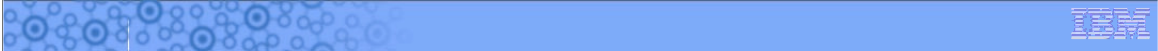




A High-Level Introduction to Web Services, SOA and ESB (And How System z Plays)

Don Bagwell
IBM Washington Systems Center
dbagwell1@us.ibm.com

© 2007 IBM Corporation



This slide intentionally left blank

A Very Large Topic Space

If you read much of the IBM literature on this subject, you'll find there are really three overlapping things being referenced ... sometimes all at once:

• Realm of consultants and CEOs

Business re-engineering and business process management

Development process life cycle concepts, management and tools

Functional Details of Architecture and Product Implementation

• Model
• Assemble
• Deploy
• Manage
• Governance

This is more the focus of this workshop.

Please understand -- each is important in its own way. All are important to the success of SOA and the value it can provide.

We'll touch on all levels, but focus mainly on the inside box.

Role of understanding concept ...

3 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

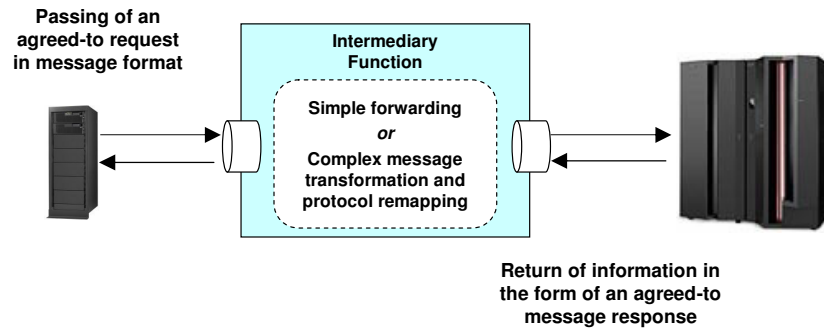
This "SOA" thing is a very broad topic. You'll find when doing other reading on the subject that three overlapping things are frequently discussed at the same time:

- The very broad topic of aligning the business itself around the concept of service orientation. This includes a focus on process management and re-engineering. This is a much higher level thing than we'll focus much on in this workshop. This is where CEOs and business consultants come into play.
- The topic of implementing a disciplined development and deployment lifecycle for your IT resources. This involves business processes and many different tools that aid in creating, monitoring and enforcing the lifecycle. The "four hexagon" picture in the middle of this chart is something you'll frequently see when reading IBM literature on the subject.
- Finally, we get to the lower level of this -- the functional details and the product implementation details offered by IBM. This is going to be the focus of this workshop. The reason why we'll focus on that is because the audience that attends Wildfire workshops traditionally is and has been people more interested in this level.

It is important to realize that by focusing on the inside box we are **not** saying the other boxes are unimportant. They are *very* important in their own ways. But the time limits of this workshop and the objective of focusing on the technical specifics mean we can't cover those areas that much.

At a Very High Level

What we're ultimately getting at here is a decoupling of application requester from the application provider. And the placement of an intermediary function to make things more flexible and dynamic:



But that's too high-level to do us much good. That's what this workshop is for: to "make real" this very high-level picture.

Objectives of this presentation ...

4

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

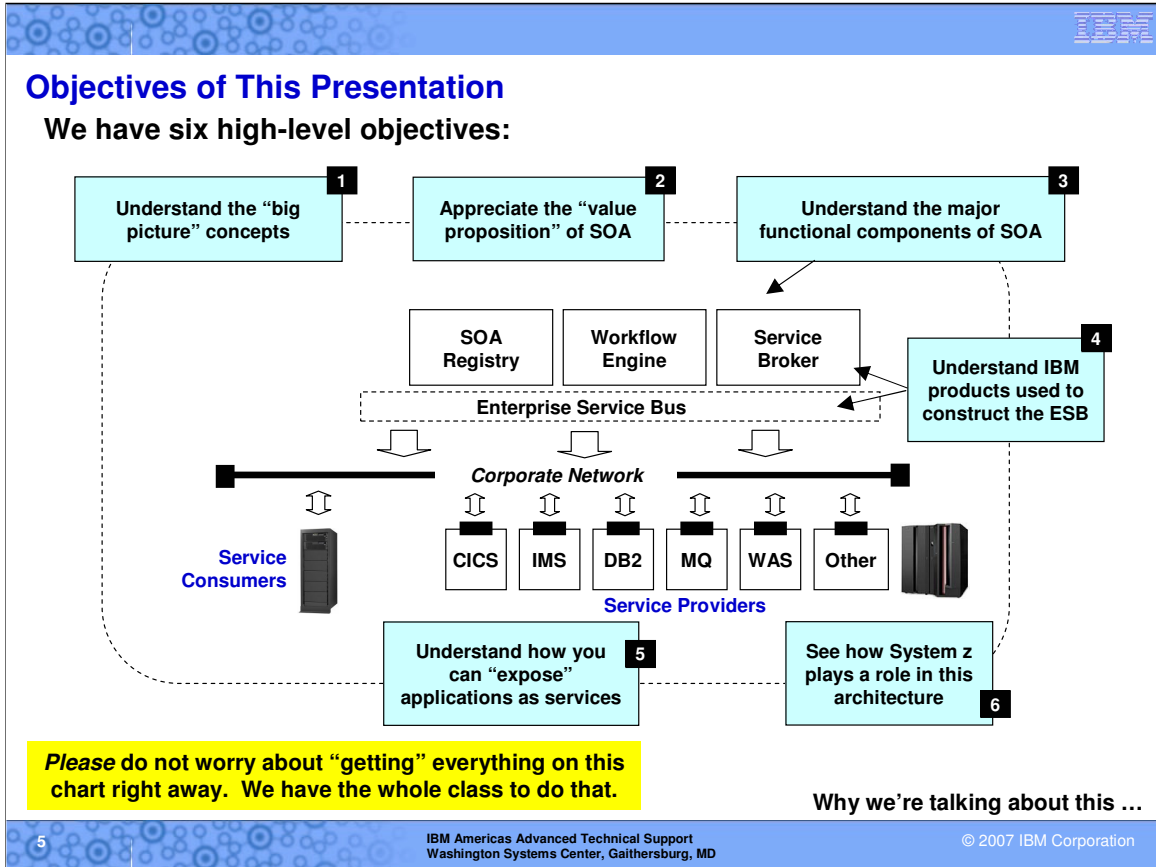
To start out, let's offer a very high-level view of the essential concept underlying what we'll look at. It's the act of uncoupling an application requester from the provider. The reason for this, as we'll see in a bit, is because when the relationship is tightly coupled, overall application architectures become very complex and very inflexible. The "loose coupling" we will speak of can be thought of *like* a remote method invocation, or a remote procedure call.

