Draft Document for Review January 29, 2007 3:05 pm

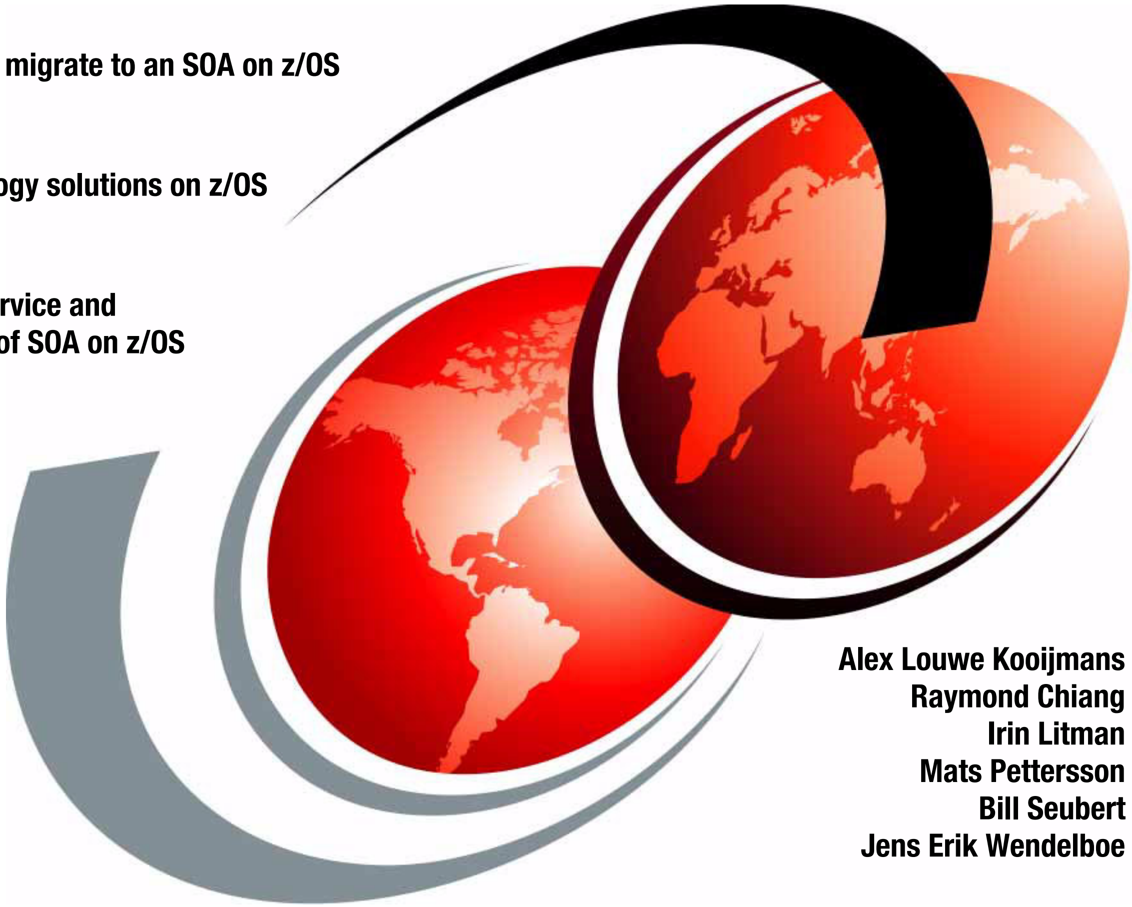
SG24-7331-00

SOA Architecture Handbook for z/OS

Scenarios to migrate to an SOA on z/OS

SOA technology solutions on z/OS

Quality of Service and governance of SOA on z/OS



Alex Louwe Kooijmans
Raymond Chiang
Irin Litman
Mats Pettersson
Bill Seubert
Jens Erik Wendelboe

Redbooks

ibm.com/redbooks

International Technical Support Organization

SOA Architcture Handbook for z/OS

December 2006

SG24-7331-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (December 2006)

This document created or updated on January 29, 2007.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

Preface

Service Oriented Architecture (SOA) is a hot topic on the agenda of many CIOs, architects and IT professionals. Most of us believe that SOA will truly make IT more flexible and that time to market of IT solutions will improve significantly.

However, in many cases the bright and shining landscape of an SOA with all its advantages is almost the opposite of today's reality in many IT environments. IT is too complex, too tightly integrated and therefore NOT flexible and the time to market of IT changes as a result of changing business requirements is not acceptable. Furthermore, technology is not in-place, and even more importantly, the organization is many times not ready for an SOA.

Even though most of us believe in the advantages of an SOA, it remains a big challenge to execute towards a full-blown implementation. In any case, when it comes to implementation of an SOA, it will be done in steps. This Redbook is about the possible approaches towards an SOA on z/OS® and the technology options available. The book will help you to define an SOA strategy on z/OS and decide on technology to be used in this journey.

The team that wrote this redbook

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

Alex Louwe Kooijmans is a project leader with the International Technical Support Organization (ITSO) in Poughkeepsie, NY, and specializes in WebSphere®, Java™ and SOA on System z™ with a focus on integration, security, high availability and application development. Previously he worked as a Client IT Architect in the Financial Services sector with IBM® in The Netherlands, advising financial services companies on IT issues such as software and hardware strategy and on demand. Alex has also worked at the Technical Marketing Competence Center for zSeries® and Linux® in Boeblingen, Germany, providing support to customers starting up with Java and WebSphere on zSeries. From 1997 to 2000, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks™ projects and delivering workshops around the world.

Raymond Chiang

Irin Litman is a mainframe connectivity/networking specialist and IBM certified. Irin is with IBM since 2003, working as mainframe connectivity specialist and as IT Architect in outsourcing projects. Previously he worked 13 years for Deutsche Bank in various roles (networking, mainframe connectivity, mainframe e-business, architecture of IT systems). His areas of interest at the moment are SOA and integration.

Mats Pettersson is a Senior IT Architect working in the Financial Services Sector in Sweden. He has 23 years of IT experience, mostly focusing on application development and systems integration. He has a broad technical experience from working on different platforms, but also from using various consulting methodologies, from business and IT alignment to implementation.

Bill Seubert is a Certified System z Software Architect in the United States. He has over 20 years experience in mainframe and distributed computing, including 17 years at IBM as a Systems Engineer, Large Systems Specialist, and an eBusiness Technical Specialist and developer. He holds a Bachelors 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 System z basics, application integration architecture and SOA, and Enterprise Modernization. He also works with IBM's Academic Initiative in building and teaching university courses for students new to the mainframe, and he has presented on how IBM is helping revitalize the mainframe workforce. Bill lives in St. Louis, Missouri but works with clients across the Americas.

Jens Erik Wendelboe is an IT Specialist in Strategy and Design and Service Delivery in Denmark. He has 32 years of experience in most parts of z/OS. His areas of expertise include WebSphere MQ and applications management. He has written extensively on application design and roles in systems programming.

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

International Technical Support Organization, Poughkeepsie Center

Jonathan Adams
Distinguished Engineer, Project and Technical leader of the IBM Patterns for e-business, IBM Software Group, IBM United Kingdom

Timothy Durniak
Development Center for Solution Integration (CSI), IBM Systems & Technology Group, Poughkeepsie

James Goethals
System z9™ Industry Infrastructure Solutions Support, IBM Systems & Technology Group, Raleigh

Mitch Green
IBM Washington Systems Center (WSC), IBM Systems & Technology Group, Gaithersburg

Thomas J Hackett
Technical sales support, Service Oriented Architecture, IBM Systems & Technology Group

Rick Robinson
Manager, EMEA Software Services Architecture Practice, Certified IT Architect, IBM United Kingdom

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction

Most of us have come across the concepts and principes of Service Oriented Architecture (SOA) by now and the benefits seem obvious, once an SOA has been fully deployed. However, the main challenge is to reach a sufficient level of SOA maturity so that the benefits really become visible. A little bit of SOA enablement here and there may introduce new technology, but not really lead to the main objectives of an SOA: a better alignment between business and IT, more flexibility of the IT environment and thus a better time to market.

SOA requires an enterprise-wide vision and strategy and a broad acceptance by the organization. Everybody in a company needs to be heading in the same direction and every single activity needs to fit in the bigger picture.

Also, an SOA is the result of a transition process. Companies have many existing applications and IT infrasructure and those will need to start playing their role in an SOA, which is not an easy task to accomplish. Getting to an SOA is like reconstructing a network of highways, without interrupting the traffic flow. A transition plan must be in place with interim steps. This situation is especially tru on the z/OS platform, as there are so many existing mission-critical applications residing on z/OS.

As you can already see, this book focuses on SOA-enablement of existing IT landscapes on z/OS, rather than on the “greenfields” approach, in which case you just follow what you learned about SOA and implement it in a new application. The real challenge is not to build a new application using SOA principles, but incorporating existing applications into an SOA. The same is true for IT infrastructure. Building a new infrastructure using SOA principles is easier than converting a fully operational one into an SOA-enabled one.

1.1 The approach in this book

In this book we focus on transition from an existing IT landscape on z/OS to an SOA-enabled one. We have found ways to “organize” this transition in such a way that the transition can be done in phases and that the right decisions can be made regarding technology. The transition approach is depicted in Figure 1-1.

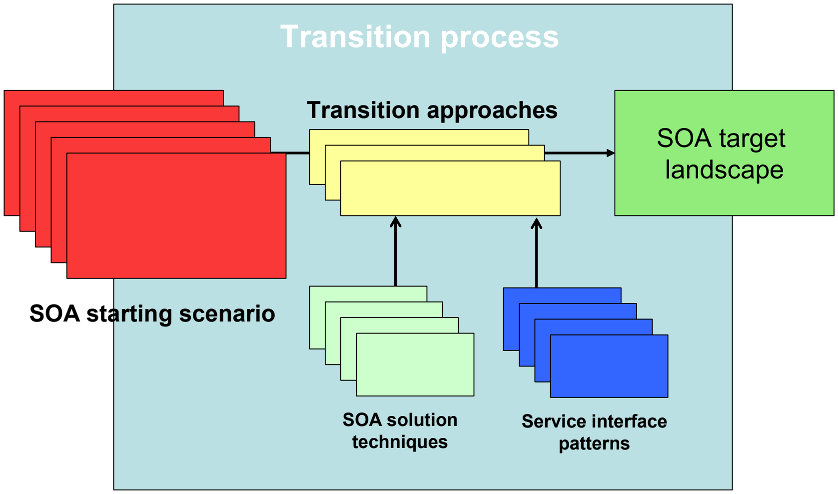


Figure 1-1 SOA transition process

We are using some terminology for the transition process:

Starting scenario Reflects the current state of technology used on z/OS. The starting scenario influences which transition approach has to be taken and which *solution techniques* are possible.

Transition approach Determines the level of SOA maturity that will be achieved.

Solution technique Technology option available to implement SOA enablement.

Service interface pattern A certain way of interfacing with the newly created services. The service interface pattern is an architectural decision, but limits the solution techniques available.

SOA Implementation scenario
The combination of a *starting scenario*, a *transition approach*, a *service interface pattern* and one or more *solution techniques* determine the *implementation scenario*.

1.1.1 Starting scenarios

The starting scenarios are discussed in Chapter 3, "Starting scenarios" on page 53. We have identified the most common scenarios on z/OS as follows:

- **3270 application**.
Applications within this scenario typically include a 3270 style user interface and all application components are run inside a transaction system such as CICS® or IMS™. This scenario is further discussed in 3.1, "Starting scenario - 3270 application" on page 55 and the applicable solution techniques are discussed in detail in 5.2, "SOA implementation scenarios for 3270 application" on page 135.
- **Multichannel**.
Applications within this scenario include a middle tier capable of serving multiple channels. The middle tier is typically implemented using J2EE™ technology and using a J2EE application server. This scenario is discussed in detail in 3.2, "Starting scenario - multichannel" on page 62 and the applicable solution techniques are discussed in detail in 5.3, "SOA implementation scenarios for multichannel" on page 162.
- **Batch**.
Application functions implemented in batch can also become part of an SOA. We discuss the batch scenario in further detail in 3.3, "Starting scenario - batch" on page 71 and the applicable solution techniques are discussed in detail in 5.4, "SOA implementation scenarios for batch" on page 183.
- **Data integration**.
In this scenario there is not really a lot of application logic, but rather tools and technology to extract, load, aggregate and integrate data from a variety of data sources. We discuss this scenario in more detail in 3.4, "Starting scenario - data access and integration" on page 78 and the applicable solution techniques are discussed in detail in 5.5, "SOA implementation scenarios - Data access and integration" on page 197.
- **Homegrown SOA**.
We consider a "homegrown SOA" as an IT environment in which some principles of SOA have been implemented already. We discuss this scenario in more detail in 3.5, "Starting scenario - Homegrown SOA" on page 87 and the applicable solution techniques are discussed in detail in 5.6, "SOA implementation scenarios for homegrown SOA" on page 213.

Of course, a combination of the above is possible.

1.1.2 Transition approaches

Depending on the level of SOA maturity that already exists and the SOA maturity to be achieved the appropriate *transition approach* can be selected. For each starting scenario (refer to the five starting scenarios mentioned in 1.1.1, "Starting scenarios" on page 3) three different transition approaches are available:

Improve Aims at *service enabling* existing assets. Service enablement means that assets get a new type of interface that opens them up for service consumers. Depending on the starting scenario and certain criteria a *solution technique* can be chosen to achieve this. The improve approach should not require any recoding or redevelopment of applications. Also, the improve scenario does not really implement an Enterprise Service Bus (ESB) yet. The SOA maturity level achieved with this approach is moderate and this approach should be seen as an interim step towards the next phase.

Adapt In this approach the focus is on *service integration*. The vehicle to achieve flexible service integration is the ESB. Implementing an ESB will have some impact, both from a technical and an organizational point of view. The SOA maturity level achieved with this approach is higher and will definitely lead to SOA benefits.

Innovate One of the biggest problems in existing assets is the level of reusability. The ideal level of granularity for a certain service may not match at all with any of the existing assets. In that case code may need to be redeveloped or "refactored" to make it better match the required service granularity. After having followed the Innovate approach, all assets should have a service interface, an ESB must be present and the services are optimzed (meaning, having the right granularity).

1.1.3 Solution techniques

For each combination of starting scenario and transition approach one or more solution techniques are available. Solution techniques include a technology solution that helps to achieve the goal of a specific phase of SOA enablement.

The solution techniques for the *Improve* transition approach focus on service enablement, the solution techniques for the *Adapt* transition approach on service integration and the solution techniques for the *Innovate* approach focus on full SOA enablement.

The solution techniques are discussed in Chapter 5, “SOA implementation scenarios” on page 129 and categorized by starting scenario / transition approach.

1.1.4 Service interface patterns

The service interface patterns are discussed in more detail in 4.3.2, “Service interface patterns” on page 119. Each transition approach can be combined with one or more service interface patterns.

1.2 The target audience for this book

This book is written from an architectural point of view and with focus on the z/OS platform. However, a lot of the information provided is useful for non-z/OS platforms as well.

If you are an architect or solution designer and you need to make decisions on SOA-enablement or transition on the z/OS platform, this book is an excellent starting point. It will provide you with patterns, transition approaches and features and functions of the technology options available.

If you are a specialist and more interested in the technical details of the solutions available, you may find some good starting points in this book. Note, however, that this book is not a “handbook” or “cookbook”.

1.3 Objectives of this book

The objective of this book is to provide information that helps to make decisions on transforming to an SOA on the z/OS platform.

- It helps you understand the traditional application landscape on z/OS and what transition approach to take towards SOA.
- It helps you to select technologies to implement SOA.
- It provides a few case studies to illustrate what is possible.

1.4 How this book is organized

The chapters in this book are orgranized as follows:

- Chapter 2, “Target SOA architecture on z/OS” on page 7 explains the SOA basics and how an ideal SOA on z/OS would look like. The chapter also positions IBM software solutions to accomodate the implementation.
- Chapter 3, “Starting scenarios” on page 53 is an overview of the application landscape as it currently exists on most z/OS systems. We have included this overview for two reasons:
  - It provides some good basic knowledge of the z/OS environment for those architects that are not that familiar with z/OS.
  - It provides a certain categorization of scenarios currently implemented and those will determine what route should be taken towards SOA enablement. For a 3270 application landscape another approach and other solution techniques may have to be used than for a batch application landscape.
- Chapter 4, “The SOA transition process” on page 95 discusses some topics improtant to know before starting any SOA transition project. It discusses methodologies, tools and patterns.
- Chapter 5, “SOA implementation scenarios” on page 129 is focusing on SOA solutions and solution techniques. For each combination of starting scenario and transition approach it will discuss available solution techniques.
- While the focus in Chapter 5, “SOA implementation scenarios” on page 129 has been on service enablement and some form of service integration, Chapter 6, “Towards service integration and process integration” on page 219 goes into more detail on the Enterprise Service Bus (ESB), service integration and process integration.
- Chapter 7, “SOA governance on z/OS” on page 257 is a chapter on an area that gets more and more attention: SOA governance. Without proper governance, an SOA may just become a one time shot and all benefits may disappear over time.
- While in the first chapters of the book the focus has been primarily on application architecture, Chapter 8, “SOA and z/OS QoS” on page 267 is focusing on the Quality of Service (QoS) aspects of the z/OS platform in an SOA. Those QoS will need to the accomodate the Non-Functional Requirements (NFRs) of the overall solution.
- Finally, Chapter 9, “SOA enablement case studies” on page 321 contains two case studies in which we apply some of the SOA theory. Both case studies are based on real implementations.

2

Target SOA architecture on z/OS

This chapter discusses the aspects of an SOA architecture on z/OS and describes how such an architecture can be realized using products from the IBM portfolio. We name it “target architecture” because we want to emphasize that this is a recommended architecture, to be reached at the end of a chain of transition stages. We describe in this chapter the end stage of the “SOA journey”.

2.1 Overview

First, in 2.2, “The context” on page 8, we establish the context in which the whole discussion and analysis takes place by describing the type of enterprise for which we plan the ideal SOA architecture.

In 2.3, “SOA basics” on page 10 we discuss basic SOA concepts and architectural principles and specify what we expect from such an architecture.

In 2.4, “IBM SOA reference architecture” on page 16 we describe the IBM SOA reference architecture and go through the conceptual building blocks of the reference architecture. After setting the stage for the SOA architecture we identify, for each SOA building block, the available options and the way these options are implemented in products on the z/OS platform.

In 2.5, “Criteria to determine whether the SOA has been implemented succesfully” on page 23 we then establish a set of criteria with which we can determine if a specific implementation option (that is, a specific combination of IBM products with their features) satisfies the requirements of the IBM SOA reference architecture.

In 2.6, “IBM options on z/OS platform for each building block of the SOA reference architecture” on page 24 we provide a summary of IBM software products available for each part of the IBM SOA reference architecture and in 2.7, “Analysis of the IBM products available for the SOA on z/OS” on page 27 we mention the most important features of those products with regards to SOA.

At that point we will be in a position to identify a set of possible implementation options for the z/OS platform (that means we have sets of products that, when put together, implement the SOA reference architecture). We examine each implementation option, describing the set of features made available by each. We differentiate between the options and explain exactly where there are overlaps and where there is missing functionality. The implementation options are discussed in 2.8, “Implementation options for the SOA architecture on z/OS” on page 41.

2.2 The context

In this section we describe the context in which the subsequent discussion will take place. We assume that we are in an enterprise with the following characteristics:

- It is a medium - to large enterprise, whose applications have evolved over years; there is a mixture of applications (and types of data processing) encompassing all scenarios mentioned in Chapter 3, “Starting scenarios” on page 53.

- The enterprise has numerous applications positioned on the z/OS platform, but also applications positioned on the distributed platforms. These applications were created at different points in time, independently of each other, and they belong administratively to diffent departments in the enterprise. Various clients access these applications, using different client technologies.
- These applications interact (inside the z/OS platform, but also between z/OS and the distributed platforms) through a variety of connectivity and middleware options; WebSphere MQ is a predominant middleware, but proprietary protocols abound, specifically in the client-server scenario.
- The applications were designed without the concept of service consumer / service provider; the interaction between the applications (and also between clients and applications) is not based on standards, but on the technologies available at the time the applications were written.
- Most applications do not have a clear separation between the presentation, business logic and data layer. There are a few exceptions, present in the multichannel scenario (J2EE enablement), where this separation was succesful.
- Real-time integration between applications is only partly done (for instance, using the WebSphere MQ protocol). A lot of integration takes place through daily or hourly FTP or batch jobs that exchange data between applications. The *E*xtract *T*ransform *L*oad (ETL) pattern is also used here.
- The applications do not appear as services for the most part; feeble attempts have been made to implement multichannel-enabled applications and “homegrown SOA” architectures.

In this context there are a lot of (maybe long-term) benefits that the organization will reap by moving to an SOA architecture and consequently enabling the applications as services.

For a detailed analysis of the benefits brought by an SOA architecture we recommend the following documents:

- Hurwitz report: Thinking from Reuse -- SOA for Renewable Business

  `ftp://ftp.software.ibm.com/software/soa/pdf/IBMThinkingfromReuse.pdf`

- CBDI Whitepaper: Business Flexibility Through SOA

  `ftp://ftp.software.ibm.com/software/soa/pdf/CBDIWhitepaperBusinessFlexibilityThroughSOA.pdf`

2.3 SOA basics

In th following sections we describe at a very high level some SOA basics (definitions, services, design points, SOA lifecycle).

2.3.1 SOA basic concepts

Service Oriented Architecture (SOA) is a business-centric IT architectural approach that supports integrating the business as linked, repeatable business tasks, or services. SOA helps users build composite applications, which are applications that draw upon functionality from multiple sources within and beyond the enterprise to support horizontal business processes.

As a gross generalization, a *service* is a repeatable task within a business process. After identifying the business processes, we identify within them the set of tasks performed within the process, then we define these tasks as services and the business process is then a *composition* of services. A number of techniques have been devised to help you identify the appropriate *granularity* and construction of services derived from the business design. The *Service Oriented Modeling and Analysis (SOMA)* technique is such an approach.

From this definition of service, *service orientation* is a way of integrating the business as a set of linked services. A *Service Oriented Architecture*, then, is an architectural style for creating an enterprise IT architecture that exploits the principles of service orientation to achieve a tighter relationship between the business and the information systems that support the business. Finally, an SOA based enterprise architecture will yield composite applications. A composite application is a set of related and integrated services that support a business process built on SOA.

2.3.2 SOA perspectives

From the perspective of the business, SOA is about modeling the business (capturing the business design), and gaining insight from that effort to refine the business design.

The Enterprise IT Architect will see SOA as being about two things, the style of architecture and the set of principles that govern it.

- SOA describes a style of enterprise architecture that structures artifacts in the information system as a set of services that can be composed to form other services.
- SOA establishes a set of principles for loose coupling, modularity, encapsulation, re-use and composability that will yield the flexibility needed to

ensure the information system is able to keep up with the rate of change demanded in the business design.

The Infrastructure IT Architect can gain value from SOA by exploiting the tools and methodologies offered by SOA for automating the business design that remains valuable to the business over time.

From the perspective of application programmers, SOA is a set of programming models and tools for building, accessing and assembling services that implement the business design together with a runtime that will execute those services efficiently.

From the perspective of the operations staff, a benefit of SOA is that it enables them to implement IT changes incrementally, replacing complex chains of machine and software dependencies with modularized services that can be substituted, tailored, modified, and deployed in a granular fashion over a virtualized infrastructure.

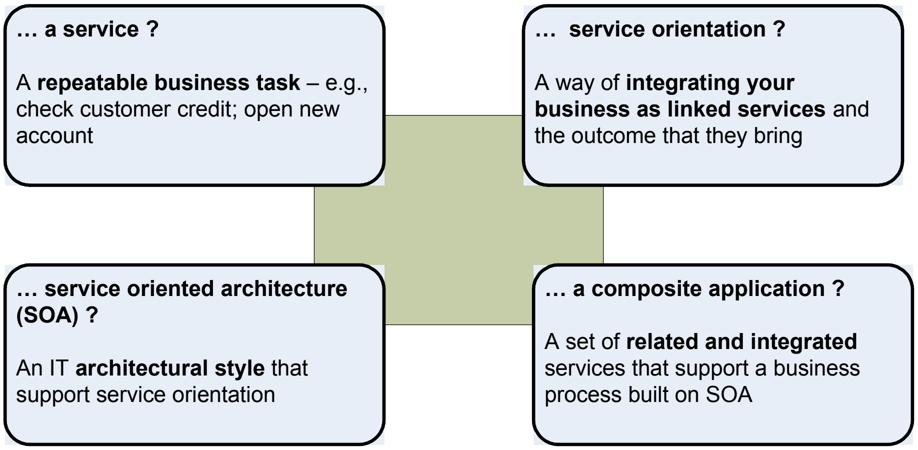


Figure 2-1 SOA definitions

2.3.3 Defining a service

An SOA is an architectural approach to defining integration architectures that are based on the concept of services. A service can be described as a function that can be offered or provided to a consumer. This function can be an atomic business function or part of a collection of business functions that are strung together to form a process.

The most commonly agreed-on aspects of a service are that:

- Services *encapsulate* a reusable business function.

- Services are defined by explicit, *implementation-independent* interfaces.
- Services are invoked through communication protocols that stress *location transparency* and *interoperability*.

Ideally, a service should be reusable and should be accessed by more than one consumer, in other words by more than one system in the architecture. It is, therefore, important to get the service description and reusability correct.

Services can be invoked independently by either external or internal service consumers to process simple functions or can be chained together to form more complex functionality or to devise new functionality quickly.

The SOA does not specify that the consumer need any specific protocol to have access to a service. A key principle in SOA is that a service is not defined by the communication protocol that it uses but instead, should be defined in a protocol-independent way that allows different protocols to be used to access the same service.

Ideally, a service should only be defined once, through a *service interface*, and should have many implementations with different access protocols. More about Service Identification in 4.1.1, “Service-Oriented Modeling and Architecture (SOMA)” on page 96.

Loose vs tight coupling

The SOA architecture and many other documents refer to the concept of *loose coupling* as opposed to *tight coupling*. But what exactly does this mean? Tight coupling means that applications are highly dependent on each other:

- Dependencies are created when two or more resources are linked together by requiring either a common platform, middleware or integration product.
- The use of inflexible message formats that cannot be changed without impacting the resources that exchange the message.
- “Hard wiring” through coding of “connectivity” information inside the applications

This “hard-wiring” of applications has its benefits:

- It is easier to perform security checking if each application knows the other.
- It is easy to fix a broken application because you always know where the partner application resides.
- Using a tightly coupled application development approach provides certain safeguards from a quality of service, security, privacy, data integrity, and complex transaction processing perspective as compared to Web services architecture.

- Tightly coupled applications know how their partner applications behave (and this implies an ability to ensure a reliable conversation between applications as well as the ability to ensure performance characteristics to a certain degree).
- Tightly coupled applications are inherently more easily managed (because both endpoints are known).

But this “hard-wiring” has disadvantages too:

- Applications need to be told by a programmer how to find each other, how to communicate with each other, and they also require long-term management / repair.
- You spend a lot of time defining the connections and relationships between cooperating applications.
- Agility is constrained as the virtualization of resources can be difficult to achieve because the resource cannot be relocated without rebuilding all of its hard-wired dependencies.
- Agility is constrained as the ability to change resources, or use an alternative provider is reduced because of the tight coupling to the current resource and/or its technology platform.

The principle of loose coupling is that dependencies between the service consumer or requestor and the service provider are minimised to enable agility.

The benefits of the loosely coupled approach are:

- Building loosely coupled applications is simpler because developers do not need to spend a lot of time defining where partner applications can be found and defining the rules that allow them to communicate.
- Maintenance of loosely coupled applications may also be easier -- for instance, should one side of a loosely coupled application break, a replacement application can be sought dynamically and run automatically.
- Loose coupling of applications provides a level of flexibility and interoperability that cannot be matched using traditional approaches.

The conclusions are:

- Tight coupling is comparatively complex and difficult, but is inherently reliable, secure, and tunable. Tight coupling constrains the ability to adapt to changing business and technology requirements.
- Loose coupling provides benefits such as dynamic lookup and heterogeneous, cross-platform interoperability, but requires an architecture and a software implementation for the security, reliability, manageability, and other mission-critical purposes.

The key benefit of loose coupling is *agility*. The service layer helps to isolate change in the providing or requesting applications. For example, the frequency of change in the business process, or the device and technology diversity used in the requesting application might be more frequent than that in the core back end systems that are used by the service provider.

In this chapter we concentrated on the building of an architecture for the loosely coupled applications, in order to reap the benefits mentioned above, and simultaneously keeping the Quality of Service benefits inherently available with a tight coupled solution. In Chapter 5, “SOA implementation scenarios” on page 129 we will see which kind of scenarios can be transformed in such a way that the resulting environment is composed of loosely coupled services.

Service granularity

Coarse grained access defines the granularity in a service or component that requires clients to make very few method invocations to get an business activity done. For instance, if a typical client needs to update a single customer record, a service based on coarse grained access would have the client update all attributes of a customer in a single invocation. Either by passing all the items as arguments to a procedure or by passing an entire customer object as the argument. As a general rule, coarse grained access is a good idea with Web services since it minimizes the number of network round trips.

The main down side to coarse grained access over *fine grained access* is that it creates rigidity in how clients work with the service. For instance, if a client was redesigned so that a single attribute of a customer is to be updated separately the other attributes,then the client might still have to update the entire customer when only one attribute has changed.

As a general rule, overly fine grained access should not be used as Web services because each method invocation requires object marshalling and demarshalling. This creates a great deal of computing and network overhead that can degrade performance far more than the efficiency gained by sending a few small pieces of data.

However, powerful counter examples of successful, reusable, fine grained services exist. More realistically, there are many useful levels of service granularity in most SOAs, as the following examples show:

- technical functions (such as logging)
- business functions (such as a service called `getAddress`)
- business transactions (such as `orderStock`)
- business processes (such as `dailyTransactionSettlement`)

We can define, for example, the coarse grained service `dailyTransactionSettlement`, which is a complex, long running service (might be a batch oriented process). This service might be composed from several fine grained services.

2.3.4 IBM SOA lifecycle

The SOA lifecycle is the framework of IBM's SOA strategy. As Figure 2-2 on page 16 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. Services may be re-used, or they may be developed new.
- **Deploy**
  Deploy the services to run-time environments, such as transaction management engines like WebSphere Application Server, CICS, IMS, and so on. 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.
- **Governance**
  Ensure that the SOA is developed and maintained in the way we originally intended it to be. A common problem in IT is that developers and project teams deviate too easily from the architecture. The quality and benefits from an SOA depends on the discipline that everybody has in following the defined architecture.

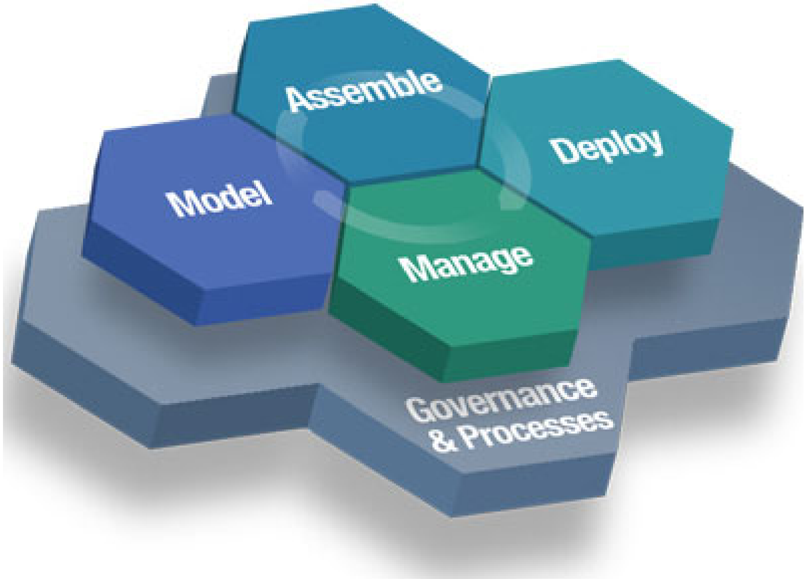


Figure 2-2 IBM SOA Lifecycle

2.4 IBM SOA reference architecture

The IBM SOA reference architecture presented in Figure 2-3 on page 17 includes building blocks that are used by the system and application architect to position the infrastructure elements and the decomposed parts of the application in a service-oriented way. In this section we describe these building blocks and their role in the architecture.

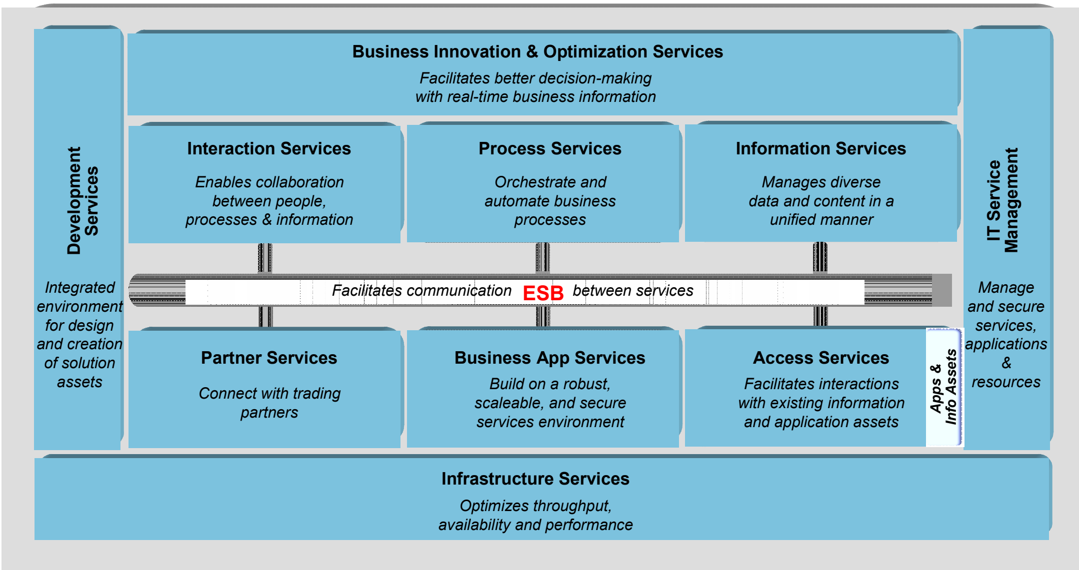


Figure 2-3 IBM SOA reference architecture

Referring to Figure 2-3, the building blocks are:

- Development Services
- Business Innovation and Optimization Services
- The Enterprise Service Bus (ESB)
- Interaction Services
- Process Services
- Information Services
- Access Services
- Partner Services
- Business Application Services
- Infrastructure Services
- IT Services Management Services

The boxes labeled *Interaction Services*, *Process Services*, *Information Services*, *Partner Services*, *Business Application Services* and *Access Services* are the parts in which we deploy application software to capture the domain logic specific to the business design. The other parts of the architecture exist to assist the rest of the SOA Lifecycle. We will use these other parts in the modeling of the business design, construction and assembly of the software, deployment of the applications, and management of the operational system and the implemented business design.

There is a reason for the way in which these building blocks were positioned in the diagram; the supporting functions (represented by Infrastructure Services, IT Services Management, Deployment Services) are positioned on the borders of the figure, surrounding with their functionality the "runtime" building blocks.

Let's go and see in detail what the IBM SOA reference architecture defines for these building blocks. Later on we will fill the blocks with the available IBM implementation options.

2.4.1 Development Services

Development Services is an essential component of any comprehensive integration architecture. These services enable different people, involved in the development of a custom application, to develop the artifacts based on their skills, their expertise, and their role within the enterprise.

- Business analysts analyze business process requirements using modeling tools that allow processes to be charted and simulated.
- Software architects model the system structure and behavior.
- Integration specialists design specific inter-connections in the integration solution.
- Programmers develop new business logic with little concern for the underlying platform.

Development Services enable joint development, asset management and collaboration among all these people. A common repository and functions common across all the developer perspectives (e.g. version control functions, project management functions, etc.) are provided through a unified development platform

2.4.2 Business Innovation and Optimization Services

Business Innovation and Optimization Services provide monitoring capabilities that aggregate the operational and process matrix in order to efficiently manage systems and processes.

These capabilities are delivered through a set of comprehensive services that collect and present both IT and process-level data. This allows business dashboards, administrative dashboards, and other IT level displays to be used to manage system resources and business processes.

The important message here is that these services get insight not only into IT processes, but also into the business processes, and so they can deliver not only the traditional IT information, but also information about the business aspects of the application.

Runtime data and statistics can be delivered, from Business Innovation and Optimization services to Development Services. This allows analysis and an iterative process of re-engineering.

2.4.3 The Enterprise Service Bus (ESB)

At the core of the SOA reference architecture is the *Enterprise Service Bus (ESB)*. This architectural construct delivers all the inter-connectivity capabilities required to use (and re-use) services implemented across the entire architecture. The ESB provides the following fundamental services:

- *Transport Services* providing the fundamental connection layer
- *Event Services* that allow the system to respond to specific events that are part of a business process
- *Mediation Services* like transformation and validation services that allow loose coupling between interacting services in the system

The Enterprise Service Bus (ESB) is a “silent partner” in the SOA logical architecture. Its presence in the architecture is transparent to the services of your SOA application. However, the presence of an ESB is fundamental to simplifying the task of invoking services – making the use of services wherever they are needed, independent of the details of locating those services and transporting service requests across the network to invoke those services wherever they reside within your enterprise.

The ESB enables the substitution of one service implementation by another with no effect to the clients of that service (this is one of the most important point in SOA design). This requires both the service interfaces that are specified by SOA and that the ESB allows client code to invoke services in a manner that is independent of the service location (location transparency), communication protocol (transport neutrality) and interaction protocol.

The fact that the ESB is interposed between participants provide the opportunity to modulate their interaction through a logical construct called a *mediation*. ESB implementations offer basic mediation constructs like:

- **Protocol switch**
  Its role is to transcode requests from one interaction protocol or API to another.

- **Transform**
  Its role is to translate the message payload fom one schema to another; this may include encryption.
- **Enrich**
  Its role is to augment the message payload with external information.
- **Route**
  Its role is to change the route of a message , selecting among service providers based on selection criteria.
- **Distribute**
  Its role is to distribute the message to a set of interested parties.
- **Monitor**
  Its role is to oberve messages as they pass throgh mediation unchanged. Can be used to monitor service levels, to support billing, and for auditing purposes.
- **Correlate**
  Its role is to derive complex events from message and event streams.

These basic mediation constructs can be combined to realize more complex patterns, like the *canonical adapter* (protocol switch followed by a transformation) or the “transform-log-route” mediation.

The ESB is not the only infrastructure component in an SOA. Although individual scenarios vary, there are other commonly occurring components whose role we should position relative to the ESB:

- The *Business Service Directory*, which provides the details of available services to systems that participate in an SOA. In order to perform routing of service interactions, the ESB obviously requires at least basic routing information, which might be provided by an ESB Namespace Directory, or by more simple means such as a routing table.

  However, this routing information is not necessarily the same as the business service directory SOA component; the role of the Business Service Directory is to provide details of services that are available to perform business functions. The Business Service Directory might be an open-standard UDDI directory, or some other registry and repository construct.

  Such implementations can achieve one of the primary goals of a Business Service Directory: to publish the availability of services and encourage their reuse across the development activity of an enterprise.

- *Business Service Choreography*, which is used to orchestrate sequences of service interactions into short or long-lived business processes.

- The *ESB Gateway*, which is used to provide a controlled point of external access to services where the ESB does not provide this natively. Larger organizations are likely to keep the ESB Gateway as a separate component.

  An ESB Gateway can also be used to federate ESBs within an enterprise. For more information on the ESB, see Keen et al. *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346.

2.4.4 Interaction Services

Interaction Services are supporting the presentation logic of the business design components that enable the interaction between applications and end users. Interactions may be tailored to role-sensitive contexts, adjusting what is seen and the behavior presented to the external world based on who the user is, what role they are performing, and where they are in the world.

2.4.5 Process Services

Process Services provide the control and management services that allow integration and automation of business processes that span people, workflows, applications, different systems and platforms and meld them into single, orchestrated application. Process services include various forms of compositional logic, the most notable of which are *business process flows* and *business state machines* (finite-state machines for business composition).

2.4.6 Information Services

Information Services provide the capabilities required to federate, replicate, integrate, analyze and transform data sources. These services provide access to the data repositories through various 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.

2.4.7 Access Services

Access Services provide bridging capabilities between legacy applications, pre-packaged applications and enterprise data stores. These access services make available the functions and data of the *existing* enterprise applications as services, thereby allowing them to be reused and incorporated (like other service components) into functional flows that represent business processes.

This includes simple wrapping of those functions and rendering them as services (in the case where the existing function is a good match with the semantic requirements of the business model in which it will be used), or in more complex cases augmenting the logic of the existing function to better meet the needs of the business design.

In the latter case, the access service may in fact invoke multiple legacy functions to achieve the semantic requirements of the service.

2.4.8 Partner Services

In many enterprises business processes involve inter-actions with outside partners and suppliers. *Partner Services* provide the document, protocol, and partner management services required for business-to-business processes and interactions.

In some ways partner services look like interaction services – projecting a view of the business to the partners, and controlling the interaction with them as an external entity. In other respects, partner services look like access services – rendering the capabilities of that partner as a service so that those functions can be composed into your business processes like any other service.

2.4.9 Business Application Services

Business Application Services implement the core business logic. These are service components created specifically as services within a business model and that represent the basic building blocks of your business design – services that are not decomposable within the business model, but that can be composed to form higher level services.

2.4.10 Infrastructure Services

Infrastructure Services provide functions for scalability and performance, security and resource virtualization capabilities. These include the hardware and software for deployment of the actual business services and service infrastructure. The Infrastrucure Services component provides the required degree of Quality of Service.

Many of the Infrastructure and IT Service Management services perform functions tied directly to hardware or system implementations, others provide functions that interact directly with integration services provided in other elements of the architecture through the ESB. These interactions typically involve services related to security, directory, and IT operational systems management.

2.4.11 IT Services Management Services

IT Services Management Services provide security, directory, IT system management, service level automation and orchestration, and virtualization functions. The infrastructure required to support an SOA and the applications cooperating in an SOA are typically dispersed over several platforms and technologies, and is therefore significantly more complex than traditional infrastructure necessary for 'self-contained' applications.

SOA Management can be thought of in three layers:

- *Business Service Management* provides for service level planning, business impact monitoring, and prioritization of event management
- *Composite Application Management* provides support for securing the SOA environment, flow content analysis, end-user response time monitoring for service requests, service problem diagnosis, and application trace information that you can then pass back to your development environment
- *Resource Management* enables orchestration, provisioning, infrastructure health monitoring, and event automation.

2.5 Criteria to determine whether the SOA has been implemented succesfully

The criteria used in determining whether the implemented architecture in an enterprise is an SOA, are derived from the SOA benefits. There are different degrees of SOA compliance, but since we are attempting to describe 'ideal" implementations we will look at the criteria for a "state of the art" SOA implementation. Also refer to Chapter 4, "The SOA transition process" on page 95 and especially 4.1, "Methodologies for analyzing the business and application environment" on page 96.

Each complete SOA implementation option should allow:

- Modeling, construction, deployment (in the runtime instances) and management of service components . This means having an integrated set of tools and runtimes that allow the consequent implementation of the SOA lifecycle.
- Positioning the services as loosely coupled, reusable and composeable elements. This means using connection and calling mechanisms between services, with clearly described interfaces (that use non-proprietary, recognized standards). This means we can replace components at will, provided they respect the defined interface. This also means having available tools for composing applications by putting together services and describing

the business flow using these services; for running the “composed” application (processes) we need a runtime able to reach the service providers and run the business logic.

- *Location transparency.* This means we should have in our SOA implementation the service directory component that allows the discovery of the services, their calling methods and quality of services.
- *Transport neutrality*. This means that the implementation should allow the services to be “called” in standardized ways, without associating them (or restricting them) to any specific transport protocol. This also means we need mediations and transformation services that will allow service consumers to call service providers, even if they implement different protocols.
- The possibility to attach, find and use internal and external legacy and packaged applications as coarse grained services, through the use of adapters, gateways and other artifacts.
- The possibility to attach, find and use “information as service” components, as well as other future components (through easy implemented extensions).
- The possibility to manage the deployed services, both from a traditional IT view but also from a business oriented view, allowing us an insight into the composite application.

Now that we defined the criteria for SOA, we move on a “SOA Journey” to achieve them. This journey will (in almost all cases) take place in steps, moving from different stages of *service enablement*, to several stages of *service integration* and lastly into several stages of *process integration*. Therefore the path to the “ideal” SOA architecture will be a succession of steps, a combination of positioning SOA infrastructure products and SOA-enablement of applications. The speed and the depth of the implementations are, as always, based on the exact enterprise landscape and, most important, of the requirements of the enterprise.

In the next section we show what the actual IBM products are for each building block of the SOA reference architecture.

2.6 IBM options on z/OS platform for each building block of the SOA reference architecture

In the following section we show the IBM implementation options (set of products) that are available on the z/OS platform. We position the products over the building blocks of the SOA reference architecture, and present an overview of the delivered functionality.

Figure 2-4 shows the available IBM products on the z/OS platform, and their positioning in the IBM SOA reference architecture.

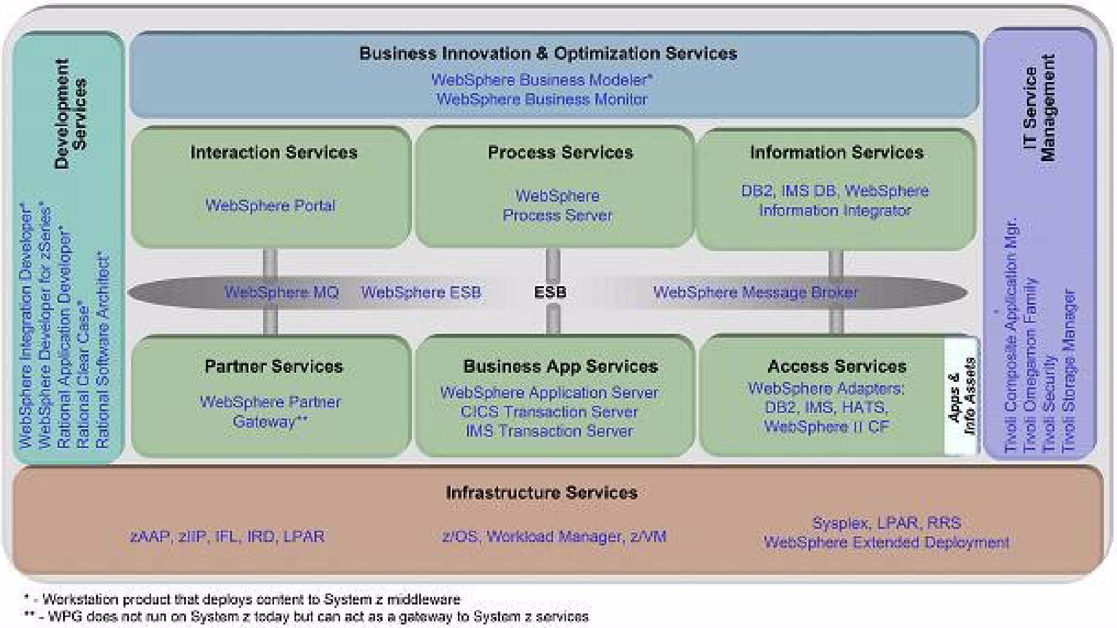


Figure 2-4 IBM SOA reference architecture and the IBM products on z/OS

2.6.1 Infrastructure Services

Here we see all IBM hardware and software features (either implemented as hardware features of the System z platform, or in the z/OS operating system, or in the Websphere products on z/OS). These are the hardware and O/S features that allow the implementation of the desired QoS levels for availability (Parallel Sysplex®), performance and optimization (Workload manager, specific usage of the processors, such as zAAP, zIIP and IFL, WebSphere XD), security and transactional capabilities (RRS).

2.6.2 Development Services

In this area we position among others the *WebSphere Business Modeler*, the *Rational® development tools* and the *WebSphere Integration Developer* products. These are the tools that allow the developer to construct the service artifacts, to define their interfaces, integrate the services in process flows, and deploy them in the runtimes.

2.6.3 IT Services Management Services

In this area we position products like *IBM Tivoli Composite Application Monitor*, the *Tivoli® Omegamon* family and *Tivoli security products,* such as *Tivoli Access Manager (TAM)*. These are supporting products that are neccessary in order to get insight into the different SOA layers (operational systems, service components, services, business processes).

2.6.4 Business Innovation and Optimization Services

In this area we see products such as *Websphere Business Modeler* and *Websphere Business Monitor*. These products are necessary when we have already created services, deployed them, and now wish to construct and deploy composite applications and process flows.

2.6.5 Interaction Services

In this area we position the *WebSphere Portal* and the *Workplace™ WebSphere Everyplace®* products. These products can be used when the enterprise comes to the point of integrating the user interfaces into a unified presentation, and allowing the user interfaces flexible access to information and services.

2.6.6 Process Services

In this area we see the *WebSphere Process Server (WPS)* product. This product will be used when the enterprise disposes of a sufficient number of services and start combining them into process flows, positioning the business rules in these flows and therefore outside the business logic component.

2.6.7 Information Services

In this area we see the IBM products that allow access to data, like *DB2®, IMS DB*, as well as products that provide the “information as a service” functionality like *Websphere Information Integrator*.

2.6.8 Partner Services

In this area we position the *WebSphere Partner Gateway* product. This product will be used at the point where the enterprise wants to access service components located at the partners, or would like to make available its services to the partners. Usually this happens after the infrastructure for the Enterprise Service Bus has been positioned.

2.6.9 Business Application Services

In this area we see products that run the services that implement business logic. These products include *Websphere Application Server (WAS)*, *CICS Transaction Server* and *IMS Transaction Manager*.

2.6.10 Access Services

Here we see the IBM products that allow access to legacy applications, like *Websphere Adapters, Host Aaccess Transformation Services (HATS)* and *Websphere IICF*. These products can be implemented when the enterprise decides to service-enable legacy applications. Later on, through the use of an ESB, the service-enabled legacy applications can be reused by new composite applications.

2.6.11 Enterprise Service Bus (ESB)

In this area we position *WebSphere ESB (WESB)* and *Websphere Message Broker (WMB)* products. The ESB is a central component in any SOA implementation, and the positioning might be necessary as soon as the enterprise has a need for “reusing” services (and in the process having the need for mediations, transformation, routing, protocol independence, location transparency, etc).

2.7 Analysis of the IBM products available for the SOA on z/OS

Now that we know what IBM products are available for each building block, it is the right time to determine the features implemented by them. But we are not going to present a “data sheet” for all the products involved; instead, we will show how specific features are there for a purpose, namely to implement SOA architecture requirements.

We will concentrate only on the features that are related to SOA, that means all features that were built into the products with the purpose of enhancing the QoS of the architecture or enable the SOA features.

IBM is following an obvious strategy in regard to the z/OS products for the SOA reference architecture. One of the activities is to go over all the building blocks, identify the products that play a role in that area, and SOA enable them. This means:

- If the product is a development tool, it is enhanced with the ability of producing SOA components to run on z/OS runtimes and the ability of

SOA-enabling existing components. For example, *WebSphere Developer for z (WDz)* is producing fully SOA-enabled business components to run in WebSphere Application Server, CICS TS, IMS TM and DB2. It produces all the necessary artifacts needed (either for the provider or for the consumer). WebSphere Developer for z is also able to enable existing components like Java Beans, CICS and IMS transactions, DB2 stored procedures. These artifacts might be WS-Binding files, SOAP proxies, presentation components, portlets, etc. The development tools provide all these features in an integrated way, ecompassing the whole cycle of application development.
Another example of a development tool is *WebSphere Integration Developer (WID)*, whcih can be used to produce process flows that in turn are able to call new or existing SOA components.
The development tools are also enabled to deploy the components on the runtimes.

- If the product is a runtime for business logic, then IBM makes sure it is sufficiently enhanced (or provides a new product) so that it is able to run SOA components. For example, WebSphere Application Server is able to run SOA-enabled components (Web services). This is also valid for CICS Transaction Server. When the products are not directly supporting Web services, IBM delivers additional tools, wrappers, adapters or gateways that allow existing components to be used as SOA services.
WebSphere Process Server is an example of a new runtime which was provided specifically to run processes.
- The development and runtime products are constantly enhanced by implementing the standards as they stabilise (for example the set of WS-* standards in the area of Web services). IBM ensures consistency in the support of the Web Services standards across the products, and therefore their interoperability, and interoperability with other software implementations.
- The runtimes are constanly enhanced to support a variation of transports, interaction patterns, languages used for development, etc. This is done to increase flexibility and allow the developers freedom of decision about many aspects of the implementation, but still ensure the services fit in the SOA architecture.
- For some of the SOA building blocks (like the ESB) IBM delivers several options in order to better fullfil the market requirements. The available options are described in Chapter 6, “Towards service integration and process integration” on page 219. Depending on the enterprise requirements, IBM recommends either the new WebSphere ESB product or the enhanced WebSphere Message Broker product.
- If the product is for information services, then IBM enhances it to allow the conept of “information as service”. More information about this subject is available in 5.5, “SOA implementation scenarios - Data access and integration” on page 197.

- If the product belongs to interactive access (like WebSphere Portal), IBM enables it to be able to participate in the SOA architecture. This means adding features that allow portlets to consume SOA components.
- IBM enhances existing management products and delivers new ones in order to be able to manage the whole lifecycle of the SOA components. Each runtime is impoved in such a way that the Tivoli management tools are able to see “inside” and manage the SOA service component

Other IBM activities make the platform z/OS more attractive for the enterprises. The enhancements go into putting more functionality and standards into the products running on z/OS, improving the QoS of the platform, and reducing the Total Cost of Ownership. The TCO reduction is implemented by means of, for example, the usage of zAAP processors, performance improvements in the runtimes, virtualisation and many others.

We will not discuss here the development tools (which are presented in 4.2, “Tools to assist in the SOA transformation process” on page 108), but concentrate mostly on the runtimes.

2.7.1 WebSphere Application Server for z/OS

This section discusses the most important features of WebSphere Application Server for z/OS Version 6.1, that directly benefit the SOA.

Core features that increase the reusability and service integration

- WebSphere Application Server for z/OS, V6.1 supports *Session Initiation Protocol (SIP)* servlets - voice clients added as SOA participants.
- Web Services Gateway enables Web services invocation by users from inside and outside the firewall with the benefit of robust security protection and a centralized point of control.
- New Web services standards including WS-Business Activity, WS-Notification and WS-I Basic Security Profile are supported.
- A powerful built-in JMS engine helps extend the reach of new and existing applications. A JMS service consumer can participate directly in the SOA.
- Extensive Web services support and close proximity to core mainframe assets allow easy reuse and development of composite applications.

Note: Note that WebSphere Application Server for z/OS provides much more functionaltiy than the features mentioned above. We only concentrate on Web services related features in this section.

Features that implement QoS

- As a fully participating member of a Parallel Sysplex, WebSphere Application Server for z/OS delivers features such as a *High Availability Manager* and backup cluster support.
- The unified clustering framework has been enhanced to provide Web services and Session Initiation Protocol (SIP) servlet clustering capabilities.
- Out-of-the-box security configurations and user registry, compliance with government standards, and stringent Web services security.
- High throughput and scalability with JDK™ 5 enhancements and improved cache off-loading.
- Leverages local connections to DB2 Universal Database™ for z/OS which eliminate unnecessary path length and significantly improve performance.
- Unique architecture that derives significant value from integration with z/OS WLM, which enables a flexible deployment environment.

> **Note:** We just mentioned some highlights related to the QoS supported in WAS for z/OS. There is much more to say about QoS in WAS for z/OS, but would require a book by itself. Refer to other Redbook and Redpaper publications in the area of WebSphere Application Server for z/OS for more information on QoS, such as security, availability and integration.

2.7.2 WebSphere Portal

This section discusses the most important features of WebSphere Portal Enable for z/OS, Version 6.0, that directly benefit the SOA.

> **Important:** At the time of writing, WebSphere Portal Enable for z/OS, Version 6.0 has been announced. More information, including the announcement letter can be found at url:
>
> `http://www.ibm.com/software/genservers/portalzos/index.html?S_TACT=103BGW01&S_CMP=campaig`

Core features that increase the reusability and service integration

- IBM WebSphere Portal delivers a complete set of capabilities that enable the assembly and orchestration of presentation components.
- WebSphere Portal supports JSR 168, a standard Application Programming Interface (API) for creating portlets as the integration component between applications and portals on a J2EE platform.

- WebSphere Portal supports the *Web Services for Remote Portlets (WSRP)* standard, for dynamically integrating business applications.
- WebSphere Portal also fully leverages IBM WebSphere Host Access Transformation Services (HATS) to quickly and easily extend legacy applications into a portal via reusable portlets.
- Orchestration supported through Human Task List (BPEL) and cooperative portlets.

> **Note:** WebSphere Portal Enable for z/OS, Version 6.0 provides much more functionality than the highlights mentioned above. A Redbook is planned to be published in 2007 with more detailed information WebSphere Portal for z/OS.

2.7.3 WebSphere Process Server

This section discusses the most important features of WebSphere Process Server for z/OS Version 6.0, that directly benefit the SOA.

Core features that increase the reusability and service integration

- Support for Web services - SOAP/HTTP, SOAP/JMS, WSDL 1.1, WS-* Standards including WS-Security and WS-Atomic Transactions.
- Support for a variety of messaging protocols including JMS 1.1, WebSphere MQ, TCP/IP, SSL, HTTP(S), and multicast for optimum flexibility and improved asset reuse (more ways to interact between service consumers and service providers).
- Standards-based connectivity to integrate applications and services.
- Easy interoperability with WebSphere Family - WebSphere Application Server, WebSphere MQ, WebSphere Message Broker.
- Messaging services for clients running in non-Java applications in C/C++ and Microsoft® .NET environments. The Web services Client is a JAX-RPC-like Web services client for C++ that enables users to connect to Web services hosted on WebSphere from a C++ environment.
- Java 2 Platform Enterprise Edition (J2EE) Connector Architecture (JCA) Version 1.0 and Version 1.5 resource adapters to access back-end systems.
- Support for the entire suite of IBM WebSphere Business Integration Adapters Support for Web services.
- Support for JMS through integrated WebSphere messaging resources (with full connectivity to existing IBM WebSphere MQ technology-based networks).

- Support for Web services (based on Java Specification Request [JSR] 109 and Java application programming interface (API) for XML (JAX)-RPC technology).
- Support for calling Enterprise JavaBeans™ (EJB™) session beans.
- Support for exposing and calling IBM CICS or IBM IMS programs as enterprise services.

Features that implement the "loosely coupled" concept

- Built on *WebSphere Enterprise Service Bus (WESB)* for service-oriented integration (therefore inherits the already available service integration).
- WebSphere Process Server handles the integration logic, transformations, routing, rules.
- Dynamic business processes are supported.
- Visually description of processes that span people, systems, applications, tasks, rules, and the interactions among them
- Supports long- and short-running business processes.
- Integrates fault handling for easy, in-flow exception handling.
- WebSphere Process Server contains a business-rule component that provides support for rule sets ("if-then" rules) and decision tables. Business rules are categorized into rule groups which hide implementation details from the consumer and which are accessed just like any other component.

Features that implement QoS

- Support for transactions involving multiple resource managers using the two-phase commit process. This capability is available for both short-running processes (single transaction) and long-running processes (multiple transactions).
- Multiple steps in a business process can be configured into one transaction by modifying transaction boundaries in *WebSphere Integration Developer (WID)*, then deployed into Websphere Process server.
- Provides transaction rollback-like functionality for loosely coupled business processes that cannot be undone automatically by the application server.

2.7.4 Websphere ESB

This section discusses the most important features of WebSphere Enterprise Service Bus for z/OS Version 6.0, that directly benefit the SOA.

Core features that increase the reusability and service integration

- Support for a variety of messaging protocols including JMS 1.1, WebSphere MQ, TCP/IP, SSL, HTTP(S), and multicast for optimum flexibility and improved asset reuse (more service consumer and service provider can participate).
- Utilize a broad range of interaction models (request/reply, point-to-point, publish/subscribe and multicast).
- Leverage advanced Web services support to incorporate SOAP/HTTP, SOAP/JMS, WSDL 1.1, Web Services Gateway. WebSphere ESB supports WS-* Standards including WS-Security, WS-Atomic Transactions and includes a UDDI 3.0 Registry that can be used to publish and manage service end point metadata.
- Message Service Client for C/C++ extends the JMS model for messaging to non-Java applications.
- Message Service Client for .NET enables legacy or .NET applications to participate in JMS-based information flows (additional service consumers).
- Web Services Client is a JAX-RPC-like Web services client for C++ to enables users to connect to Web services hosted on WebSphere from within a C++ environment.
- J2EE client support from WebSphere Application Server, including Web services Client, EJB Client, and JMS Client.

Features that implement the “loosely coupled” concept

- Websphere ESB will handle the integration logic, transformations, routing and rules.
- Customized routing - Transport/protocol specific routing and content based routing.
- Protocol transformation between a variety of protocols: HTTP, IIOP, JMS.
- Format transformation between standards: XML, SOAP, JMS messages, and many more when used with adapters.
- Supplied mediation function for database interaction.
- Support for message logging to database and message augmentation by database lookup.
- Administrator support for reconfiguring service interactions.
- Avoid system downtime by adding or replacing integration logic dynamically.

Features that implement QoS

- Inherits from WebSphere runtime the scalability, clustering, and fail-over capabilities.
- Utilizes common WebSphere Administrative Console to enable system management across WebSphere Application Server, WebSphere ESB, and WebSphere Process Server.
- Support for global transactions. A global transaction is required when mediating and routing messages must be coordinated into a single transaction, or when several mediation handlers in a mediation handler list must be coordinated into a single transaction. Setting the global transaction property ensures transactional integrity between a mediation that accesses the resources owned by other resource managers, and the messaging engine.
- Addresses end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability (SCA security and message level security on top of the security features delivered by WebSphere Application Server, Java, and the z/OS platform).
- Integrates tightly with IBM Tivoli security, directory, and systems management offerings (Tivoli access manager, Tivoli directory, Tivoli Composite Application Manager for SOA).

2.7.5 Websphere Message Broker

This section discusses the most important features of WebSphere Message Broker for z/OS Version 6, that directly benefit the SOA.

Core features that increase the reusability and service integration

- Numerous connectivity options allow practically each service to attach to the Message Broker.
- Integrated WebSphere MQ transports - for Enterprise, Mobile, Real-Time, Multicast and Telemetry end points - extend the scope of client types.
- Provide native JMS interoperability, acting as a bridge between any combination of different JMS providers.

Note: WebSphere Message Broker for z/OS Version 6 offers a full range of ESB functions. It is beyond the scope of this book to discuss them all in detail. Refer to 6.3, "Stage two - "Service Integration"" on page 223 for more information on the ESB and the role of WebSphere Message Broker for z/OS. Also refer to *Implementing an Advanced ESB using WMB V6 and WESB V6 on z/OS*, SG24-7335.

Features that implement the "loosely coupled" concept

- WebSphere Message Broker handles the integration logic, transformations, routing and rules.
- Distribute any type of information across and between multiple diverse systems and applications.
- Reduce the number of point-to-point interconnections and simplify application programming by removing integration logic from the applications.
- Validate and transform messages in-flight between any combination of different message formats, including Web Services, other XML and non-XML formats.
- Route messages based on (evaluated) business rules to match information content and business processes.
- Dynamically reconfigure information distribution patterns without reprogramming end-point applications.
- Mediates (provides routing, transformation and logging) between Web Service requesters and providers.
- Mediates between Web Services and other integration models as both a service requester and a service provider.

Features that implement QoS

- Option to use Java for processing and transformation allows for off-loading of this function onto zAAP processors for System z implementation.
- WebSphere Message Broker supports both WebSphere MQ queue sharing and queue clustering. WebSphere MQ queue sharing is a unique concept for high availability only available on z/OS.

2.7.6 WebSphere Service Registry and Repository

This section discusses the most important features of WebSphere Service Registry and Repository (WSRR), that directly benefit the SOA. Note that, at the time of writing, WSRR is not available on z/OS.

Attention: At the time of writing WSRR was not available yet for the z/OS platform. The announcement, however, can be found:

`http://www.ibm.com/fcgi-bin/common/ssi/ssialias?infotype=an&subtype=ca&appname=Demonstration&htmlfid=897/ENUS206-315`

Core features that improve the reusability and service integration

- Enables the publication, discovery, subscription, and governance of services. The registry component records the definition of the services (what they offer) and where they are located. In addition, the repository covers details such as how the services are used, by whom, and why.
- Enables dynamically finding the service based on the description of the intefaces and QoS requirements

2.7.7 WebSphere Host Access Transformation Services

This section discusses the most important features of WebSphere Host Access Transformation Services (HATS), that directly benefit the SOA.

Features that enable existing applications as services / creation of new services

- Provides programmed access to 3270 host transactions through standard Web services interfaces.
- Combines data selected from multiple host sources with Java technology-based applications to create new WebSphere applications. You can encapsulate transactions with host systems into reusable business objects, such as Web services, Java beans or Enterprise JavaBeans (EJB).

Features that increase the reusability and service integration

- HATS can run directly in the WebSphere Portal environment and take advantage of integration with other portlets in the portal, so the portlets can become service consumers of 3270 SOA-enabled applications.

Features that implement QoS

- Leverages the security-rich features provided by IBM WebSphere Application Server and IBM WebSphere Portal. Secure Sockets Layer (SSL) and Secure HTTP (HTTPS) provide robust security between the host application, the mid-tier server and the end user.

2.7.8 CICS Transaction Server

This section discusses the most important features of CICS Transaction Server Version 3.1, that directly benefit the SOA.

Core features that improve the reusability and service integration

- CICS is capable of being a Web Service provider and consumer through a variety of native (CICS Web services support) and adapter (CICS Transaction Gateway) technologies. This dual provider/consumer role means that CICS is now a full participant in the B2B and B2C world of e-portals and e-marketplaces.
- CICS supports Web services sent over the HTTP and WebSphere MQ transports for flexible deployment options dependant on the requirement of applications.
- CICS supports SOAP-enablement features like Link3270 bridge, JCA, Service Flow Modeler (SFM[1]) and the Service Flow Runtime (SFR). The Service Flow Modeler can be used to model the flow between CICS services, create interactions with CICS applications and expose the flow as a Web Service. The Service Flow Runtime contains adapters and other supporting code that allow the flow to run. SFM and SFR are presented in “Variation 2: CICS Service Flow Feature (CICS Transaction Server V3.1)” on page 139.
- CICS TS V3.1 supports new technical constructs such as *pipelines* and *handlers*, which enable the processing of the headers the SOAP messages and therefore allow the implementation of the WS-* specifications.
- CICS TS V1.3 supports the following specifications: SOAP 1.1 and 1.2, WS-I Basic Profile 1.1, WS-Security (SOAP message security), WS-Coordination, WS-AtomicTransaction.
- CICS TS V3.1 supports new technical constructs such as *channels* and *containers*, that on one hand improve features for the business logic (enhanced CICS data exchange), on the other hand form the infrastructure basis for the SOAP/Web services artifacts.
- CICS TS V3.1 supports transactions written in Java, enabling more flexibility for the implementation of the service providers.
- Web services tooling is available for creating a Web service out of an existing program (bottom-up) or creating a program based on available WSDL (top-down).

[1] The Service Flow Modeler (SFM) is part of the WebSphere Developer for z product.

2.7.9 IMS Transaction Manager

This section discusses the most important features of IMS Transaction Manager , that directly benefit the SOA.

> **Important:** Most of the features mentioned below will become available in IMS V10. The announcement letter can be found at:
>
> `http://www.ibm.com/common/ssi/fcgi-bin/ssialias?subtype=ca&infotype=an&appname=iSource&supplier=897&letternum=ENUS206-238`
>
> The IMS SOAP Gateway and MFS Web support are available in IMS V9 too.

Core features that improve the reusability and service integration

- IMS is capable of being a Web service provider through *IMS SOAP Gateway* and other access components.
- *Integrated Connect XML Adapter support for COBOL* enables reuse of IMS applications as Web services.
- *IMS calllout* enables IMS applications as clients to interoperate with business logic outside of the IMS environment (e.g. Web service or J2EE application).
- *IMS SOA Composite Business application support* enables integration of IMS transactions in SOA-based composite business applications.
- *MFS Web support* enables access to existing IMS MFS based transactions from WebSphere Application Server. Refer to “Variation 3: IMS using MFS Web Support” on page 141 for more details.
- *IMS Connector for Java PL/I application support* alows IMS transactions to be enabled as Web services without IMS application changes.

2.7.10 DB2

This section discusses the most important features of DB2 for z/OS, that directly benefit the SOA.

Core features that improve the reusability and service integration

- DB2 can act as service provider through the *Web services Object Runtime Framework (WORF)*, which runs in WebSphere Application Server.

- DB2 can act as service consumer, using DB2 *User Defined Functions (UDFs)*. Optionally, the DB2 XML extender can be used to parse the results from an incoming SOAP message.
- DB2 supports native XML and XQuery processes that access XML. The XQuery modules can be used as SOA bindings in Web services that run in WebSphere Application Server. This means that we have an additional possibility to reach data stored in DB2.
- DB2 can also be accesed by PHP Web services deployed in WebSphere Application Server, through the "*Zend Core for IBM*" package. Zend Core for IBM is a certified PHP development and production environment that includes tight integration with the DB2 family of database servers, and native support for XML and Web services. For more information, including usage scenarios, please consult:

  `Redbook Powering SOA with IBM Data Servers, SG24725`

Features that implement QoS for the SOA architecture

- The positioning of WORF inside the WebSphere Application server enables DB2 service providers to take full advantage of the QoS offered by WAS.

2.7.11 SOA systems management on z/OS

A service has not only a set of calls and responses, it has many other characteristics such as performance, availability, capacity, security (all together considered under the label "Quality of Service" or "QoS"). SOA is not only exposing how to call a service, but also defining a set of characteristics for how the calls will be serviced:

- How fast should the service respond (according to SLA)?
- When will the service be available (according to SLA)?
- Who may make calls to the service (authentication, authorisation)?
- How many calls can be done in a certain period of time (performance, capacity)?
- What calls must be logged and which attributes (security)?
- How should calls be routed (capacity)?

Figure 2-5 on page 40 shows the integrated SOA management view, which is exactly what we need. It is a group of Tivoli products that delve into the different layers of the SOA architecture and coordinate the information extracted, allowing for the management of the SOA infrastructure and of the services that run on the infrastructure.

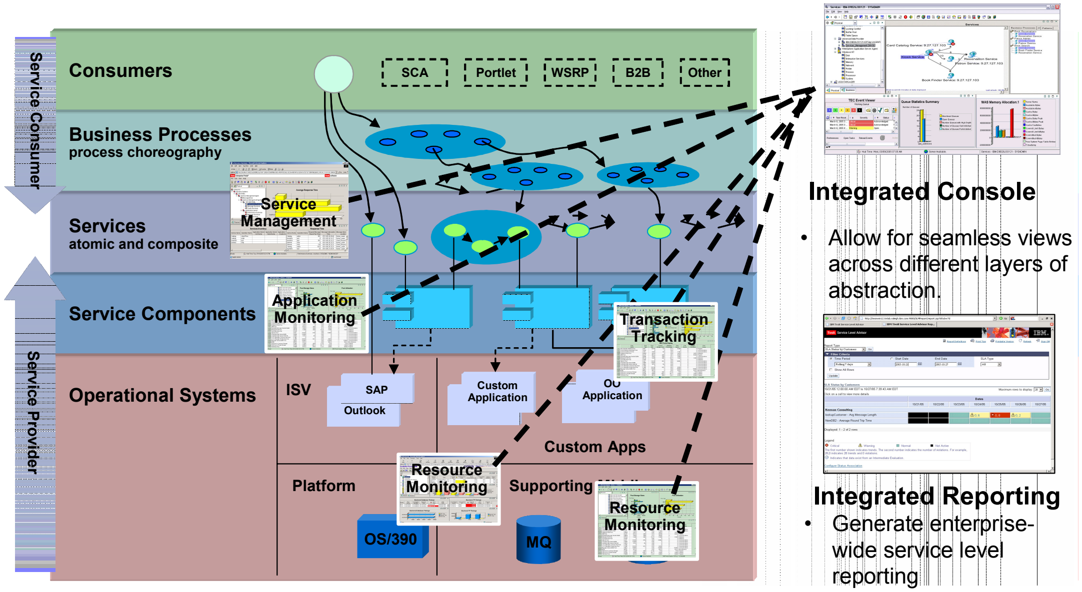


Figure 2-5 Integrated SOA management with IBM Tivoli on z/OS - a complete view

Figure 2-6 shows the Tivoli products mapped on the SOA reference architecture:

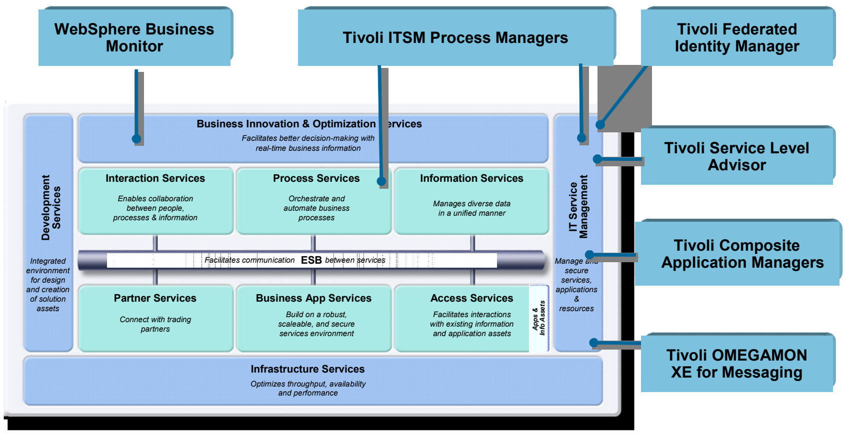


Figure 2-6 Mapping of IBM Tivoli products to the SOA reference architecture

2.7.12 IBM Tivoli Composite Application Manager for SOA V6.0

This section discusses the most important features of Tivoli Composite Application Manager (ITCAM) for SOA, that directly benefit the SOA.

Core features that implement QoS for the SOA architecture

- service problem identification and resolution through content views that allow a drill-down from services to applications to IT resources
- service management alerts and automation
- integrated service level reporting and monitoring
- understand, manage and track service performance

2.7.13 IBM Tivoli Composite Application Manager for WebSphere

This section discusses the most important features Tivoli Composite Application Manager (ITCAM) for WebSphere, that directly benefit the SOA.

Core features that implement QoS for the SOA architecture:

- quick problem analysis across components of the SOA reference architecture (Portal, J2EE, CICS, IMS)

2.8 Implementation options for the SOA architecture on z/OS

At this time we know the options available on each building block of the IBM SOA reference architecture, and we described in detail the functionality delivered by the implementations. It is time to create sets of products (what we call here implementation options), and see if the criteria set is fulfilled. We also identify the gaps between the options, and differentiate between them.

The main differentiator between the options described is the “depth” of the SOA implementation, and that is manifested in the level of the coupling of services and the amount of orchestration that is achieved.

> **Attention:** The SOA implementaion options should be seen as examples. Of coarse, it is possible to create variations of the three implementation options discussed. The purpose of the SOA implementation options is to show to what extent progress can be made towards SOA enablement depending on the products used.

2.8.1 SOA implementation option 1 - "service enablement"

The rationale behind this option is: "Let's see what this SOA enablement will give us for a very limited application set" (selected from several scenarios). Let us use some of the IBM tools and products available for SOA enablement, see how they work in production. We'll tread carefully and with limited budgeting.
In this implementation some applications were service-enabled and some reusability was achieved; we installed some pieces of the SOA infrastructure. Some SOA criteria were fulfilled."

The diagram in Figure 2-7 shows, for SOA implementation option 1, the positioning of the IBM products relative to the IBM SOA reference architecture.

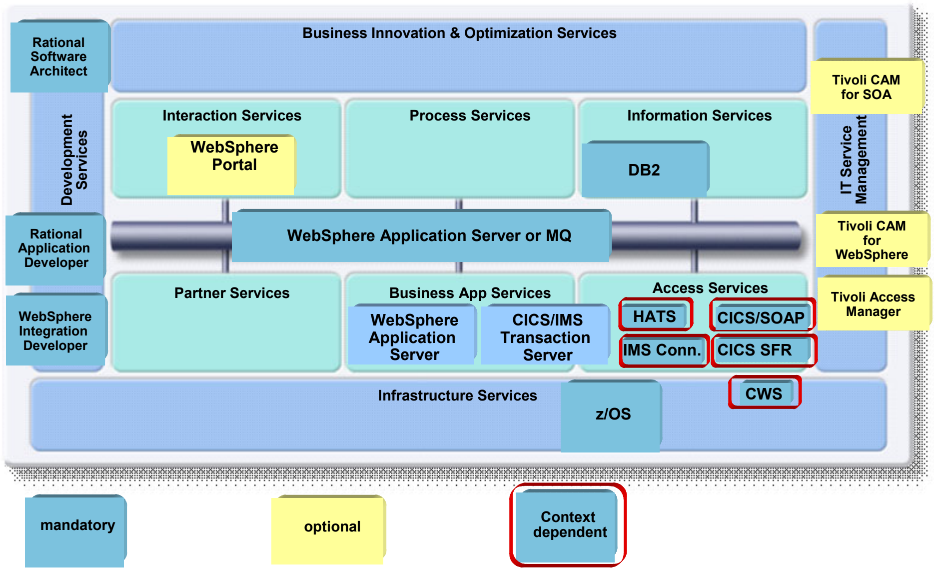


Figure 2-7 Mandatory and optional IBM products for service enablement

Table 2-1 on page 43 shows the characteristics of this SOA implementation option.

Table 2-1 Characteristics of SOA implementation option 1 - service enablement

Implementation Option 1	Characteristic
Name	Service enablement
When to implement	► need to service-enable a few applications ► need some reusability of services ► have some of the described scenarios in the enterprise, and want to migrate them to SOA ► have already some applications that may be converted easily as services
Infrastructure services	Provided by z/OS (mandatory)
Development services	Rational Application Developer or an equivalent tool (mandatory) or WebSphere Integration Developer (optional)
IT Services Management Services	Tivoli family (optional)
Business Innovation and Optimization Services	WebSphere Business Integration Modeler, WebSphere Business Integration Monitor (both not necessary at this stage)
Interaction Services	Websphere Portal (optional, depending on context)
Process Services	Websphere Process Server (not necessary at this stage)
Information Services	DB2, IMS DB .. (as required in the context of the enterprise)
Partner Services	Websphere Partner Gateway (not necessary at this stage)
Business Application Services	WebSphere Application Server (mandatory), CICS Transaction Server, IMS Transaction Manager (as required by the context)
Access Services	WebSphere Adapters (as required by the context), HATS, CICS Web Services support, CICS Web Support, IMS Connect, IMS SOAP Gateway (depending on scenario)

Implementation Option 1	Characteristic
ESB	WebSphere ESB **/** Websphere Message Broker (optional, only if the context requires, otherwise implement a light-ESB using WebSphere MQ)

Table 2-2 shows the various fulfillment degrees of SOA criteria, for the applications enabled as services (means "what we get by implementing this option").

Table 2-2 SOA criteria fulfilled in SOA implementation option 1 - service enablement

SOA criteria	Have we reached this?
modeling, constructing, deployment	*yes*, necessary tools are in place
management of services	*no* (Tivoli management not in place)
loosely coupled services	*moderate* (because no real ESB in place)
reusable services	*yes, but of limited use since no ESB is in place*
composable services (processes)	*no*, modeler and runtime for process server are not there
location transparency	*moderate* for MQ solutions, *no* for all the others (reason: no real ESB in place)
transport neutrality	*no* (no ESB with transformation in place)
legacy applications as coarse-grained service	*yes*
insight into business services	*no* (Tivoli management not in place)

2.8.2 SOA implementation option 2 - "service integration"

The rationale behind this option is: " we know what we might get out of SOA, we have already Web services projects and small implementations in the J2EE area. Let's start by putting a mini-SOA in place and see how it plays out. Select a few applications (preferrably from different scenarios, maybe a legacy one and a J2EE one), convert them to SOA in a tactical (using some of IBMs tools) or strategic (top-down analysis) way. Achieve some SOA criteria fast for some urgent projects. Integrate the applications that already implement Web services in the SOA architecture, see if reusability is achieved. If it plays well we'll expand".

It is “SOA-light” with the addition of a few more infrastructure components and with enlarged SOA-enablement . More SOA criteria were fulfilled, we are on the right way.

The Figure 2-8 shows the positioning of the IBM products relative to the SOA reference architecture.

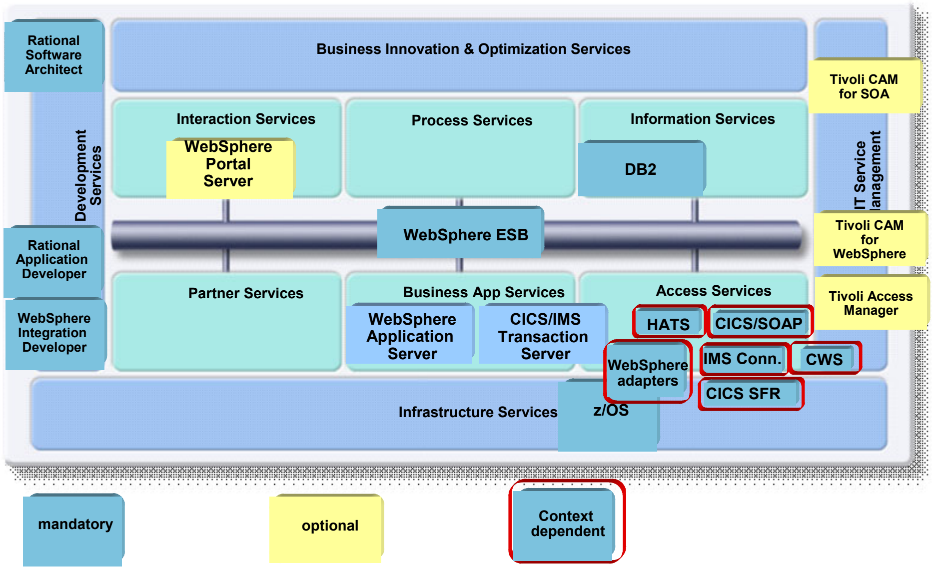


Figure 2-8 Mandatory and optional IBM products for service integration

Table 2-3 on page 46 shows the characteristics of this SOA implementation option.

Table 2-3 Characteristics of SOA implementation option 2 - service integration

Implementation Option 2	Characteristic
Name	Service integration
When to implement	► need to fulfill quite some SOA criteria in a more strategic way ► need reusability of the existing Web services, and a loosely coupled way of integration, inlcuding mediation ► have more time and budget to invest in a strategic SOA infrastructure ► have some of the described scenarios in the enterprise, and want to SOA-enable them ► have already some service-enabled applications that may be integrated
Infrastructure Services	Provided by z/OS (mandatory)
Development Services	WebSphere Integration Developer or WebSphere Message Broker Toolkit (mandatory) on top of the tools mentioned in SOA implementation option 1
IT Services Management Services	Tivoli family (optional)
Business Innovation and Optimization Services	WebSphere Business Modeler, WebSphere Business Monitor (optional and not necessary at this stage)
Interaction Services	Websphere Portal (optional)
Process Services	Websphere Process Server (not necessary at this stage)
Information Services	DB2, IMS DB ..(as required in the context of the enterprise)
Partner Services	Websphere Partner Gateway (not necessary at this stage)
Business Application Services	WebSphere Application Server (mandatory), CICS Transaction server, IMS Transaction Server (as required by the context)

Implementation Option 2	Characteristic
Access Services	WebSphere Adapters (as required by the context), HATS, CICS Web Services support, CICS Web Support, IMS Connect, IMS SOAP Gateway (context dependent)
ESB	WebSphere ESB or WebSphere Message Broker (either one is mandatory) for service providers that implement Web services

Table 2-4 shows the various fulfillment degrees of SOA criteria, for the applications integrated as services (means "what we get by implementing this option"):

Table 2-4 SOA criteria fulfilled in SOA implementation option 2 - service integration

SOA criteria	Have we reached this?
modeling, constructing, deployment	*yes*, necessary tools are in place
management of services	*no* (Tivoli management not in place)
loosely coupled services	*moderate to full* (ESB is in place)
reusable services	*yes, and because of the ESB also practical*
composable services (processes)	*no*, modeler and runtime for process server are not there
location transparency	*moderate* for MQ solutions, *yes* for all that use ESB
transport neutrality	*yes* (for all that use ESB)
legacy applications as coarse-grained service	*yes*
insight into business services	*no* (Tivoli management not in place)

2.8.3 SOA implementation option 3 - "process integration"

The rationale behind this option is: "We are in a bind. We have numerous projects that cannot progress because of the monolithic structure of the applications. We need to reuse components that are locked inside the applications, and we want to integrate them, but we don't have the architecture that allows this and also not the necessary infrastructure.

