reference architecture, and that it is important to any SOA implementation. The ESB provides the abstraction layer between the service requester and service provider. In older, non-SOA applications, linkages between applications are frequently hard-coded and may be difficult to manage and maintain. To change relationships between programs, application changes may be necessary as business or technology evolves.

IBM's definition of an enterprise service bus is that of an *architectural construct* rather than a specific product. The ESB should support four primary functions:

- *Routing* of messages between services, removing the direct one to one relationship between endpoints.
- *Conversion* of transport protocols between requester and service, for example, SOAP to MQ, FTP to EXCI, and so forth.
- *Transformation* of message formats, for instance, transformation of XML to binary.
- *Handling of events* from disparate sources. Events are received from the ESB endpoints and *correlated* to trigger new events based upon decisions in the ESB.

Any product or products that perform those requisite functions can be classified as an ESB implementation.

The WebSphere Enterprise Service Bus and WebSphere Message Broker products are the core ESB products in IBM's portfolio. They provide the four primary ESB functions, and when combined with the WebSphere Process Server for orchestration and WebSphere MQ for connectivity, IBM clients have a very robust infrastructure for an SOA. WebSphere Message Broker is available on z/OS and Linux on System z, and as of March, 2006, WebSphere ESB became available on Windows, Linux, and UNIX, and is planned on z/OS in June, 2006.

The System z platform is the ideal location to place an ESB. Most IBM clients have many existing assets and applications on the mainframe and, considering the principle of proximity to transactions and data, locating the ESB close to the assets makes sense. In addition, the high reliability, availability, and security of System z are key attributes for an ESB. The ESB is a critical component of the architecture, routing service calls for all SOA transactions. It does not make sense to put the most critical part of the architecture on a platform that does not possess the highest quality of service characteristics.

A more comprehensive discussion of the components of the IBM SOA reference architecture (also known as the *Integration Reference Architecture*) is found at:

http://www-128.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html

Integrating existing mainframe applications in an SOA

Most mainframe users have a keen interest in the method used to integrate existing applications into an SOA. This generally means that they intend to expose existing transactions or data via a service interface. Some IBM customers have been following SOA design principles for a long time but have not used current standards such as SOAP in their implementations.

Exposing applications as services is the primary goal of the bottom-up SOA design model. As mentioned previously, this approach has one inherent drawback: the existing applications may not have the granularity that matches the business service model developed through processes such as CBM. This can sometimes be corrected by means such as service aggregation, wrapping existing applications with a J2EE veneer, or other similar methods.

But if there are drawbacks to the bottom-up style, why pursue that rather than simply rewriting existing applications? A common concern is that existing applications are often written in supposedly “dead” languages like COBOL or PL/I, and clients should be eliminating them. However, many studies have shown that significant savings may be realized by reusing existing code rather than rewriting. An often-quoted statistic states that it is up to five times more expensive to rewrite an application than to reuse existing code to achieve the same functionality. Furthermore, with an estimated 200-250 billion lines of COBOL code in existence today (and no sign of that number declining) and potentially several billion new lines of COBOL being developed every year, programmers will be maintaining and developing new code far into the future. Reuse becomes a very attractive strategy to harvest existing code assets and save money during an SOA implementation.

IBM’s *enterprise transformation strategy* is a path to follow that will assist in integrating mainframe applications in SOA.

Enterprise transformation strategy: improve, adapt, innovate

As discussed previously, SOA focuses on the concept of reusability, which means that application components should be built in a way that facilitates reuse and avoids re-creation of components from scratch when existing assets could satisfy the requirements. The combination of existing data and applications on the mainframe and the concept of reuse through transformation leads to a fast start-up into the SOA model. This is the concept of *enterprise transformation*.

Enterprise transformation and application modernization are critical to an enterprise’s application development strategy since the high cost and risk of rewriting existing applications is often an inhibitor. If a company could reuse existing well-tuned and proven code that has been implementing business function satisfactorily for years, savings can be significant. Aside from the high cost and risk of migration of these applications, performance, reliability and scalability requirements often make the current environment the best choice. When considering reuse of existing assets, developers experienced in traditional languages must be considered an important part of the overall e-business development team. As availability of these development skills continues to tighten in the marketplace, it becomes more critical to create a development environment where existing assets can be maintained and extended using current development approaches and available skills.