Note: don't take that analogy too literally. It's a useful analogy when kept at the concept level.

The second piece of this is the placement of a function between requester and provider that acts as an intermediary. The intent of this intermediary function is to provide a point of flexibility, as well as a place where more complex message handling can take place if that's needed. (It won't always be needed.)

So the requester sends its request, which passes through the intermediary function towards the provider. The provider receives the request, handles it, and returns the requested information. The intermediary function returns the answer to the requester.

If we stopped there you'd be left unsatisfied. It's too high-level. Too abstract. The purpose of this workshop is to make that essential picture more "real" by mapping actual product implementation and architecture examples to this picture.



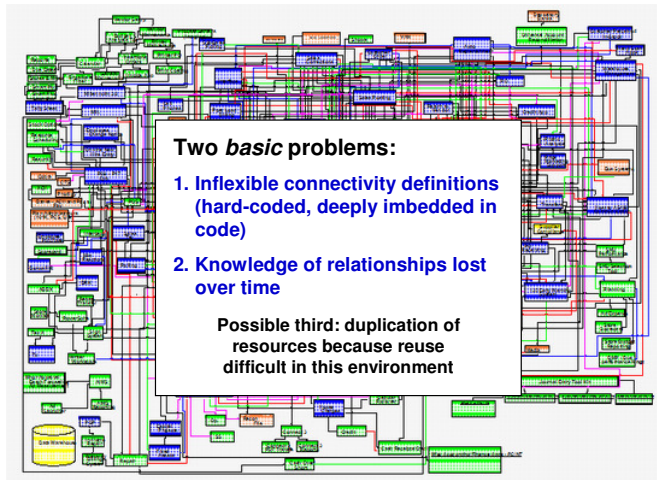
This picture will be used as a prop to introduce the six main objects we have for this first presentation:

1. First, to understand the big picture and its related concepts. The picture on this chart illustrates that high-level big picture. At the end of this presentation you should be able to see what this picture is trying to convey.
2. Understand the value of “Service Oriented Architecture” and have some level of appreciation for that value.
3. Understand what some of the major functional components of SOA are. This gets a little tricky because different people have different things listed as the “major components” of SOA. For this presentation we’ll try to stick to a fairly small list of them.
4. See what IBM products implement the major functional components just outlined in bullet number 3.
5. Understand how you can take an existing system and “expose” it as a service. This is an important piece of the puzzle because it gets to how you can start incrementally, and how SOA does **not** need to be a “rip and replace” strategy.
6. Finally, to see how and why System z can play in this picture. What you’ll see is that there’s no reason System z can’t play, and in fact System z brings to the table the same inherent strengths System z has in general.

It’s important at this point that you **not** try to understand every element of this picture. It’s just a starting point to discuss the objectives we have for *this* presentation.

What's Behind This ... Why Are We Talking About This

Over the years our application systems have become very complicated, with tightly-coupled relationships that are often little understood.



← Actual application architecture map from real-life customer

Changes to any part of this are ...

- Difficult to determine what impact there is on other components
- Expensive to analyze, often expensive to implement
- Delays often result in missed opportunities

Two objectives: eliminate tight (hard-coded) interconnections, and create a way for one program to dynamically seek, find, and bind to another.

Let's introduce the notion of a "Service" ...

6

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

We'll start by pointing out what's behind all this talk about "Services" and "Service Oriented Architecture." And it has to do with the way our applications have become interconnected over the years. Nobody started out intending things to end up like this; it's just how things evolved over the years.

Note: the picture above -- which looks like a blur of color and lines -- is a miniaturized version of an actual application architecture map from a real customer. And that's just *one page* of many. We're not making this stuff up ... the complexity out there is very real.

What's the problem with something like this?

- If it works, and you never consider any changes to it, it's fine. But it doesn't always work, and there's never a day that someone isn't asking for something changed or added.
- When changes are considered, it becomes a challenge to determine what the impact of the proposed change will be. If the proposal is to change the input requirements for application XYZ, then it's important to understand what other applications use XYZ, and what impact a change to XYZ will have on them.
- It becomes very expensive to do the analysis trying to figure out the impact, and it's often very expensive to implement the change because it ends up touching other things.
- So what happens is there's a time lag introduced between the awareness of the need for a change and the actual implementation. During that time lag opportunities for sales, or cost reduction, or increased productivity are lost.

Fundamentally, the problem is two-fold: the applications interconnection definitions are often define in fixed and deeply imbedded places; and over time we've lost track of what and how things connect to each other. Therefore, our broad objective is to overcome that.

A “Service”

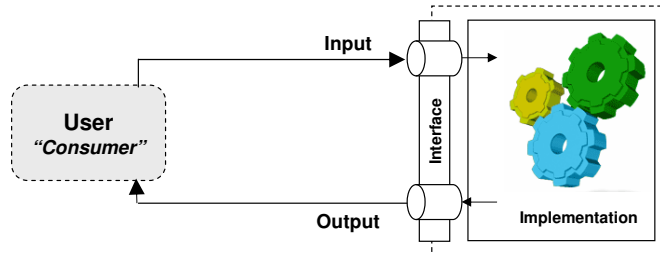
A **discrete** set of business or technical functionality that can be **identified**, has a **defined** set of input and output, and is **reusable**

Discrete – can be contained within a definite and known “fence”

Identified -- it’s recognized as a service and people acknowledge it as a service

Defined – the input and the outputs are known and understood

Reusable – is not just a one-time thing



Service
“Producer”

Exactly how the service is implemented behind the interface doesn't really matter to the consumer of the service

There's nothing revolutionary about this. What's different is that we're coming to a point where improvements in technology have allowed us to do this better than before:

- Settled on a universal and common networking protocol -- TCP/IP
- Networking bandwidth is increasingly available, cheap and reliable
- The idea of “industry standards” has matured and is embraced rather than resisted
- Java as a platform-unaware language has opened up a new world of interoperability

An example of a “Service” ...