Our applications use several protocols and interaction models, but the whole thing becomes unmanageable as we install new applications or try to change existing ones. We need to put something in the middle, to do all this "translation" and to "loosen" the dependencies between the applications. We have urgent requirements for new applications, we have seen that we can create process flows that reuse some of our existing applications (or application parts), but we need to make out of those application parts "reusable services".

Our partners are connecting to our applications using various methods, and every new partner is a costly change. We know what SOA might do for us, and we trust the concept. We will apply the IBM methodologies for identifying business services, decomposing application, etc, but we will also use tactical solutions (wrapping) where necessary. Our application landscape contains instances of all initial scenarios.

We have the funding for this project, and we need a state-of-the art SOA implementation. We have to move fast on several areas simultaneously (in SOA-enablement and SOA-infrastucture), and achieve fast results, but we have to be careful to keep the SLA and performance levels as required".

It is, as the name describes, a full blown implementation of the IBM SOA reference architecture using a lot of existing IBM products. It fulfills all SOA criteria. It is recommended for a specific type of enterprise, with a specific set of requirements and a specific application landscape.

Figure 2-9 on page 49 shows the positioning of the IBM products relative to the IBM SOA reference architecture.

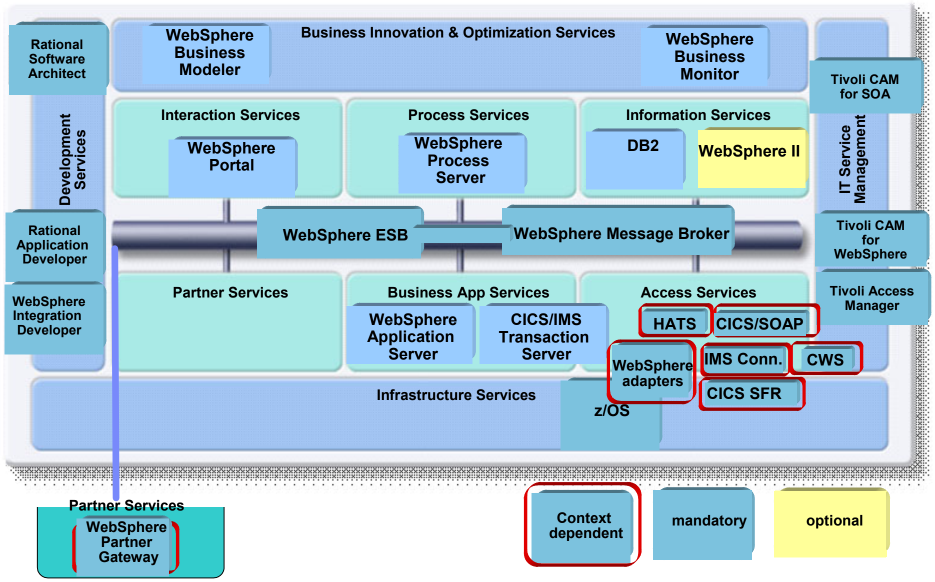


Figure 2-9 Mandatory and optional IBM products for process integration

Table 2-5 shows the characteristics of this SOA implementation option.

Table 2-5 Characteristics of SOA implementation option 3 - process integration

Implementation Option 3	Characteristic
Name	Process integration
When to implement	► need to completely fulfill SOA criteria ► have a high number of different connectivity requirements ► need high reusability of services ► need to execute a fast move to SOA, simultaneously in the area of SOA-enablement and SOA infrastructurei ► need a fast implementation of mediations, transformation, routing ► have all described scenarios in the enterprise, and want to migrate them fast to SOA ► have already some applications that may be converted easily as services ► need to integrate user interfaces and reach easily the services

Implementation Option 3	Characteristic
Infrastructure Services	Provided by z/OS (mandatory)
Deployment services	WebSphere Integration Developer (mandatory) on top of the tools mentioned for SOA implementation options 1 and 2
IT Services Management Services	Tivoli family (initially optional, to be installed when there are a number of services in place)
Business Innovation and Optimization Services	WebSphere Business Modeler (initially optional), WebSphere Business Monitor (as soon as the business requires it)
Interaction Services	Websphere Portal (mandatory)
Process Services	Websphere Process Server (initially optional, to be installed when process flows are necessary)
Information Services	DB2, IMS DB, Wesphere Information Integrator (as required in the context of the enterprise)
Partner Services	Websphere Partner Gateway (on distributed, if the context requires it)
Business Application Services	WebSphere Application Server (mandatory), CICS Transaction Server, IMS Transaction Manager (as required by the context)
Access Services	WebSphere Adapters (as required by the context), HATS, CICS Web Services support (context dependent), WebSphere IICF (context dependent)
ESB	WebSphere ESB *and/or* Websphere Message Broker

Table 2-6 on page 51 shows the various fulfillment degrees of SOA criteria, for the applications integrated as processes (means “what we get by implementing this option”):

Table 2-6 SOA criteria fulfilled in SOA implementation option 3 - process integration

SOA criteria	have we reached this ?
modeling, constructing, deployment	*yes*, necessary tools are in place
management of services	*yes* (Tivoli management tools in place)
loosely coupled services	*full* (both ESB products are in place)
reusable services	*yes*
composable services (processes)	*yes*, modeler and runtime for process server are in place
location transparency	*yes* (for all that use one of the ESB products)
transport neutrality	*yes* (for all that use one of the ESB products)
legacy applications as coarse-grained service	*yes*
insight into business services	*yes* (Tivoli management products are in place)

2.8.4 Conclusion

The implementation options discussed in the previous three sections indicate how progress can be made in SOA enablement by adding products or solutions to infrastructure. We have also seen that in existing IT environments SOA should be implemented in steps and according to budget and time constraints. A “big bang” approach should be avoided.

3

Starting scenarios

This chapter describes possible scenarios that will be used as starting points for our discussions how to SOA enable an IT landscape. Each starting scenario covers one or more variations, and is described using certain characteristics.

We will not cover the reasons *why* to move from a starting scenario to an SOA. In this book, we consider that these decisions are already taken. We will focus on possible ways *how* to SOA enable the different scenarios.

We use the IBM SOA reference architecture to illustrate where the different building blocks and characteristics of a scenario fit into the overall solution. The IBM SOA reference architecture is described in 2.4, "IBM SOA reference architecture" on page 16.

The reason for mapping the starting scenarios to the IBM SOA reference architecture is to identify what is missing in the current scenario and therefore to be better able to define what is need when it comes to migrating the starting scenario towards an SOA. Because there is text in an IBM SOA reference architecture building block does not necessarily mean that the scenario is already SOA enabled.

The main starting scenarios that we will use throughout this chapter are:

3270 application 3.1, "Starting scenario - 3270 application" on page 55

Multichannel 3.2, "Starting scenario - multichannel" on page 62

Batch 3.3, "Starting scenario - batch" on page 71

Data integration 3.4, "Starting scenario - data access and integration" on page 78

Homegrown SOA 3.5, "Starting scenario - Homegrown SOA" on page 87

Within each scenario variations may exist.

> **Note:** It is very likely that an IT landscape on z/OS has a combination of the above mentioned starting scenarios. In that case, different transition approaches and different solution techniques may be required for each identified scenario and its associated applications and IT infrastructure.

3.1 Starting scenario - 3270 application

The 3270 scenario is a common scenario. It consists of a 3270 terminal connected to a host application that runs transactions in CICS or IMS. Data resides in DB2, a IMS DL/I database or a VSAM file.

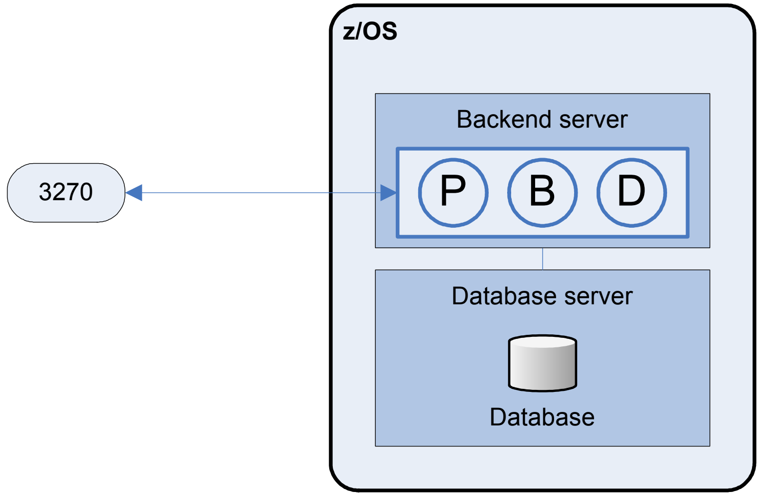


Figure 3-1 Conceptual overview of the 3270 starting point scenario

> **A note about notation:** In the diagrams in this chapter, presentation logic is represented by the circle with a "P". The business logic is represented by the circle with an "S", and the data access logic by the circle with a "D".
>
> Technical functions are represented by the circle with a "T".

3.1.1 A typical 3270 application

In many cases, a 3270 application has evolved during the years. It might have been developed during the 1970's or 1980's. The result of this is that there may be different coding standards and tecniques used, which is reflected in complex code to maintain.

Usually, the application is divided in a presentation and a business layer, but separation between those might not always be strictly implemented as illustrated in Figure 3-2 on page 56. Common technical functions are used for DB access.

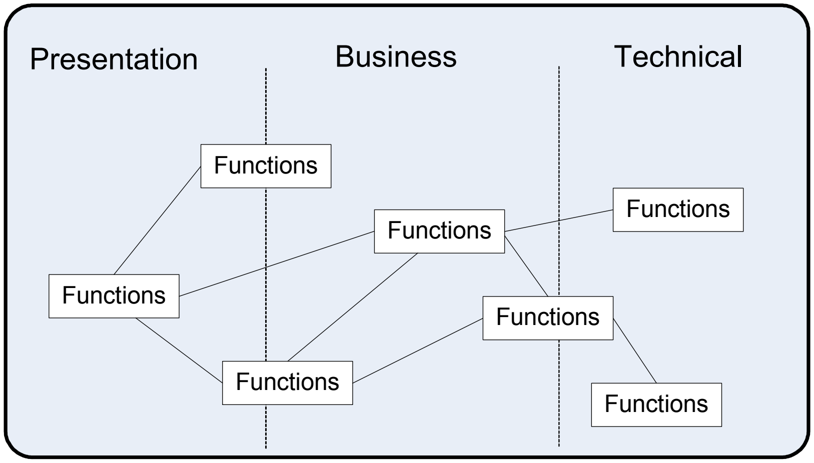


Figure 3-2 Example of separation between business, application and technical logic in a 3270 application

3.1.2 3270 application variations

In this section we demonstrate two technology variations of the 3270 scenario:

- One variation uses a CICS 3270 application as a starting point, as shown in Figure 3-3.

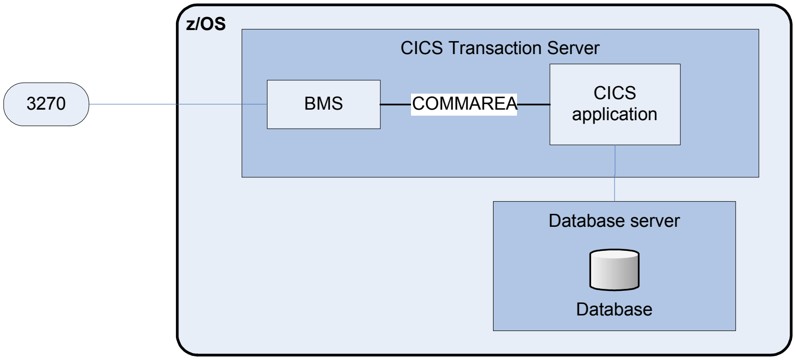


Figure 3-3 Variation showing a logical view of a 3270 CICS application

In this variation, the CICS application is structured using *Basic Mapping Support (BMS)* for the presentation logic and using the CICS COMMAREA for communication between the components, which results in distinct layers between presentation and business logic.

- Another variation uses a 3270 IMS application as starting point, as shown in Figure 3-4.

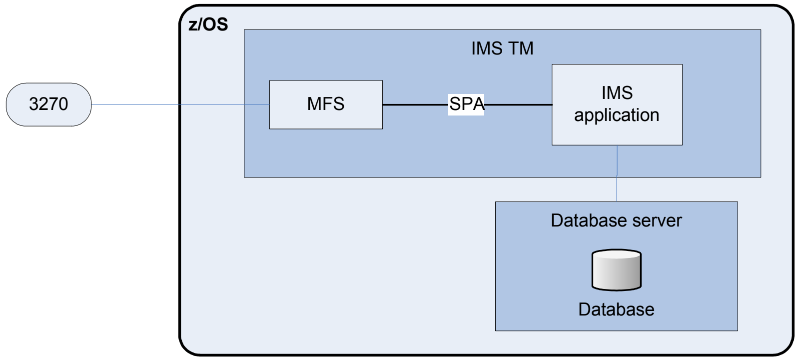


Figure 3-4 Variation showing a logical view of a 3270 IMS application

Like in the CICS variation, this application is layered, but using IMS techniques. *Message Format Service (MFS)* is used as the screen mapper, and *Scratch Pad Area (SPA)* is used for communication. Also in this variation there is a clear layering between presentation and business logic.

The two variations look very similar when illustrated using this notation. But later on, when it comes to the migration scenarios, there are quite a few different technology and product options to consider.

Of course there are other areas to consider than the actual user interface and the communication mechanism used within the specific transaction manager. An important area is the data model. If you want to SOA enable one 3270 (silo) based application, you may have one data model to handle also. If you have multiple (silo) applications, you probably have almost the same number of data models as well, with possible overlapping terms and entities. This is a complex area in itself and we cover data implications separately in 3.4, “Starting scenario - data access and integration” on page 78.

3.1.3 3270 application characteristics

It is interesting to see that in each application, old or new, we try to address the same things. Obvious is that we try to address presentation and data access, but, in fact, also the other types of service that we have defined in the IBM SOA reference architecture may already be present in traditional applications, such as the 3270 applications that we are talking about in this chapter.

Figure 3-5 on page 58 shows the characteristics of a 3270 application, in light of the IBM SOA reference architecture.

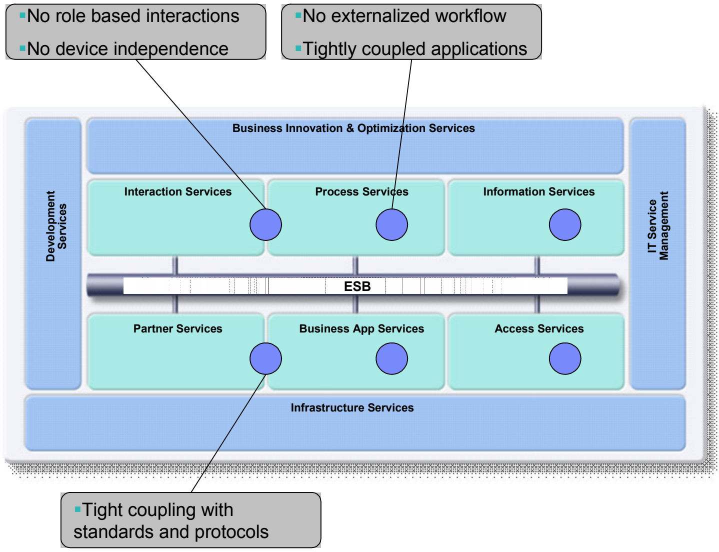


Figure 3-5 3270 scenario fit into the IBM SOA reference architecture

Looking at Figure 3-5, the characteristics of the 3270 application are:

- Presentation logic is implemented in CICS or IMS.
- Business logic layer is implemented in CICS or IMS as well.
- Data resides in DB2, an IMS DL/I database or VSAM files.
- Integration with external systems is based on either native techniques, such as APPC or TCP/IP, or WebSphere MQ.
- Most commonly used languages are Cobol, PL/I, C, C++, and in some cases even Assembler. In some cases the programs may already be written in the Java language.
- There is no support for device independence. Naturally, a 3270 application targets a 3270 device.
- The navigation flow for the dialogues is kept and implemented within of the application. The dialogues are tightly coupled with each other, and there is most probably not an external tool for coordinating the (screen) flow. The same is true for the *mediation*, which is hand coded.

Interaction Services

There are different ways for a user to access a 3270 application. Usually, it is done from a terminal window, but there may be a layer in between, for example using:

- Host Access Transformation Services (HATS) or Host On Demand (HOD).
- A “fat client” connecting to IMS using *Open Transaction Manager Access (OTMA)* and using *IMS Connect*.
- A “fat client” connecting to CICS using th *CICS Transaction Gateway (CICS TG)*[1] or using *CICS Web Support (CWS)*.

Some characteristics in the IBM SOA reference architecture that are usually not supported by a 3270 application are:

- Role-based interactions, where the application is adjusting according to a specific role.
- Device independence, where the same business logic can be invoked from different “channels”.

Process services

Process Services is a key element in an SOA and implies that there is a “point of control” keeping track of the flow and dependencies between the various service components in that flow. For a more comprehensive overview of Process Services refer to 2.4.5, “Process Services” on page 21.

We can generally say that there are no Process Services out of the box in a 3270 application environment, but some 3270 application environments may have gone far enough to include hand-coded Process Services to achieve ceratin workflow requirements.

What can be done quite easily in a 3270 application environment running in CICS or IMS is to keep a central point of control for the screen flow. However, screen flow is not the same as workflow and definitely not the same as Proces Services. In contrast with Process Services as part of an SOA, the dialogue flow within a traditional 3270 application is maintained by the transaction manager in which the application is running and is usually controlled from menus. There is no coordination done of the overall workflow, and there is a high dependency between the different dialogues in the application.

A workflow in an SOA may contain human as well as automated steps. Both forms of workflow are not supported by a 3270 application environment, but as said earlier, a lot can be achieved by hand coding.

[1] CICS TG only supports 3270 interfaces when using the External Call Interface (ECI) interface. This interface is only supported when CICS TG runs on a distributed UNIX® or Windows® platform.

Business services in an SOA, that are choreographed by, for example BPEL, are relatively coarse-grained. 3270 programs usually represent a low level of granularity. There will most probably not be a one-to-one mapping between a 3270 program and a service in an SOA.

Information Services

A 3270 application usually relies on DB2, IMS DL/I or VSAM as data sources. Access to data according to the IBM SOA reference architecture is done through *Access Services* as described in “Access Services” on page 61

ESB

On a high level, the role of the ESB is to connect loosely coupled services. As discussed in 2.4.3, “The Enterprise Service Bus (ESB)” on page 19, it includes the following fundamental services:

- Transport services
- Event services
- Mediation services

A 3270 application in itself is usually not enabled for an ESB, but there is some functionality provided by the infrastructure, which could be considered to have flavors of an ESB. This is the case, for example, when:

- the transport is provided by WebSphere MQ or “in-house” developed middleware
- WebSphere Message Broker (WMB) is used for integration
- there is some level of event handling and mediation implemented in the application logic itself

Partner Services

There are many standards and protocols for interaction with partners, such as:

- WebSphere MQ
- Sockets
- SNA / APPC
- SWIFT
- EDI

As with Interaction Services, the IBM SOA reference architecture states that there should be a loose coupling between the business logic and the partner services logic, preferably through an ESB. This is probably not the case in the 3270 application scenario.

Business Application Services

According to the IBM SOA reference architecture, the components found in this building block are usually created with a component model, or service perspective in mind. Examples of implementations are J2EE and possibly .Net. Therefore a typical 3270 application would not reside here.

Access Services

For our 3270 application scenario, access services would need to provide support for the following enviornments:

- CICS Transaction Server
- IMS Transaction Manager
- WebSphere MQ
- DB2
- IMS Database Manager
- VSAM

Infrastructure Services

The infrastructure in this scenario is completely based on z/OS and the qualities that this environment provides.

Development Services

Development of a 3270 application usually includes screen design tools and templates for code structure (e.g. database access).

The development environment does not necessarily need to be based on a GUI, but in most cases an *Integrated Development Environment (IDE)* provides more productivity and better time to market. An IDE is typically used on a Windows or Linux workstation and the applications are deployed onto the z/OS environment.

New development is most likely done in Cobol or C/C++, but Java is picking up as well.

Depending on the application architecture and the programming style, there may not be a clear separation of presentation and business logic in a typical 3270 application.

IT Services Management Services

As with Infrastructure Services, described in “Infrastructure Services” on page 61, z/OS provides all the functionality for security, automation, provisioning and so on. Additionally, IBM Tivoli or other third-party tools may be in use.

3.1.4 Challenges when moving to an SOA

In an SOA reusability and "separation of concerns" are very important aspects. To gain full benefit from an SOA, most 3270 applications will need to be decomposed and refactored to achieve the right level of service granularity and "separation of concerns", which will mean at a minimum loose coupling between the user interface logic and the business logic. The code of existing 3270 applications may be difficult to decompose.

There are probably identical code blocks across different applications doing nearly the same, possibly acting on the same data.

Business rules are embedded in the code. Changes to business rules imply large efforts to analyse, design and implement such changes in the code.

It is not uncommon that there are many different programming styles used, since these applications are developed from the early 70's until today.

3.2 Starting scenario - multichannel

With *multichannel*, we mean an architecture that is able to support various *channels*. A channel is associated to the usage of a specific *device*. As illustrated in Figure 3-6 on page 63, examples of channels are ATM, fat client, browser, PDA, smartphone etc. All these different channels connect to the same infrastructure and make use of the same business and data access logic.

Therefore, the multichannel scenario is hard to describe as one scenario. It is rather a combination of several different integration and access styles.

As a starting point, it is defined with the following attributes:

- There is support for different channels, e.g. browser via HTTP/HTTPS, fat client via RMI/IIOP, ATM via TCP/IP etc.
- There is usually a mid-tier, where the different channels converge. Usually this mid-tier hosts components that wraps business and technical functionality.
- Back-end services and transactions are callable via a standardized API or through a service interface.
- Using client/server concepts, meaning there is a fat client, with both business and presentation logic, and sometimes even with direct data access from the client.

But there are also other characteristics that could make up a multichannel application:

- Messaging-centric systems, e.g. native WebSphere MQ-enabled IMS and CICS transactions.
- ‘Headless’ transactions without a service interface.
- Web and J2EE applications, developed as silos, or as reusable component based application architectures (non-SOA).
- APPC/CPI-C applications.

Figure 3-6 shows the typical multichannel architecture, in which all channels communicate with a common layer implemented in a J2EE application server. The common layer then accesses all sorts of backend transaction and database servers for business logic and data access logic. The J2EE application server may also perform business and data access logic itself.

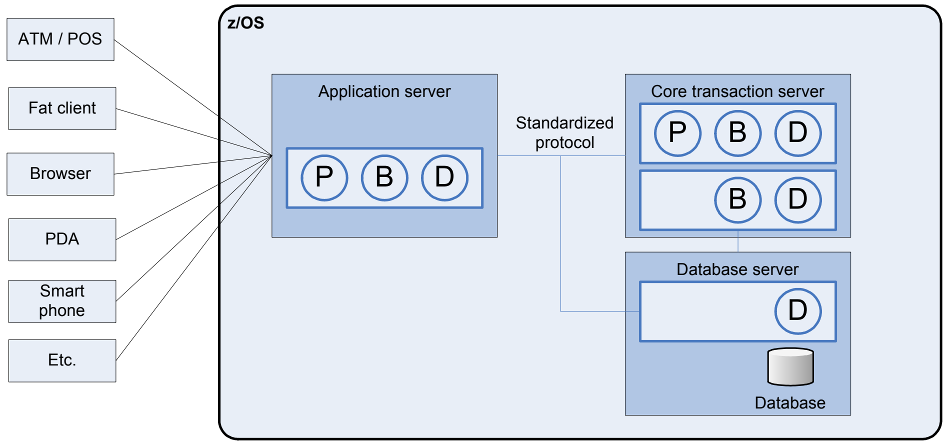


Figure 3-6 Conceptual overview of the multichannel starting scenario

The presentation logic for the different channels, is handled by the components in the application server. The core transaction functions are exposed through *proxy* components in the application server. This proxy functionality will most probably be found in the business layer of the application server. This is illustrated in Figure 3-7 on page 64.

We assume that the core transaction server logic has been divided in business and data access functions. This is illustrated in Figure 3-8 on page 64.

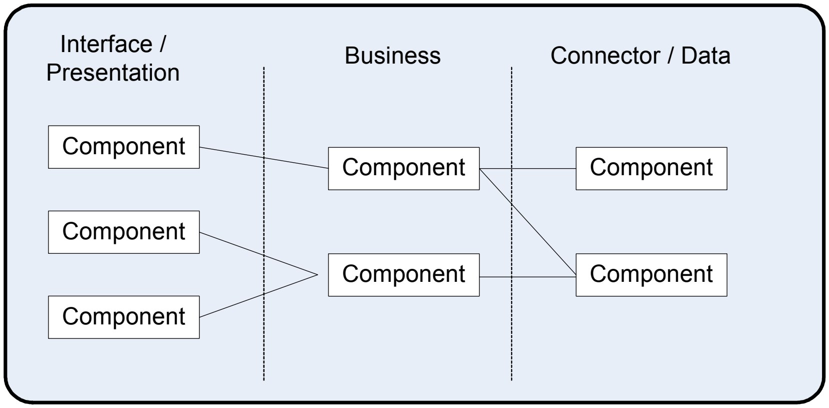


Figure 3-7 Logical structure of the component layer in the application server

The core business functionality is present in the backend transaction server. It may be accessed through, for example, connectors or messaging middleware like WebSphere MQ.

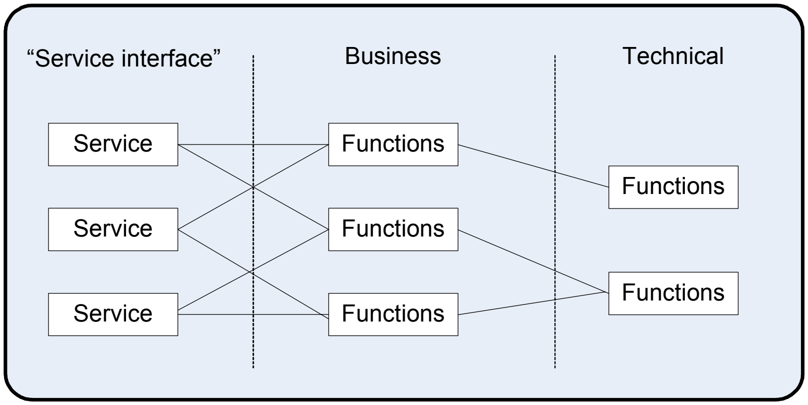


Figure 3-8 Logical structure of the core functions in the transaction server layer

Data in this scenario resides, as usual, in DB2, DL/I databases or in VSAM files.

Integration with external systems is based on WebSphere MQ, APPC or TCP/IP or other protocols over TCP/IP, such as HTTP.

Most commonly used languages in the transaction server are Cobol, PL/I, C and even some Assembler. In the application server typically Java, but Cobol, PL/1 and C /C++ may also be present depending on the server technology used.

There is a certain degree of loose coupling, especially if WebSphere MQ is used.

A large degree of encapsulation is implemented and reuse is provided through published interfaces. However, different services may have been implemented for similar or even identical functions.

The Quality of Service (QoS) is at a high level.

3.2.1 Multichannel variations

The application server may be a J2EE server like WebSphere Application Server (WAS) or the middle-tier may also be implemented in a transaction server like CICS or IMS (although it will be more difficult to support multiple channels).

In either case the integration with the core business transactions may go through messaging middleware like WebSphere MQ. If the application server is a WAS, the core backend transactions may be accessed through connectors.

If the application server is a CICS or IMS transaction server it may also be accessed by direct, synchronously calling functions.

Different technologies can be exploited for dynamic workload balancing. In the frontend application server, dispatchers, sprayers and eventually Workload Manager (WLM) can be used.

A proprietary service directory may be implemented providing some level of abstraction and indirection of the function invocations. Location transparency may also be provided and reuse is enforced by publishing of functions and their interfaces.

In case fat clients are used, all the presentation logic and some business logic exists on the client. It is not likely that it is possible to reuse any of these code entities directly.

3.2.2 Multichannel characteristics

The architecture in the multichannel scenario assumes that there has been some degree of standardization and structuring around common services and communication protocols.

However, the infrastructure components are still tightly coupled. Usually, there are different technologies and standards involved. There are also most probably multiple implementations of similar core functions.

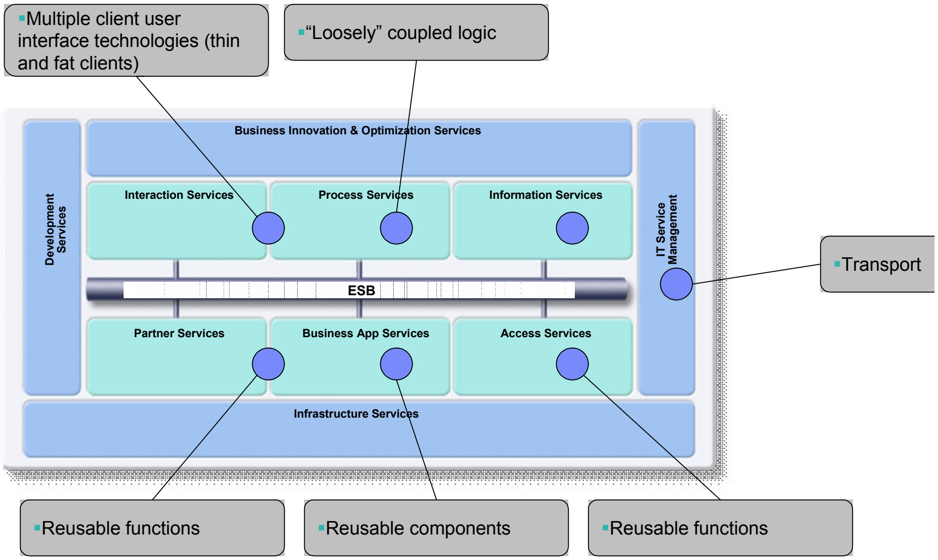


Figure 3-9 Multichannel starting scenario fit into the SOA reference architecture

As we did for the 3270 application starting scenario we will now discuss some characteristics for the multichannel starting scenario, in the context of the IBM SOA reference architecture.

Interaction Services

Significant for the Interaction Service in this scenario, is that there are multiple client user interface technologies involved. There may also be multiple communication protocols that need to be supported. This is the reason why we have named this starting scenario “multichannel”.

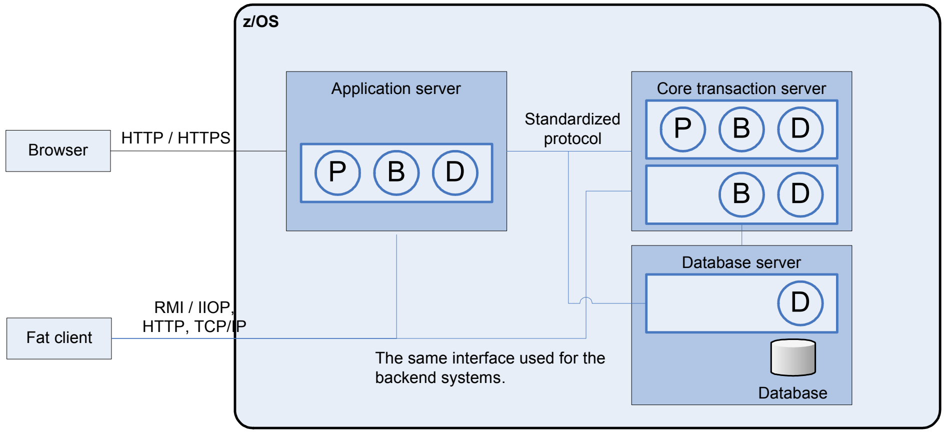


Figure 3-10 Multichannel starting scenario considerations

Client user interface technologies

In the case of a Web browser based solution, there is no logic executing on the actual client, except maybe for some JavaScript™ or a Java applet. The navigation and the dialogues are created on the application server tier, for example using a JSP™ or servlet. The entire flow, and the Model / View / Controller pattern is in this case implemented and controlled from the application server tier.

In the case of a fat client, the workflow and dialogues are implemented in the client tier. The client handles all the dialogue flows, and knows which programs to call. Those interfaces may be provided by the application server, or maybe even directly from the core transaction server or the database server.

Communication protocols

As illustrated in figure Figure 3-10, the protocol used for the thin (browser) client is HTTP/HTTPS. For the fat client, there are several options, for example, HTTP, RMI/IIOP, TCP/IP or WebSphere MQ.

There may be some level of role based interactions in this scenario. In the thin client case, access to the server be provided by a portal running inside the application server tier. In the fat client case, it could be implemented by a framework or technology in the application itself. The drawback of implementing role based behavior in both thin and fat clients using this model, is that there will be very limited reusability between the two, and the maintenance cost is high.

There may also be some level of device independence implemented in the application server tier. For example, it may not be a very big effort to adjust to a

PDA or other small devices, as long as they use a standard protocol, such as HTTP.

Process Services

Although there may be a fairly loose coupling between the presentation logic and the business logic, the application flow is still not coordinated by external tooling in the area of workflow or process choreography. It is contained within the application, in most cases making use of a framework.

Information Services

In our scenario, there is no direct access to operational data, and the application usually relies on DB2, IMS DL/I or VSAM as data source. Access to data is according to the IBM SOA reference architecture done through access services as described in “Access Services” on page 61.

ESB

In this scenario, there may be some level of ESB functionality, and that is probably in the area of transport services. We use a standardized middleware for accessing the functions provided by the core applications.

Although it may be implemented in different places, there may also be some kind of mediation going on in the application server tier, or in the fat client.

There may be some level of event handling implemented in the application logic.

Partner Services

There are many standards and protocols for interaction with partners. In this scenario, there might be some reusable functions evolving.

For example:

- WebSphere MQ
- Sockets
- SNA / APPC
- SWIFT
- EDI

As with interaction services, the IBM SOA reference architecture states that there is a loose coupling between the business logic and the partner services logic, preferably through an ESB.

In this scenario, it may be that the functions provided by the application server tier act as an entry point to services. But there are still in-house developed solutions, which are not serviced or callable by an external standardized tool.

Business Application Services

This scenario is using functions in the Business Application Services building block. For the thin client, the user interface, navigation and business logic reside in the application server. Most likely implemented using J2EE, exposing reusable components.

Access Services

Access Services are heavily used for accessing the backend core functions. Reusable functions are created in the core backend systems. Support is needed for accessing:

- WebSphere MQ
- CICS Transaction Server
- IMS Transaction Manager
- Proprietary middleware may be implemented

The backend core functions need support to access data in:

- DB2
- IMS DL/I
- VSAM

Infrastructure Services

The infrastructure in this scenario is mostly based on z/OS and the qualities that this environment provides.

Since there is additional support for different clients in this scenario, there are also requirements for other infrastructure services. E.g. management of Windows, Linux and PDA clients. So there are parts of the infrastructure where the the z/OS level of QoS cannot be guaranteed. This is especially true with a fat client scenario.

Development Services

This scenario introduces a heterogenous development environment. Development in this kind of scenario spans several technologies, standards and languages, such as:

- Core transaction server.

On the technical level, support is needed to modify the backend transactions so they can provide standardized access through the chosen middleware interface.

Some kind of a dictionary is required to keep definitions and requirements of the services provided by the transaction server.

Development is most likely done in Cobol, C/C++, PL/I or Assembler, and in some cases Java.

- Application server.

  A development environment for J2EE is required in order to build the components for the application server tier.

  The components that are deployed to this tier, support presentation logic as well as some business logic. Support for calling the backend core functions is provided through APIs and frameworks.

- Fat client.

  A development environment is required to support the specific platform of the fat client. Fat clients may be deployed on for example Windows, Linux or PDAs.

IT Services Management Services

As described in “Infrastructure Services” on page 61, z/OS provides the functionality for security, automation, provisioning etc. There are also client platforms and various communication protocols to manage. Support is required for more environments than just z/OS .

3.2.3 Challenges when moving to an SOA

In this scenario there has already been implemented some level of separation and decomposition of the presentation and business logic. But usually there are overlapping functions deployed on different platforms.

Different standards and technologies are used, which could make it hard to reuse functionality.

There may also be different levels of granularity on the different service interfaces, because the services are tailored for a specific channel.

Key questions to consider when migrating to an SOA are:

- Keeping QoS at a high level.
- Reusing core services and data. Can the applications be reused, or do they need to be rewritten?

- Enablement of open standards. Which standards to use?
- Client technologies. Are Web based solutions based on Portal, Ajax, or rich client good enough? Or do the user interactions require the characteristics of a fat client.
- z/OS integration with distributed technologies (.Net).

3.3 Starting scenario - batch

A typical scenario probably present in all enterprises is the batch scenario. Figure 3-11 gives an illustration.

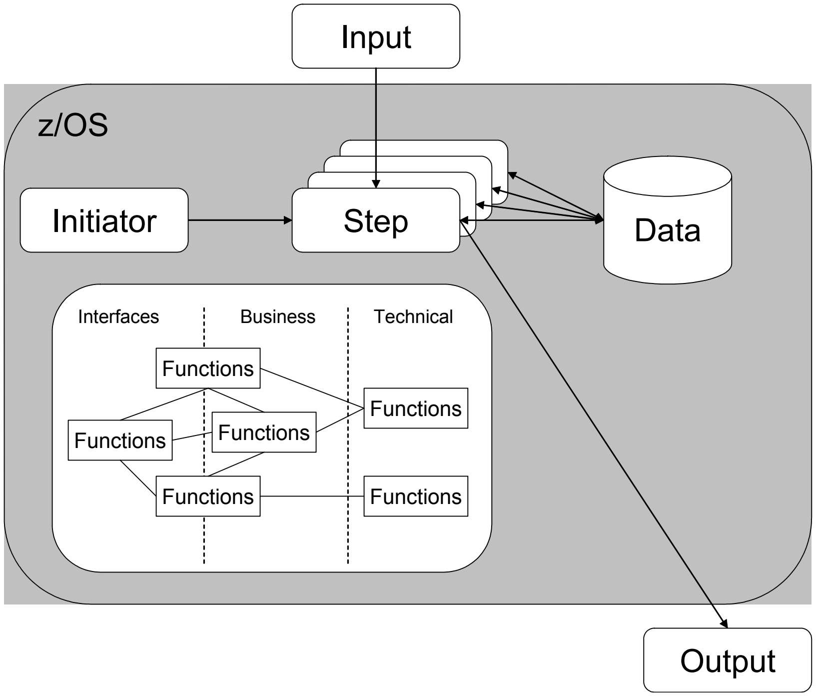


Figure 3-11 Logical overview of the batch starting scenario

The batch job is initiated inside z/OS, either triggered:

- from a scheduling mechanism or tool, such as *Tivoli Workload Scheduler (TWS)* for z/OS (formerly OPC)
- directly from TSO
- via the internal reader in a transaction server like CICS or IMS

- by WebSphere MQ (WMQ) or WebSphere Message Broker (WMB)

A typical batch job consists of one, but usually more steps, organized by means of the JCL.

The application logic in a program step may include a multitude of program modules. The programs implement a mix of interface logic, business logic, data access logic and technical logic, like for instance logging.

Each individual step may receive input from different sources like:

- JCL parameters
- SYSIN data sets
- temporary data sets from earlier steps
- persistent data sources like databases, keyed data sets or Generation Data Groups (GDG) data sets
- WebSphere MQ queues from processes inside the enterprise or from a business partner

and produce output to different medias and locations like:

- return codes used to manage the subsequent flow of the job, information to the system management infrastructure and possibly involving the operations staff.
- temporary data sets to subsequent steps. May be used for pure temporary information, subsequent printing or perhaps EDI or FTP processing and so on.
- persistent data sources like databases, keyed datasets or GDG data sets
- WebSphere MQ queues internally located or occurring at a business partners site
- The printing subsystem via SYSPRINT

The programming language is typically a third generation language or it could be a fourth generation language.

In the batch scenario the Quality of Service (QoS) is at a very high level. High availability and continuous operations may be achieved by running several LPARs in a parallel sysplex or a Geographically Dispersed Parallel Sysplex™ (GDPS®) with dynamic workload balancing provided by tools such as WLM. And scalability may be provided simply by adding more capacity. In many cases, very large and complex batch environments run unattended or with a minmum of support staff on-site.

The application flow is characterized by:

- high volumes of data to be processed
- high performance needed to process the high volumes of data

The jobs usually run unattended, are scheduled automatically and recovery from error conditions is usually happening automatically as well. Exceptions may require human intervention by an operator.

There may be several dependencies, for instance:

- Must run in batch service window due to dependencies on Online Transaction Processing (OLTP).
- Must run in a certain sequence with predecessors and successors.
- Dependencies on time, such as which time of the day or which day of the month.

3.3.1 Batch variations

Examples of variations of the application design are the following:

- All logic is implemented in one job step, with or without a clear separation of different types of logic.
- Separation of types of logic has taken place and implemented in separate job steps as well, for example, one step for interface logic, one step for data access and so on.
- Separation of types of logic has taken place and implemented in separate subroutines of one program.

3.3.2 Batch characteristics

Again, we will describe the characteristics of this scenario in the context of the IBM SOA reference architecture and thinking about possible service enablement.

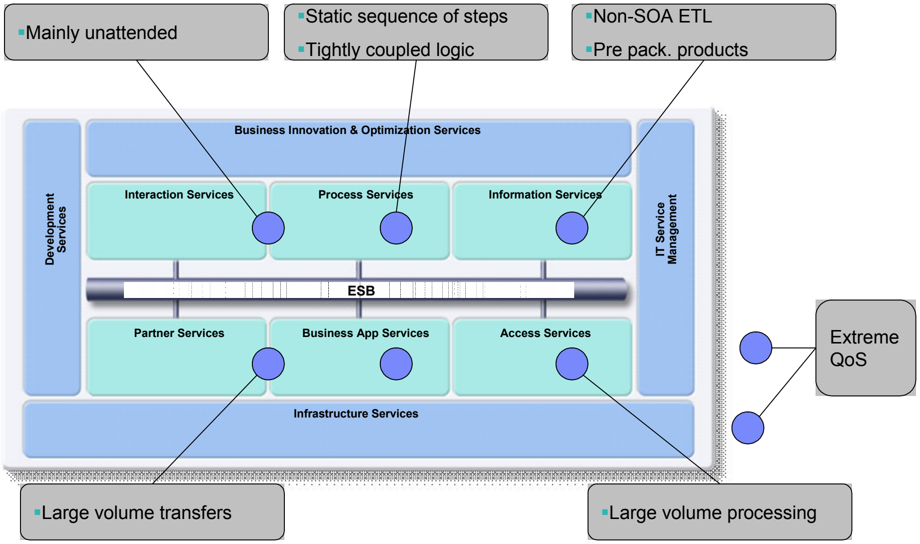


Figure 3-12 Batch scenario fit into the IBM SOA reference architecture

Interaction Services

The jobs are initiated by a scheduler or manually. No user interface is involved from an application point of view.

Operator intervention may be required, either for device mounting or handling of events and exceptions.

Batch jobs may be triggered by messages entering a WMQ queue used for reception of data or maybe just for triggering.

There may be some calendar or mail integration provided to inform business users of the progress and completion of the job.

Process Services

To some extent there is a certain workflow within a batch environment. There are dependencies between jobs and between steps in jobs. Jobs and job steps may happen conditionally, depending on the outcome of another job or job step. Those dependencies are usually maintained, monitored and executed from a job scheduler tool.

The flows of jobs and job steps is usually quite static and at the time of execution there is no human intervention, unless an exception takes place.

Information Services

Batch is an excellent environment for extracting, transforming, processing and loading the vast amounts of data present in the databases, for instance for building and maintaining a Data Warehouse or creating daily business reports.

ESB

Integration between components (programs executed in jobsteps) is typically taking place by means of data sets, such as GDG, VSAM, QSAM or sequential data sets. Also UNIX files are possible, as well as so-called “TEMP” data sets. Also, WebSphere MQ is a possibility within a batch environment, in which case programs in jobsteps read from or write to queues in WebSphere MQ.

Obviously, in traditional batch environments we do not see the full function ESB positioned for a modern state-of-the-art SOA, as discussed in 6.3, “Stage two - “Service Integration”” on page 223. Everything needed to transform data is done inside programs and routing is typically done in a very static way.

A limited degree of location transparency for logic and data is present. Job initiation is managed by a scheduler. The steps and their interrelations are described by JCL and processed accordingly. The physical representation and location of data is typically determined by data and queue definitions and only indirectly referenced from the program logic.

Partner Services

Data is interchanged with business partners through:

- WebSphere MQ
- EDI
- FTP
- and possibly proprietary protocols

Interchange is handled in large batches with multiple entities included, not as individual records or messages.

Business Application Services

There may be a very limited interaction with existing service enabled functions.

Access Services

Access Services are provided by the data and database access methods:

- DB2 delivers transaction (2-phase commit) support and relational data structures which may have a relatively loose coupling to the program logic.

- IMS DL/1 also delivers transaction (2-phase commit) support, but the data structures are more tightly coupled to the program logic.
- VSAM file support.
  Data structures are very tightly coupled to the program logic.
- QSAM dataset are used for transportation of temporary data between the job steps, storage for persistent state data or parameters, or as a container for large batches of data to be archived or sent to printing.
- WMQ delivers asynchronous persistent and non-persistent messaging between steps or jobs.

With DB2 there may be a relatively strong data structure based on application, department or even enterprise level data model.

With the other access methods, the data structure is not likely to be on a higher level than department and in many cases even just on application level.

Infrastructure Services

Infrastructure Services are provided by:

- z/OS delivering high performance, accountability, auditability, system integrity, security and so on
- Parallel sysplex or GDPS delivering high availability, continuous operations, scalability
- A scheduling tool such as Tivoli Workload Scheduler for z/OS
- Workload balancing through WLM
- Security management through RACF® or an equivalent SAF-based product
- System Automation

Development Services

Programming tools are used for direct coding, possibly with language sensitive editors.

Programming languages typically include Cobol, PL/I, Assembler, C/C++ and REXX, but again, Java is picking up here as well.

The programming components consist of source modules, load modules, listings and others. They may be managed by dictionaries or repositories along with other development components like data description and analysis documents and supported by version control and change systems.

IT Services Management Services

IT Services Management Services are provided by Tivoli or other 3d party tools.

3.3.3 Challenges when moving to an SOA

Typically, millions of records are processed in a batch jobs. The approach is that you initiate one function at a time processing all the records in one or a few logical loops with only one instance of the initial and termination logic. This is very efficient. If all functions including the initial and termination logic would be invoked for each data record, performance would decrease by orders of magnitude. Also, many times, operations on data are grouped and a set of operations can only be considered complete and succesful if all individual operations have been complete and succesful. Many times, batch output contains header and trailer records, needed for autiding purposes.

In many cases, job steps exist before and after a program to just do sorting, grouping or aggegration of records, and creating totals for auditing at the same time. In this case, there is hardly any business logic executed.

The volume itself is a challenge and requires extremely high performance and high throughput. May batch service windows just fit and there cannot be any delay in the jobs without jeapordizing the OLTP window.

Moving batch components into an SOA will be a challenge, but, as we will discuss in 5.4, “SOA implementation scenarios for batch” on page 183, there are ways to integrate batch into an SOA in a “light” manner. There are different points of view here.

Depending on the transition approach and the level of SOA maturity required, batch logic may need to be refactored to achieve the right level granularity and thus reusability. Decomposition of batch logic may not be easy. There are probably lots of nearly identical code blocks across different batch jobs doing nearly the same functions, possibly on the same data.

The jobs run smoothly with high performance and reliability. Any type of change will most probably decrease the Quality of Service to some extent.

In general, there is a risk attached to make big changes in the z/OS batch environment. It may affect reliability and performance. Especially if the changes are of a structural or architectural kind.

3.4 Starting scenario - data access and integration

We describe here the architectural scenarios representing *data access* (from batch and transaction-oriented systems) and those representing *data integration*.

When we speak about data integration we mean the process by which the information (stored on multiple, disparate data assets across the enterprise, but representing, for different information systems, the same piece of information - for example a customer record) are presented to the applications in an unified, trusted way and are consistenly updated.

The variations of the scenario show the ways most enterprises implement data integration today.

3.4.1 Data on z/OS

Most of the enterprises have a mixture of data, accessed by z/OS-based applications. We can categorize the data storage on z/OS as follows:

- Relational Database Management System (RDBMS), in most cases this is DB2.
- Non-relational database management system, which is usually IMS DL/1.
- Data sets, which are transactionally accesible using different file access methods (sequential, indexed, non-indexed etc.), for example VSAM.
- Sequential MVS™ data sets (partioned, GDG)
- UNIX files

In most cases XML can be stored natively. DB2 and IMS support native XML data storage. Sequential MVS data sets and UNIX files can contain XML too.

Each of thes types of data storage has a specific access method. The usual way to get to this data is through:

- The support for data access delivered in the traditional transaction servers like CICS and IMS. CICS and IMS are highly integrated with DB2, IMS DL/1 and VSAM datastores.
- A Java based application server (WebSphere). Java provides standard APIs for relational (SQL-based) database access by means of JavaDatabase Connectivity (JDBC™) and SQLJ and UNIX sequential file access by means of java classes. IBM also provides specific frameworks to access sequential and VSAM data sets from Java. Refer to J*ava Stand-alone Applications on z/OS Volume II*, SG24-7291, for details.

- The batch support delivered by the operating system.
- Or, through the positioning of an additional integration layer (with different depths and functions, starting with an ”integrator” and ending with a “data warehouse”).

3.4.2 Data access and integration variations

In the next sections we will discuss a number of variations of the data access and integration scenario:

- Direct data access in “Direct data access” on page 79.
- ETL (Extract, Transform and Load) in “Batch ETL (file transfer)” on page 80 and “Messaging ETL” on page 81
- Data replication in “Data replication” on page 82
- Data integration and data warehouse in “Integrator / Data Warehouse” on page 83

Note: Note that the data access scenario is usually existing in combination with any ot the other scenarios discussed in this book (3270 application, multichannel, batch or homegrown SOA).

The ETL, data replication and data integration scenarios might exist stand-alone on z/OS. This means that only the data and eventually the infrastructure and tooling for data extraction, replication and integration are located on z/OS, but all the applications using this data are located elsewhere. This would be the case when z/OS is only used as a “data server”.

Direct data access

The scenario of direct data access is used in most enterprises today; it is shown in Figure 3-13 on page 80. The characteristic of the scenario is that the data access logic is tightly embedded within the application (this means that the application has to know - and is dependent of - the structure of the data.) Changes in the data structure and representation (due to application requirements to change the data model) are very difficult to implement and maintain. The access method is determined by a combination of the programming languageand the type of data storage being used.

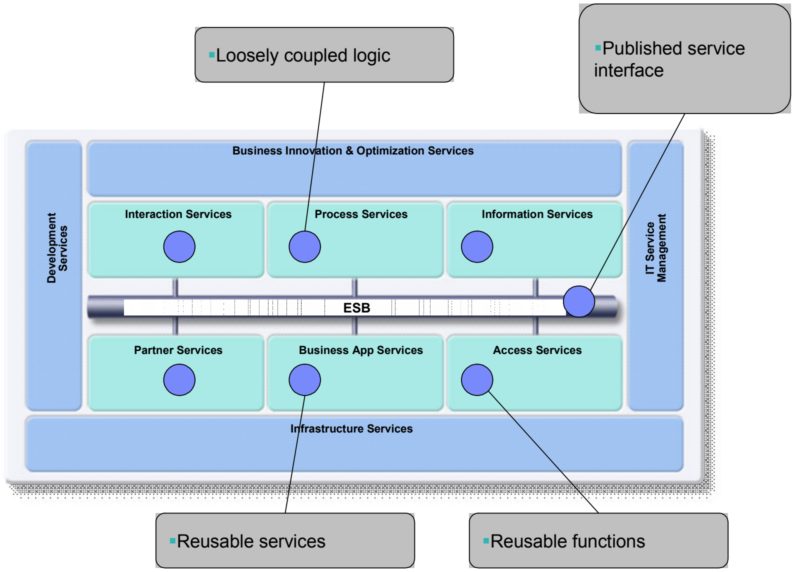


Figure 3-13 Direct access to data

Batch ETL (file transfer)

The very well known scenario of *batch ETL* (*E*xtract *T*ransform *L*oad) is used in most enterprises today. It was necessary to implement such an approach because of the new application requirements and technology changes collide with the existence of legacy applications.

Since most applications are "tightly coupled" with data, ETL was the technology that allowed the applications, with no changes on the application side, to have the "consistent" view of the enterprise data at least at one point, namely at the beginning of each business day (meaning after the nightly batch runs that synchronises all data assets). The scenario is shown in Figure 3-14 on page 81.

As we can see, in this case the task of transporting the information updates, storing the information in the correct format, and achieving consistency of the data is relegated to the integration layer that functions totally separated from the application. In fact the integration layer works usually only when the applications do not access the data!!

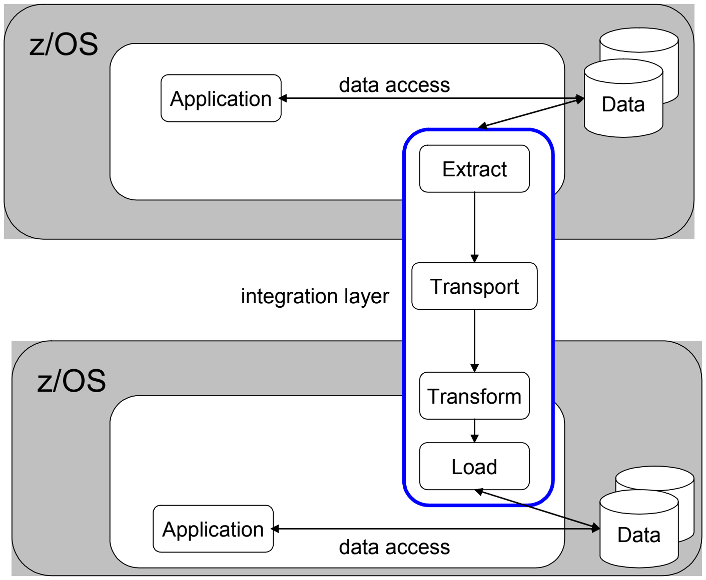


Figure 3-14 Integration through batch oriented ETL

Messaging ETL

A variation of the ETL scenario exists when the applications send the “updated view” of the information by means of a messaging infrastructure (for example, WebSphere MQ). We name this scenario *messaging ETL*. This is shown in Figure 3-15 on page 82.

In this case the integration layer is used by the applications to tranport the “updated information”, but the task of transforming and updating the information is now done by the applications themselves.

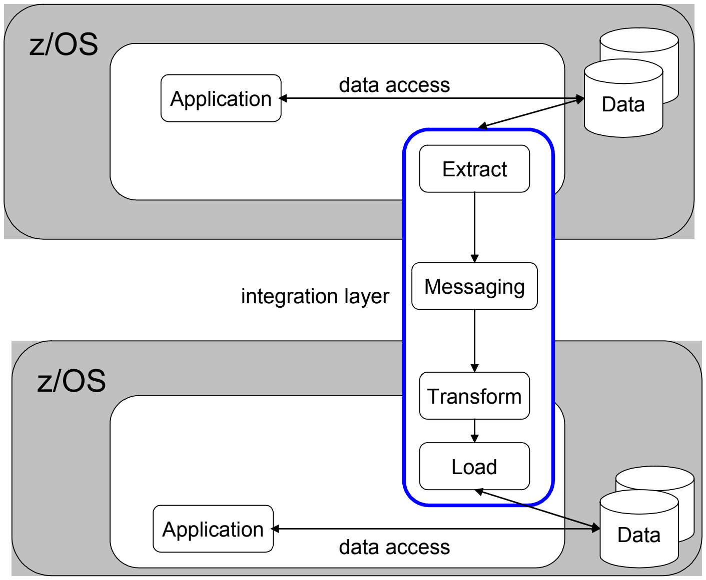


Figure 3-15 Integration through messaging ETL

Data replication

This *data replication* scenario can be found when the enterprises use data propagation technologies (for example DB2 data propagator, but also technologies for establishing consistency for VSAM data sets). Figure 3-16 on page 83 shows this scenario.

This technology allows near real-time consistency of the data.

As we can see, in this case the task of transporting the information updates, storing the information in the correct format, and achieving consistency of the data is relegated to the integration layer that functions totally separated from the application.

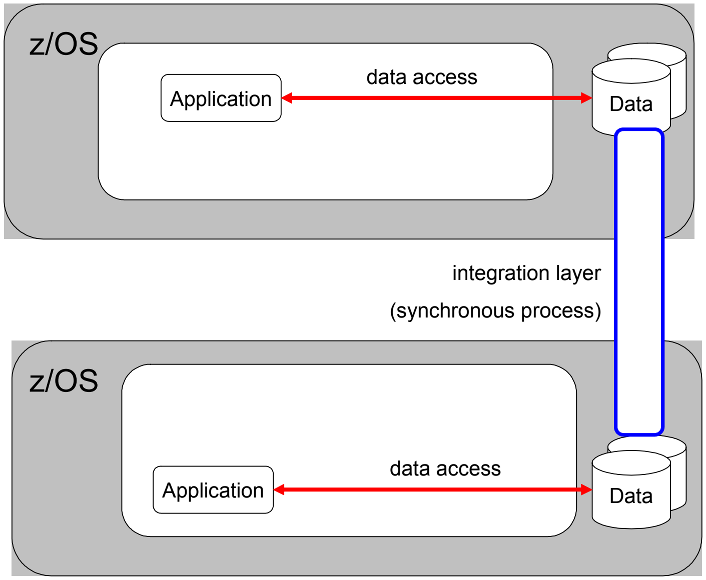


Figure 3-16 Integration through data replication

Integrator / Data Warehouse

These two variations are used by companies with a heterogeneous data infrastructure that have installed an integration layer that covers access “transparently” to this data (besides doing aggregation, correlation, cleansing, federation and other activities).

The functionality depth of the solution distinguishes between Integrator and Data Warehouse. Figure 3-17 on page 84 and Figure 3-18 on page 84 show these scenarios.

In both scenarios, all activities that are necessry for maintaining the consistency level of the information are relegated to the integration layer, and are completely transparent for the application.

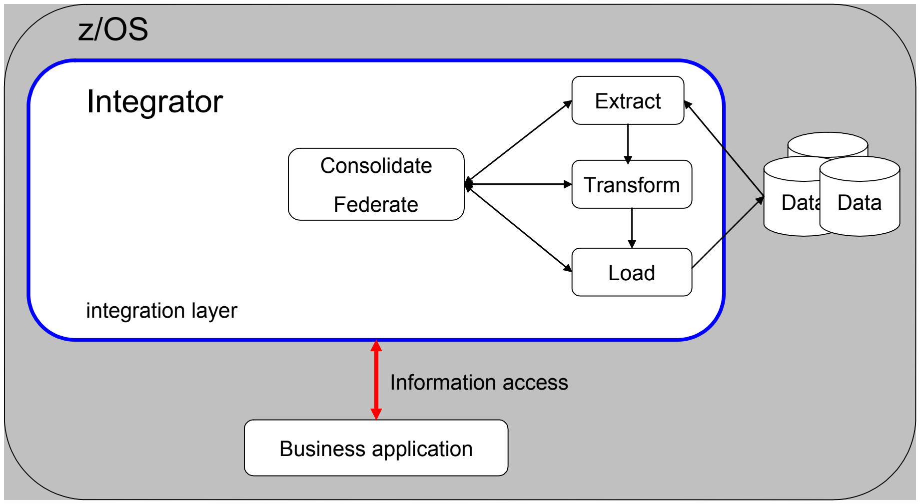


Figure 3-17 Integrator scenario

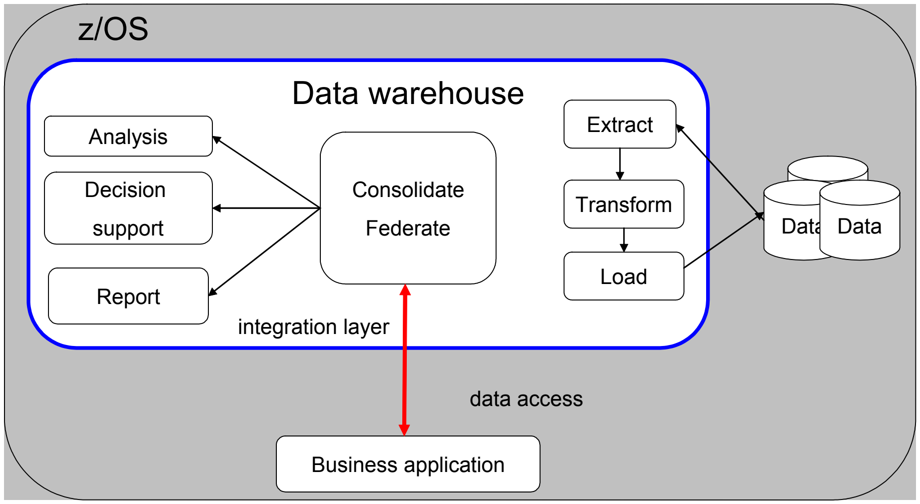


Figure 3-18 Data Warehouse

3.4.3 Characteristics

In the following sections we describe the characteristics of the "data access and integration" scenarios, as they relate to the IBM SOA reference architecture. We also indicate possible SOA-enablement patterns.

ESB

In the batch ETL variation the integration layer can be thought as a “mediation combined with transport services” (function of an ESB). It has a mediation character (transforms records from a format into another, eventually enriches the records with additional information) and a transport character (uses one of the existing file transfer products on z/OS). It is tightly coupled with the application, because the “jobs” doing the ETL function have to know everything about the application they are sending records to (they have to know things like destinations, record formats, etc).

In the messaging ETL variation, the integration layer can be thought as just a “transport service” (function of an ESB). The mediation and transformation function is done inside the application, and therefore even more tightly coupled that in the batch ETL variation.

In the data replication scenario the integration layer can be thought as a ‘technology tool” allowing a consistent view of the data. This means that from a SOA point of view we see no SOA-enablement here.

The “Data Warehouse” and “Integrator” scenarios can be seen as information tools that allow “information” to be presented in a consistent way, hiding all operations taking place on the underlying datastores. Both implement transformation (ESB function), federation (Information services function), and the data access layer. The data warehouse implements much more business logic (specialized queries, multi-dimensional analysis). We see here potential for the implementation of SOA services.

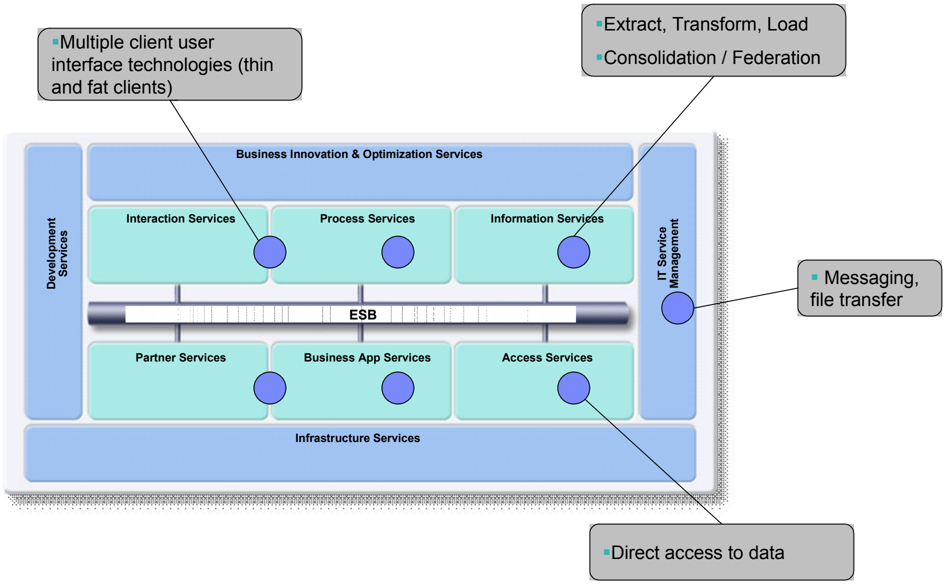


Figure 3-19 Data integration scenarios related to the IBM SOA reference architecture

Partner Services

Some of the variations presented occur also in the B2B scenarios, where applications belonging to one business interact with applications of the partners. ETL batch and messaging (where the messages respect an industry standard) are common occurences in B2B patterns. Therefore these scenarios can also be seen as belonging to the SOA building block “Partner services”.

Access Services

Direct DB access is the rule (either directly through the application or indirectly through ETL jobs), except in the variation “Data Warehouse” where the information access is separated from the real database.

Infrastructure Services

All solutions work on z/OS and as such inherit the Quality of Services of the platform.

3.4.4 Challenges when moving to an SOA

The Integrator/Data Warehouse scenario can be easily integrated into an SOA, if the products that implement the scenario offer SOA services (through Web services interfaces, WebSPhere MQ interface or otherwise). We will see if this is possible when describing the solution techniques in 5.5, "SOA implementation scenarios - Data access and integration" on page 197.

For the ETL batch variation we might put an ESB in place that implements "file-drop" access points, mediation, routing and transformation services.

For the ETL asynchronous (messaging) variation we might try to separate the "transformation" function from the application, implement the Extract, Transform and Load as services, and let them run as sequenced mediations inside of an ESB.

Another possible solution is to encapsulate the ETL batch with a SOA-enabled wrapper and treat it as a "service batch process" (refer to 5.4, "SOA implementation scenarios for batch" on page 183). This might make sense if the encapsulated ETL batch can be designed as a reusable service.

For the scenarios that are relevant to B2B connections we can explore the use of the WebSphere Partner Gateway[2].

These ideas will be discussed in more detail in chapter 5.5, "SOA implementation scenarios - Data access and integration" on page 197.

3.5 Starting scenario - Homegrown SOA

Implementing a service-based approach is not something new. Some companies are doing this already for years. Achieving SOA attributes, such as location transparancy, mediation and reusability has been a goal for many IT organizations for a long time. Those situations where a "service" approach has been taken, but not necessarily following the concepts for an SOA as we define them today, is called a *homegrown SOA*.

> **Important:** The word "homegrown" SOA may sound negative, but we have chosen to use this title to be able to make a distinction between service oriented implementations from the past and the ideal SOA implementation as depicted in Chapter 2, "Target SOA architecture on z/OS" on page 7. There are definitly differences and we want to highlight them, so that we can propose recommendations to improve the homegrown SOA to become a much better SOA.

The two principles of the *homegrown* SOA scenario are, firstly, that the scenario is aimed at achieving some of the SOA benefits, such as location transparancy and reusability, and secondly, that the scenario consists of "Roll Your Own"

[2] The WebSphere Partner Gateway is not available on z/OS.

frameworks and abstraction layers. It has evolved during the years, and is not completely based on open standards. It has been primarily developed with the objective to meet the requirements of integrating applications in a more flexible way, but not necessarily using open standards. Many times WebSphere MQ is used as the underlying infrastructure.

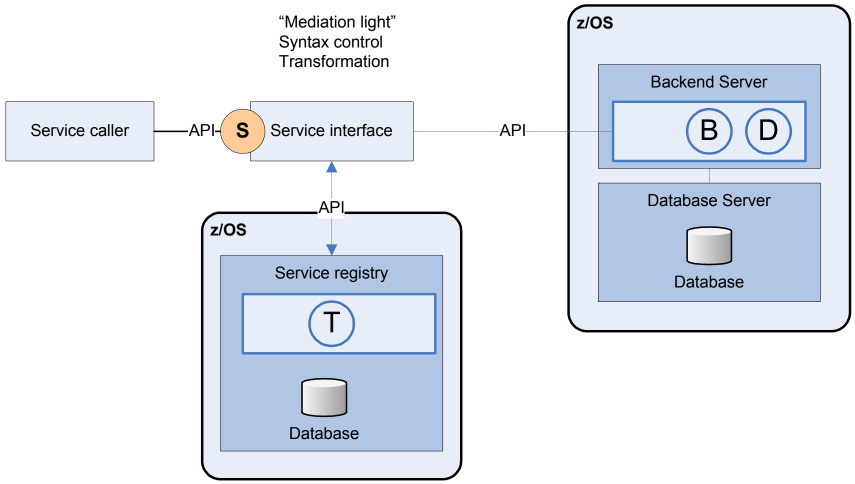


Figure 3-20 Logical view of the homegrown SOA starting scenario

In this scenario, the logic has been divided in service (interface), business and technical functions. As illustrated in figure Figure 3-20, the services are exposed in the "Service interface". Core business functionality is present in the backend transaction server and are accessed through messaging middleware like WMQ or any other type of communication feature or protocol.

Note: The service interface in this concept is an interface accessible through any type of protocol. This protocol may be high-level (MQ) or low-level (TCP/IP) and very open or not. Many homegrown SOAs from the past have not had the objective to use open standards, but just standards that work to get the job done in that specific situation.

In our current SOA definitions we require the service interface to be accessible from any type of commonly used open protocol.

Furthermore, the charactaristics of this scenario are:

- The data resides in DB2 or IMS DL/I databases or in VSAM data sets.
- Integration with external systems is based on WMQ, APPC or possibly TCP/IP sockets.

- Most commonly used languages in the transaction server are Cobol, PL/I, C and even some Assembler. In the application server typically Java is used.
- There is a certain degree of loose coupling, especially if WMQ is used.
- A large degree of encapsulation is implemented and reuse is provided through published interfaces. However, different functions may have been implemented for similar or even identical functions.
- No particular service flow is implemented.
- A departmental or application wide datamodel may exist. It is possible but not likely that an enterprise wide, consolidated datamodel exists.
- Data and function modeling tools are probably used.
- The Quality of Service (QoS) is at a high level.
- The service caller may exist inside an application server on z/OS or a distributed platforms or it may be a client program.

Some attributes of the service interface are:

- It is logically functioning as a “bus”.
- It includes light mediation and transformation.
- Interfaces are proprietary or, in some more modern homegrown SOAs, already based on SOAP.
- It uses dynamic lookup in a “service registry”, which provides location transparency. This service registry should be seen as a fairly simple datastore that contains a “binding” between the logical call from the application and the physical destination of the message or request.
- There is some level of transport protocol independence, e.g. hiding the transport protocol from the application.
- Native transaction support is available in some supported protocols (APPC, J2C connectors, IMS OTMA, CICS EXCI and so on).
- High level of QoS (availability, continuous operations, dynamic workload management and so on) if th entire solution runs on z/OS.

3.5.1 Homegrown SOA variations

The variations existing in this scenario are determined primarily by the extent in which functionality is implemented that mimics SOA functionality, such as an ESB, the usage of open standards, service orchestration, separation of concerns and so on. If all of this done very well and adequate th “homgrown” SOA may be qualify as a real SOA, robust and flexible enough for the future.

So, we may have a variation in which an ESB has been implemented, but just not using open protocols. Routing and protocol conversion may been taken care of in the “ESB”, but transformation may still reside inside application programs. So, there are many graduations thinkable in this scenario.

There may also be homegrown SOA scenarios where everything is still based on traditional technology (CICS or IMS), but where a “service” concept implemented has already been implemented.

3.5.2 Homegrown SOA characteristics

The architecture in the homegrown SOA starting scenario assumes that there has been some degree of standardization and structuring around business services and communication protocols.

The infrastructure components are loosely coupled. Usually, there are different technologies and standards involved. There are also most probably still multiple implementations of similar core functions.

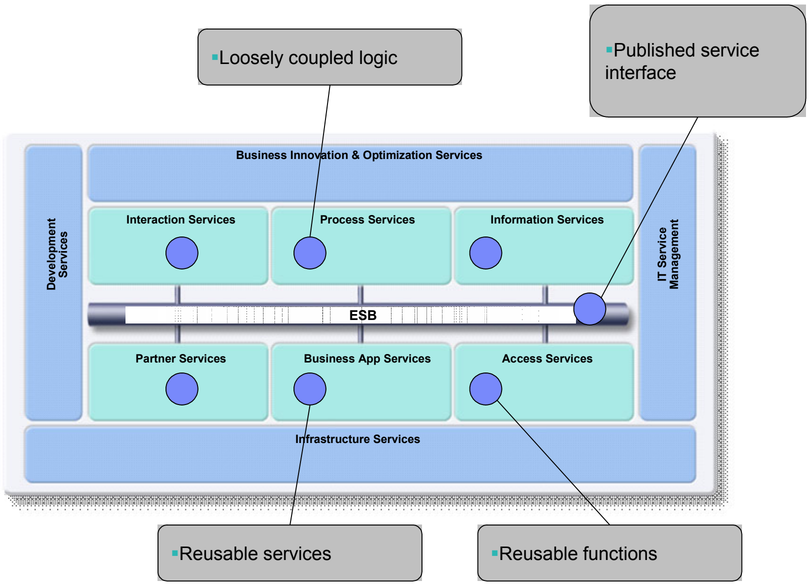


Figure 3-21 Homegrown SOA scenario fit into the IBM SOA reference architecture

Some characteristics of the homegrown SOA scenario, in the context of the IBM SOA reference architecture are mentioned in the following sections.

Interaction Services

In the homegrown SOA scenario the interaction (user interface) has usually not been designed as a set of reusable service components with open standards and connected to an ESB. Instead, any type of user interface technology is used for each specific situation. Quite a few homegrown SOAs have adopted the “portal” concept, but are still not using a service concept in the (user) interaction layer.

The pre-dominant communication protocol used between the user interface device and the interaction layer is HTTP and between the interaction layer and the backend business logic and data access logic could be anything depending on where the interaction layer and the backend components are implemented. If both the interaction layer and business/data access logic are implemented in a J2EE application server, the communication will be J2EE as well.

Process Services

Services can be used as atomic services/components. There may be some level of service choreography implemented, using frameworks or tools. The concept of a “process engine” is fairly new and may be found in some recent homegrown SOAs.

Information Services

Many homegrown SOAs have a certain abstraction in accessing data. A common data access layer is quite common and application programs do not all go directly to data, but do this through this data access layer by means of abstract calls. However, accessing data as a service is less common and may only be found in more recent homegrown SOAs.

ESB

The ESB functionality provided in the homegrown SOA is usually related to the area of transport services. Homegrown SOAs, in most cases, have their mediation, if any, hand-coded inside the application layer or inside the “bus” (which is hand-coded as well in most cases).

- Protocol independence is implemented to a certain degree or not at all and and not necessarily based on open standards.
- There is a standardized middleware for accessing the backend systems such as CICS and IMS.
- Routing is provided by the bus, which is handcoded in many cases.
- Mediation, including transformation, as we define it in the IBM SOA reference architecture is done in the application layer or in the bus, if required.
- Event handling infrastructure is most probably not used. There might be an implementation using pub/sub with MQ.

Partner Services

There may be call-outs to partners, but they are usually not SOA enabled themselves.

Business Application Services

Adapters or connectors for core business applications are provided by the homegrown SOA, sometimes using own code and sometimes using products from vendors.

Access Services

Access Services are heavily used for accessing the backend core functions. There is support for:

- WebSphere MQ
- CICS
- IMS
- Proprietary middleware may be implemented

The backend core functions need support to access data in:

- IMS DB2
- DL/I
- VSAM

Infrastructure Services

The infrastructure in this scenario is mostly based on z/OS and the qualities that this environment provides.

Development Services

There is no specific tooling used for a homegrown SOA. Homegrown SOAs are usually based on J2EE and the usage of messaging (WMQ).

On the technical level, support is needed to modify the backend transactions so they can provide standardized access through the chosen middleware interface.

Some kind of a dictionary is required to keep definitions and requirements of the services provided by the transaction server.

IT Services Management Services

As described in “Infrastructure Services” on page 61, z/OS provides the functionality for security, automation, provisioning etc.

3.5.3 Challenges when moving to an SOA

In the homegrown SOA scenario there has obviously been some level of decomposition of the presentation and business logic. Also, homegrown SOAs ahve been implemented with loose coupling and reuse in mind. But usually, there are still overlapping functions deployed on different platforms.

Also, homegrown SOAs are not per definition based on open standards, which makes it sometimes hard to open up the services for other consumers.

Key questions to consider when migrating to an SOA according to the IBM SOA reference architecture are:

- How to keep the QoS at a high level.
- How to implement open standards.
- How to abstract mediation (especially transformation) out of the application layer or the self-written bus.
- How to implement the interaction layer as services.

In 5.6, “SOA implementation scenarios for homegrown SOA” on page 213 we will take a look at different approaches to improve a homegrown SOA to achieve more of the SOA benefits.

4

The SOA transition process

Now that the “starting scenarios” have been identified and described, it is time to examine the process of transitioning from existing applications and transactions to services, ready to participate in a full SOA.

The topics examined in this chapter are:

- methodologies to analyze the existing business and application environment, discussed in 4.1, “Methodologies for analyzing the business and application environment” on page 96
- tools used to inventory and examine existing applications for eventual reuse and transformation, discussed in 4.2, “Tools to assist in the SOA transformation process” on page 108
- a “pattern-driven” approach to moving from “core” applications to SOA-centric services, discussed in 4.3, “A pattern-driven approach to transition from “core” applications to services” on page 118

4.1 Methodologies for analyzing the business and application environment

In this section we will list different methodologies, all with the purpose to assist in developing an SOA.

4.1.1 Service-Oriented Modeling and Architecture (SOMA)

A key recommendation for development of an SOA is to adopt and adapt a solution development methodology to identify, design and build SOA components. The *Service-Oriented Modeling and Architecture (SOMA),* describes a set of product and technology agnostic modeling, analysis and design activities and techniques to build a SOA.

The SOMA process is comprised of three major steps,as depicted in Figure 4-1:

- Identification
- Specification
- Realization

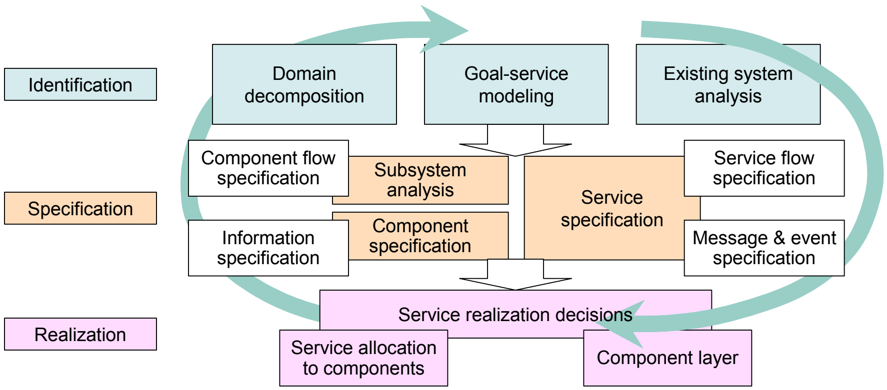


Figure 4-1 Service-Oriented Modeling and Architecture (SOMA)

Through its methods, activities and techniques, SOMA addresses a number of antipatterns that have been encountered by practitioners on engagements involving identification and design of services. Activities and techniques within each of the three steps provides guidelines and solutions to avoid occurrences of these antipatterns. As we review each of the SOMA steps, we identify and describe the associated antipatterns.

Process and methodology: RUP, SOA, and SOMA

Rational Unified Process® (RUP®) is based on software engineering best practices, offers a configurable framework and is scalable to support enterprise initiatives. Therefore, all aspects of RUP can also be applied to the development of an SOA. RUP provides a systematic approach to bridge the gap between business and IT to support a major area of concern, identification of services and how business processes are realized through execution of services.

RUP also provides support for both the bottom-up and top-down approaches by acknowledgement of existing design elements and through activities such as architectural analysis to identify architectural elements such as services. Figure 4-2 illustrates the position of SOMA within the RUP life cycle.

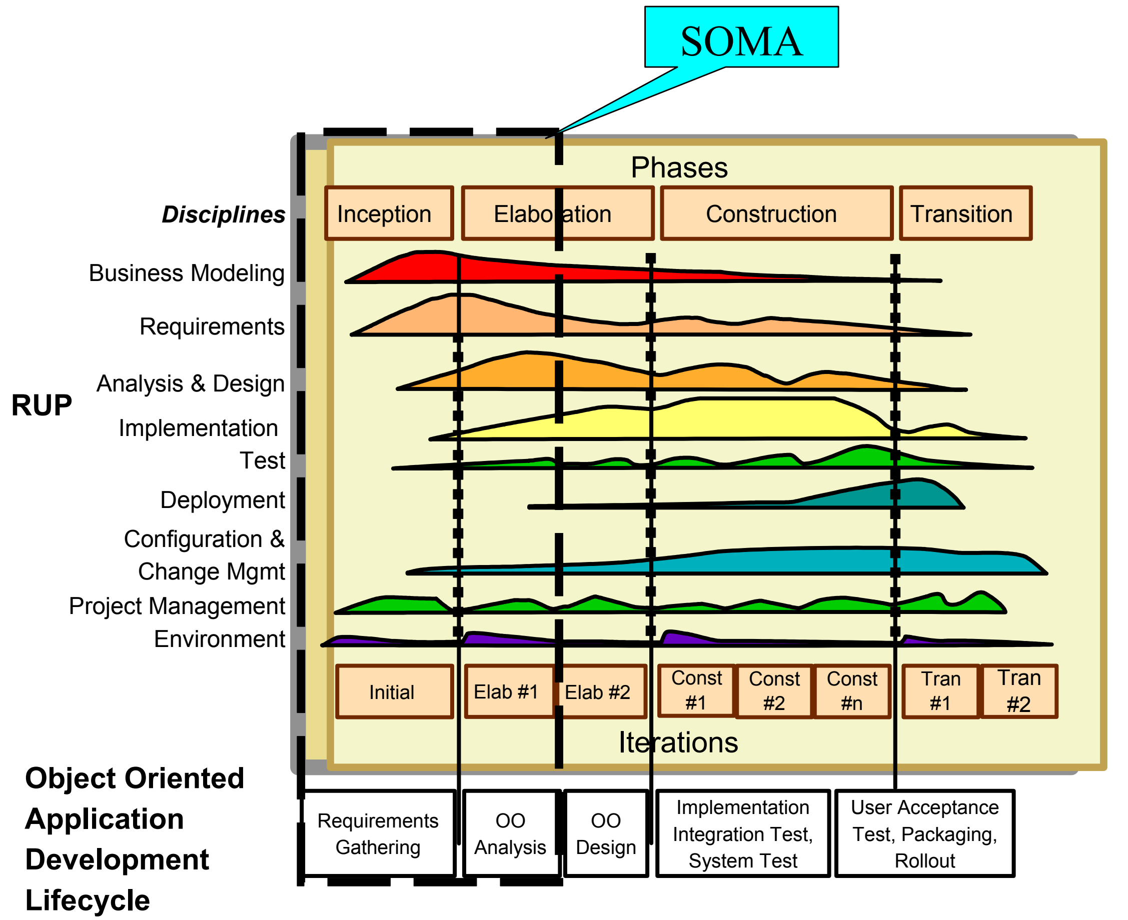


Figure 4-2 Position of SOMA within RUP life cycle

Yet, there is some work in progress to provide additional support within RUP such as incorporating content for describing SOMA techniques and artifacts. Some extensions are required such as introduction of the service model as an artifact and some supporting activities.