SOA has particular importance in the enterprise transformation practice because it has the power to unlock existing applications and data and expose them as services, which provides extended value to the business. Considering the investments that companies have made in developing applications and the importance those applications have to the core business, service enablement using standards-based interfaces helps to extend the life cycle of those applications and leverages existing investments.

The move towards SOA is not going to happen overnight. It is not an appliance to be installed on top of an existing infrastructure, thus making it SOA ready. The implementation process is a set of granular changes applied to the existing architecture and the exploitation of new technologies that will gradually make the environment *SOA-ready*.

IBM's enterprise transformation strategy supports this gradual, granular process that will help to enable existing assets as services. It defines three solution frameworks, sometimes referred to as *styles of transformation*, to help customers convert IT assets from siloed applications to shared resources, and then to interdependent software components and services.

The strategic goal is to innovate by enabling the creation of new software components that have strategic business value and support an on-demand environment. But that takes time, effort, and money. So, more immediate (tactical) solutions are required as well. For example, the ability to transform siloed applications to shared resources can enable better ways of interacting with customers, partners, and suppliers. Some changes to applications will require immediate solutions that can drive business value, but may not be as flexible in adapting to changes in the business process. The three transformation styles - *Improve*, *Adapt*, and *Innovate* - address both tactical and strategic service enablement solutions.

While IBM identifies these three approaches for modernizing a client's enterprise, an IT organization will likely not use just one style. Businesses often choose multiple styles of modernization for different types of solutions, so the deployment scenario is determined case by case.

The discovery phase

A key question, therefore, is: “Where do we start?” A good place to start is in what is known as the *discovery phase*. Many existing applications lack documentation, or the documentation is out of date, or it is in someone’s head and they have left the company. In such situations, before modernization activities begin, it is necessary to find out the location of the application code, what it does, which applications and components it interacts with, and so forth. This is the discovery phase of the modernization process, and IBM has specific tools that support the activities in this phase. These discovery tools are designed to assist in documenting relationships among applications and sub-routines and creating visual representations that show asset relationships effectively and efficiently. These tools are intended to extract components and associated data items from existing code, simplifying the difficult and time-consuming task of building components manually. They can also perform impact analysis to understand data movement and application interaction during run time and record the results for use in future application development and test. Both static (code-based) and dynamic (run time-based) analyses are supported. The IBM discovery tools are:

- WebSphere Studio Asset Analyzer (WSAA) is a static analysis tool designed to assist in maintaining and extending existing assets through impact analysis, connector builder assistance, and graphical application understanding. WSAA is designed to help enterprise customers on their journey to e-business by providing knowledge about their static environment (finding and reusing application code and the components that connect that code). It also helps them to understand their dynamic environment (understanding what code is executing in run-time environments).
- CICS Interdependency Analyzer is a dynamic run time analysis tool that gives developers and operations staff information about CICS applications and their components. It is used by developers when trying to deconstruct and decompose an application into services and by operations teams in the analysis and movement of workloads across a distributed CICS environment.
- Asset Transformation Workbench is a workstation-based tool that is suitable for application re engineering, code extraction, and Web service generation. It supports traditional programming languages, such as COBOL, as well as some 4GL languages, like Natural.

Once a client understands existing business assets and the impact of changes to those assets, IBM’s development tools can be used to more effectively develop new e-business applications. These tools support extending existing applications to e-business without modifying existing code. This rapid development and deployment capability enables quick return on investment while more substantial development projects are under way that might involve creation of a more complete SOA. WebSphere development tools provide support for J2EE development and traditional language development, and they also support a mixed workload environment. Mixed workloads include the J2EE and Java support provided by WebSphere, and the back-end business applications written with COBOL and PL/I and running in CICS and IMS.

Transformation styles examined

As stated earlier, the three transformation styles of *improve*, *adapt*, and *innovate* cover both tactical and strategic service-enablement solutions.This section presents a more detailed look at these styles.

Improve

The *improve* style is characterized by the use of new technologies to Web- or service-enable applications at the *user interface* level, without changing those applications and with minor changes or additions to the middleware infrastructure. Improve is often the first step towards SOA, where clients simply enable existing applications with SOAP or MQ protocols to facilitate integration. This style concentrates on transforming the user experience by providing a more sophisticated and productive user interface for applications.

With Improve, companies can quickly enter the SOA model and achieve a rapid return on investment by extending existing applications using SOA standards (such as SOAP and XML), but without delving into all facets of SOA. This is a tactical approach intended to help clients achieve fast results with low investment, which leverages existing skills while preparing the infrastructure and honing the users' skills for more complex scenarios. Referring back to the three SOA development and design approaches, this follows a bottom-up approach to service enablement.