7

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

So we get to the point where we can introduce the concept of a “Service.” We’ll use this for the rest of this workshop. The top of the chart offers the definition, with further explanation about what some of the keywords are relating to. For now, let’s focus on the picture.

A “Service” is a something that someone else can use ... or in computer language, “invoke.” To use or invoke a service, the requester (or service “consumer”) needs only to know that the service exists, where it’s located, and what’s required to invoke it and what can be expected in return. In this sense the service has a kind of standardized, defined set of input and output requirements.

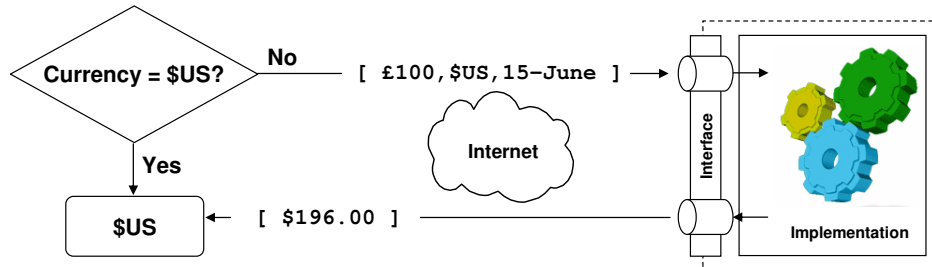
Note: a service does not need to be a computer program. It can be a human behind a desk who does something on your behalf. But for the purposes of this workshop, we’re focusing on a service being a computer-implemented thing.

The key to this is that the actual implementation of the service -- what goes on behind the interface, in other words -- is something the service consume doesn’t care about. I don’t care how the Post Office gets my letter from here to there ... all I know is that I have to put an address and ZIP code on the envelope and turn it over to them.

There’s absolutely nothing revolutionary about this. We’ve been striving to achieve something like this in the computer industry for decades. (Subroutines are a kind of “service,” the whole client/server thing had elements of this in mind, distributed objects and remote procedure calls also share this basic notion.) What’s different is the times in which we live -- things have come to be that facilitate SOA much better -- a common networking protocol (TCP/IP ... you can’t begin to imagine how important it is that debate is over); cheap and readily available bandwidth; the coming of age of industry standards ... not only the standards but the idea of embracing standards rather than seeing them as a threat; and finally Java as a kind of common runtime language. All have made the adoption of SOA much easier. Let’s now look at an example of a service.

An Example -- Currency Exchange

IBM's Travel Expense Reimbursement application does not do its own foreign currency conversions ... it uses an external service for that:



Could IBM have coded an internal subroutine to do currency conversions? Sure. But very good converters exist on the web and in this case IBM took advantage of them.

For this to work, several things need to be in place:

- IBM application needs to know about the service and where it is located
- IBM application needs to know the interface requirements: parameters, sequence, format

Understanding what services are available, where they're located and what interface requirements they have is a key aspect of SOA. More coming.

Another Example of a "Service" ...

IBM has a travel expense application used to file for reimbursements. When a foreign currency is involved, that application needs to know the exchange rate so it can convert the foreign currency to \$US. Originally that exchange was left up to the person filling out the reimbursement report -- we often took a guess, relying on some memory of what exchange we got at the airport currency exchange booth.

A few years ago the designers of the TEA application realized that there are websites out on the Internet that provide real-time foreign currency exchange rates. Rather than assuming the guessed-at exchange rate is correct, the application designers decided to go out to the web and get the actual exchange rate to validate the value provided by the employee. So the expense reimbursement application was updated to send a URL to some website, fetch the actual exchange, and save it in the expense report's information.

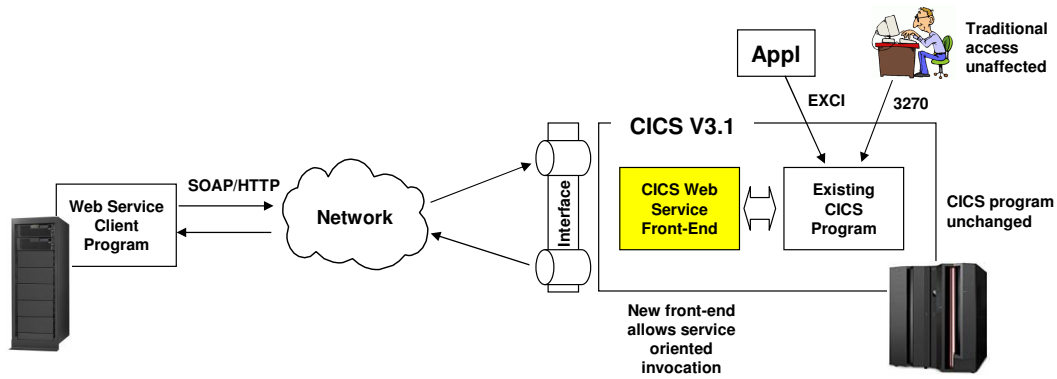
This couldn't have occurred by magic. For this to work, several things needed to be in place:

- The expense application developer needed to know that a foreign currency exchange website existed and where it was located.
- The developer then needed to know what that website required as input parameters. For example, a currency exchange application is going to need, at a minimum, the "from" currency and the "to" currency, the amount of the "from" currency and probably a date as well because currency exchange rates fluctuate all the time. Further, the developer needed to know what to expect back.

This idea of knowing about a service, its location and its interface requirements is a key element of SOA. We'll see a lot more of that when we get to discussions of Web Services.

Another Example -- CICS Web Service

We've not really defined what a "web service" is ... but for now be aware that CICS has the ability to front-end existing CICS programs with a web service interface ... "exposing" the CICS program as a service:



Key point is that traditional CICS program can be turned into a message based "service" which can then be used by "service consumers" in your network

Both a technical and business concept ...

Here's another example of a "service" ... this time it's an existing CICS application that's been exposed as a "Web Service." We'll cover what exactly a "Web Service" is in more detail in a bit ... for now, just understand that it's a service with an industry standard format for describing the interface requirements and the exchange of data.