Service identification

Service identification begins by applying three complimentary techniques including domain decomposition, existing asset analysis and goal to service modeling to identify candidate services, candidate enterprise components, and flows.

The most important outcome of these activities is the *service model*. The service model is comprised of candidate services that, ultimately, support business services, processes and goals of the enterprise.

A key aspect of the identification step is that it employs a *meet-in-the-middle* approach including a combination of top-down, bottom-up and middle-out analysis techniques. In many cases a pure bottom-up approach is taken. However, this approach typically leads to poor definition of services that are driven mainly by architecture of legacy application interfaces and not from a business perspective.

Domain decomposition represents the top-down approach where business domains are decomposed into functional areas across the value net. Through this technique we can establish the scope of the effort. After domains have been decomposed into functional areas, each area can then be further decomposed into processes and sub-processes and high-level business use cases. Experience shows that the business use cases are considered good candidates for exposure.

Existing asset analysis represents the bottom-up approach where we analyze and leverage APIs, transactions and models from legacy and packaged applications as possible candidate services.

Goal-service modeling provides a middle-out approach that relates services to goals, sub goals, KPIs and metrics of the enterprise. This technique provides a certain level of validation in the form of a completeness check in that it may reveal candidate services that were not identified through the top-down and bottom-up activities.

Finally, *subsystem analysis* expands on subsystems identified during domain decomposition and specifies interdependencies and flows between them.

Antipatterns

Techniques described within this step provide some best practices that can be applied to avoid some very common antipatterns related to identification of services. We summarize two key antipatterns:

- Service Proliferation Syndrome

  First, there is a strong tendency to equate Web services with SOA where, in reality, Web services is an entry point towards SOA adoption. It is possible to create an SOA that does not use Web services and it is also possible to use Web services in a way that cannot really be considered service-oriented. Consequences of this viewpoint manifest themselves into a proliferation of services, commonly referred to as the *Service Proliferation Syndrome,* often resulting in inappropriate exposure of services that are not business aligned.

  Determining the most appropriate level of service granularity is one of the most challenging aspects of service-oriented modeling. The rule of thumb is to model as coarse grained as possible. While fine grained services are also possible, ultimately, the challenge is to find the balance between coarse and fine grained services that meet the business needs. Techniques described here help to determine the appropriate granularity of services and minimize proliferation of services that are too fine-grained.

- Silo approach

  The second antipattern involves what is commonly referred to as the *silo approach* where services are identified based on isolated applications rather than a applying more holistic, enterprise focus. In addition to the recommendation mentioned earlier in this chapter to establish a governance framework that ensures cross-tower service identification and communications, the meet-in-the-middle approach emphasized within SOMA promotes exploration and identification of common services and helps to shift thinking towards modeling of services at the enterprise level.

Service specification

In this step, we identify and specify components that will be required to realize services. Some major activities might include:

- identification of rules, services, attributes, dependencies and variation points for each component.
- exploration of non-functional requirements required by consumers of the services
- specification of messaging, event specifications and management definition
- state management decisions

One of the most important activities within this step is to make crucial decisions to determine which services should be exposed. Some high-level questions related to service design analysis and principles can provide guidance in making these determinations, yet there are many more aspects to take into consideration:

- traceable
  - Can the service be traced back to goals and objectives of the organization?
- stateless
  - Does the service require information or state between requests?
- discoverable
  - Can the service be exposed externally to the enterprise?
  - Does the service have a well defined interface and externalized service description?
- reusable
  - Does the service serve the interest of other processes?
  - Can this service be reused to realize many higher-level business processes?

Service allocation

In this step, we assign services to subsystems identified during previous activities. In addition, we assign services and components that realize them to layers in the SOA as depicted in Figure 4-3 on page 101. A key activity within this step is documenting architectural decisions that relate to both application and technical operational architecture designed and used to support the SOA realization at runtime.

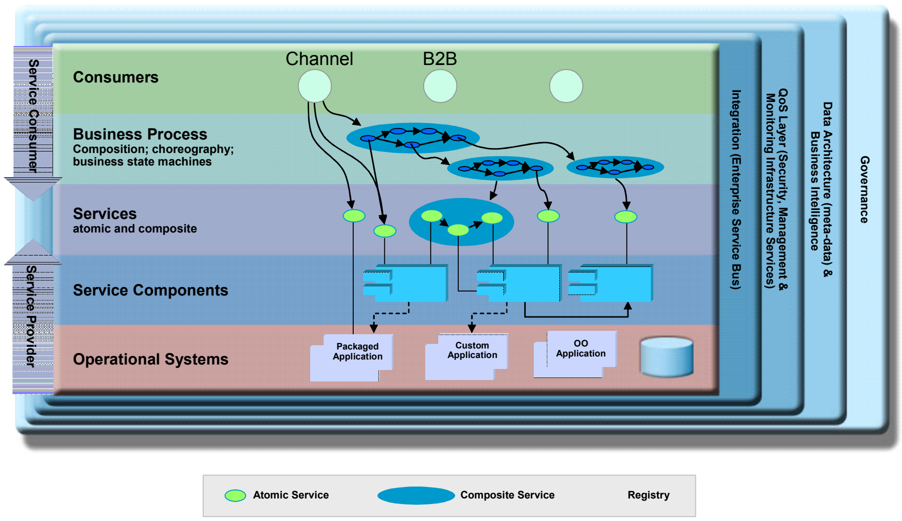


Figure 4-3 Layers of SOA

Through these techniques, we ensure that we find a place for all services and that they can be traced back to business goals and components.

Antipattern: componentless services

There is a tendency to jump directly into development and implementation of Web services without clear association with owning components. This antipattern typically occurs in projects where there is a lack of architectural discipline. Architecture patterns are neither considered nor applied.

Once again, lack of good service modeling and design techniques ultimately result in serious violations of basic principles such as modular design, information hiding, encapsulation and layering. These issues can lead to significant cost and effort related to potential reengineering.

In addition to leveraging J2EE and general EAD best practices, application of SOMA techniques such service allocation will help in avoiding occurrences of this antipattern.

Service realization

Realization is a key architectural step that involves the exploration of alternatives to realize the implementation of services. In addition, we identify and assess technical constraints to ensure technical feasibility of realization of services specifically for those identified during existing asset analysis

Alternatives for implementation of services goes beyond buy versus build. Some realization alternatives are:

- building new component functionality (custom build).
- Transforming legacy to enable reuse of functionality exposed as services.
- Integrating by wrapping existing systems.
- Buying and integrating with third-party products.
- Subscribing and outsourcing parts of the functionality, especially with Web interfaces. Subscription assumes that an enterprise application integration model has been implemented and there are services to subscribe to, in a hub and spokes architecture.

Transformation considerations

A service's implementation can be realized by wrapping an existing system with a message queue service or a Web service. However, in many cases, mere exposure of existing functionality is not sufficient. Componentization of the existing system or a small subset of the system must take place to properly expose the functionality required. A key factor is the scope of the componentization. Avoid a common tendency to break the entire existing system down into parts. Select an appropriate subset and transform it through componentization.

Antipattern: chatty services

Similar to the identification and specification steps, there are antipatterns related to realization of a SOA. Though all three steps prescribed within SOMA provide techniques to avoid occurrence of these antipatterns, the consequences of this particular pattern are more related to realization.

In many cases, developers are asked to begin direct replacement of existing APIs with Web services without much thought to the benefits of SOA and conformance to SOA design and architecture principles. This antipattern, referred to as chatty services stemming from a chatty dialog communicating small pieces of information, may result in severe degradation in performance. In addition, significant development costs may be incurred to aggregate the fine-grained services into a service model that can actually realize benefits of SOA. Guidance and techniques to avoid occurrence of this antipattern really spans all three steps of SOMA.

- At the identification step, the meet-in-the-middle approach can help to ensure completeness of the model, attain an appropriate level of service granularity and ensure that there is traceability back from the service to business objectives and goals of the enterprise.

- At the specification step, application of a litmus test considering key service design principles can help to make the appropriate decisions involving service exposure.
- At the realization step, we assess the technical feasibility of services identified through existing asset analysis to further validation the decision of services to be exposed.

Summary

In summary, experience shows that service-oriented modeling, analysis, and design is necessary to build an SOA. It is important to reiterate some of the key aspects and benefits of SOMA:

- Techniques are designed to enable target business processes through identification, specification and realization of business aligned services forming the foundation of a SOA
- Builds SOAs that align clients' business goals and directly ties business processes to underlying applications through services, helping businesses realize benefits more rapidly.
- Approach helps to ensure that goals set by business process modeling can actually be implemented to generate the greatest result in an efficient manner.

Application of these best practices, guidelines and techniques in identification of services and solutions can help to realize the many benefits offered through SOA.

For additional information about the identification of services and solutions, refer to the following sources:

- Redbook: *Patterns: SOA Foundation Service Creation Scenario,* SG24-7240*:*

  `http://www.redbooks.ibm.com/abstracts/sg247240.html`

- *Service Oriented Modeling and Architecture*:

  `http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/`

- *Elements of Service-Oriented Analysis and Design*:

  `http://www.ibm.com/developerworks/webservices/library/ws-soad1/`

- *SOA realization: Service design principles*:

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

- IBM SOMA and/or IBM SOA:

  `http://www.ibm.com/services/soa`

4.1.2 Service Integration Maturity Model (SIMM)

Service Integration Maturity Model (SIMM) was originally created as a way to judge the maturity (or readiness) of an organization and their business environment to move toward a SOA. The SIMM engagement or workshop is conducted at a fairly high level with facilitators as well as business and technical Subject Matter Experts (SME) from the organization being reviewed.

A SIMM engagement may involve many steps, but may be reduced to a set of key steps, e.g. to identify key pain points, and therefore places where the most return for minimal work may be obtained. It is also a good way for groups to get started with SOA by helping to identify a project that may be low hanging fruit.

A pain point is a way of identification of problem areas in the business unit which prevent alignment with business goals. Pain points come in many forms, including business process problems, legacy application problems, staffing, funding, capacity, or skills. In addition, pain points may have emerged from rapid changes in the business environment. SIMM uses pain points to start to identify opportunities to improve the business unit processes and applications while aligning with an SOA+ approach.

SIMM helps define a roadmap for incremental IT transformation linked to business transformation.

Service Integration maturity is assessed using different views, as illustrated in Figure 4-4 on page 105.

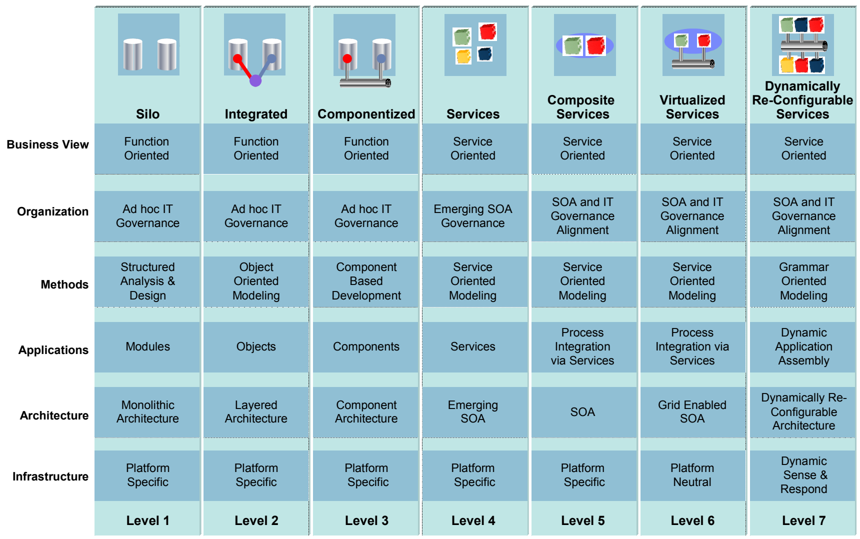


Figure 4-4 The Service Integration Maturity Model (SIMM)

Views assessed in SIMM are:

- Business

  How well does the business understand, design, implement and execute its business processes?

- Organization

  How effective is the Business/IT organization?

- Methods

  How well are business goals understood by IT?

- Application

  How well can the IT team perform its mission?

- Architecture

  How advanced is IT thinking?

- Information

  How the core data operations are performed, transformed and the data is managed within the enterprise?

- Infrastructure

  How capable is the IT plant?

Working the way through the model, current maturity and future to-be state is assessed.

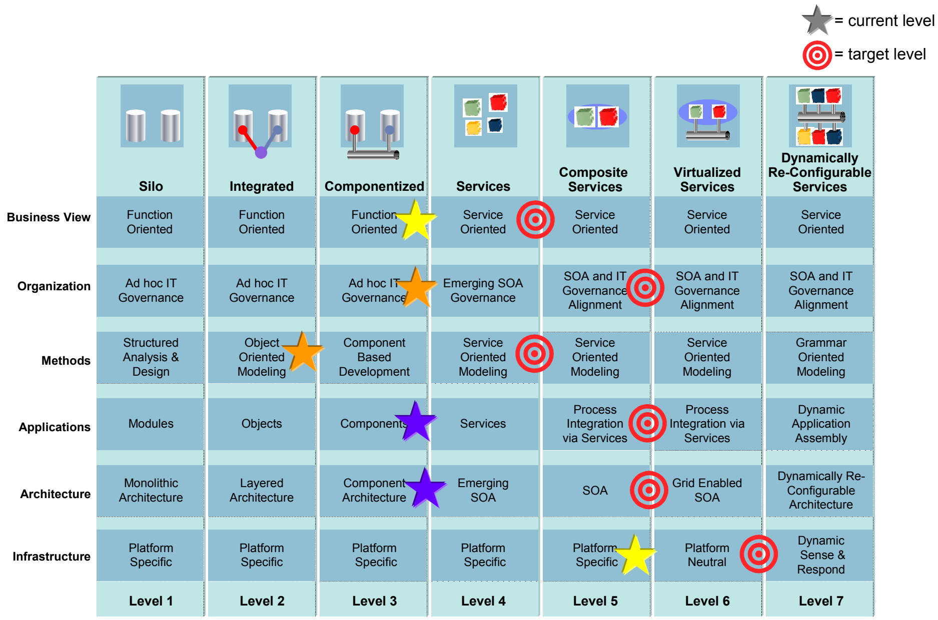


Figure 4-5 Service Integration Maturity Model Assessment

Using the gap between to-be and current state, action items are developed. This is illustrated in Figure 4-6 on page 107.

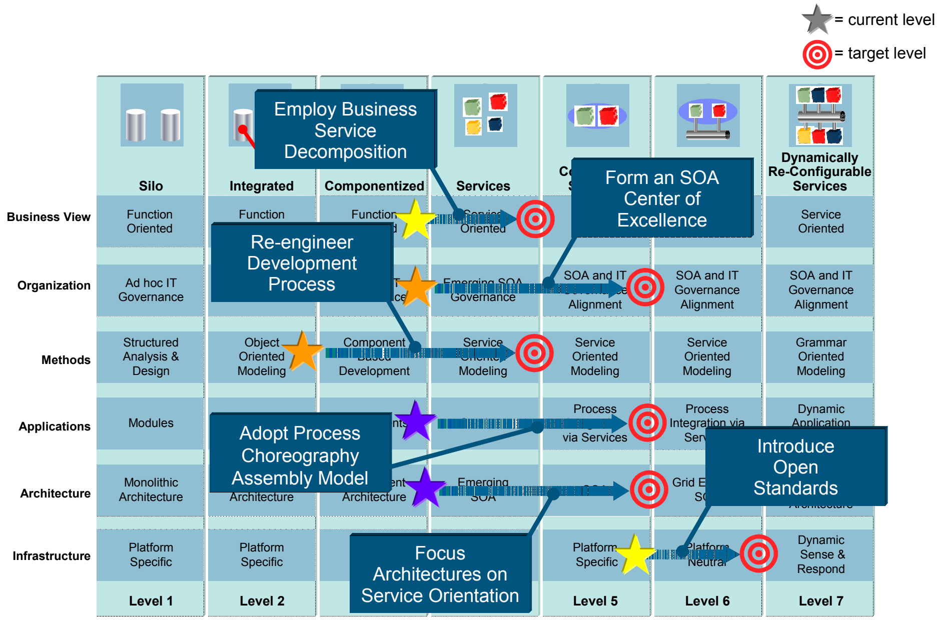


Figure 4-6 Service Integration Maturity Model with illustrating actions

4.1.3 SOA Readiness Assessment

The *SOA Readiness Assessment* is a questionnaire tool based on the SOA maturity model. SOA capabilities are assessed, in the area where a company stands when it comes to adopting SOA.

With targeted recommendations for improving the maturity level, the IBM SOA assessment tool can help unlock the full value of SOA.

For additional information

Refer to the following sources for more information:

- IBM SOA homepage:

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

- SOA Readiness Assessment:

  `http://www.ibm.com/software/solutions/soa/soaassessment/index.html`

4.2 Tools to assist in the SOA transformation process

Many efforts to move from the core system “starting scenario” to a target scenario involve some amount of analysis of existing source code and possibly extraction of business logic and/or rewrite of the existing code to create SOA-compliant services.

IBM has defined the “SOA Lifecycle” as four stages in a closed loop process: *Model, Assemble, Deploy***,** and *Manage* (see 2.3.4, “IBM SOA lifecycle” on page 15 for more details on the SOA Lifecycle). The SOA Lifecycle defines most of the key tasks necessary to create new SOA-compliant services AND to service-enable existing core applications.

The process of core system modernization (also known as “enterprise transformation” or “legacy modernization”) falls primarily in the *Assemble* stage of the SOA Lifecycle. That process involves a number of key steps:

- discovery of existing code modules and other assets (JCL, system definition files, etc.)
- discovery of interrelationships between artifacts
- deep investigation of existing code and data structures to identify service definitions and business rules
- refactoring of code structure to “clean up” source code for efficiency, performance, reuse and readability purposes
- programming work to make modifications to discovered or refactored source code
- version control and storage of application artifacts before, during, and after the re-engineering process

These application development-specific activities within the SOA Lifecycle fall within a Business-Driven Development (BDD) process, as illustrated in Figure 4-7 on page 109.

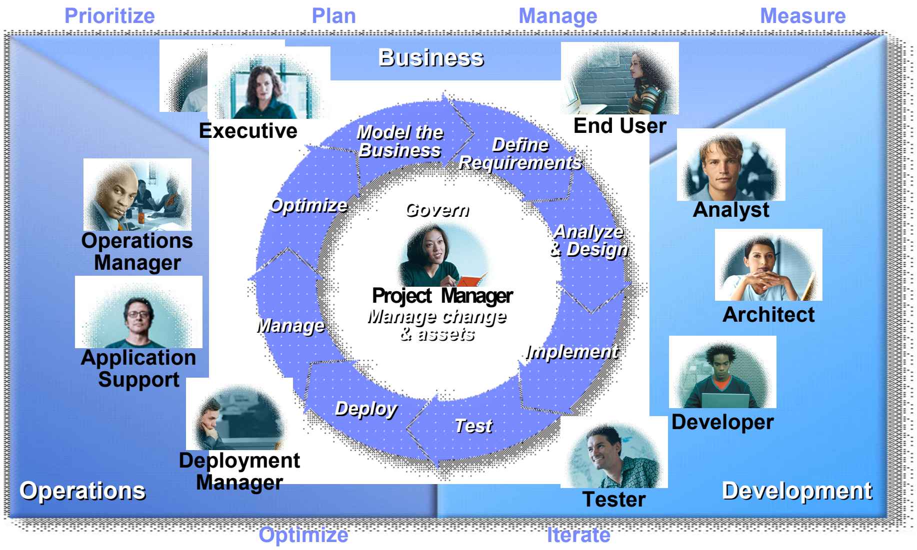


Figure 4-7 The Business Driven Development process

In this section we are concentrating on the development/re-engineering aspects of the BDD process (lower right side of Figure 4-7). Later in the book we will discuss what is needed for the Deploy and Manage stages of the SOA Lifecycle and the associated aspects of this Business-Driven Development process.

4.2.1 Tools used in the Model stage

Over the last several years, a number of tools have been developed to assist the architect and application developer in modeling, coding and assembling applications. There has been much attention paid to object-oriented design and modeling. Tools that work with the Unified Modeling Language (UML)[1] have been used for OO-based application analysis and design, data modeling, defining object relationships, and describing the overall structure of the application.

While UML is still in use for these purposes, other more coarse grained, business focused modeling tools are now in use for business modeling in SOA. *WebSphere Business Modeler* is IBM's key product for creating models of business processes and coarse grained composite transactions.

And, *WebSphere Integration Developer (WID)* assists in transforming the business processes defined using the WebSphere Business Modeler into a *Business Process Execution Language (BPEL)* representation. This

1 http://www.uml.org/

BPEL-defined process can later be run and managed using a run-time engine such as the *WebSphere Process Server (WPS).*

Since this chapter concentrates on the underpinnings for core system service enablement, we will not examine those tools in detail here. Tools such as the WebSphere Business Modeler are used primarily in "top-down" application designs, where the process or composite application are defined first, and this design drives the underlying service granularity. In this chapter, we are looking at service enablement from the "bottom-up," where existing applications are migrated to an SOA-compliant form using several transformation techniques, discussed in 4.3.3, "Transition approaches" on page 122.

In addition to modeling processes and applications, the developer and architect must also manage requirements. IBM's *Rational RequisitePro®* tool can be used for requirements gathering and management, to feed the process of re-engineering existing code assets or for construction of new services.

4.2.2 Discovery and refactoring tools used in the Assemble stage

From an application modernization and transformation perspective, the artifacts that are produced from the Model stage of the SOA Lifecycle are primarily process models and service definitions that come from the modeling tools such as WebSphere Business Modeler, WebSphere Integration Developer (WID), and *Rational Software Architect (RSA)*. In "Service identification" on page 98, the concept of "bottom-up" versus "top-down" versus "meet-in-the-middle" development was discussed.

The Assemble stage is where the "meet-in-the-middle" occurs. Models created during the Model stage are combined with the services created during the Assemble stage to form composite applications. But in order to create those services, much discovery and analysis of existing assets must take place to harvest source code and business rules and to maximize reuse of those existing assets.

WebSphere Studio Asset Analyzer

WebSphere Studio Asset Analyzer (WSAA) is a static analysis tool that provides a developer or an architect with a high-level view of the application inventory, and optionally allows them to drill down into the application to examine its stucture and determine component inter-relationships. Dependencies between the application artifacts can be observed, and those dependencies can be used to determine the level of effort needed to make appropriate modifications or rewrites to the application for SOA enablement.

WSAA is referred to as a "static analysis" tool because it only analyzes what the administrator tells it to - there is no awareness of running applications. If a partitioned data set filled with source code is fed into WSAA, it will analyze it, even if that code has not actually executed in decades. Dynamic tools, such as the *CICS Interdepedency Analyzer*, collect data on running systems and provide information on components actually in use. However, this information is not as complete as that provided by WSAA (since source and other related artifacts are not available at run-time). In addition, dynamic tools may not catch information on programs that run only on occasion, such as monthly or year-end jobs.

The high-level architecture of WSAA is shown in Figure 4-8.

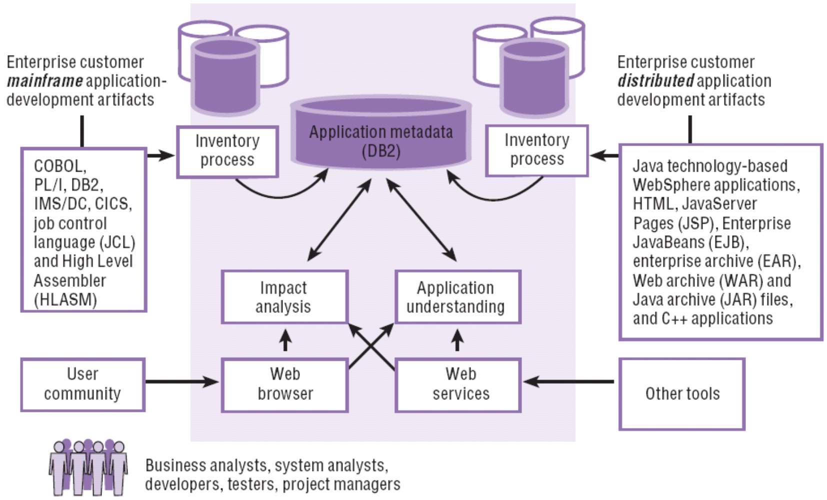


Figure 4-8 WebSphere Studio Asset Analyzer (WSAA) overview

Note that WSAA is not restricted to host assets - it can also examine and report on artifacts from Java, C++, and other Web and distributed applications. The artifacts are fed to the WSAA inventory process engine, WSAA adds the artifacts to the DB2-based repository, and then the user, either via a Web or Web services (used by other non-browser tools) interface, can perform inquiries and analysis on those artifacts added to the repository.

The results of a typical WSAA query are shown in Figure 4-9 on page 112. Here, a batch job is decomposed, showing the job steps and the DD names (files) used by the job. If possible, the sub-components are hot-linked so the user can drill down and examine the various parts of the job (programs, files, databases, etc.)

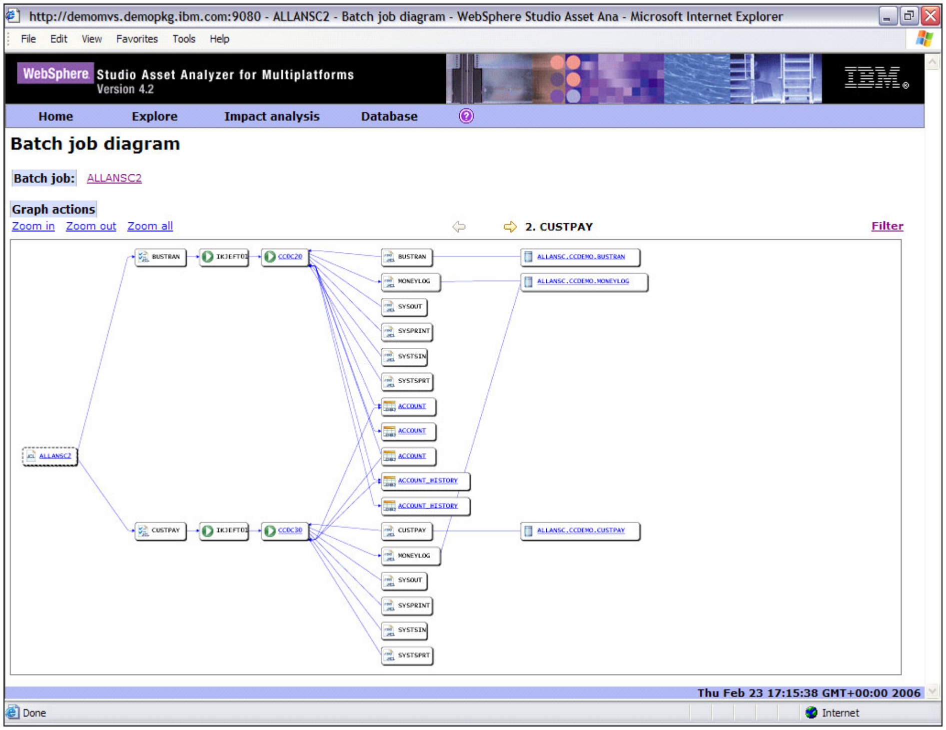


Figure 4-9 Batch job analysis in WebSphere Studio Asset Analyzer

A moderate amount of detail about program content, including the ability to browse source code, is available through WSAA, which is intended to provide a high-level view of the application structure and interdependencies between components. However, detailed analysis and decomposition of program assets is better accomplished in the Asset Transformation Workbench, discussed in the next section..

Asset Transformation Workbench

The l*Asset Transformation Workbench*[2] *(ATW)* provides detailed reports, metrics and visualizations of existing mainframe applications. The foundation of ATW is a knowledge base that contains information that describes the applications. Surrounding the knowledge base are a number of key capabilities and features[3]:

[2] ATW is provided by Relativity Technologies and is resold by IBM under the “Asset Transformation Workbench” name.

[3] Feature descriptions for ATW can be found at http://www-306.ibm.com/software/awdtools/atw/features/

- Detailed reports, metrics, documentation, and visualizations of the enterprise applications are readily accessible to project leaders and architects using the workbench.
- A browser-based module to allow team members to use ATW-generated reports
- Integration with IBM WebSphere Studio Asset Analyzer to allow users to perform high-level analysis in WebSphere Studio Asset Analyzer and pass the application insight through a software bridge for use in Asset Transformation Workbench
- Powerful analysis and assessment tools to help accelerate ongoing maintenance and enhancements
- Tools to expose and help manage business rules, which can simplify application reuse for SOA initiatives
- Re-architecting tools to help increase the productivity of teams restructuring and componentizing applications
- A Reuse Analyzer for ATW (technical preview) to help quickly assess an application's suitability for reuse in a SOA

There are several main components of ATW (see Figure 4-10 on page 115) that use the knowledge base:

- **Application Analyzer**
  The *Application Analyzer* is a non invasive interactive module that creates a comprehensive repository of system relationships including source code, system files, data definition language (DDL), screen maps, and more. It can help perform impact analysis, generate interactive graphical system diagrams, create system documentation, and browse source code in context-sensitive mode.
- **Application Profiler**
  The *Application Profiler* provides technical and business users with information to effectively understand and plan enterprise applications, without impacting the source code. It is a browser-based tool and provides users such as support, quality assurance, and business analysts insight into systems without requiring specialized knowledge or skills. Technical users that are unfamiliar with their enterprise applications can use Application Profiler to access documentation, understand system structure, and determine the impact of code changes. Non-technical or business users can use Application Profiler to examine system-level reports and to assess where to direct resources in order to enhance or renovate the application portfolio. When used with the Business Rules Extension, the Application Profiler module can help organize and annotate business rules without disturbing development.

- **Business Rules Extension**
  The *Business Rules* feature is an optional extension that helps navigate complex code and identify, document, and organize business rules. It identifies candidate rules using developer-driven sophisticated search algorithms. This process generates a list of rules for the targeted application, allowing analysts to view each rule and verify its inclusion. After rules have been found they can be documented and organized, allowing future users to understand the use of each rule. And because the rules are tagged, analysts can locate the rules within the code and modify them to respond to business process changes.
- **Application Architect**
  The *Application Architect* feature is an optional extension that uses sophisticated algorithms to partition code into new components. The componentization of logic results in a structured architecture that can reduce complexity and facilitate modernization. By componentizing enterprise code, developers are able to greatly increase the performance of frequently used programs. Application Architect can help to ensure that the components created are complete, working programs in accordance with the functionality of the original application.
- **Reuse Analyzer**
  The *Reuse Analyzer* extension (in ATW Version 2.1, a “technology preview”) can:
  - Categorize CICS and IMS programs written in COBOL by the type of work they do (screen, business logic, data access, hybrid, etc.).
  - Identify some potential architectural "traps" that would require remediation before making a particular program or program call hierarchy available as a Web service.
  - Create Web Services Description Language (WSDL) files corresponding to selected data elements in your program you wish to make available in a Web service. (A WSDL file can then be used with XML Services for the Enterprise, which is a feature of WebSphere Developer for zSeries.)

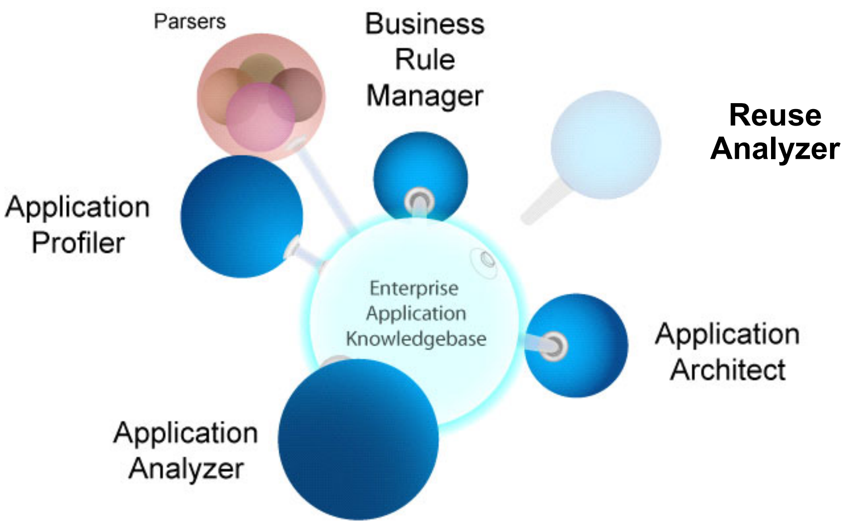


Figure 4-10 Asset Transformation Workbench features

The Asset Transformation Workbench is used when a detailed view of the application is needed for work on re-engineering an application. The WebSphere Studio Asset Analyzer is used when a higher level view of the application assets is needed.

ATW is a participant in the Model stage of the SOA Lifecycle, but it also has some participation in the Assemble stage, as it provides key features, such as the Application Architect and Reuse Analyzer, that assist in the actual code creation.

4.2.3 Code development tools used in the Assemble stage

Once discovery has been conducted and the appropriate code resources have been identified for reuse or redevelopment, business rules have been isolated, and data structures have been determined - this is the “Analyze and design” portion of the Business Driven Development model (see Figure 4-7 on page 109), the next step is the actual service creation, which is referred to as “Implement” in the BDD model.

The Asset Transformation Workbench, discussed in section “Asset Transformation Workbench” on page 112, can be thought of as a partial participant in this stage, as there is some modification of code that can be done with ATW. The Application Architect component permits the developer to refactor the existing code by partitioning it into better structured components. But it is not an Integrated Development Environment (IDE) intended for writing code.

WebSphere Developer for zSeries

IBM's *WebSphere Developer for zSeries (WDz)* is a superset of the Rational Application Developer IDE for Java development. WDz adds the ability to build applications and services using traditional programming languages, including COBOL, PL/I and Assembler.

WDz enables the host developer with the same kind of tools that have traditionally been available only to distributed system developers. It can be used to build new SOA-compliant services and to re-engineer existing host programs to participate in an SOA. It is ideal for equipping traditional host developers with new tooling that will improve their productivity, while equipping them to transition to a Java-based development model, or to equip Java developers with the ability to work with mainframe code assets.

Some key features that WDz provides for the host developer include:

- Local or remote edit, syntax check and compile of COBOL, PL/I and Assembler programs.
- Features including programming language "code assist" and automatic generation of JCL (based on standardized skeletons) to reduce complexity for inexperienced host developers.
- Interactive debugging of traditional language programs using the same kind of debugger that workstation developers use - and with live code running on z/OS.
- *XML Services for the Enterprise (XSE)*, a feature that permits the developer to generate service definitions from existing host applications or to generate code for the mainframe using existing service definitions in WSDL.
- *Enterprise Generation Language (EGL)*, a fourth-generation language (4GL) that greatly simplifies development by enabling the developer to code services using EGL and generate the executables either in COBOL or Java (COBOL generation is unique to WDz).
- Job control and monitoring using the Remote System view to provide functions similar to the SDSF function found under TSO.
- Support for the creation of CICS Web Services (CICS Transaction Server V3.1).
- *CICS Service Flow Modeler* for creation of services and transaction flows that execute under the CICS Transaction Server V3.1 *Service Flow Feature* (see "Variation 2: CICS Service Flow Feature (CICS Transaction Server V3.1)" on page 139 for more details)

Figure 4-11 on page 117 shows what a typical WDz session might look like.

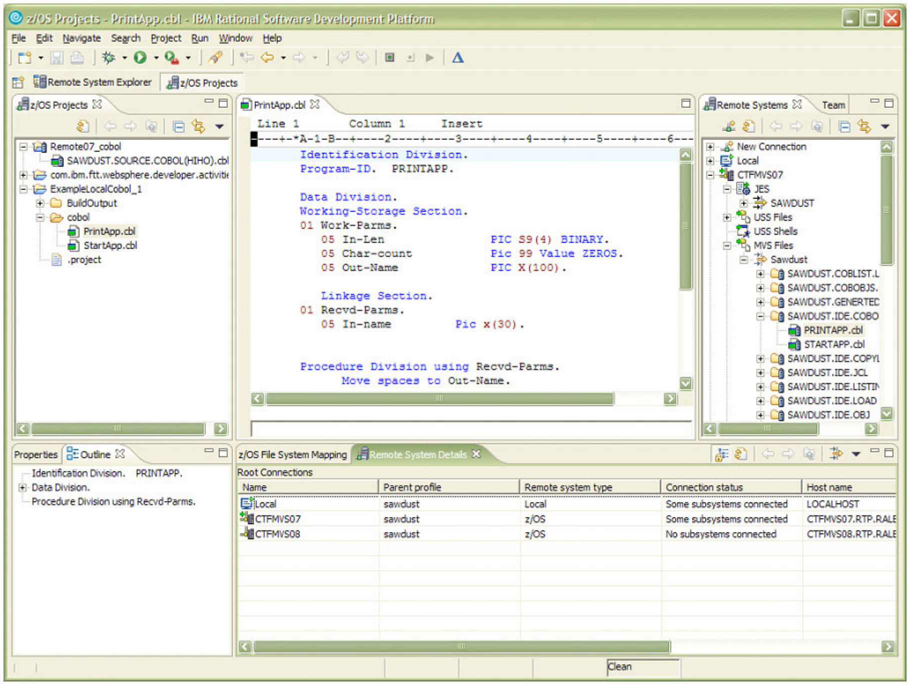


Figure 4-11 WebSphere Developer for zSeries

A WebSphere Developer for zSeries user can concentrate solely on service-based development, or he can use WDz to build a complete, end-to-end composite application. WDz includes the entire Rational Application Developer toolset, so a developer can use frameworks such as Struts to develop a front-end user interface and tie it to host services, written in traditional programming languages, and coded using the z/OS Project or EGL perspectives.

4.2.4 Tools used in the Manage stage

Author Comment: A section needs to be written on Manage tools.....

4.2.5 Interrelationships between tools

The tools detailed in this section all work together within the SOA Lifecycle to model, assemble, deploy and manage SOA-compliant composite applications. Figure 4-12 on page 118 illustrates the interrelationship between the tools mentioned here.

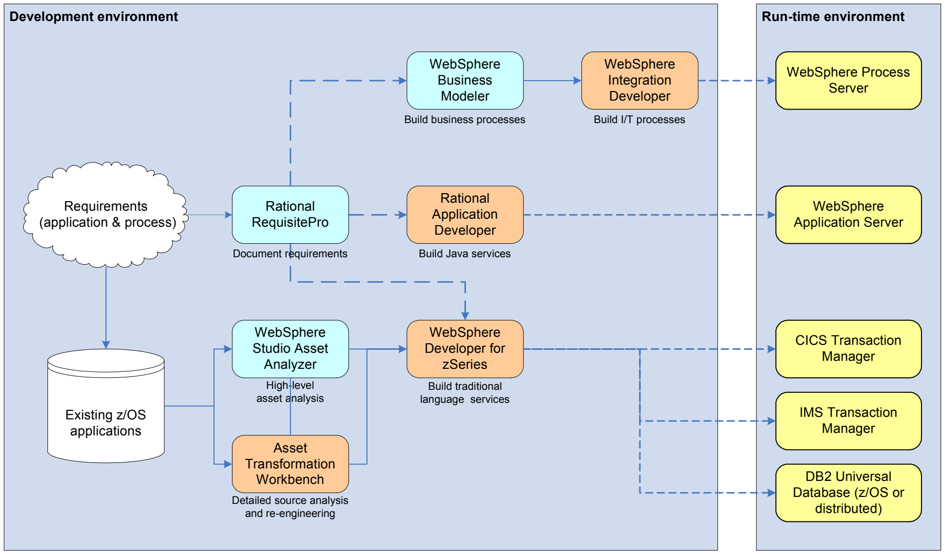


Figure 4-12 An overview of the SOA development tools

4.3 A pattern-driven approach to transition from "core" applications to services

IBM has established a methodology for designing IT systems that is based on "patterns." This methodology, under the umbrella of "Patterns for e-business"[4] analyzes a problem in the following sequence:

1. examine *customer requirements* for the business problem
2. identify a *business pattern* that matches requirements
3. select an *Integration or Composite pattern* that narrows down how the business problem is to be solved
4. select the appropriate *application pattern* that will determine how the application logic is partitioned
5. select a corresponding *runtime pattern* for the IT infrastructure
6. use the *product mapping* for the runtime pattern to construct the specified operational infrastructure

This pattern-driven approach accelerates the analysis and design of a solution by leveraging prior experiences and the similarities between those and the new business problem..

[4] http://www-128.ibm.com/developerworks/patterns/

Similar to the approach used with the Patterns for e-business, it is appropriate to use such a process for examining existing core system assets and designing a solution based upon commonly-used “transition scenarios.” Through our collective experiences and those of other organizations in IBM, such as IBM Global Business Services, we have identified the “starting scenarios” described in Chapter 3, “Starting scenarios” on page 53, the “SOA implementation scenarios” that we examine in Chapter 5, “SOA implementation scenarios” on page 129, and the sequence of events necessary to move from one to the other.

The transition scenarios target different SOA “ambition” levels, or different levels of desired SOA maturity.

The use of techiques to move from “starting scenarios” to “implementation scenarios” is an IT-centric technique. It does not necessarily take into account the functional requirements of the application. It does, however, require consideration of the needs of the business, as there are *Non-Functional Requirements (NFRs)* that must be considered, including performance, scalability, reliability, security, etc. These will drive the selection of particular implementation technologies that fall within the end-point migration scenarios.

4.3.1 Starting scenarios

The starting scenarios have been identified and described in Chapter 3, “Starting scenarios” on page 53. They require no further examination here.

4.3.2 Service interface patterns[5]

When analyzing the various core system transformation projects in the IT world today, we see four basic service interface patterns emerge.

We have identified each of them with a single letter (N, A, R, and B).

Note: in the associated diagrams, P represents presentation logic, B represents business logic, and D represents data access logic.

Native service interface (N)

The *native service interface* solution enables a core system for SOA by utilizing SOA features that are native to the transaction server, database manager or whichever infrastructure component hosts the application component. Frequently, support for protocols such as SOAP, JMS or WebSphere MQ are built-in to those systems. An example of this is the CICS Web Services support included with CICS Transaction Server Version 3.1. In this new feature, existing

[5] The Legacy Transformation Practice of IBM Global Services has documented these four patterns as “Native,” “Adapter,” “Modularized,” and “Brokered”

CICS transactions may be accessed directly in CICS using Web services protocols (SOAP, WS-Security, etc.). The pattern is shown in Figure 4-13.

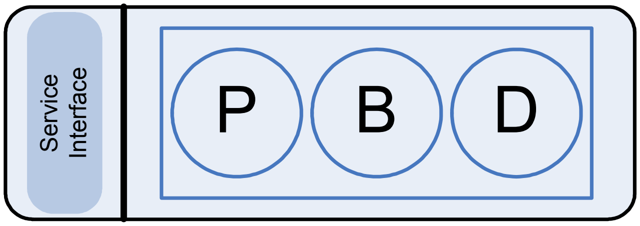


Figure 4-13 The native service interface pattern

Adapter-provided service interface (A)

In many cases, a core system is unable to communicate via service-oriented protocols, as in the "native service interface" pattern. In these situations, an adapter may be employed, if available, to translate between the proprietary interface to the system and SOA-compliant protocols. The Adapter usually "lives" outside the infrastructure component that hosts the application component to be service-enabled. Adapters have been employed in the past to provide integration with middleware such as Web application servers. In this pattern, the adapter provides an interface to SOA-based calling services. This pattern is illustrated in Figure 4-14.

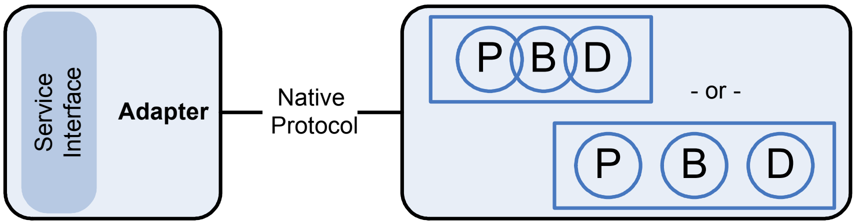


Figure 4-14 The Adapter-provided service interface pattern

Brokered/mediated service interface (B)

It is often easier to employ an intermediary Enterprise Service Bus, or "broker," to provide the service interface to the transitioned core application. The ESB can insulate the core application from the need to comply with new protocols, and it can transform the message content so it can be processed by other services that connect through the ESB.

Often, this brokered/mediated pattern involves the use of WebSphere MQ or JMS as a message transport. But, most message broker and ESB products can support multiple transport protocols, inbound and outbound.

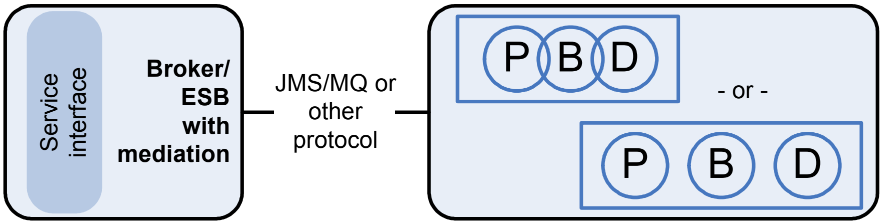


Figure 4-15 The brokered/mediated service interface pattern

Redeveloped **code with native service interface (R)**

This pattern involves taking existing core system code and re-writing it to conform to SOA-compliant structure and protocols. This often involves refactoring the source code so it has a more "SOA-friendly" service structure. The newly-modularized code would then be invokable directly via SOA-compliant protocols, usually SOAP-over-HTTP or SOAP-over-JMS/MQ.

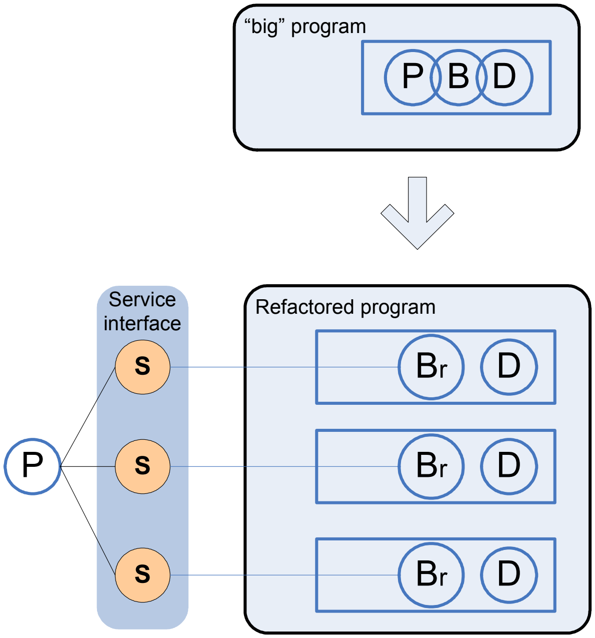


Figure 4-16 Refactoring of "big" program

Figure 4-16, illustrates how an application with presentation, code and data logic highly dependant on each other is refactored into a clear separation of concerns. Presentation logic is completely taken out of from the refactored program, and is here considered to be a service consumer.

The refactoring pattern also covers how to find the right level of granularity on the services that are exposed.

4.3.3 Transition approaches

As the architect examines the existing core system and determines its current implementation, he/she must determine which transition approach to use for service enablement. This will be driven largely by the technique chosen to perform the migration.

The dependency between our service interface patterns and transition approaches (*Improve*, *Adapt* and *Innovate*) is illustrated in Figure 4-17 on page 122. Inside each stage of maturity (improve, adapt, innovate), we can apply different service interface patterns (native, adapter, brokered/mediated and redeveloped as discussed previously). There are different levels of ambitions and cost associated with each step.

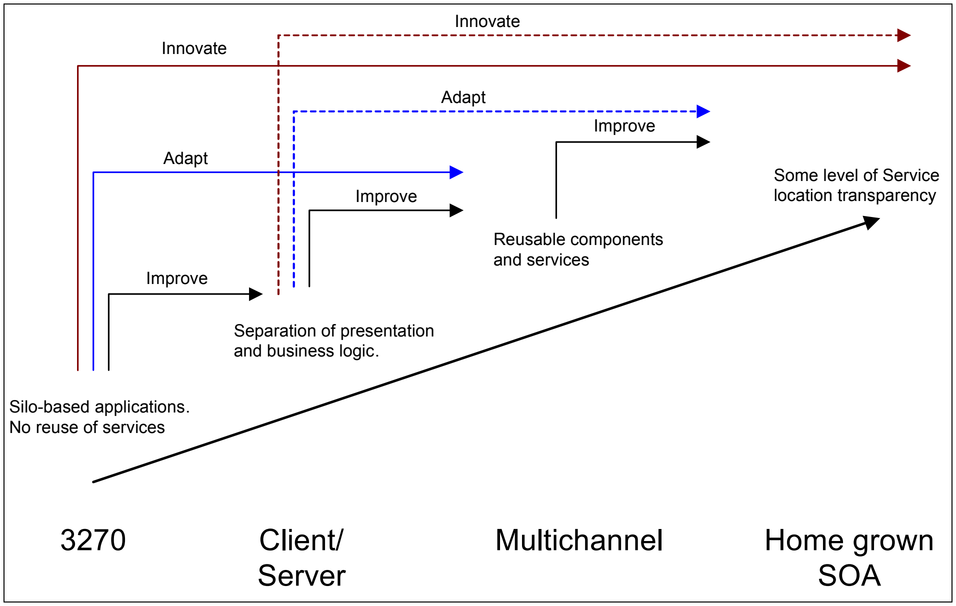


Figure 4-17 Levels of SOA maturity related to transition approaches

Author Comment: Diagram needs to be updated.

There are three primary approaches for transforming an existing core application, as discussed in "Improve the application" on page 123 through "Innovate by re-designing and re-developing the application" on page 124.

Improve the application

In the *Improve* approach we can distinguish between a flavor that involves a user interface and a flavor that does not involve a user interface.

Improve approach with a user interface

In this approach, the user interface, usually a 3270 interface, is wrappered inside a Web application server environment and at the same time the application component is made callable as a service.

A common example is the wrapping of 3270 applications using middleware such as IBM's *Host Access Transformation Services (HATS)*. HATS can be used to wrap a native 3270 application as a Web Service, or to expose that 3270 application as a web server application.

Improve approach without a user interface

In this scenario we do not have an existing (3270) user interface and we only need to "wrapper" the call to the business logic. The "wrapper" communicates with the calling service using a standardized interface, and communicates with the wrapped application using that application's native protocol.

The use of the Improve approach usually leads to the adoption of the "N" (native service interface) or "A" (Adapter-provided service interface) service interface pattern, as dicussed in "Native service interface (N)" on page 119 and "Adapter-provided service interface (A)" on page 120 respectively.

Adapt the application connectivity

This approach is a bit more intrusive to the existing application than "Improve".

To *Adapt* the connectivity implies the separation of business logic from the presentation and/or data logic. Once this is done, the business logic can be accessed as a separate component and reused appropriately. Sometimes, an existing core application is already "componentized" so that the business logic can be accessed directly.

IBM has recommended this separation-of-concerns practice for CICS and IMS developers for many years. But older code often has all tiers of logic intertwined, making it difficult to easily split the logic. In this case, either a less invasive approach must be taken (*Improve*), or the application must undergo significant change and redesign (*Innovate*, see "Innovate by re-designing and re-developing the application" on page 124).

Use of the Adapt transition approach often leads to the adoption of the "N", "A", or "B" service interface patterns, since it usually requires the partitioning of the application logic.

Innovate by re-designing and re-developing the application

The *Innovate* approach is the most invasive to the application.

In this approach, the end result is a restructured, rewritten application that natively conforms to SOA-compliant standards and protocols. However, it is possible that the application may simply be partitioned and refactored, with all mediation and connectivity logic left to the mediation layer (Enterprise Service Bus).

The *Innovated* application is fully modularized so that it can be more easily reused in the future.

Use of the *Innovate* approach usually leads to the adoption of the "R" or the "B" service interface patterns.

4.3.4 Characteristics of the service interface patterns and transition approaches

The transition approaches and associated service interface patterns a number of advantages, disadvantages and differentiating characteristics, as shown in Table 4-1.

Table 4-1 Characteristics of the service interface patterns and transition approaches

Service interface pattern	Common transition approach	Complexity	Relative cost	Relative performance
Native service interface	*Improve*	*Low* Create a native service interface for existing code code "as-is"	*Low* Requires no extra middleware, but uses either a "wrapper" or built-in Web services support.	*Medium*

Service interface pattern	Common transition approach	Complexity	Relative cost	Relative performance
Native service interface	*Adapt*	*Medium* Frequently requires small amount of code modification to access partitioned business logic	*Medium* Requires no extra middleware, but does require more human interaction to modify code	*Medium* to *high* No intermediate middleware, but native transaction manager must translate between SOA protocol and native interface
Adapter-provided service interface	*Improve* or *Adapt*	*Low* to *medium* If "Improve", no changes to code. Tooling and middleware accesses 3270 presentation logic directly If "Adapt", potential need to partition code into business and presentation logic. Additional middleware is required in both cases.	*Medium* Requires additional middleware Code changes are not significant, particularly if an Improve migration technique is used.	*Low* to *medium* Middleware layer adds additional overhead, but can be mitigated if middleware is placed on same OS instance with application. "Improve" solutions incur more overhead, as original user interface is still in use, invoked by adapter middleware
Brokered	*Adapt*	*Medium* Usually requires little code change if broker supports native application protocol or if adapters are used	*Medium* to *high* Although small amount of code change necessary, more middleware infrastructure required. Required infrastructure can mitigate code changes and may be required for full SOA, regardless.	*Medium* Intermediate middleware infrastructure required, adding some overhead, but usually uses native protocol between broker and application

Service interface pattern	Common transition approach	Complexity	Relative cost	Relative performance
Redeveloped	*Innovate*	*High* Involves significant re-engineering and rewrite of application code	*High* Personnel cost for code modification is high. Tool cost can be high, but good tooling is critical to success of redevelopment. High redevelopment cost can be offset in the long run by more efficient development for future SOA applications.	*Medium* to *high* Modernization can result in more efficient code and modularization that maximizes efficiency of composite SOA applications.

4.3.5 Applying the transition approaches and service interface patterns

Similar to the Patterns for e-business, the use of starting and transition scenarios for core system transformation also dictates a sequence of events that guide the migration from an identified starting pattern to a SOA-compliant system.

The high-level view of the transition process:

1. Identify a starting scenario, based upon the existing core system attributes. Chapter 3, “Starting scenarios” on page 53 provides a number oc common starting scenarios.
2. Determine Non-Functional Requirements (NFRs) and Quality of Service (QoS) requirements for the migrated system. The Service Level Agreements (SLAs) play an important role here.
3. Based upon NFRs, QoS requirements and the desired SOA maturity level, select the appropriate transition approach (Improve, Adapt or Innovate). We discussed the Improve, Adapt and Innovate approaches earlier in this chapter.
4. Decide on the desired service interface pattern, as discussed earlier in this chapter. The choices are native service interface, adapter-provided service interface, brokered/mediated service interface and redeveloped code with native service interface.

5. Choose the desired SOA implementation scenario(s). Refer to Chapter 5, "SOA implementation scenarios" on page 129 for a catalogue of the most useful scenarios.
6. Select the solution technique(s) that fit the technological environment.
7. Use appropriate core system modernization tools to perform the transition.

> **Attention:** The above steps may have different outcomes for different applications that need to be SOA-enabled. However, one of the things to keep in mind is to try to focus on a common runtime for most of the new SOA applications.

5

SOA implementation scenarios

This chapter will examine the process of moving from a certain starting scenario (as discussed in Chapter 3, "Starting scenarios" on page 53) to different levels of SOA enablement. It will cover:

- A general process for making architectural decisions, discussed in 5.1, "The architectural decision process" on page 130.
- An analysis of how the three transition approaches (Improve, Adapt, Innovate) can be used to accomplish the SOA enablement. We will provide runtime topologies for each SOA implementation scenario. In some cases, the tools needed to accomplish the migration will also be discussed.
- A brief review of the advantages and disadvantages of the different options for each scenario/approach.
- A sample decision process review to illustrate how a company might select the appropriate transition approach and technologies appropriate to the SOA implementation scenario.

5.1 The architectural decision process

After an architect has identified the starting scenario for the existing application, the analysis process quickly moves to the question: “How should I make this application SOA-enabled?” The selection of a transition approach and the associated transition process is as much an art as a science. The architect must take into account many technical factors, as well as some that have nothing to do with the structure of the application or the existing IT infrastructure.

There are many architectural methodologies in the IT world. IBM uses several design methods, including the IBM Global Services Method, TeAMethod, Rational Unified Process (RUP), and a number of others originating inside and outside IBM. In general, a methodology for creating a solution architecture involves several basic steps:

1. examine the existing IT environment
2. analyze customer requirements, both functional and non-functional
3. study use cases to ensure requirements are met
4. define system context
5. develop architectural overview, based upon architectural decisions
6. develop operational (physical) architecture

There are several key areas of the architectural methodology that are of interest:

5.1.1 The existing IT environment

Many aspects of the existing IT environment will drive the selection of a particular transition approach (Improve, Adapt, Innovate), service interface pattern (native service interface, adapter-provided service interface, brokered/mediated service interface or redeveloped code) and SOA implementation scenario. For example:

- What are the current release and maintenance levels of the z/OS-based transaction and database managers (IMS, CICS, DB2, WebSphere, other non-IBM systems)?
- What products are installed?
- What existing “SOA infrastructure” products are installed (for example, WebSphere Application Server, WebSphere MQ, WebSphere Process Server, etc.)?
- What languages and compilers are in use?
- What language(s) was used to create the application in question?
- What IT architecture and product standards are in place?

- and many others...

Most of the transition approaches have multiple options for implementation, and several of the solution techniques require certain products or releases to be present. The presence or lack of a required product will often drive the decision to a particular variation.

5.1.2 Functional and non-functional requirements

Functional requirements describe *what* a solution does. *Non-Functional Requirements (NFRs)* describe *how* a solution does what it does. An example of a functional requirement is “the solution must present a window with the customer’s name, address and phone number”. An example of a non-functional requirement is “the solution must provide one second response time to the end-user.”

Note that we dicuss the QoS of the z/OS platform associated to the NFRs dicussed in this section in more detail in Chapter 8, “SOA and z/OS QoS” on page 267.

When selecting a transition approach to enable an application for SOA, the Non-Functional Requirements tend to be the most important to making decisions. The Non-Functional Requirements that are of greatest interest include:

- *Performance*
  How “fast” must the solution function? Is the measurement of end-user response time or internal? The use of Service Level Agreements (SLA) is important - is the performance of the solution meeting the requirements agreed upon by the end-user community?
  - When selecting a solution, choose the option that provides optimal performance while still fulfilling the remainder of the functional and Non-Functional Requirements. The best performing solution is not always the best solution overall.
- *Scalability*
  An application that scales is one that can provide an increased transaction load without impacting the Quality of Service provided. Different infrastructure options provide differing scalability options. For example, a message broker on a particular server model can process a certain number of messages per second. How can that number be increased (scaled)? By adding more server instances (*horizontal scaling*)? By increasing the capacity of the hardware (*vertical scaling*)?

  - When selecting a solution, ensure that the technical variation uses components that can be scaled to increase the transaction rate to meet requirements.
- *Flexibility*
  Flexibility is a rather nebulous concept, but an important one. One of the key aspects of SOA is to provide a more flexible environment for implementing new applications and changing existing ones. SOA introduces flexibility by abstraction of functions such as network protocols, mediation of message content, workflow, and provides separation of business logic from the other parts of the application. This allows the developer to more easily assemble composite applications from the “parts” - the services.
  - When selecting a solution, ensure that flexibility is maximized while maintaining the other requirements. A more flexible solution may save money in the long run, even though it may introduce additional performance overhead or other “negatives.”
- *Reliability and Availability*
  Reliability and it’s related concept, availability, define how *resilient* the application is to failures in the application itself and in the hardware and software infrastructure that supports it. Reliability and availability are impacted by the complexity of the infrastructure (points of failure) and the means by which the components recover from failure. Often, mechanisms such as server clustering and failover can improve reliability and availability by maintaining application availability even though one instance of the infrastructure component or application has failed.
  - When selecting a solution, be aware of the ways that the components maintain availability. Some variations have more complex infrastructures than others, and as a result have more potential points of failure. Be prepared to mitigate potential failure points by using the middleware facilities such as clustering. Resist the temptation to eliminate middleware simply because of availability issues - often, middleware provides value that offsets its potential for failures. Removing a component such as an ESB may slightly improve overall availability, but it can greatly reduce the flexibility of the solution, incurring additional cost.
- *Security*
  Security requirements usually relate to two functions: authentication and authorization. How does the solution allow for the verification of the individual’s identity, and how does it authorize the user to access resources? Also, functions such as encryption and key management are important to the security of SOA-based solutions. Some computing platforms (z/OS, for example) tend to be inherently more secure than others. Platform security usually depends on the presence of operating system security managers, such as RACF, or other inherent hardware or software security features,

including mainframe features like hardware-based encryption, storage protect keys, and built-in public-key certificate infrastructures.

  - When selecting a solution, consider the security features of the platform where the services and the infrastructure will reside. Consistent with SOA principles, make sure that as many of the security functions as possible can be externalized from the services and defined in security infrastructure.

- *Skills*
  The presence or lack of skills is often one of the main architectural decision criteria. Does the programming staff have skill in the needed programming languages? Does the proposed solution introduce new infrastructure components that will require training? How many administrators will the solution require? Does the solution introduce new operating systems? The cost of acquiring skill and the personnel with the skills must be considered when designing the solution.

  - When selecting a solution, consider the types of skills necessary, in service development and in administration of the middleware included. While an SOA implementation is intended to reduce development time and simplify the development process, it often increases the complexity of the infrastructure, and new skills may be needed to administer the underlying SOA middleware.

- *"Organizational ideology"*
  The "ideology" (sometimes known as "politics") of the organization can be the most sensitive decision point to deal with when designing a solution. Everyone, even consultants who are supposed to be "unbiased," has personal preferences. There is often an institutionalized bias against particular solutions. For example, a team may have had a bad vendor experience or dealt with a product that was particularly buggy, and therefore has a bias against that vendor or product. The corporate or IT leadership may have been hired from another company that did things a certain way, and their familiarity with that method or technology influences their decisions. There are many factors that influence organizational ideology, and not all of them are based on fact or logic. The architect must be aware of these "preferences" and design a solution that takes them into account

  - When selecting a solution, be aware of the informal preferences of the organization, particularly of those who will be approving the architectural decisions. If selecting an infrastructure component or a platform that is in jeopardy due to ideology issues, be prepared to back the selection with reasonably unbiased evidence and comparisons with other options. The use of a standardized architectural decisions document is a good approach for documenting the decision process.

5.1.3 The architectural overview and operational architecture

The IT architect is often tempted to jump directly to the creation of the architectural overview and definition of a physical/operational architecture without performing a complete analysis and architectural decision process (a “dinner napkin” design). However, the other steps in the process mentioned above are critical to arriving at a working architecture that is well-documented and justifiable to the decision-makers in the enterprise.

The definition of the target architecture by using the pattern-based approach in this book helps build an architecture overview and physical architecture by using established patterns, rather than by “building from scratch.” The patterns described in this book are not the complete solution, but are starting points for building the architectures that enable migration to an SOA-based architecture from the starting scenarios described earlier.

5.1.4 Selecting a transition approach and solution technique

The architectural decision process detailed in 5.1, “The architectural decision process” on page 130 should be applied to the selection of the transition approach, solution technique and SOA implementation scenario. An architect has to find answers to the following questions:

- *What is the current IT infrastructure?*
  Depending upon the existing IT environment, some of the solution techniques may or may not make sense. Some solution techniques require specific products and/or releases.
- *What does the existing core application do? How is it implemented*
- *What starting scenario does the existing application map to?*
- *What are the requirements for the new, SOA-compliant service that is being designed?*
  In particular, the Non-Functional Requirements will drive the selection of a solution technique.
- *How much time and effort can we expend on this - is this a “quick-fix” project, or are we building a service that is expected to have a long “life-span?”*
  The answer to this question will provide much insight to determine whether to use an “Improve,” “Adapt,” or “Innovate” transition approach to modernization and service enablement.
- *What resources and skills are available to perform the work?*
  If developer resources are scarce, then a transition approach or solution technique that depends upon code changes may not be a good idea. Or, if an organization has no Java resources and the solution is based on WebSphere Application Server and Java, then that solution may not be a good choice.

- *How much money is available?*
  If funding is not available, a solution technique that requires new infrastructure may not make sense.

Given the answers to these and other questions, the architect can examine the transition approaches and solution techniques and determine which approach and solution technique is best for the organization. The architect should evaluate each option with those questions and the NFRs in mind.

5.2 SOA implementation scenarios for 3270 application

This section describes variations of the SOA implementation scenarios for 3270 applications. We described the typical 3270 application as a starting scenario in 3.1, "Starting scenario - 3270 application" on page 55. Table 5-1 provides a list of the variations discussed throughout this section.

Table 5-1 Summary of 3270 transition variations

Transition approach	Variation	Described in
Improve	Variation 1: service enablement of 3270 prgrams using IBM Host Access Transformation Services (HATS)	"Variation 1: IBM Host Access Transformation Services (HATS)" on page 138
Improve	Variation 2: service enablement of CICS transactions using the CICS Service Flow Feature (CICS Transaction Server V3.1)	"Variation 2: CICS Service Flow Feature (CICS Transaction Server V3.1)" on page 139
Improve	Variation 3: service enablement of IMS 3270 transactions using the IMS MFS Web Services Support feature	"Variation 3: IMS using MFS Web Support" on page 141
Adapt	Variation 1: service enablement using Web Services support in CICS	"Variation 1: Native Web services access to CICS transactions" on page 146
Adapt	Variation 2: service enablement using J2C connector (CICS TG) to CICS	"Variation 2: J2EE Connector Architecture access to CICS" on page 147

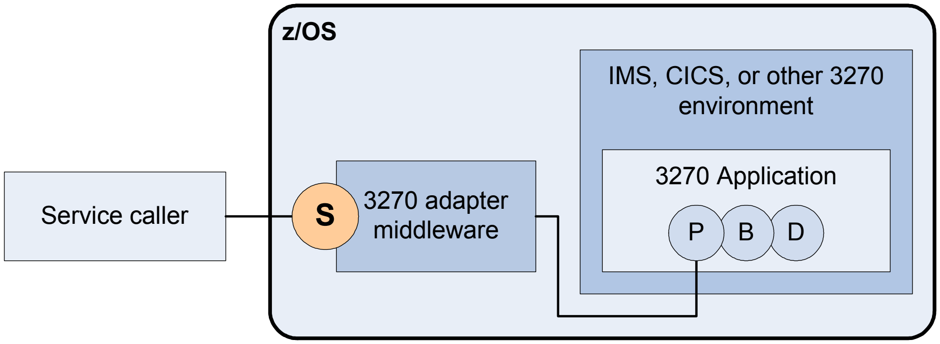
Transition approach	Variation	Described in
Adapt	Variation 3: service enablement using WebSphere MQ / JMS access to CICS	“Variation 3: WebSphere MQ access to CICS” on page 150
Adapt	Variation 4: service enablement using J2C connector (IMS Connect) to IMS	“Variation 4: Using IMS Connect” on page 152
Adapt	Variation 5: service enablement using IMS SOAP Gateway to access IMS	“Variation 5: Using the IMS SOAP Gateway” on page 154
Adapt	Variation 6: service enablement using WebSphere MQ / JMS access to IMS	“Variation 6: Using the WebSphere MQ to IMS Bridge” on page 157
Adapt	Variation 7: service enablement and integration using an ESB	“Using the Adapt approach with a broker/ESB” on page 159
Innovate	Redeveloped code with a clear separation of concerns and high reusability.	5.2.3, “Using the Innovate transition approach” on page 160

5.2.1 Using the Improve transition approach

Following the *Improve* transition approach for 3270 applications generally involves providing a new front-end interface. Many customers simply choose to implement a HTML/Web front-end on the 3270 application. However, this does not provide a SOA-compliant service interface. Since improving the application addresses the problem at the user interface, an improve solution interacts with the existing application via the 3270 data stream. All input and output messages to the 3270 application are done via the 3270 native protocol, and no changes are made to the underlying application

The logical architecture for an “Improve” approach is shown in Figure 5-1.

Figure 5-1 Logical architecture for Improve approach



A note about notation: In the diagrams in this chapter, the service interface is represented by the circle with an “S”. The placement of the service interface is important to the architecture of the SOA-enabled host solution, and the solution technique used for the solution dictates the location of that service interface.

Solution techniques for “Improve”

When using an Improve transition approach against a 3270 starting scenario, all of the solution techniques follow the *adapter-provided service interface* pattern (refer to “Adapter-provided service interface (A)” on page 120). A 3270 presentation adapter is used to translate between the incoming Web Services messages and the 3270 data stream. The variations dicussed differ from the rest of the Adapter patterns in their native access to 3270 presentation logic.

The remainder of the adapter-provided service interface pattern variations, described in section “Adapter-provided service interface (A)” on page 120, are derived from the Adapt transition approach, and they access the 3270 application’s business logic directly, bypassing the 3270 layer.

We now look at three solution techniques for the Improve approach and the adapter-provided service interface pattern:

- IBM Host Access Transformation Services (HATS)
- CICS Service Flow Feature (SFF)
- IMS MFS Web Services support

Variation 1: IBM Host Access Transformation Services (HATS)

A common approach to Improve is the use of the *Host Access Transformation Services (HATS)* product. HATS is a WebSphere application that performs two main functions:

- translate a 3270 application into a Web interface
- translate a 3270 application into a Web Services interface

This is accomplished with a combination of the HATS runtime and tooling running under Rational Application Developer (RAD). The HATS developer uses the HATS "perspective" in RAD to navigate through the 3270 application and capture the various inputs, outputs, and record the interactions needed to communicate with the 3270 application.

The process generates a series of integration objects - Java beans that can subsequently be used in either a Web server interface or to generate a SOA/Web services interface. These Java beans are deployed to the WebSphere Application Server and run using WebSphere and the HATS runtime. HATS can be used to service-enable *any* 3270 (or 5250 or VT-based) transaction, including CICS, IMS, TSO, and so on.

Solution technique implementation

The HATS variation is an implementation of the "adapter-provided service interface" pattern. HATS, running in WebSphere, is the "adapter" that transforms the inbound Web Services protocol into a 3270 data stream that is understood by the host application. In a HATS solution, the service interface is provided by WebSphere Application Server, since the services are running inside WAS.

Process overview

SOA-enabling a 3270 application using HATS involves several steps:

1. The developer uses the HATS perspective in RAD to navigate through the 3270 application and records the user interactions, input and output fields, etc.
2. The developer creates Java beans from the HATS artifacts.
3. The developer uses Web services wizards in RAD to create services from the HATS Java beans.
4. The developer deploys Java artifacts to WebSphere Application Server.
5. The Java artifacts may be exposed as services.

Figure 5-2 on page 139 shows the HATS (running on z/OS) Web services solution.

Note: This is only one topology for running HATS. The HATS server is also supported on distributed platforms, including Linux and Windows.

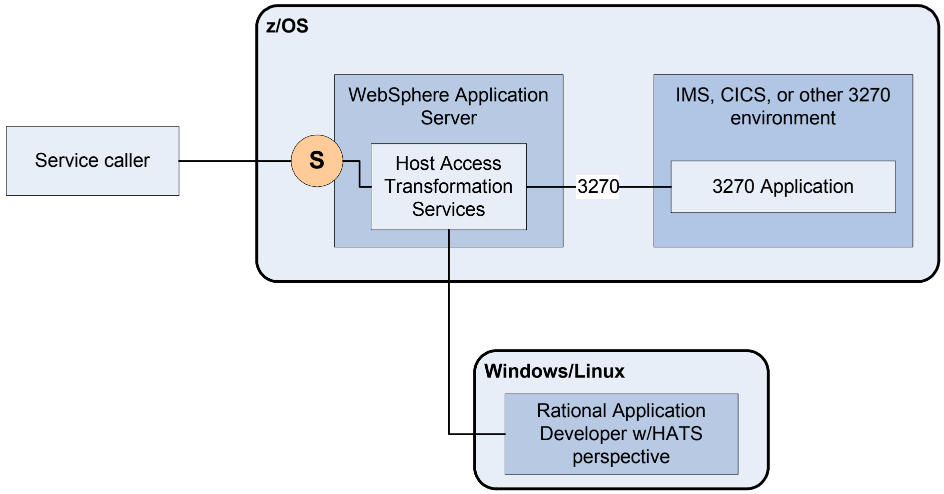


Figure 5-2 Improve using IBM Host Access Transformation Services (HATS)

For detailed information

- Redpaper: Host Access Transformation Server Concepts and Architecture - `http://www.redbooks.ibm.com/abstracts/redp3706.html`
- Redbook: Using IBM WebSphere Host Access Transformation Services V5 - `http://www.redbooks.ibm.com/abstracts/sg246099.html`
- Online InfoCenter: `http://publib.boulder.ibm.com/infocenter/hatshelp/v60/index.jsp`

Variation 2: CICS Service Flow Feature (CICS Transaction Server V3.1)

In CICS Transaction Server V3.1, a new feature is available to Improve CICS applications. The *Service Flow Feature*, announced in November, 2005 (Announcement Letter 205-303) is a no-charge, separately-orderable feature that allows the developer to create CICS business services that are composed of a sequence of CICS application interactions. These applications may be 3270 applications or other CICS applications that do not use the 3270 interface. Service Flow Feature is comprised of two components:

- *Service Flow Modeler (SFM)*, used to model the flow between CICS services, create the interactions with the CICS applications (3270 and/or non-3270 COMMAREA-accessible), and expose these services as a Web service.

- *Service Flow Runtime (SFR)*, which delivers the adapters and other supporting code to execute the flows and 3270 interactions created using the Service Flow Modeler.

The Service Flow Modeler (SFM) is a feature of *WebSphere Developer for zSeries (WDz)*, an Eclipse-based development tool built on top of the Rational Application Developer base. CICS Service Flow Feature ships with a number of limited-use licenses of WDz, specifically for use with Service Flow Feature and the CICS Web Services feature. If a customer wishes to use WDz for purposes other than the creation of CICS Web services or SFM flows, they must purchase a license for the full WDz product.

In a Service Flow Feature solution, the service interface is provided by CICS itself, as the CICS Transaction Server has its own SOAP listeners, provided by CICS's Web Support and/or the WebSphere MQ Trigger Monitor.

Solution technique implementation

While the CICS Service Flow Feature solution is generally considered to fit an "adapter-provided service interface" pattern, it could also be considered a "native service interface" pattern, as the adapter function and the Web service interface are entirely self-contained within the CICS transaction manager.

Process overview

SOA-enabling a 3270 CICS application using the Service Flow Feature solution involves:

1. The developer uses the CICS Service Flow Modeler in WDz to define the 3270 interactions and flows between 3270 programs (if required).
2. The developer generates artifacts to be deployed to CICS (message adapters, server adapters, JCL to create RDO definitions and properties files, etc.).
3. The artifacts are transferred to CICS.
4. Service is ready for use.

Figure 5-3 on page 141 shows the physical architecture for a CICS Service Flow Feature SOA solution technique.

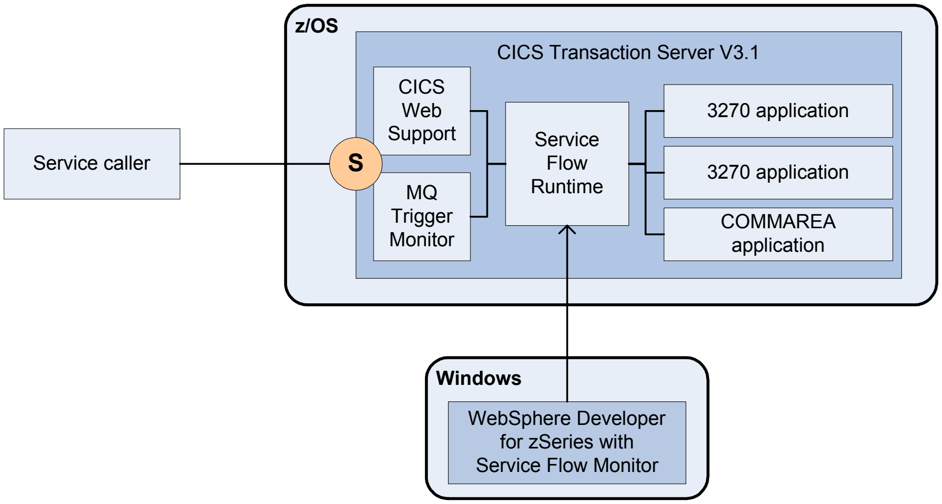


Figure 5-3 Improve CICS applications using CICS Service Flow Feature

For detailed information

- Redbook: Application Development for CICS Web Services - `http://www.redbooks.ibm.com/abstracts/sg247126.html` (Chapter 7)
- Product manual: CICS Service Flow Feature for CICS TS for z/OS V3.1 Run Time User's Guide (SC34-5899) - downloadable from `http://www-306.ibm.com/software/htp/cics/tserver/v31/library/`
- Online InfoCenter: `http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp` (see section titled “CICS Service Flow Runtime V3.1)

Variation 3: IMS using MFS Web Support

Message Formatting Services (MFS) is an IMS facility that formats messages to and from terminal devices and IMS application programs. It is used to separate the application logic from the device logic. This is illustrated in Figure 3-4 on page 57.

IMS MFS Web Support is a solution that enables IMS MFS applications to be published as Web services, EJBs or Java Beans. It consists of two parts:

- *IMS MFS Web services support*.
  Figure 5-4 on page 142 illustrates IMS MFS applications being exposed as a service using IMS MFS Web services support.

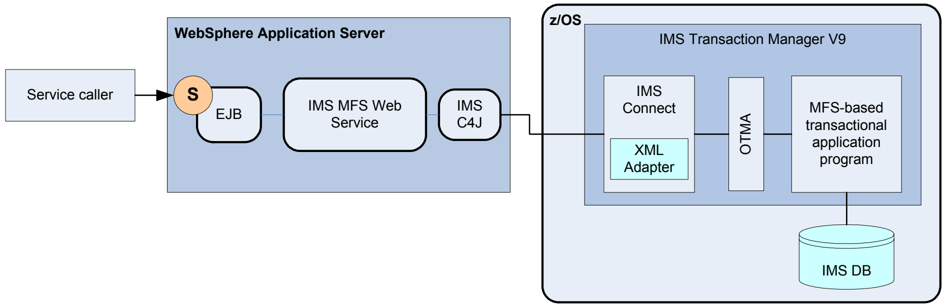


Figure 5-4 Overview of IMS MFS Web services support

- *IMS MFS Web enablement*. Figure 5-5 shows IMS MFS applications being reused as a Web application. In this cases there is no service interface. This is called the IMS MFS Web Enablement support, which is also part of IMS MFS Web Support.

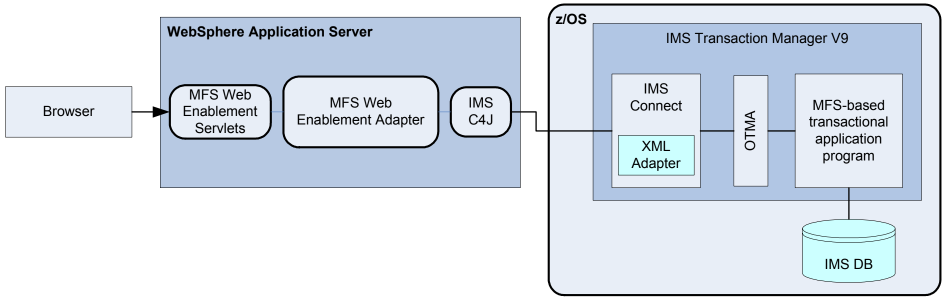


Figure 5-5 Overview of IMS MFS Web enablement

Solution technique implementation

Both IMS MFS Web services and IMS MFS Web enablement implement the “Adapter-provided service interface” pattern since they translate between a proprietary protocol and an SOA-compliant protocol.

Process overview

For the IMS MFS Web services solution, these are the steps:

1. Developer uses the IMS MFS Web services tooling, which is included in WebSphere Integration Developer (WID), and imports the require MFS source files.
2. The service is defined from the MFS source file, and the following artifacts are generated

  - XMI files
  - MID/DIF
  - MOD/DOF
  - MFS table
  - WSDL
3. From WSDL files, the following are generated:
  - Input/Output Beans
  - Format Handlers
  - Service Proxies
4. Service is published and deployed to WAS, which makes it available as a Web service, EJB or Java bean.

For the IMS MFS Web enablement solution, the process is more or less the same, except that there is no WSDL or EJBs generated. Instead, there are Java servlets and stylesheets generated, that will be deployed in WAS.

For detailed information

- Redbook:
  http://www.redbooks.ibm.com/

Observations using the Improve transition approach with the SOA implementation scenario - 3270 application

Advantages:

- Quick implementation - simple tooling and deployment, quick time-to-market
- No impact on existing applications
- Low complexity
- HATS is a generalized 3270 adapter (a "universal 3270 adapter") that functions with any 3270 application
- With the CICS Service Flow Feature variation, no additional software required, besides CICS Transaction Server V3.1
- CICS Service Flow Feature performs the service enablement inside CICS, reducing points of failure and overhead

Disadvantages:

- The HATS variation requires additional software (HATS, WebSphere Application Server)
- Additional overhead incurred at 3270 translation layer
- Modifications to 3270 user interface can impact functionality of integration
- Service granularity. IMS 3270 transactions are not designed for participating in an SOA, and may not have the right granularity for being invoked as a service.
  It may be necessary to perform service modeling techniques in order to achieve the right level of granularity and reuse.

5.2.2 Using the Adapt transition approach

An "Adapt" transition approach for transforming a 3270 application to SOA requires *direct* access to the business logic of the application and bypasses the 3270 presentation layer. This may be a relatively easy task, depending upon the structure of the existing application code.

For example, some CICS applications have been partitioned to separate the presentation interface from the business and/or data logic. Older CICS applications often have the presentation and BMS code interspersed within the business logic. In programs where the logic is partitioned, a COMMAREA construct is often used to pass parameters between the calling and called programs.

In CICS, this calling approach is known as *Distributed Program Link (DPL)*. Most of the variations to those solution techniques which use an Adapt transition approach require a COMMAREA (or similar construct in IMS) to function.

The logical architecture for the solutions based on an Adapt transition approach is shown in Figure 5-6.

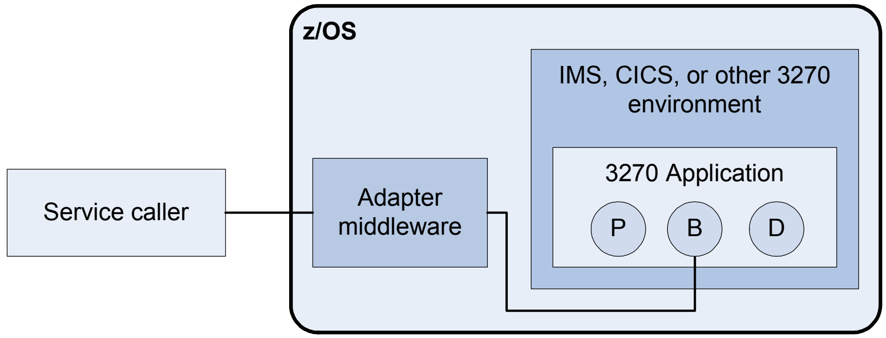


Figure 5-6 Logical architecture for the Adapt transition approach

Solution techniques for "Adapt"

Solutions derived from the Adapt transition approach all access the business logic directly, with no "reuse" of the 3270 business logic. There are three service interface patterns (refer to 4.3.2, "Service interface patterns" on page 119) that are the end-result of an Adapt transition approach:

- *Native service interface*
  Application code is partitioned and business logic is accessed directly within the bounds of the transaction manager or database server. The "adapter middleware" and the service interface are provided natively, inside the transaction manager or database server and not by an external software product or component.
- *Adapter-provided service interface*
  Application code is partitioned within the transaction manager or database server, making business logic directly accessible. The service interface is placed in the "adapter." External middleware is used to provide the adapter functionality.
- *Brokered/mediated service interface*
  Application code is partitioned in the transaction manager or database server, making business logic directly accessible. The service interface is provided by a broker or ESB. Applications are often accessed via a messaging protocol (JMS or WebSphere MQ). The support for those protocols can either be native to the application or provided by the transaction manager or database server. Modern brokers, now referred to as the "Enterprise Service Bus"

(ESB), can integrate with existing transactions using other protocols, such as SOAP or native interfaces to transaction systems.