Product solutions that support the improve style

- IBM's Host Access Transformation Services (HATS) product is a common implementation tool for this phase. HATS has the ability to service-enable a 3270 or 5250 application by exposing it via the SOAP protocol, in addition to its traditional function of Web-enablement of host applications. HATS intercepts the application's data stream and converts it into either HTML or Web services formats. It is a WebSphere application and can be deployed on multiple server platforms, including z/OS and Linux on System z. To create HATS applications, the HATS Studio, an Eclipse-based application that runs within the Rational Application Developer family, is used to walk through the existing host applications and develop the Web or service interface. The HATS architecture is shown in Figure 5.

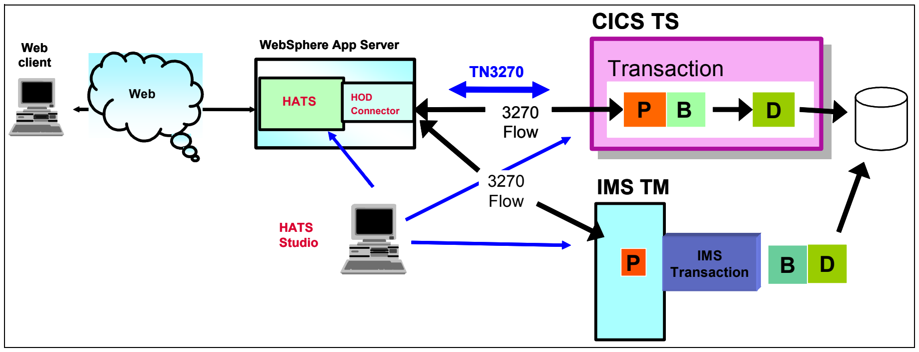


Figure 5 The HATS architecture

- The CICS Service Flow Runtime (CICS SFR), a new feature of CICS Transaction Server Version 3.1, provides a similar function to HATS, but it is self-contained within the CICS server. While CICS SFR does not provide the Web-enablement that HATS does, it does provide the choreography of CICS terminal transactions and can expose a single transaction or multiple orchestrated transactions as a service or services. The CICS Service Flow Modeler, a feature of the WebSphere Developer for zSeries, is the tool used to define the flows and service interfaces exposed via the CICS SFR.
- The WebSphere Portal Server. Another approach that provides an improve-type solution at the user interface is a *portal*. Portals can provide a new face to applications that were previously difficult to use or not integrated with others. In addition, new standards for SOA integration in portals, such as Web Services for Remote Portlets (WSRP), make it easier to include a portal in an SOA implementation. The WebSphere Portal Server is IBM's solution for providing a unified user interface for integration of SOA applications "at the glass."

Adapt

Adapt is a further step in sophistication beyond improve. It goes a bit farther in terms of application transformation and enhancement of the existing infrastructure. Adapt comes closer to full SOA architecture, but requires more investment in business componentization and in IT services. The result is more flexibility to the business and the value this represents.

Adapting existing connectivity enables broader application integration and provides the ability to incorporate core applications into more modern application flows. This allows customers to leverage existing applications to develop better customer, partner, and supplier relationships. But the connection of many different existing applications to new applications and new architectures poses a compatibility problem: data formats, communication protocols, and existing programs frequently cannot communicate without some sort of intermediary. So, the requirement emerges for an intermediate broker to be used for data mediation, protocol transformation, messages routing, and so forth. This functionality is where the requirement for an enterprise service bus (ESB) usually emerges. The ESB is not a direct component of the adapt transformation style, but it does serve as the hub to connect the applications, transactions, and services that are transformed using the adapt techniques.

Other elements of the SOA reference architecture are also involved in this phase, including information and access services. The service management infrastructure is often planned and designed during this phase.

Some examples are: a CICS application consumes services provided by a Microsoft® .NET application, where applications are to be SOAP-enabled and connected via the ESB for data transformation; an IMS transaction exposed as a service to be invoked by a WebSphere application through use of the Java 2 Connectivity (J2C) protocol; a DB2 table to be accessed via a Web service request as part of a composite business application.