So, imagine an existing CICS application that you use today. Perhaps it's being used by other applications using the External Call Interface (EXCI) of CICS, or perhaps it's being used by people at traditional 3270 terminals. But now you want to make it available to a "service consumer" as a standard "Web Service." What to do?

In CICS V3.1 there is a support for developing a Web Service "front-end" to such applications. This "front end" is what implements the interface -- the input and its requirements; the output and what it produces -- and then turns and works against the existing CICS application. A Web Service client out on the network somewhere needs only know that this new "Service" is available, it's location, and how to invoke it. The Web Service client definitely does **not** need to format a COMMAREA. Nor does the Web Service client need to know anything about the real CICS application behind the Web Service interface.


There are several key points here:

- The existing CICS application remains unchanged. This is another piece of code out front of it that implements the Web Service *interface*.
- Existing users of the CICS application continue to use it as they have all along.
- The Web Services client may now start using the CICS application as a "service"
- Any new needs to use that CICS application may now start using it as a service as well

And thus we've created a discrete, reusable *service* with a defined set of input and output. And we've done that without disrupting existing users. And we've laid the groundwork for new things to use the service without having to "hardwire" them into the existing CICS application.

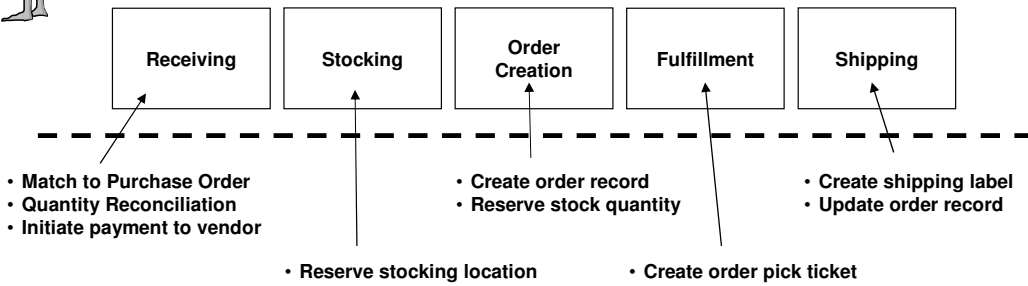
Two Perspectives of the Same Thing

Depending on who you are and how you approach this, the concept of a “Service” takes on different meanings




Business manager or business consultant
View business process as a set of functional services linked in a specified flow

This is where discussions of process re-engineering and business alignment to services orientation comes from. More business consulting than I/T architecture



```

graph LR
    subgraph Business_View [Business manager or business consultant]
        R[Receiving] --> S[Stocking] --> OC[Order Creation] --> F[Fulfillment] --> SH[Shipping]
    end
    subgraph IT_View [IT specialist or architect]
        R1[Match to Purchase Order  
Quantity Reconciliation  
Initiate payment to vendor] --> S1[Reserve stocking location] --> OC1[Create order record  
Reserve stock quantity] --> F1[Create order pick ticket] --> SH1[Create shipping label  
Update order record]
    end
    R --- R1
    S --- S1
    OC --- OC1
    F --- F1
    SH --- SH1
  
```



IT specialist or architect
View as a set of computing actions – programs, subroutines, transactions, etc.

Both important! This is why you often see discussions that cross over from technology into business consulting language

Service Oriented Architecture ...

10
IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD
© 2007 IBM Corporation

At this point we need to pause and point out that “Service Oriented Architecture” means different things to different people. It depends on what perspective one takes. To the IT specialist or architect, they’re going to view “services” as IT-related components. So to them the hypothetical shipping and receiving application flow is represented by a series of IT components -- a service to match a received shipment to the purchase order; another service to make sure the quantity received is equal to the quantity ordered; another service to reserve the stocking location in the warehouse, etc.

But to the business manager or consultant, they’ll be viewing it not necessarily as computer-related application functions, but discrete functional components of the business.

Both views or perspectives are important, and they serve different purposes. Both will be part of any SOA discussion, depending on who you’re talking to. We bring this up because this is why you’ll see many write-ups on SOA switch between pure computer talk to business-speak, and back again.

Now we can try to define what “Service Oriented Architecture” means.

Service Oriented Architecture

From www.ibm.com:

Service oriented architecture (SOA) is a business-driven IT architectural approach that supports integrating the business as linked, repeatable business tasks, or services.

Personally, I don't think the exact definition is all that important. More important:

- You understand the concept of a "service"
- You understand the implied value of a loosely coupled "service" rather than a tightly coupled connection to another application's interface ... flexibility
- You understand that "SOA" is a *path* towards the use of more and more services in your I/T architecture ... not a "thing" or an "all-at-once" proposition
- You understand that there's more to it than *just* services. We have yet to introduce the Enterprise Service Bus and the function within it. Much more to come.

Approaching this incrementally ...

11

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

This definition comes from the www.ibm.com website. It's as good a definition as any. And there are a lot of definitions floating around out there.