Variation 1: Native Web services access to CICS transactions

Beginning in CICS Transaction Server Version 2.3, a native SOAP interface to CICS transactions became available. In Version 2, this was known as the “SOAP for CICS” feature and was a separately orderable CICS feature. As of CICS Transaction Server Version 3.1, the support for SOAP has been enhanced and is now known as the “CICS Web Services” feature. In the Version 3.1 feature, the new WS-I standards[1] such as WS-Security, WS-Basic Profile, and WS-Transaction are supported. In addition, the creation of COMMAREA-to-XML mappings and other related “adapter code” creation tasks have been simplified.

The CICS Web Services feature provides a SOAP interface to existing CICS transactions. SOAP can flow via either HTTP or JMS/MQ, and there is support for either inbound (CICS as a service provider) or outbound (CICS as a service consumer) requests. Inbound/outbound SOAP messages are processed by a CICS BTS pipeline which removes the payload from the SOAP message and parses the XML contents to produce a COMMAREA or a CONTAINER to be passed to (or from) the CICS program that is being called (or that is calling out).

Solution technique implementation

The CICS Web Services feature is an implementation of the “native” service interface pattern. SOA-compliant connectivity to the CICS application is provided natively by CICS Transaction Server via the Web Services feature. All connectivity and processing of the inbound and outbound Web services protocols is handled by CICS. The service interface is located inside CICS, and is provided by the CICS Web Services feature.

Process overview

SOA-enabling a 3270 CICS application using the CICS Web Services feature is known as a “bottom-up” implementation. It involves these steps:

1. Obtain the CICS application language structures (COPYBOOKs) and modify/simplify if necessary.
2. Using the language structures as input, run the CICS Web Services Assistant to generate the WSDL, WSBind files, and other artifacts to be deployed to CICS. Deploy them to the z/OS HFS.
3. Define the necessary resources to CICS, including the transport (HTTP or MQ) PIPELINE, URIMAP and WEBSERVICE definitions.
4. Test and use the service.

[1] Details on the Web Services Interoperability Organization and its standards can be found at `http://www.ws-i.org`

Figure 5-7 shows the physical architecture for an SOA solution based on CICS Web Services.

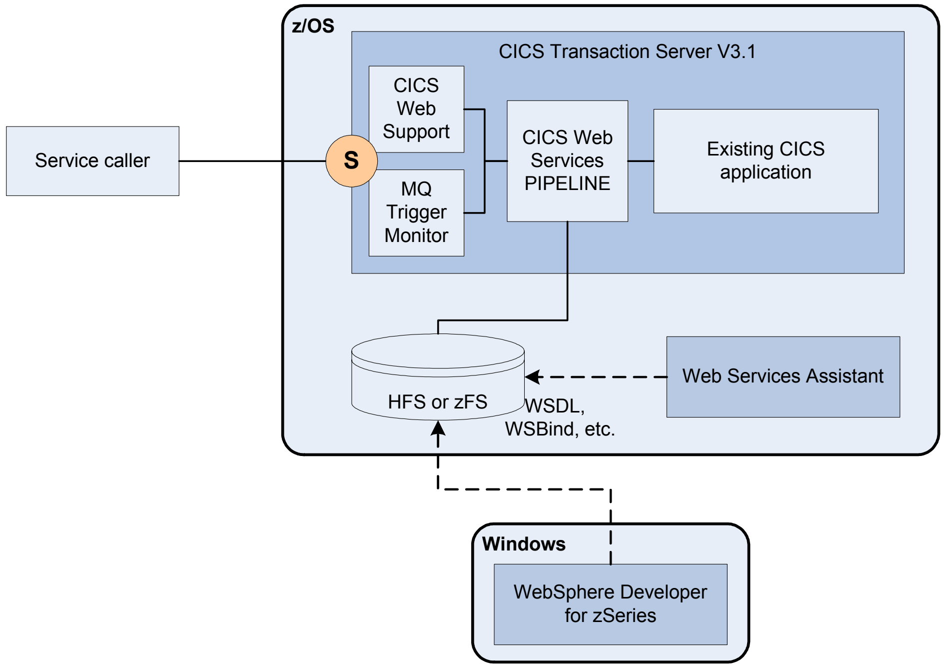


Figure 5-7 Adapt CICS applications using CICS Web Services

For detailed information

- Redbook: Application Development for CICS Web Services - `http://www.redbooks.ibm.com/abstracts/sg247126.html`
- Redbook: Implementing CICS Web Services - `http://www.redbooks.ibm.com/abstracts/sg247206.html`
- Product manual: CICS Web Services Guide (SC34-6458) - downloadable from `http://www-306.ibm.com/software/htp/cics/tserver/v31/library/`
- Online InfoCenter: `http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp`

Variation 2: J2EE Connector Architecture access to CICS

CICS provides the ability to invoke a transaction from a Java2 Enterprise Edition (J2EE) application program. The J2EE Connector Architecture (J2C, or J2CA) is the standard mechanism for calling applications from a J2EE program synchronously. It is used to invoke applications running in IBM transaction managers including IMS and CICS, and other “Enterprise Information Systems” such as PeopleSoft®, SAP®, Oracle® ERP, and others.

An overview of the J2EE Connector Architecture is shown in Figure 5-8.

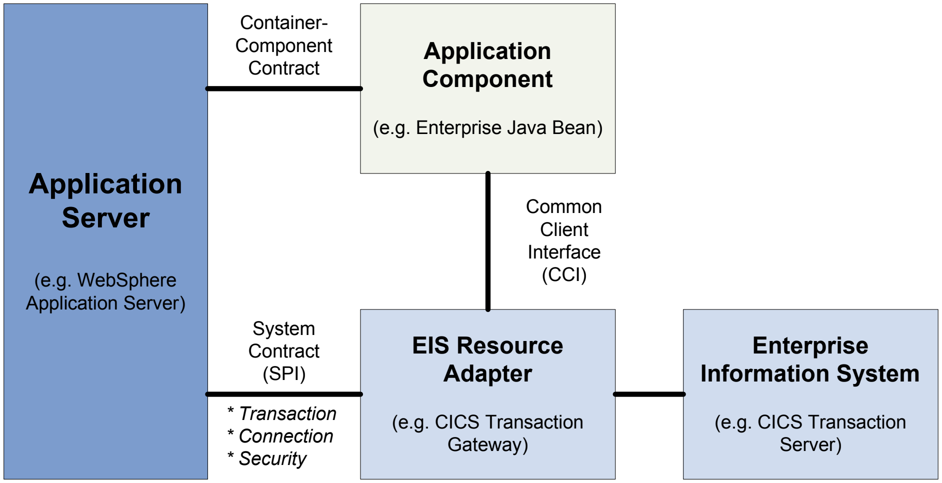


Figure 5-8 J2EE Connector Architecture

CICS transactions can be invoked via the J2C API from within a J2EE program. A J2EE application running in an application server that supports J2C, such as the WebSphere Application Server, can use the J2C classes to call a transaction. The EIS Resource Adapter, as shown in Figure 5-8, is an “intermediary” that transforms the J2C syntax into a form that the EIS understands. To access CICS Transaction Server applications, IBM’s implementation of the EIS Resource Adapter is the *CICS Transaction Gateway (CICS TG)*. CICS TG transforms the native Java J2C API into calls to the CICS transactions.

As with the other “Adapt” solutions, the CICS program’s business logic must be seperate from the presentation logic and DPL-accesible through a COMMAREA. CICS TG takes the input parameters from the J2EE program, formats them into a CICS-acceptable COMMAREA, invokes the program, and passes the response back to the J2EE caller.

The Rational Application Developer tool and its “J2EE Connector Tools” feature provide wizards and Java classes to assist the developer in building J2EE applications that use J2C to access a CICS program. Chapter 6 of the IBM Redbook titled *WebSphere for z/OS Version 6 Connectivity Handbook*, SG24-7064-02 provides details on the development process.

Solution technique implementation

The J2C access to CICS variation is an implementation of the “Adapter-provided service interface” pattern. The J2EE application running in the Web application server (WebSphere Application Server or other compatible server), combined with the CICS Transaction Gateway, provide an “adapter” that is external to CICS. The service interface is located in the Web application server.

Process overview

The major steps for developing a service that uses J2C access to CICS are:

1. Install and customize the CICS Transaction Gateway.
2. Obtain the CICS application language structures (COPYBOOKs) and modify/simplify if necessary.
3. Use Rational Application Developer and the appropriate wizards to generate Java classes that populate the fields in the COMMAREA to be passed to the CICS transaction and call the transaction.
4. Use RAD’s Web services wizards to define the service interface to the J2C application and produce the WSDL.
5. Deploy the resulting EAR file and other artifacts to WebSphere Application Server.
6. Access the service.

Figure 5-9 on page 150 shows the physical architecture for a SOA solution based on J2C access to CICS.

> **Note:** There are several different topology implementation options for WebSphere Application Server and CICS Transaction Gateway. For example, both the WebSphere Application Server and the CICS Transaction Gateway can run “off platform” on distributed servers. The IBM Redbook titled “*CICS Transaction Gateway for z/OS Version 6.1*” (SG24-7161) describes the topology options and their implementation processes. Figure 5-9 only shows the “z/OS-centric” topology.

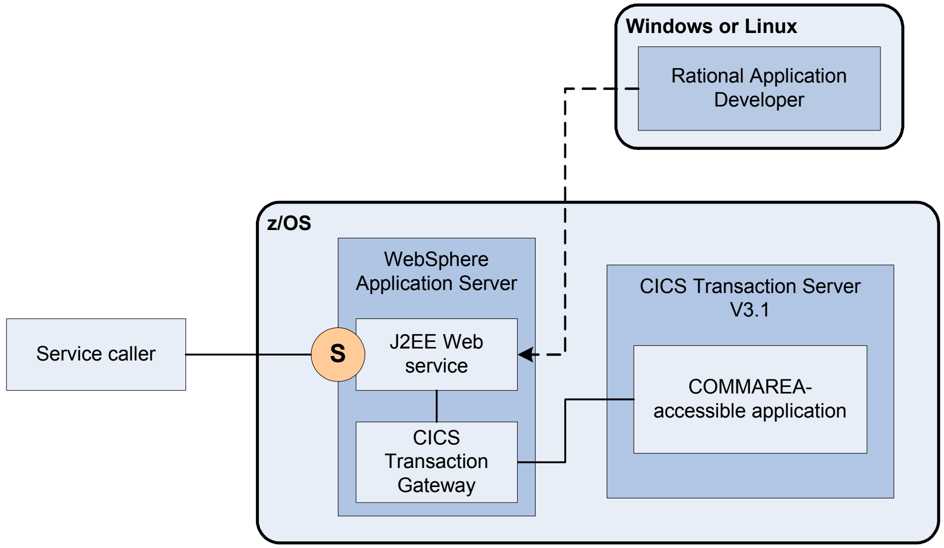


Figure 5-9 Adapt CICS applications using the J2EE Connector Architecture

For detailed information:

- Redbook: CICS Transaction Gateway for z/OS Version 6.1 - `http://www.redbooks.ibm.com/abstracts/sg247161.html`
- Redbook: Revealed! Architecting e-businss Access to CICS **-** `http://www.redbooks.ibm.com/abstracts/sg245466.html`
- Redbook: WebSphere for z/OS V6 Connectivity Handbook - `http://www.redbooks.ibm.com/abstracts/sg247064.html`
- Redpaper: WebSphere for z/OS to CICS and IMS Connectivity Performance - `http://www.redbooks.ibm.com/abstracts/redp3959.html`
- Online InfoCenter: `http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp` (see section titled “CICS Transaction Gateway.”)

Variation 3: WebSphere MQ access to CICS

CICS transactions can be invoked via a message received on a message queue. This function is provided by a combination of WebSphere MQ and either the WebSphere MQ CICS Bridge (shipped with WebSphere MQ) or the WebSphere MQ Trigger Monitor task. In both cases, the “service caller” invokes the program by using the MQ API to place a message on a queue that is monitored either by the Bridge or the Trigger monitor.

The MQ message payload is the COMMAREA to be passed to the CICS program. When the Bridge or the Trigger Monitor receives the inbound message, they invoke the CICS program and optionally place the response COMMAREA in a message on a REPLY-TO queue. A REPLY-TO queue is also used for error conditions that result from the call.

The *CICS Link3270 Bridge* or the older CICS 3270 Bridge component can also be used with the WebSphere MQ CICS Bridge to invoke a 3270 program that does not have a COMMAREA interface.

Neither the WebSphere MQ CICS Bridge nor the WebSphere MQ Trigger Monitor provide a WSDL-described Web services interface. SOAP is not used as the message format - it is “native” MQ. This is a different scenario than using MQ or JMS as a SOAP transport. If a Web services compliant solution is needed, an Enterprise Service Bus (ESB) can be used to place a SOAP-compliant layer between the service caller and the Bridge or Trigger Monitor. This will mean a *Brokered/Mediated service interface* pattern.

Solution technique implementation

The WebSphere MQ access to CICS variation is an implementation of the “adapter-provided service interface” pattern. The WebSphere MQ middleware and the Bridge and Trigger Monitor features act as the “adapter” between the service requester and the CICS application. While these applications are not represented as Web services, they still are services which are enabled for use within a composite application.

Process overview

There are several key steps to implement a WebSphere MQ interface to a CICS application:

1. Install and configure either the WebSphere MQ CICS Bridge or the WebSphere MQ Trigger Monitor.
2. Code the service requester program to create a COMMAREA-compatible data area to pass to the service.
3. Add MQ API calls to place a message containing the COMMAREA and appropriate MQ headers onto a queue that is monitored by the WebSphere MQ-CICS Bridge or the Trigger Monitor.

Figure 5-10 on page 152 shows the physical architecture for WebSphere MQ access to CICS transactions via the WebSphere MQ-CICS Bridge.

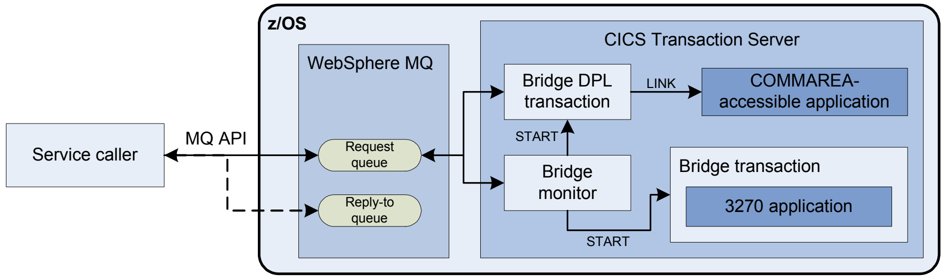


Figure 5-10 Adapt CICS applications using the WebSphere MQ-CICS Bridge

For detailed information

- Redbook: Revealed! Architecting e-businss Access to CICS - http://www.redbooks.ibm.com/abstracts/sg245466.html
- Redpaper: WebSphere for z/OS to CICS and IMS Connectivity Performance - http://www.redbooks.ibm.com/abstracts/redp3959.html
- Whitepaper: Using a message-based approach to integrate your CICS system with your entire IT infrastructure (G325-2113) - http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=G325-2113-00

Variation 4: Using IMS Connect

The "Adapt" scenarios for IMS illustrate how backend functions can be exposed as services. This is done by creating a new layer with service interfaces based on EJB components, which in their turn, call the backend IMS transaction. These EJB components will enrich the functionality, and be able to provide logic, which will increase the value and ease of reuse in a modern SOA.

IMS Connect enables TCP/IP connectivity to IMS. A client can be any kind of client, as long as it has TCP/IP connectivity. For this example, the caller is an EJB.

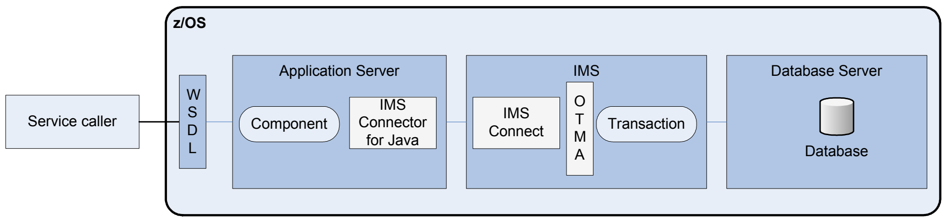


Figure 5-11 Overview of the IMS Connect scenario

In this scenario, the client to IMS is a J2EE application server, already running on z/OS. Logic is implemented in EJBs. Based on these EJBs there are service definitions published using WSDL and accessible from different channels, e.g. through SOAP. The EJBs use the J2EE Connector Architecture (J2C) to call IMS Connect. JCA is discussed in "Variation 2: J2EE Connector Architecture access to CICS" on page 147.

Example of the flow of a message using IMS Connect

We now discuss the flow of a request, as shown in Figure 5-12.

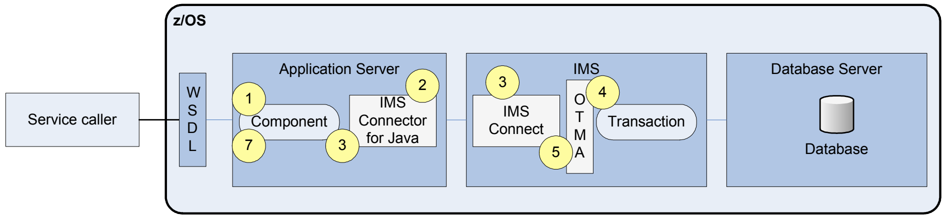


Figure 5-12 Walkthrough of the IMS Connect example

1. The application server receives the SOAP message from the client application. It processes the SOAP header and calls the EJB that matches the input request.
2. The EJB uses the J2EE Connector Architecture (J2C) to obtain a connection to IMS Connect, and builds the required structure(s) for the call. Then it executes the call.
3. IMS Connect receives data from the application server (TCP/IP) client, performs basic editing and translation, invokes security, and prepares the message in the OTMA format.
4. OTMA gets the message, and passes it on to the receiving transaction.
5. Response from the transaction is passed to OTMA, and IMS Connect.
6. IMS connect formats the response in a way that the application understands, and the control is returned to the EJB.
7. The EJB processes the reply and returns a reply to the service caller.

Solution technique implementation

The IMS Connect scenario is an implementation of the aAdapter-provided service interface" pattern. The J2EE application running in the Web application server (WebSphere Application Server or other compatible server), combined with the IMS Connector for Java, provide an "adapter" that is external to IMS. The service interface is located in the Web application server.

Process overview

1. Install and configure IMS Connect. OTMA (Open Transaction Manager Access) is already part of IMS.
2. Install and configure WebSphere Application Server. Configure connection management to IMS.
3. You can choose to code the logic yourself in order to use IMS Conector for Java. Or you can use a tool, such as WebSphere Integration Developer or Rational Application Developer to generate the necessary structures and beans.
4. Deploy the application to the application server.

For detailed information

- Redbook: IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity - `http://www.redbooks.ibm.com/abstracts/sg246794.html`
- Product documentation: IMS Connect `http://www-306.ibm.com/software/data/db2imstools/html/imsconnectv12.html`
- WebSphere Integration Developer `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp`

Variation 5: Using the IMS SOAP Gateway

One way to open up existing IMS transactions, and to make them easily accessible from different service callers, is to expose them as Web services.

IMS SOAP Gateway is a Web services solution that enables IMS applications to interoperate outside of the IMS environment through Simple Object Access Protocol (SOAP). By doing this, the IMS transaction can easily be reused from different channels through the IMS SOAP Gateway.

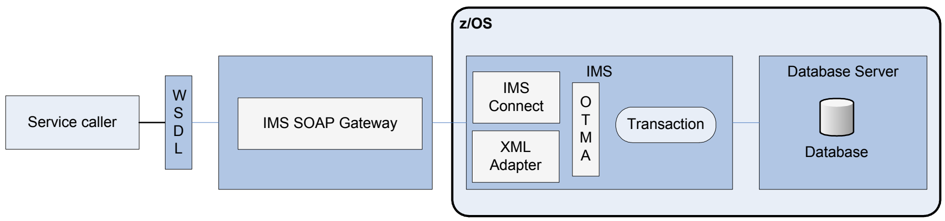


Figure 5-13 Overview of the IMS SOAP Gateway

Solution technique implementation

IMS SOAP Gateway fits the "Adapter-provided service interface" pattern, since it translates between a proprietary protocol and an SOA compliant protocol.

IMS SOAP Gateway consists of two main components:

- *IMS SOAP Gateway server*, which is the node that provides the SOAP service and the WSDL interface.
- *IMS SOAP Gateway deployment utility*, which is a utility that enables you to set up properties and create runtime code for the IMS SOAP Gateway.

Process overview

1. Create a Web Service Description Language (WSDL) file for the IMS application.
2. Deploy the Web service interface to IMS SOAP Gateway and define the connection and correlation information by using the deployment utility.
3. After you deploy the WSDL file, the IMS application is available as a Web service.
4. You can create your desired client application to send SOAP messages to your IMS application through IMS SOAP Gateway.

Tools used to create the WSDL for the IMS SOAP Gateway

IBM WebSphere Developer for zSeries is an application development tool that helps with the development of traditional mainframe applications. It helps you to easily generate the artifacts needed to transform your IMS application into a Web service to be used with the IMS SOAP Gateway runtime.

By simply taking a COBOL copybook for your IMS application that describes the input and output message format, it generates the following Web service artifacts:

- Web Services Description Language (WSDL) file, which provides a Web service interface of the IMS application so that the client can communicate with the Web service.
- COBOL converters and driver file, which help you to transform the XML message from the client into COBOL bytes for the IMS application and then back to XML.
- Correlator file, which contains information that enables IMS SOAP Gateway to set IMS properties and call the IMS application.

Example of the flow of a message using IMS SOAP Gateway

We now discuss the flow of a request, as shown in Figure 5-14 on page 156.

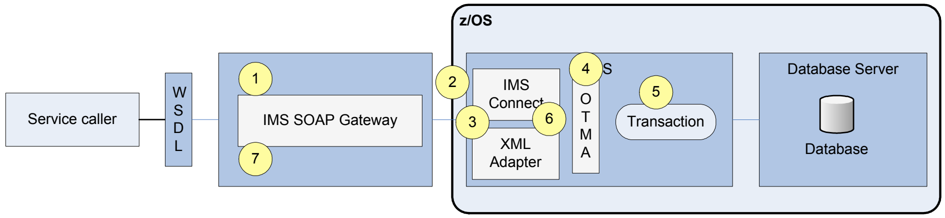


Figure 5-14 Walkthrough of the IMS SOAP Gateway example

1. The IMS SOAP Gateway server receives the SOAP message from the client application. It processes the SOAP header (XML) and retrieves the appropriate correlation and connection information for the input request.
2. The IMS SOAP Gateway sends the input XML data to IMS Connect using TCP/IP after adding the appropriate IMS Connect header.
3. IMS Connect calls the XML Adapter which in turn calls the XML Converter to perform the XML to IMS application format transformation.
4. It then calls the transaction using OTMA. From this point on, the processing is the same as a normal transaction flow.
5. The transaction gets executed and the output is queued.
6. The output message from IMS is then transferred back to IMS Connect, which calls the XML Adapter in order to perform the transformation of the IMS application format to XML. IMS Connect sends the output XML message back to IMS SOAP Gateway using TCP/IP.
7. IMS SOAP Gateway server wraps a SOAP header on the output message and sends it back to the client.

A few words about XML transformation

In the scenario, we have assumed that all XML transformation is done by the XML Adapter in IMS.

If you instead choose to handle the XML transformation in your application without utilizing the IMS Connect XML Adapter, you obviously do not need to invoke the XML Adapter.

In this case, the incoming XML message is sent directly to IMS Connect and then to the IMS application and the same is true in reverse. The IMS application creates an XML output message which is sent to IMS Connect, IMS SOAP Gateway and finally to the Web service client.

For detailed information

- Redbook: IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity - http://www.redbooks.ibm.com/abstracts/sg246794.html
- Product documentation: IMS SOAP Gateway http://www-306.ibm.com/software/data/ims/soap/index.html
- “WebSphere Developer for zSeries” on page 116

Variation 6: Using the WebSphere MQ to IMS Bridge

The WebSphere MQ IMS Bridge provides MQ connectivity to IMS backend transactions. A client can therefore be any kind of client that can put messages on an MQ queue.

In this scenario, the client is a J2EE application server, already running on z/OS. Logic is implemented in EJBs. Based on these EJBs there are service definitions published using WSDL and accessible from different channels, e.g. through SOAP. The EJBs use the JMS Provider for WebSphere MQ to access queues in a WebSphere MQ queue manager. Messages are forwarded by WebSphere MQ to the bridge, based on the queue settings in the WebSphere MQ queue manager.

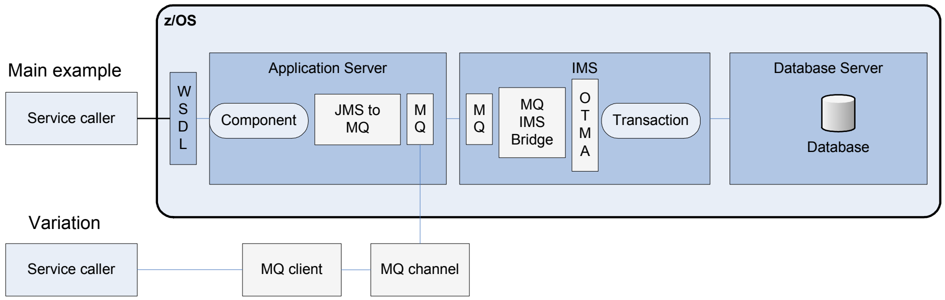


Figure 5-15 Overview of the WebSphere MQ IMS Bridge

Example of the flow of a message

1. The application server receives the SOAP message from the client application. It processes the SOAP header and calls the EJB that matches the input request.
2. The EJB uses JMS calls to obtain a connection to WebSphere MQ, and builds the required structure(s) for the call. Then it executes the call.
3. WebSphere MQ receives the message from the application server, and forwards it to the receiving queue manager. The WebSphere MQ IMS Bridge retrieves the message, and prepares the message in the OTMA format.

4. OTMA gets the message, and passes it on to the receiving transaction.
5. Response from the transaction is passed to OTMA, and MQ IMS Bridge.
6. A message is returned to the reply queue, and is fetched again by the EJB.
7. The EJB processes the reply and returns a reply to the service caller.

> **Attention:** Asynchronous retrieval of the reply message in the Web application server requires the usage of “Message-Driven Beans” and the configuration of so-called Listener Ports (JMS 1.0) or Activation Specifications (JMS 1.1) in the application server.

Solution technique implementation

The WebSphere MQ IMS Bridge example is an implementation of the “Adapter-provided service interface” pattern. The J2EE application running in the Web application server (WebSphere Application Server or other compatible server), combined with WebSphere MQ, provide an “adapter” that is external to IMS. The service interface is located in the Web application server.

Process overview

1. Configure WebSphere MQ, including a queue manager and queues
2. Install and configure the WebSphere MQ IMS Bridge
3. Define the WebSphere MQ queue and its queues to the application server.
4. OTMA (Open Transaction Manager Access) is already part of the IMS installation.
5. Develop the (Web service) application running in the application server using the JMS APIs. Eventually, develop MDBs for the reply messages.
6. Deploy the application to the application server.

For detailed information

- Redbook: IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity - `http://www.redbooks.ibm.com/abstracts/sg246794.html`
- Product documentation: WMQ IMS Bridge `http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqsat.doc/csq826y.htm`
- WebSphere Integration Developer `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp`

Using the Adapt approach with a broker/ESB

All of the technology variations in 5.2, "SOA implementation scenarios for 3270 application" on page 135 address the problem of enabling an existing 3270 transaction as a service and accessing it directly.

As an organization moves forward in the maturity of their SOA implementation, it will probably employ a message broker or Enterprise Service Bus as a mediation layer between the service requesters and service providers. 2.4.3, "The Enterprise Service Bus (ESB)" on page 19 discusses the basic functions of the ESB.

One of the major functions of an ESB is to translate protocols between the requester and provider. Most of the service enablement technologies used with the Improve and Adapt transition approaches can be used in conjunction with an ESB. The ESB becomes the service interface for all of the services under its control. This scenario implements the *brokered/mediated service interface* pattern. The ESB provides the service interface and leverages the appropriate service enablement technology to invoke the called service.

Figure 5-16 on page 160 illustrates the relationship between the ESB and the service enablement options. The service requester calls a service, the broker intercepts that call, routes the message to the proper endpoint service provider, and in the process converts the transport protocol to the one supported by the service provider. Optionally, the ESB will transform the message content into a format acceptable to the service provider and/or service requester.

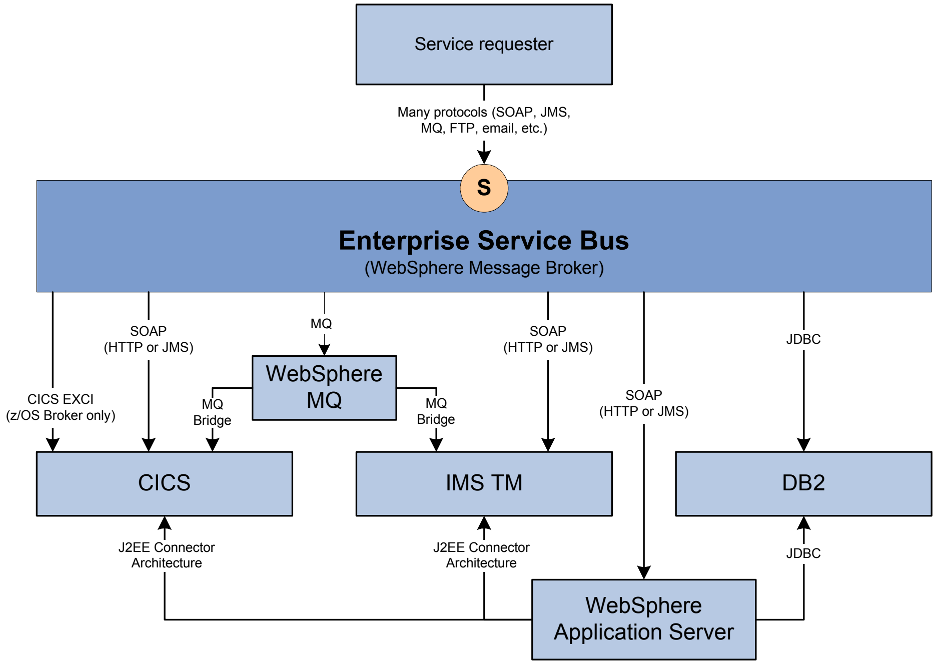


Figure 5-16 Overview of a brokered/mediated service interface architecture

For more information on ESB and service integration, refer to Chapter 6, “Towards service integration and process integration” on page 219;

5.2.3 Using the Innovate transition approach

Sometimes, an organization working on service enablement of a 3270 transaction may find that the code is so messy and unworkable that it requires a complete rewrite. This can be a very costly and involved process, and it negates some of the cost savings that are provided by reuse of assets. If appropriate tools are used to harvest and reuse business rules and existing business logic, savings can still be realized, and the resulting code is better structured for reuse and may provide better performance or reliability as a by-product of the effort.

The Innovate transition approach can allow the organization to do a better job of structuring the services with the appropriate level of granularity. In 4.1, “Methodologies for analyzing the business and application environment” on page 96, the concepts of top-down versus bottom-up SOA implementation and modeling approaches such as SOMA was covered. The use of these techniques becomes more applicable in a situation where code is being re-written, since the service granularity and appropriate levels of modularity can be implemented more easily in re-written code than when using programs that do not necessarily match the requirements identified in the top-down analysis using the modeling tools.

A key question in the innovation effort is: "*is the 3270 interface still needed*?" Many organizations are restructuring their applications to be accessed by Web interfaces, but there is still a requirement for 3270 access. This may be a matter of "dual use" by users who are accustomed to the 3270 interface and do not wish to change, and by new users and/or those who prefer a browser interface. Not all applications lend themselves to a browser UI. In some cases, "hands on keyboard" is a more efficient way to enter data than point-and-click.

Other decisions must be made when undertaking an Innovate project. For example:

- What programming language should be used? This decision is driven by a number of factors, but in an Innovate effort, one of the key factors is the amount of existing business logic that can be extracted from the existing programs.
- What transaction manager should be used? For example, a program may be redeveloped using Java, but that does not mean the application cannot continue to run on IMS or CICS, which both support transactions coded in Java.
- Does the re-engineered application conform to the proper level of granularity and provide the approrpriate service interface?

The other architecture decision questions must be considered also, including Quality-Of-Service requirements, skills, funding, etc. Architects should make sure they employ a methodology for architectural decisions when reviewing these items.

The logical architecture for an Innovate-based transition approach to service enabling a 3270 transaction is shown in Figure 5-17 on page 162.

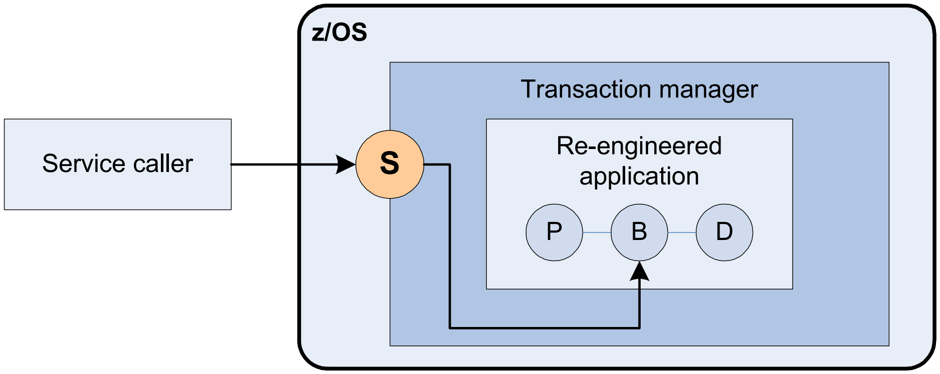


Figure 5-17 Logical architecture for the Innovate transition approach

Solution techniques and "Innovate"

When using an Innovate approach to service enablement, the solution technique used is the *Redeveloped service interface* pattern. However, using this technique also results in an application that takes on the appearance of an application that is service-enabled using the *Native service interface* pattern, since the service interface is ideally, though not necessarily, natively provided by the transaction manager or database server for the re-developed programs.

5.3 SOA implementation scenarios for multichannel

As described in 3.2, "Starting scenario - multichannel" on page 62, the multichannel architecture may already consist of reusable components, and to some degree also reusable services.

There are many variations of multichannel applications possible and we discuss a few very common ones in this chapter. This is not an exclusive list. Common to all variations is that there is a middle-tier implemented between the channel and the back-end, using components based on Java or J2EE. The scenarios discussed in this chapter are listed in Table 5-2 on page 163.

In the variations in this chapter, we will only deal with changes in the J2EE application server layer, and not with changes in the core backend functions. For changes in the core backend, please have a look at 5.2, "SOA implementation scenarios for 3270 application" on page 135. The multichannel and 3270 application scenarios can be combined in order to provide a more complete picture of the transition process.

Table 5-2 Summary of multichannel variations

Transition approach	Variation	Described in:
Improve	Variation 1: service enable an existing Java servlet / JSP-based Web application.	“Variation 1: Service-enabled Web application” on page 164
Improve	Variation 2: service enable an existing J2EE application	“Variation 2: Service-enabled J2EE application” on page 166
Improve	Variation 3: service enable an existing JMS-based application	“Variation 3: Service-enabled J2EE application using JMS / Message Driven Beans” on page 167
Adapt	Variation 1: service integration using WebSphere ESB	“Variation 1: Using the Adapt approach using WebSphere ESB as broker” on page 171
Adapt	Variation 2: service integration using WebSphere Message Broker	“Variation 2: Using the Adapt approach using WebSphere Message Broker as broker” on page 173
Innovate	Variation 1: service enablement and integration of a client/server application	“Variation 1: SOA enablement of a fat client (client/server) application” on page 177
Innovate	Variation 2: Using WebSphere Portal to provide interaction services	“Variation 2: Using Portal” on page 180

5.3.1 Using the Improve transition approach

Just to refresh what we mean with the “Improve transition approach”. We have defined it as an approach, which exposes application functions as a service. It usually involves a piece of new technology which adds a standardized interface to be used by a calling application.

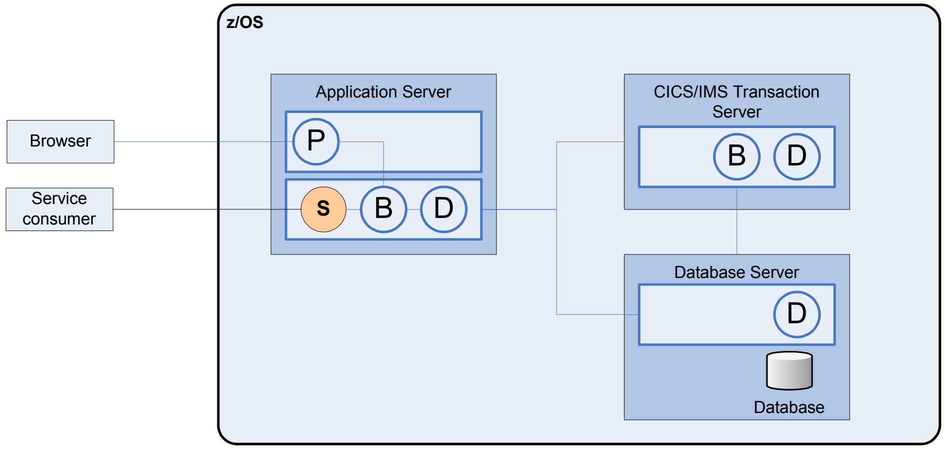


Figure 5-18 Logical overview of the Multichannel Improve transition approach

Solution techniques for "Improve"

The use of the Improve transition approach usually leads to the adoption of the "adapter-provided service interface" pattern. In the multichannel scenario, we might also see some pieces of the "native service interface pattern", since there might be slight modifications to the components in the application server layer.

The core functions in the backend transaction server remain untouched.

Variation 1: Service-enabled Web application

Many existing Web applications are based on *Java Server Pages (JSP)* or servlet technology. In the early days when there was no EJB support, the application logic was built using standard Java Beans. Those applications have had the primary objective of adding a better user interface to an existing (legacy) application. We also called this approach "Web enablement". There was not much focus on separation of concerns and reusability.

For this scenario, we assume that the backend functions in CICS and IMS will remain the same, and called via the same interface as before. The changes are in the Java layer and have the objective to make encapsulated business functions (which are usually implemented in the Web applications by means of Java Beans) accessible as Web services.

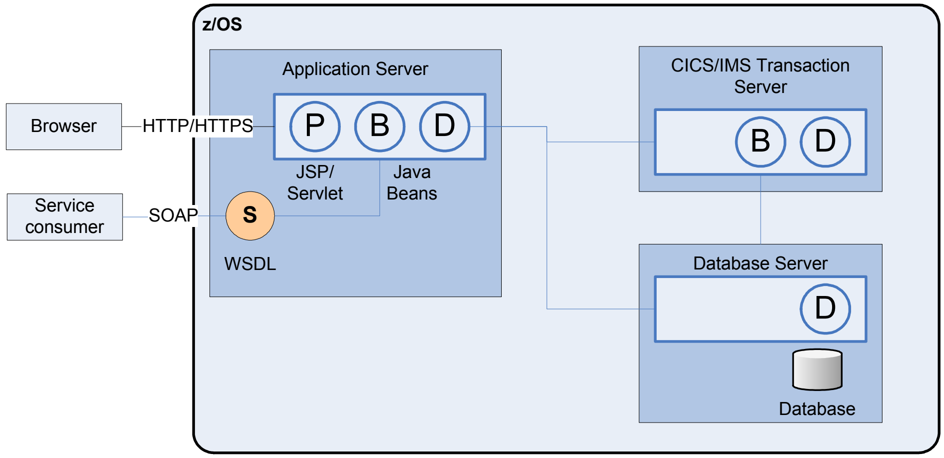


Figure 5-19 Target architecture of the "service-enabled Web application" variation

When transformed, this architecture may be viewed as the starting point for "Variation 2: Service-enabled J2EE application" on page 166.

Solution technique implementation

The solution technique in this scenario could best be described as a "native service interface" pattern technique, because the service interface will be implemented in the same runtime component as where the service itself is implemented.

Process overview

Web service-enabling a Web application involves the following steps:

1. The developer uses the Rational Application Developer or equivalent tool to create a service interface from the servlet or Java Bean, using the provided wizards.
2. Eventually, a Java service proxy can be created to be included in a Java service consumer.
3. The WSDL created in the first step can be used by the developer of any service consumer and the Java service proxy created in the second step can be used by a Java service consumer.

For detailed information

- Redbook: Rational Application Developer V6 Programming Guide - `http://www.redbooks.ibm.com/abstracts/sg246449.html`

Variation 2: Service-enabled J2EE application

The starting point for this variation is that the application architecture is based on J2EE, including EJBs. Compared to the Java servlet enablement variation, EJB technology brings added functionality for transactions and security.

This variation illustrates how the EJB components can be enabled for reuse, and called by other channels. In this case, via the SOAP protocol.

For this scenario, we assume that the backend functions in CICS and IMS will remain the same, and called via the same interface as before. The changes we apply are in the J2EE layer, implemented in the application server.

This scenario, like the "Variation 1: Service-enabled Web application" on page 164, shows that it is the J2EE application server that provides the technology enhancements to add support for new protocols and channels.

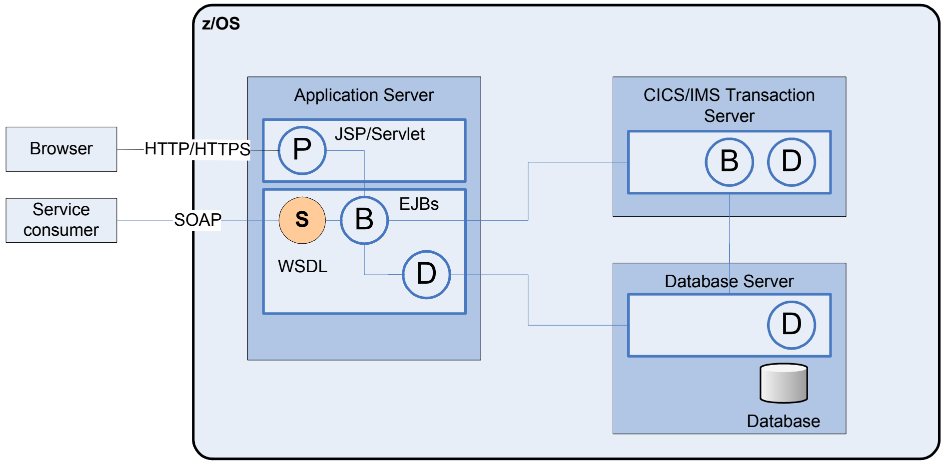


Figure 5-20 Overview of the service-enabled J2EE application

Figure 5-20 shows the enablement of a WSDL interface to the existing EJBs. Once your component architecture is based on EJBs, it is not a major task to add support for additional standards, such as SOAP.

WSDL is generated by the Rational Application Developer tool, and the EJBs and WSDL is deployed to the WebSphere Application Server.

Solution technique implementation

The service-enabled J2EE application variation is an implementation of the "native service interface" pattern. The J2EE application is Web services-enabled using functionality in the J2EE application server (WebSphere Application Server or equivalent). The service interface is located inside the J2EE application server.

Process Overview

Service-enabling an EJB-based application involves the following steps:

1. The developer uses the Web Service wizards in Rational Application Developer to create WSDL from the stateless session EJB.
2. Eventually, a Java service proxy can be created to be included in a Java service consumer.
3. The WSDL created in the first step can be used by the developer of any service consumer and the Java service proxy created in the second step can be used by a Java service consumer.

For detailed information

- Redbook: Rational Application Developer V6 Programming Guide - `http://www.redbooks.ibm.com/abstracts/sg246449.html`

Variation 3: Service-enabled J2EE application using JMS / Message Driven Beans

The starting point is an application architecture that is based on EJBs and JMS as asynchronous communication protocol, opposed to the variation discussed in “Variation 2: Service-enabled J2EE application” on page 166 where HTTP or RMI-IIOP were the primary protocols used.

This variation illustrates how the EJB components can be reused better from other channels by service-enabling the MDB/EJB-based functions.

For this scenario, we assume as in the previous variations that the backend functions in CICS and IMS will remain the same, and called via the same interface as before. The changes we apply are in the J2EE layer, implemented in the application server.

This scenario shows that the runtime functionality for Message Driven Beans (MDB) will be provided by the J2EE Application Server.

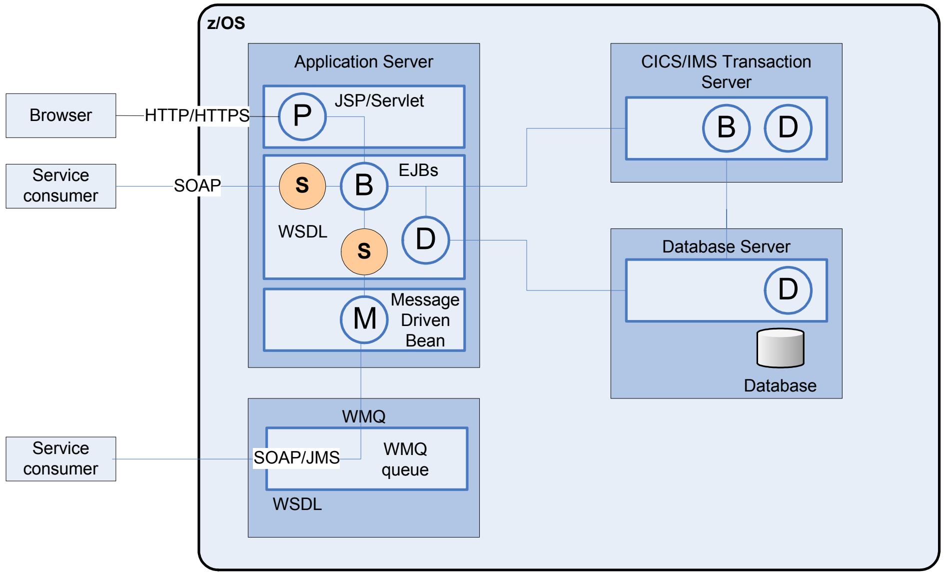


Figure 5-21 Overview of service-enabled J2EE application using JMS / Message Driven Beans

Solution technique implementation

The service-enabled J2EE application using JMS / Message Driven Beans variation is an implementation of the "native service interface" pattern. The J2EE application is Web services enabled using functionality in the J2EE application server (WebSphere Application Server or equivalent) combined with the usage of Message Driven Beans. The service interface is located inside the J2EE application server.

The wizards in Rational Application Developer can be can be used to create an EJB with a bean type of Message Driven Bean. The wizard creates appropriate methods for the type of bean.

Process Overview

1. The WebSphere MQ administrator creates any additional necessary queues and connections to be used by the new solution, if not already there.
2. The developer uses the Enterprise Bean wizard in Rational Application Developer to create the enterprise bean with the type of Message-Driven Bean. All necessary methods are automatically created.
3. One reason for creating a new Message-Driven Bean, is to separate the business logic from the interface. The Message-Driven Bean can easily call

the target EJB, so the MDB can be thin, and implement most logic in the *onMessage()* method.

4. The EJBs are configured for deployment, packaged into EAR files, and deployed in the WebSphere Application server.
5. Client code is created using the technology of choice. The client needs to be configured to use the appropriate queue name, which triggers the MDB.

For detailed information

- Techical documentation: Using message-driven beans in applications - `http://publib.boulder.ibm.com/infocenter/wasinfo/v5r1//topic/com.ibm.websphere.zseries.doc/info/zseries/ae/tmb_ep.html`

5.3.2 Using the Adapt transition approach

The ”Adapt” transition approach is more intrusive than “Improve”. It means that there will be more changes in the original code and functions, than in the previous examples.

There is a need for a more clear separation of presentation, business and data access logic.

There may also be a need to change the core functions in the back-end applications.

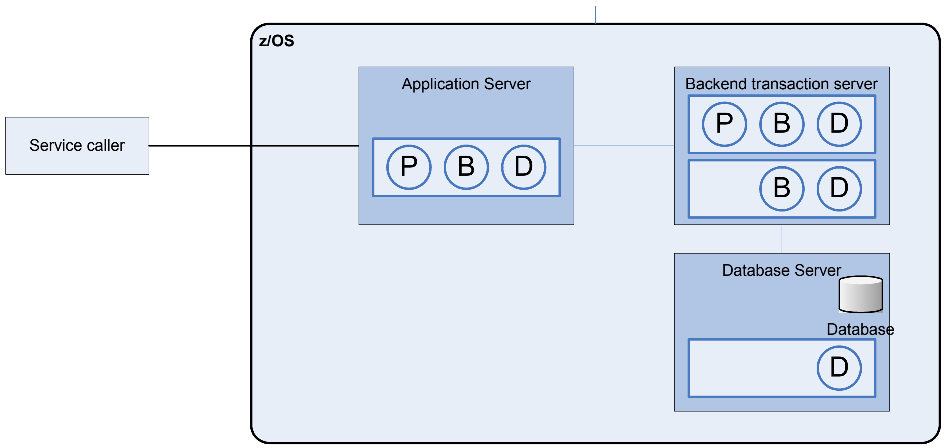


Figure 5-22 Overview of the Multichannel “Adapt” transition scenario

Solution techniques for "Adapt"

The "Adapt" variations for the multichannel SOA implementation scenarios will involve the "adapter-provided service interface" and "brokered/mediated service interface" patterns.

Using the Adapt approach with a broker/ESB

The starting point for this variation is that the application architecture is service enabled, but services are not being called over an ESB yet. Hence, any routing, mediation or event services logic (if this exists) is still manually implemented in the application. There may be different types of service bindings used, and the service consumers need to be able to support the bindings of the service provider.

For this scenario, we assume that the backend functions in CICS and IMS will remain the same, and called via the same interface as before, i.e. through one of the protocols supported by the application server. Usually, this is either a J2C connector or JMS. Typically, the service interface is in the application server.

This variation introduces a common bus between the service requesters and service providers. Instead of calling a service in the application server directly over HTTP or JMS, the call will now go logically through the ESB. This variation is illustrated in Figure 5-23.

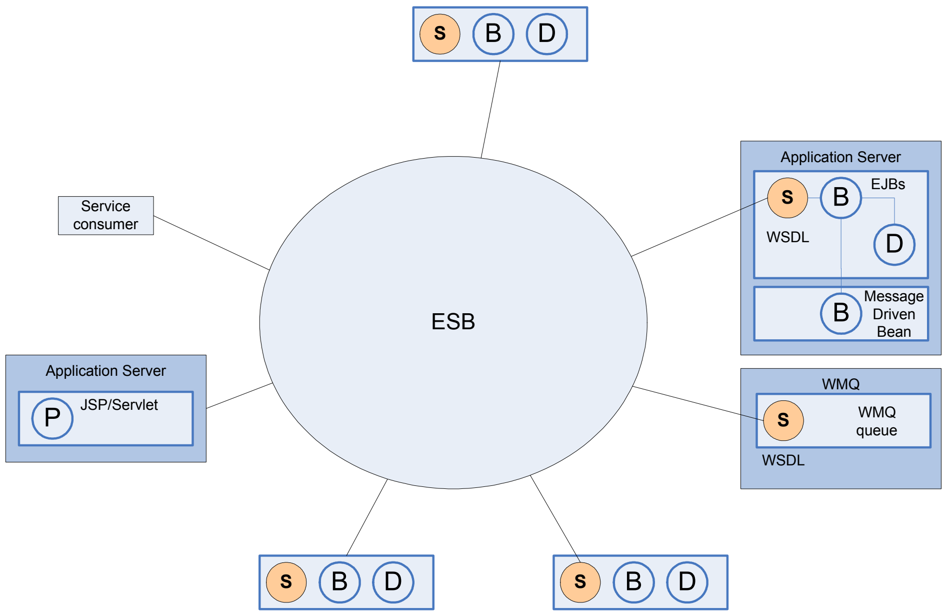


Figure 5-23 Overview of the "Adapt" transition approach with a broker/ESB

For more information about the ESB architecture see “What are the capabilities of an ESB?” on page 228.

Variation 1: Using the Adapt approach using WebSphere ESB as broker

In this variation we instantiate the generic ESB described previously with the product WebSphere ESB. From the description of the starting point we know that the applications are enabled as services, but the routing, mediation and event services logic is still manually implemented in the application. We also know that there may be different types of service bindings used, and the service consumers need to be able to support the bindings of the service provider.

An example, with an application containing mediation is shown in Figure 5-24.

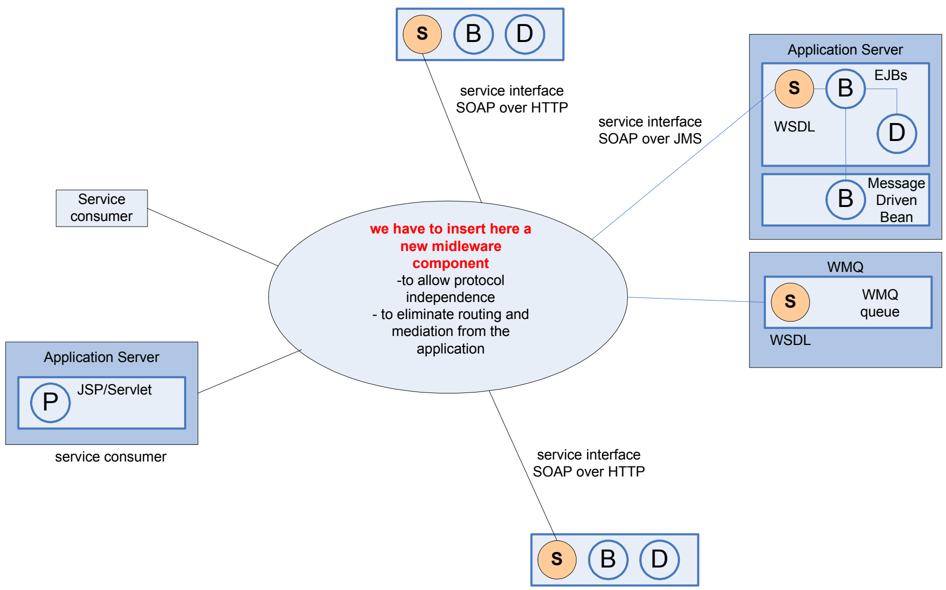


Figure 5-24 Example of the Adapt approach using WebSphere ESB

The first activity is to identify the application parts that implement routing, mediation and event processing and position these inside the WebSphere ESB. So there is some amount of analysis, decomposition and restructuring work necessary.

The second activity concerns the service bindings used: we will leave the service binding untouched, but eliminate the necessity for the service consumer to know that a particular service can be called only with a specific service binding. We are also creating, in the process of restructuring, new services (named S1). Let us take the following example: the service, implemented in the application server, can be called using SOAP over JMS. Since we don't want to put restrictions on the service consumer, we are using the flexibility provided by WebSphere ESB to mediate between the call of the service consumer and the service, translating (in ESB language called "mediating") as necessary.

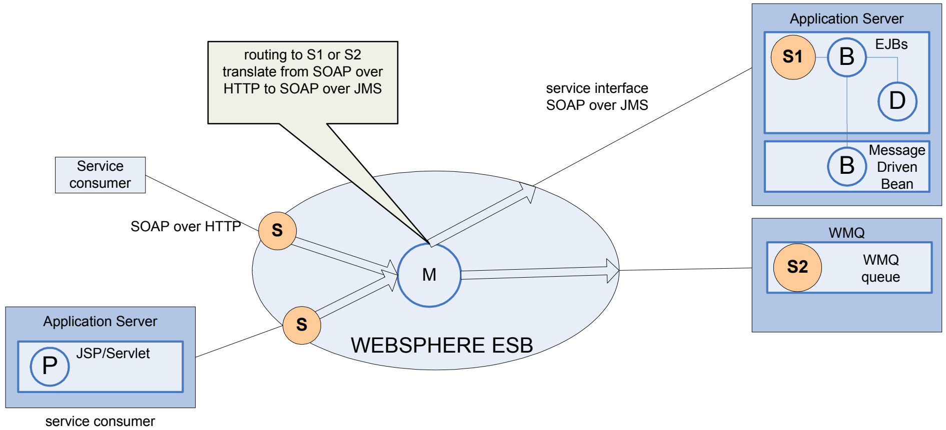


Figure 5-25 Implementation of routing and mediation for the Adapt approach with WebSphere ESB

Solution technique implementation

The solution technique in this scenario could best be described as a "brokered/mediated service interface" pattern, because the service interface accessed by the consumer will be implemented in the ESB.

The best scenario to implement WebSphere ESB is when the applications support the interaction pattern called Web Services. In the simplest case all services are co-located inside the WebSphere Application Server. The collocation produces improvements in performance (reducing path length, eliminate some network traffic).

Process Overview

1. The developer restructures the application by decomposing the routing/mediation parts, and creating eventually more service implementations.
2. The developer uses the wizards in WebSphere Integration Developer[2] to create the SCA components (with their imports and exports) that represent to the outer world the services. The initial WSDL is known from the application

[2] WebSphere Integration Developer (WID) is IBM's tool to develop SCA components and mediation for WebSphere ESB.

(we said that the prerequisite was that all applications are service enabled); the WSDL can be imported and used in the SCA. The SCA components will have the bindings used by the clients. Note here that the ESB (the central middleware) adapts to the bindings supported by the clients, and not vice versa.

3. The developer uses the wizards in WebSphere Integration Developer to create the mediations and routing components.
4. The WebSphere ESB administrator deploys the mediation and routing components in the WebSphere Application Server runtime (where the ESB itself runs).

For detailed information

Author Comment: References to do (ALK)

Variation 2: Using the Adapt approach using WebSphere Message Broker as broker

In this variation we instantiate the generic ESB described previously with the product WebSphere Message Broker (WMB). From the description of the starting point we know that the applications are enabled as services, but the routing, mediation and event services logic is still manually implemented in the application. We also know that there may be different types of service bindings used, and the service consumers need to be able to support the bindings of the service provider.

When we speak here about the applications enabled as services we do not mean Web services; there is a mixture of applications, some have standard (Web Services) service interfaces, some have non-standard service interfaces, accessed over different protocols and different interaction patterns. The situation is shown in Figure 5-26 on page 174.

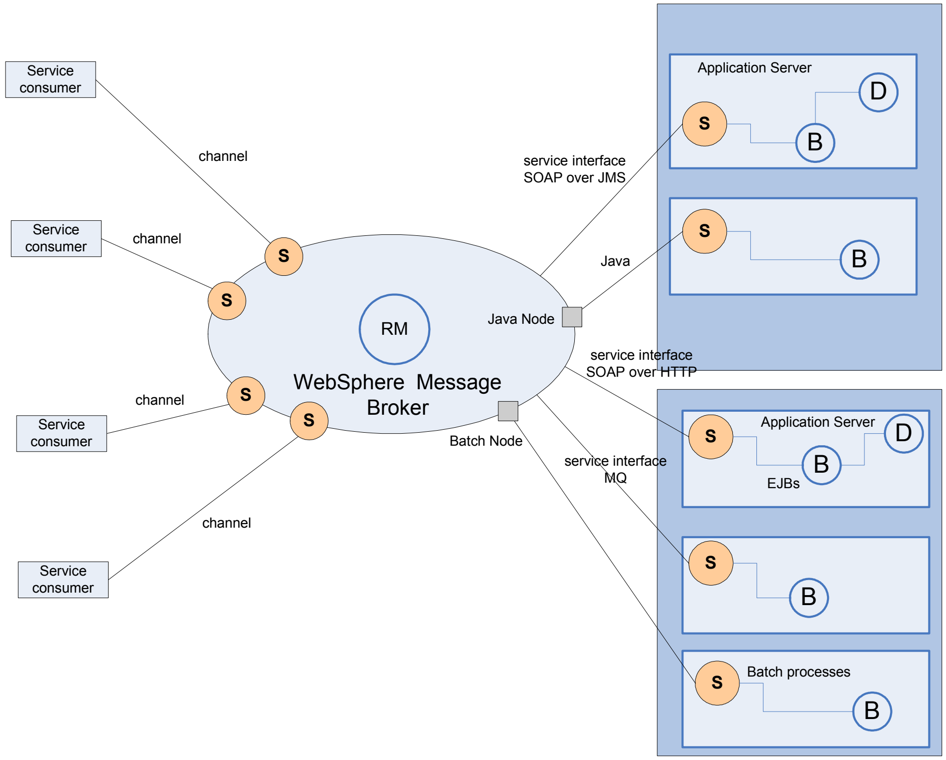


Figure 5-26 Logical design for the Adapt approach with WebSphere Message Broker

Again we position between the service consumer and this multitude of service providers (with a varied array of protocols and interface patterns) a piece of middleware to simplify the connectivity, centralize routing, mediate protocols and content. This piece of middleware is in this case the Websphere Message Broker. The words “different protocols and a multitude of protocols and interaction patterns” represent the “power” feature of the WebSphere Message Broker, so we know that from the point of view of connectivity we are on the right way for solving the problem. As known, WebSphere Message Broker makes available numerous message processing nodes, like Database node, Compute node (with the speciality Java) , Filter node, Batch node, MQ node. JMS node, WebServices nodes (HTTP(s) request and reply nodes), Aggregation node, XSLT node and many others. These nodes allow us to implement the necessary translations, routing and mediations.

Solution technique implementation

The solution technique in this scenario could best be described as a “brokered/mediated service interface pattern, because the service interface accessed by the consumer will be implemented in the WebSphere Message Broker.

Process Overview

1. The developer restructures the application by decomposing the routing/mediation parts, and creating eventually more service implementations. The bindings and interaction pattern of the service interfaces can be created based on the business requirements; WebSphere Message Broker will take care of the necessary translation between the service consumer and the service providers
2. The developer uses the WebSphere Message Broker Toolkit to create the message flows. The initial WSDL is known from the application (we said that the prerequisite was that all applications are service enabled); the WSDL can be created or imported, and then used in the WebServices nodes. WMB makes available for the WebServices nodes a WSDL generator and a WSDL import wizzard, with high flexibility for modelling SOA messages, WDSL types and interaction patterns.

   As an example, the message flow for defining a WebService might look like in the Figure 5-27 on page 176. The nodes Implementation, Implementation1 and Implementation2 are the places where the developer, using the nodes available, implements the processing, and parts of the processing is the call of existing services. We see that the developer has a high flexibility in implementing the Web service both in the area of connectivity (multiple transports) and interaction patterns, but also in the the access to back-end services (multiple node types).

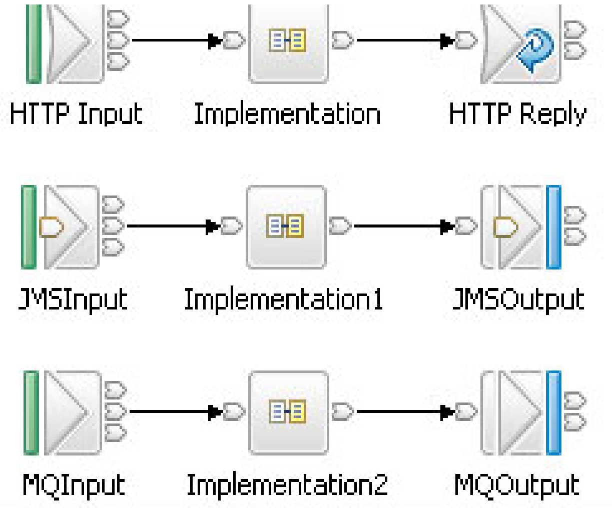


Figure 5-27 Example of implementation of a Web service in the WebSphere Message Broker

3. The WebSphere Mesage Broker administrator deploys the message flows to the runtime.

For detailed information

Author Comment: References to do (ALK)

5.3.3 Using the Innovate transition approach

The "Innovate" transition approach is even more intrusive than "Adapt". In this approach an existing multichannel application will need to be changed to meet a fairly high SOA maturity level. In any case this includes a full "separation of concerns" and a high reusability level of services.

At this point we would want to have the presentation logic in the various channels clearly separated from the rest of the application. Also, standards-based service interfaces should be in place and an ESB must be implemented for mediation, routing and event services.

It is likely that the backend functions will also need to be integrated using Web services standards and through an ESB. This, however, may lead to the implementation of one or several scenarios as discussed in 5.2, “SOA implementation scenarios for 3270 application” on page 135.

Variation 1: SOA enablement of a fat client (client/server) application

Simplified, a *fat client* application is an application with a combination of presentation and business logic implemented on the client device. It is not unusual that this presentation and business logic is very tightly integrated. In some cases, even data access logic is implemented on the client.

> **Note:** A client nowadays can be a variety of devices, such as a Windows workstation, PDA device, smartphone, ATM, kiosk or any other device that can communicate with a server.

The communication protocol with the server can be any protocol, HTTP, TCP/IP, MQ, RMI/IIOP and even SNA-based communication is not impossible. This is illustrated in Figure 5-28

Most commonly used languages for a fat client are Java, C, C++ or Visual Basic®.

The backend server could provide a service interface based on for example CORBA, or it could be as simple as providing connectivity to data.

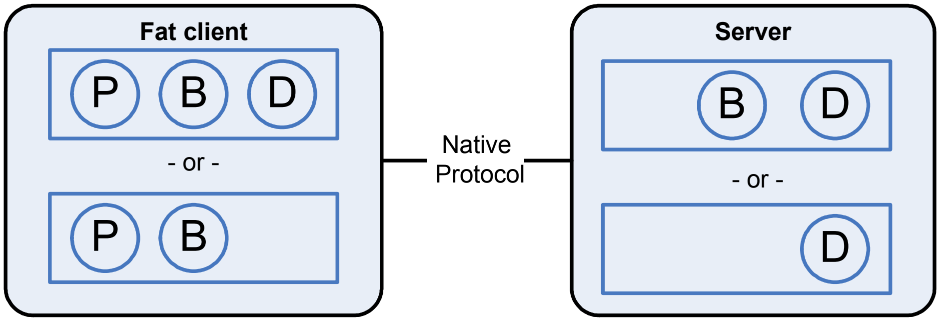


Figure 5-28 Logical overview of the Fat client scenario

The fat client starting point

The starting point for the scenario is the example in Figure 5-29 on page 178 that is labeled “Fat client”. The examples labeled “Rich client” and “Thin client” are evolutionary steps towards an SOA, but could also be the starting points.

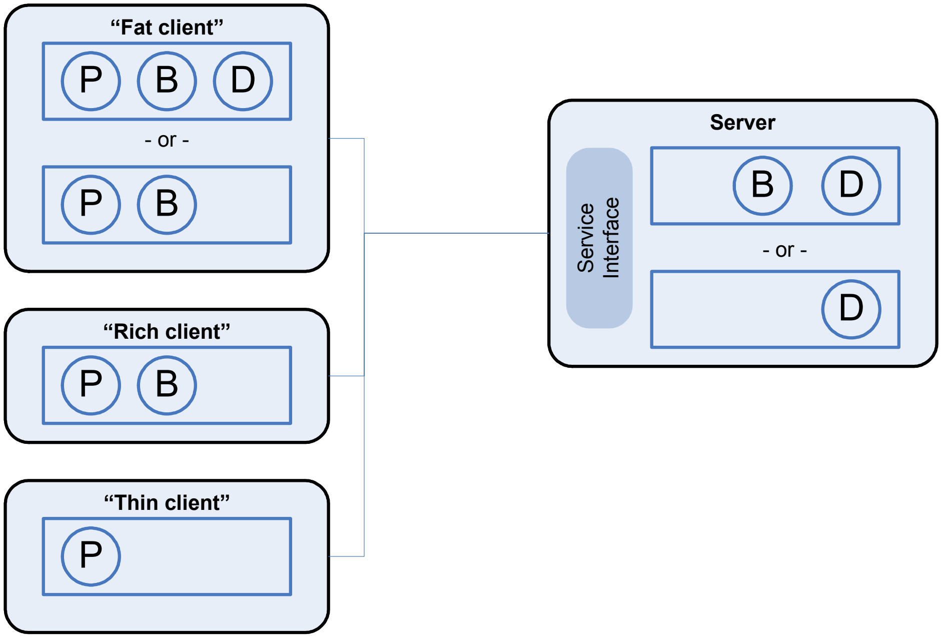


Figure 5-29 Possible evolution towards the innovate approach of the migration of a fat client

Solution technique implementation

The solution technique in this scenario could best be described as a mix of "Adapter-provided service interface" and "native service interface" patterns.

- Adapter, because there will be a standard service interface in front of the core backend functions. There is though a possibility that the core functions need to be changed to fit into the new architecture.
  The service interface is located in the core transaction server.
- Native, in case backend services are called directly using native SOAP calls.

Process overview

To apply the Adapt transition approach to this example, the following needs to be considered.

- On the server tier:
  a. Can an adapter-provided service interface pattern be applied?
     - What can be done to standardise the native protocol between the client and the server?
     - Is the protocol based on proprietary communication, and how can that be opened up?

   - What platform is the server running on. If it's CICS or IMS, maybe the techniques that are described in the 3270 transition scenario chapter could be applied.

   b. Can a native service interface pattern be applied?

   - What about the application architecture? Are the services structured in a way so they can be packaged and used by another protocol than they originally are built for.

   c. Can the a broker/mediated service interface pattern be applied?

   - Could messaging, such as WebSphere MQ be used?

- On the client tier:

   a. Can the presentation logic be separated from the business logic?

   - Is the application layered in a 3-tier model?
   - How about data access? Is it clearly separated in the code?
   - Are new requirements on the client platform introduced? User interaction, security, role based interactions etc.?

   b. Can the business logic from the "fat client" be reused and packaged as server side components? Or is it necessary to rewrite using another technology maybe?

One possible way to go is to migrate to a "Rich client" (illustrated in Figure 5-29 on page 178), with most business logic deplyed on the server. Maybe callable via SOAP, or as EJBs through RMI/IIOP.

Some client applications require the speed and input validation functions provided by the fat/rich client, so maybe that is the target? Other applications might provide the right functionality running in a Portal, with role-based interactions.

There are many considerations in this scenario, and not one single answer to these questions. It is very much about deciding on the strategic and tactical steps for the migration.

The rich client starting point

If the application already is at the level of a *rich client*, it is probably a lot easier to migrate to an SOA and a thin client based solution. The reason for this is that there is a more clear distinction between presentation, business and data logic.

The platform of the rich client provides access to an open standards based protocol for communication with the services on the backend server.

For detailed information

- Redbook: Patterns: SOA Client - Access Integration Solutions - `http://www.redbooks.ibm.com/abstracts/sg246775.html`

Variation 2: Using Portal

A variation of the innovate transition approach consists in the possibility of redeveloping the presentation part of the application, positioning it on a Portal and use standard interfaces to access the services. This solution opens in fact an additional channel (we are in the multi-channel case) for the application. There are many sub-variations in the Portal use.

The Portal solution can be implemented only if we have a clear separation of the presentation from the business part of the application.

The portal solution capability for composing applications takes advantage of the functions available in the WebSphere Portal Server; these are placed into the category “on the glass” and “under the covers”. In the category “on the glass” we have:

- Portlet technology
- Web services technology
- Web clipping technology
- Java Server Faces (JSF) technology

In the category “under the covers” we have:

- Adapters technology
- Broker technology (in its two possible instances WebSphere ESB and WebSphere Message Broker)
- Web services technology
- Service Data Objects (SDO) technology
- Information services technology (like XQuery, SQL)

The simplest implementation of a Portal integrates access to services using the above mentioned technologies. The existing services can be either Web services located in application servers, or information services, or even “green screen” applications which can be accesed by the HATS portlet. Figure 5-30 on page 181 shows the logical architecture of such a Portal solution. This shows the portlets accessing directly services (through their Web services capability, for example) or using the mediator implemented by the ESB.

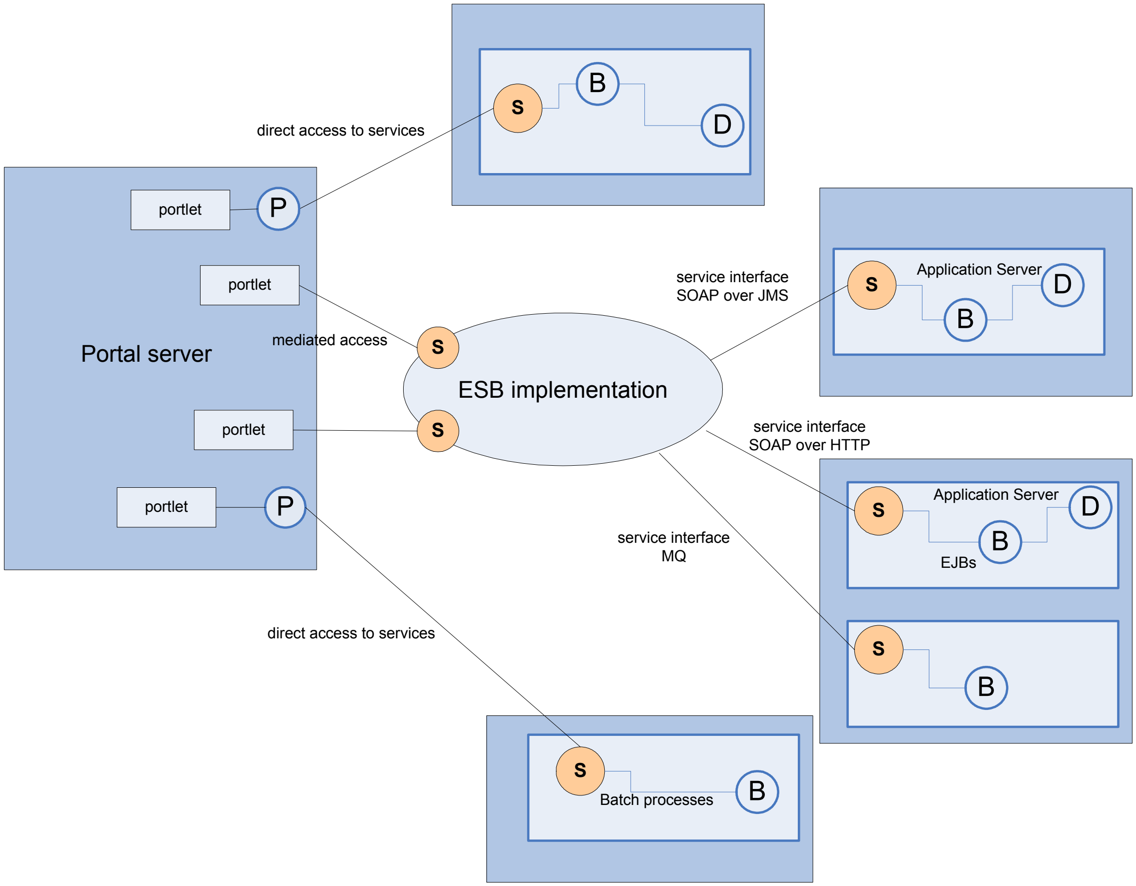


Figure 5-30 Logical architecture for a simple portal implementation

A more sophisticated portal solution (and the one that allows creating composite applications) is the one in which the portlets themselves are part of new business logic. In this case we use the capability of the portlets to interchange data and messages, the BPEL-based choreography engine and human-task list, and the possibility of specifying business rules for composing applications.

The portlets are not only availabe to exchange data and messages, but they can be automatically triggered when changes occur in other portlets. The interface between WebSphere Portal and WebSphere Process Server contributes to complete the solution.

The logical architecture of this implementation is shown Figure 5-31 on page 182.

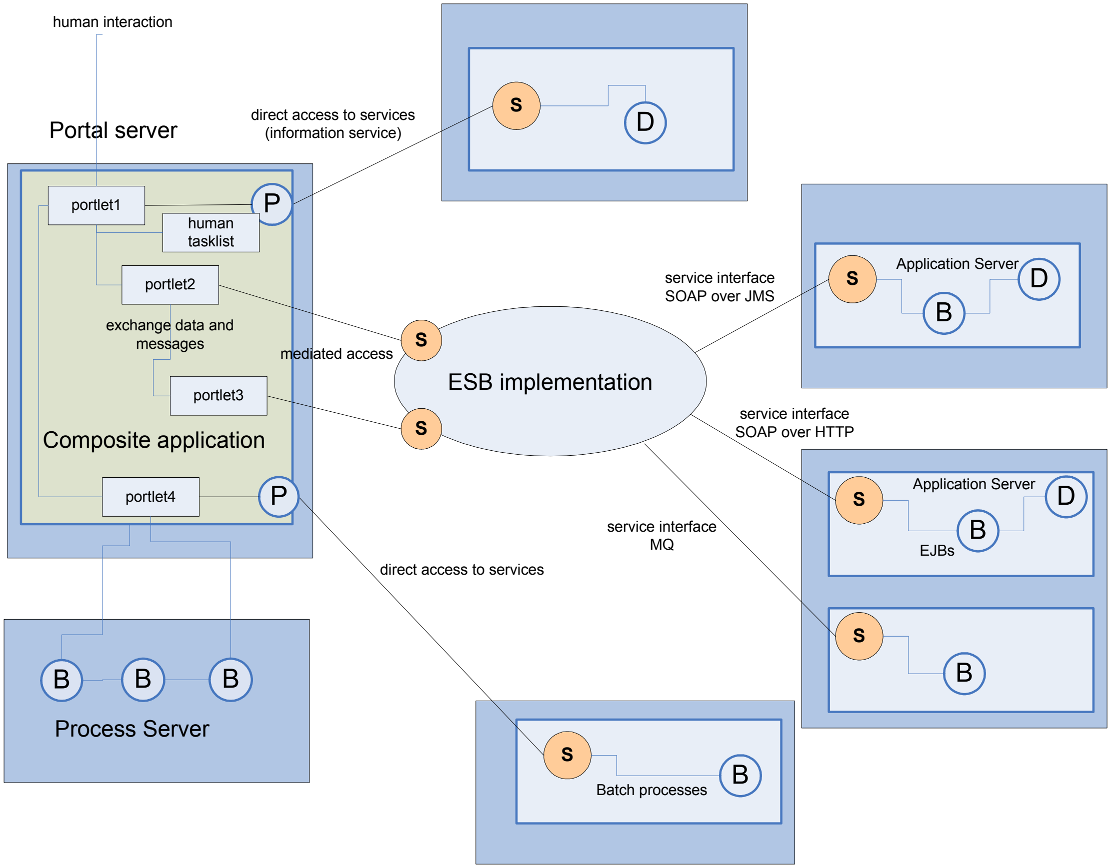


Figure 5-31 Logical architecture for a composite portlet application

Solution technique implementation

The solution technique in this scenario could best be described as a mix of the "adapter-provided service interface" pattern and the "native service interface" pattern.

- Adapter, because there will be a standard service interface in front of the core backend functions. There is though a possibility that the core functions need to be changed to fit into the new architecture.
- Native, in case backend services are called directly using native SOAP calls.

Process overview

1. The portal designer uses the tooling to produce the page layout and the portlet design.

2. The administrator deploys the produced artifacts in the runtime.

5.4 SOA implementation scenarios for batch

Service enabling batch applications involves establishing new interfaces to initiate or invoke batch components as services.

A batch job may be considered as one probably large component. Or, it may be considered consisting of several minor components exposed by steps or perhaps even more granulated by individual programs or subroutines inside one step.

The flow of batch jobs or the flow of steps within a batch job can be considered a workflow, a composite service consisting of several atomic services implemented by the individual jobs or job steps.

Figure 5-32 shows how batch is invoked.

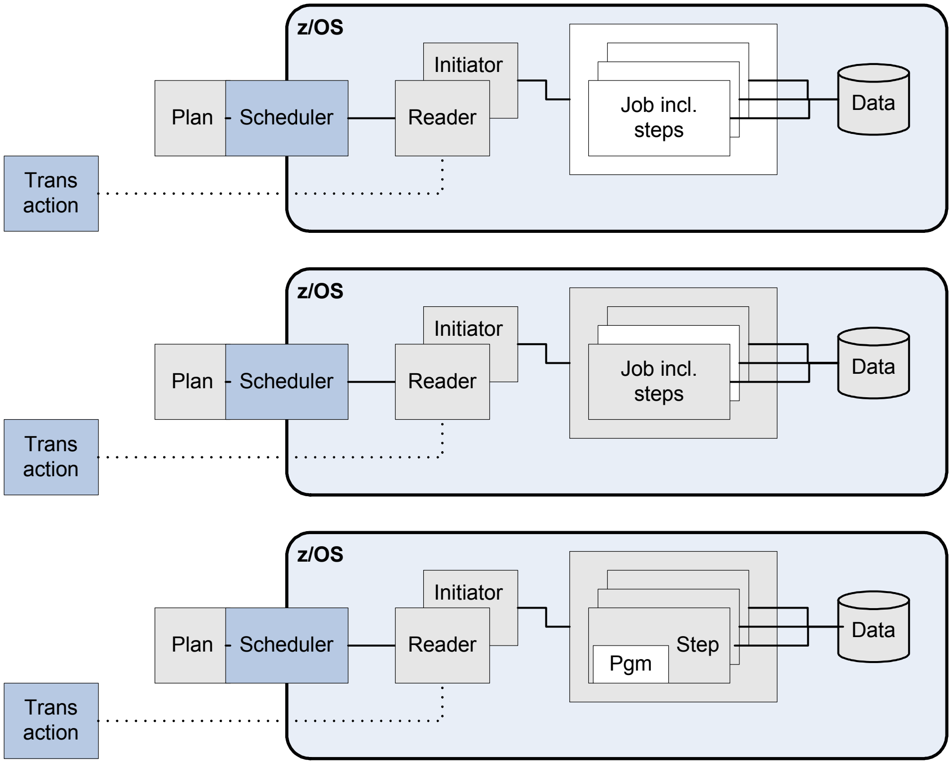


Figure 5-32 Overview of batch invocation

Based on a plan the scheduler invokes the batch job involving the reader and the initiators.

Batch jobs may also be invoked from TSO or a transaction server by writing directly to the internal reader.

With WebSphere MQ (WMQ), batch jobs can be triggered by messages arriving on a queue.

The batch job consists of one or more steps including one or more programs. It receives input in the form of parameters, sysin, datasets, messages or database rows. And it delivers output in the form of condition codes, sysout, sysprint, datasets, messages and database rows.

Furthermore, the batch job can be considered a workflow consisting of one or more automatically invoked activities.

In this chapter we will look at batch both from a consumer and provider perspective.

Table 5-3 shows a summary of the variations of the SOA implementation scenarios for batch jobs that will be discussed in this chapter. We will see that service-enabling of batch clearly still provides less options than we have seen in the previously discussed OLTP scenarios.

In addition to the traditional approaches, a very pragmatic "leave be" approach is described. It is not a real transition approach as no existing code is reused, but it may turn out to be a very popular solution in many organizations.

Table 5-3 Summary of batch variations

Transition approach	Variation	Discussed in
Improve	Making a job or job step reusable as a service without changing any code	5.4.2, "Using the improve transition approach" on page 187
Adapt	Variation 1: Improve the granularity of the services in batch and integrate them	5.4.3, "Using the adapt transition approach with batch as the service provider" on page 189
Adapt	Variation 2: Call reusable Web services from batch programs, eventually through an ESB	5.4.4, "Using the adapt transition approach with batch as the service caller" on page 191

Transition approach	Variation	Discussed in
Innovate	Fully reuse functions inside batch from anywhere	5.4.5, “Using the innovate transition approach” on page 192
“Leave be”	Special case: Leave batch be and develop new SOA	5.4.6, “A practical, mixed approach” on page 196

5.4.1 Multi entity handling in batch services

Before moving to the individual transition approaches a batch characteristic should be taken into account. No matter which transition approach you may choose, you are probably facing the challenge that large amounts of data and data entities are processed in each program invocation.

As shown in Figure 5-33, a batch program needs a service to process several records or messages.