There are many technical ways to implement the adapt style. Some of the common technologies used are:

- Native SOAP access to transactions or data - CICS provides a native SOAP interface in CICS Transaction Server V2 and V3 via the SOAP for CICS (V2) or CICS Web Services (V3) features. With these features, CICS can be a Web services provider *or* consumer. IMS Version 9 now includes the IMS SOAP Gateway,[13] which exposes IMS transactions to SOAP-based Web services consumers, but IMS cannot itself act as a Web service consumer - it is only a provider. Also, DB2 can expose its data to external SOAP requesters using the DB2 WORF[14] feature.
- Java 2 Connectivity (J2C) - As of IMS Version 9, the IMS Connect feature is now included with IMS Transaction Manager for providing direct invocation of IMS transactions from Java 2 applications. CICS transactions are accessed with J2C using the CICS Transaction Gateway, which serves as the intermediary between J2C and the CICS transaction.
- Messaging - WebSphere MQ and Java Messaging Services (JMS) can also be used to invoke transactions and access data from existing sources. Both IMS and CICS have bridge programs that permit a messaging-enabled caller to place a message on a queue that triggers the invocation of an existing IMS

[13] See `http://www-306.ibm.com/software/data/ims/soap/` for more information on this topic.
[14] `http://www-128.ibm.com/developerworks/db2/zones/webservices/worf/`

or CICS transaction. And DB2 can interact with the WebSphere MQ system via user-defined functions invoked through SQL.[15] Figure 6 illustrates how MQ or JMS can be used to access existing transactions.

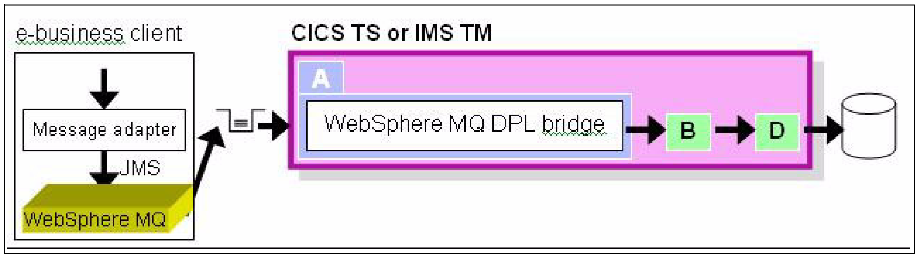


Figure 6   Messaging access to CICS transactions

- Information integration - A common requirement is to adapt connectivity to existing information sources. There are a number of federated data tools on the market. IBM provides the WebSphere Information Integrator Classic Federation for z/OS (IICF) product to build a consolidated view of disparate data sources. IICF provides a consolidated SQL interface to many different relational and non-relational data stores on z/OS, including DB2, VSAM, IMS DB, and several non-IBM database systems, including CA Datacom, CA IDMS, and Software AG's Adabas.

Product solutions that support the adapt style

– CICS Transaction Gateway

– IMS Connect (and equivalent functionality included in IMS Version 9)

– WebSphere Adapters

– WebSphere MQ

– WebSphere Information Integrator Classic Federation for z/OS

– SOAP for CICS / CICS Web Services feature

– WebSphere ESB

– WebSphere Message Broker

Innovate

The *innovate* style of modernization is characterized by the creation of new applications that are fully compliant with the SOA model. Accordingly, with reference to SOA strategic approaches, this style would be mostly top-down, where business processes are modeled using a modeling tool such as the

[15] See http://www-128.ibm.com/developerworks/db2/library/techarticle/wolfson/0108wolfson.html for more details.

WebSphere Business Modeler. Deployed applications would invoke a service or services (applications) that make up the composite business applications or processes. This may require that a totally new service or application be developed or it may involve the transformation and reuse of an existing application to meet the business requirements.

Transforming the application structure and architecture requires the highest degree of investment, but pays off with the greatest business value and process flexibility. In this Innovate style of transformation, core applications are restructured to provide the greatest amount of business benefit. This allows customers to more rapidly innovate and change their business processes using existing IT applications to create new and differentiated market solutions.

In order to innovate, tools are required to design and deploy new applications, and a server is needed to orchestrate and direct the newly created business processes. The WebSphere Process Server orchestrates the invocation of multiple services in the SOA by taking a process model that is represented in Business Process Execution Language (BPEL) and executing that model by directing the invocation of the various services in the composite application (flow). The flows are built using the WebSphere Business Modeler for high-level process modeling, and WebSphere Integration Developer for lower-level IT implementation of the process.