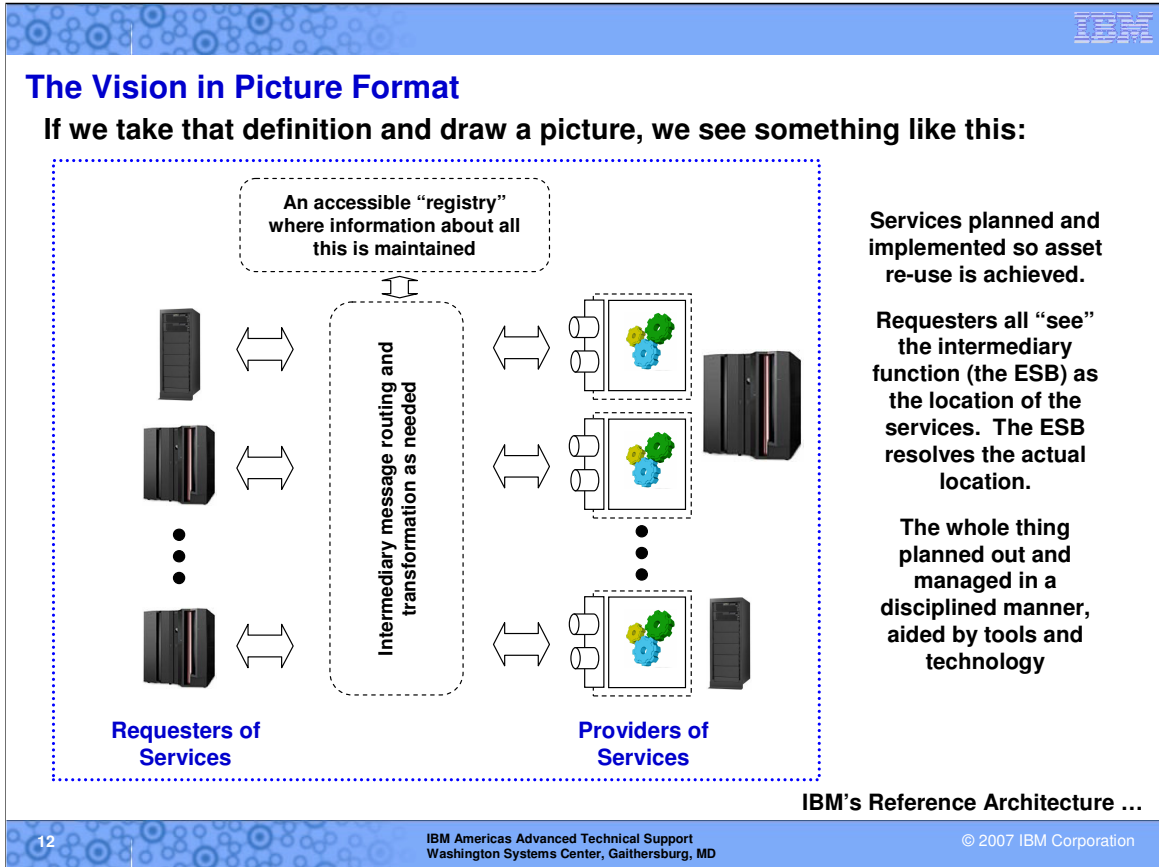
Author comments:

It's easy to get too caught up in the wording of such definitions. Having a definition is important to put a "stake in the ground," but no definition is perfect and to the extent trying to make sense of a definition causes your understanding of SOA to be hindered, it's best not to rely too much upon any one definition.

The key, I think, is this:

- First and foremost, you have a handle on what's meant by a "service." A service is a discrete function with a defined set of input requirements and a defined set of output results.
- You keep an eye on what's important here -- what a "loosely coupled service architecture" provides is an increase in *flexibility*. Changes to the function behind the interface won't affect users of the service if the interface itself hasn't changed. And changes to the interface itself can be made if the consumers of the service have a programmatic way of understanding the change and taking it into account. (This last part gets to the question of "binding" to the service and understanding its requirements. Web Service clients use something called a "WSDL file" to do this. We'll discuss this in more detail later.)
- You understand that SOA is not really a product, or a big switch that auto-magically transforms your company overnight. It is more than anything a philosophy or mindset; one which is implemented with computer stuff we'll describe here.
- Finally, you understand that SOA is more than just services. We're starting there as the first key concept to grasp. But we have more to bring into the picture ... the ESB, the notion of a service repository, the notion of process coordination (known as "choreography").

If you keep those essential points in mind, you'll be okay as you see different definitions of SOA float by.



Using the definition from the previous page, we can draw a picture that looks something like what's shown here. A set of "service requesters" access their desired services through the intermediary function (which, as you'll see, is the Enterprise Service Bus, or ESB), and the requests flow to the service providers.

The services are designed in such a way that re-use is achieved to a greater degree than before.

Flexibility is achieved because the ESB is acting as a "shield" of the actual service implementation behind it. Changes to a service implementation can be hidden from the population of service users, some of which you may not have control over.

This environment is then managed and controlled in a systematic and logical manner using practices of discipline, aided by technology and tools.

This is the vision ... at a high level.

IBM's picture of this is captured in the IBM Reference Architecture ...

The IBM SOA Reference Architecture with Product Mappings

This is another way to show the architecture and products mappings. This is a more logical view rather than the physical view from previous page.

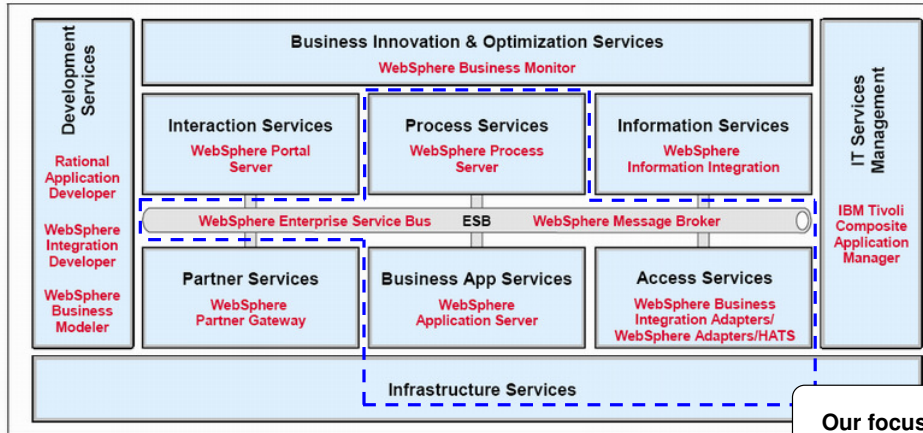


Figure 3-1 IBM SOA Reference Architecture with product mapping



From "Getting Started with WebSphere Enterprise Service Bus V6" Redbook
redbooks.ibm.com
 SG24-7212

Approaching SOA incrementally ...

The picture for "SOA" used in this presentation is a somewhat simplified illustration. The IBM SOA Reference Architecture presents a much more comprehensive view. It's another way to look at this subject.

For this presentation we chose to limit our focus so we could drill down a little bit more into some of the technical details. That's not to say the stuff we don't focus on is unimportant ... it's just that time dictates we limit the scope a bit.

The picture above came from the Redbook SG24-7212, which has an excellent writeup on the whole of SOA. It's a good book to download.

One thing we wish to emphasize ... we are not talking about a "rip and replace" strategy. We recognize the need to approach this incrementally.

Incremental Implementation

SOA does *not* imply ripping up your entire infrastructure and replacing it with something new. It can be done incrementally:

Imagine this is a logical representation of your existing application architecture ...