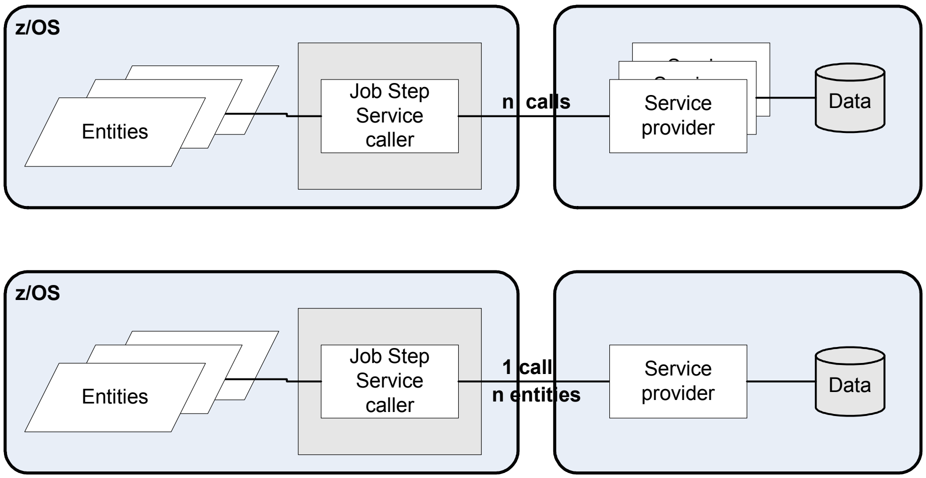


Figure 5-33 Multi entity handling in batch

Logically, it can be done in either of two ways:

1. The service caller invokes the service provider once for each entity to be processed.
2. The service caller invokes the service provider once and expects the provider to process all entities in the same instance.

Service call per entity

This is a more pure SOA approach and data can easily be exchanged as parameters, for instance as XML in a SOAP message.

A natural consequence is a considerable resource consumption due to a corresponding number of external calls including housekeeping.

Service call once for all entities

This may be a less pure SOA approach, but a more pragmatic one. With this approach a higher lever of performance can be achieved. The overhead of invoking a service many times individually can be avoided. It works like a kind of caching.

But how can the large amount of data entities be exchanged? WMQ gives a good answer to that. Messages can be exchanged between reusable services in a platform and location independent manner. With WebSphere Message Broker (WMB) it may even be possible to achieve protocol transparency.

In case of WMQ, “pipelining” can be exploited to achieve a level of parallelism. The service provider can start processing the messages while the caller is still producing messages. This naturally depends on the demands for transaction support and units of work.

With WMQ, even further parallelism can be achieved by asynchronously triggering more services.

The multi-entity call approach also provides means for accumulations and other cross entity information and calculations usually seen in batch.

Call approach determination

In order to determine which approach is the most appropriate, some factors should be considered:

- What is the intended use of the service provider?
- Can it be determined whether it is expected to be invoked with one entity at a time or with large amounts of entities? The transition approach selection may be dependent on that.
- Can the service be deployed in two implementations with controlled redundancy?
- With parallel processing of multiple entities provided by WMQ, batch job duration can be reduced thereby narrowing the batch window.

Under any circumstance, it is likely - and recommended - that changing the existing well running batch jobs could and should be questioned.

5.4.2 Using the improve transition approach

The logical architecture for an "Improve" solution is shown in Figure 5-34.

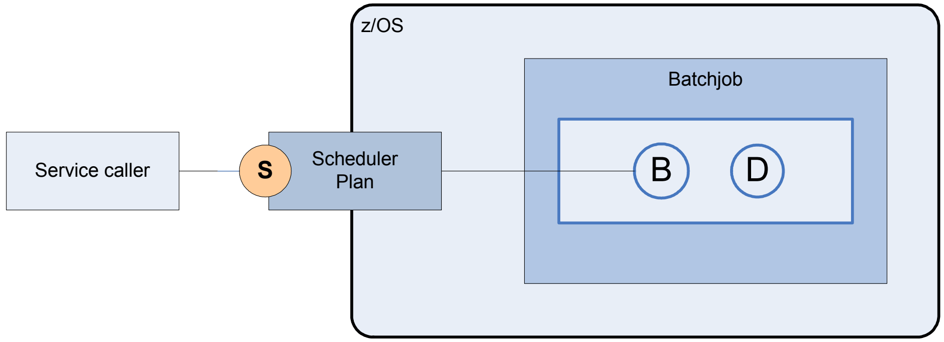


Figure 5-34 Logical architecture for the "Improve" batch variations

Considering the batch job as a service, it may be invoked in its complete state. It can be invoked by the scheduler as a main service or it can be invoked from an external service caller as a Web service[3].

Solution technique implementation

The original batch job may be split up in several batch jobs in order to make each of these service callable in the desired granularity. Or it may be considered a service including all the original steps. In the improve solution the batch job may be considered a sort of workflow, but it will not be considered a composite service as the individual steps are not invoked as services.

In order to invoke a batch job as a service, a service interface has to be provided. Tivoli Workload Scheduler provides support for Web services interfaces, so any scheduling function can be accessed. Otherwise, a separate interface to the scheduler involved must be developed to access information in the scheduling plan and have the job initiated and controlled. Invoking the reader directly is not considered a good idea. If a job scheduler is implemented, you probably want to manage all jobs through this and do not accept direct write to internal reader. In any case, you cannot call batch programs directly without a job or a started task.

The existing program logic can stay unchanged.

- If the complete batch job is intended to be exposed as a service, no changes are needed at all, neither in the logic nor in the flow (JCL).
- If one or more steps are intended to be exposed as services, no changes are needed in the logic, but new flows must be created. It must be ensured that

[3] This depends of course on the capabilities of the scheduler product being used.

the service receives all the input it needs to execute the service function, and that it can provide a proper response and output to the service caller.

If a subprogram or a subroutine is intended to be exposed as a service, this cannot be done by the improve solution. Some level of reprogramming is required and it will fall into one of the two subsequent transition approaches, adapt or innovate.

In any case, when the service is implemented as a batch job, by nature an asynchronous process, no synchronous response will be provided to the caller. An asynchronous response may be provided via WMQ or a mailing service indicating the batch job completion or progress.

Process overview

SOA-enabling a batch application with the improve approach involves the following steps:

1. Identify and select service candidates among jobs and jobsteps.
2. Establish new JCL flows if necessary.
3. Establish the Web service components to interface to the scheduler. Exploit a standard Web services interface if available or write your own.

For detailed information

- IBM Tivoli Workload Scheduler, homepage:
  `http://www.ibm.com/software/tivoli/products/scheduler/`
- Product documentation: Tivoli Workload Scheduler, Reference Guide.
  `http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.tivoli.itws.doc/srf_mst323.htm`

Observations when using an improve transition approach with the batch scenario

Advantages:

- Quick implementation.
- No impact on existing applications. Can run unchanged. Low risk.
- Low complexity in implementation.

Disadvantages:

- Real value of reusable logic may not be exploited.
- Business services not really exposed.

5.4.3 Using the adapt transition approach with batch as the service provider

The logical architecture for the “Adapt” transition approach is shown in Figure 5-35.

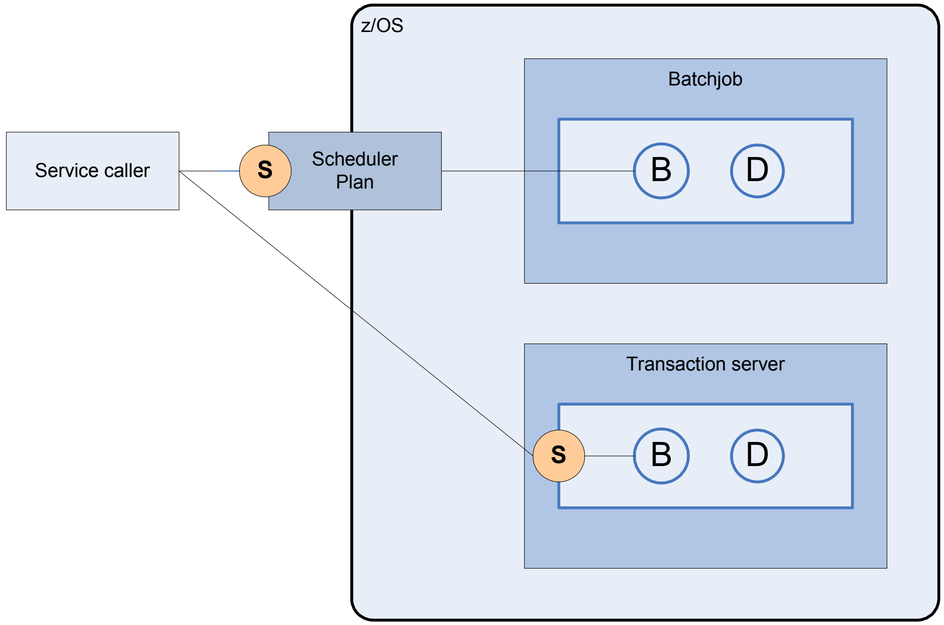


Figure 5-35 Logical overview of the batch “Adapt” transition approach

Two variations are available:

1. Batch-to-batch. Batch is kept as the runtime environment.
2. Batch-to-transaction. Reusable programs or subroutines are refined and redeployed in a transaction server.

Batch-to-batch

The batch-to-batch solution can be considered an extension to the improve transition approach.

Reusable subprograms and subroutines may be put in separate steps and service enabled the same way as in the improve solutions. This implies that they can not be invoked directly but always through the scheduler.

In this solution too, a SOA interface must be existent in order to access the scheduler plan and have the job initiated and controlled.

In this case, changes to program logic is necessary. It requires some code analysis to ensure that the service gets all the input information it needs to execute the desired action and that it can provide a proper response and output to the service caller. The SOA interface itself however is provided by the job scheduler as a web service or by a homegrown component.

Batch-to-transaction

In the batch-to-transaction solution, reusable subprograms and subroutines are adjusted to run as services in a transaction server like CICS or IMS. Probably not in WAS as they are not written in Java.

The services are deployed as standard business services but they have not been implemented based on a real methodological approach.

Process overview

SOA-enabling a batch application using the batch-to-batch solution involves the following steps:

1. Identify and select service candidates among jobs, steps, subprograms, subroutines.
2. Establish new JCL flows.
3. Establish the Web service components to interface with the scheduler. Exploit standard Web services interface if available or write your own.

SOA-enabling a batch application using the batch-to-transaction solution involves the following steps:

1. Identify and select service candidates among jobs, steps, subprograms, subroutines.
2. Prepare and deploy the services in a transaction server.
3. Establish the Web service components to interface the transactions.

For detailed information

- IBM Tivoli Workload Scheduler, homepage.
  `http://www.ibm.com/software/tivoli/products/scheduler/`
- Product documentation: Tivoli Workload Scheduler, Reference Guide.
  `http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.tivoli.itws.doc/srf_mst323.htm`
- Redbook: Application Development for CICS Web Services.
  `http://www.redbooks.ibm.com/abstracts/sg247126.html`
- Redbook: Implementing CICS Web Services.
  `http://www.redbooks.ibm.com/abstracts/sg247206.html`

- Redbook: IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity
`http://www.redbooks.ibm.com/abstracts/sg246794.html`

Observations when using an adapt transition approach with the batch starting scenario

Advantages:

- Relatively quick implementation. Some coding needed.
- Value of reusable logic exploited to a certain extent.
- Relatively low complexity in implementation.

Disadvantages:

- Risk getting redundant code. Existing batch jobs may not be changed.
- Business services exposed only to a certain extent.
- Perhaps too quick solution. Does not really implement SOA. A real methodological approach should probably be taken instead.

5.4.4 Using the adapt transition approach with batch as the service caller

The logical architecture for batch as the service caller is shown in Figure 5-36 on page 191.

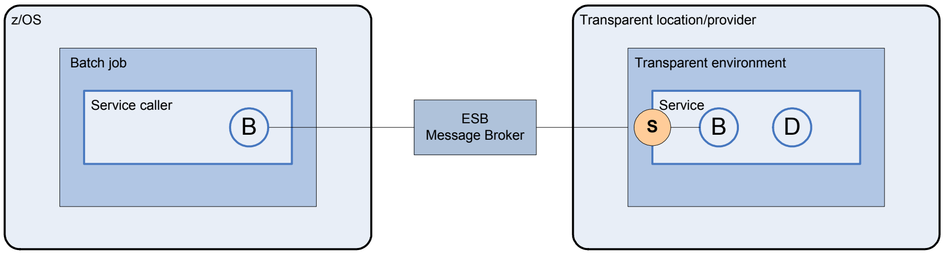


Figure 5-36 Logical overview of the batch as a service caller variation

A batch program can act as a service caller. It may invoke SOA compliant services through an ESB. It may even invoke services in batch through a native or another SOA interface as described in the improve solution paragraph above.

Considerations concerning invoking services from batch as compared with other invocation methods are:

- Calling sub-programs natively provides better performance than service invocation.
- Can create unwanted wait situations for the batch job. Waiting for online transactions to complete. Waiting for synchronous or pseudo synchronous partner processes to complete.
- May be used in connection with asynchronous WMQ based communication with partners. Implies dependences to message broker availability and responsiveness.

Process overview

Preparing a batch application to act as a service consumer involves exploitation of standard interfaces to the ESB installed.

Observations when using the adapt transition approach with the batch starting scenario

Advantages:

- Reuse of real business services from batch.

Disadvantages:

- Large performance impact if high volume calls.
- Delays or instability may occur as a result of external callouts.

5.4.5 Using the innovate transition approach

The logical architecture for an “Innovate” solution is shown in Figure 5-37.

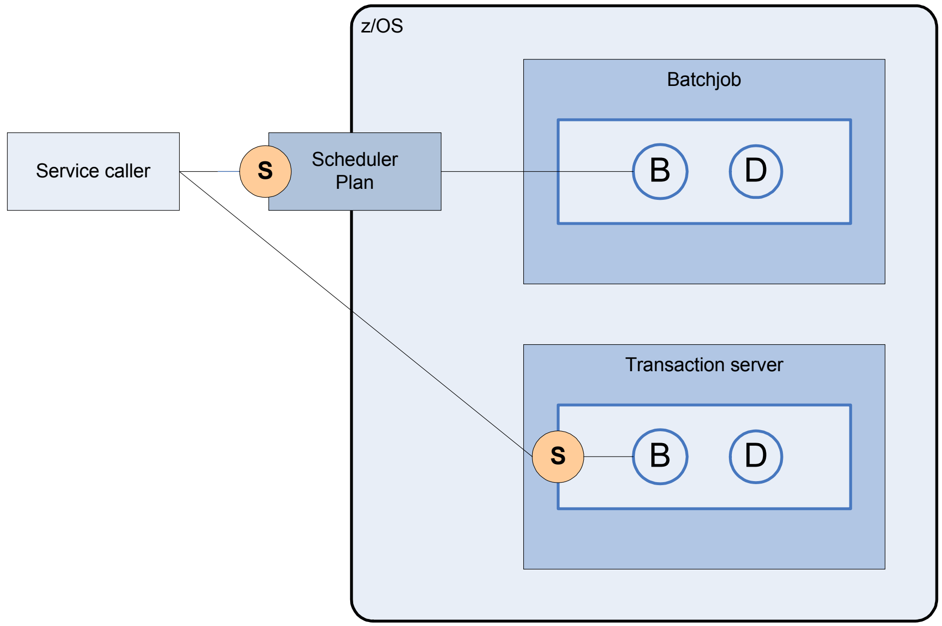


Figure 5-37 Logical overview of the batch "Innovate" variation

Solution technique implementation

The logical architecture for the innovate transition approach looks similar to the adapt transition approach. The core logic could be used unchanged but it is likely that re-engineering is needed and that new business services appear as a result of a methodological approach. The most probable deployment destination is a transaction server like WebSphere Application Server (WAS), CICS or IMS.

It is possible that some re-engineered services can be deployed in batch.

The differences by enabling a service in an application server and a batch job are:

- If the service is deployed in an application or transaction server, the service caller can have a synchronous response, as opposed to the batch implementation where no direct response back is created.
- In a transaction server the services can be invoked synchronously or asynchronously.
- With the use of message protocols in a transaction server, even pseudo-synchronous invocation is possible. Pseudo-synchronous invocation provides the opportunity to call several services asynchronously, in parallel and correlate the responses (reply-messages) in the caller.

- In the transaction server implementation, the service interface is directly connected to the service code itself where in the batch implementation, the interface is connected (indirectly) to the scheduler.

From batch job to process flow

A batch job can be considered a flow of activities in a process. See Figure 5-38.

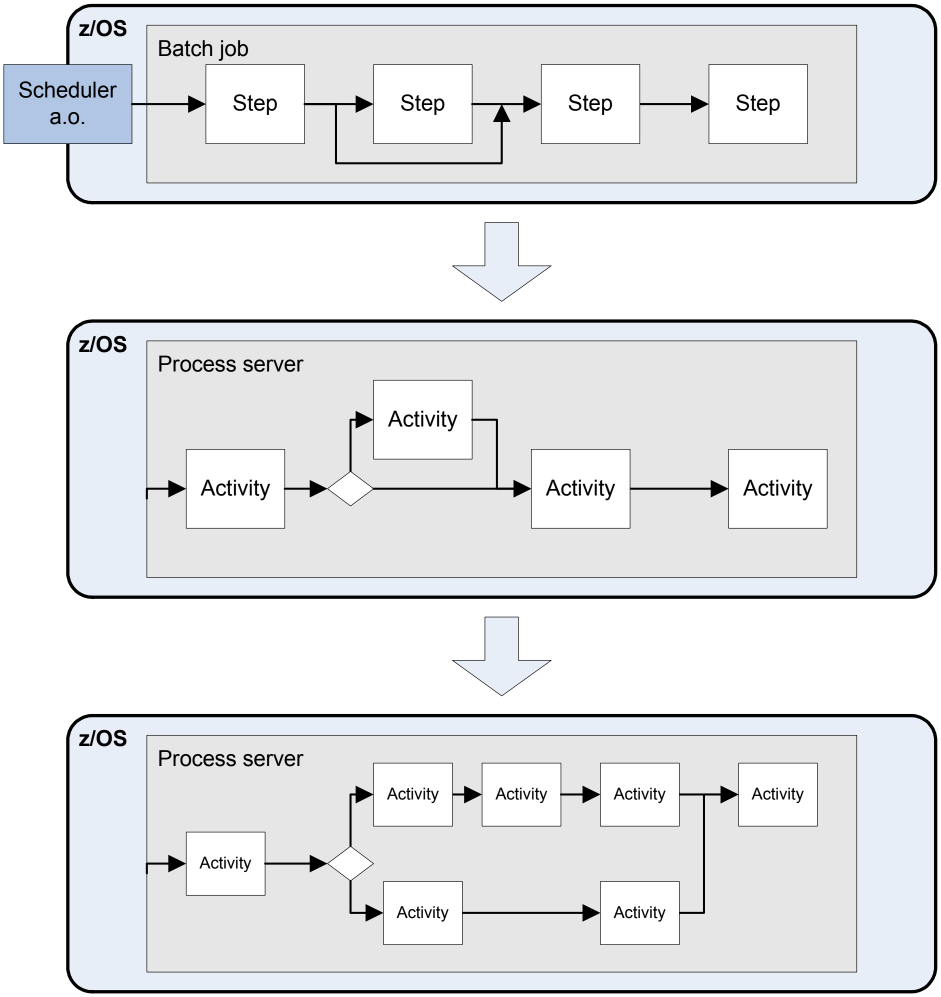


Figure 5-38 Illustration of a batch job as a sequence of activities

The flow of logic in a batch job expressed in steps, programs and embedded sub-programs may be considered a composite service consisting of separate atomic services. The navigation between the activities is based on a binary determination whether a step in the predefined sequence will be executed or not, managed by condition codes.

In this innovate transition approach:

- The atomic services must be identified as a result of a structured methodology. A decomposition based on the step structure may not reflect the basic logical service structure.
- Business rules may be extracted from the code and implemented in the process flow, making the atomic activities more granular and reusable.
- When a proper decomposition has been produced, atomic services can be developed and deployed.
- A composite service can be implemented including several atomic services. It may be deployed in a message broker or a process server.
- Even if it may be technically possible to implement it as a batch job it may not be a good idea. Batch jobs must run unattended with no manual interference. Call outs from batch can also affect the batch operation. There may be external interrupts or delays causing stops or exceeded batch durations which may result in extended batch windows.

Process overview

SOA-enabling a batch application with the innovate transition approach involves the following steps:

1. Identify and select candidate code.
2. Redevelop the code observing standard SOA methodologies.
3. Prepare and deploy the resulting services as standard services in a transaction server.
4. Establish the web service components to invoke these standard services.
5. If applicable, create and exploit composite services.

For detailed information

- Redbook: Application Development for CICS Web Services.
  `http://www.redbooks.ibm.com/abstracts/sg247126.html`
- Redbook: Implementing CICS Web Services.
  `http://www.redbooks.ibm.com/abstracts/sg247206.html`
- Redbook: IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity
  `http://www.redbooks.ibm.com/abstracts/sg246794.html`

Observations when using an innovate approach with the batch starting scenario

Advantages:

- Real business services produced and exposed according to standards.
- Large level of reuse.

Disadvantages:

- Large impact on existing applications.
- Change introduces risk.
- Requires skills and experience. In both existing and new technology.

5.4.6 A practical, mixed approach

A practical and feasible approach in many companies could be to:

1. keep existing batch jobs unchanged
2. follow the evolution and implement new code as standard services
3. as a consequence, live with the resulting redundancy for some time

This approach may imply a larger effort maintaining the applications, but it is certainly a lot easier to implement. It may turn out to be the only possible approach considering the scarcity of skills in these disciplines.

Observations when using this mixed approach with the batch starting scenario

Advantages:

- Very fast implementation. No effort at all.
- No operational risks.
- Ultimate (existing) level Quality of Service.
- New real business services can be produced and exposed according to standards.
- Some level of code reuse may be exploited.
- Low level of skills dependencies on short term.

Disadvantages:

- Redundant code to be maintained for a long time.
- Requires skills and experience. In both existing and new technology. Also in the long run.

5.5 SOA implementation scenarios - Data access and integration

In this chapter we describe SOA implementation scenarios for all variations described in the “data access and integration” starting scenario. These variations are:

- Data access
- Integrator
- Batch ETL
- Messaging ETL

When discussing the SOA implementation scenario for *Integrator* we will discuss the idea of ”information as a service”. We describe which IBM products are available on the z/OS platform to enable the concept of “information as a service”, looking at them in the context of the IBM SOA reference architecture (that means we will not discuss here the functionality available in the area of aggregation, replication, consolidation, analytics, special queries, etc., but just the functionality that enables the concept of “information as a service”).

Table 5-4 on page 198 shows a summary of the variations we discuss.

Table 5-4 Summary of data access and integration variations

Transition approach	Variation	Described in
Improve	Variation 1: Data access	“Data access using the Improve transition approach” on page 198
Adapt	Variation 1: Data access	“Data access using the Adapt transition approach” on page 201
Adapt	Variation 2: Integrator	“Integrator using the Adapt transition approach” on page 203
Adapt	Variation 3: Batch and messaging ETL	“ETL using the Adapt transition approach” on page 209

We are structuring the sections based on the targeted level of SOA enablement maturity (improve, adapt, innovate), similar as discussed in the other SOA implementation scenarios.

5.5.1 Variation 1: Data access

In the following sections we discuss SOA enablement using an Improve and Adapt transition approach of data access.

Data access using the Improve transition approach

The Improve transition approach looks at the way data access is taking place, and will provide a SOA-enabler for the access. Considering the starting scenario, access usually takes place from CICS/IMS transactions or batch programs, and these components either:

- access data directly, using native data interfaces made available by z/OS and the database subsystems
- or may use (in case of DB2) stored procedures to encapsulate business logic and get data

The logical architecture for the Improve transition approach for data access is shown in Figure 5-39 on page 199.

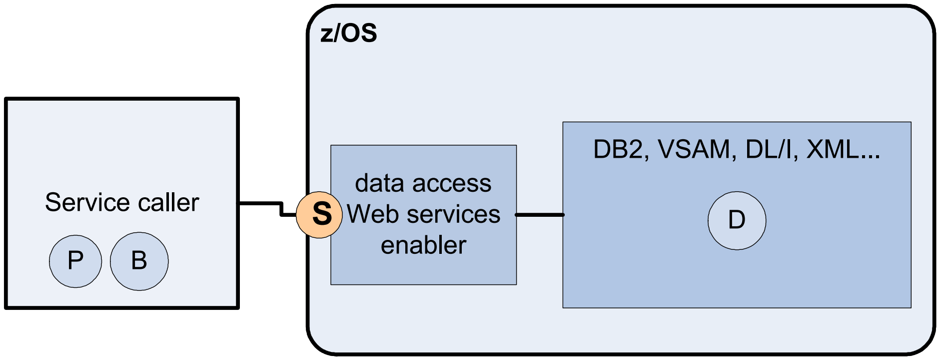


Figure 5-39 Logical architecture for SOA enablement of data access

In the case of DB2 and its stored procedures functionality we can devise another logical architecture for the Improve transition approach, shown in Figure 5-40. As seen, the business part of the application is split between the service caller and DB2.

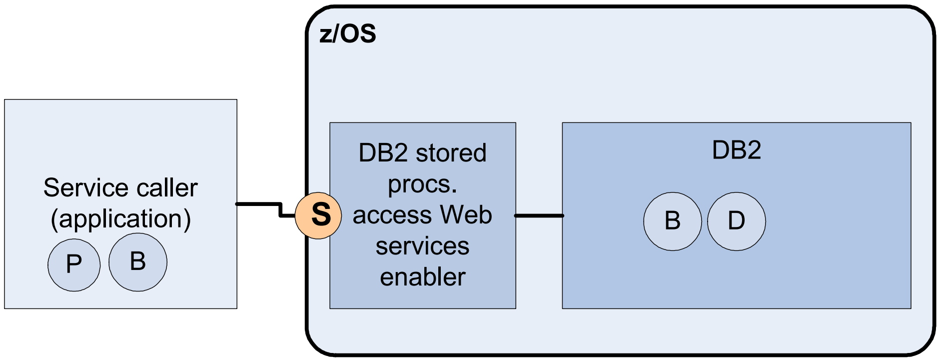


Figure 5-40 Logical architecture for SOA enablement of stored procedures access

Products used

DB2 for z/OS allows enabling DB2 data and stored procedures as Web services through the *Web Services Object Runtime Framework (WORF)*. So, DB2 becomes a service provider in SOA terminology, a SOAP enabled service provider.

WORF runs inside WebSphere Application Server, and provides an environment to create XML-based Web services that access DB2.

A Web service is defined by using a *Document Access Definition Extension (DADX)* file. In the DADX file we define Web services based on SQL statements and stored procedures.

Based on the definitions in the DADX file, WORF performs the following actions:

- Resource-based deployment and invocation.
- Automatic service redeployment at development time when defining resources change.
- HTTP GET and POST bindings in addition to SOAP.
- Handling the generation of any Web Services Definition Language (WSDL) and UDDI information that the client application needs.
- Providing DB2 JDBC connection information for the target database.
- Can be easily generated via WSED/RAD tooling.
- Formatting the result into XML, converting types as necessary.
- The URL to invoke the Web service specifies the DADX file as well as the SQL operation (method) that has to be invoked

Figure 5-41 shows the implementation of this solution.

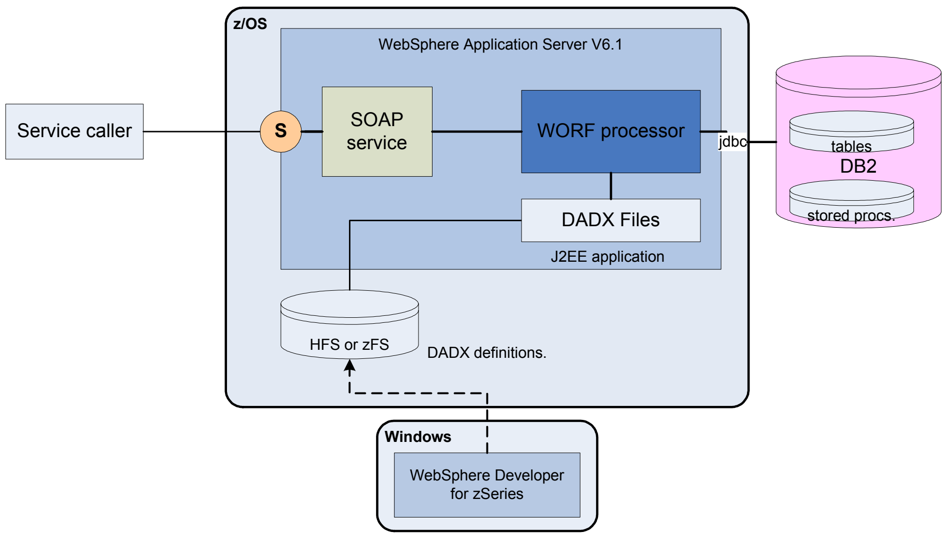


Figure 5-41 DB2 access enabled through WORF

Observations when using an Improve transition approach with the DB2 access/stored procedures variation

Advantages:

- Quick implementation - simple tooling and deployment, quick time to market.
- Low complexity.
- No change in the business logic (stored procedures).
- Can be used to encapsulate stored procedures as SOA-enabled services and improve reusability (can be called as a web-service from anywhere in the SOA implementation).

Disadvantages:

- Requires WebSphere to position WORF.
- Additional overhead compared to native data access.

Data access using the Adapt transition approach

In the Adapt transition approach we take a look at business logic positioned in DB2 (stored procedures) and the way it can behave in a SOA-enabled environment. This business logic can not only access data in DB2, but, through user-defined extensions, can access external Web services. Therefore we have a new logical architecture, which is shown in Figure 5-42. In this case the logical architecture is at the same time the product implementation architecture.

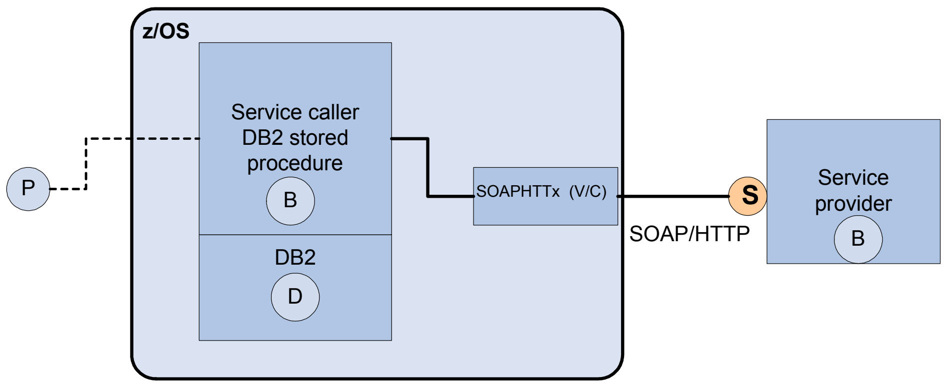


Figure 5-42 Logical architecture of Adapt transition approach using DB2 stored procedures

Products used

IBM DB2 for z/OS has made available *User-Defined Functions (UDFs)* for implementing Web services consumers in DB2. These new Web service consumer UDFs enable the database system to directly invoke SOAP-based Web services using SQL statements. This eliminates the need to transfer data between Web services and the database. The Web services consumer converts the results obtained by calling WSDL interfaces into DB2 table or scalar functions.

When a service consumer receives the result of a Web services request, the SOAP envelope is stripped and the XML document is returned. An application program can process the result data and perform a variety of operations, including inserting or updating a table with the result data.

Internally, DB2 provides these actions using a SOAP UDF:

- Receiving input parms from SQL statement.
- Composing HTTP/SOAP request based on the input to the UDF.
- Invoking TCPIP socket call via z/OS USS APIs and send HTTP/POST request.
- Receiving reply back from Web service provider.
- Validating HTTP headers.
- Stripping SOAP envelope and return SOAP Body (include namespace referenced in SOAP envelope) to DB2 client application.

This DB2 feature presents us with an interesting new architecture (shown in Figure 5-43). In this architecture we see the possibility to indirectly SOAP-enable batch COBOL programs as Web services consumers. We also see the participation of DB2 services (provider and consumer) in an ESB (either WebSphere ESB or WebSphere Message Broker).

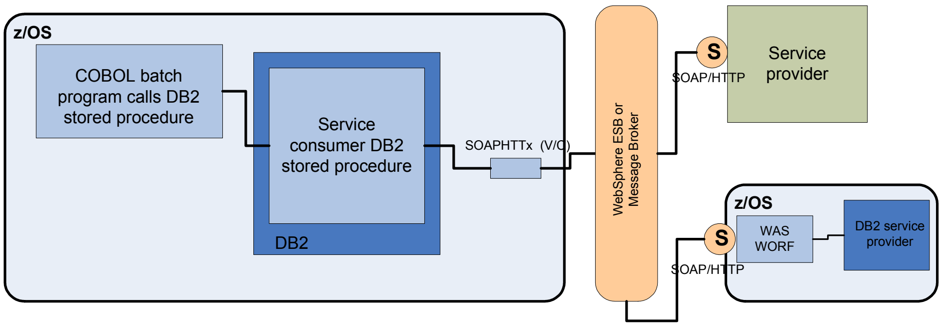


Figure 5-43 DB2 stored procedures as service consumer and service provider in SOA

Observations when using an Adapt transition approach with the DB2 consumer variation

Advantages:

- Relatively easy to implement, some tooling support.
- Full participation of DB2 in an ESB.
- Can be used to indirectly SOA-enable COBOL batch programs.

Disadvantages:

- Requires changes in the DB2 / stored procedures business logic.
- Additional overhead compared to a native data access.

5.5.2 Variation 2: Integrator

In the following sections we discuss SOA enablement using an Adapt transition approach of data/information integration.

Integrator using the Adapt transition approach

In this topic we describe the SOA implementation scenarios for SOA-enablement of the integrator scenario. Generally speaking we see the integrator scenario and ask ourselves: what is being delivered by the integrator? The answer is clear: *information*. Therefore we must find a way to describe and enable the discovery of information in the SOA world, and in this way the concept of “information as a service” appears.

Why “information as a service”?

In today’s large organizations, built over time through evolving technology, using different product capabilities, changing application requirements, expansion and mergers, different parts of the enterprise use different information management systems to store and search their critical data. Each of these disparate systems carry overlapping information, managed using various technologies and data formats. As these information systems become more complex, they become more difficult to change and increasingly expensive to maintain or develop.

The same problem that we have with applications systems exist in the data integration area (multiple interfaces, spaghetti connectivity, high complexity, redundant data positioned in different databases, no or little governance). We would like to be in a situation like the one described in the Figure 5-44 on page 204.

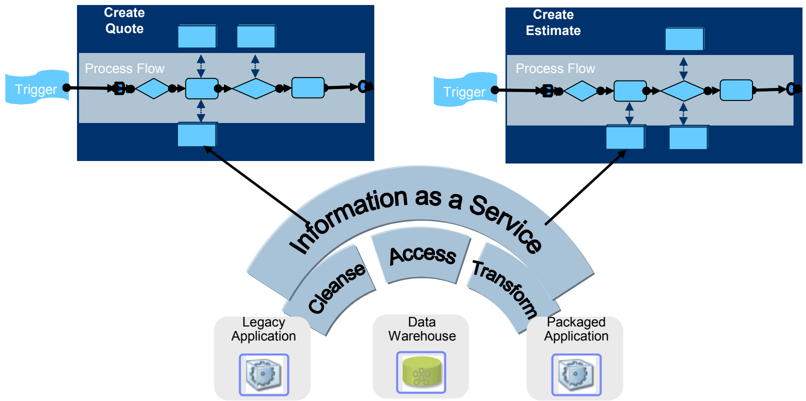


Figure 5-44 Information as a service presented as an layer separating the applications from data

The SOA building block “Information Services

The discussion in this section revolves around the content of the Information Services SOA building block. The building block contains a number of disciplines, out of which we will focus on information integration services.

We define a new type of service for the information, and we apply the SOA criteria for this service. The information service as defined will be reusable, have an implementation-independent interface, will be location transparent and transport neutral; it will behave as any other SOA service.

An *Information service* is a type of service that allows the managers of shared information assets (such as customer information or product descriptions) a consistent, auditable and secure way to share this asset while maintaining control over how it is used. For the service consumer, the Information Service is a trusted information source provisioned by people who understand the meaning of the data and have responsibility for maintaining it.

And now we see one of the SOA principles at work: by separating the interface from the implementation, service providers are free to change how and where the data is produced and managed internally. For example, new application requirements might prompt an enterprise decision to store some of the information assets as XML native inside DB2. Having an interface above the service provider (in this case DB2 with XML) allows the service consumer to retrieve its data without being aware of the internal change that has taken place.

Invoking an Information Service is one of many ways of programmatically accessing data. It is not appropriate for all types of data access. A typical Information Service will have one or more of the following characteristics:

- It will deliver *information*, that means there is a process that prepares pieces of data from (possible different sources) and delivers it.
- Data preparation might include *aggregation, synchronization, standardization, duplication, reformatting, conversion*, etc.
  These combinations of data preparation tasks form data integration patterns. Choosing the right pattern depends on the amounts of data, where the data came from, when it needs to be delivered and in what form.
- *Integrate multiple data sources*.
  Many business processes require information that results from processing large sets of data, often from multiple sources. Bringing together this data can be a data intensive process that calls for specialized data management tools.
- *Provide virtualized views*.
  Virtualization provides transparency that masks the differences and implementations of underlying data sources from users. Ideally, it allows data from multiple heterogeneous sources to appear to the user as a single system. The service consumer does not have to be aware of where the data is stored (location transparency; again a SOA criteria), what language or programming interface is supported by the data source (invocation transparency), if SQL is used, what dialect of SQL the source supports (dialect transparency), how the data is physically stored, whether it is partitioned and/or replicated (physical data independence, fragmentation and replication transparency) or what networking protocols are used (transport neutrality). The user should see a single uniform interface, complete with a single set of error codes (error code transparency).
- *Reusability*.
  To be reusable, Information Services must be at the right level of granularity to encapsulate an information need that is repeatable. An information service should be designed in such a flexible way that it allows future changes, without the need for changing the interface (extension of the interface should be possible, the service consumers that use the older interface do not need to change).

IBM products that can be used to create “information as a service” components

The only product available on z/OS is WebSphere Information Integrator in several flavors (classic federation, replication, Event Publisher for DB2/IMS/VSAM). We will address only the SOA-relevant elements, not the detailed functionality of the product. We will refer to the product in the remaining of this section as WebSphere Information Integrator.

Websphere Information Integrator

WebSphere Information Integrator (WebSphere II) allows users to write applications as if all of the data were in a single database, when, in fact, the data may be stored in a heterogeneous collection of data sources.

By providing an SOA interface to these capabilities the Information Server gives users real-time access to integrated business information across and beyond the enterprise by publishing reusable services. These operations include the ability to:

- *Connect* to any data or content, wherever it resides. It provides direct, native access to relevant information sources (note: access to information, not to data).
- *Understand* and *analyze* information, including its meanings, relationships and lineage.
- *Cleanse* information to ensure its quality and consistency.
- *Transform* information to provide enrichment and tailoring for its specific purposes.
- *Federate* information to provide a unified view to people, processes, and applications.

WAS II delivers information services using an SOA framework called the IBM *WebSphere Information Services Director (WISD)*. WISD facilitates *reuse* of the information service, by ensuring consistent definitions, packaging, and rules applied to the data. WISD facilitates governance by centralizing control and management.

WISD uses open standards and is is deployed on a J2EE-based foundation framework that provides flexible, distributable and configurable interconnections among the many parts of the architecture through accepted SOA standards.

WISD allows the same service to support multiple protocol bindings, all defined within the WSDL file. This improves the utility of services (they can be used by different client types) and therefore increases the likelihood of reuse and adoption across the enterprise.

An information service can be invoked over the following bindings:

- *Web service.*
  Web services compliant consumers can access the information services.
- *SOAP over JMS.*
  In a messaging environment, the Information Services Director can automatically generate an asynchronous JMS queue listener (Message-Driven Bean) and route incoming messages into Information

Services. It can adapt the output of an information service into a SOAP message that can be posted to one or more JMS queues or topics.

- *EJB*.
  For Java-centric development, the Information Services Director can generate a J2EE-compliant EJB (stateless session bean) where each Information service is instantiated as a separate synchronous EJB method call.
- *Service Component Architecture (SCA)*.
  This binding provides a client programming model and consistent way of describing components as services available the WebSphere Enterprise Service Bus product.

We return now to the Adapt transition approach. What we are trying to do is to use the functionality delivered by the IBM products (previously described) in a SOA-enabled architecture. Therefore we are interested to expose the "information as a service" components, like shown in Figure 5-45.

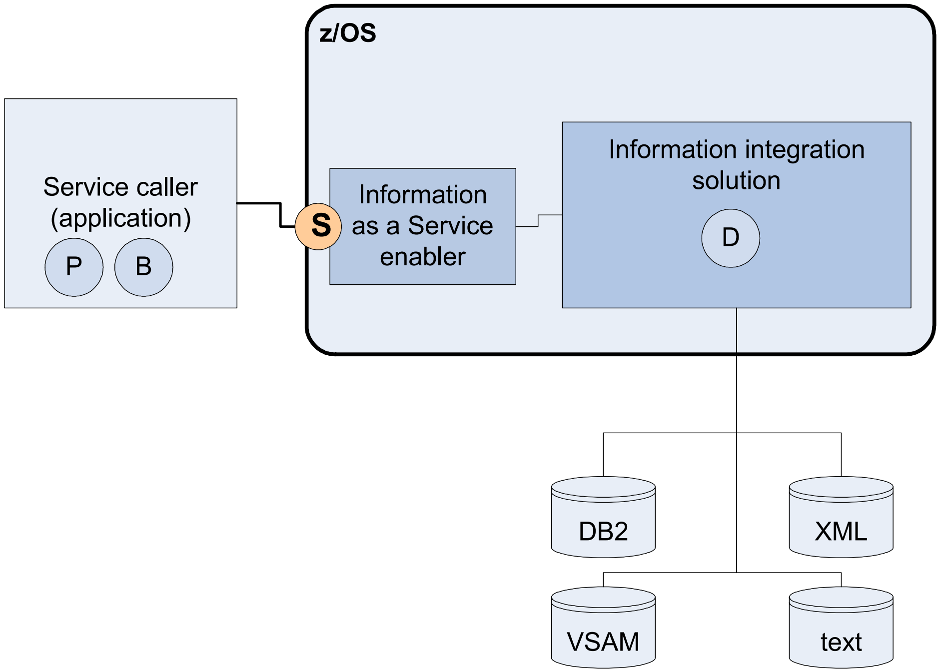


Figure 5-45 Logical architecture for the integrator scenario

Products used

We can use, for the purpose of discussion, IBM WebSphere Information Integrator. We have previously seen that the product can deliver its "information service" over several bindings. Therefore we can envision several implementations of the logical architecture. Figure 5-46 shows some possible implementations:

- An implementation where the SOAP-enabled service consumer access WebSphere II over the provided Web services binding.
- An implementation where the access is over an MDB (generated by WebSphere II), placed on the WebSphere Application Server, enabling it to receive JMS message while being able to call the backend Integrator.
- An implementation where WebSphere II generates an EJB with backend methods, and places the EJB on the WebSphere Application Server. This means any Web services enabled client can now (over the WebSphere Application Server), access the EJB and therefore the backend integrator.
- An implementation in which the “information as service” is reached using the SCA binding over the WebSphere ESB bus.

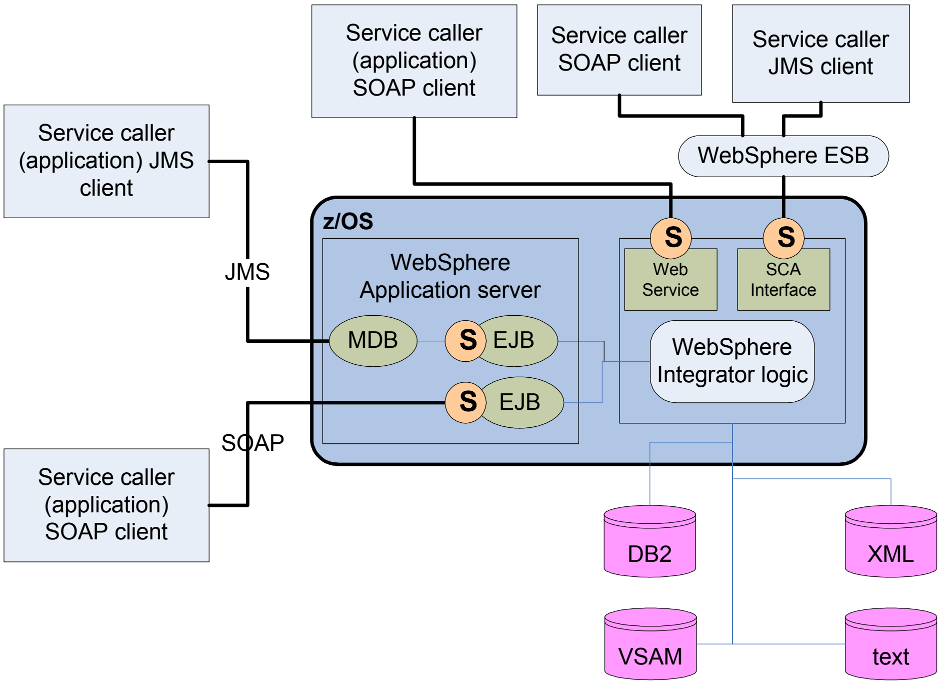


Figure 5-46 Different implementations of WebSphere II SOA-enabled

Which of the several possible implementations can be implemented in a specific enterprise depends on many factors (not the least being the complexity of the landscape, the availability of the IBM products that will run the SOA-enabler interfaces and the performance required).

Observations when using an Adapt approach with the Integrator Starting scenario

Advantages:

- Implementation of the SOA-enabled "information as a service".
- All derived services inherit SOA characteristics and benefits.
- Allows new and innovative use of the "information service", decoupling business logic from data positioning/structure/representation.
- Allows easy creation of new data models and their access as information services.
- No changes in data implementation are required.
- All the advantages of the Integrator related to data management and access.

Disadvantages:

- Requires WebSphere Information Integrator and possibly other WebSphere products for more flexible SOA-enablement (WebSphere Application Server, WebSphere ESB)
- Additional overhead compared to a native data access.

5.5.3 Variation 3: batch and messaging ETL

In the following sections we discuss SOA enablement using an Adapt transition approach of batch and messaging Extract Transform and Load (ETL).

ETL using the Adapt transition approach

From the logical point of view the batch ETL and messaging ETL look the same, as shown in Figure 5-47 on page 210. As you can see, we interpret the ETL layer as an ESB. In case of the batch ETL the ESB will contain file-drop attachments, and the mediations are taking place inside the bus. In case of a messaging ETL the ESB will contain messaging attachments.

We can implement the Extract Transform Load (ETL) functionality as mediation flows inside the ESB, letting the bus be concerned with the reliable transport of the data; the business rules that control the flow can also be implemented in the ESB. As the business rules that govern the ETL process are implemented inside the ESB, we have a dynamic and easily changeable application architecture.

If we implement this SOA concept we will have reduced considerably, for an enterprise that has many file transfers, the number of connections (each application connects to the ESB, instead of any-to-any between applications).

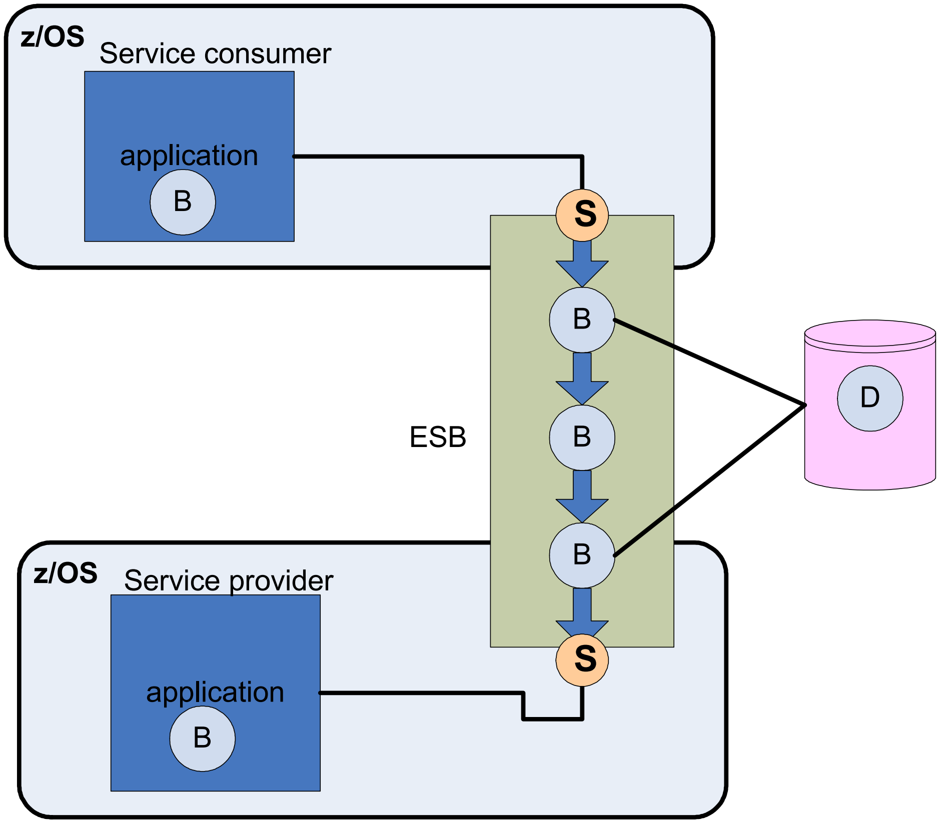


Figure 5-47 Logical architecture for batch ETL and messaging ETL

The logical structure that we create for all applications in the enterprise is shown in Figure 5-48 on page 211.

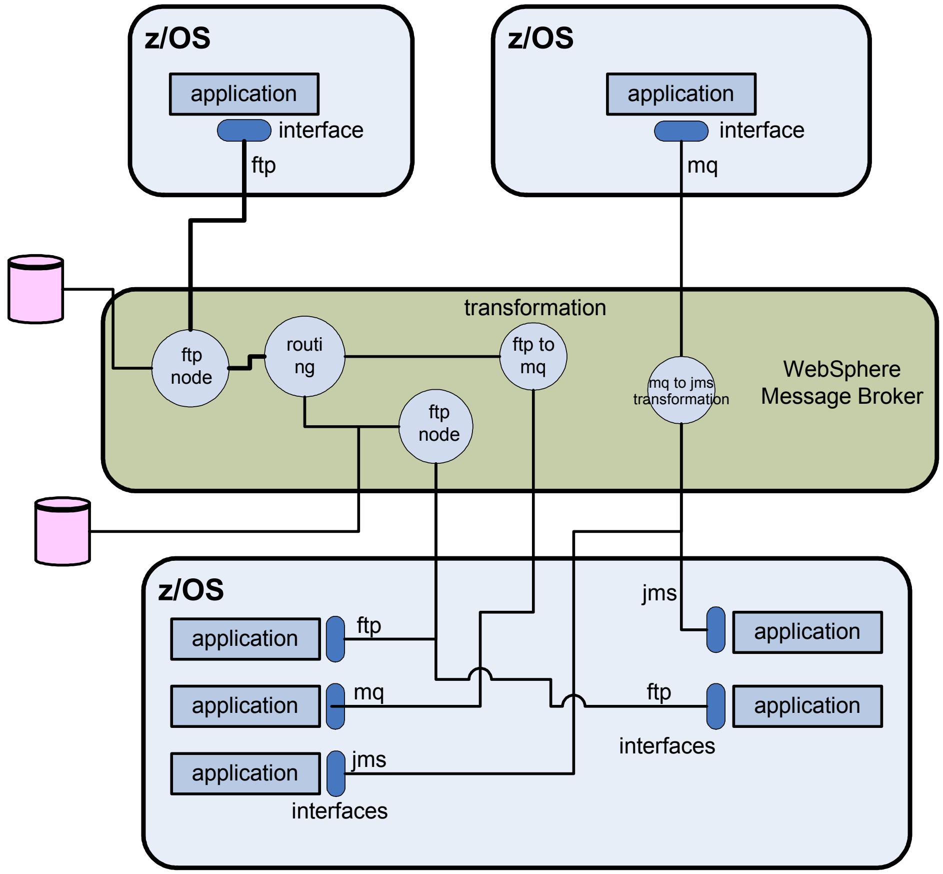


Figure 5-48 Enterprise SOA-enabled batch/messaging ETL

Products used

In the case of batch and messaging ETL we could use IBM WebSphere Message Broker with the necessary nodes and mediations. The physical implementation might look like in Figure 5-49 on page 212.

The consequences of this implementation are manyfold:

- The application (as a service consumer or provider) has only one attachment point to the WebSphere Message Broker. There is no any-to-any configuration necessary.
- The mediations and transformations are implemented as services inside the WebSphere Message Broker, and as such can be written as reusable components. They run in a controlled way and can be managed. The flow of

mediations can be designed with existing tooling and deployed to the ESB runtime.

Due to the many interfaces (and node implementations) supported by WebSphere Message Broker it is possible to establish communication relationships between service consumers and providers that have different interfaces (for example a JMS/MQ service provider can receive FTP files as sequences of JMS/MQ messages).

.

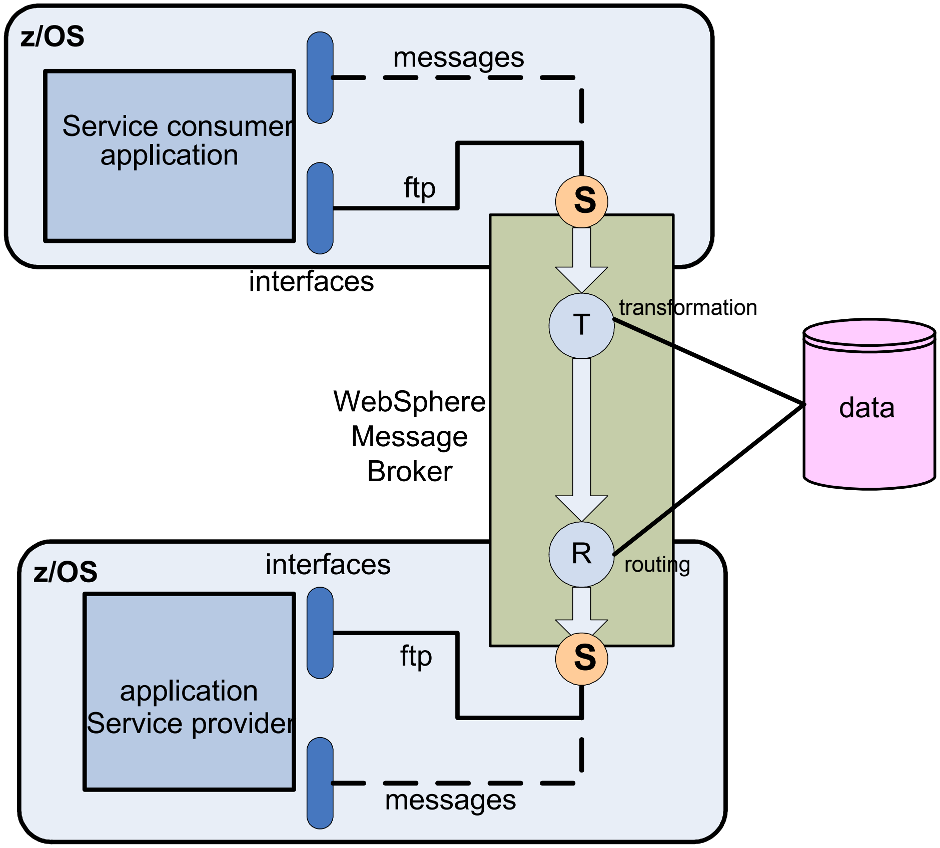


Figure 5-49 Implementation solution for batch/messaging ETL with WebSphere Message Broker

Observations when using an Adapt transition approach with the batch and messaging ETL starting scenario

Advantages:

- Simplify the configuration of middleware and connectivity (enterprise wide).
- Increased flexibility for the applications.
- Interface independent.
- Publish/subscribe model possible for “ftp” based integration.
- Controlled transformation through configurable business rules.

Disadvantages:

- Installation of an ESB is necessary.
- Performance overhead compared with native messaging / native FTP.

5.6 SOA implementation scenarios for homegrown SOA

In 3.5, “Starting scenario - Homegrown SOA” on page 87 we discussed the scenario in which companies have tried to implement some kind of Service Oriented Architecture, with the objective to increase reusability and implementing transparency. We have mentioned that there are many flavors of this scenario and it is impossible to discuss them all. However, most of those existing Service Oriented Architectures do have a few things in common:

- There is an abstraction layer before accessing the real business logic, either implemented in a J2EE application server or a backend transaction manager or database server.
- In most cases, the service interface is more a “wrapper” than a “native” service interface.
- The transport between different servers or containers is usually taking place using JMS / WebSphere MQ. In some cases, when the application environment is all J2EE, we also see RMI-IIOP as transport mechanism.

Many of the “homegrown” SOAs do not comply to the SOA definitions as we have them today and the maturity level is relatively low. In the homegrown SOAs we do see the use of standards-based protocols and we do see some form of location transparency, although implemented with homegrown frameworks. Reusability is many times not ideal yet, as a major refactoring effort to improve service granularity in the business logic has not taken place. “Spaghetti” logic is still being invoked as “spaghetti”, but using a standards-protocol and a self-built mechanism to provide location transparency. So, in most homegrown SOAs there is work to do to really achieve the promised benefits of a modern SOA!

Table 5-5 provides a summary of the variations discussed in this chapter.

Table 5-5 Summary of homegrown SOA implementation variations

Approach	Scenario	Summary
Improve	Introducing open standards	"Open standards based SOA Interface" on page 215
Adapt	Using an open standards-based service registry	"Open standards based Service Registry" on page 216
Innovate	Various scenarios possible based on maturity level	

The SOA implementation scenarios are based on the starting scenarios as described in 3.5, "Starting scenario - Homegrown SOA" on page 87.

5.6.1 Using the Improve transition approach

Based on our starting scenarios, there is at least one option where the "Improve" transition approach can be used. That is how to make the SOA Interface callable via open standards based protocols, if that is not already the case.

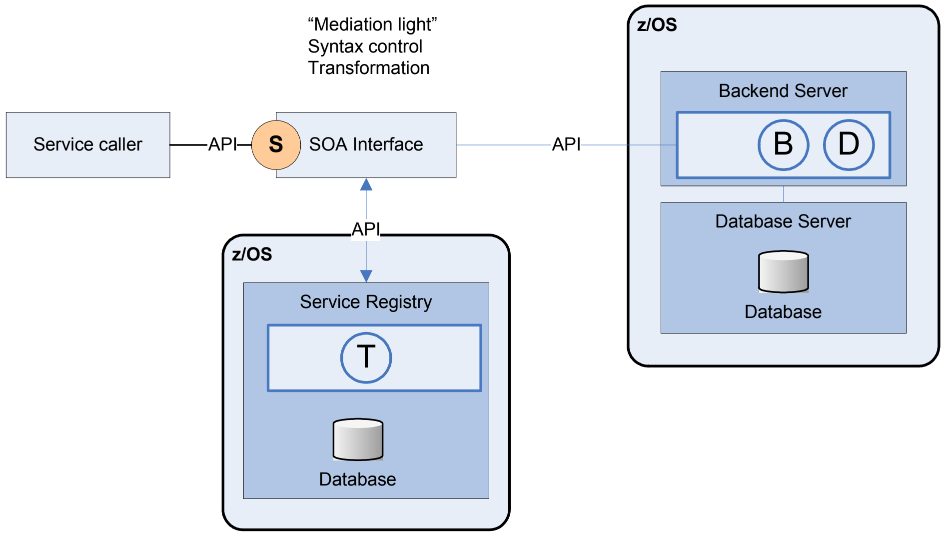


Figure 5-50 Overview of the homegrown SOA Improve transition scenario

Solution techniques for "Improve"

The use of the Improve transition approach usually leads to the adoption of the "Adapter-provided Service interface" pattern. In this scenario we will illustrate how the in-house developed standard interfaces can be exposed via Open Standards based protocols.

The core functions in the backend transaction server remain untouched in this migration scenario.

Open standards based SOA Interface

Many homegrown SOA solutions use WebSphere MQ as transport protocol. On top of WMQ, there is logic implemented that masks away some of the funtionality typically available in a broker or ESB. That is what we call the *SOA Interface* in Figure 5-50 on page 214. It is the SOA interface that is the entry point for the call to the service.

The SOA interface provides functions to provide some level of location transparency, such as correlating a queue to a service name. There may also be some level of support for mediation, syntax control and transformation.

The *Service Registry* is located in DB2 or another datastore, exposed through a technical API (developed in-house).

For this scenario, we assume that the backend functions in CICS and IMS will remain the same, and called via the same interface as before. The changes we apply are in the area of how the services are called.

Solution technique implementation

The solution technique in this scenario can best be described as an "Adapter-provided Service interface" pattern, because we implement an "adapter" in front of the Service Registry.

Process overview

Considerations when moving a homegrown SOA to an open standards based service interface are the following

- What can be done to standardize the native protocol between the caller and the provider?
  - Is the protocol based on proprietary communication, and in that case, how can that be opened up?
  - Maybe the techniques that are described in the 3270 transition scenario chapter could be applied.

- What about the application architecture? Are the services structured in a way so they can be packaged and used by another protocol than they originally are built for?

There are many considerations in this scenario, and not one single answer to these questions. It is very much about deciding on the strategic and tactical steps for the migration.

What we have illustrated is a way to enable the SOA Interface, to be accessible via an open standards based API and protocol, e.g. SOAP.

For detailed information

- Redbook: Patterns: Extended Enterprise SOA and Web Services - `http://www.redbooks.ibm.com/abstracts/sg247135.html`

5.6.2 Using the Adapt transition approach

The purpose of this scenario is to illustrate how to:

- Create a loose coupling between the caller and the homegrown SOA interface.
- Enable a service caller to use open standards based protocols to access the Service Registry in order to find a service.
- The Service Registry can be accessed via Open Standards based protocols, and in that case, how the core back-end functions can be exposed as services.

We assume that the SOA interface is callable via SOAP, as described in the 5.6.1, “Using the Improve transition approach” on page 214.

Open standards based Service Registry

Figure 5-51 on page 217 depicts how the Service Registry is used in calling a service. Note that in this example we do not focus on how services are published, but only how they are looked up.

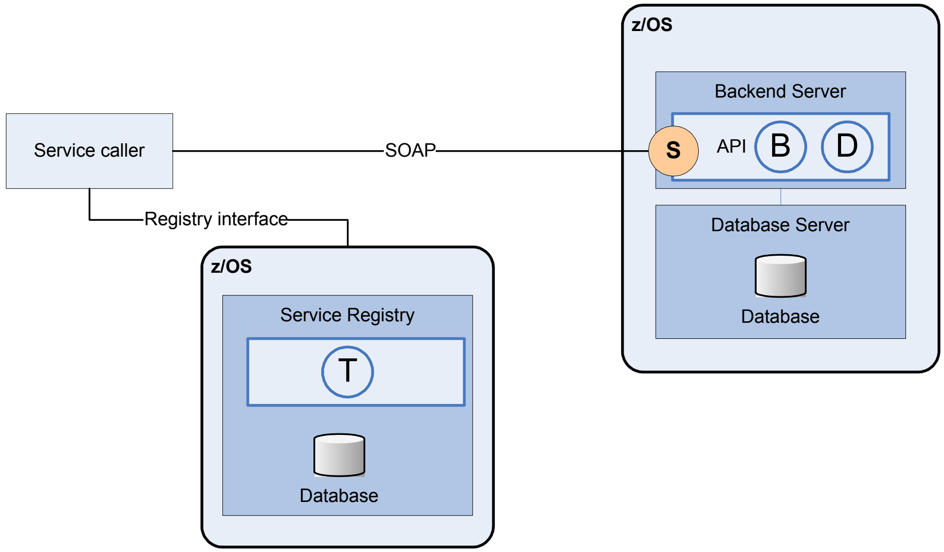


Figure 5-51 Overview of the homegrown SOA Adapt transition approach

Process overview

Considerations when moving a home grown SOA to an open standards based service registry are the following:

- What can be done to standardize the protocol between the client and the service registry?
  - Should the service registry at all be in-house developed? Or is there a product available that provides the functionality?
  - Which protocol should be used?
  - Maybe the techniques that are described in the 3270 transition scenario chapter could be applied.
- What about the application architecture? Are the services structured in a way so they can be packaged and used by another protocol than they originally are built for?

There are many considerations in this scenario, and not one single answer to these questions. It is very much about deciding on the strategic and tactical steps for the migration.

What we have illustrated is a way to enable the SOA Interface, to be accessible via an open standards based API and protocol, e.g. SOAP.

For detailed information

- Redbook: Patterns: Extended Enterprise SOA and Web Services - `http://www.redbooks.ibm.com/abstracts/sg247135.html`

5.6.3 Using the Innovate transition approach

An Innovate transition approach for a homegrown SOA could be, for example, refactoring business logic to achieve a better granularity of the services or the implementation of a full ESB, if not already implemented.

It depends on the maturity of the homegrown SOA and the ambition level to what extent an Innovate transition approach is necessary.

6

Towards service integration and process integration

In this chapter we discuss the following topics:

- Define the transition stages towards service integration and process integration (by defining SOA implementation blocks) and present the SOA-related activities to be implemented during these stages.
- Describe the reasons for moving from stage to stage.
- Describe the status of the enterprise after undertaking the transition scenarios described in chapter 5.
- Indicate which IBM products will be positioned from stage to stage, and how their features are exploited.

6.1 The SOA implementation block approach

The approach called "SOA implementation blocks" allows an enterprise to select individual elements of the SOA architecture to implement. Blocks can be implemented one at a time or in groups designed to meet an immediate business need. Some organizations will start by adding one block at a time as they build a full SOA architecture. These organizations will achieve significant business value with each block they add. Other organizations may design a solution that initially requires multiple blocks to implement. Still others may start off with a design that doesn't pause until all the SOA blocks are in place. The process has then the following steps:

1. Select blocks specific to the requirements.
2. Implement an immediate solution.
3. Execute a simple quick start.
4. Extend to more complex requirements.
5. Add additional blocks "as needed".

There are many ways to define the implementation blocks, their content and granularity. Of course each enterprise will apply the activities defined in the block according to its requirements; this is no "one fit for all solution", but a process by which the enterprise can see what has to be done generally in a building block and then select the parts relevant for it.

We chose an approach favoured in other IBM documents that defines the SOA implementation blocks in the following ways:

6.1.1 Stage one - "service enablement" implementation block

The following implementation block and its associated tasks leads to a stage of being "service enabled".

- *Basic Web services.*
  This implementation block contains activities from the following list:
  – Create services from tasks in new or existing systems.
  – Enable external Web Services using a gateway.
  – Integrate using "point to point".
  – Implement different technologies (J2EE, .NET, CICS, IMS, etc).
  – Implement static SOAP binding.
  – Build client stubs using WSDL.
  – Position internal Web services.
  – Use wrappers or adaptors.
  – Enable connectivity to packaged applications (CRM, ERP, Supply Chain).

– Enable connectivity to legacy systems and data.

6.1.2 Stage two - “service integration” implementation blocks

The following implementation blocks and their associated tasks lead to a stage of being “service integrated”.

- *Enterprise Service Bus exploitation.*
  This implementation block contains activities from the following list:
  – Enable integration of applications and business processes across the enterprise.
  – Introduce routing and transformation capabilities.
  – Introduce more advanced security and mediation.
  – Use message brokering and ESB.
  – Implement portal exploitation and page aggregation.
- *Advanced services adoption.*
  This implementation block contains activities from the following list:
  – Use Service Gateways.
  – Introduce more complex aggregations of services exposed across multiple applications.
  – Include transactional semantics.

6.1.3 Stage three - “process integration” implementation blocks

The following implementation blocks and their associated tasks lead to a stage of being “process integrated”.

- *Business services exploitation.*
  This implementation block contains activities from the following list:
  – Implement service oriented integration of business functions.
  – Expose coarse grained business services.
  – Implement provisioning and lifecycle.
  – Implement policy-based business services.
- *Business process orchestration.*
  This implementation block contains activities from the following list:
  – Introduce business process modeling.
  – Begin process choreography.
  – Implement external business rules.
  – Introduce flow and event management.
  – Implement compensation.

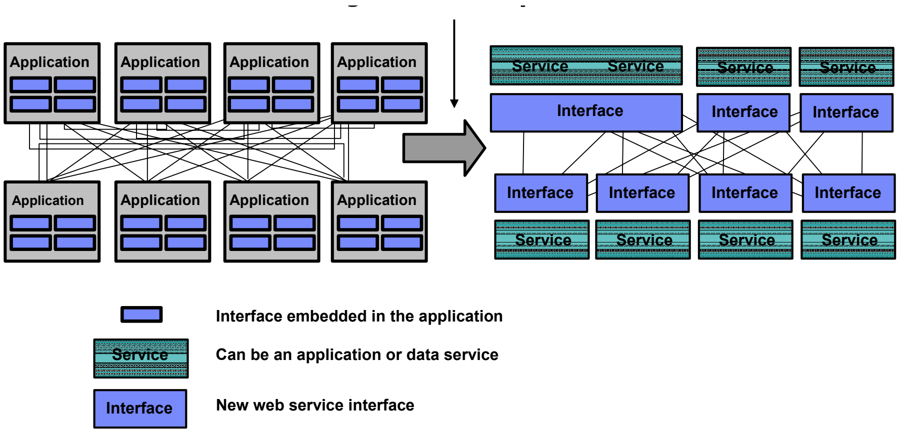
- *Discovery and dynamic binding.*
  This implementation block contains activities from the following list:
  – Locating services exposed on the ESB - use a service registry for discovery.
  – Introduce “consumer discovers supplier” patterns.
  – Implement dynamic consumption of WSDL.
  – Implement dynamic Invocation of service.

The implementation blocks mentioned determine the level of SOA maturity being achieved. Moving from one implementation block to the enxt one improves SOA maturity and brings the enterprise close to the full SOA benefits, asuch as agility, flexibility, business-IT alignment and time to market.

6.2 Stage one - “service enablement”

Most of the activities described under “basic Web services” must have been completed before moving into “service integration” and “process integration”.

To resume, in this stage we created lots of services out of existing tasks, packaged and legacy applications, and also used wrappers and adapters to improve them. We created client stubs out of WSDL definitions and integrated the client stubs into the service containers. We integrated “point-to-point” between consumers and providers to use (and reuse) the just created services. And we arrived at the following situation, depicted in the Figure 6-1 on page 223.



- The new interfaces (for web services) were created through wrappers, adapters or taking advantage of native web services support
- We have decoupled the interface from the service
- We have reduced the number and the complexity of the interfaces
- We made the services reusable (conditional of same transport protocol)

Figure 6-1 Situation of the enterprise after undergoing service enablement

6.3 Stage two - “Service Integration”

For the purpose of this book we define service integration as consisting of the following implementation blocks: *Enterprise Service Bus exploitation* (further discussed in 6.3.1, “Implementing the block “ESB exploitation”” on page 226) and *Advanced Web Services Adoption* (further dicussed in 6.3.2, “Implementing the block “Advanced Services Adoption”” on page 236).

We want to reach the loose coupling situation, therefore we have to break the point-to-point integration. We also want to increase reusability, but we cannot afford to continue to build an unmanageable number of logical and physical connections. There is no common place to store the definitions of these connections, so each application will have to have an increasingly bigger configuration file with static definitions for service providers. Of course, if we continue this way, we increase the administrative overhead. There is also no dynamics available, so it will not benefit the business. Figure 6-2 on page 224 shows the architecture we have at this moment.

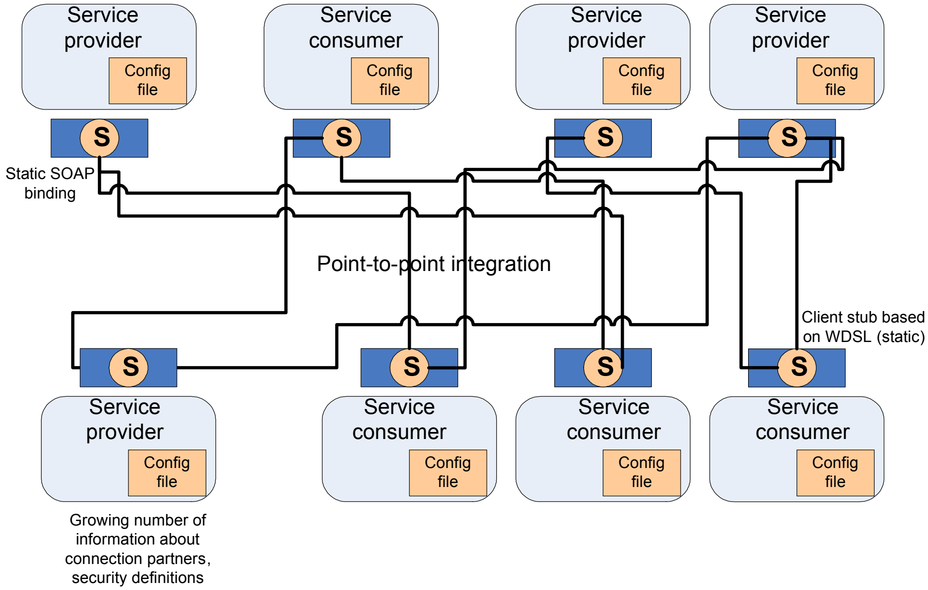


Figure 6-2 Services are there, but limitations abound

Attention: In several figures we will see the *letter S positioned in a circle*. This represents a piece of service implementation, which can be either a stub (proxy) on the client side, or a binding on the service provider side.

We want to also start aligning the IT services created in the previous stage with the business services. We have already seen that the "Improve" and "Adapt" transition approaches produce services (SOA-style), but they are very seldom aligned with the real business services. Since we are thinking in the future, we want to put at the disposal of the business analyst as many business services as possible, that he can manipulate at will in a secured environment.

That means that we have to identify these business services, and see that these are either newly created or composed out of existing IT services and some other pieces of infrastructure. Figure 6-3 on page 225 shows a business service (called "Service provider D") that contains pre-existing service primitives or parts of them (it contains the business logic of Service A, part of Service B, and part of Service C, all connected in a flow); we want to put the service interface on the business service so that it can be used (called) by processes (we think here also in the future). And in this way we start an intelligent way of reusing our "primitive" services.

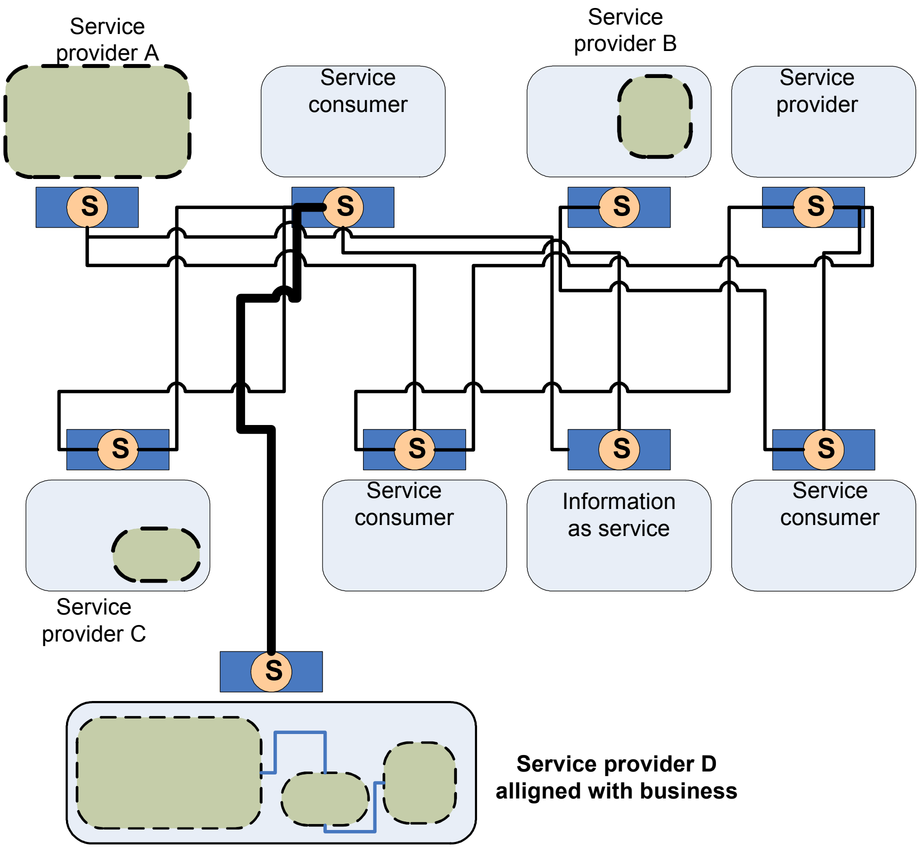


Figure 6-3 Service aligned with the business

And when doing all this we will not forget data, the 'forgotten" child of SOA. We'll remember that data as well should be exploited as "information", so we will start putting federation structures in place.

The clear conclusion of this discussion is that we have to put in place a sort of ESB, like the one shown on Figure 6-4 on page 226.

All service consumers see the ESB as the "proxy" for the services they might request. That's why we have the "S" {service) positioned on the contour of the ESB. All applications have now a very thin configuration file (containing the information about where to find the ESB). The ESB is taking care now of "calling" the right service. We just concentrated all connectivity logic in one place. At this early ESB stage all configurations entries are statically coded inside the ESB (through administrative tools).

Note: The figure shows a “generalized” ESB, some interfaces might not be supported by certain ESB products.

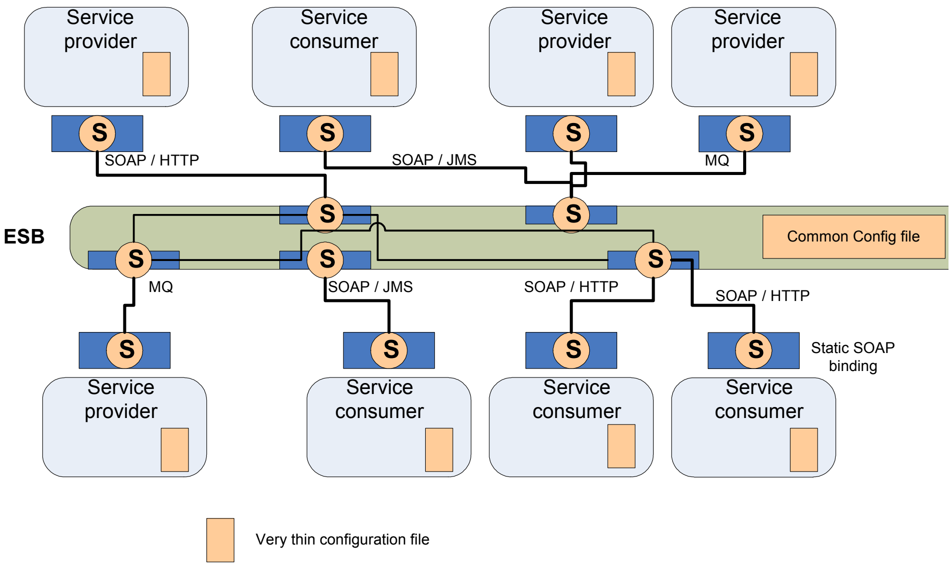


Figure 6-4 The ESB has really become part of the infrastructure

6.3.1 Implementing the block “ESB exploitation”

In this building block the main objective is to start exploiting an ESB. All applications that play on the “ESB” ground will need to reach loose coupling. The ESB will need to provide for support of multiple integration patterns, multiple transport protocols, usage of mediations, transformations and routing. Service consumers and producers should be able to speak different protocols and we should be no longer restricted to the SOAP protocol, both for the service consumer and requester. Our architecture looks now like in Figure 6-5 on page 227.

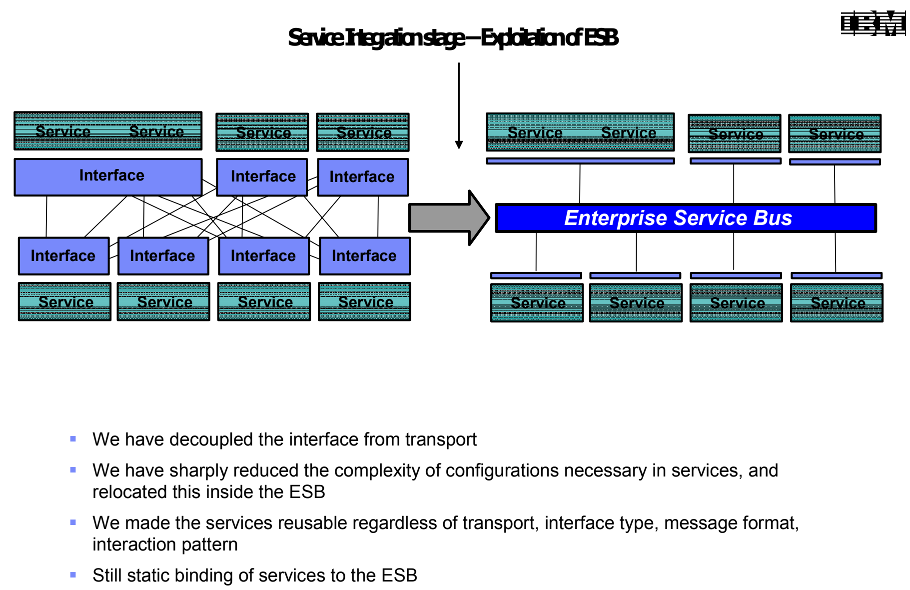


Figure 6-5 Status at the end of the “Exploitation of ESB”

By installing an ESB we implemented all the activities required by this building block. The degree of implementation can be seen in Table 6-1 on page 232; this is of course highly dependent of the features delivered by the ESB chosen. The only activity left for a short discussion is the *portal exploitation* and *page aggregation*. The reasoning behind this is (beside the advantages given by a portal itself) to allow the portlets - as service consumers - to be integrated into the SOA architecture and to make the services available to the portlets. Now, with the advent of the ESB, we can see the following architecture, shown in Figure 6-6 on page 228

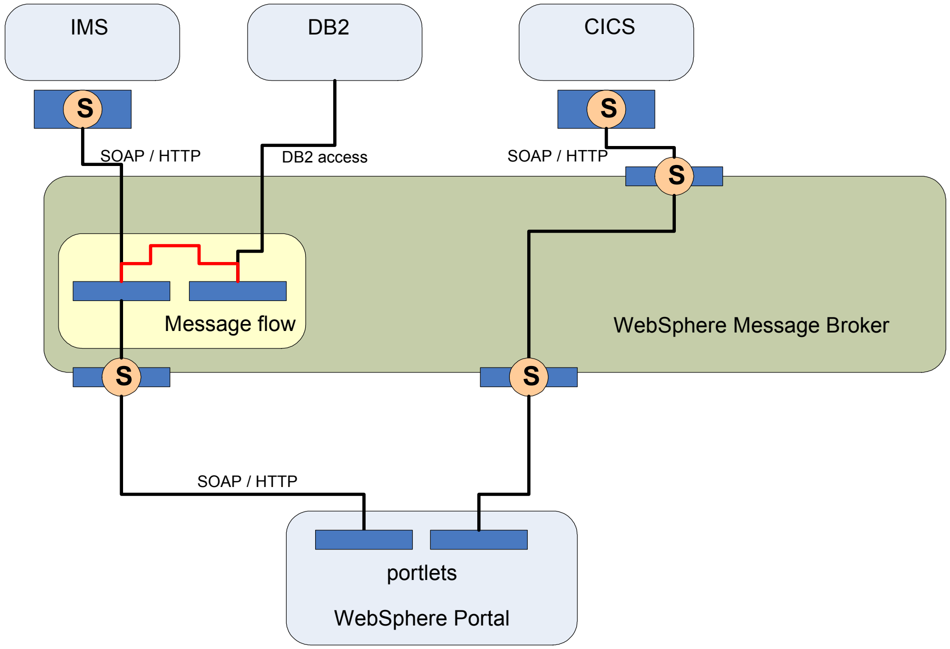


Figure 6-6 Portlets enable presentation integration through ESB

As you can see in Figure 6-6, portlets can access the message flows running inside WebSphere Messgae Broker (through SOAP/HTTP), but can also reach directly all SOA-enabled services (through the broker function). The architecture enables the portal (located in the IBM SOA reference architecture - Interactive Services) to aggregate information coming from different services.