Another significant value that SOA brings to companies is an architecture that is programming language neutral, thereby allowing companies to continue to develop new applications using traditional languages, which leverages existing skills, tools, and investments. This is important to mainframe shops since senior application developers can continue to use preferred languages and tools, which provides improvements in productivity and economics.

However, there is room for improvement in productivity beyond the traditional tools and techniques. New development tools can be used to further improve the productivity of traditional language developers. The WebSphere Developer for zSeries (WDz) gives COBOL, PL/I and System z assembler developers an integrated development environment (IDE) for building host applications and improves productivity through more efficient editing, interactive debug tools, and helpers for developers who are not completely familiar with traditional language constructs. WDz can also be used for building newer Java applications, using the same familiar (Eclipse-based) IDE. WDz also provides the ability to build applications using the Enterprise Generation Language (EGL), which is a 4GL-style language that simplifies the coding task and can generate application artifacts in either Java or COBOL.

The discovery and code harvest tools described previously are also pertinent to the innovate style. These tools are used to identify existing code that can be reused when constructing new services and composite applications. The

WebSphere Studio Asset Analyzer and CICS Interdependency Analyzer can provide much assistance in identifying existing assets that can be extracted and reused in new SOA-compliant applications and services.

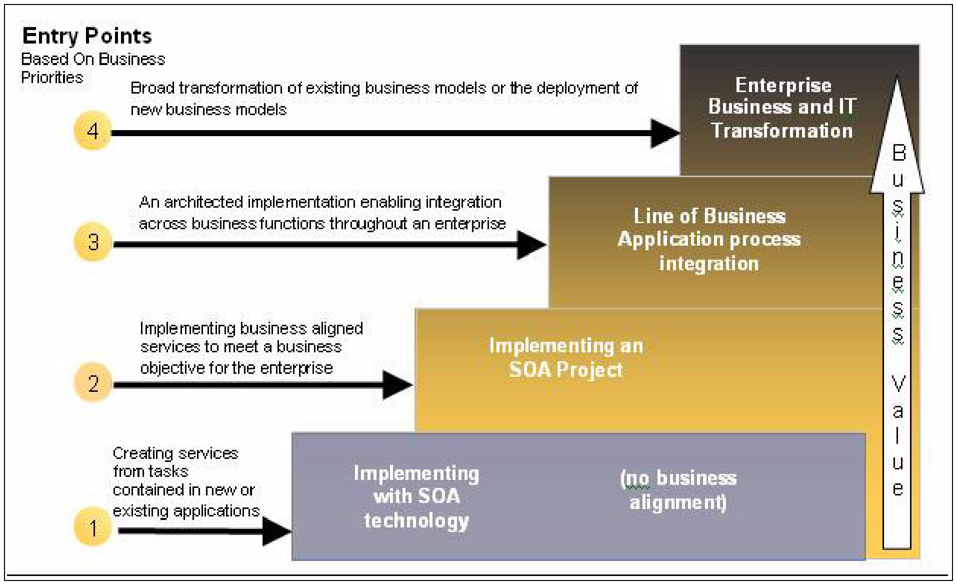
Product solutions that support the innovate style

- WebSphere Developer for zSeries
- WebSphere Business Modeler
- WebSphere Integration Developer
- WebSphere Studio Asset Analyzer
- CICS Interdependency Analyzer

The SOA maturity model and System z

After examining the principles of SOA and the technologies that are involved in its implementation, one might be tempted to believe that SOA is too complicated, and decide to stick with existing architecture and infrastructure. However, it would be a mistake to assume that an organization must leap directly to a full-scale SOA implementation all at once. IT organizations are at many different levels of maturity, and each will take a different path to a fully realized service-oriented architecture.

IBM has identified various “entry points” for building an SOA. Figure 7 on page 36 shows these entry points and how they involve increasing complexity of implementation, as well as increasing business value, from bottom to top. A client can choose to jump in at any of the entry points.