You've identified two functions you think would make good reusable services

- Check inventory function within application
- Create shipping label function within application

Develop service interfaces for these ... exposing them as services for re-use by other applications

Overly simplified?
Maybe a little ...but the underlying concept is sound. Recall the CICS example from a few charts ago

Existing interfaces unchanged

Building composite applications ...

14

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

A big trap in this SOA thing is thinking it's a massive "rip and replace" operation. It *can* be, if as a company you've decided to re-engineer your business around the concept of reusable services. Some companies have done just that with success. But it doesn't mean all companies must do that. Many are opting for the other approach -- a slow, gradual implementation. And this is done by identifying existing things that would make good "reusable services" and providing a "service interface" to them.

The picture above shows a slightly different representation of the "complex interconnected application architecture blob" from before. Now imagine you've identified two functions you wish to use as a pilot project for service implementation -- the "Check Inventory" function and the "Create Shipping Label" function.

The notion here is that you could develop the "service interface" for these two, without affecting others or the complex structure as a whole. (How exactly one develops a "service interface" is something we'll cover in the next unit for Web Services, and in the ESB unit when we describe WESB and WMB, the two products IBM has that implements the ESB concept.)

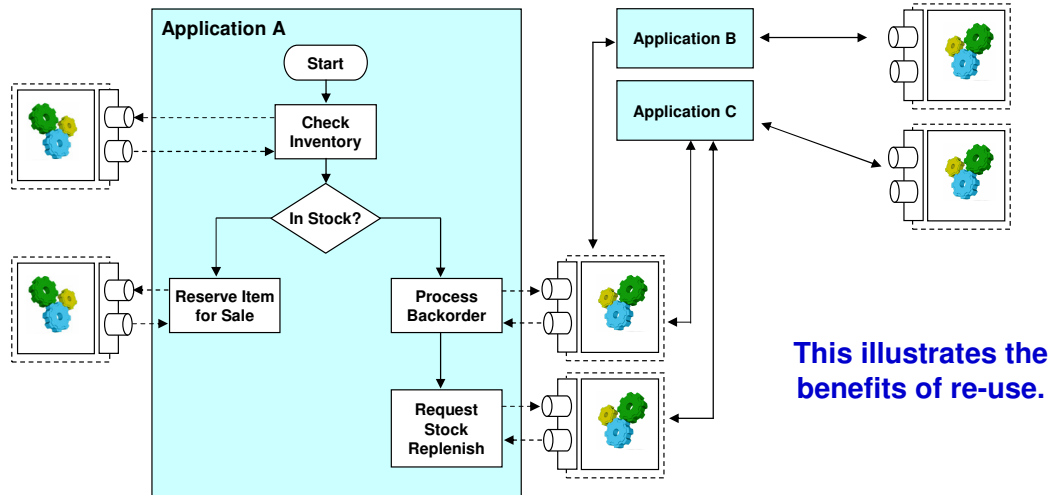
With the service interfaces developed for "Check Inventory" and "Create Shipping Label," now some new application that needs those two functions may simply "invoke the service" to get what it needs. It doesn't need to be "tightly coupled." It is now "loosely coupled."

Overly simple? Sure ... but it helps illustrate the key concept of starting with an identified set of discrete services and exposing them as services without affecting your existing infrastructure.

Let's say you have a bunch of services created. What's the big deal in that? Well, it then allows you to start building "composite applications" -- applications that invoke a string of different services to perform its functions.

Composite Applications Built on a String of Services

Extending the concept ... once a library of reusable services has been built, future applications can be built by stringing services together:



This illustrates the benefits of re-use.

A real-life example ...

Again, imagine you've built up an inventory of reusable services, all with defined interfaces. Now you're getting ready to construct a new application (we'll call it "Application A"). Rather than rewrite the same function over again, or do "tightly coupled" things like link-edit the other function into the application or make sure some `CLASSPATH` is updated, all you need to do is code the new application to go out and invoke the services in the order it needs. The whole application probably won't be just invoking one service after another, but hopefully a large portion of it will be that.


But here's the value -- imagine you've got two other applications -- B and C -- and they also need to use some of the same services as Application A. All those applications would need to do is invoke them. Application A doesn't need to know about B or C at all. Further, if you ever had to go into those two services that all the apps are use -- "Process Backorder" and "Request Stock Replenishment" -- and tweak the internal implementation, applications A, B and C remain unaffected ... provided the interface remains as it was.

This is good ... the connectivity has been decoupled a bit. This helps illustrate the benefits of reuse.

IBM Corporation
© 2007 IBM Corporation
All rights reserved.

Real Life Example of a Composite Service Application

IBM's internal "Online Travel Reservation" invokes a string of external services to locate flight, hotel and car availability:


⇒

Key in dates, destinations, etc.

⇒

Planning your trip...

- Retrieving your corporate policy
- Shopping for the best airfare
- Establishing available flights
- Identifying available hotels
- Identifying available cars
- Packaging your trip

The application then invokes a list of services to determine airfare, which flights have availability, which hotels are available, etc.

Those services are standard industry services; others use them too.



My Trip
Traveler and Expense Information
 Traveler: DONALD BAGWELL
 Corporate Policy
 Tucson, AZ, US to Los Angeles Intl Arpt, CA, US Friday, 3/2/200
 US Airways flight 2941 In Policy
 Alternate flight times available
 Tucson Intl Arpt - Tucson, AZ, US
 08:15 AM departure
 Sky Harbor Intl Arpt - Phoenix, AZ, US
 09:03 AM arrival
 Note that this flight is operated by NESA AIRLINES DBA AMERICA WEST EXPRESS
 Equipment: Canadar 900 [class Y]
 Class of Service: Economy/Coach
 Seat Assignment: View Seat Map
 Total Flying Time: 3 hours, 2 minutes
 On-Time Rating: 90%
 US Airways flight 2704 In Policy
 Sky Harbor Intl Arpt - Phoenix, AZ, US
 09:32 AM departure
 Los Angeles Intl Arpt - Los Angeles, CA, US
 10:17 AM arrival
 Note that this flight is operated by NESA AIRLINES DBA AMERICA WEST EXPRESS
 Equipment: Canadar Jet
 Class of Service: Economy/Coach [class Y]
 Seat Assignment: View Seat Map
 On-Time Rating: 60%

This application is coded to go to specific locations for specific services.
That's fine; it works ... but it's still a point-to-point architecture

Illustration of point-to-point ...