What are the capabilities of an ESB?

First, let us see what an ESB should be able to do. There are many opinions about what an ESB should contain and which features are required at which stage. The main functions are, of course, *routing, transformation, transport* and *switching*. But there are degrees of detail. We decided to use the terms ESB basic and ESB advanced, which we will define next.

The *basic capabilities* should be:

- Support high volumes of interactions.
- Allow for centralized management, administration and control.
- Support various interaction patterns (message-oriented, event-driven, synchronous request/response, asynchronous fire and forget, etc).
- Provide for mediations - resolve differences between the applications attached to it.
- Allow connection independence (transport neutrality).

- Handle routing requests.
- Perform protocol transformations.

The *advanced capabilities* should be:

- Implement a service directory (in order to allow consumers to locate services exposed on the ESB).
- Support multiple networks and protocols.
- Integrate databases.
- Support application adapters.
- Support language Interfaces (Java, C++, etc).
- Implement security (authentication, authorization, encryption) for the services exposed on ESB.
- Provide message transformations, message enrichment, message and service aggregation.
- Support logging.
- Implement advanced management, monitoring, administration and policy-driven service-level management.
- Implement business rules.

All the above mentioned functions are differentiating elements between products. Some of the functions will be implemented in an ESB-type of product, some in other layers (for example, business rules are implemented in IBM WebSphere Process Server).

Available ESB options

There are several options of building an ESB on the z/OS platform:

Option 1 - a “minimal” ESB based on WebSphere MQ

In this case the ESB has only the possibility of connecting services that “speak” MQ or that have adapters that “speak” MQ. Any service consumer that produces SOAP messages will have to put them inside MQ messages to be transported through the ESB. At the other end, a service provider will retrieve the MQ message and separate and process the SOAP envelope.

Any mediation, data transformation, or routing function must be specifically coded by the developer (without specific tooling) and deployed in the runtime (as triggers and exits). There is no support for protocol transformation.
Services are called “implicitly” by consumers through dropping messages on their corresponding input queue; the replies are picked from the output queue.

This type of implementation does not fulfill the basic ESB characteristics.

Option 2 - A "light" ESB based on WebSphere Application Server Service Integration Bus (SIB)

The *Service Integration Bus (SIB or SIBus)* can be seen as a communication structure inside of WebSphere Application Server that can be used by J2EE applications to exchange messages using the JMS APIs. The SIB is part of WAS since Version 6. The features of the SIB are:

- Provide managed communication for synchronous, asynchronous and event-based messaging.
- Can be expanded through a "link" (simply a connection definition to a messaging engine on another SIB, or an external WebSphere MQ Queue Manager).
- This ESB will deliver for the service consumers a reduced amount of connectivity options, specifically those allowed by the J2EE platform. Supported application attachments are:
  - Web services:
    - Requestors can use JAX-RPC API.
    - Providers run in WebSphere Application Server as stateless session beans and servlets (JSR-109).
    - Requestors or providers can attach via SOAP/HTTP or SOAP/JMS.
  - Messaging applications:
    - Inbound messaging using JFAP-TCP/IP (or wrapped in SSL for secure messaging). JFAP is a proprietary format and protocol used for service integration bus messaging providers.
    - MQ application in an MQ network using MQ channel protocol.
    - JMS applications in WebSphere Application Server V5 and v6.1.
    - MQ client protocol.
  - Message Driven Beans (MDBs):
    - With EJB 2.1, Message Driven Beans (MDB) in the application server that listen to queues and topics are linked to the appropriate destinations on the service integration bus using JCA connectors.

The service providers themselves can be reached through the connectivity allowed by WebSphere Application Server, either internally as Web services, or externally as CICS/IMS/MQ SOAP-enabled services (through the SOA enablements described in the Chapter 5, "SOA implementation scenarios" on page 129), or through WebSphere Adapters to packaged applications.

Any mediation, protocol/data transformation, or routing function must be specifically coded by the developer and deployed in the runtime (mediation handlers). That means that if a service consumer speaks SOAP/HTTP, and the service provider MQ, a protocol transformation piece of code has to be written and deployed. The same is valid if some data transformation is necessary between incompatible formats. The “wiring” between service consumer, mediations and service provider is static (administrative configuration).

This ESB fulfills some basic characteristics. We see in this “light” ESB some limitations, but it might be a valid option for an enterprise who does not need the features delivered by WebSphere ESB or WebSphere Message Broker out-of the-box.

Option 3 - a “full service ESB”

There are two implementations: WebSphere ESB (WESB) and WebSphere Message Broker (WMB). They are both full service ESBs because they implement the basic functionality (transport services, mediation services, routing services, event services) and much more.

WebSphere ESB has a limited amount of connectivity options, which impact the types of service consumers and providers that can participate. This limitation, though, might be irrelevant in an enterprise that has mostly J2EE applications. The WebSphere Message Broker has its strong points in the areas of universal connectivity and data transformation.

Table Table 6-1 on page 232 compares WebSphere ESB and WebSphere Message Broker, among others, from the point of view of connectivity and data transformations.

Selection criteria for an ESB

Table 6-1 on page 232 compares the two IBM ESB products on several SOA-related criteria.

> **Important:** Note that the specifications of the IBM ESB products are very time sensitive. Functions and levels of specifications may change rapidly. We recommend to always confirm the latest specs on the Web sites when making a decision.

Table 6-1 Comparison between WebSphere ESB and WebSphere Message Broker

Function	WebSphere Enterprise Service Bus V6.0	WebSphere Message Broker V6.0
Connectivity	► TCP/IP, SSL, HTTP(S), IIOP ► JMS V1.1 (point-to-point, pub/sub) ► JMS/MQ (using MQLINK configuration)	► TCP/IP, SSL, HTTP(S) ► JMS V1.1 (point-to-point, pub/sub) ► Native WebSphere MQ ► Supports WebSphere MQ Transport, WebSphere MQ Everyplace Transport, Multicast Transport, Real-time Transport, SCADA Transport, Web Services Transport, JMS Transport ► CICS, VSAM using SupportPacs ► Files using WebSphere Message Broker File Extender
Web services support	► SOAP/HTTP(S), SOAP/JMS, WSDL 1.1 ► Supports WS-I Basic Profile V1.1 ► UDDI V3.0 Service Registry ► WS-Security, WS-Atomic Transactions ► Client support: J2EE client, Message client for C/C++ and .NET, Web services client	► SOAP/HTTP(S), SOAP/JMS, WSDL 1.1 ► Supports WS-I Basic Profile V1.0 ► Client support: JMS client, Message client for C/C++ and .NET, Web services client, MQI client
Adapter support	► WebSphere Adapters and WebSphere Business Integration Adapters[a]	► WebSphere Business Integration Adapters[b]
Mediation programming model	► WebSphere ESB has a programming model based on Service Component Architecture (SCA)and SMO (SDO plus message header plus context).	► WebSphere Message Broker supports the use of ESQL, numerous Nodes (Compute Node, Database Node, Filter node, JavaCompute node, native JMS Input/Output node, HTTP Node, Filetransfer node, etc.) and flows.
Message logging	► Provides prebuilt mediation primitives for message logging	► Provides prebuilt message flow nodes for message logging

Function	WebSphere Enterprise Service Bus V6.0	WebSphere Message Broker V6.0
Message transformation	► Protocol transformation between HTTP, JMS, IIOP ► Custom transformation logic can be implemented in Java, XSLT ► Supports transformation of XML, SOAP, JMS message data format (many more if used with adapters)	► Protocol transformation between any protocols available as input or output nodes (HTTP, JMS, MQ, and more) ► Custom transformation logic can be implemented in Java, ESQL, or XSLT ► Supports transformation of self-defined messages (XML), built-in predefined messages (SOAP, MIME, and more), and custom predefined messages (MRM)
Message routing	► Content and transport/protocol based routing ► Provides prebuilt mediation primitive for message routing, or custom build mediation using Java ► Supported through SCA	► Content and transport/protocol based routing ► Custom routing logic can be implemented in Java or ESQL
Database access	► Built-in database lookup mediation primitive	► Built-in nodes for database access (Datanode ESQL, Java, graphical mapping)
Validation	► Validation of the input message against its schema by configuration of primitives	► Validation of input and output message against its schema definition.
Event-driven processing	► Supports event-driven processing by leverage adapters for capture and dissemination of business events	► Supports complex event processing (processing of events formed by several earlier ones)
Security	► HTTPS support ► Authentication and authorization as part of J2EE ► Support for WS-Security	► HTTPS support ► Authentication and authorization by the operating system environment
Quality of service	► Assured delivery support by service integration bus ► Transaction support provided by WebSphere Application Server ► Configurative within SCA module components	► Assured delivery support by WebSphere MQ ► Transaction support by WebSphere MQ (limited for JDBC connections) ► Configurative within node properties

Function	WebSphere Enterprise Service Bus V6.0	WebSphere Message Broker V6.0
Integration styles supported	► Request/reply, Publish/subscribe	► multiple integration styles (request/reply, publish/subscribe and others)
Management	► High availability and scalability provided by WebSphere Application Server environment ► Built-in administration tools as part of the WebSphere Admin Console ► Import bindings can be modified using the WebSphere Administration Console ► Common Event Infrastructure (CEI) support. Entry, exit and failure events can be activated on all SCA components within the mediation modules ► Common Base Event browser for viewing events from the CEI	► A high level of availability can be achieved using multiple brokers in combination with WebSphere MQ clustering and WebSphere MQ queue sharing. Queue sharing is a unique feature of WebSphere MQ on z/OS ► Built-in administration tools

a. Not all adapters are available for the z/OS platform.
b. Not all adapters are available for the z/OS platform.

If we consider the two full ESB implementations, and we have the task of selecting, we can use following short comparison.

WebSphere Enterprise Service Bus

WebSphere Enterprise Service Bus (WESB) is designed to provide the core functionality of an ESB for a predominantly Web services based environment. It is built on WebSphere Application Server, which provides the foundation for the transport layer. If the user has a lot of Web services in its environment and applications are primarily J2EE based, WebSphere Enterprise Service Bus is likely to be the better product to use. Building an ESB that is based entirely on WebSphere Enterprise Service Bus is an option when Web services support is critical and the service provider and consumer environment is predominantly built on open standards. However, if integration with non-Web service standards-based services is a major requirement or there is very heteregenous landscape of applications, then WebSphere Enterprise Service Bus might not be the right choice.

WebSphere Message Broker

WebSphere Message Broker (WMB) provides a more advanced ESB solution with advanced integration capabilities such as universal connectivity and any-to-any transformation for data-centric deployments. It can handle services integration as well as integration with non-services applications. WebSphere MQ provides the transport backbone for messaging applications. Typically, customers who need a higher performance and throughput product in a message-centric environment would use WebSphere Message Broker.

One interesting benefit available with WebSphere Message Broker is the possibility of exposing Message Flows as Web services (therefore functioning as service provider) as well as Message Flows calling Web services (as service consumer). The implementation is done through the combination HTTP input node / HTTP reply node. This architecture is shown in Figure 6-7.

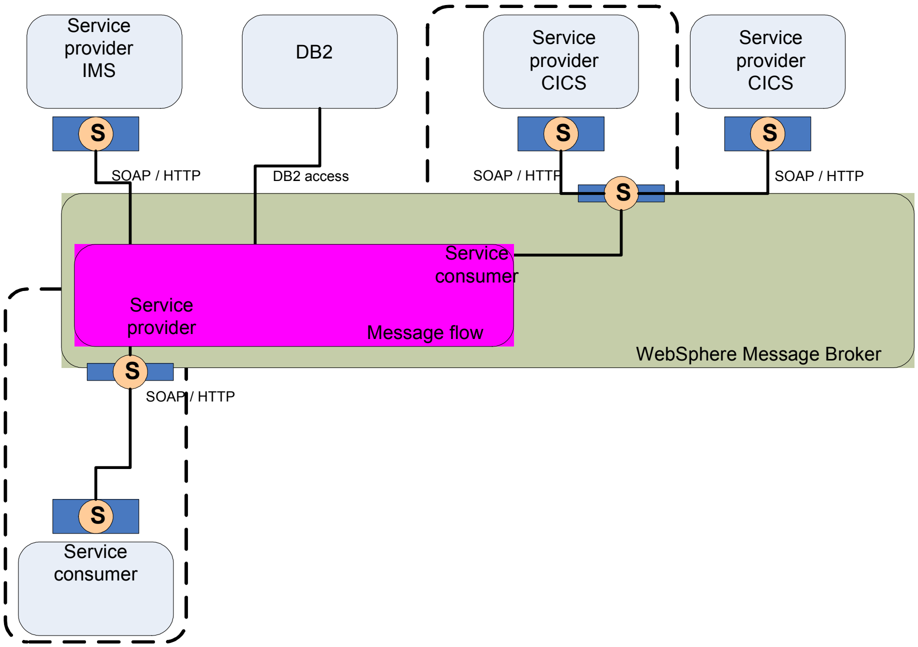


Figure 6-7 Message flow with two roles: service provider (towards client) and service consumer (towards IMS, DB2)

Here are a few typical WebSphere Message Broker Web services scenarios, and we notice the role played by WMB:

- Make use of a Web service (consumer role)
  - Message Flow invokes a Web service to retrieve data or perform a function (for example it can invoke a CICS Web service).
- Provide a Web services “front end” to an existing application (service-enables indirectly the application).
  - Message Flow called by Web services client invokes an MQ enabled COBOL program and returns result to client.

- Route a Web services request (just as a proxy)
  - Message Flow forwards client request to Service Provider based on message content. Reply from Provider returned to flow and passed back to client.
- Serve as Web services aggregator (consumer and business logic implementation)
  - Message Flow makes requests to other Web services and aggregates their replies into a single response.

You can find more information about WebSphere ESB on z/OS at:

`http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wsps.ovw.doc/welcome_wps_ovw.html`

You can find more information about WebSphere Message Broker at:

`http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.websphere.wesb.doc/tasks/twesb_inst.html`

6.3.2 Implementing the block "Advanced Services Adoption"

In this section we discuss the activities contained in the SOA implementation block *Advanced Services Adoption*, and show how to implement them.

Use of a Web Service Gateway

We would like to have in our SOA infrastructure a point of control where all service requests are coming in and get mapped to service providers. We would also like to be extremely flexible, an this can be reached if the target location, message format and transport protocol are shielded from the service consumer.

Some usages for a Web Services Gateway are:

- Externalize an internal Web service (making it available for external service consumers, and in the process, secure it).
- Internalize an external Web service (making an external Web service accessible to service consumers as if were an internal Web service)
- Protocol transformation.
- SOAP Proxy service.
- Process abstraction: the service invocation approach must be flexible enough to cope with events such as switching frequently between external providers of a similar service without requiring changes to the application.
- Increased flexibility: a service provider needs the flexibility to change the deployment infrastructure without notifying all the service requestors.

- Single deployment of a Web service to multiple end point listeners.
- Configurable intermediation through mediations.
- Authentication and authorization support (per service and/or operation) using basic User ID/Password and SSL support.

The *IBM Web Services Gateway* is a runtime component that provides configurable mapping between Web service providers and requesters. Services defined with WSDL can be mapped to available end point listeners. The IBM Web Services Gateway is included with IBM WebSphere Application Server Network Deployment V6. It is based on the Service Integrated Bus (SIB) and is tightly integrated with WebSphere Application Server.

The basic components of the Web Services Gateway are:

- End point listeners that define the entry-points into the gateway and carry the Web service request and response through the gateway.
- Mediations that are used to intercept service invocations which come into the gateway and act upon the services.
- Services that are described with the help of a Web Services Description Language (WSDL) document.
- UDDI references to manage the publishing of an exposed Web service to a private or public UDDI registry.

Figure Figure 6-8 shows the basic components of the Web Services Gateway.

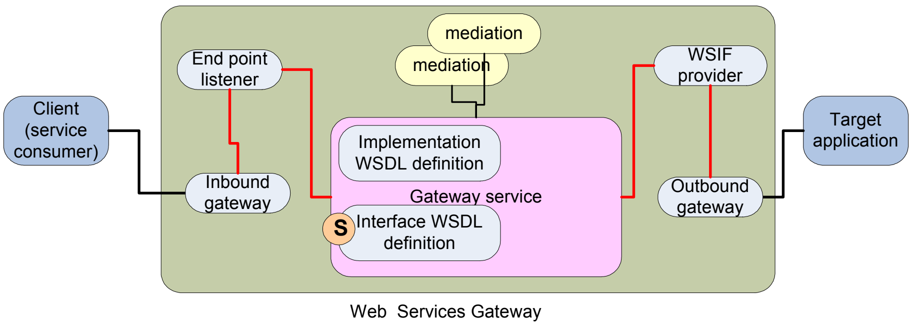


Figure 6-8 Basic structure of the Web Services Gateway

The entry point to the gateway is defined by an *End Point Listener (EPL)*. An EPL is a piece of software that defines the protocol you can use to access the gateway. The incoming message is assessed on arrival through the EPL to determine which service is required. Each service (defined in a WSDL document) has to be bound to one or more EPLs. One or more mediations can be bound to a service for manipulating both request and response messages.

The WSDL service definition specifies the provider service interface and implementation used to access the target service. A request to the Web Services Gateway arrives through an EPL and is translated into an internal representation of the service. With the help of mediations for the request, a request can be logged, intercepted, or generally manipulated. After this an appropriate provider is used to communicate with the target service. The provider in the gateway acts as a client for the target Web service. The response from the target service flows along the exact same path back to the provider.

The process of deploying a target service into a gateway EPL generates two different *external* WSDL files; an implementation definition and an interface definition. These new WSDL files can be exported for use by client applications, and are the externalization of the service capabilities offered by the *internal* target service. The implementation WSDL definition is used to simplify the connection process for a client, particularly when dynamic invocation is being used. Having obtained the implementation definition, the client can then access the WSDL interface definition produced by gateway, which provides full information about the target service (as presented externally by the gateway). The IBM Web Services Gateway uses the *Web Services Invocation Framework (WSIF)* API from Apache to decouple invocation from deployment within the Web Services Gateway.

The following features can be configured at runtime:

- End Point Listeners (EPLs)
- JAX-RPC handlers
- SIBus mediations
- Gateway and proxy services
- UDDI lookup and publishing
- WAS and Web services security

For more information please consult the following:

- "An introduction to Web Services Gateway", by Chandra Venkatapathy and Simon Holdsworth

  `http://www-128.ibm.com/developerworks/library/ws-gateway`

- Web services gateway at InfoCenter

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/sibuswsgw/index.html`

Complex aggregation of services across multiple applications

Aggregation is the ability to decompose a service request into multiple outbound requests and re-assemble the replies to send an aggregated response.

Aggregation is supported in the WebSphere Message Broker through built-in *nodes* (`AggregateControl`, `AggregateReply`, `AggregateRequest`, `MQGet`).

Currently aggregation is not supported in WebSphere ESB.

Aggregation of Web services is supported through the BPEL runtime in WebSphere Process Server. A discussion of the powerful control flow features offered by BPEL is out of the scope of this book. The implementation of the BPEL language in the WebSphere Process Server is described in WebSphere Process Server V6.0 Business Process Choreographer Programming Model, at:

`http://www-306.ibm.com/software/integration/wps/library/infocenter/`

For more information please consult:

- “Business Process Choreography in WebSphere: Combining the power of BPEL and J2EE”, by M. Kloppmann et al., at

  `http://researchweb.watson.ibm.com/journal/sj/432/kloppmann.pdf`

Transactional capabilities

We will discuss now the transactional capabilities in the WebSphere Message Broker and in the WebSphere Process Server.

WebSphere Message Broker supports transactional message flows. The *Transaction* property can be set for specific nodes, with different consequences. The Transaction property can be set to:

- *Automatic* - means any updates, deletions, and additions performed by the node are committed or rolled back when the message flow processing completes. If the message flow completes successfully, all changes are committed. If the message flow does not complete successfully, all changes are rolled back. In order to coordinate all processing done by the message flow we select this value.
- *Commit* - the action taken depends on the system to which the message flow has been deployed. On distributed systems, any work that has been done to this data source in this message flow to date, including any actions taken in this node, is committed regardless of the subsequent success or failure of the message flow. On z/OS, actions taken in this node only are committed regardless of the subsequent success or failure of the message flow. Any actions taken before this node under automatic transactionality are not committed, but remain within a unit of work and might either be committed or rolled back depending on the success of the message flow

There are a number of nodes where we can set the *Transaction* property. Among them are: `Compute` Node, `Database` Node, `DataDelete` Node, `DataInsert` Node, `MQInput` Node, `MQOutput` Node, `MQReplay` Node, `JMSInput` Node, `JMSOutput` Node.

For WebSphere Message Broker on z/OS transactions are always globally coordinated (that means the property *coordinatedTransaction* is always on). Coordination is provided by *Resource Recovery Services (RRS)*.

WebSphere Process Server supports transactional process flows. The interaction between a service client and a service is governed in the runtime by *service qualifiers*. Service qualifiers are quality of service specifications that define a set of communication characteristics required by an application (like, among others, transaction management and security level). An application communicates its quality of service needs to the runtime environment by specifying service qualifiers (as metadata to the service). Quality of Service qualifiers can be specified on a SCA reference, interfaces and implementation. WebSphere Process Server processes rely upon the underlying WebSphere Application Server capabilities for transaction, security, and workload management. For business processes, WebSphere Process Server supports transactions involving multiple resource managers using the two-phase commit process to ensure atomic, consistent, isolated, and durable (ACID) properties. This capability is available for both short-running flows (single transaction) and long-running flows (multiple transactions). Inside a business process multiple steps can be grouped into one transaction by modifying transaction boundaries in WebSphere Integration Developer.

6.4 Stage 3 - "process integration"

We have just finished the two stages that encompass :service enablement" and "service integration". Figure 6-9 on page 241 shows the activities done and the new pieces added to the SOA infrastructure.

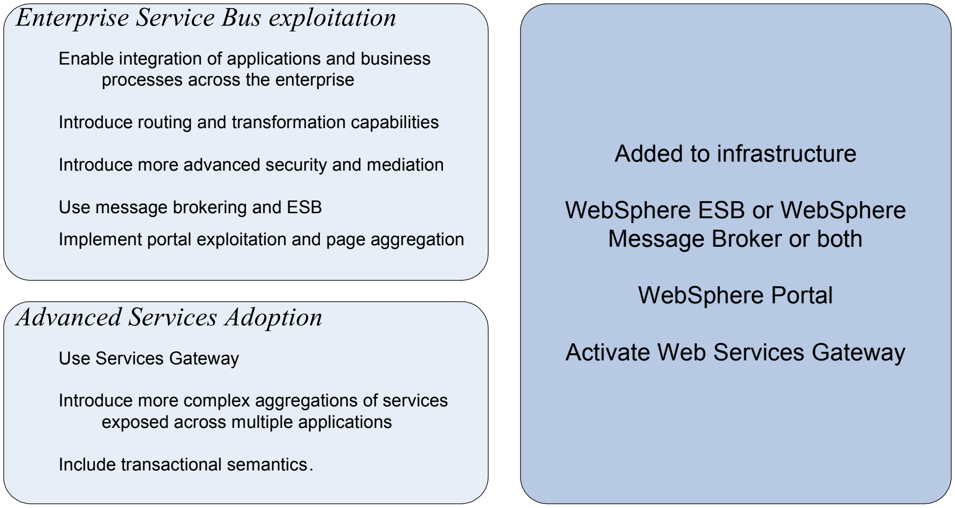


Figure 6-9 End of the "service integration" stage

Our architecture is shown in Figure 6-10 on page 242. It is now time to take advantage of the SOA-enabled services and of the architecture that enable their transparent communication, start aligning IT services to the business and start creating new business processes. But first, what is *process integration*? One definition might be the following: *process integration* is the creation and management of the logic that links applications and services together to implement a business function.

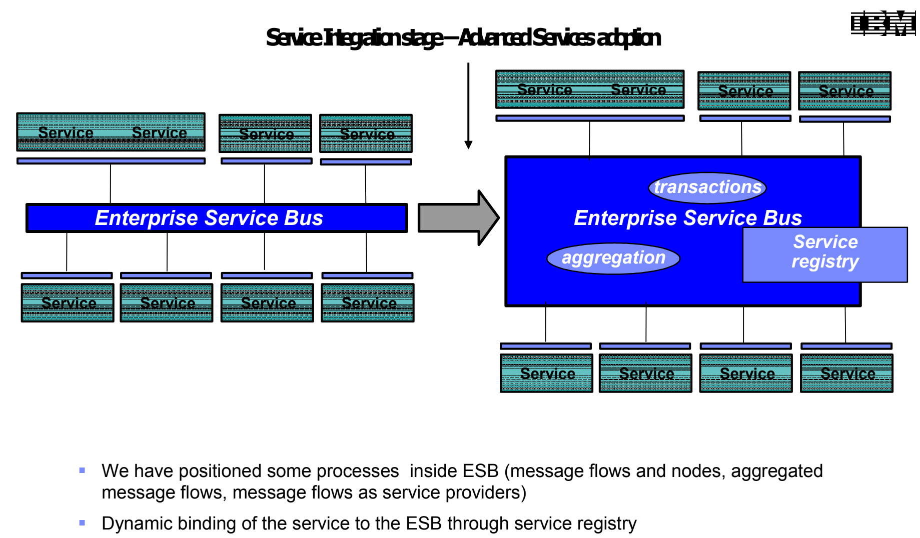


Figure 6-10 Status after the second stage of "Service Integration"

6.4.1 Implementing the block "Business Services Exploitation"

In the following sections we will discuss the implementation of the building blocks associated to process integration.

Implement service-oriented integration of business functions

One possibility was shown in Figure 6-7 on page 235, with message flows functioning as service providers. In this example a business service was created using existing "primitive" services (with some business logic located in CICS, IMS, and with some data access from DB2). Considering the many number of nodes of different types made available by WebSphere Message Broker (e.g. Java Compute Node, JMS Nodes, CICS Node, Database Node, HTTP node and many File Nodes) we have a flexible way of creating business services.

Another possibility is to use the decomposition tools (such as WebSphere Studio Asset Analyzer) to identify pieces of the existing applications that fit the service definition, and chain them together in message flows, thus creating business services.

Yet, another possibility is to use process choreography and orchestration tools and technologies to link services into composite applications, fully aligned with the business; this option is possible when installing the WebSphere Process Server and using WebSphere Integration Developer (WID). The functionality of the WebSphere Process Server will be described in the section “6.4.2, “Implementing the block “ Business Process Orchestration”” on page 244 “.

Expose course grained business services

Using either WebSphere Message Broker or WebSphere Process Server we can create new business services (either completely new or a combination with existing services). From a business point of view it makes sense to model most processes as course grained services, which happens to correspond with the slight technological advantage they enjoy over the fine grained services (less calls, less network traffic, less object marshalling/demarschalling).

Implement policy-based business services

Policies are simply sets of “if-then” business decisions that can be simply expressed and can be used by several applications in an enterprise.

The idea behind this activity is to increase the agility of the application by separating the business rules from the application. This “decoupling” allows easy changes in the behavior of the application, through changes in the business rules.

One possibility is to use the ESB and place the policies inside a message flow (for example as a Java node). The other option is to implement the policies through WebSphere Process Server. In the WebSphere Process Sever, business rules are assembled into Business Rules groups. They are exposed in the runtime as an SCA component. The SCA component can be accessed by multiple processes.

Implement provisioning and lifecycle

Provisioning and *lifecycle* refer to the activities necessary for the management of a Web service starting with the development phase, over deployment, usage and concluding with retirement. These activities are strongly related to the service registry and repository. The registry and repository contains information that is used during the whole lifecycle of the service. IBM’s product for this functionality is the *WebSphere Service Registry and Repository (WSRR)*, and the strategy is that for all phases of the service lifecycle IBM products will be integrating with WSRR.

During the lifecycle of the service there are several interaction points with WSRR. During creation of the service there is a need for the developer to know what services are available and how to call them. Therefore his development tools need integrated access to WSRR (repository function).

At runtime the mediation components of the ESB (both WebSphere ESB and WebSphere Message Broker) will need interaction to WSRR in order to perform service selection, enforce service policies, etc. (registry function).

Figure 6-11 shows how different components of the IBM SOA reference architecture (and from all lifecycle parts) interact with WSRR.

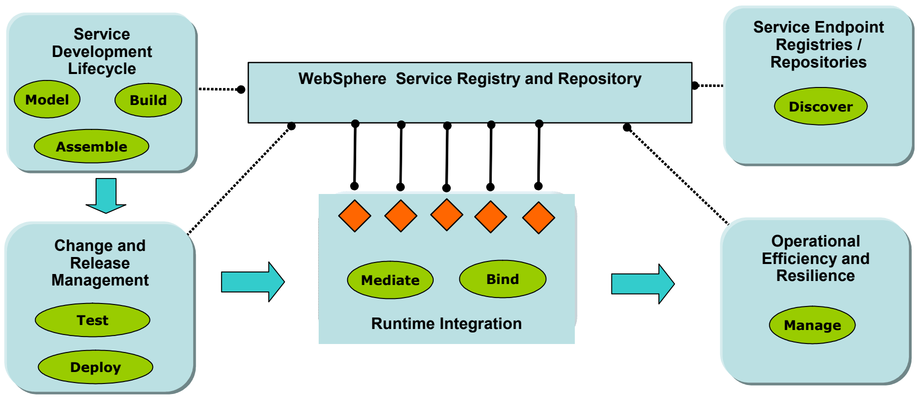


Figure 6-11 SOA service lifecycle and interactions with WSRR

6.4.2 Implementing the block “ Business Process Orchestration”

As defined, the implementation block “Business Process Orchestration” contains the following activities:

- Introducing business process modeling
- Beginning process choreography
- Implementing external business rules
- Introducing flow and event management
- Implementing compensation

All these activities can be implemented on z/OS through the installation of WebSphere Process Server (as runtime), WebSphere Business Modeler and WebSphere Integration Developer (both as development tools) and WebSphere Business Monitor as a management tool. To begin with, let’s look at the structure of the WebSphere Process Server.

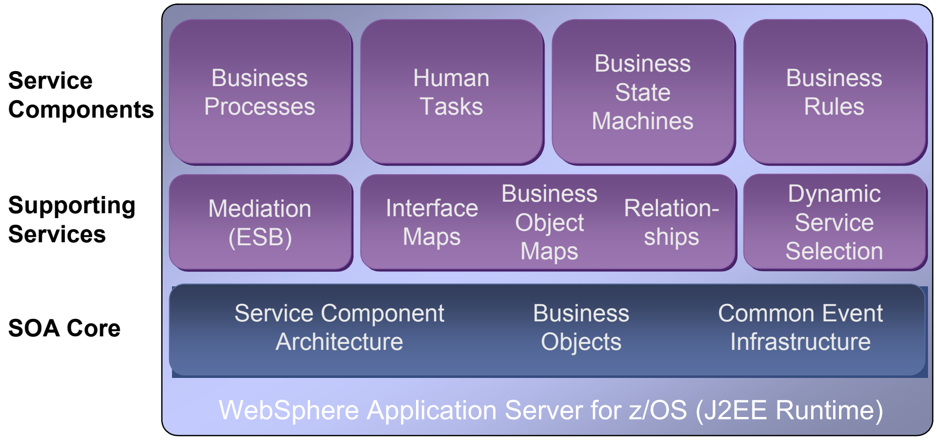


Figure 6-12 Architectural model of WebSphere Process Server

WebSphere Process Server is implemented on top of WebSphere Application Server and WebSphere ESB. WAS provides the J2EE and Web services runtime and WebSphere ESB provides the ESB functionality. The SOA core layer in WebSphere Process Server consists of:

- Service Component Architecture (SCA)
- Business Objects
- Common Event Infrastructure (CEI)

On top of this SOA core layer lies the service components and supporting services layers. WebSphere Process Server implements a number of components and services that can be used in an integration solution. In the service components layer, you find the following:

- Business processes

  The business process component in WebSphere Process Server implements a WS-BPEL compliant *process engine*. WS-BPEL is a language that allows specifying the behavior of business services:

  – behavior between Web services
  – behavior as Web service

- Human tasks

  Human tasks in WebSphere Process Server are stand-alone components which can be used to assign work to employees.

- Business state machines

  A *business state machine* provides a way of modeling a business process by representing them based on states and events.

- Business rules

  Business rules are a means of implementing and enforcing business policies through externalizing business functions. This enables dynamic changes of a business process.

These components can use the features of a number of supporting services in WebSphere Process Server. Most of these can be classified as some form of transformation. There are a number of transformation challenges when connecting components and external services, each of which is being addressed by a component of WebSphere Process Server:

- Interface maps

  Very often interfaces of existing components match semantically but not syntactically. *Interface maps* allow the invocation of these components by translating these calls. Additionally business object maps can be used to translate the actual business object parameters of a service invocation.

- Business object maps

  A *business object map* is used to translate one type of business object into another type of business object.

- Relationships

  In business integration scenarios it is often necessary to access the same data in various backend systems for example an ERP system and a CRM system. A common problem for keeping business objects in sync is that different backend systems use different keys to represent the same objects. The *relationship* service in WebSphere Process Server can be used to establish relationship instances between objects in these disparate backend systems. These relationships are accessed from a business object map when translating one business object format into another.

- Dynamic service selection

  A selector component allows dynamic selection and invocation of different services, which all share the same interface.

- Mediation

  This component is inherited from WebSphere Enterprise Service Bus.

The primary development tool for WebSphere Process Server is WebSphere Integration Developer (WID). This is the same tool used for WebSphere Enterprise Service Bus development tasks.

You can find more information about IBM WebSphere Process Server V6 at:

`http://www.ibm.com/software/integration/wps/`

The important facts are that both WebSphere Process Server and WebSphere ESB share the same programming model, based on data representation as *Service Data Objects (SDO)*, invocation based on *Service Component Architecture (SCA)*, and composition based on BPEL+ extensions. The focus of the programming model is on assembling solutions rather than implementation details. A number of adapters is available to enable communication with EIS and non SOA-enabled legacy systems. Supported by tooling, it is clear that the two products deliver, based on the common basis, a coherent part of the SOA infrastructure. Since WebSphere ESB is the routing and transformation basis for WPS it is obvious that the SCA components will communicate without problems with SOA-enabled services reachable over WESB. These are mostly Web services, so if the business process requires just these, we can stay with the combination WPS and WMB.

Figure 6-13 shows the connectivity available to SCA components running in WebSphere Process Server and WebSphere ESB.

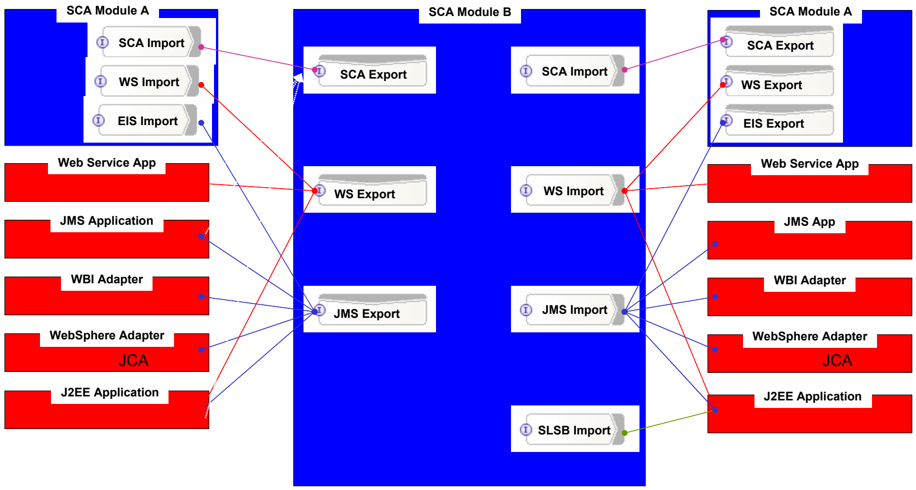


Figure 6-13 SCA connectivity to other components and services

The WebSphere adapters supported by the SCA components are:

- WebSphere Flat File adapter
- WebSphere JDBC Adapter
- WebSphere PeopleSoft Enterprise adapter
- WebSphere Siebel® Business adapter
- WebSphere SAP adapter

A more interesting aspect is the integration between WebSphere Process Server and WebSphere Message Broker. WMB can consume WPS XML generated messages and can generate WPS consumable messages, using several interfaces. The integration is shown in Figure 6-14.

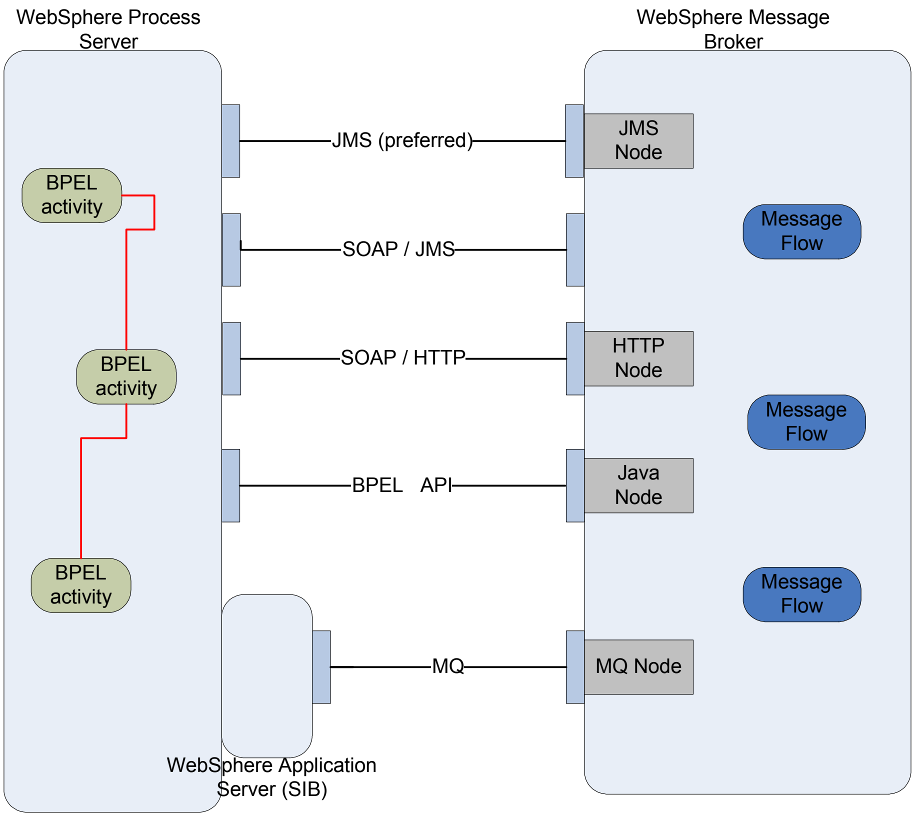


Figure 6-14 Integration between WebSphere Process Server and WebSphere Message Broker

Since WebSphere Message Broker supports a wealth of interfaces for non- Web services applications, we see the need to present some scenarios where each product applies its strength to implement the solution.

The first scenario is shown in Figure 6-15 on page 249. In this scenario an WebSphere MQ service consumer accesses WMB, where the necessary routing and transformation takes place (the strengths of WMB). When a human interaction is necessary the message flow reaches for WPS, which does its part. In this scenario WebSphere Message Broker controls the flow.

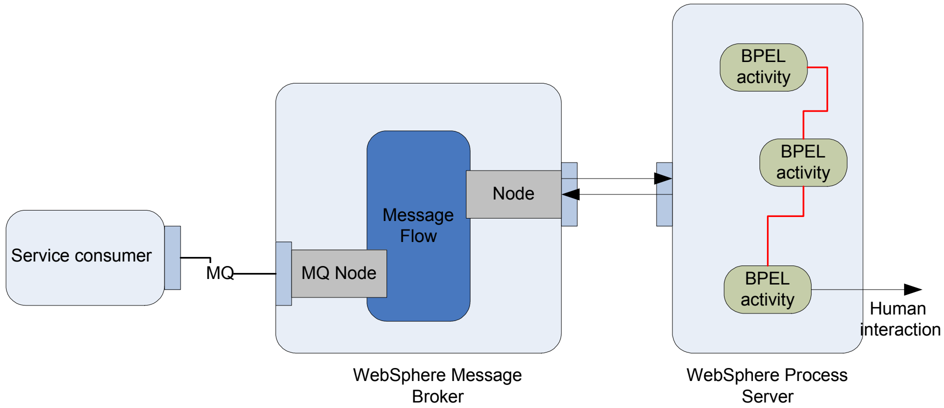


Figure 6-15 WebSphere Message Broker controlling the flow

The second scenario, shown in Figure 6-16, presents a situation where the WebSphere Message Broker has to handle multiple transports, transformations, message splitting (which it can do very well), and then start a process with several threads. This is better done by WebSphere Process Server.

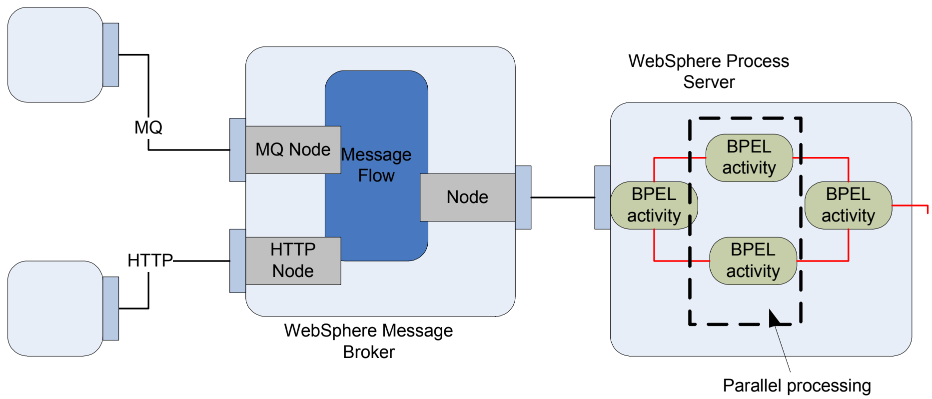


Figure 6-16 WMB gives control to WPS for simultaneous processes

The last scenario, shown in Figure 6-17 on page 250, describes a situation where WPS executes the business process, choreographs the Web services, and drives WMB to access messaging applications. WMB then brokers the reply message to WPS, which continues to run the process.

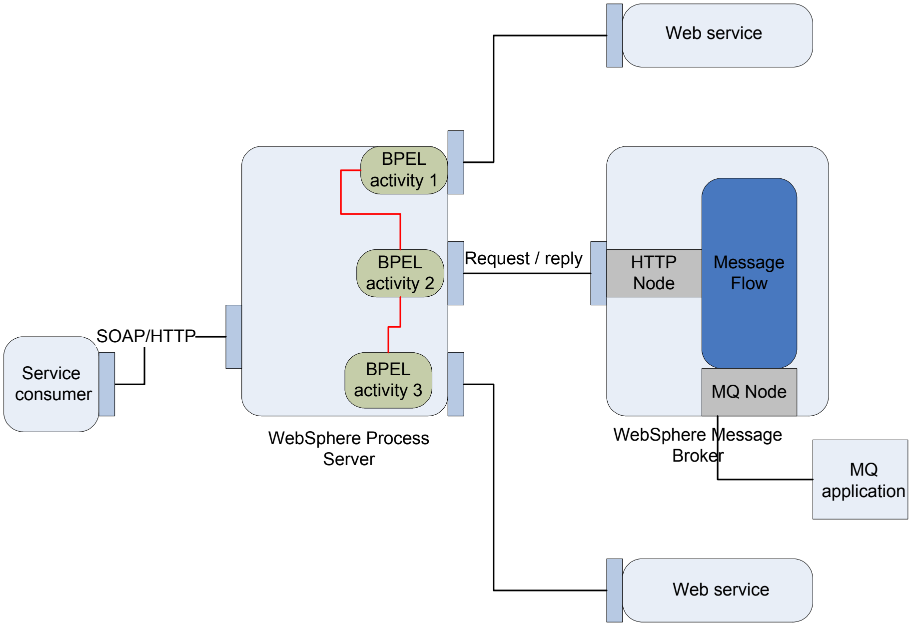


Figure 6-17 WPS as choreographer of services, using WMB for access to messaging application

We can generally recommend to use WebSphere Process Server when the business process:

- orchestrates multiple processes
- needs human intervention
- has parallel processes in the same flow
- makes almost exclusive use of Web services (WPS is strong in accessing Web services)

Use WebSphere Message Broker when the business process:

- needs extensive routing and transformations
- uses stored procedures
- accesses applications needing industry-standard format
- needs high performance
- has multiple transports in the same flow
- has complex publish / subscribe topology

Out of these recommendations situations may arise where both products are necessary.

6.4.3 Implementing the block "Discovery and Dynamic Binding

In the following sections we will decsribe how to implement the "Discovery and Dynamic Binding" block.

Locate services exposed on the ESB and introduce "consumer discovers supplier" patterns

These activities can be enabled by installing a service registry. IBM offers the product *WebSphere Service Registry and Repository (WSRR)*. A discussion of the general role played by WSRR in the lifecycle of the service (assembly, deployment, runtime, management) has taken place in the previous section.

We are here interested in the activities mentioned above, therefore we will limit our discussion to the way in which WSRR supports us in implementing them.

The location of the service is one of the most important activities, because it touches the base of SOA (loose coupling). We want our service consumers to know as less as possible about the services they will call; the best situation is when they only know the name of the service they require.

Let's see how some of the IBM SOA runtime products interact at runtime with the WSRR. The major user of WSRR at runtime is the ESB, in the form of the mediation component. We will refer to the mediation component as "the requestor". Several actions have to be taken care of:

- *Service endpoint selection* - this means finding, based on the requestor's metadata, the candidate services, applying an algorithm to select one, and routing the request to the service (see Figure 6-18 on page 252).
- *Service availability management* - this means deciding what to do in case a service is not available, selecting alternatives, etc..
- *Policy enforcement* - this means delivering to the requestor the policy information (the contract between consumer and provider), so the enforcement takes place.

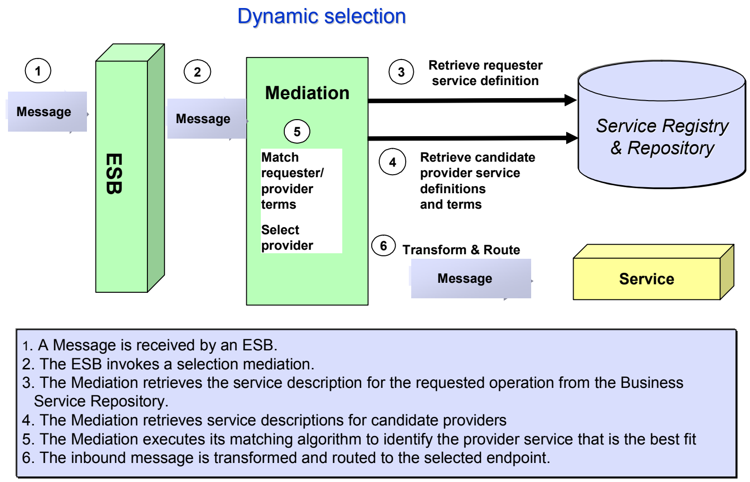


Figure 6-18 WSRR runtime selection and invocation interactions

The integration with WebSphere ESB is done through:

- a new pre-built mediation “Endpoint Lookup”
- optional caching (to reduce interaction with WSRR)
- dynamic endpoint selection

The integration with WebSphere Message Broker is done through:

- new mediation capabilities (Nodes), for example `SRRetrieveITservice` (retrieves PortType) and `SRRetrieveEntity` (retrieves metadata)
- optional caching

Implement dynamic invocation of service and dynamic consumption of WSDL

We describe in the following section a set of activities which, each one in turn, help to refine the loose coupling of the SOA infrastructure, by eliminating even further dependencies between service consumer and service provider.

Web Services Invocation Framework (WSIF)

There are two ways of binding to Web services: *static* and *dynamic*.

- In the *static* process, the binding is done at design time. The service consumer obtains a service interface and implementation description through a proprietary channel from the service provider (by e-mail, for example), and stores it in a local configuration file. So this means the service consumer is dependent upon the WSDL stored in its configuration.

- The *dynamic* binding occurs at runtime. While the client application is running, it dynamically locates the service using a UDDI registry and then dynamically binds to it using WSDL and SOAP. Therefore there is no dependence here between service consumer and the WSDL.

We do not want to tie the client code to a particular protocol implementation, because it is inflexible, restricts possible connectivity, and is hard to change. Therefore we are interested in a process that allows the invoking of Web services, independent of the format of the service or the transport protocol through which it is invoked.

The Apache *Web Services Invocation Framework (WSIF)* provides a standard Java API to invoke services, no matter how or where the service is provided, as long it is described in WSDL. WSIF enables the developer to move away from the native APIs of the underlying service, and interact with representations of the services instead. WSIF is WSDL-driven and provides a uniform interface to invoke services using WSDL documents. So, if a SOAP service you are using becomes available as an EJB, for example, you can change to RMI/IIOP by just modifying the WSDL service description, without needing to modify your service consumers.

This API is used in WebSphere Integration Developer and in the WebSphere Application Server runtime to construct and manipulate services defined in WSDL documents. The architecture allows new bindings to be added at runtime. WSIF has the following advantages:

- Multiple bindings can be offered for services, and bindings can be decided at runtime.
- Switching protocols, location, and so forth, without having to recompile the client code.
- A stubless and totally dynamic invocation of a Web service.

WSIF is an intermediary between service consumer and service provider, and as such shields the consumer from the possible bindings of the provider

WebSphere V6 on z/OS includes a UDDI Version 3 Registry that can be used to obtain this dynamic behavior.

For more information on this subject please consult following documents:

- Dynamic Discovery and Invocation of Web services, by Damian Hagge
`http://www-128.ibm.com/developerworks/webservices/library/ws-udax.html?t=egrL296&p=invocationws#author`

The Selector function inside WebSphere Process Server

Dynamic selection functionality is available also in WPS.

The *selector* objective is:

- Determine dynamically which implementation to invoke based on some defined set of criteria. The predefined implementation uses dates, but it is possible to specify other selection criteria via XML configuration. Custom selection algorithms can be plugged into the selector component by manual coding.
- Decouple the client application from a specific target implementation. Change of target does not require change of client.
- Allows SCA target implementations to be added to the selector dynamically without requiring a restart of the application or server.

6.4.4 The end of the journey

At the end of stage three, there will be an infrastructure and application architecture that provides process integration. Figure Figure 6-19 on page 255 shows the activities done and the IBM products (with specific features) that have been activated.

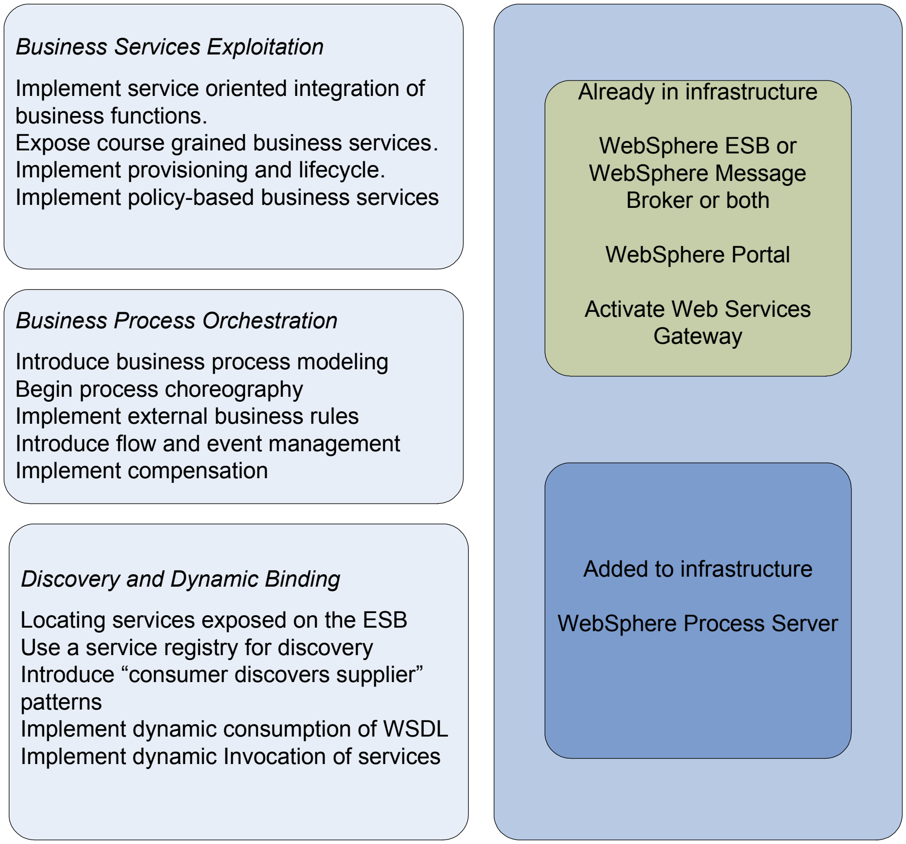


Figure 6-19 Stages from Service Integration to Process Integration.

7

SOA governance on z/OS

The z/OS mainframe environment is well-known for the high Qualities of Service (QoS) that it delivers - security, scalability, reliability, high-availability, and other characteristics. These traits are present, in part, because of the design of the hardware, operating system, and middleware. But another key reason for the mainframe's high level of QoS is the rigor of process, procedure, and governance that surrounds the mainframe in most customers' installations. For example, high system reliability and availability is achieved not just through the ability for hardware components such as the central processors to fail over, or because z/OS can trigger a functional recovery routine when an application attempts to overlay storage. Reliability and availability also stems from change control, rigorous testing, application standards, and other controls that help prevent planned and unplanned outages.

The practice of managing policy and maintaining controls over the platform and architecture is often referred to as *governance*. The word originates from the word "govern," a word that can imply varying degrees of control. One definition of "govern" is "to exercise a directing or restraining influence over; guide," which is not a very daunting definition. However, another definition states: "to exercise continuous sovereign authority over," which has a slightly more dictatorial implication. And this is one of the keys to governance - to control without imposing so much restraint that it constricts the functioning of the organization. Some "urban legends" about the mainframe imply that the mainframe is too inflexible or slow to respond to change. In fact, the mainframe is no different from any other platform with respect to flexibility or responsiveness. But the governance policies of the enterprise often are at issue, and in many cases those restrictions in question are for good reason and greatly contribute to the superior qualities of service of the mainframe.

Since SOA implementations are, by nature, a loosely-coupled collection of services and infrastructure, a strong governance process is necessary to balance the near-anarchy of an unconstrained proliferation of services and service providers by using governance practices to help keep things in order. The fact that SOA uses separation of concerns to abstract business logic from infrastructure logic (communication, mediation of data and connectivity, etc.) means there will likely be a more complex infrastructure of application artifacts, middleware and servers needed to run it. More components means more complexity, and more complexity means more controls to prevent that complexity from becoming an issue. In his paper titled “Introduction to SOA Governance,” Bobby Woolf states:

> *Governance becomes more important in SOA than in general IT. In SOA, service consumers and service providers run in different processes, are developed and managed by different departments, and require a lot of coordination to work together successfully. For SOA to succeed, multiple applications need to share common services, which means they need to coordinate on making those services common and reusable. These are governance issues, and they're much more complex than in the days of monolithic applications or even in the days of reusable code and components*.[1]

The complication that stems from the sharing of services across the enterprise makes governance even more important than ever, especially considering the cross-organizational nature of the deployed services and infrastructure.

[1] http://www-128.ibm.com/developerworks/library/ar-servgov/

7.1 What gets governed?

SOA governance concentrates on policies applied to the service lifecycle. There are many other areas of governance that are not specific to SOA, such as enterprise I/T standards, problem and change management, etc. Some of these will be impacted by the SOA implementation - I/T governance and SOA governance are not mutually exclusive. SOA governance focuses on the design, development, and maintenance of shared business services. One of the key aspects of this is the agreements that are forged between the service providers and service consumers. Key to these agreements are the people involved and the process for governing the service lifecycle.

Woolf[2] states that, in contrast with I/T governance, SOA governance:

- acts as an extension of IT governance that focuses on the life cycle of services to ensure the business value of SOA
- determines who should monitor, define, and authorize changes to existing services within an enterprise

7.2 Who governs?

An enterprise that has the resources for a formalized organization may form a “center of excellence” or similar team to create the governance policies and procedures. A board of knowledgeable individuals may act as the guiding body and serve to mediate agreements between the interested parties, including the service provider organizations and the development teams who are consumers of the services.

Governance is largely a political function. It does not determine designs, infrastructures and other technical issues - it determines <u>how</u> those decisions are made. IBM’s *SOA Governance Lifecycle*[3] demonstrates how SOA governance is not a technology issue. This Lifecycle consists of:

Plan	establish the governance need
Define	design the governance approach
Enable	put the governance model in action
Measure	monitor and manage the governance processes

[2] Ibid

[3] http://www-306.ibm.com/software/solutions/soa/gov/lifecycle/

None of these are directly relevant to technology. The SOA Governance Lifecycle defines how to establish and maintain the governance framework, not the SOA itself.

In the mainframe environment, infrastructure, applications, and the decisions and policies that surround them tend to be highly centralized. Applications are (mostly) not distributed and do not cross organizational boundaries. There are well-defined groups of architects, developers and administrators responsible for the various aspects of the system. While this centralized paradigm does not eliminate the need for governance, it does make it easier. However, the introduction of business-aligned services that are, at least in part, developed by business areas, and the use of distributed infrastructure for components such as an enterprise service bus or process server, all contribute to a more distributed architecture with many more people and teams with an interest in how policy is made.

7.3 Aspects of SOA governance

SOA governance consists of a number of different areas of emphasis. We will examine several of them in terms of how they impact an SOA implementation using resources on the mainframe. The key aspects of SOA governance are:

- Definition
- Development lifecycle
- Versioning
- Migration
- Registries
- Message model
- Monitoring
- Ownership
- Testing
- Security

We now will review a few of these aspects that have particular relevance in the development or hosting of services and infrastructure on the mainframe.

7.3.1 Service definition

Definition of the service is one of the most fundamental functions in the service lifecycle. The architect/developer must determine the service's functionality, scope and granularity, and the composition of the service interface. Since a service is to be composed of business logic and is considered a "repeatable business task," this must be taken into account during the service design - how is the business logic to be partitioned in a way that maximizes the future reusability of the service?

Reflecting back upon prior sections of this book and the mainframe SOA migration patterns and strategies, it is apparent that the "bottom-up" approach of reusing mainframe assets may constrain the service definition task. Since most SOA implementations are not "green field" projects (ie. there are existing assets to be reused), there must be at least some bottom-up and meet-in-the-middle analysis and development done to harvest the existing assets. While many existing mainframe transactions were designed to perform "repeatable business tasks" at the user interface level (for example, functions invoked via CICS menus), the internal structure of these transactions often does not match the business granularity presented on the surface.

As the architect and developer develop a service definition, a detailed knowledge of the asset inventory that might feed that service is critical. The asset discovery tools discussed in 4.2.2, "Discovery and refactoring tools used in the Assemble stage" on page 110, including WebSphere Studio Asset Analyzer and Asset Transformation Workbench, become important in the service definition process by providing a detailed view of the application artifacts and their structure. This can help determine the suitability of the asset for reuse.

7.3.2 Service versioning and migration

Once a service is created, it is not likely to stay static forever. Business requirements and rules will change, and modifications to the service will be necessary. If an adequate governance structure is in place that keeps track of service providers and consumers, it may be possible to gauge the impact of change on a service. Service definition (see above) should be done in a way that defines an interface that can "flex" with the service with time.

The z/OS environment has a long history of attention to backward compatibility and avoidance of impact of change. Consideration of the impact of service migration is closely related to the change control practices that are common in most mainframe customers. Versioning of services should be considered when designing a SOA governance model. In fact, service versioning is one of the livelier areas of discussion amongst architects today![4]

Service migration may become an especially significant issue for migrated z/OS transactions that are reused using the Adapter-provided service interface pattern

[4] A simple Google search for "SOA versioning" returned hundreds of hits, many of which discuss the various difficulties in version control of SOA-based services

and the Improve transition approach. Tools such as the CICS Service Flow Feature or WebSphere Host Access Transformation Services depend upon the 3270 user interface to execute the existing host transactions. Over the years, such techniques have been succeptable to breakage due to change at the user interface. However, now that organizations have started providing the UI via a Web browser, the 3270 interface has become significantly less volatile than in years and decades past.

Theoretically, loose coupling of services from caller to provider and the infrastructure to support it (Enterprise Service Bus, service registry) can help insulate change. The ESB can route service requests to the appropriate caller and/or service, and the registry provides the appropriate endpoint address, depending upon the version requested (or a default).

7.3.3 Service registries

Many developers, when faced with the need to create a new function for an application, will simply build from scratch. Why? Sometimes the urge to create from scratch stems from a personality style, it may result from a development organization suffering from the “Not Invented Here” syndrome (ie. “it can’t be good if our organization didn’t create it...”), or there may simply be a lack of information on the pre-existence of other assets that meet the requirements for that function. In the latter case, a service registry can ease the job of finding and reusing existing services.

A service registry can increase the service reuse in an organization by providing a searchable directory of services that advertises the existence of service functions. The registry provides information on the service interface and how to invoke it, and it provides details on the location of the service for design-time or run-time resolution. As mentioned in the prior section, a service registry can help insulate the service caller from changes in a service or new versions that have been provisioned.

IBM’s registry solution

In September, 2006, IBM announced the *WebSphere Service Registry and Repository* (WSRR).

WSRR is the master metadata repository for service descriptions, including traditional Web services implementing WSDL interfaces with SOAP/HTTP bindings as well as a broad range of SOA services that can be described using WSDL, XSD and WS-Policy decorations. As the integration point for service metadata, WSRR establishes a central point for finding and managing service metadata acquired from a number of sources, including service application deployments and other service metadata & endpoint registries and repositories, such as UDDI. With WSRR, service visibility is controlled, service versions are managed, proposed changes are analyzed and communicated, usage is monitored and other parts of the SOA foundation can access service metadata with the confidence that they have found the copy of record. WSRR focuses on a minimalist set of metadata-describing capabilities, requirements and semantics of services. It interacts and federates with other metadata stores that support specific phases of the SOA life-cycle and capture more detailed information about services relevant in those life-cycle phases; examples of specialized repositories include a reusable asset manager in development or configuration management database in service management.

The product architecture of WSRR is shown in Figure 7-1

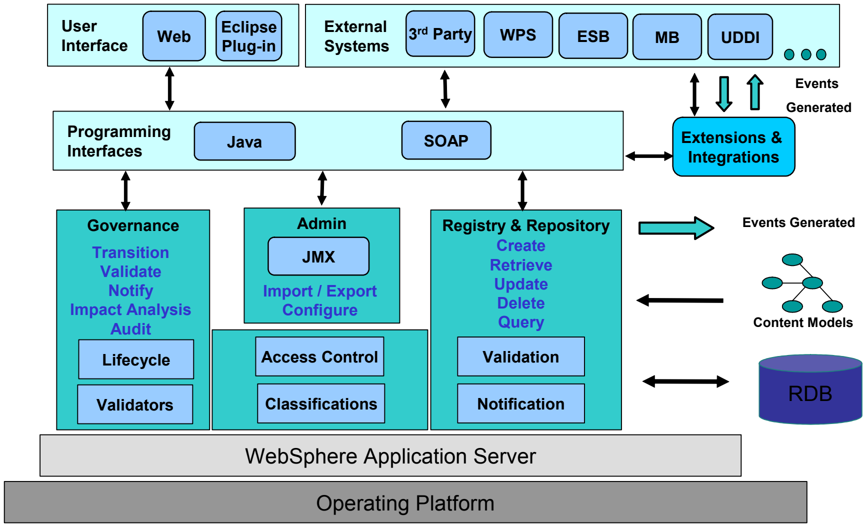


Figure 7-1 WebSphere Service Registry and Repository architecture

WSRR fulfills the requirements of a SOA service registry by providing the following major functions:

- **Service publishing and inquiry**
  Publish, retrieve, query and manage documents which describe services (a metadata repository). Documents may be XML, WSDL, XSD, WS-Policy, or other formats.
- **Service registry**
  Register services, so that information about the service, including endpoint location, interface description, etc. can be resolved at design or run time.

- **Event notification**
  Changes to WSRR content can trigger notifications to subscribers (JMS pub/sub or e-mail notification)
- **Governance model**
  WSRR facilitates governance of entities within the repository by directing the governance process by use of state machines that define the entity's lifecycle.
- **Programming interfaces**
  Both Java and SOAP interfaces are provided for CRUD operations against the metadata repository.
- **User interfaces**
  Web and Eclipse-based user interfaces are provided. The Eclipse interface is intended for use by developers and analysts; the web interface can be used for metadata management and governance.

Figure 7-2 shows an overview of the WSRR from a solution perspective.

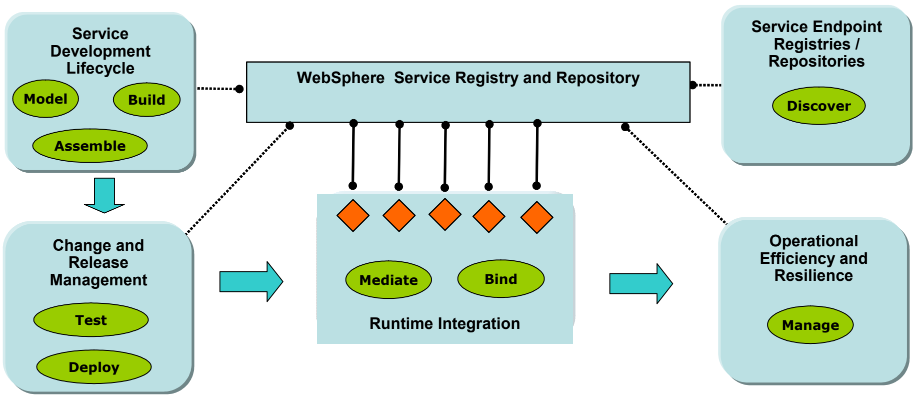


Figure 7-2 Solution view of WSRR

The WebSphere Service Registry and Repository plays a key role in completing the vision of SOA as a loosely-coupled abstraction of business logic and infrastructure. Without a registry to resolve service endpoints at runtime, it is difficult to completely disconnect the service consumer and provider, as the endpoint infrastructure must still somehow be reflected in the services. Also, WSRR helps to mitigate the other difficulties in SOA governance, including version control, migration, and tracking the development lifecycle of services.

For mainframe-based services that have been migrated using the SOA migration techniques described earlier, the WSRR provides the place to store the descriptions of the services, such as the WSDL and other XML artifacts. WSRR will provide the run-time registry for the ESB to locate and resolve service endpoint addresses to fulfill the loose coupling requirement. And, while the z/OS platform will provide the optimal levels of security, reliability, etc. for the registry, WSRR can be hosted on any platform supported by the product, even if the services are located on z/OS.

7.3.4 Service monitoring

The introduction of SOA, loose coupling of services, and new infrastructure components including ESBs, process servers, and service registries poses new problems of complexity in mainframe environments that formerly consisted simply of z/OS running CICS transactions. To maintain the levels of QoS to which mainframe customers are accustomed, new, rigorous monitoring practices are needed. There are several areas of monitoring that are important:

Business monitoring

The traditional concept of “monitoring” encompasses the monitoring of I/T components - CPU utilization, I/O saturation, response time, etc. SOA’s close ties between I/T and business dictate a slightly different approach to monitoring. Business people need access to metrics that show the state of business functions - a “business dashboard” is a common implementation that permits the business analyst to monitor various measurements of business operation. For example, the insurance executive may wish to monitor the number of claims being processed in a period of time by the claim processing center. Or a bank manager might want to monitor the flow of funds through his branch.

The decoupling of business logic from I/T logic makes it easier to monitor business-related metrics. Composite applications and workflows can be instrumented to produce statistics that are fed into business monitoring tools, and those statistics can be provided back to the modeling tools used to create the workflows and processes to further improvements. IBM provides the *WebSphere Business Monitor* to give business people the ability to monitor such business metrics and key performance indicators.

I/T monitoring

The complexity of the SOA infrastructure dictates the need for new, end-to-end monitoring tools that can show not only the status and performance of services on their respective transaction servers, but also the availability and performance of the end-to-end process and/or transaction. End-to-end performance monitoring has been a challenge for many years, but the complexity of a SOA-based application that spans many application, database, ESB, process, directory and security servers is far greater than a CICS 3270 transaction!

Service level agreements (SLAs) are critical to the proper governance of a SOA implementation. Without an SLA, a customer may not be able to adequately provide sufficient levels of performance or scalability, as there is no benchmark by which to gauge success. How good is “good enough?” How fast is “fast enough?” Without a SLA, an I/T organization may be on a never-ending quest for faster/better transaction performance.

IBM’s Tivoli software brand provides many monitoring tools that help the I/T staff measure and manage an SOA-based application, provide an end-to-end picture of the application, and determine if SLAs are being met. The *Tivoli Composite Application Manager for WebSphere* and the *Tivoli Composite Application Manager for SOA* both aid the end-to-end monitoring requirement. The WebSphere monitor concentrates on the WebSphere Application Server and can give information about services executing there, including those that invoke back-end services on CICS or IMS, per the Adapter-provided service interface pattern. The SOA monitor gives a higher-level view of the SOA service at a web services protocol level.

Accounting

One of the sometimes-controversial aspects of I/T governance is resource accounting (in some contexts referred to as “chargeback”). Organizations usually must account for resource usage, and sometimes charge the users of resources for the consumption of those resources. In a loosely-coupled SOA implementation, it may not always be possible to accurately account for the consumption of resources. The loose-coupling may not always maintain the identity of the service consumer from end to end. The need to maintain this identity can drive the architectural decisions of what migration pattern and approach to use to move a mainframe application to SOA.

Mainframe z/OS customers have always had the ability to collect vast amounts of data to use for resource accounting. The z/OS System Management Facility (SMF) is an inherent part of the operating system and can measure virtually any activity that occurs under z/OS. SMF records are created by all of the major IBM transaction managers (CICS, IMS, WebSphere, WebSphere MQ), by the database managers, and most other users of the system. The SMF repository can be used to report on system performance, utilization, and resource consumption - at a very granular level. Organizations wishing to charge back for resource consumption can use SMF to create the billing needed. However, the architect must design the SOA implementation so that SMF data is accurately gathered to reflect the “true user” of the service. Some SOA enablement mechanisms may not easily transmit the identity of the service caller without custom coding. For example, J2EE Connector Architecture connections can maintain user identity from a web application server caller back into CICS or IMS, but a WebSphere MQ connection may require extra code to populate the appropriate message headers.

8

SOA and z/OS QoS

Most of the content in this book is really about application architecture matters. We have had many discussions about how to structure SOA applications, how services should integrate and what software products to use based on their features. We have said very little in all the decision making about QoS factors. In fact, many IT decisions are based on QoS impact rather than just functional matters. We should ask questions such as:

- Does a certain service interface solution techniqe perform?
- What is the the throughput of a certain ESB solution?
- Can we propagate security context in our service call using a certain solution technique?
- If a service is not available, is there automatically an alternate available?
- Does our service provider scale?

Most of us think about performance, scalability, reliability, integrity and perhaps one or two others in terms of QoS. The full list of QoS is longer, though. z/OS is traditionally strong in Quality of Service and it seems that SOA requires more QoS than traditional IT environments. The main reason for this is because in a properly implemented SOA assets will be increasingly shared and reused.

This chapter talks about the QoS aspects in the implementation of an SOA on z/OS, or better, how to make use of them, as QoS is already there as part of the

System z hardware and z/OS operating system. Focus will be on availability, scalability, security and workload management. We will also talk about TCO.

The chapter is organized as follows:

- 8.1, “Overview” on page 269 is an overview section.
- 8.2, “Quality of Service on the System z platform and inside z/OS” on page 271 discusses the QoS inherently part of the System z hardware and z/OS operating system.
- 8.3, “Quality of Service of the SOA building blocks” on page 276 talks about the QoS delivered as part of the SOA enablement products available.
- 8.4, “Quality of service of the SOA architecture” on page 304 discusses QoS aspects delivered by the SOA architecture itself.
- In 8.6, “Managing QoS with SOA on z/OS” on page 319 we have a brief discussion on management of QoS on z/OS.
- 8.5, “QoS and our implementation scenarios” on page 304 contains a discussion on how the QoS discussed apply to our implementation scenarios and to what extent they play a role in making decisions in terms of solution techniques.
- We conclude this chapter in 8.7, “Conclusion” on page 320.

8.1 Overview

Quality of service (QoS) is a major theme in today's IT environments. The Quality of Service of the platform and of the software products running on that platform are an important area in the decision processes of IT departments. It is important to know how an SOA architecture can benefit from the strong QoS of the z/OS platform, but also each solution technique as discussed in Chapter 5, "SOA implementation scenarios" on page 129 has different QoS characteristics. The QoS aspects of a certain solution technique may be more improtant than the functions and features.

A service (in the sense defined by SOA) has, besides the functionality it implements, qualities that describe how it delivers the business function. Those qualities are, for example:

- the reliability of the service
- the conditions for reachability (availability)
- the security cover under which it acts
- the capacity and performance that it delivers
- the transactional capabilities

The SOA environment implemented on z/OS receives the Quality of Service features from several areas:

- The System z hardware and the physical system infrastructure on which the software products (middleware) are positioned.
- The z/OS operating system with its built-in subsystems, such as Workload Manager (WLM), Resource Recovery Services (RRS) and solutions such as Parallel Sysplex and Geographically Dispersed Parallel Sysplex (GDPS).
- The features of the software products that build the underlying SOA infrastructure (that means the products from the portfolio supplied by IBM, described in the previous chapters).
- The inherent characteristics and features of the SOA architecture itself, such as the standards being used.

These areas can be viewed as layers, which are positioned one over the other (layer 1 at the basis):

Layer 1 The hardware and operating system layer.

Layer 2 The layer constructed with the products that implement the SOA building blocks.

Layer 3 The layer built by positioning the application over the SOA building blocks.

Figure 8-1 shows this layered structure.

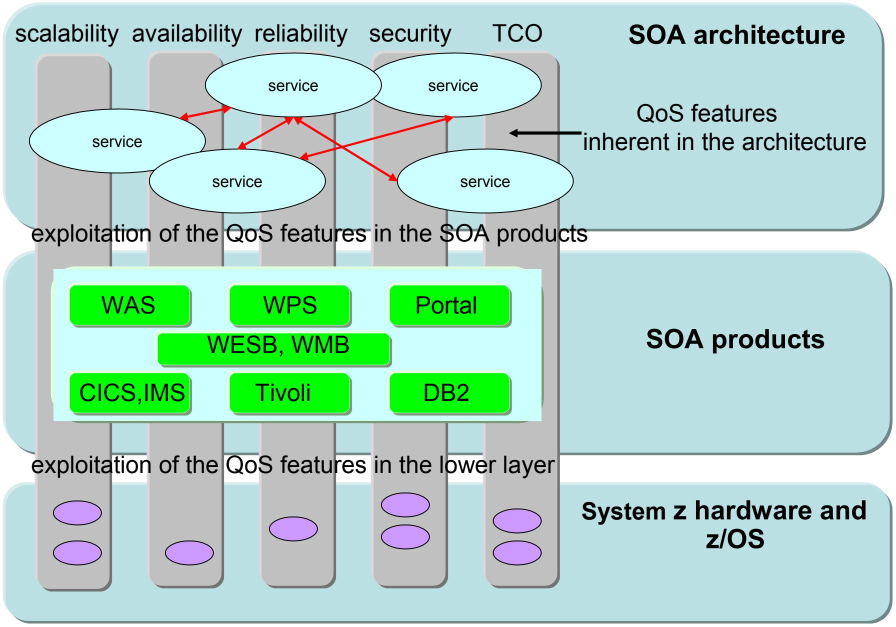


Figure 8-1 SOA QoS features - layer structure

> **Attention:** Note that in this book we focus on the QoS “scalability”, “availability”, “reliability”, “security” and “Total Cost of Ownership”. There are more QoS, such as maintainability, manageability and so on, but leave those out of the scope of the dicussion.

We think that this layer positioning and the way in which the upper layers exploit the functionality of the lower layers increase the QoS of the SOA solutions positioned on z/OS. We describe in the following sections how this “exploitation” is implemented.The QoS features discussed in the next sections are scalability, availability, reliability, security, Total Cost of Ownership (TCO).

8.2 Quality of Service on the System z platform and inside z/OS

The Quality of Service features of the System z hardware platform in combination with the the z/OS operating system are well and in detail described in several redbooks and other publications (for example, *IBM System z Strength and Values,* SG24-7333). However, we will summarize the most important QoS in the next sections.

8.2.1 Scalability

Scalability is the ability to accept increased workloads without degradation of service. In reality this means that an infrastructure needs to be capable to easily find the additional resources needed for this increased workload, and without running into bottlenecks. Resources here means anything needed to run the workload, such as memory, CPU, DASD, address spaces, threads and so on. Resources can be at a “hard” level, such as real memory and CPU, for instance, or at a “soft” level, such as threads, address spaces or virtual memory. “Hard” resources will, of course have a given limit (i.e. the number of CPUs installed in the system), but “soft” resource maximums are configurable and tunable to accept peak workloads.

The main function available in z/OS to manage the workload in accordance with the given resources is the workload management. The function of the workload management is very simple: keep the machine busy while staying within the agreed service levels. The key features that implement this are:

- Dynamic allocation of I/O, CPU between LPARs.
- A sophisticated implementation of the z/OS *Workload Manager (WLM)*.

8.2.2 Availability

Availability comes in many flavors. The level of availability should depend on the bsuiness requirements, but we already know that in a full blown SOA availability becomes more important, because of the increased reuse of IT assets. System z and z/OS are designed for *High Availability (HA)*, without having to add any additional software products. It is “part of the box”. Note, however, that some levels of availability may require multiple LPARs, or even multiple footprints. With System z and z/OS, “silver”, “gold” and “platinum” levels of availability can be achieved. Very simply, without getting into too much technical detail, the levels would be as follows:

Silver One footprint (machine) with multiple LPARs (system images), allowing for non-disruptive maintenance at the operating system level, planned and unplanned outages of an LPAR and intelligent workload balancing between LPARs dynamically. All of this with an already high level of availability.

Gold Multiple footprints in the same location with one or multiple LPARs, allowing for non-disruptive maintenance at both the hardware and operating system level, planned and unplanned outages of an LPAR or an entire machine and dynamic intelligent workload balancing between LPARs and machines. All of this with a very high level of availability.

Platinum Multiple footprints in mulitiple sites, allowing for non-disruptive maintenance at both the hardware and operating system level, planned and unplanned outages of an LPAR, an entire machine or an entire site and dynamic intelligent workload balancing between LPARs and machines. This would deliver extremely (continuous) high availability at the hardware and operating system level.

The most important technology available in the System z hardware and the z/OS operating system that enforce the high availability functionality is:

- Inclusion of *self-healing* attributes in the hardware in order to prevent downtime caused by system crashes. The operating system takes advantage of the self-healing attributes of the hardware, and extends them by adding functions such as recovery services for all operating system code, address space isolation, and storage key protection.
- Elimination of planned outages through concurrent hardware changes.
- Elimination of planned outages through *Enhanced Driver Maintenance*. This allows upgrades of the Licensed Internal Code for processors of all types (CPs, IFLs, ICFs, zAAPs), memory, and I/O adapters that are transparent to the application.
- On demand capacity upgrades (CUoD and CIU are processes that allow a temporary or permanent upgrade in the CPU, memory, I/O ports).
- Removal of *Single Point Of Failure (SPOF)* through the Parallel Sysplex architecture (non-disruptive attachment and removal of LPARs, servers, non-disruptively installation and maintenance of hardware and software). Parallel Sysplex is discussed a bit more in detail in “Parallel Sysplex” on page 273.
- Business continuity solutions like Geograpically Dispersed Parallel Sysplex (GDPS) (with its variations GDPS/PPRC synchronous mirroring, GDPS/XRC asynchronous mirroring, GDPS/Global mirroring with asynchronous PPRC

replication technology). GDPS is discussed a bit more in detail in “Geographically Dispersed Parallel Sysplex (GDPS)” on page 274.

Parallel Sysplex

Centralizing all components on a single system does not imply that mainframe computing is limited to one single box or to one single operating system, on the contrary. Mainframes have the capability of being partitioned in up to 60 logical partitions (LPARs) and each one has the ability to run one of the five System z operating systems. In that case, there are management tools and resource sharing at a box level, and at operating system level.

Figure 8-2 shows a simplified view of the concept.

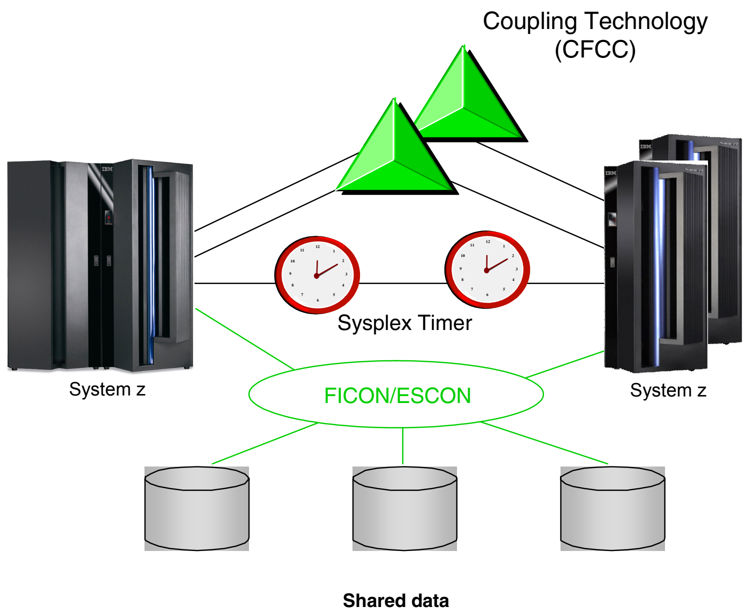


Figure 8-2 Parallel Sysplex - simplified view

The z/OS operating system provides the capability of creating a cluster of up to 32 systems named *Parallel Sysplex*, where a system can be a full box or just an LPAR. Most computing platforms today have clustering capabilities, but Parallel Sysplex is a completely different kind of clustering solution because its capability of sharing every resource between the elements in the cluster and the capability to dynamically reconfigure, add or remove resources.

z/OS allows all members in a cluster to share all data, even up to the record level. All other cluster implementations, at best, allow to partition data amongst the elements of the cluster, and each system can access just the data attached to it.

Parallel Sysplex also provides significant network optimizations for communication across its cluster members. Once a client request reaches the *Sysplex Distributor* there is no more external network traffic required; all traffic flows over the System z hardware. As a consequence, network latency is kept to a minimum and typical network issues you normally see in a physical de-centralized infrastructure, are inherently absent. Even when the Parallel Sysplex is physically spread over several different boxes, the communication between them flows over high speed fiber optic connections managed by the *Cross Coupling Facility (XCF)*, a specific protocol for those connections, with a magnitude of GigaBytes of transfer rate. Within a physical machine, communication between z/OS images is accomplished through memory to memory and there is no network protocol faster than that.

Parallel Sysplex is key in achieving high availability in any of the SOA building blocks discussed later in this chapter. It is the technology that makes it possible to run the same workload on different system images, while using the same data image (provided DB2 data sharing is enabled too).

Geographically Dispersed Parallel Sysplex (GDPS)

For higher availability and *disaster recovery* purposes, a Parallel Sysplex can be configured in a *Geographically Dispersed Parallel Sysplex* (GDPS) mode. There are two GDPS modes:

- *GDPS/PPRC*, a configuration where we have a Parallel Sysplex distributed over two sites, connected together up to 100 km, with data synchronized and shared continuously. One site (part of the sysplex) is acting as a primary, and the second site is acting as a secondary, in stand-by mode. GDPS controls and automates a full swap to the backup site in case of failures.
- The second mode of operation for GDPS is *GDPS/XRC*, where distance between sites can be more than 100 km, theoretically without limitation. In GDPS/XRC the sysplex does not span both sites, but instead a full system image is swapped to the alternate site in an emergency situation.