Entry Points
Based On Business Priorities
Broad transformation of existing business models or the deployment of new business models
4
An architected implementation enabling integration across business functions throughout an enterprise
3
Implementing business aligned services to meet a business objective for the enterprise
2
Creating services from tasks contained in new or existing applications
1
Enterprise Business and IT Transformation
Line of Business Application process integration
Implementing an SOA Project
Implementing with SOA technology
(no business alignment)
Business Value

Figure 7 SOA entry points

IBM is not the only company or organization to identify differing levels of maturity in SOA implementation. CBDi, a consulting firm in the UK that specializes in SOA and Web services, has also identified a four-step path to SOA maturity[16]:

- Phase 1: Early Learning - Exploratory activities center primarily on application integration. Service deployments are low-risk and primarily internal to the organization.
- Phase 2: Integration - Begin to consider business drivers in the SOA implementation. Still internally-focused, but concentrated more on business processes than in the Early Learning phase.
- Phase 3: Re engineering - Move to the enterprise level in scope. Focus on management, measurement, and monitoring of services. Centered on “business product” thinking, where the service becomes the business product.
- Phase 4: Maturity - Ubiquitous, federated services. Many providers of the services, both within and outside the enterprise.

System z customers who are working on a bottom-up approach to SOA are often very interested in the technical aspects of SOA. Their maturity can be accurately defined by other more IT-focused maturity characterizations. David Linthicum, an IT consultant and author of many books on Enterprise Application Integration and SOA, wrote an article that was published in the SOA WebServices Journal in

[16] Source: http://roadmap.cbdiforum.com/reports/maturity/maturity2.php

late 2004[17] that does an excellent job of describing the various levels of maturity that we often see in our zSeries SOA engagements. Linthicum categorizes the levels of maturity as follows:

- Level 0 SOA: Sends SOAP messages from system to system
  - Leverages Web services technology for integration
  - No real notion of services
- Level 1 SOA: Level 0 + messaging/queuing system
  - A rudimentary ESB that moves information via queues
  - Still no real notion of services, although the messaging interface resembles a service interface
- Level 2 SOA: Level 1 + transformation and routing
  - A more complete ESB that enables a higher degree of abstraction between services
- Level 3 SOA: Level 2 + a common directory service
  - An ESB-centric architecture that enables dynamic binding of service interactions through a business service directory
  - Often includes directory standards such as LDAP or UDDI
- Level 4 SOA: Level 3 + brokering and managing true services
  - More dynamic connectivity between services, enabled by the directory service
  - Also includes more robust management of the service architecture/infrastructure - service discovery, access, and management
- Level 5 SOA: Level 4 + process orchestration
  - Enables the creation of composite applications (meta-applications) to solve business problems
  - Addresses problems of persistence and user interaction; should provide a mechanism for services to interact with users via portals

This progression represents many of the System z customers that are involved with building an SOA. For mainframe customers, we often simplify Linthicum's model with the following maturity model:

1. Service enablement: Use integration technologies to expose mainframe transactions as services.

[17] Source: Reprinted by permission of the publisher, from http://webservices.sys-con.com/read/47277.htm , Published Dec. 2, 2004
Copyright © 2006 SYS-CON Media. All Rights Reserved.

2. Service integration: Use ESB technologies to provide the integration abstraction layer that links the services. Tools most often used at this stage include WebSphere Message Broker, WebSphere ESB, and the underlying transports. This step often includes information (data) integration and would bring in tools such as the WebSphere Information Integrator Classic Federation for z/OS.
3. Process integration: Use process choreography/orchestration tools and technologies to link services into composite applications and business processes. Tools often used here are WebSphere Process Server and WebSphere Portal Server for user interaction.

The notion of a maturity model for SOA provides a useful framework for assessing the sophistication of an SOA implementation. Clients should realize that it is not necessary to start at any particular point in the progression. IBM's SOA Entry Points approach does a good job of demonstrating this - an enterprise can enter at any points in the hierarchy and expand their SOA implementation over time. The simplification of the Linthicum Levels model provides a very basic way to explain the major inflection points in an SOA infrastructure implementation.

Conclusion

We have examined the basics of SOA and how a customer might approach placing services on System z, hosting SOA infrastructure on System z, and integrating existing mainframe applications through Enterprise Transformation. System z and z/OS bring the following key strengths to an SOA implementation:

- Security
- Reliability
- Scalability
- Reduced cost of ownership
- Reuse of assets

The vast majority of IBM's software portfolio is fully supported on System z, and takes advantage of the key strengths identified here. Figure 8 shows how the products on System z fit into the SOA reference architecture.