16
IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD
© 2007 IBM Corporation

Inside of IBM we use a tool called "Online Travel Reservations" (OTR) to make our travel plans. It has a Java client component that talks to a server-based component, which then goes out and gathers information from various external services such as airlines, hotels and car rental companies. We provide information about dates and times and locations, and it goes out and looks for flights and hotels that meet our criteria.

It should be obvious that IBM does **not** maintain its own database of flight schedules and seat availability or whether a hotel is sold out. It gets that information from external systems that are *services* to IBM's travel application.

Note: whether it's a "Web Service" I do not know.

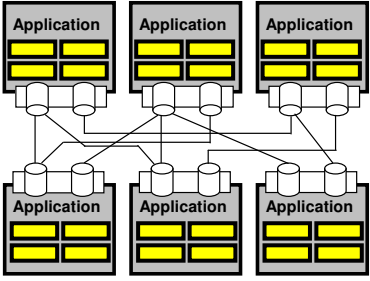
It does this information gathering in a sequential fashion, as the captured bitmap illustrates. So this application is really a composite application; one that invokes a string of other services to fetch in the information it needs. The services being invoked are *not* "for IBM's use only" ... they're standard industry services made available to others who need the same type of information.

This application knows about the location and interface requirements of those services because the application developer researched it and made sure the OTR application worked with it. That's fine ... it works. IBM's OTR is loosely coupled to these services but discovery of where these services reside is most likely not dynamic, and the interface requirements are most likely statically maintained by the application.

Note: I'm speculating here ... I do not know exactly how this application is coded. It's been around for several years and my guess is it does not yet use things like Web Services or an ESB to locate the services. One day it will.


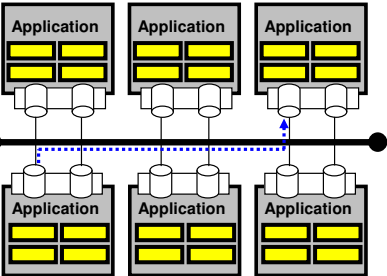
Services Alone Does Not an SOA Make

Our picture is still point to point:



It's a good start ... the applications are more loosely coupled. The implementation are "hidden" behind their service interfaces.

But to realize even more flexibility, what we need is a kind of universal exchange bus that takes care of connections.

Here, requesters of services do not need to know where the service is located; they simply connect to the "bus." The bus then routes to the service.

Ideally, this bus would do other things -- transform protocols, route based on message content, enforce quality of service, etc.

This picture is still a bit abstract

Introducing the "Enterprise Service Bus" ...

17

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

If we revert to the illustration model we used before, what we now see is that our applications are all implemented with a service interface. That's a good start ... the applications are more loosely coupled to one another. The complexity of the implementation behind the interface is hidden, so now all requesters need to know is where the service is located and what its input and output requirements are.

But the connectivity is still point-to-point. And when your enterprise has hundreds or thousands of services, and thousands or tens of thousands of requesters, then the picture becomes very complicated again. What we need is some kind of very smart, universal exchange bus that sits in the middle of everything and takes care of connecting requester to provider.

The lower picture illustrates this in the abstract. Now rather than a maze of interconnections, we have this "bus" running down the middle. Requesters connect to the bus and the bus routes the message to the service. All anyone needs to know about is how to connect to the bus; the bus does the rest.

And in an ideal world, that bus would do more than just route messages ... it would have the ability to route based on message content; to transform the messages if needed so applications that couldn't normally talk to one another now could; to enforce quality of service rules; and to provide security services to everyone.

This "ideal world" thing is the "Enterprise Service Bus," or ESB for short.

The picture above is a bit too abstract. Let's make it just a bit more real.

Command Control Mapped Onto Existing Network

The “Enterprise Service Bus” is really additional command and control intelligence added to your physical network. Implemented in middleware:

A more “physical view” of it:

Middleware that provides key function in an SOA environment:

- **Messaging services**
Support different message types; content-based routing; guarantee message delivery.
- **Management services**
Monitor performance; enforce SLA
- **Interface services**
Support web services standards and provide “adapters” for non-standard interfaces
- **Mediation services**
Transform messages between formats.
- **Security services**
Encryption, authentication, authorization

Necessary things to keep order and keep this from being chaos

Go back to first picture ...

This concept can be slippery ...

- Function mapped onto *existing* network, not a replacement of the network with some kind of separate “bus”. However, messages are handled inside middleware that implements the ESB.
- Not necessarily a single middleware product; multiple products may be combined to build up ESB functionality
- It might be best to avoid strict definitions of what constitutes an ESB. Perhaps a focus on the major function “types” to the right is better way.