Both modes use the so called *HiperSwap* capability which provide the ability to activate replicated data in the disaster recovery site without application outage.

8.2.3 Reliability

Reliability is another Quality of Service with a broad meaning. In practice reliability is associated with the ability to run programs with transactional attributes and make sure data is not becoming corrupted. In the broad sense, however, reliability is also depending on the level of security on the system and other QoS.

The System z hardware provides built-in hardware redundancy (exemplified in continuos enhancements in hardware). and the z/OS operating system provides *Resource Recovery Service (RRS)* for global transactions with two-phase commit. RRS manages resources in a way that they are registered in a global transaction only when they attempt to make changes to resources under its control.

Besides the hardware and the operating system, CICS, IMS and DB2 have their own built-in features for reliability ensuring two-phase commit, integrity of data and backward and forward logging mechanisms for recovery purposes.

Resource Recovery Services (RRS)

Author Comment: ALK todo: description on RRS

8.2.4 Security

Security is important and again very broad. Security encompasses many things, such as authentication, authorization, encryption, auditing and so on.

The System z platform, the z/OS operating system and its security server subsystem provides a full range of security features:

- Dedicated security server (z/OS security server, including RACF, LDAP server, PKI infrastructure and much more).
- Support for a variety of *encryption standards* to keep current with industry and government security regulations - all inside the z/OS cryptographic services.
- *End-to-end data protection* that helps keep data uncorrupted and uncompromised. The generic product used is IBM Encryption Facility for z/OS, encompassing IBM Encryption Facility for z/OS, the Encryption Facility Client and the DFSMSdss™ Encryption Feature.
- Integration of security with the network with built-in technology resistant to hackers.
- Protection of the data in the network (encryption techniques, VPNs).
- Support for protecting of system resources and data from unauthorized access (TCP application support for RACF).
- Protection of the system from the network (IP packet filtering, traffic regulation, intrusion detection services).
- Managed access to critical data through Multiple Level Security (MLS integrated in RACF and used by different subsystems).

8.2.5 Total Cost of Ownership (TCO)

In this book we will stay away from whether the TCO of System z is good or bad, but mention the features that have been provided over time to lower the TCO. It is IBM's direction to keep on improving the TCO of the System z platform. The TCO improvements provided are:

- Simplification and automation of the processes running on z/OS (features in the area of systems management, reducing personnel costs through automation).
- Introducing features in the standard delivery (for example making security features standard - to be compared with a car where security features are implemented as standard, not like add-ons).
- Simplify the disaster recovery implementation and therefore reducing the cost necessary to put it in place.
- Implement sophisticated workload management techniques in order to use very efficiently (means cost effective) the resources on the z/OS platform.
- Changing the commercial rules, changing the licences and the way to price the CPU usage. The z/OS platforms provides multiple hardware and software pricing options that will suit different requirements.
- Offload CPU loads to specialty processors, which are priced differently to make it more attractive to run new workloads on the platform.
- Capacity on demand offerings.

8.3 Quality of Service of the SOA building blocks

In this section we review the software products used in the SOA building blocks and describe the Quality of Service provided by these products. The section is organized in the following way: we consider the QoS features and show, for each of them, how the products that represent the SOA building blocks implement these features.

Note, however, that we will not describe transaction servers and DB2 (although in scope); we recommend the details in the other existing documentation and redbooks.

8.3.1 Scalability

Scalability is a very important Quality of Service. It is present in all products running on z/OS and reflected mostly in the way in which these products implement workload management.

In the following sections we will discuss how this workload management is supported in each of the products playing an important role in SOA on z/OS.

We will elaborate on the following products:

- WebSphere Application Server as the foundation infrastructure for WebSphere ESB and WebSphere Process Server (discussed in “WebSphere Application Server and workload management” on page 277).
- WebSphere ESB and WebSphere Process Server (WPS) in “WebSphere Process Server (WPS), WebSphere ESB, and workload management” on page 282.
- WebSphere Message Broker (WMB) in “WebSphere Message Broker and workload management” on page 282.
- WebSphere Portal in “WebSphere Portal and workload management” on page 284.

WebSphere Application Server and workload management

We have several possibilities to scale using WebSphere Application Server on z/OS: vertical scaling and horizontal scaling. We discuss first the *vertical scaling*.

As part of the vertical scaling process within a WAS server, the Workload Manager will start as many servant regions as required (within imposed restraints) to process the workload and meet the defined goals. If a given servant is overloaded, it is temporarily bypassed in favor of less busy servers. If a servant fails, other servants take over the work and the servant is recovered. When the servants are no longer needed, they are automatically stopped.

Horizontal scaling is especially effective in environments that contain many smaller, less powerful systems/nodes. Client requests that overwhelm a single system can be distributed over several systems.

On the z/OS platform there are *two wokload managers: zWLM* and *eWLM*, which cooperate in order to supply optimal workload management services:

- zWLM is the workload manager of z/OS and it has an operating system view on the WebSphere Application Server.
- *eWLM (enterprise WLM)* is involved with application response monitoring, and feeds information back to the zWLM.

The workload management is concerned with the optimal distribution of work requests to the processes (on z/OS, address spaces). The work requests are, in the case of WebSphere Application Server: HTTP requests, servlet requests, messages, Web services and EJBs.

We discuss in the following sections two aspects:

- classification of work requests
- distribution of work requests

Classification of work requests

Workload management in z/OS is based on the concept of grouping work into *service classes*. The incoming work request is classified to a service class and the WLM schedules the resources to complete the work request according to this service class. Figure 8-3 on page 279 shows how WebSphere Application Server work requests are classified into service classes.

The following components are illustrated in Figure 8-3 on page 279:

- *Work qualifier*
  WebSphere Application Server for z/OS associates each work request with a *work qualifier* that identifies a work request to the system.
- *Classification rules*
  Classification rules associate a work request, as defined by its work identifier, to a WLM service class.
- *Service class*
  z/OS WLM organizes work into workloads and *service classes*. The service class for a group of work defines the performance goal and business importance.
- *Performance goals*
  There are three kinds of *performance goals*: *response time*, *execution velocity*, and *discretionary*. The response time goal indicates the response time goals for individual transactions, the execution velocity goals are suitable for started tasks or batch jobs, and the discretionary goals are for low priority work.
- *Business importance of the work*
  The business importance for a service class defines how important it is to achieve the performance goal for that service class. At runtime, the workload management component manages workload distribution, and allocation of resources to competing workloads. High priority workloads get guaranteed, consistent results, for example, response time, throughput, etc.

A few software constructs inside WAS are used to manage, control and deliver the classified work units to their respective servant adress spaces.

- *Queuing services*
  The controller region is queueing work requests to workload management for execution in servant address spaces; it listens for work requests and puts them on the Workload Management queue. The Workload Management component of z/OS dispatches the work to the servant region according to the WLM policy specified by the work identifier.

- *Enclaves*
  Enclave services allow performance management of a transaction across multiple address spaces and systems inside a Parallel Sysplex. The controller region creates the enclave and associates the transaction to this classified enclave. Then the transaction is queued, waiting to be served by an available thread in a servant region.
- *Application Environments*
  Application Environments allow WLM to start (or stop) servant address spaces in order to meet transactions performance goals, as the workload varies.

Figure 8-3 illustrates the flow from the WebSphere controller region, through WLM, to the servant regions.

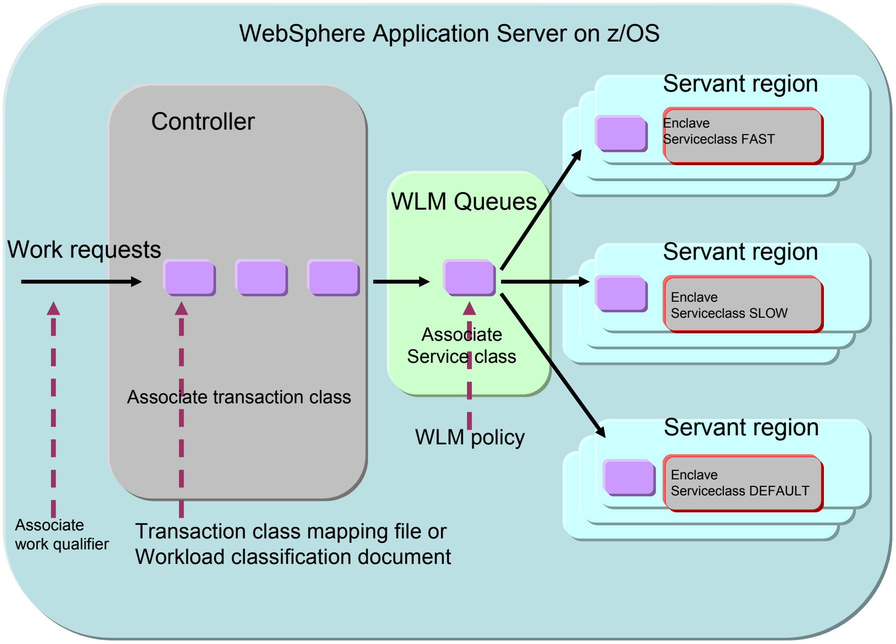


Figure 8-3 Workload management in WebSphere Application Server on z/OS

WebSphere Application Server can assign a *Transaction Class (TC)* to a work item by using a Transaction Class mapping file for HTTP requests, or by using a workload classification document for HTTP, IIOP, or MDB inbound requests.

- *Transaction Class mapping file.*
  This file allows us to associate a set of URIs to a specific Transaction Class. At execution time, WLM will use the Transaction Class to associate the work request to a service class.

- *Workload classification document.*
  This is a common .xml file for the classification of inbound HTTP, IIOP, and MDB work. The `InboundClassification` element defines the type of work that is to be classified by type of work specific child elements. The statement used is:

```
<InboundClassification type="iiop | http | mdb"
schema_version="1.0"
default_transaction_class="value">
```

  IIOP work can be, in a very flexible way, classified based on the following J2EE application artifacts (specified through the `iiop_classification_info` element.):

  – Application name
    This is the name of the application containing the EJBs. It is the display name of the application, which is not necessarily the name of the .ear file containing all the artifacts.
  – Module name
    This is the name of the EJB .jar file containing one or more EJBs (there can be multiple EJB .jar files contained in an .ear file).
  – Component name
    This is the name of the EJB contained in a module (or EJB .jar) (there can be one or more EJBs contained in an EJB .jar file).
  – Method name
    This is the name of a remote method on an EJB.

  HTTP work can be classified based on the following J2EE application artifacts:

  – Virtual Host Name
    This is the host name in the HTTP header to which the inbound request is being sent.
  – Port Number
    This is the port on which the HTTP catcher is listening.
  – URI (Uniform Resource Identifier)
    This is the string that identifies the Web application.

  Message Driven Beans can be associated to a transaction class. The following are the filter elements that you can use for MDB classification:

  – Listener Port (Endpoint)
  – Message Selector (XML tags)

We see how that the classification of work units is very granular, because we have a flexible way of classifying the work requests; having this granularity, WLM is able to schedule appropriately the work units and to “fill the box”.

Distribution of HTTP requests

Distribution of HTTP requests takes place through connection dispatching; we describe a solution available on z/OS, without using any external technology.

Connection dispatching is the routing of TCP connections from a dispatching (or distributing) node to a group of target servers. The dispatching node receives data from the client and forwards it to the appropriate server, which can reply directly to the client. All systems in this cluster provide information about their workload to a dispatching entity, which is generally referred to as a *distribution manager*. This manager is responsible for distributing connection requests from clients to the target systems where the application servers are running. The distribution is based on the current workload information collected by the distribution manager.

The *Sysplex Distributor* is a state-of-the-art connection dispatching technology that is used among z/OS IP servers. The dispatching entity in this solution is a z/OS system in a Parallel Sysplex (called the *distributing stack*), and the target servers are exclusively z/OS systems in the same Parallel Sysplex. With this technique, the client sees a traditional TCP/IP connection with a server, not being aware of the existence of the distribution manager.

Sysplex Distributor extends the notion of automatic *Virtual IP Address (VIPA)* takeover to allow for load distribution among target servers in the sysplex. The Sysplex Distributor is advertising ownership of some IP address by which a particular service is known (called *Distributed VIPA* or *DVIPA)*. The Sysplex Distributor makes use of Workload Manager (WLM) and its ability to determine server load. WLM informs the distributing stack of the target server loads so that the distributing stack may make the most intelligent decision regarding where to send incoming connection requests.

Additionally, Sysplex Distributor has the ability to specify certain policies in the Policy Agent so that it may use QoS information from target stacks in addition to the WLM server load. Further, these policies can specify which target stacks are candidates for clients in particular subnetworks.

The connection routing technology of Sysplex Distributor allows a VIPA to move nondisruptively to another stack.

The limitation of the solution is that the Sysplex Distributor does not support affinity to a specific target host. The choice of the target is selected according to WLM metrics, policies, and configurations and could possibly be a different target than in the last request.

This solution works well with stateless applications. For applications that need to keep state, the WebSphere plug-in should be the one in charge of dispatching directly to the WebSphere servers.

Distribution of servlet requests

The servlet requests are dispatched by the HTTP server plug-in to one of the available application servers; the routing is *server-weighted*. These servers are known to the HTTP server plug-in by the plugin_cfg.xml configuration list; this list delivers information about the primary and backup servers, including their weight.

Distribution of messages

Messages are distributed using the partitioned messaging destination feature, that spreads the messages across multiple messaging engines.

Distribution of EJB requests

EJB requests are spread using a daemon which asks WLM on z/OS for a recommended endpoint, then takes the decision and sends the EJB request to the selected EJB container.

Full details (and alternative solutions) can be found in the Redbook "*Architecting High Availability using WebSphere V6 on z/OS*", SG24-6850. The same book provides a detailed discussion of the theme of HTTP sessions (affinity, session management, persistence).

WebSphere Message Broker and workload management

WebSphere Message Broker (WMB) supports goal-oriented resource allocation. When a Message Broker V6 execution group address space starts, it can be assigned to a Workload Manager (WLM) service class, which in turn is assigned a specific goal during the WLM configuration process. The ability to assign WLM service classes to message processing workload has two significant benefits:

- As work passes through subsystems that are WLM enabled, the service class can be maintained (resources such as CPU and IO "follow" users' work).
- WLM classification means that in the event of a resource constraint (at peak time for example), high priority workloads can receive appropriate resource to ensure that critical workloads are completed at the expense of less important work.

WMB has the possibility of assigning threads to CPUs, assign message flows to execution groups, or assigning message flows to multiple brokers deployed within the sysplex broker domain. All these reconfiguration activities are dynamic.

WebSphere Process Server (WPS), WebSphere ESB, and workload management

WebSphere Process Server and WebSphere Enterprise Service Bus for z/OS are built on top of WebSphere Application Server for z/OS. The information in the previous section about WAS in relation to WLM is therefore relevant here, because WebSphere Process Server and WebSphere Enterprise Server Bus are executing in the WebSphere Application Server address spaces.

Clustering can be employed to create a high availability Message Engine (ME) for the WebSphere Process Server. The configuration has the following features:

- Provides for multiple Message Driven Beans to utilize a common queue and a single persistent store.
- WLM decides which server will run the Message Engine.
- If the active Message Engine fails then the HA (High Availability) manager will activate a new Message Engine on an available server.
- WLM routes the JMS clients to the currently active Message Engine.

In order to increase scalability the logical approach is to add more Message Engines to the server cluster, and using the “JMS destination partitioning” to partition the messages across the Messaging Engines in the cluster. This feature can be used only for specific types of applications (those where cluster affinity and stateful messages are not relevant).

As with WPS, topologies for WebSphere Enterprise Service Bus for z/OS are mainly imposed by those of WebSphere Application Server for z/OS: sysplex and server clustering are the main considerations when you need high availability. Internally, messages passed through mediations in WebSphere Enterprise Service Bus are persisted in the Service Integration Bus. These Service Integration Buses are made out of Messaging Engines that live inside application servers.

The configuration of Service Integration Buses is very flexible. You can have individual Messaging Engines, or clusters containing multiple Messaging Engines that can share workload, be highly available, or both.

The following options are available:

- *Workload sharing configuration*
  Multiple Messaging Engines (one per application server) clustered together, Message Engines are restricted to running in their application servers. The last level of workload balancing provided by the WLM at servant level.
- *High availability Message Engine configuration*
  Single Messaging Engine running in a cluster. It lives in more than one application server so that there is failover to the rest of servers in the case of maintenance or a system crash. If the cluster is horizontal and sysplex wide, WLM policies for workload through the sysplex apply.
- *High availability Messaging Engines with workload sharing*
  A combination of the two previous options. Multiple Messaging Engines running in a cluster with Messaging Engines being able to failover to one or

more application servers in the cluster. Both, local WLM policy for servants in an application server and sysplex-wide policy for the cluster apply here.

A comprehensive discussion is out of the scope of this book. We can see that the combination of WebSphere Application Server, WebSphere Process server and WebSphere ESB provide high availability. For detailed information please consult Redpaper "*z/OS technical overview: WebSphere Process Server and WebSphere Enterprise Bus*", REDP-4196.

WebSphere Portal and workload management

WebSphere Portal runs in WebSphere Application Server and inherits the workload management support discussed in "WebSphere Application Server and workload management" on page 277.

8.3.2 Availability

The level of availability provided by the SOA products is determined by the clustering capabilities of the product itself and the fact whether the product exploits Parallel Sysplex or not.

All the SOA products within our scope, including backend systems, such as CICS, IMS and DB2 exploit Parallel Sysplex. Parallel Sysplex means that multiple system images can be tied together and behave as one logical system. If one LPAR drops out, the other LPARs take over the workload.

All the WebSphere products (Application Server, MQ, Message Broker, ESB, Process Server and Portal) and CICS and IMS provide clustering techniques and DB2 provides data sharing. Clustering means "cloning" of a server that can process the same workload. This requires each "clone" to have the same program logic deployed to it and have access to the same data.

We will now discuss in more detail the sysplex exploitation and clustering capabilities per SOA product.

WebSphere Application Server

To exploit Parallel Sysplex with WebSphere Application Server in a meaningful way and achieve full high availability, the following conditions must be met:

- A Parallel Sysplex must have been installed, with one or a duplicate Coupling Facility (CF).
- DB2 data sharing must be enabled.
- WebSphere Application Server must be installed in a Network Deployment (ND) topology.

- Clustering must have been configured between servers on multiple LPARs.

The above would allow to run the same workloads on multiple servers in multiple LPARs having access to the same data image.

WebSphere ESB/Websphere Process server

WebSphere ESB and WPS for z/OS can be installed and configured to run in a WebSphere Application Server Network Deployment configuration. Being able to run WESB and WPS workloads in a sysplex environment requires the same prerequisites as mentioned in “WebSphere Application Server” on page 284.

WebSphere Message Broker

WebSphere Message Broker can also exploit the sysplex. The way in which WMB exploits the sysplex is through *WebSphere MQ shared queues*. Input and output requests can be persisted in those shared queues, which are stored in the Coupling Facility (CF). All participating LPARs and the brokers running in those LPARs will have access to the same queue image. If an LPAR o a broker in an LPAR would drop out, other brokers on other LPARs can continue processing the same workloads.

Workload scaling is also facilitated by the sysplex. Message flows can be scaled across the sysplex by deploying to multiple brokers within the sysplex broker domain. This reconfiguration process is dynamic and does not require a restart. Similarly, brokers, execution groups, and message flow instances can be removed as needed.

WebSphere Portal

WebSphere Portal for z/OS can be installed in such a way that it takes advantage of the underlying WebSphere Application Server implementation. The Portal can be installed in a WebSphere Application Server cluster for either horizontal (availability) or vertical (scalability) situations.

8.3.3 Reliability

We have seen in 8.2.3, “Reliability” on page 274 that the System z hardware and z/OS operating system provide for underlying infrastructure for reliability. One of the important aspects of reliability is the ability to run transactions and apply bulletproof recovery mechanisms in case of failures. Even though the hardware and the OS provide a robust infrastructure, reliability can only be achieved if the SOA core products within our scope exploit transaction and recovery mechanisms themselves too. As we can expect, all IBM software products in the core SOA reference architecture and the traditional backend systems are designed from the beginning to fully support transactionality and recoverability.

Again, we will discuss for each SOA core product the capabilities. We leave the backend systems, such as CICS, IMS and DB2 out of scope, but you can assume that there is nothing to worry about in those systems.

WebSphere Application Server

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through the XAResource interface and participates in distributed global transactions with transaction managers that support the CORBA *Object Transaction Service (OTS)* protocol (for example application servers) or the *Web Service Atomic Transaction protocol*. For global transactions in which both WAS and another resource manager take place and which are both run on the same LPAR, Resource Recovery Services (RRS) is exploited to coordinate transactions.

WebSphere Application Server handles transactions with three main components:

- A *transaction manager* that supports the enlistment of recoverable XAResources and ensures that each such resource is driven to a consistent outcome, either at the end of a transaction, or after a failure and restart of the application server.
- A *container* in which the J2EE application runs. The container manages the enlistment of XAResources on behalf of the application when the application performs updates to transactional resource managers (such as databases). Optionally, the container can control the demarcation of transactions for enterprise beans that are configured for container-managed transactions.
- An *API (UserTransaction)* that is available to bean-managed enterprise beans and servlets that enables such application components to control the demarcation of their own transactions.

WebSphere has administration tools enabling granular setting of the transaction properties; these properties are saved in the configuration files.

The transaction support delivered for access to backend systems (EIS, databases, transaction servers) is implemented through the *J2EE Connector Architecture (JCA)*. In this case the supplier of the backend system delivers a JCA resource adapter with the following characteristics:

- Provides JCA-compliant connectivity between J2EE components and an EIS.
- Plugs into an application server.
- Collaborates with the application server to provide services, such as connection pooling, transaction, and security services. JCA defines the following set of *system-level contracts* between an application server and EIS:

- A *connection management contract* lets an application server pool connect to an underlying EIS, and lets application components connect to an EIS. This leads to a scalable application environment that can support a large number of clients requiring access to EISs.
- A *transaction management contract* between the application server transaction manager and an EIS supports transactional access to EIS resource managers. This contract lets an application server use a transaction manager to manage transactions across multiple resource managers. This contract also supports transactions that are managed internally to an EIS resource manager without the necessity of involving an external transaction manager.
- A *security contract* enables a secure access to an EIS. This contract provides support for a secure application environment, reducing security threats to the EIS and protecting valuable information resources managed by the EIS. The resource adapter implements the EIS-side of these system-level contracts.

- Implements the *Common Client Interface (CCI)* for EIS access. The CCI defines a standard client API through which a J2EE component accesses the EIS (through the JCA resource adapter). This simplifies writing code to connect to an EIS. The resource adapter provides connectivity between the EIS and the enterprise application via the CCI.
- Implements the standard *Service Provider Interface (SPI)*. The SPI integrates the transaction, security, and connection management facilities of an application server (JCA Connection Manager) with those of a transactional resource manager.

To show a single example, let us look at the implementation for DB2 access. WebSphere application server on z/OS implements a resource adapter as shown (for DB2 access) in Figure 8-4 on page 288.

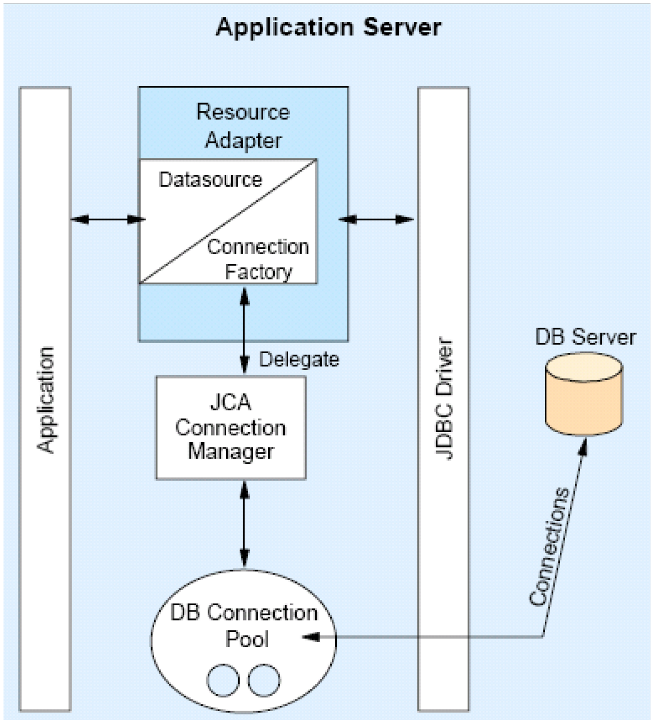


Figure 8-4 Resource adapter in J2EE connection architecture

A detailed description of the transactional capabilities that are available when connecting to backend systems (CICS, IMS, DB2) can be found in the redbook "*WebSphere z/OS connectivity architectural choices*", SG24-6365. This book describes how transactions can encompass EJBs and resources available under the control of other transaction managers.

WebSphere Message Broker

WebSphere Message Broker supports transactional message flows. RRS is used for context management and commitment control between resource managers if necessary. Figure 8-5 on page 289 shows a complex example of a transactional message flow that requires such a coordinator.

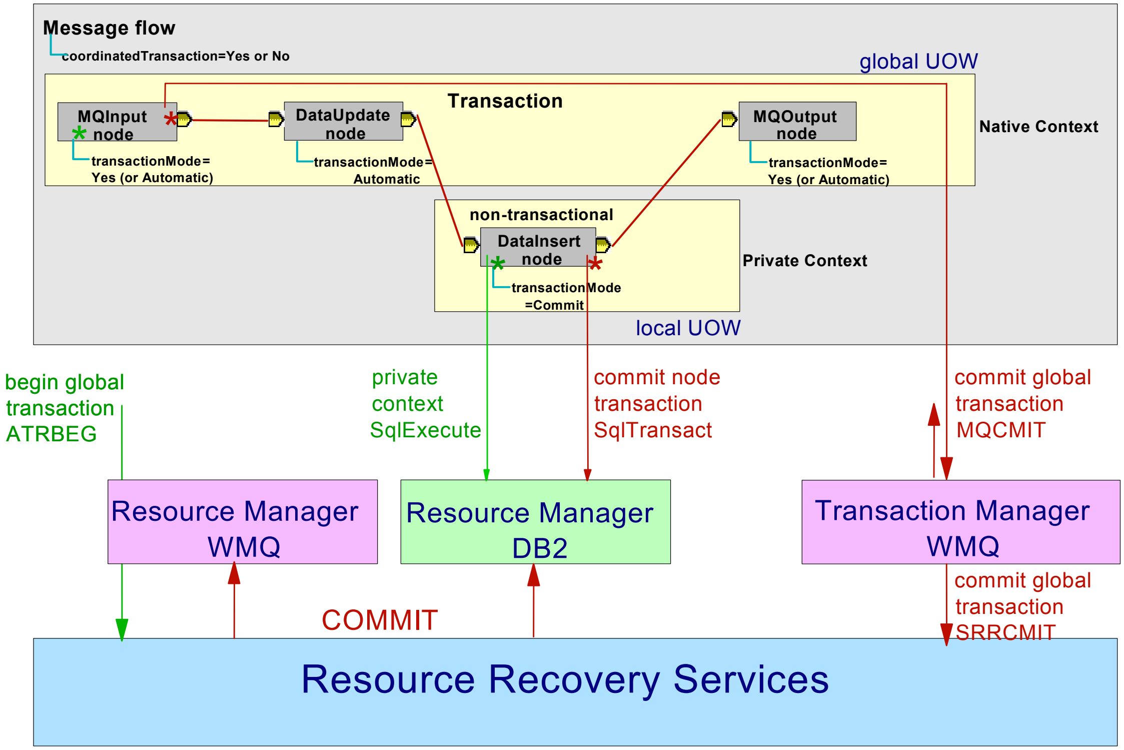


Figure 8-5 Example of transactional flow and coordination in WebSphere Message Broker

WebSphere Process Server

WebSphere Process Server takes advantage of the underlying z/OS RRS service. It offers support for transactions involving multiple resource managers using the two-phase commit process to ensure ACID properties. This capability is available for both short-running processes (single transaction) and long-running processes (multiple transactions). Multiple steps of a business process can be grouped into one transaction by modifying transaction boundaries in WebSphere Integration Developer.

Because not all service invocations support two-phase-commit transactions, WebSphere Process Server also includes recovery capabilities. If a failure occurs in the middle of running an integration application, the server detects it and allows an administrator to manage the failed event from the failed event manager.

SCA transactions are also supported. SCA presents all elements of business transactions – access to Web services, Enterprise Information System (EIS) service assets, business rules, workflows, databases and so on – in a service-oriented way.

WebSphere Enterprise Service Bus

WebSphere ESB takes also advantage of the underlying z/OS RRS service. This helps implementing transactional properties in the mediations. You can configure a mediation handler to run within a global transaction. A global transaction is required when:

- Mediating and routing messages must be coordinated into a single transaction.
- Several mediation handlers in a mediation handler list must be coordinated into a single transaction.

Setting the global transaction property ensures transactional integrity between a mediation that accesses the resources owned by other resource managers, and the messaging engine. A global transaction encompasses all the mediation operations that are run within the bus for the duration of the mediation. The global transaction ends when the mediation completes its processing. If a mediation transaction rolls back, all transactional changes also roll back. When the transaction rolls back, the mediated message remains on the pre-mediated part of the bus destination and becomes eligible to be mediated again. The redelivery count assigned to a message increments each time a mediation transaction rolls back. If the redelivery count exceeds the limit configured for the bus destination, the message is sent to the exception destination.

8.3.4 Security

The introduction of SOA brings with it additional considerations in terms of security. The concept of SOA itself, with its principle of loosely coupled and abstracted services means identity validation of both consumers and providers is harder to manage. Any application plugged into an SOA architecture is likely to have different identity mechanisms and security policies. Users will most likely have different privileges for different applications, and thus they will need to be authenticated for each of the applications that are used via the SOA framework. In addition we have to take care of the communication between the loosely-coupled elements of the new applications.

Our objectives are:

- To build an architecture that allows end-to-end identity management, namely one that is able to determine access rights for every application and user involved.
- In this context, use secure connection protocols and security features to enable data confidentiality and integrity.

- Client (service requester) authentication and authorization (via LDAP and/or

Let use look at Figure 8-6 showing a diagram we have used earlier in this book, but this time with some security interaction points. In such an architecture with quite some integration a proper security design becomes important.

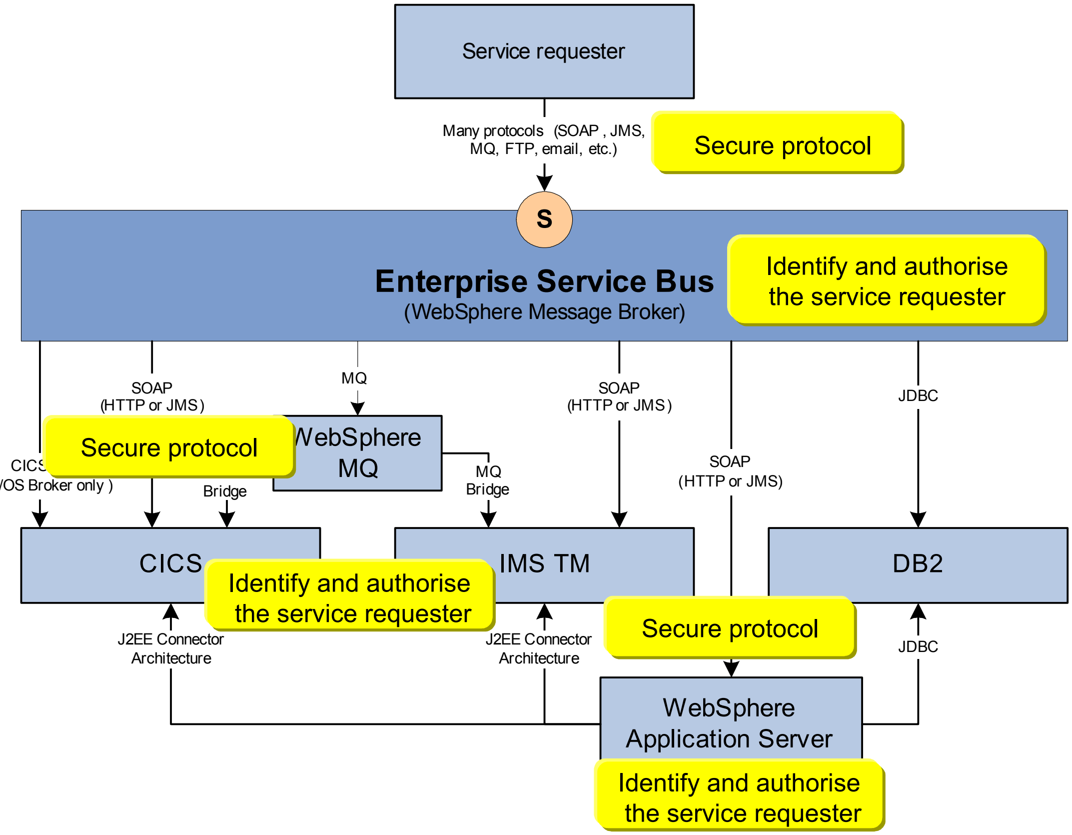


Figure 8-6 Example of security interaction points in a SOA architecture - WMB, WAS, MQ, backend transaction servers and DB2

As we said, we must introduce secure protocols and identify and authorize the requesters at specific points in the architecture. There are a lot of options available. For example, for the authorization of the service requestors, the security interaction point might be placed, depending on the architecture implemented, either in the Web Services gateway (see Figure 8-7 on page 292), or in the WebSphere ESB (see Figure 8-8 on page 292).

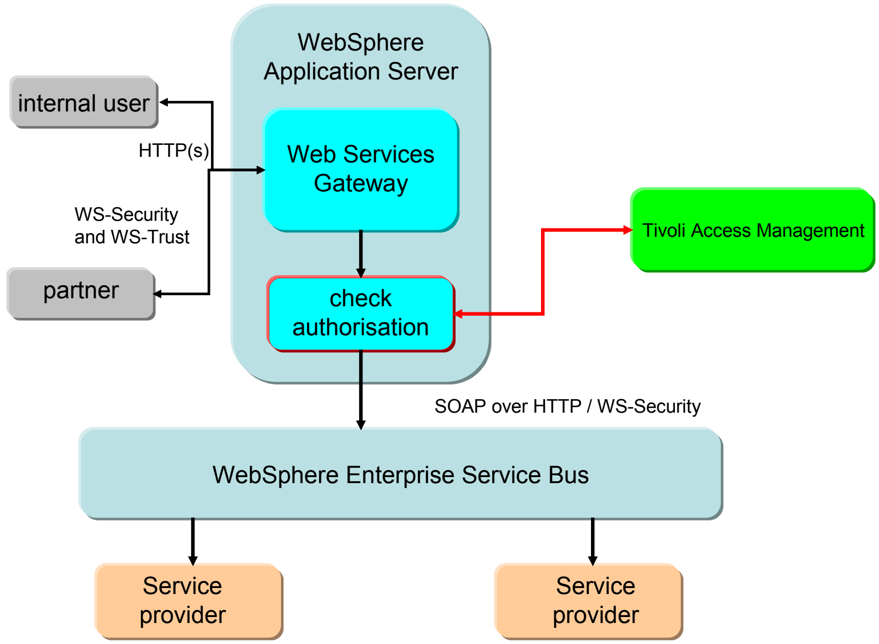


Figure 8-7 Security authorization placed in Web Services Gateway

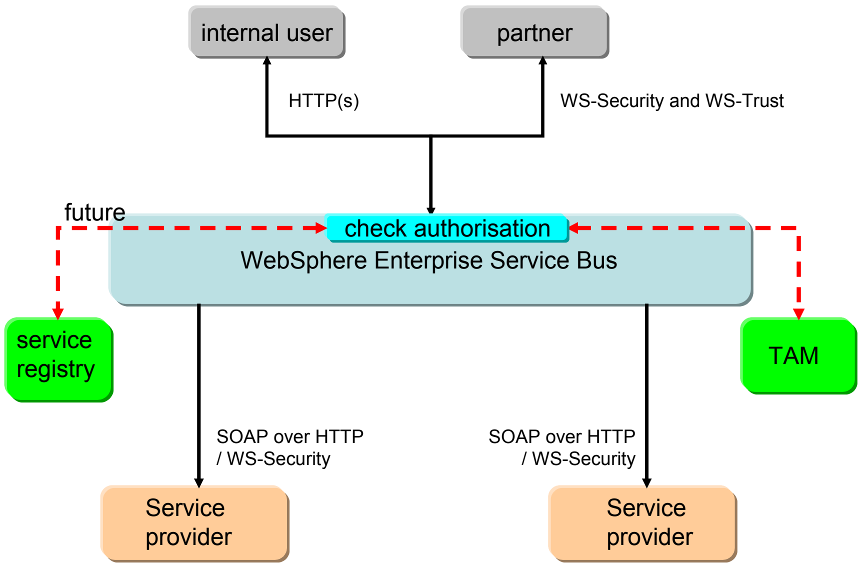


Figure 8-8 Service authorization positioned at WebSphere ESB

The scope of the security discussion here is limited. We show in the following sections that the IBM products used in the SOA architecture blocks have security interaction points and use the security infrastructure of z/OS in a consistent way. In order to show that we go in the next sections in turn over each product. For each product we will use the same structure:

- Show security interaction points for the components of the product and which resources are protected at that point.
- Show the security products that are used. We show if the RACF is used, or a plugable registry, or just configuration files of the product.

WebSphere Application Server

z/OS security has special features that make it one of the most secure systems available. The security layers, as they are used in conjunction with WAS, are shown in Figure 8-9.

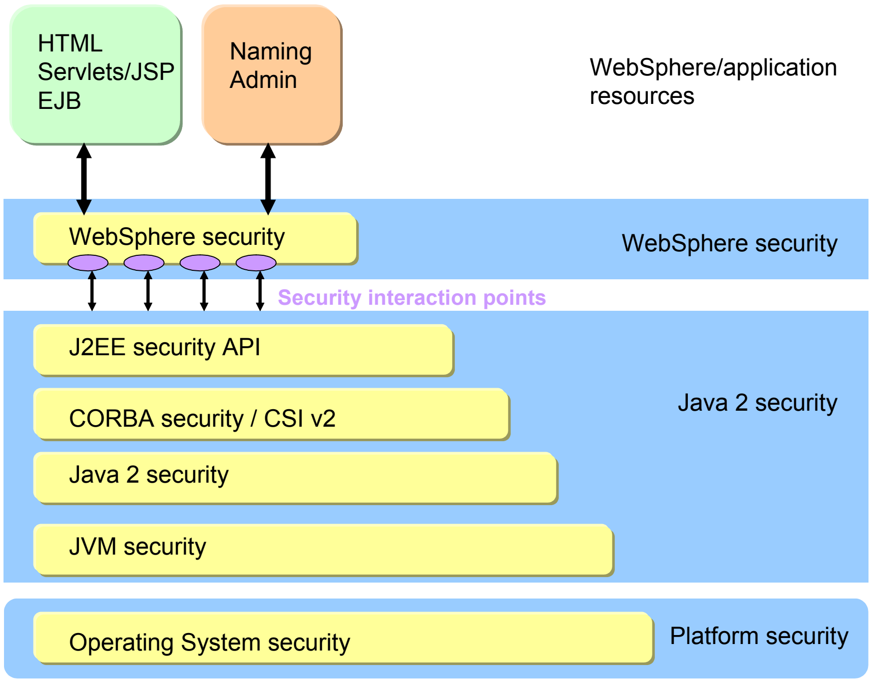


Figure 8-9 Security layers on z/OS as seen by WebSphere Application Server

We will now briefly summarize the security layers as shown in Figure 8-9.

Operating System security

At this level RACF protects the WAS configuration files and many other z/OS resources used by WAS.

Java 2 security

At this point we protect access to J2EE application components and the J2EE runtime (Java Virtual Machine). This layer also includes the “J2EE security API”, which gives control to the developer and deployer to define security (primarily authorization) on resources such as servlets/JSPs and EJB methods based on roles. Users and groups are assigned to these roles during application deployment, using deployment descriptors.

WebSphere security

At this level WebSphere global security settings protect access to system resources such as file I/O, sockets, and properties.

A more detailed view of the security in the WebSphere environment in an end to end context is shown in Figure 8-10.

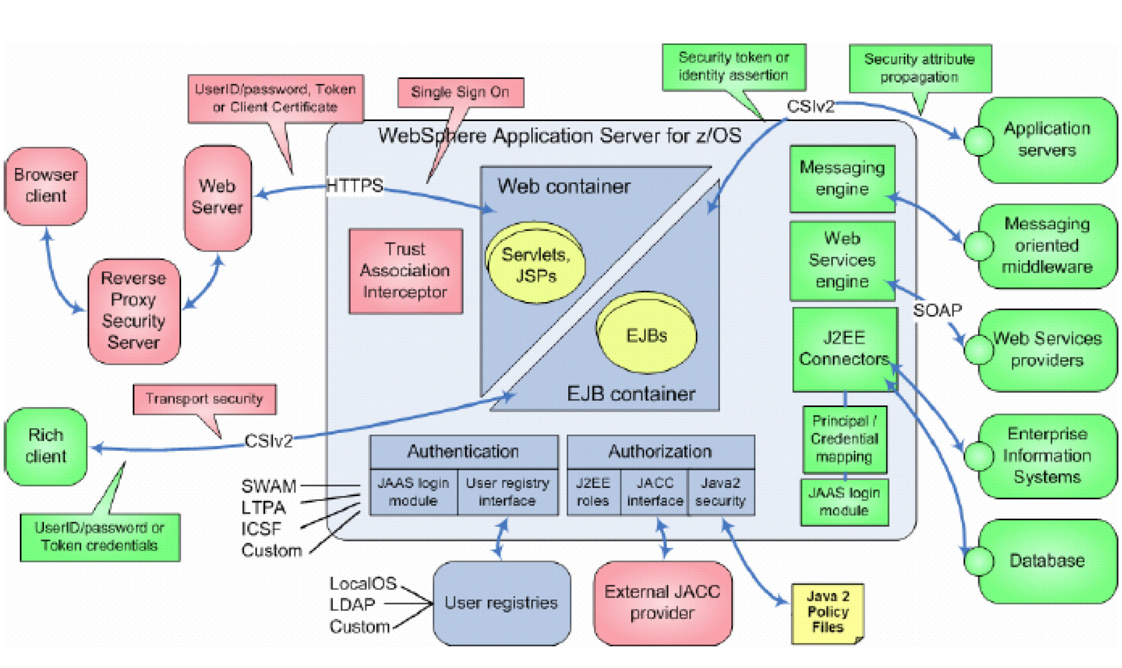


Figure 8-10 End to end security in a WebSphere Application Server for z/OS environment

Pluggable security registry

Usually the plugable registries support specific protocols. for example, in order to access Tivoli Access Manager WAS supports *Java Authorization Contract for Containers (JACC)*, which is a specific set of APIs.

J2EE container security

At this level we use the J2EE API (as previously described), *Java Authentication and Authorization Service (JAAS)* and the EJB security collaborator to authenticate Java client requests to Enterprise JavaBeans (EJBs).

To secure access to resources we can use EJBROLE. Using the APPLDATA segment of the RACF EJBROLE profile for the identity to be used for the RunAs role allows RACF control.

Web Services security

When Web Services standards are used, an additional security layer may be used, at the level of the SOAP message. Note that SOAP messages are sent inside JMS messages or HTTP requests and the security available at those levels applies as well. For example, a SOAP message sent inside an HTTP request can be encrypted using SSL.

When we talk about “Web Services security” we refer to the security that can be applied to the SOAP message and the invocation of the Web service; we do not refer to all the security features and standards available at the JMS message or the HTTP request level.

Web services messaging relies on two protocol layers: the (SOAP) *message layer*, and the *transport layer*. Security can be implemented at both layers. WAS secures the message layer by implementing *WS-Security* specifications (these specifications are shown in Figure 8-11 on page 296)

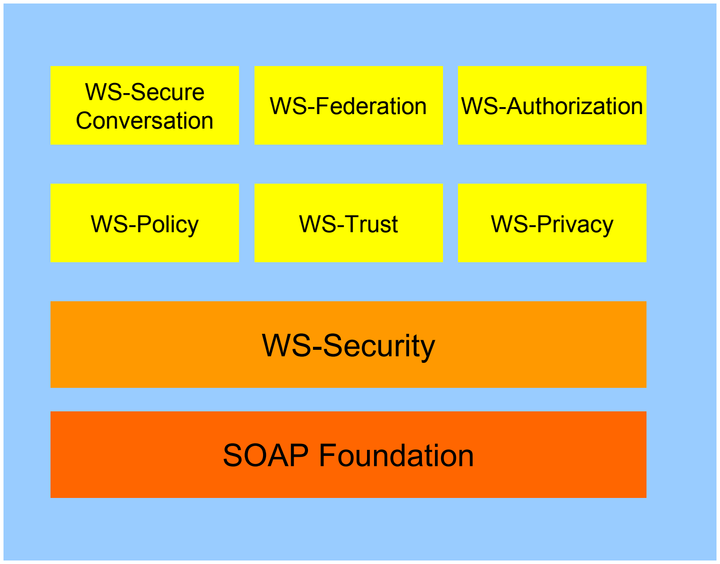


Figure 8-11 WS-Security specifications

The transport layer (HTTP, RMI/IIOP, MQ) is secured through the implementation of authentication headers and through encapsulation in the SSL/TLS.

We return now to the SOAP layer to look into more detail on how WebSphere Application Server implements the WS-Security specifications. Generally, WS-Security is a message level standard that defines how to secure SOAP messages, using:

- *XML Digital Signature*
  Digitally sign the SOAP XML document, providing integrity, authenticity, and signer authentication (JSR 105 describes the Java programatic implementation).
- *XML Encryption*
  Process for encrypting data and representing the result in XML providing confidentiality (JSR 106 describes the java programatic implementation).
- *XML Canonicalization*
  Provides normalized XML document that can be digitally signed and verified.

WebSphere Application Server supports the following WS-Security authentication mechanisms via the insertion of a security token:

- *Basic Authentication*
  The security token includes the user name and password information, and is generated as <wsse:UsernameToken> with <wsse:Username> and <wsse:Password>.

- *Signature*
  The security token includes the X.509 certificate of the signer of the data and is generated as <ds:Signature> with <wsse:BinarySecurityToken>.
- *ID assertion*
  ID assertion includes a user name only, since the identity is asserted, and is generated as <wsse:UsernameToken> with <wsse:Username>.
- *Custom*
  This mechanism includes a custom-defined token.
- *LTPA*
  Use of an LTPA token is a WebSphere-specific customer token, generating a <wsse:UsernameToken> with <wsse:Username>.

For each of the options there is extensive support available in the WAS configuration dialogs. The Web Services security constraints are defined in the IBM extension deployment descriptor and the binding file based on the Web Service port. At the application level this means that the Web Services configuration is stored in the two following extension deployment descriptors:

  – ibm-webservices-ext.xml
  – ibm-webservices-bnd.xml

Web container security

At this level the Web container interacts with RACF to:

- *Authenticate the Web client* (granularity by URI). The behavior is specified in the WAS configuration file.
- *Authorize the Web client.* The behavior is specified in the WAS configuration file.

Backend resource access security

At this point WAS makes available several options for securing access to backend resources. We need to make sure that:

- access to backends is secured
- the connection to backends is a secure conduit
- the identity of the requester with all its security attributes is propagated to the backend

There are some general options: we can choose whether to use common identity (JAAS aliases) for connection pool or to project or assert an end user identity.

In order to secure the access to CICS we can use CICS Identity Projection. Two methods are possible:

- *Thread Identity* support for connection identity for local connections.

- *Identity assertion* of a WebSphere provided identity to CICS.

The connection to CICS supports SSL, and the SSL parameters are defined using the J2C Connection Factory - SSL Custom Properties (configuration file in WAS).

In order to secure access to IMS we can use IMS Identity Projection. two methods are possible.

- *Thread Identity* support for connection identity for local connections.
- *Identity assertion* of a WebSphere provided identity to IMS (under specific conditions and under the usage of IMS Connect trusted user support).

The connection to IMS supports SSL, and the SSL parameters are defined using the IMS J2C SSL Custom Properties (configuration file in WAS).

In order to secure access to DB2 we can use thread identity and thread security support for local connections.

WebSphere Process Server (WPS) and WebSphere Enterprise Service Bus (WESB)

WebSphere Process Server and WebSphere Enterprise Service Bus for z/OS are built on top of WebSphere Application Server for z/OS and, therefore, inherit the security capabilities of WAS.

SCA security

SCA provides its own capabilities for security. SCA definitions are observed by WebSphere Process Server and WebSphere Enterprise Service Bus everywhere in their architecture and development. SCA applies security by defining quality of service qualifiers. The two SCA qualifiers for security relevant to WebSphere Process Server and WebSphere Enterprise Service Bus are:

- *SecurityIdentity*
  This is the J2EE role under which a component will be executed - regardless of the invoking J2EE role.
- *SecurityPermission*
  The required J2EE role to invoke an operation.

Message level security

Message level security is concerned with the flow of messages in transit. There are many ways to secure these messages, including using the WS-Security specification that is supported within WebSphere Process Server and WebSphere Enterprise Service Bus for z/OS. The security activities are:

- *Authentication of users* when they attempt to connect to a Service Integration Bus. Users attempting to establish a connection might have to provide a user

ID and password. These are authenticated against the same registry that the application server uses. Further access checks on the user name can be performed when the connection accesses a destination (to send or to receive a message), creates a temporary destination, or accesses a foreign bus.

- *Ensuring confidentiality and integrity* of the messages in transit. To ensure that communications are secure you can set up secure transport chains and select secure transports to protect the data transmitted along the link using SSL or HTTPS. WS-Security can also be used for SOAP messages, especially if they are asynchronous, because no handshake is needed with it.
- *Control of access to bus* for the Message Engines.

Runtime security

Security in WebSphere Process Server and WebSphere Enterprise Service Bus is controlled by definitions in the configuration files, and by access to the native or the pluggable registry. The configuration files include aliases to which default users are mapped and security roles to which users must be granted access in order to invoke these components. The components with predefined aliases and roles are:

- Business process choreographer — aliases and roles
- CEI — aliases
- SCA — aliases and roles
- Human tasks engine — roles

Applications security

Securing applications is enabled by setting global security (in the configuration file). At this level we are concerned with authentication and access control.

- *Authentication*
  Enabled when global security is on, then clients must be authenticated. The main authentication methods for clients are:
  - Web clients: HTTP Basic Authentication.
  - Java clients: Java Authentication and Authorization Service (JAAS.)
  - Web services: WS-Security/SOAP authentication.
  - Additionally all of these clients can use SSL authentication, or Lightweight Third Party Authentication (LTPA). This mechanism allows choosing between two different authentication possibilities (local user registry, or LDAP (local or remote)).
- *Access control*
  Is implemented usually by assigning J2EE roles to components.

Adapter security

Adapters have to be secured as well, depending on the information they process and their connections to specific Enterprise Information Systems. Within WebSphere Process Server or WebSphere Enterprise Service Bus, an adapter is an SCA import or export. This import or export can have SCA security qualifiers defined for it to determine the role under which the adapter runs and the role that is required to be authorized to access the adapter.

Enterprise information system specific security information (such as the user ID that the transaction in the enterprise information system should run under) can usually be specified in the connection factory of the adapter.

Figure 8-12 shows the way WPS and WESB integrate in the existing security layers on z/OS and WebSphere.

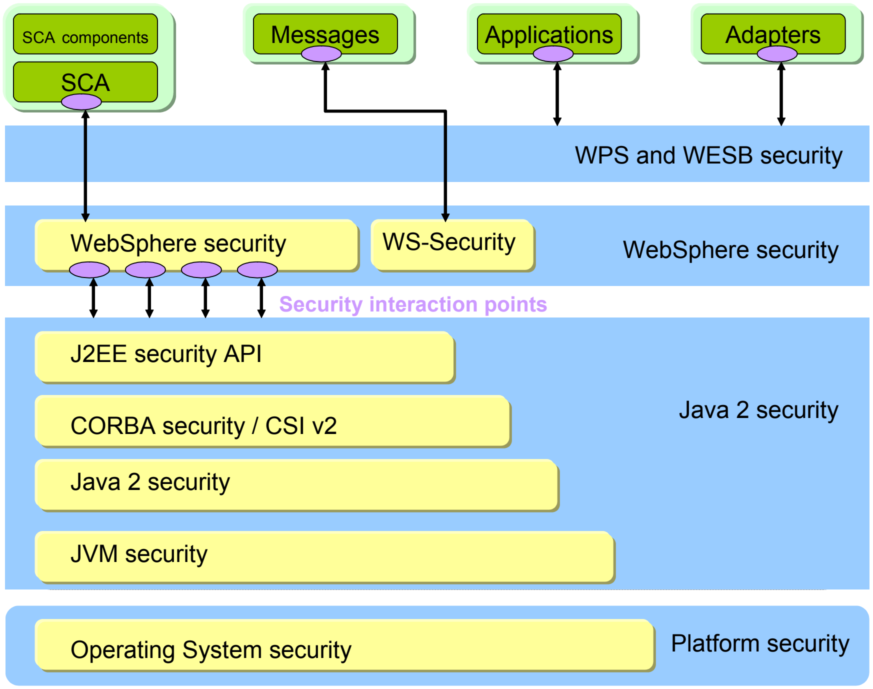


Figure 8-12 WPS and WESB security

WebSphere Portal

Security for Portal Server on z/OS is provided through the *Custom User Registry (CUR)* feature of WebSphere Application Server on z/OS.

Portal users can typically be a few hundred existing z/OS intranet users who probably have user IDs managed through RACF, and potentially thousands of new Internet or extranet users. The challenge is to provide these new users direct and secure access to transactions and data via the Portal. These new Portal users do not need RACF logon to the z/OS system, but do need to be authenticated and granted access to the specific application they need to run which is handled by WebSphere Portal via a portlet. This is where a separate registry using the *Lightweight Directory Access Protocol (LDAP)* as a Custom User Registry comes into play.

Figure 8-13 shows an overview of the security implementation of WebSphere Portal Server on z/OS.

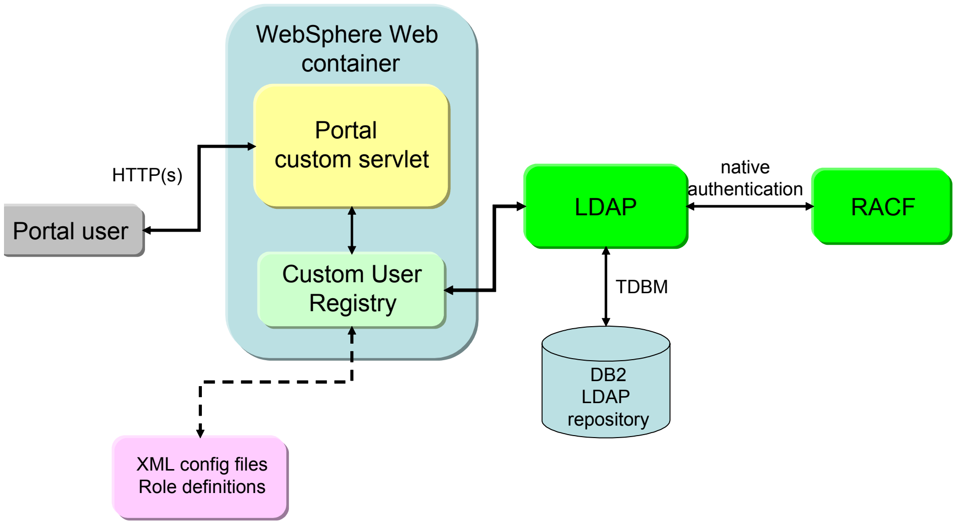


Figure 8-13 Security implementation for WebSphere portal on z/OS

Portal security

WebSphere Portal services requests from users or Web clients by authenticating a user ID and password against the Custom User Registry using the Portal Custom Servlet. It provides then the first layer of protection to internal Portal resources, such as portlets, places and pages. For CUR, an authorization table is provided via an XML file. Several files play an important role here:

- *authtable.xml*
  This XML file contains the authorizations for each Web application installed on the J2EE server for which the custom user registry is being used to authenticate requests. The authorizations are based on roleName and groupName definitions.

- *authtablelist.xml*
  This CUR authorization table XML file is used to define the applications and its authorization lists. The authorization table is managed by the administrator to grant users and groups access to the J2EE resources on a per application/portlet basis.

The native authentication feature uses LDAP with TDBM, but from a z/OS perspective the authentication is actually performed by RACF using all its usual stringent rules. From a management perspective there is no need for administration of multiple registries or synchronization of passwords. More importantly, from a WebSphere Portal perspective, RACF users and non-RACF users can be defined in the same LDAP directory and Portal users are unaware of any differences from what is normally done to log into the Portal.

Native authentication allows connection between the LDAP server and RACF wherein the user ID and password that is used to authenticate to LDAP is actually passed to the System Security Server to be verified. This setup allows new Internet or extranet Portal customers to authenticate directly against the LDAP server, while existing RACF Intranet users would be authenticated using their RACF user ID and password.

WebSphere Message Broker

The following elements can be secured at this point:

- *Topic-based security*
  Access to messages on particular topics is controlled using Access Control Lists (ACLs).
- *Authentication services using real-time nodes*
  An authentication protocol is used by a broker and a client application to confirm that they are both valid participants in a session. This is done through *SSL authentication* protocol known as mutual challenge-response password authentication. This protocol is supported by Real Time nodes, HTTP listener and WebSphere MQ java client.
- *Message protection using real-time nodes*
  Message protection provides security options to prevent messages from being read or modified while in transit.

8.3.5 Total cost of ownership

In this section we give just a few examples of the way TCO for the SOA products is decreased. 2.7, “Analysis of the IBM products available for the SOA on z/OS” on page 27, where we describe in detail the features of the products available as SOA implementation blocks, shows that each version of the product delivers:

- Enhancements in *tooling* for development, deployment, maintenance, problem handling and resolution, general management (means reducing personnel costs in deployment and management).
- Enhancement in *performance* (means reducing consumption, optimizing the resource usage, better workload management and distribution).
- Enhancements in *technology* (means better productivity in using the products).
- Enhancements in *integration* (means reducing costs in integrating the product with the other elements of the SOA architecture, reducing the TCO for the whole SOA platform).

WebSphere Message Broker

There are a number of features that improve the TCO for WMB on z/OS. Among others we mention reporting and chargeback through SMF, using zSeries Application Assist Processor (ZAAP) for the Java compute Node, XLST Node, JMS real-time node. This means that message transformation will be able to offload a significant percentage of its workload to the dedicated zAAP processors. Other performance improvements (in areas like parser, ESQL, aggregation implementation, and others) increase performance and consequently reduces the TCO.

WebSphere Application Server

As described before, the TCO can be reduced by several means.

One of them is reducing the cost for development, deployment and management. This was done through the implementation of JDK 5.0 innovations, a new Automation Toolkit, tight integration with Rational tools, Application Server Toolkit (AST) enhancements, new Console command assistant, bundle with IBM support assistant.

Another way of enhancing the TCO is making sure that the product is more efficient, more secure, and these capabilities should be delivered “out of the box” as standard. This was accomplished, among others, through enhancement in scaling and workload prioritization, implementation of J2SE™ 5, zAAP usage.

Yet another way of reducing the TCO is to enable easier integration with other products of the same family and with products of other vendors (this reduces integration costs and implicitly decreases the TCO). This was accomplished through the implementation of new technologies (SIP servlet support), implementation of the newest Web Services standards (Web Services Notifications WS-N, Web Services Interoperability Basic profile WS-I BSP, enhancements in WS-Security, and others).

8.4 Quality of service of the SOA architecture

In this section we review the IBM SOA reference architecture and we show which of the its feature contribute to specific areas of QoS. The way in which an application is positioned over the SOA building blocks, the way in which it exploits the SOA building blocks, is "creating" QoS features.

SOA brings by itself, by its own nature, Quality of Service:

- Increasing flexibility, by allowing the changing, replacing and new positioning of services (remember that we are in a loose-coupled environment) without touching the other parts of the application.
- Reduce the TCO by reusing services (services have standardized interfaces and therefore the reusability is increased).
- Reducing the "time to market" by having the flexibility to react to any market developments (expand services as required without impacting existing customers, change business rules on the fly with the result of flexible processes).
- Increase availability of the application by "cloning" of services.
- Build high availability architectures by adding service redundancy.

8.5 QoS and our implementation scenarios

In the following sections we will talk about how the QoS discussed earlier in this chapter apply to the variety of implementation scenarios, discussed in Chapter 5, "SOA implementation scenarios" on page 129.

We have defined an *implementation scenario* as a combination of a *transition approach* and one or more *solution techniques*. An example of a transition approach is "Improve" and an example of a solution technique is "CICS Service Flow Feature". Each of the solution techniques offers a certain level of QoS in the various areas we discussed earlier in this chapter and it is not that hard to find out those QoS for an individual solution technique. The challenge, however, is to assess the End to End QoS of a combination of solution techniques. If two solution techniques being applied are both highly secure, it does not mean that the end to end solution will be highly secure. We will see that a lot depends on the integration capabilities of the solution techniques in terms of QoS.

Note to Reviewer: Need to fill in QoS for WebSphere II

8.5.1 Scalability in our implementation scenarios

An SOA scales if the end to end solution scales. This means that in all layers (hardware, OS, middleware) and tiers, the SOA needs to be able to scale. The solution techniques being applied and the environment in which they run (if not a stand-alone solution) must be scalable. We have dicussed scalability attributes of the SOA core products in 8.3.1, “Scalability” on page 276, but we haven’t really talked about the individual solution techniques yet.

Table 8-1 shows for each solution technique whether it runs by itself or inside a managed environment (such as WAS or CICS) and what scalability characteristics are available.

Table 8-1 Scalability in our solution techniques

Solution technique	Tasks accomplished	Managed environment (runs in)	Scalability characteristics
WebSphere Application Server	Run business application services	Stand-alone	► WLM enablement ► Clustering of WAS servers (ND topology) ► WAS XD features (ODR)
WebSphere Portal	Run user interaction services	WebSphere Application Server	► See WebSphere Application Server
WebSphere ESB	Service integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere Process Server	Process integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere MQ	Service integration	Stand-alone	► WLM enablement ► Clustering of queues

Solution technique	Tasks accomplished	Managed environment (runs in)	Scalability characteristics
WebSphere Message Broker	Service integration	Stand-alone	► WLM enablement ► Clustering of brokers within a sysplex
WebSphere Information Integrator	Service enablement	TBD	► TBD
Host Access Transformation Services (HATS)	Service enablement of a 3270 program	WebSphere Application Server	► See WebSphere Application Server
CICS TS V3.1	Run business application services	Stand-alone	► WLM enablement ► Clustering of CICS servers through CICSPLEX
CICS Service Flow Feature (SFF)	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Web Services Support	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Transaction Gateway (CICS TG)	Service integration	► JCA resource adapter runs in WAS ► CICS TG Daemon runs stand-alone (only required in remote setup)	► See WebSphere Application Server ► Scalability through implementation of multiple CICS TG Daemons on one or multiple LPARs

Solution technique	Tasks accomplished	Managed environment (runs in)	Scalability characteristics
IMS V9 TM	Run business application services	Stand-alone	► WLM enablement ► Clustering of IMS servers through IMSPLEX ► Built-in scaling through internal prioritization and workload management.
IMS Connect	Service integration	Stand-alone (runs in own address space on z/OS)	► Scalability through implementation of multiple IMS Connect address spaces on one or multiple LPARs
IMS MFS Web Services Support	Service enablement	► Service interface runs in WAS ► Service runs in IMS ► IMS Connect used as connector	► See WebSphere Application Server ► See IMS V9 TM ► See IMS Connect
IMS SOAP Gateway	Service enablement	► IMS Soap Gateway does not run on z/OS ► Service runs in IMS ► IMS Connect used as connector	► Depends on Windows/UNIX scalability capabilities ► See IMS V9 TM ► See IMS Connect

Solution technique	Tasks accomplished	Managed environment (runs in)	Scalability characteristics
DB2 (stored procedures)	Run information services	► Service interface runs in WAS (WORF) ► Service runs in DB2 SP	► See WebSphere Application Server ► WLM enablement

8.5.2 Availability in our implementation scenarios

The availability of an SOA depends on the sum of the availability of all the components and services involved. As with scalability, this applies to layers (hardware, OS, middleware) and tiers. The solution techniques being applied and the environment in which they run (if not a stand-alone solution) must be available. We have dicussed availability attributes of the SOA core products in 8.3.2, “Availability” on page 284, but we haven’t really talked about the individual solution techniques yet.

Table 8-2 shows for each solution technique whether it runs by itself or inside a managed environment (such as WAS or CICS) and what availability characteristics are available.

Table 8-2 Availbility in our solution techniques

Solution technique	Tasks accomplished	Managed environment (runs in)	Availabilty characteristics
WebSphere Application Server	Run business application services	Stand-alone	► Clustering of WAS servers (ND topology) ► Automatic Restart Manager (ARM) ► WAS XD features, such as ODR and application versioning
WebSphere Portal	Run user interaction services	WebSphere Application Server	► See WebSphere Application Server

Solution technique	Tasks accomplished	Managed environment (runs in)	Availabilty characteristics
WebSphere ESB	Service integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere Process Server	Process integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere MQ	Service integration	Stand-alone	► WebSphere MQ shared queues ► Clustering of queues
WebSphere Message Broker	Service integration	Stand-alone	► Clustering of brokers within a sysplex ► See WebSphere MQ for MQ-related availability features
WebSphere Information Integrator	Service enablement	TBD	► TBD
Host Access Transformation Services (HATS)	Service enablement of a 3270 program	WebSphere Application Server	► See WebSphere Application Server
CICS TS V3.1	Run business application services	Stand-alone	► CICS Parallel Sysplex Manager (CPSM) ► Transaction mirroring
CICS Service Flow Feature (SFF)	Service enablement	CICS TS V3.1	► See CICS TS V3.1

Solution technique	**Tasks accomplished**	**Managed environment (runs in)**	**Availabilty characteristics**
CICS Web Services Support	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Transaction Gateway (CICS TG)	Service integration	► JCA resource adapter runs in WAS ► CICS TG Daemon runs stand-alone (only required in remote setup)	► See WebSphere Application Server ► Availability through implementation of multiple CICS TG Daemons on one or multiple LPARs
IMS V9 TM	Run business application services	Stand-alone	► Clustering of IMS servers through IMSPLEX
IMS Connect	Service integration	Stand-alone (runs in own address space on z/OS)	► Availability through implementation of multiple IMS Connect address spaces on one or multiple LPARs
IMS MFS Web Services Support	Service enablement	► Service interface runs in WAS ► Service runs in IMS ► IMS Connect used as connector	► See WebSphere Application Server ► See IMS V9 TM ► See IMS Connect

Solution technique	Tasks accomplished	Managed environment (runs in)	Availabilty characteristics
IMS SOAP Gateway	Service enablement	► IMS Soap Gateway does not run on z/OS ► Service runs in IMS ► IMS Connect used as connector	► Depends on Windows/UNIX availability capabilities ► See IMS V9 TM ► See IMS Connect

8.5.3 Reliability in our implementation scenarios

We have dicussed reliability attributes of the SOA core products in 8.3.3, “Reliability” on page 285, but we haven’t really talked about the individual solution techniques yet.

Table 8-3 on page 311 shows for each solution technique whether it runs by itself or inside a managed environment (such as WAS or CICS) and what reliability characteristics are available.

Table 8-3 Reliability in our solution techniques

Solution technique	Tasks accomplished	Managed environment (runs in)	Reliability characteristics
WebSphere Application Server	Run business application services	Stand-alone	► Built-in transaction management ► Resource Recovery Services (RRS) for global transaction coordination
WebSphere Portal	Run user interaction services	WebSphere Application Server	► See WebSphere Application Server

Solution technique	Tasks accomplished	Managed environment (runs in)	Reliability characteristics
WebSphere ESB	Service integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere Process Server	Process integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere MQ	Service integration	Stand-alone	► Resource Recovery Services (RRS) for global transaction coordination ► WebSphere MQ functions to guarantee message delivery
WebSphere Message Broker	Service integration	Stand-alone	► See WebSphere MQ for MQ-related reliability features
WebSphere Information Integrator	Service enablement	TBD	► TBD
Host Access Transformation Services (HATS)	Service enablement of a 3270 program	WebSphere Application Server	► See WebSphere Application Server
CICS TS V3.1	Run business application services	Stand-alone	► Built-in CICS transaction management
CICS Service Flow Feature (SFF)	Service enablement	CICS TS V3.1	► See CICS TS V3.1

Solution technique	**Tasks accomplished**	**Managed environment (runs in)**	**Reliability characteristics**
CICS Web Services Support	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Transaction Gateway (CICS TG)	Service integration	► JCA resource adapter runs in WAS ► CICS TG Daemon runs stand-alone (only required in remote setup)	► See WebSphere Application Server ► Reliability through 2-Phase Commit support (RRS)
IMS V9 TM	Run business application services	Stand-alone	► Built-in IMS transaction management
IMS Connect	Service integration	Stand-alone (runs in own address space on z/OS)	► Reliability through 2-Phase Commit support (RRS)
IMS MFS Web Services Support	Service enablement	► Service interface runs in WAS ► Service runs in IMS ► IMS Connect used as connector	► See WebSphere Application Server ► See IMS V9 TM ► See IMS Connect
IMS SOAP Gateway	Service enablement	► IMS Soap Gateway does not run on z/OS ► Service runs in IMS ► IMS Connect used as connector	► Depends on Windows/UNIX availability capabilities ► See IMS V9 TM ► See IMS Connect

8.5.4 Security in our implementation scenarios

Security is a complex topic and each solution technique has numerous security aspects. We can only provide the highlights for each solution technique.

Table 8-3 on page 311 shows for each solution technique whether it runs by itself or inside a managed environment (such as WAS or CICS) and what security characteristics are available.

Table 8-4 Security in our solution techniques

Solution technique	Tasks accomplished	Managed environment (runs in)	Security characteristics
WebSphere Application Server	Run business application services	Stand-alone	► Full J2EE and Java security ► Web Services security ► Authorization and authentication supported by LDAP or RACF ► Encryption supported by hardware crypto ► Thread-level security (when local) ► Single Sign On ► CSIv2 support for RMI-IIOP requests
WebSphere Portal	Run user interaction services	WebSphere Application Server	► See WebSphere Application Server
WebSphere ESB	Service integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere Process Server	Process integration	WebSphere Application Server	► See WebSphere Application Server

Solution technique	Tasks accomplished	Managed environment (runs in)	Security characteristics
WebSphere MQ	Service integration	Stand-alone	► Queue authorization and authentication ► Thread-level security (when local)
WebSphere Message Broker	Service integration	Stand-alone	► See WebSphere MQ for MQ-related security features
WebSphere Information Integrator	Service enablement	TBD	► TBD
Host Access Transformation Services (HATS)	Service enablement of a 3270 program	WebSphere Application Server	► See WebSphere Application Server
CICS TS V3.1	Run business application services	Stand-alone	► CICS security
CICS Service Flow Feature (SFF)	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Web Services Support	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Transaction Gateway (CICS TG)	Service integration	► JCA resource adapter runs in WAS ► CICS TG Daemon runs stand-alone (only required in remote setup)	► See WebSphere Application Server ► Encryption ► Authentication ► Thread-level security (when in same LPAR)

Solution technique	Tasks accomplished	Managed environment (runs in)	Security characteristics
IMS V9 TM	Run business application services	Stand-alone	► IMS security
IMS Connect	Service integration	Stand-alone (runs in own address space on z/OS)	► Authentication ► Thread-level security (when in same LPAR)
IMS MFS Web Services Support	Service enablement	► Service interface runs in WAS ► Service runs in IMS ► IMS Connect used as connector	► See WebSphere Application Server ► See IMS V9 TM ► See IMS Connect
IMS SOAP Gateway	Service enablement	► IMS Soap Gateway does not run on z/OS ► Service runs in IMS ► IMS Connect used as connector	► Depends on Windows/UNIX availability capabilities ► See IMS V9 TM ► See IMS Connect

8.5.5 TCO in our implementation scenarios

A good TCO assessment requires to look at many aspects of the solution, such as hardware utilization cost, software license cost, operational cost, maintenance cost, skill maintenance and so on. It is beyond the scope of this book to review each solution technique using those aspects. And, if we would try this, the outcome may still not be accurate without any benchmarking. What we can do is to indicate the usage level of specialty processors in each solution technique. The usage level of specialty processors, such as the zAAP will influence the TCO significantly. This will give you the opportunity to compare solution techniques for this aspect.

Table 8-5 Total Cost of Ownership: specialty processor usage

Solution technique	Tasks accomplished	Managed environment (runs in)	Specialty processor usage
WebSphere Application Server	Run business application services	Stand-alone	► Most system code and application code qualifies to run on zAAP. ► Average usage between 60% and 70%.
WebSphere Portal	Run user interaction services	WebSphere Application Server	► See WebSphere Application Server
WebSphere ESB	Service integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere Process Server	Process integration	WebSphere Application Server	► See WebSphere Application Server
WebSphere MQ	Service integration	Stand-alone	► No zAAP usage.
WebSphere Message Broker	Service integration	Stand-alone	► Java Compute Node code qualifies for zAAP.
WebSphere Information Integrator	Service enablement	TBD	► TBD
Host Access Transformation Services (HATS)	Service enablement of a 3270 program	WebSphere Application Server	► See WebSphere Application Server
CICS TS V3.1	Run business application services	Stand-alone	► CICS Java programs qualify for zAAP.

Solution technique	Tasks accomplished	Managed environment (runs in)	Specialty processor usage
CICS Service Flow Feature (SFF)	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Web Services Support	Service enablement	CICS TS V3.1	► See CICS TS V3.1
CICS Transaction Gateway (CICS TG)	Service integration	► JCA resource adapter runs in WAS ► CICS TG Daemon runs stand-alone (only required in remote setup)	► No significant zAAP usage.
IMS V9 TM	Run business application services	Stand-alone	► IMS Java transactions qualify for zAAP.
IMS Connect	Service integration	Stand-alone (runs in own address space on z/OS)	► No significant zAAP usage.
IMS MFS Web Services Support	Service enablement	► Service interface runs in WAS ► Service runs in IMS ► IMS Connect used as connector	► See WebSphere Application Server ► See IMS V9 TM ► See IMS Connect
IMS SOAP Gateway	Service enablement	► IMS Soap Gateway does not run on z/OS ► Service runs in IMS ► IMS Connect used as connector	► Not applicable ► See IMS V9 TM ► See IMS Connect

8.6 Managing QoS with SOA on z/OS

The ability to manage an SOA is a QoS by itself (manageability). And again, this is a QoS in which the System z platform is very strong. However, to be able to manage an SOA, we need more than hardware and OS features. We need to be able to manage the different levels in the entire architecture, from hardware to business process.

In the 2.4.11, "IT Services Management Services" on page 23 we discussed the necessity of having tools that manage the new infrastructure, and we discussed some Tivoli offerings. But we do not manage just every piece of the SOA infrastructure, we can manage the application as a whole.

We should be able, using the existing tooling, to answer questions like:

- Which are the services (Web services) which did not fulfil the defined SLA
- How many services of a specific type were executed during a specific time interval?
- What was the workload of services during a specific time interval?
- Which security policies are implemented at this moment?
- Who are the users that accessed a specific Web service during a specific time interval ?

Figure 8-14 on page 320 shows, as an example, the way in which Tivoli products cooperate and integrate themselves in the SOA architecture in order to manage the QoS of the applications in the area identity management, policy management and access management. The lower layer represents the management portal, where all QoS information consolidation and presentation takes place.

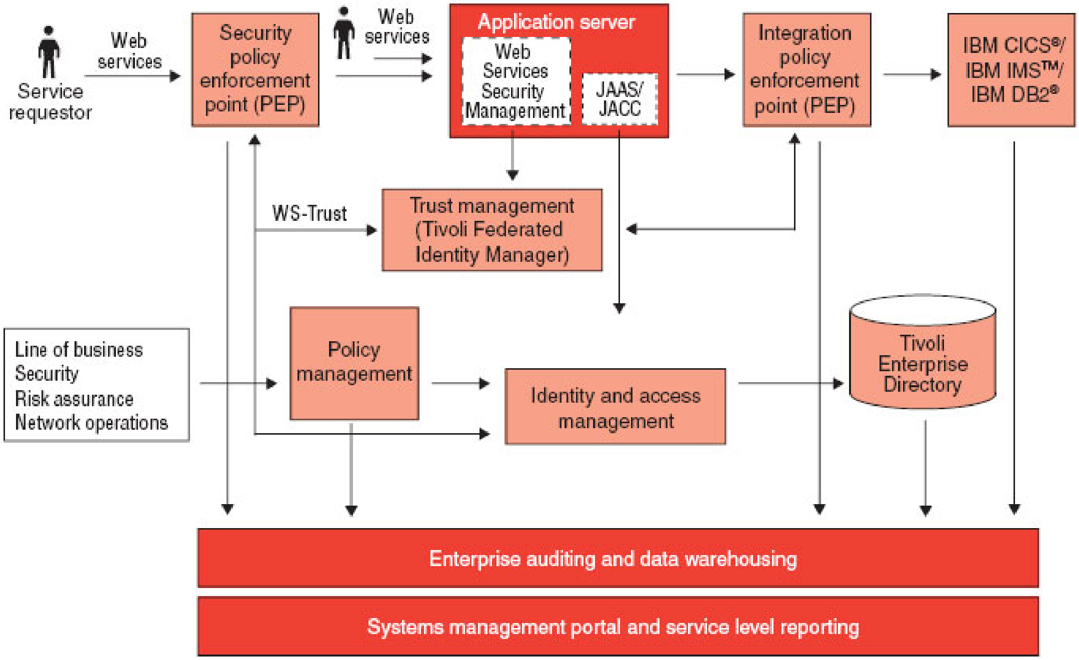


Figure 8-14 Tivoli products deliver management of QoS

8.7 Conclusion

We have shown the QoS delivered by a SOA architecture positioned on z/OS, pointing that these QoS can be seen as combination of features from the platform, the operating system, the products and the SOA architecture itself.

We have also seen that we have the means for managing (monitoring, administering, reporting) the QoS features.

9

Chapter 9.

SOA enablement case studies

In this chapter we present a two case studies about SOA enablement on z/OS. The cases studies play in different industries, have a variety of side objectives and a different solution scenario as well.

In 9.1, "SOA enablement case study 1: IBM Life Insurance Solution Showcase" on page 322 we present a SOA enablement case study playing in the insurance industry.

In 9.2, "Case study 2: A bank deployed a large-scale SOA solution based on Web services" on page 354 we present a similar case study, but in the field of the banking industry.

9.1 SOA enablement case study 1: IBM Life Insurance Solution Showcase

The IBM Life Insurance Solution Showcase provides a demonstration of Service Oriented Architecture in an insurance industry context. Its infrastructure integrates the functions of multiple Independent Software Vendors (ISV's) and IBM strategic software on multiple platforms.

It is implemented as a Variable Universal Life (VUL) policy sales demonstration and a production scale infrastructure demonstration running a realistic load of VUL sales plus other insurance industry background workloads. The production scale infrastructure showcase includes scenarios for capacity upgrades and outage situations, and the activity is monitored live for the audience during a briefing.

The showcase focuses on automating the business process of selling VUL insurance products reducing the 28 day long process to one that can be done (for the simple case) in 30 minutes or so. There is great business value in being able to sell and issue a policy to a client in one visit because it results in higher sales by reducing the chance that a client will decide against buying a policy (the “not taken” rate). The showcase demonstrates that SOA technology can be applied to make business processes more efficient and flexible, and can directly result in higher revenues, faster speed to market, and lower costs.

Although this showcase demonstration shows the selling of variable life insurance policies, the underlying IT architectural principles are applicable to the other business scenario as well.

9.1.1 SOA in an insurance industry context

SOA benefits can be applied to virtually any industry setting, but the insurance industry generally has competitive and efficiency pressures that are driving customers to look at the architecture seriously:

- Improved speed to market
- Competitive pressure
- Changing regulations and business processes
- Mergers and acquisitions
- Cost efficiency and outsourcing

The showcase makes the point that customers can use SOA technologies to improve speed, efficiency, and flexible change to meet increasingly demanding business environments.

9.1.2 The business problem

The VUL application in the showcase is the automation of the multiple step business process that is used to sell variable life insurance. Without automation, the process is a manually intensive coordination of many applications via independent and different user interfaces, coupled with the requisite handling of paper between many of the steps. The manual process has the following issues that need to be addressed:

- Eliminate delays caused by snail mail between agent, customer and main office.
- Eliminate errors caused by multiple points of data reentry, multiple user interfaces.
- Provide means to measure the effectiveness of marketing campaigns.
- Provide means to control/monitor process adherence.

The enterprise has the following requirements:

- Leverage/reuse/integrate existing IT capability and skills.
- Provide the infrastructure to enable the company to be agile and responsive to constantly changing business processes and market dynamics.
- Provide minimum risk (i.e. preference for evolutionary vs. revolutionary change, keep the business running during change, do not break the business while changing).
- Reduce the Total Cost of Ownership (TCO).

9.1.3 Enterprise view

The Enterprise view of the architecture is the high level description of how the services are delivered, as shown in

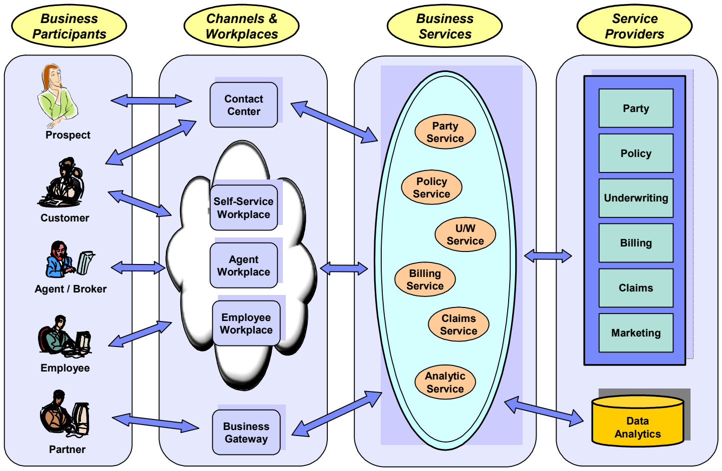


Figure 9-1 The Enterprise view of the IBM Life Insurance Showcase

The Enterprise view of the IBM Life Insurance Showcase blends business and application concepts with architectural aspects in a layered view including business participants, channels & workplaces, business services, and service providers.

It follows the style of a Service Oriented Architecture where an application is composed of the orchestration of a set of services, supplied by service providers.

- **Business Participants**
  The participants can be individuals or computer systems that need to interact with business services of the insurance carrier.
- **Channels & Workplaces**
  The participants interact with the business services through a "channel" that is responsible for providing/controlling access appropriate to the participant's needs.
- **Business Services**
  Business services are grouped into sets of discrete actions that must be performed as part of working within the insurance company. Business services may be implemented by one or more service providers. A business service may be composed of interacting with one or more service provider.
- **Service Providers**
  Service providers are applications/components that perform the "back office" level work.

9.1.4 Application architecture

The Application architecture defines and describes the application concepts and capabilities used in designing the application. These concepts and capabilities are necessary in applying technology to business design such that it supports separation of concerns, minimizes technical duplication, maximizes sharing, utilizes a common infrastructure, and supports independent construction of applications and components as well as integration. It is the application architecture that provides a solution or application to the business needs.

The Application architecture provided by the IBM SOA for Insurance Reference Architecture is a slightly modified version of the layered IBM SOA Center of Excellence solution stack. In this more technical view, the application architecture firmly supports the separation of concerns through layering. It is the responsibility of the application architecture to define the application principles necessary to guide and govern enterprise application development and maintenance.