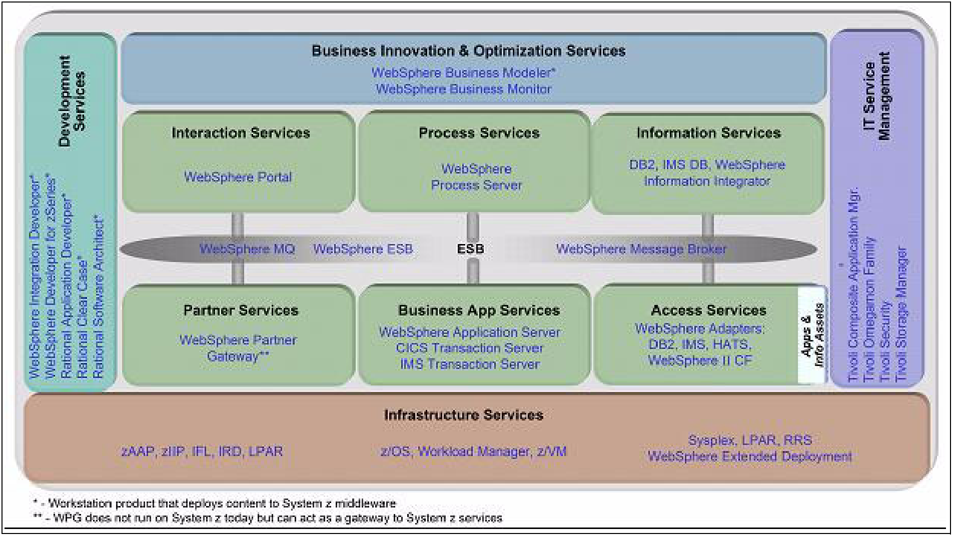

Business Innovation & Optimization Services
WebSphere Business Modeler*
WebSphere Business Monitor
Development Services
WebSphere Integration Developer*
WebSphere Developer for zSeries*
Rational Application Developer*
Rational Clear Case*
Rational Software Architect*
Interaction Services
WebSphere Portal
Process Services
WebSphere Process Server
Information Services
DB2, IMS DB, WebSphere Information Integrator
WebSphere MQ
WebSphere ESB
ESB
WebSphere Message Broker
Partner Services
WebSphere Partner Gateway**
Business App Services
WebSphere Application Server
CICS Transaction Server
IMS Transaction Server
Access Services
WebSphere Adapters: DB2, IMS, HATS, WebSphere II CF
Apps & Info Assets
IT Service Management
Tivoli Composite Application Mgr.*
Tivoli Omegamon Family
Tivoli Security
Tivoli Storage Manager
Infrastructure Services
zAAP, zIIP, IFL, IRD, LPAR
z/OS, Workload Manager, z/VM
Sysplex, LPAR, RRS
WebSphere Extended Deployment
* - Workstation product that deploys content to System z middleware
** - WPG does not run on System z today but can act as a gateway to System z services

Figure 8   SOA reference architecture with System z products

What is the next step? Some clients are already well along the SOA maturity path, while others are just getting started. Some clients have mainframe organizations that are being asked to expose existing applications as services as part of a larger IT effort. IBM provides many different service offerings to help our clients in their path towards SOA. A System z Infrastructure Architecture Workshop (zIAW) is an easy way for our clients to gain a better understanding of IBM's offerings for SOA on the mainframe, and in particular how these offerings would benefit a specific application of their choosing. Contact your local System z Software Sales Representative for further information on holding a zIAW.

Appendix: An overview of service-oriented architecture

In 2002, IBM CEO Sam Palmisano began to articulate a new vision of how business and IT could be more dynamic and responsive. This model, now referred to as “On Demand Business,” has been discussed in many contexts, including “utility computing,” “autonomic computing,” and others. The IBM On demand glossary defines “on demand business” as:

A company whose business processes—integrated end-to-end across the company and with key partners, suppliers and customers—can respond with flexibility and speed to any customer demand, market opportunity or external threat. An on demand business has four key attributes: it is responsive, variable, focused and resilient. [18]

The key words in that definition are “integrated,” “flexibility,” and “speed.” These words describe what makes service-oriented architecture (SOA) relevant to the IBM On Demand strategy. On demand is about a tighter affinity between the interests of the business and how IT supports those interests. In Figure 9 that linkage is represented as “Business and IT processes.” It is important for IT to adopt an architectural approach that facilitates that synergy between business and IT. For many enterprises, SOA is that approach.