Requirements

The IBM Life Insurance Showcase use case of "Sell Variable Life Insurance" requires the following:

- Reuse existing party infrastructure - presume that the insurance company already has WebSphere Customer Center product installed.
- Reuse existing policy infrastructure - presume that the insurance company already has LiDP "The Administrator" product installed.
- Eliminate data reentry between tasks - automate the data passing between the applications.
- Enable future channels - the agent workplace is the first channel; others will come geared towards clerks, help desk, etc.
- Enable regulatory compliance/auditing - there are regulatory requirements that must be met/proved in the selling of any investment.
- Enable rapid deployment of new offerings - the process for selling doesn't vary between types of offering, what changes is the offering contents.
- Enable targeted offerings - offerings that are geared towards a given set of clients are "targeted". A list of "targets" requires the analysis of data against some criteria.
- Eliminate paper (and associated delays) - self evident, so long as there is no regulatory requirement for paper, or use of electronic means of data transfer between parties is preferred.
- Enable measurements - provide a means to measure the process of selling insurance using a business dashboard.

- Minimize human procedural errors - humans take shortcuts, forget things, and don't know.
- Minimize human interactions - eliminate mundane, repeatable tasks, maximize the time for the humans to do what cannot be automated.

Non-Functional Requirement (NFRs)

An internal IBM study conducted by IBM Global Business Solutions focused on insurance industry non-functional system requirements and produced 23 categories that a technical architecture must deliver. The IBM Life Insurance Solution Showcase helps to show how SOA architecture in an insurance context addresses the highest priority areas, as follows

- **Productivity**
  The degree to which a system or component aids or impedes the effectiveness and efficiency of the user and/or business process. Productivity is typically quantified by measuring the elapsed time from start to end of a user performing a specific business process or scenario - including user talk and think time.
- **Performance**
  The degree to which a system or component accomplishes its designated functions within given constraints. Performance is typically quantified by averaging the elapsed times from start to end of a particular system transaction. This measurement of responsiveness can be measured from the user or component perspective.
- **Capacity**
  The requirement to have adequate resources available for the workload to complete in an appropriate time. Capacity is typically quantified by measuring peak utilization of system components and total throughput of a particular type of workload over a period of time.
- **Scalability**
  The degree to which a system or component can increase throughput capacity by adding additional resources in an effort to maintain performance objectives.
- **Availability**
  The system's readiness for use. Availability is typically quantified by the percent of time that the system is ready for use.
- **Recoverability**
  The ability to put the system back into a ready or usable state after a failure.
- **Security**
  The solution's ability to provide secure management of systems access and data.

With the exception of security, the workload scenarios are geared to highlight how the solution can contribute to each area.

Decisions

The SOA for Insurance Reference Architecture dictates the application architecture, as shown in Figure 9-2, for the IBM Life Insurance Solution Showcase. The design decisions for implementation of these layers are noted below.

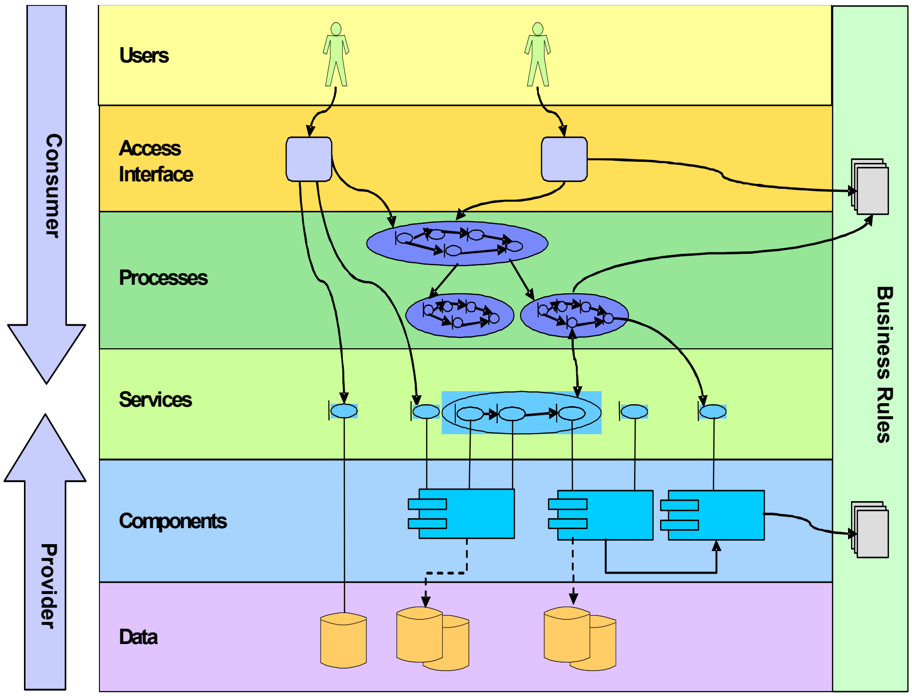


Figure 9-2 The IBM Life Insurance Solution Showcase application architecture

The IBM Life Insurance Solution Showcase application architecture follows a layered structure where applications are no longer provided by monolithic products but by the orchestration of services.

The multiple layers shown in Figure 9-2 create a separation of concerns that address the requirement set:

Access Interface Layer
Provides the view into the IT systems for different users. This enables the development of different channels. The Life Solution uses the framework provided by IBM WebSphere Portal and LDAP for security.

Processes Layer
Provides a definition of what tasks, in what order, under what conditions, need to be accomplished by what resource (user or system). This enables procedural control and monitoring. This enables the elimination of data reentry by using a user interface to capture data once to then "push it around" using the business process logic. The Life Solution uses Business Process for Execution Language for Web Services (BPEL4WS) for this process modeling as provided by IBM WebSphere Business Integration Server Foundation. (In the next phase of the application, the BPEL engine will use WebSphere Process Server).

Services Layer
Separates what needs to be done from its implementation. This enables the aggregation of existing infrastructure with new infrastructure, speeds deployment of new implementation, and standardizes interfaces to functions. ACORD[1] messages will be used as the definition of data messages that are passed around; this helps simplify composition of services for insurance.

The implementation of the services is a design decision; the Life Solution used Simple Object Access Protocol (SOAP) over HTTP for those services called by the business process. Other services used HTTP messages.

Component Layer
Provides the implementation of the service which in this reference implementation provides the "home" to existing "traditional" (monolithic) products. This layer is required to provide a services interface in SOAP over HTTP or work through a service layer provided for it.

One vender product has a proprietary socket interface that was wrapped by a service interface written by the system integration team. Other products created service layers that met the specifications of the demonstration process, whereas other products already supported a service interface that accepted ACORD messages.

Data Layer
Is the home to all the data used in the organization. Data should be accessed via some sort of service layer to provide a means to enable control access and change.

The IBM Life Insurance Solution Showcase used DB2 Content Manager to serve as the electronic repository for all the binary representation of what used to be paper, Agent information in LiDP, and WebSphere Customer Center for the client database.

1 ACORD (The Association for Cooperative Operations Research and Development) is a global, nonprofit insurance association whose mission is to facilitate the development and use of standards for the insurance, reinsurance and related financial services industries.

Business rules
Provides a codification of business level controls separate from the procedural flow. This enables the business to deploy variants to a business process without deploying a new one.

The IBM Life Insurance Solution Showcase used several products that provided "rules"; Allfinanz xpertBridge (an underwriting product) has external XML files that define the underwriting rules, LiDP has proprietary mechanism that defines policy rules.

Access Interface (channel and workplaces)

The channel architecture includes the definition and description of enterprise level concepts and capabilities of the channels and system interfaces that connect business participants to the business and IT systems throughout the enterprise.

A channel, used more commonly to describe access to the business, is comprised of both logical (agent or direct) as well as more physical channel mediums (in person or phone). An actor interface, used more commonly to describe IT level access, is comprised of various human interfaces as well as system to system interfaces.

The Life Solution was required to:

- Demonstrate users can have a centralized environment from which to coordinate the different activities required by their role.
- Demonstrate different users can have different views based upon different role.
- Use Internet standard implementation (HTTP, Web browser, thin client based solution).
- Use IBM products and selected IBM business partner ISV products.
- Use standards based messages (ACORD).
- Provide a Web services interface.

The Showcase uses the IBM WebSphere Portal Server. This ensured that the user interface would be based on Internet standard implementations and used the aggregation capabilities of a portal to provide to users a 'workplace' (collection of portlets) that suits that user's role.

When the users access the system with single signon, the credential of the user is propagated to the business process component. The security information is stored in an LDAP server.

With this approach the same portlet (capability) can be reused for different roles. For instance, a "view policy" portlet is the same for both an insurance agent and a client - the difference being an insurance agent can use the same portlet to see the policies for any of his clients whereas the client is restricted to viewing only theirs.

The IBM Life Insurance Solution Showcase implemented the "Agent Work Place" which is a single environment to be used to access the business services required to accomplish their role as insurance agent. This is shown in Figure 9-3.

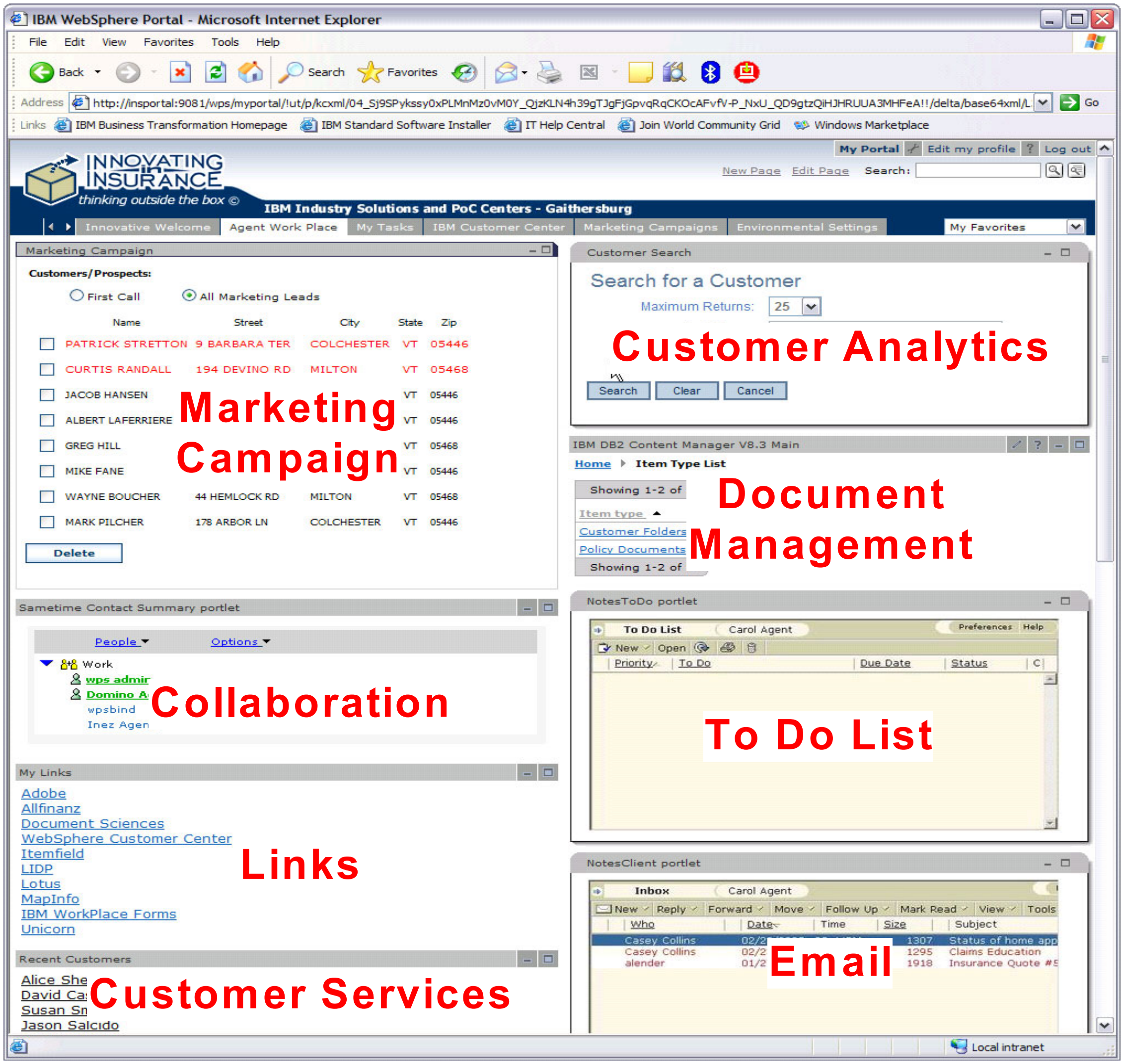


Figure 9-3 Agent Work Place user interface

Existing business process flow

The following is an approximation of the manual business process that the Life Insurance Solution automates:

1. The "Innovating in Insurance Company" uses analytic software from Mapinfo combined with client records from its client database (WebSphere Customer Center) to send out a marketing campaign letter to a set of existing clients.
2. Individual agents receive a printed list of clients to call on to sell a new Variable Life Insurance Policy - a life insurance policy backed by stocks or stock funds.
3. Either the client contacts the agent or the agent contacts the client.
4. The agent interviews the clients to collect the parameters to generate an illustration (An illustration shows the potential value of an investment over a period of time and set of economic assumptions).
5. The following steps are required to generate an illustration:
6. Enter the parameters into the policy administration system (LiDP)
7. LiDP generates the illustration
8. the secretary prints it
9. she packages it with other required 'boiler plate' documentation
10. Agent gives it to the client for review
11. The client reviews the illustration and if they wish to continue with the purchase they must first sign the illustration document acceptance agreement (a regulatory requirement).
12. The agent then assists the client in completing an insurance application.
13. The insurance application is sent to the "home office" for processing:
    a. the secretary enters the application information into the policy administration system (LiDP)
    b. she updates the client records to reflect the pending insurance sale (WebSphere Customer Center)
    c. she sends the application over to an Underwriter for review and approval.
14. Underwriter notifies the client for any requests for clarification via mail .
15. Client provides clarification via mail or email
    a. This step could include arranging for medical appointments, release of medical records, etc.)
16. Iterate until the Underwriter can make a determination.
17. If the Underwriter does not approve; a rejection letter is sent to the client.

18. If the Underwriter approves; the secretary assembles a policy package containing all the prior pieces of data, plus the requisite legal 'boiler plate' and send that to the client.
19. The client then is required to sign the policy (to enter into a contract with the Insurance company) and make the initial payment to bind the contract.

Process model of IBM Life Insurance Solution Showcase

A process is a standardized, coordinated set and flow of activities and sub-processes which collectively realize a business objective or policy goal. A business process defines the flow between business activities (the smallest unit of work meaningful to a business person).

The existing manual business process was analyzed with the help of tools like WebSphere Business Modeler. A set of processes and sub-processes are identified. BPEL4WS is used to describe the flow of activities among them. Interaction diagram is also used during design stage to visualize the data flow between the processes.

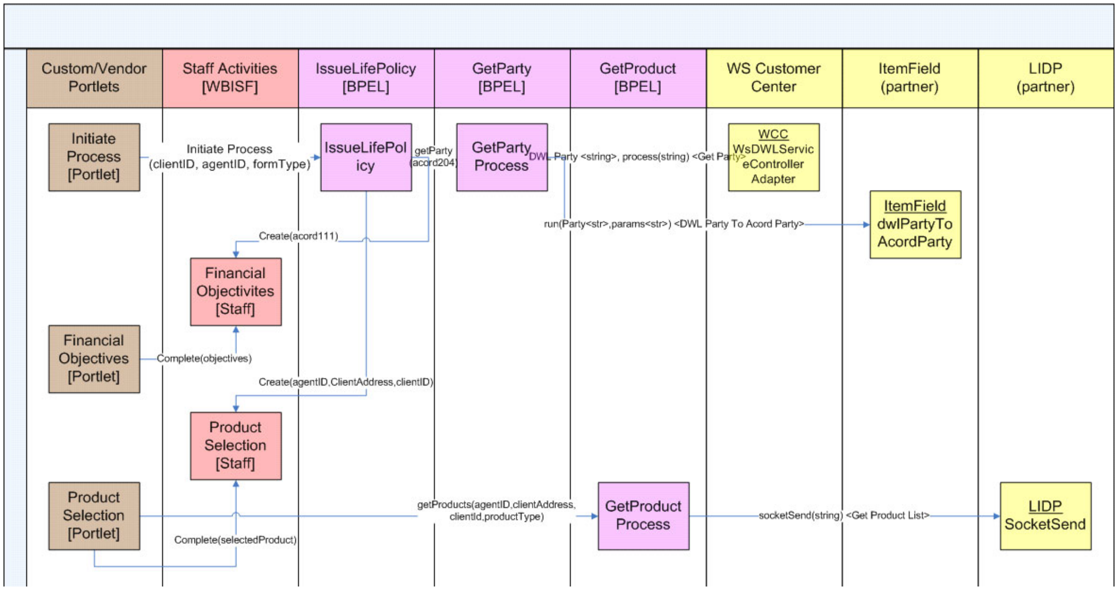


Figure 9-4 Part of the interaction diagram for the showcase

After analysis, the following processes are identified:

- `GetParty`
- `IssueLifePolicy`

- `GetProposal`
- `GetProduct`
- `CreateIllustration`
- `UpdateApplication`
- `UnderwritePolicy`
- `ManuallyUnderWrite`
- `CreatePolicyPackage`
- `ReceivePayment`

We used WebSphere Studio Application Developer Integration Edition to design these processes which utilize the services exposed in the service layer. In a WebSphere Proces Server version of this solution thistask would have been done with WebSphere Integration developer (WID).

Services and component model

A *service* is a software resource with an externalized specification. This service specification is available for searching, binding and invocation by a service consumer.

A *service component* is a realization of a subsystem, a logical grouping of functionally cohesive business aligned services, which is important enough to the enterprise to be managed and governed as an enterprise asset.

Table 9-1 shows the list of services provided to the business process model and their corresponding service components

Table 9-1 List of services to support the business model

Services Component	Services provided
Party	► `Get Party` ► `addPartyInteraction` ► `addContact` ► `updateContact`
Policy	► `getProductList` ► `generateProposal` ► `crateNewBusinessRecord` ► `updateBusinessRecord` ► `issuePolicy`
Underwriting	► “simple” underwriting ► `getUWresult`
Illustration	► Create Illustration

Services Component	Services provided
Documentation	► Compose document ► Update document ► Store document ► Creation forms ► Sign forms electronically
Geospatial	► Provide geospatial information for sales analysis
User Interaction	► Single Sign On
Data Mediation	► `DWLPartyToAcordParty` ► `Acord103toDWL` ► `DWLResponsetoUpdate`

One requirement of the showcase is it can reuse existing infrastructure provided by ISVs. For example, the Policy service component is already provided by LiDP.

The line up of service components and their realization is as following:

Party Component
For the creating, reading, updating and deleting of the data associated with the "parties" (clients, agents, etc.) who are participants in the business processes. The WebSphere Customer Center product is used to provide this functionality. Persistence is provided by DB2 hosted on zOS.

Policy Service Component
Related to insurance policies. It is realized by LiDP, but an insurance carrier can have multiple such policy service providers each geared towards specific policy 'types'. Other ISV product(s) that manages other types of policies are being looked in the future.

Underwriting Service Component
For performing (life) insurance underwriting. It is implemented by Allfinanz.

Illustration Service Component
For creating illustrations. It is implemented by LiDP but can be other "illustration" calculators. This component relies on the documentation service for formatting and presentation.

Documentation Service Component
For the creation, updating, handling, storing, and managing of documents within the insurance company. This is a composite of subsystems implemented by Document Science, IBM Workplace Forms™, Adobe and DB2 Content Manager.

The Adobe PDF format was decided upon to display to end users because of the ubiquity of the Adobe PDF viewer on most personal computers. It was not required to use this format for form presentation and capture.

Document creating and updating is the responsibility of Document Sciences. It generates PDF files and requires the functionality of an Adobe product - Adobe Form Server, to help with the differences between how Adobe vs. IBM Workplace Forms provide electronic signatures that are stored within the PDF file.

Document storage is the responsibility of DB2 Content Manager, which is used to store documents and forms used through the process. In addition, Adobe uses it to store the templates of the forms; IBM Workplace Forms opted not to deploy in this fashion for the demonstration. The document life cycle management capabilities of DB2 Content Manager were not exploited in the demonstration deployment.

There are two different approaches to human interaction. Each provides electronic form creation, data capture and human interaction (e.g. fill in the box, sign here, etc.).

Adobe - uses a product called Adobe Document Server to assemble forms from a set of templates stored in DB2 Content Manager. For the demonstration, a custom portlet was written that presented a series of screens to do form data capture (instead of using the capabilities of the Adobe browser plug-in). The Adobe browser plug-in was only used to support electronic signature capture and for final document display.

IBM Workplace Forms - uses an XML 'package' that is rendered by a browser plug-in. The demonstration portlet was responsible for creating this xml package and sending to the IBM Workplace Forms browser plug-in. The package contains everything, including MIME encoded attachments, so as to provide an audit trail of what was electronically signed for. The approach also provides for off-line work but this capability was not demonstrated. IBM Workplace forms rely on a document converter to generate the required PDF structures for electronic signatures.

Geospatial Service Component
For providing geospatial information. The IBM Life Insurance Solution Showcase demonstration is predicated on geographic data, income information and other demographics being analyzed together to select potential sales leads for a new variable life insurance offering.

Data Mediation Component
For the translation of message data between components. Itemfield provides this functionality. In early phases, this was embedded as a discrete step in the business process flow. In the current phase, Itemfield function was extracted and

made into a separate Web service component to create a service interface for general use.

User Interaction Service Component
For providing a user interface. This is implemented primarily by WebSphere Portal for the visual functions, and the WebSphere Business Integration - Server Foundation for process related actions.

Security Component
Providing control over user access and authorization. Portal security using LDAP running on z/OS was used in this implementation to authenticate the users.

Process Orchestration Component
For controlling and tracking the assignment of work to resource. IBM WebSphere Business Integration - Server Foundation, which was provided as part of the IBM WebSphere Portal Server distribution, served as the implementation. It was deployed on z/OS.

Figure 9-5 on page 337 shows the component architecture of the IBM Life Insurance Solution Showcase.

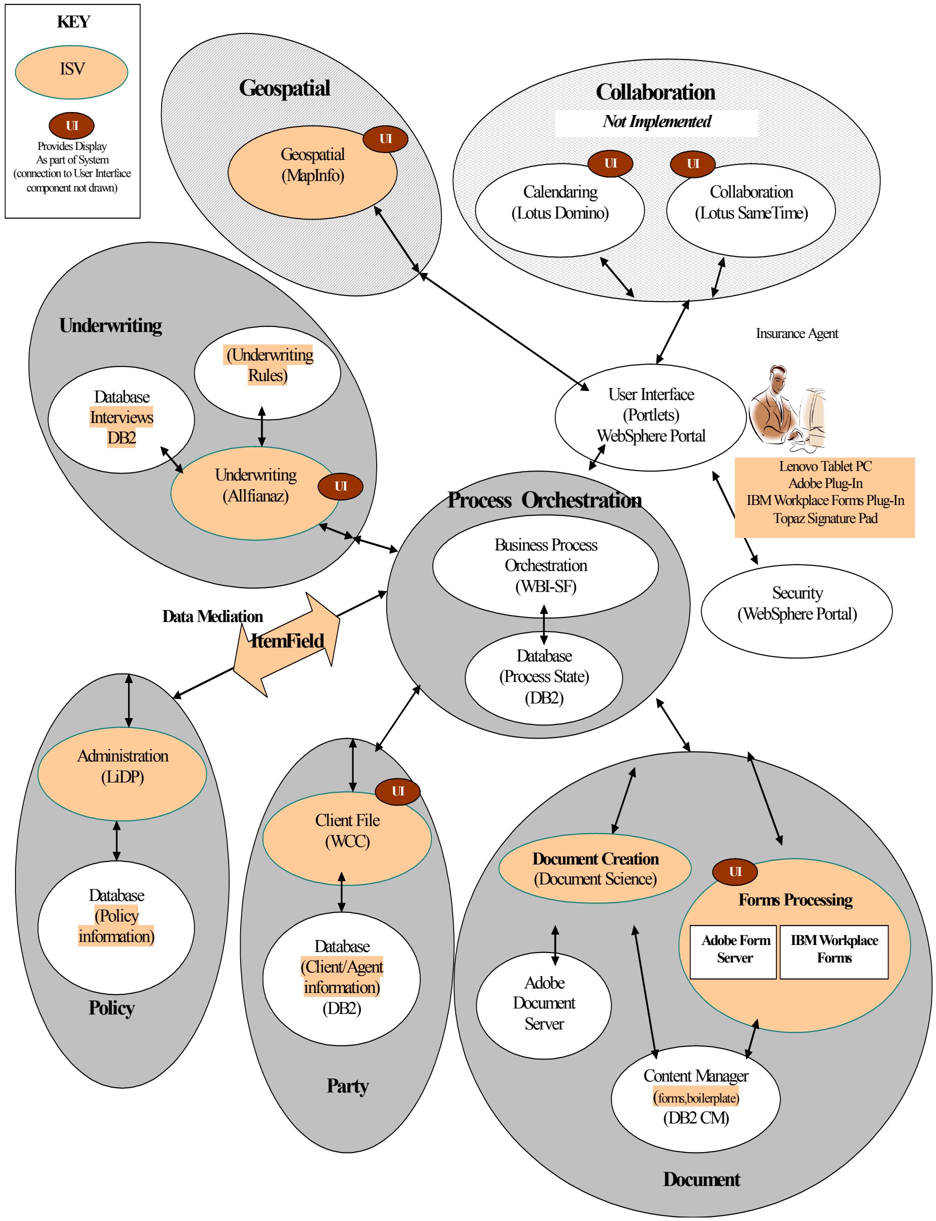


Figure 9-5 The Component Architecture showing IBM and ISV products that comprise each component

Protocol between business process and service components

SOAP over HTTP is chosen since it is most widely available among the ISVs. However, one vender product supports only ACORD/TCPIP but not SOAP/HTTP. Moreover, its TCP socket implementation is not multi-threaded, so multiple ports are opened, one for each thread. Managing which port to connect is the job of the service consumer. We developed a J2C connector with connection management so that the business process can invoke the service using SOAP/HTTP.

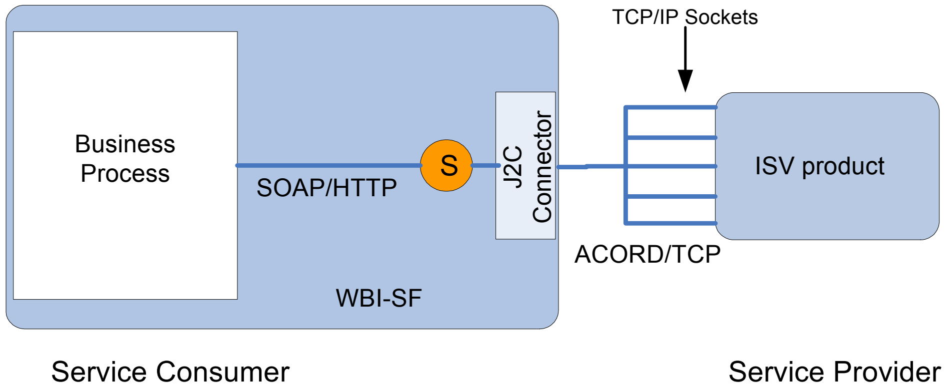


Figure 9-6 J2C Connector as a gateway between SOAP/HTTP and ACCRD/TCP

Data model

A data model consists of all information and data both structured and unstructured that represent business artifacts or aspects within the enterprise or with which the enterprise does business.

Structured information and data includes organized and formal data stores both operational and analytical, whereas unstructured information and data includes images, documents, and other text based content.

The IBM Life Insurance Solution Showcase has a "realistic" data problem; no single data repository (physical implementation) is feasible because the data is spread across multiple representations implemented by multiple products. There needs to be an enterprise level data model that federates the disparate data sources.

In the showcase, the BPEL was built with explicit steps to keep this loose confederation of application centric data bases (models) synchronized. In a production environment these synchronization tasks should be broken out into a set of separate services that would permit all business process from access/updating the "federated data stores" in a consistent manner.

All the ISVs agreed to use ACORD messages as the means to communicate, except WebSphere Customer Center which uses its own proprietary format.

WebSphere Customer Center and LiDP agreed to use a common key for correlating their two data models. The business process was responsible for keeping the two data models synchronized. For a production deployment the synchronization actions used by the business process should be turned into a service.

WebSphere Customer Center did not have time to directly consume ACORD messages and relied on data mediation functions provided by Itemfield to convert ACORD messages to/from WebSphere Customer Center formats. This mediation was encoded as part of the business logic for early phases, and was moved to a separate data mediation service in the current implementation.

ACORD messages

Here is the list of the ACORD (*Association for Cooperative Operations Research and Development*) transaction codes (aka. ACORD messages) used:

103 - New Business Submission

111 - Illustration Request

204 - Party Inquiry

228 - Producer Inquiry

301 - Party Search

500 - Form instance request

503 - Denial of Risk

505 - Holding Status Change

508 - Payment

510 - Form instance update

The ACORD `OLifeExtension` object was used to extend each ACORD message used in the showcase with the additional data required to maintain the process session identity and state throughout the process flow.

Note that the use of the `OLifeExtension` object introduces the possibility of making the ACORD messages very specific to the enterprise that is implementing the solution. Care should be taken to avoid over using extensions so that the messages will be more easily exchanged with other enterprises, outsourcers, and ISV products.

Rules model

The rules model includes the definition and description for the business rules that shape the activities and structure of the enterprise. Various types and levels of business rules exist, but in general a business rule is anything that defines or constrains one aspect of the business that is intended to assert business structure or influence behavior of the business.

The process required to sell an insurance offering does not vary significantly between offerings, the steps are similar with variations (addition / removal of steps) based on a criterion associated with a specific offering.

The SOA for Insurance Reference Architecture includes the concept of using rules to determine process flow control that separates the process steps from the variables that control which steps need to occur.

Some of the ISVs used in the Life Solution implemented within their products approaches that can be considered "rules":

- Allfinanz has rules within the Underwriting Engine.
- LiDP has rules per policy that describe characteristics of the policy and the capabilities of the agents.

9.1.5 Operational model

An operational model contains the distribution of an IT system's components onto nodes, together with the connections necessary to support component interactions, in order to achieve the IT system functional and non-functional requirements, within the constraints of technology, skills, and budget.

Design principles and decisions

The following principles were followed to drive the design decisions:

1. All the critical data is on z/OS. DB2 running on z/OS can be configured to provide a quality of service to support a highly available deployment. DB2 (used by WebSphere Customer Center) will be located on z/OS; Document Manager will be located on z/OS.
2. All applications will run near the data. The goal was to have all products run either in z/OS or under z/VM® running with SUSE Linux. This would exploit the quality of service capabilities of z/OS and z/VM. Network security issues could also be mitigated because the network connectivity within System z configured with hipersockets and VLANs is inherently more secure than wired networks.
3. Not more than one service provider per machine. This separation may not be totally representative of production deployments however, this project

required that each IBM ISV had remote access to a machine for deployment of their product and this arrangement greatly simplified coordinating this. With this configuration each ISV or IBM product installer/configurator will be capable of managing their own machine(s) permitting parallel development/testing activity and debug.

4. WebSphere Application Servers other than the WBI-SF nodes and WebSphere Portal Server will be single node (no Network Deployment - ND).
5. Support remote access. The infrastructure must provide at least SSH access for non-IBMers and IBMers alike. The solution was built in the IBM Gaithersburg center and has the following infrastructure features:
   a. Internet facing SSH servers with managed users for non-IBMers.
   b. IBM intranet facing SSH servers to support IBMers.
   c. The lab network is accessible from the IBM intranet, but the machines within the lab cannot access the IBM intranet.
   d. The machines in the lab can access the Internet.
6. Application security was handled by Portal security and LDAP on z/OS. Simple portal identity management was used in the form of a portal user ID and password.

Correlation to technical architecture

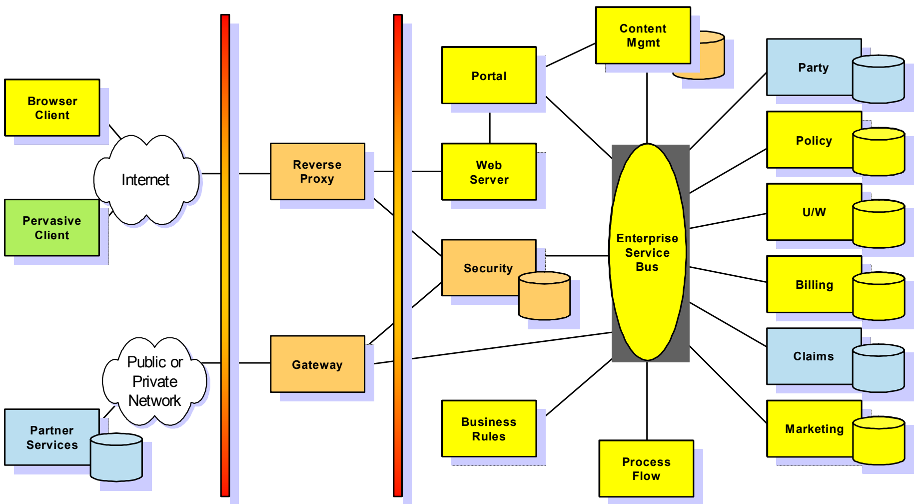


Figure 9-7 The 3-tier style view of the Component Architecture

The diagram shown in Figure 9-7 on page 341 is logical and not a single physical instance of a single server deployment but is physically dispersed, with each component deployment providing some part of the functionality of the "bus".

The communication between components is in the form of SOAP XML messages that follow the ACORD schema sent via HTTP over TCP/IP. Each component has at least an HTTP server, and those that are J2EE compliant, an IBM WebSphere Application Server.

File uploads/downloads to/from DB2 Content Manager used HTTP file transfer, but data updates were accomplished via a JAVA API and JDBC (Information Integration for Content (II4C) product).

9.1.6 Infrastructure (technical) architecture

An infrastructure or technical architecture defines and describes a shared infrastructure that provides the required qualities of service across the enterprise.

Requirements

- Position the appropriate use of the IBM eServer™ line and STG Technologies. The insurance industry is a large System z customer but the industry ISVs are not necessarily capable of running on zOS.
- Each ISV must be deployed independently of each other, are enabled to install/configure/support their product, but are restricted from accessing other ISV's environments.
- Select platforms that ISVs can properly support or fit the ISV marketing interests.
- Create an infrastructure that will enable the follow-on iterations to build out the infrastructure to support high availability use cases.
- Follow the On Demand Operating Environment Architecture.

Decisions

Rationale for the use of z/OS

- Within the insurance industry z/OS is traditionally used to house mission critical, business vital data.
- WebSphere and DB2 data sharing on z/OS provide transactional integrity and automatic fail over.
- LDAP on z/OS (which could be combined with RACF), provides the foundation for system and application security.

- The Reliability, Availability, Serviceability inherent with System z architecture and design.

Rationale for the use of Linux hosted by z/VM

- Exploit the existing hardware and skills.
- Provides a flexible, secure operating environment for each application.
- Ease of deployment of new images.
- Provides virtual network :
  - Secure (cannot be tapped)
  - Speed (memory speed)
  - Reduced network infrastructure (# of wires, hubs, switches)

Rationale of the use of LPARs & Parallel Sysplex

- Provides failover
- Provides load balancing across the partitions.
- Integrated Resource Director provides for dynamic provisioning of server resources.
- Hipersockets provides secure communication between LPARs.

The operational model shows that the spirit of the infrastructure architecture was held to but there was some design decisions that were made due to the abilities of the ISVs and IBM products.

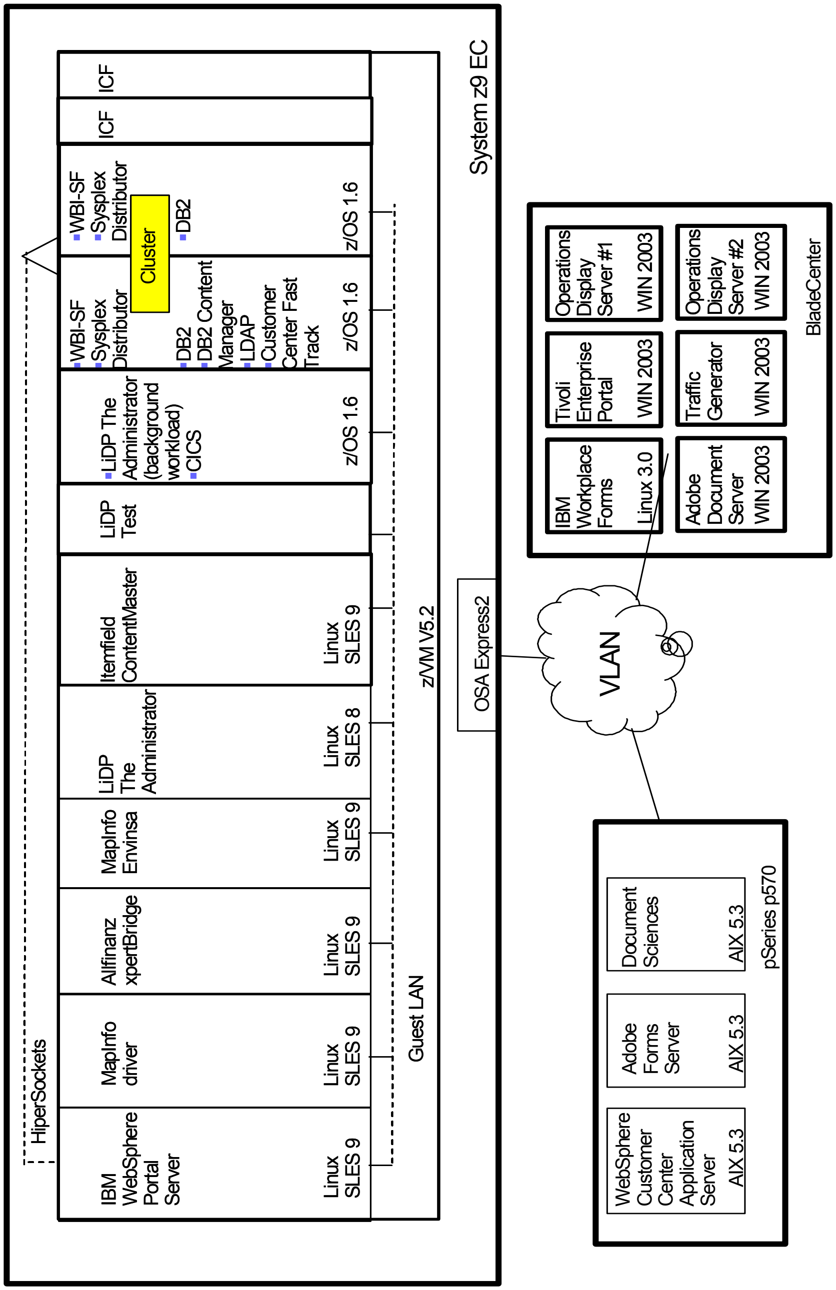


Figure 9-8 Lineup of Service Providers on the physical servers

The operating environment contains 23 independent servers, 5 different operating systems (z/OS, z/VM, Linux, AIX®, Windows 2000) and 3 platforms (System z, System p™ and BladeCenter®), The resiliency of the z/OS Parallel Sysplex provides the required business data continuity, and is paired with System p LPARs management to provide failover in the AIX environment.

Correlation to the SOA Reference Architecture

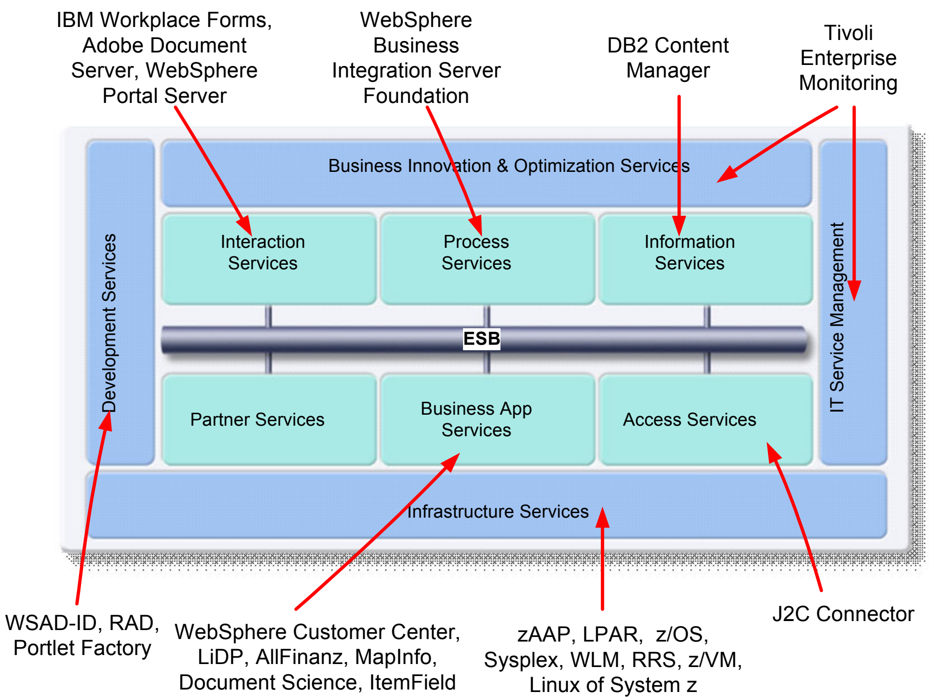


Figure 9-9 Products used in the IBM Life Insurance Solution Showcase mapped to the SOA Reference Architecture

The components that make up the IBM Life Insurance Solution Showcase functionality readily map to the SOA Reference Architecture. A primary goal was to classify this mapping and to identify future projects that will focus on implementations of some of these components as they pertain to a broader effort of IBM Infrastructure Solutions development.

9.1.7 SOA tooling

The team used standard tools for implementing and monitoring the SOA application and system environment.

The tools that were chosen were the most current at the time and would work with WBI-SF and the portal coding tasks that needed to be done. As the team migrates the systems to WebSphere Process Server, the tooling will change.

WebSphere Studio Application Developer - Integration Edition (WSAD-IE)

WebSphere Studio Application Developer - Integration Edition WebSphere Studio is a comprehensive integrated tool suite for dynamic e-business java application development and deployment to WebSphere Business Integration - Server Foundation systems. In particular it provides support for business process definition and development in business process execution language (BPEL).

WebSphere Integration Developer will be used when the team migrates the system to WebSphere Process Server.

Rational Application Developer Version 6

IBM Rational Application Developer helps developers to design, develop, analyze, test, profile and deploy high-quality Web, Service-oriented Architecture (SOA), Java, J2EE and portal applications.

It includes full support for the J2EE programming model, integrated portal development features, Unified Modeling Language (UML) visual editing capabilities, code analysis functions and automated test and deployment tools.

In the development of the IBM Life Insurance Solution Showcase, it was used mainly for Portal coding.

WebSphere Portlet Factory

IBM WebSphere Portlet Factory is used to extend WebSphere Portal capabilities with tools and technology to create, customize, maintain, and deploy portlets. WebSphere Portlet Factory's ease of use and development features streamline the portlet development process to speed WebSphere Portlet deployments. In the Life Insurance Showcase, WebSphere Portlet Factory is being considered as a way to show business metrics in place of Tivoli Audit records. This would provide a very impressive business dashboard if implemented. The Portlet Factory team is looking at the Showcase data to see what can be done in terms of a Business Dashboard portlet.

Tivoli Monitoring

IBM Tivoli Monitoring is a set of products that can be used to monitor and optimize the performance and availability of the entire IT infrastructure.

Through a single customizable workspace portal (Tivoli Enterprise™ Portal Server), the products can be used to manage the health and availability of the IT infrastructure, end-to-end, including operating systems, databases and servers, across distributed and host environments. IBM Tivoli Monitoring detects bottlenecks and potential problems in essential system resources, provides alerts, and helps to automatically recover from critical situations to ensure that business critical applications are up and running.

In this nnsurance implementation it was used to:

- Isolate problems and scalability issues in the infrastructure.
- Monitor the system health.
- Bridge between business and technical monitoring.
- Provide a view of the system activity during the briefings.

The products do not provide SOA application statistics on systems running WBI-SF, so the team implemented business critical audit records which were monitored to track and display business volumes during the briefing.

Figure 9-10 on page 348 shows a enterprise view of the servers with health indicator.

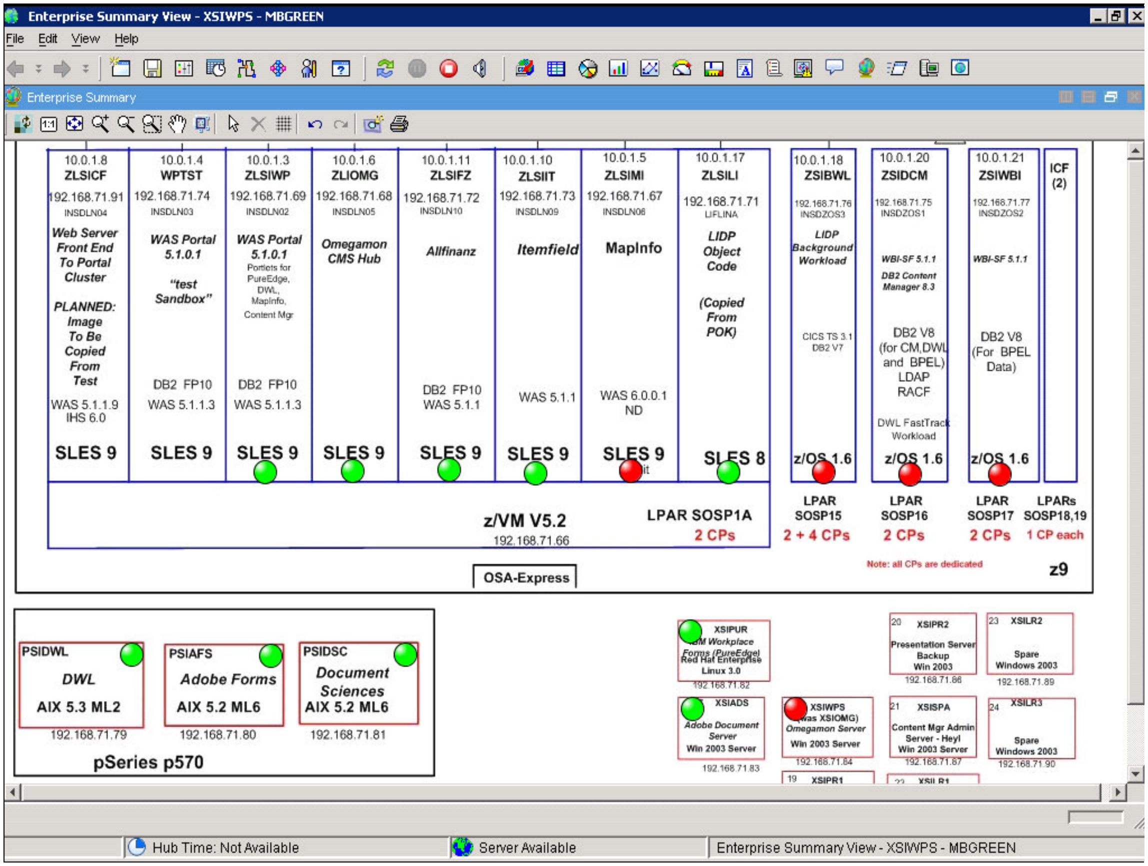


Figure 9-10 A Tivoli Enterprise Portal Server system monitoring screen

Figure 9-11 on page 349 shows the CPU utilization of each server.

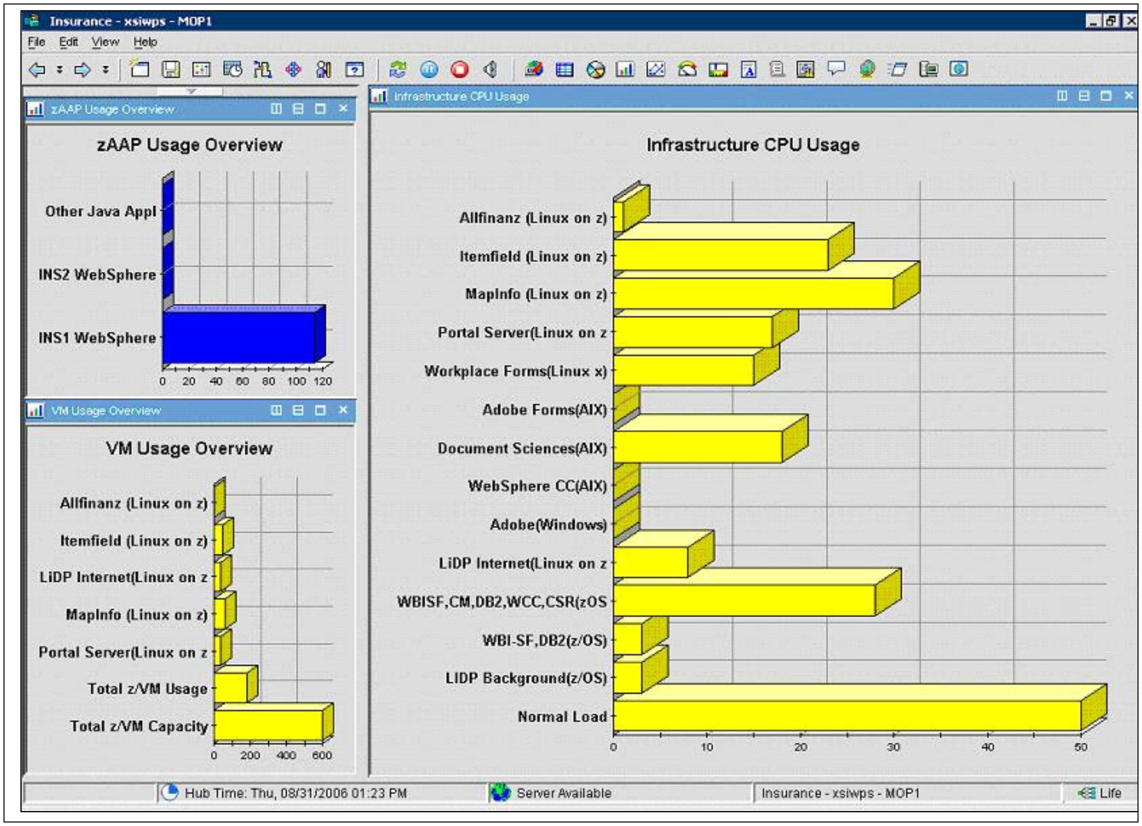


Figure 9-11 A Tivoli Enterprise Portal Server CPU utilization Monitoring Screen

Figure 9-12 on page 350 show the business transaction volume (such as agent interview per hour) and the technical resource usage (such as zAAP processor usage) on the same screen.

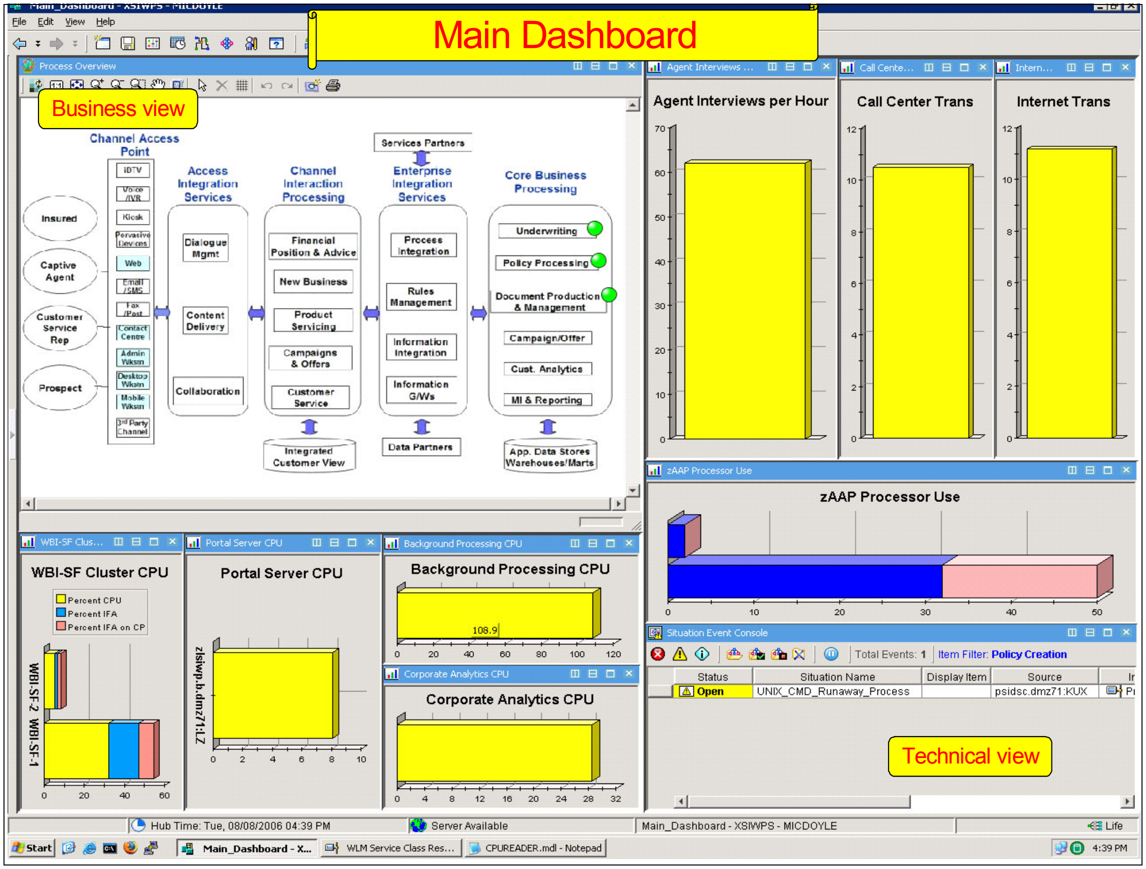


Figure 9-12 Main dashboard to bridge business and technical volumes

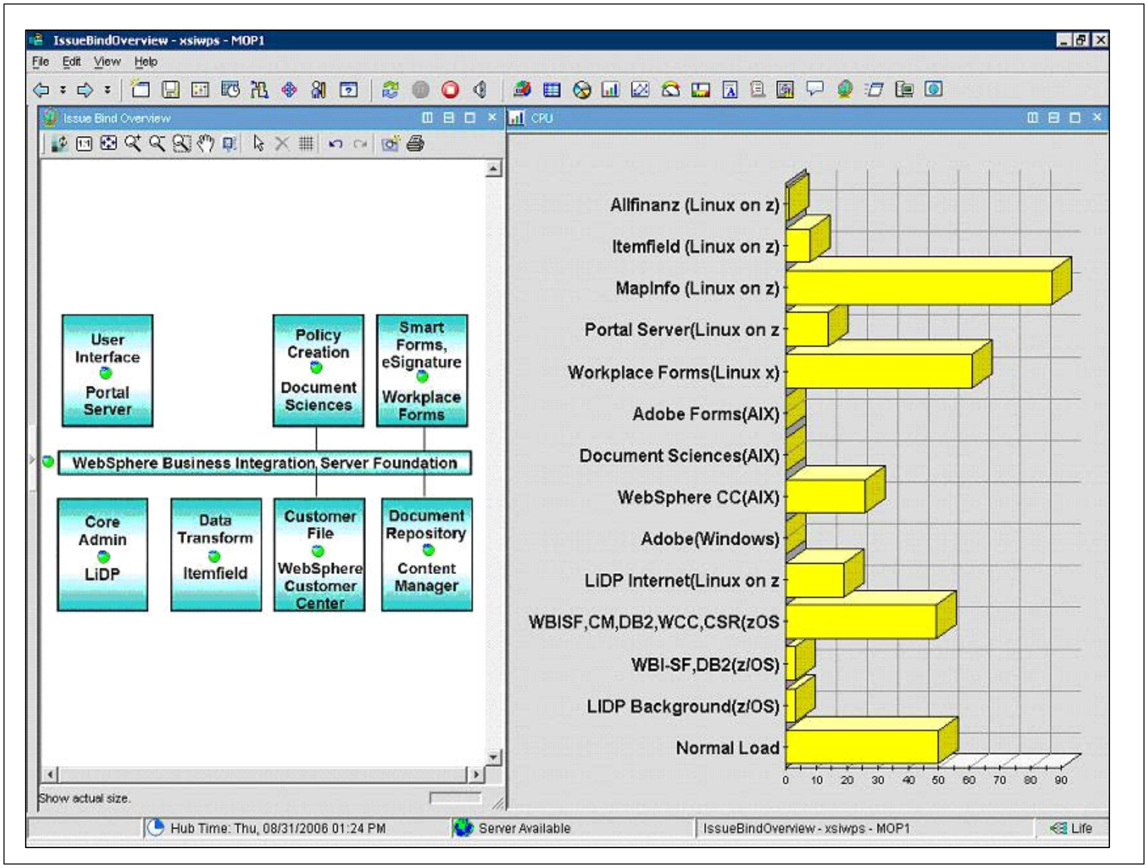


Figure 9-13 Issue/Bind display for problem drill down

Tivoli Enterprise monitor provides the drill down feature for performance problem isolation.

Figure 9-14 on page 352 shows a different view of business process volumes. Instead of number of interview per hour, this screen provides a cumulative view of business volume throughout the day.

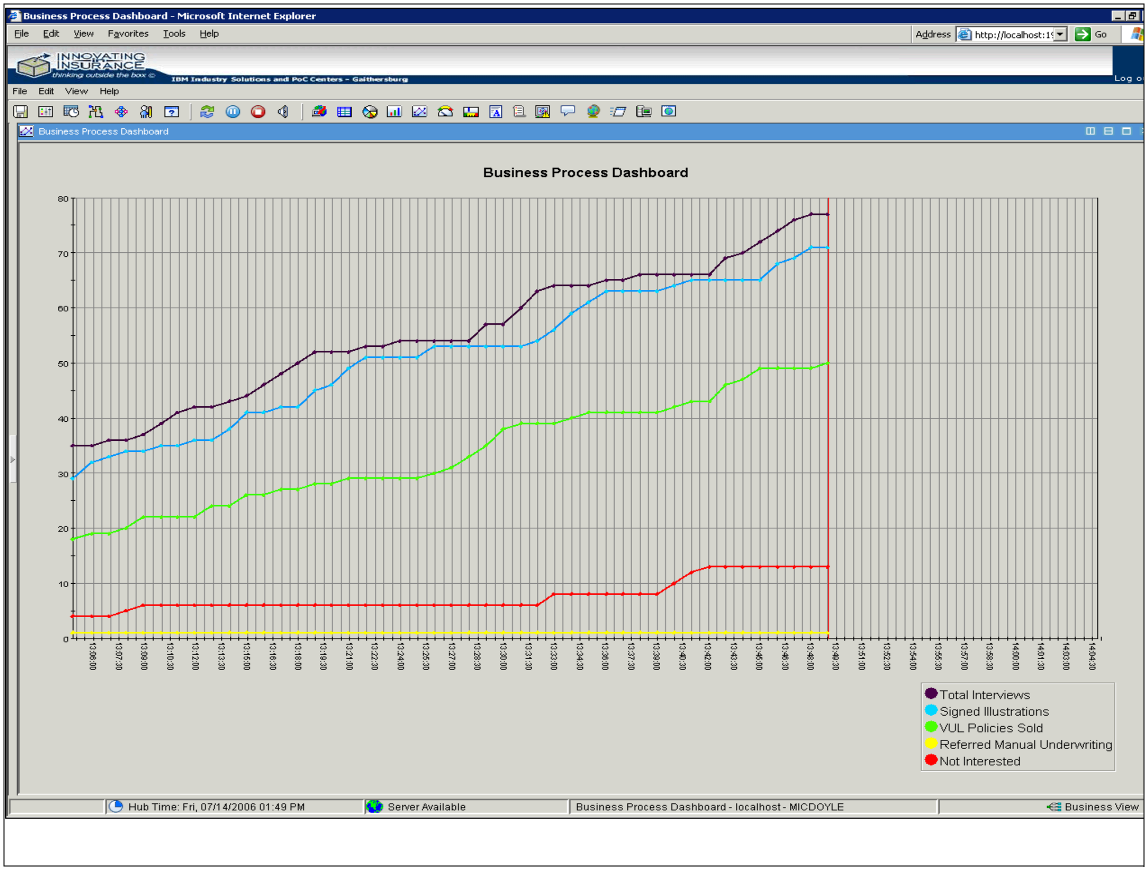


Figure 9-14 Business Metric Monitoring

9.1.8 Conclusion

The main purpose of the IBM Life Insurance Solution Showcase has been to provide a platform for demonstrations that prove the viability of SOA applications on System z centric infrastructures. A secondary purpose of the development of the application and development of the infrastructure was to confirm specific proof points for scalability and resilience.

Scalability

The project clearly proved scalability. The target rate of 500 sales interviews per business day was easily met . During the full day briefings, the team is delivering a rate of 2000 sales interview per business day and there does not appear to be any obvious bottleneck to continuing that scalability with the current infrastructure other than a threading issue with one of the vender software packages.

Resilience

Resilience was proven with both planned and unplanned outage scenarios. One scenario involved gracefully quiescing one of the two WBI-SF servers to apply some maintenance and then bring it back up. During the outage, the workload of scores of agents (simulated using a load driver) and one live agent continue throughout the scenario. The live agent's work is projected for the clients in the briefing while the planned outage occurs; we have not encountered any problems or response time slowing to date.

For the unplanned outage scenario, one of the two WSI-SF address spaces is cancelled using a z/OS operator "C" command. Although a number of error messages are seen, the robot agents and the live agent continue through this error without losing work, just as in the planned outage scenario. In one situation, the agent was missing some data in a portlet, but retried the step and got through it without problems.

For additional information

The IBM Life Insurance Solution Showcase Web site can be found at:

```
http://www.ibm.com/systems/z/solutions/showcase/
```

and:

```
http://www.ibm.com/industries/financialservices/doc/content/landingd
tw/1357314103.html
```

9.2 Case study 2: A bank deployed a large-scale SOA solution based on Web services

This case study is based on an IBM engagement. It is an SOA solution based on Web services developed at a major bank. We identify the business scenario, design objective, architecture decisions, design overview and key challenges encountered.

9.2.1 Business driver

This bank has over 1200 branches, with on average, over 5500 tellers working daily. The teller platform used to be a DOS application which quickly becomes obsolete. Also the existing cheque processing involves a lot of manual intervention. By reengineering the teller platform using SOA technology, the bank expects to save over $20M annually.

Apart from the cost saving, the project has some non-quantified benefit, such as modernization of platform, positioning for code reuse, enhancing business analysis as there will be consolidated transaction records of what the customer and teller does.

9.2.2 Design objectives

- Design a teller services platform to meet the cpu cost and response time objective.
- Develop services that can be reused by other Lines Of Business.
- Design for data center independence
  - prepare for the possibility of data centers topology change due to merging or accruing a subsidary
- Accommodate changing business needs with minimal impact to the design.
- Build a layered application architecture.
- Position to consolidate business logic on the J2EE platform.
- Design for “branch server” independence
  - cost saving by removing the server in every branch

9.2.3 A typical business flow

Figure 9-15 on page 355 depicts an example of a business flow associated to teller transactions.

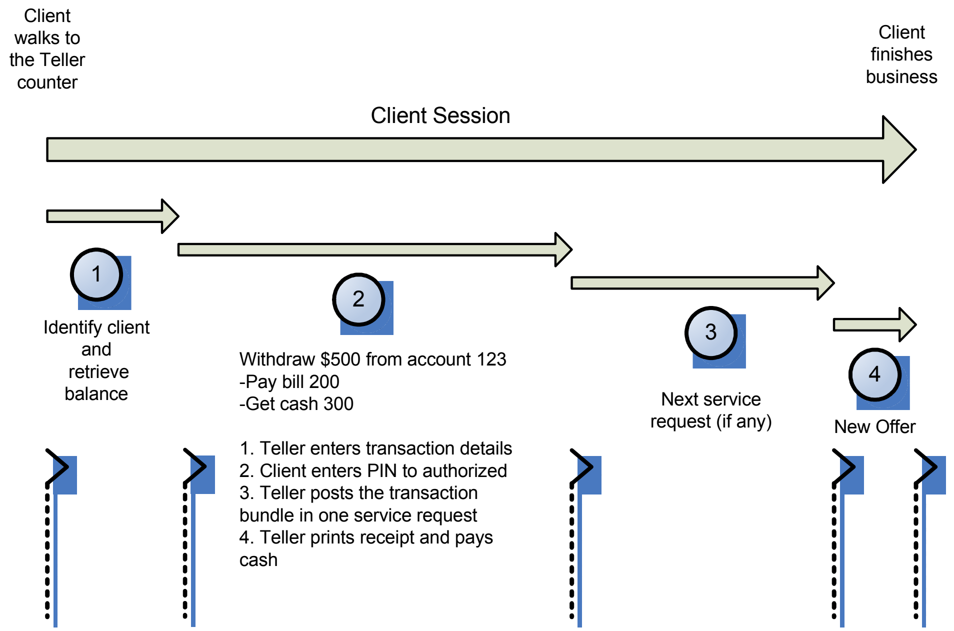


Figure 9-15 Typical business flow

Here is the typical interaction between a teller and the customer:

1. A customer walks up to the teller counter. The teller identifies him using the bank card or other form of identity document, then retrieves the information about this customer including his account balance.
2. The customer is planning to perform two transactions: withdraw $500 from his checking account and make a payment of $200 towards his credit card. The teller enters this sequence of transactions via the application user interface running on his workstation. This bundle of transactions is sent to the data center in **one** service request. Now, the following is done by the application:
   - decompose the request
   - perform authorization
   - fulfill the withdraw
   - handle the payment
   - update the application journal
   - send **one** reply back to the teller

   At the teller counter the receipts are printed and the cash is given out.

3. If the customer has another service request, the process will be similar to the previous request.
4. The teller has accessed the CRM application, and may use this opportunity to offer new banking services to the customer e.g. ask if the customer is interested in a new credit card.

The duration of the client session is an important indicator of how productive the teller is. The bank will want to have the duration as short as optimally possible with the help of technology.

9.2.4 Architectural decisions

The following architectural decsions have been taken before commencing.

Service oriented design and analysis

The Service oriented design and analysis can be grouped into three major steps: identification, specification and realization decisions, as shown in Figure 9-16 on page 357. Usually, these steps are carried out iteratively until a satisfactory decision can be made.

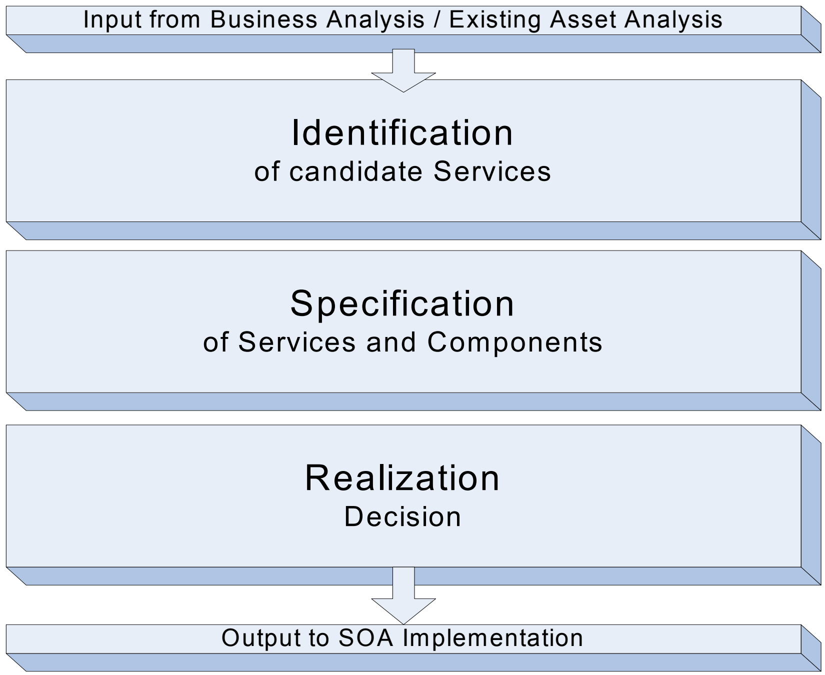


Figure 9-16 Service oriented design and analysis

Identification

The objective of identification is to identify candidate services that are worth implementation. The list of candidates comes from the result of business requirement analysis (top down approach) or from existing asset analysis (bottom up approach).

Specification

For each candidate service, the following needs to be defined: dependencies, composition, exposure decisions, quality of service constaints. Also, decision regarding the management of state within a servicehave to be taken. Then, the services components and relationships to realize the services are defined.

Realization decision

This phase determines in which architectural layer the services components are placed along with rules to define how the layers interact with each other. Then, the mapping of logical runtime patterns to technical infrastructure is made, after an evaluation of technical contraints and Quality of Service.

Logical design of the teller service platform

Figure 9-17 shows the logical architecture for the solution. It is based on a “Adapter-provided Service interface” pattern, as discussed in “Adapter-provided service interface (A)” on page 120.

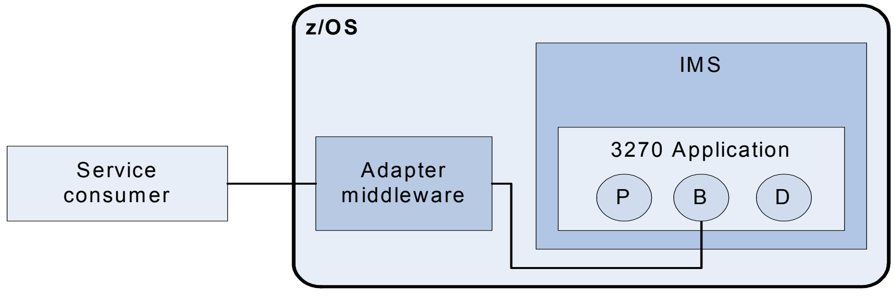


Figure 9-17 Logical runtime pattern of the new teller platform

Architecture of the services consumer

The service consumer is the user Interface Interface (UI) running in a Window workstation used by the teller. This workstation connects to other devices such as a passbook printer, bank card reader and cash dispenser via existing driver programs. Apart from the UI, this workstation also hosts other VB.NET business applications. For the implementation of the teller application UI, the following options were considered:

- presentation layer implementation

  The choices were:

  – J2EE application client

  – VB.NET rich client

  – Browser

  The decision has been taken to use a VB.NET rich client.

  Some remote branches of the bank have relatively low bandwidth connection to the data center. A typical business transaction may require several “interactions” from the teller. To keep the duration of the client session short, the communication with the data center will be kept to the minimum. With a rich client, there will be more flexibility on the UI, e.g. it can be programmed to bundle multiple transactions in one service request and submit to the data center. It can also perform preliminary data validation without involving the host. Although a browser solution has the advantage of being easier to maintain and cost less, it was ruled out as the delay imposed by the network would be unacceptable.

VB.NET was also selected as the customer has significant investment in skill and reusable asset in this area.

- Data Sharing and formatting with existing VB.NET applications in the workstation

  The choices were:

  – Use existing framework for VB application
  – Develop new framework
  – Other products

  The following decision has been taken:

  – Use an existing framework for easier integration with other .NET applications.

Architecture of the services provider

Figure 9-18 on page 360 shows the architecture on the service provider platform.

J2EE is the strategic platform for the business logic and “backend” core applications are hosted in IMS and DB2 in z/OS. The following options are considered:

- The protocol between the service consumer and provider

  The choices were:

  – SOAP over HTTPS
  – SOAP over MQ

  The decision takes was:

  – SOAP over HTTPS

  While MQ has the advantage of guaranteed delivery, it requires an MQ queue manager at the branch. This contradicts one design objective of this project, namely, to remove the branch servers for cost saving. Therefore, SOAP over MQ was ruled out. As HTTP is inherently not a very reliable protocol, there is application logic in both the service consumer and provider layers to handle the errors of lost messages.

- In the service provider layer, the services exposed to the consumer are coarse-grained and there is application logic running in the WebSphere Application Server to encapsulate fine grain transactions provided by the backend. The connection to the backend is via J2C connectors. Our business scenarios require both synchronous and asynchronous connections to IMS. For the synchronous one, we use IMS Connect. For the asynchronous transactions, MQ is used.

- To connect to DB2, the Type 2 JDBC driver is used for performance.

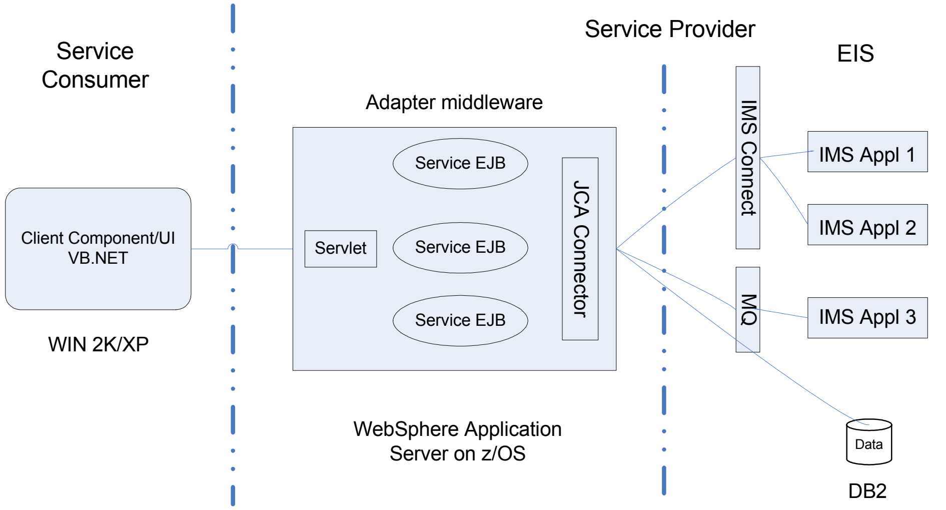


Figure 9-18 Product mapping of teller platform

9.2.5 Solution overview

In the following sections we will describe the solution overview, both from a client (teller) perspective and a service provider perspective.

Teller application layers

Figure 9-19 on page 361 shows the solution for the teller application.

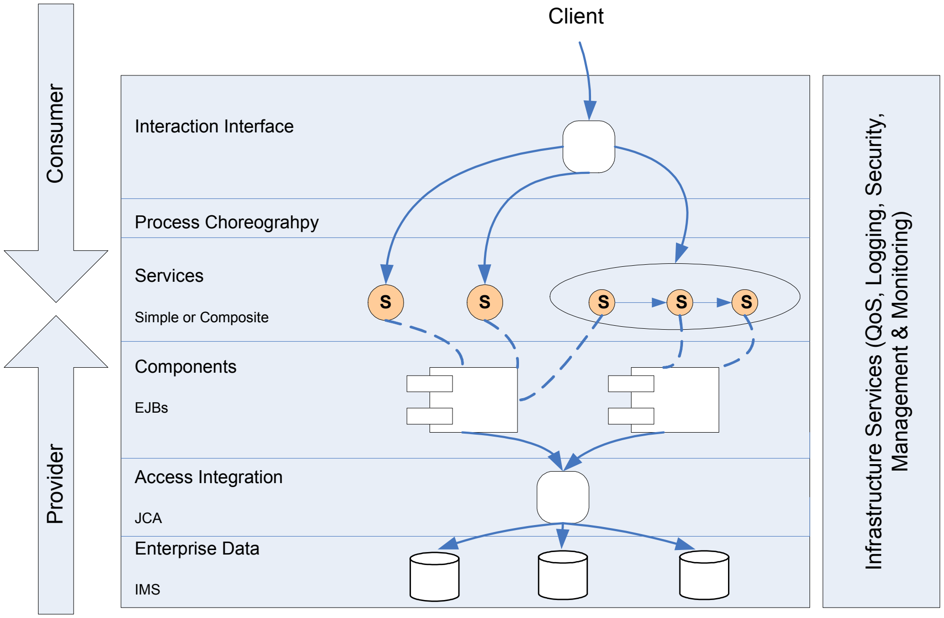


Figure 9-19 Application layers in the teller application

Interaction Interface layer

This layer routes the request to the appropriate service while shielding the services from the message format and transport protocol. This facilitate future implementation of alternative protocols such as SOAP over MQ.

Process Choreography layer

The current release of the application does not exploit this layer

Services Layer

This layer provides the abstract service interfaces to the consumer and separates the implementation logic. It also encapsulate similar business functions supported by different Enterprise Information System (EIS) into one service. For example, “withdraw from checking”, “saving” or “loan accounts” can be handled by one common service. The services can be “atomic” or ‘composite’ which invokes other services.

Component layer

The layer implements the logic of the service. It containers stateless session EJBs which connect to the IMS via the Connection Integration layer.

Access Integration layer

This layer decouples the IMS from the EJBs. It is a business requirement for the EJBs to connect to one of the IMS in three different centers, via IMS Connector for JAVA or JMS. This layer will make the decision of which protocol to use and where the request should send to based on the rules in a configuration file.

9.2.6 Key challenges in the Services Provider layer

In the following sections we discuss some of the main key issues that needed to be resolved at the service provider side of the solution.

Lost messages

The connection between service consumer and provider is via http(s) which is inherently unreliable. Lost message are not acceptable in a financial institution. Here are the scenarios for a lost message"

- Lost incoming service request message to the provider.
  - There is no impact, the consumer can resend the request.
- Lost outgoing response message from the service provider.
  - The transactions are executed, but the service consumer is not notified of the result and may resend the same request again. To protect against the re-driving of the same transactions, a strategy is needed to uniquely identify each service request bundle, track the result of the individual transaction within the bundle, and reconstruct the response messages if necessary.
  - This is implemented by an application journal in DB2, it is known as "intent table". Each service request bundle comes in with a unique identifier generated from the rich VB.NET client. The whole request is first stored in the journal with this identifier as the key. As each transaction in the bundle is processed, the outcome is stored in the journal. If a service request end up with a duplicate key exception when it is stored in the journal, then the request has been handled previously. The response messages can be rebuild based on the unique request identifier and the transaction records in the journal.

Transaction management

The customer has a policy of not implementing two-phase commit support in IMS, so the transactions run in "commit mode one, sync mode zero", meaning IMS commits the transaction when it receives it. If no further response is heard from IMS, the caller cannot tell whether the transaction is successful or not. In our case where a service request may be composed of multiple IMS transactions, a strategy is needed to handle the situation when no response is received from IMS.

The solution is to use the application journal which records the whole request and the outcome of each transaction in the bundle. If there is any “in doubt” situation, there is a business process of agent assisted reconciliation of those transactions.

Multiple processing centers

The bank has three data centers to serve customers in three geographic locations. The data is not consolidated i.e. the account information in the IMS system of data center “1” is different from that of data center “2”. Suppose a customer has an account in a city served by data center 1 and he is on travel to the city served by data center 2. If he would go to a branch to withdraw some cash from his account, the teller should be able to complete the business transaction regardless which city this customer comes from.

The solution is to use a router pattern in the service integration adapter layer where the routing rules are externalized in the configuration. In other words, when the account number of this customer indicates his information is stored in data center 1, the application in data center 2 should connect to the IMS in data center 1 and complete the service.

Should there be a requirement to consolidate the data centers, the routing rules can be changed easily to reflect that.

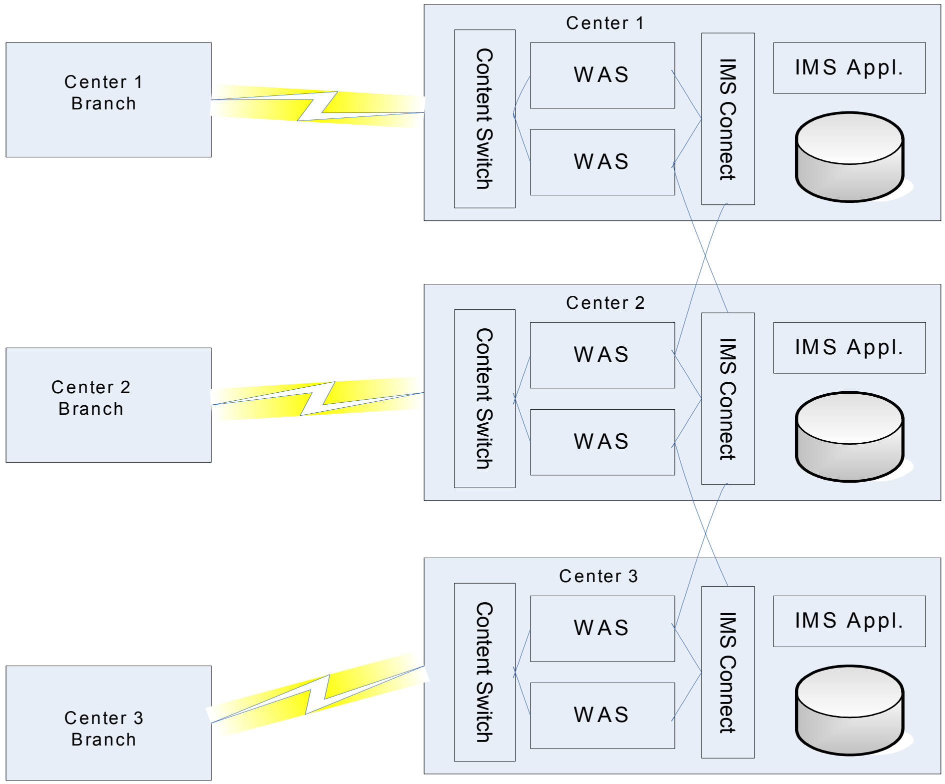


Figure 9-20 Design for IMS location independence

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 366. Note that some of the documents referenced here may be available in softcopy only.

- *????full title???????,* xxxx-xxxx
- *????full title???????*, SG24-xxxx
- *????full title???????*, REDP-xxxx
- *????full title???????*, TIPS-xxxx

Other publications

These publications are also relevant as further information sources:

- *????full title???????,* xxxx-xxxx
- *????full title???????,* xxxx-xxxx
- *????full title???????,* xxxx-xxxx

Online resources

These Web sites are also relevant as further information sources:

- Description1

  `http://????????.???.???/`
- Description2

  `http://????????.???.???/`
- Description3

  `http://????????.???.???/`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

E

F

G

H

I

X

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5” spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM Redbooks

SOA Architcture Handbook for z/OS

(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

IBM Redbooks

SOA Architcture Handbook for z/OS

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Redbooks

SOA Architcture Handbook for z/OS

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

IBM Redbooks

SOA Architcture Handbook for z/OS

(0.2”spine)
0.17”<->0.473”
90<->249 pages

(0.1”spine)
0.1”<->0.169”
53<->89 pages

IBM

Redbooks

SOA Architcture Handbook for z/OS

(2.5” spine)
2.5”<->nnn.n”
1315<-> nnnn pages

IBM

Redbooks

SOA Architcture Handbook for z/OS

(2.0” spine)
2.0” <-> 2.498”
1052 <-> 1314 pages

Draft Document for Review January 29, 2007 3:05 pm

SOA Architcture Handbook for z/OS

IBM

Redbooks

How to get to an SOA on z/OS

IBM software solutions on z/OS

Quality of Service and governance of SOA on z/OS

Service Oriented Architecture (SOA) is a hot topic on the agenda of many CIOs, architects and IT professionals. Most of us believe that SOA will truly make IT more flexible and that time to market of IT solutions will improve significantly.

However, in many cases the bright and shining landscape of an SOA with all its advantages is almost the opposite of today's reality in many IT environments. IT is too complex, too tightly integrated and therefore NOT flexible and the time to market of IT changes as a result of changing business requirements is not acceptable. Furthermore, technology is not in-place, and even more importantly, the organization is many times not ready for an SOA.

Even though most of us believe in the advantages of an SOA, it remains a big challenge to execute towards a full-blown implementation. In any case, when it comes to implementation of an SOA, it will be done in steps. This Redbook is about the possible approaches towards an SOA on z/OS and the technology options available. The book will help you to define an SOA strategy on z/OS and decide on technology to be used in this journey.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7331-00 ISBN