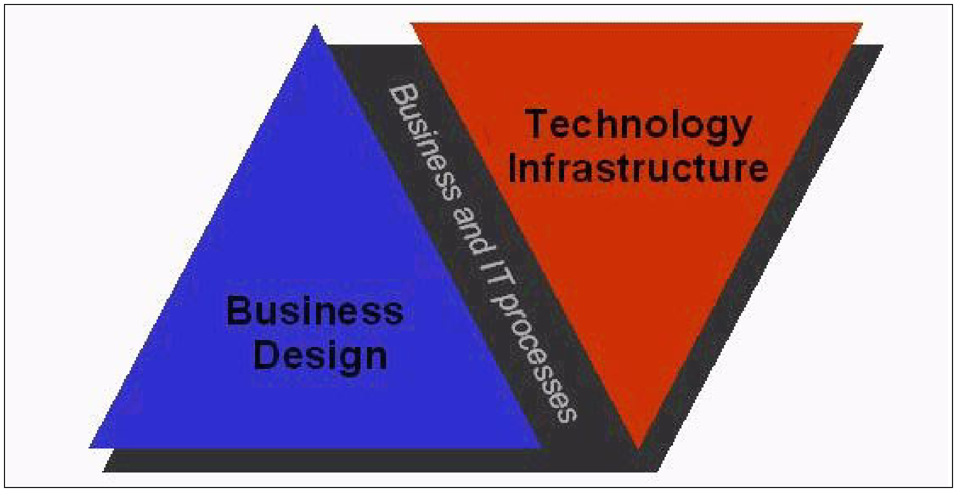


Figure 9 The on demand relationship between business and IT

Service-oriented architecture is a term that has many definitions, and fortunately, all are relatively similar. There is general acceptance in the IT industry about the

[18] Source: IBM On demand glossary
http://www-306.ibm.com/e-business/ondemand/us/toolkit/glossary_o.shtml

nature of SOA, but there is not a consensus about some of the underlying technologies within SOA.

For the purposes of this paper, SOA is defined by the IBM developerWorks Web site (`http://www-128.ibm.com/developerworks/webservices/newto/`) as follows:

Service-Oriented Architecture (SOA) is a component model that inter-relates an application's different functional units, called services, through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner that should be independent of the hardware platform, the operating system, and the programming language in which the service is implemented. This allows services, built on a variety of such systems, to interact with each other in a uniform and universal manner.

This is a good technical definition, but the same article makes a more relevant statement about SOA with respect to on demand business:

The need for loosely-coupled systems rose from the need for business applications to become more agile based upon the needs of the business to adapt to its changing environment such as changing policies, business strengths, business focus, partnerships, industry standing, and other business-related factors that influence the very nature of the business. You can refer to a business that can act flexibly in relation to its environment an on demand business, where change occurs in how things are done or work as necessary on demand.

This quote illustrates the critical nature of the relationship between SOA and on demand – SOA provides an architectural foundation for IT applications that provides the "flexibility" and "speed" that are referred to in the definition of on demand business. For businesses that are seeking to become on demand businesses, the SOA approach for application development and deployment makes sense. An application design/development model that allows architects and developers to design and build composite applications from assembled and orchestrated services makes faster and more flexible development possible. Changes to application business rules can be accelerated, and the reuse of application resources is improved dramatically.

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working with the International Technical Support Organization, Poughkeepsie Center.

Bill Seubert is a Certified zSeries Software Architect in the United States. He has over 20 years of experience in mainframe and distributed computing. He holds a Bachelor of Science degree in Computer Science from the University of Missouri, Columbia. His areas of expertise include z/OS, WebSphere integration software, and software architecture. Bill speaks frequently to IBM clients on the topics of zSeries basics, integration architecture and SOA, and enterprise modernization. He also works with IBM's Academic Initiative in building university curricula for students new to the mainframe, and he has presented on how IBM is helping revitalize the mainframe workforce. Bill is based in St. Louis, Missouri but works with clients across the Americas.

Daniel Raisch is a Senior Certified IT Architect. He has 25 years of experience in IT, mostly related to the mainframe. He holds a degree in Mathematics and Computer Science from Universidade Federal do Rio de Janeiro, Brazil. Daniel has worked extensively with customers extending core applications to new technologies and has written several redbooks. He can be reached by e-mail at raisch@br.ibm.com.

Thanks to the following people for their contributions to this project:

Lydia Parziale
International Technical Support Organization, Poughkeepsie Center

Patti Schatz, Timothy Sipples, Mike Benson, Bob Liburdi
IBM Sales and Distribution, Software Sales

Notices

This document created or updated on July 20, 2006.

Trademarks