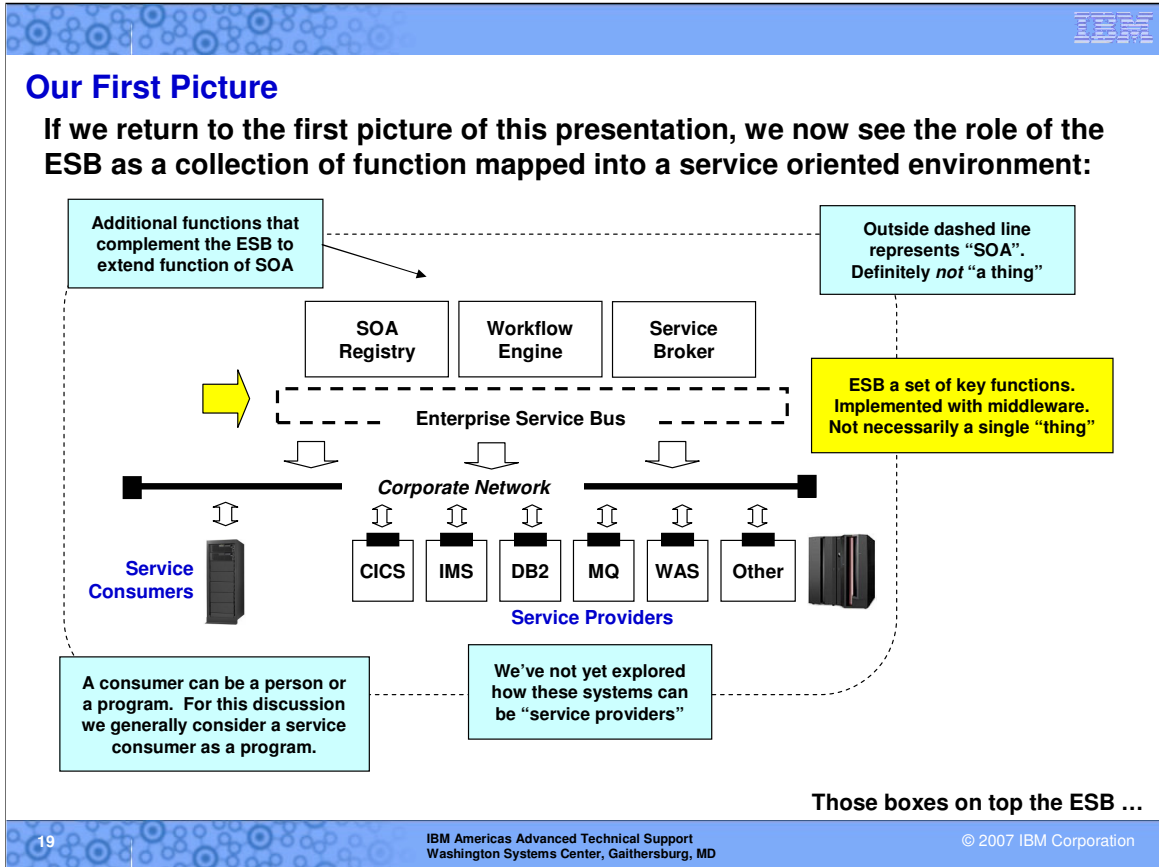
18 IBM Americas Advanced Technical Support Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

Here we see the “Enterprise Service Bus” in a more physical representation ... as function mapped on top your existing network. Service consumers and service providers connect to this existing network. When something requests a service, this request flows to the middleware that implements the ESB, and that middleware then determines where the service is located and routes the message there.

But the ESB does more than just simple routing. It also does the things mentioned in the bullet list on the right side of the chart. We’ll see how those things come into play when we take a closer look at the products that implement the ESB -- WebSphere Enterprise Service Bus (WESB) and WebSphere Message Broker (WMB).

We need to be a bit careful here ... the concept of the ESB can be a bit slippery.

- We’ve already made the point that the ESB is function that maps onto your existing network, and that it’s function implemented in middleware. It is not a new *physical* bus, but rather a logical bus on top your existing real network.
- Messages that are handled by the ESB do flow up into the middleware product that implements the ESB and then back out again.
- It’s not necessarily a single middleware product. Multiple products may be involved.
- Just like the definition for “SOA,” a strict definition for “ESB” might be something you avoid. There are a lot of different definitions out there. For now, just think of the ESB as function that brings service requester and service provider together, and provides a higher level of command and control over the whole process.



Let's now cycle back to our starting picture and see how the ESB fits into the overall scheme. As we stated, the ESB is a set of function that's mapped onto your existing network. It is implemented as middleware that is made available to service consumers and service providers, and is the means by which consumers gain access to services offered.

The three additional boxes you see above the ESB represents additional function, above and beyond the essential functions of the ESB, that extends the picture to be more "SOA." SOA itself is definitely **not** a "thing" or a single product, but rather a collection of functions as well as an overall philosophy of orienting the architecture around the concept of reusable services.

Let's take a slightly closer look at those boxes on top the ESB ...

Registry, Workflow Engine and Service Broker

Let's briefly look at the additional functions we mentioned on the previous chart:

The diagram illustrates three components (SOA Registry, Workflow Engine, and Service Broker) positioned above a dashed-line box labeled 'Enterprise Service Bus'. Below the ESB is a thick black line representing the 'Corporate Network'. Three downward-pointing arrows connect each component to the Corporate Network line.

SOA Registry

- Maintains information about the services:
 - Interface descriptions (WSDL)
 - Non-standard interface descriptions
 - "Meta-data" such as relationships between services, service level agreements, governance rules

Workflow Engine

- Executes defined "business processes" -- strings of services and non-I/T activities that constitute a defined process within company
- Invokes "linking code" as needed between services
- Capable of instituting exception processing as needed

Service Broker

- Coordinates the matching of request with the associated service
- Typically will work with the ESB to handle the lower-level connection protocols
- Likely to see this as function imbedded in the middleware that implements the ESB

These are functions ... not necessarily "products." But in some cases the functions are implemented as products. Let's look at how these things are physically implemented.

Another way of looking at this ... more comprehensive ...

20 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

Some pictures of the SOA solution include additional functions "above" the ESB. For the purposes of this presentation we'll show three: the Service Broker function, the Workflow Engine and the SOA Registry. These are functions, not necessarily products ... but in the case of IBM some products do implement them.

- **Service Broker** -- the function that facilitates the matching of requester to provider. This is a pretty fundamental function, and in most cases this function will be part of any middleware product that implements the ESB.
- **Workflow Engine** -- this is the function that helps coordination -- or "choreograph" -- the flow of higher level business processes by executing application processes and routing work items to human-oriented tasks ... all in a "flow" that you designate. When some other code is needed to prepare the work to flow from one station to the next, this function can invoke that code to do the necessary work.
- **SOA Registry** -- when the number of services you have in your "library" grows, it will be helpful to have a function to keep track of it all ... to store the necessary artifacts in a single place ... to be able to query for what services are available. The SOA registry is just such a thing.

WebSphere Message Broker -- an Implementation of an ESB

An example of an IBM middleware product that implements some of the functions we mentioned on the previous chart is WebSphere Message Broker

WebSphere Message Broker is an execution environment for something called a “Message Flow” ... a program that takes inbound messages, performs transformations you define, and routes to the desired service

(We have much more to say about WebSphere Message Broker)

Some key points to keep in mind:

- WebSphere Message Broker is implemented as a series of address spaces on z/OS. It's based on MQ.
- WebSphere Message Broker is not the only IBM product that implements an ESB ... WebSphere Enterprise Service Bus and WebSphere Process Server do as well.
- Not all the SOA functions we mentioned earlier are implemented in Message Broker. It does act as a service broker, but it is not a service registry and repository. But it is capable of *working with* IBM's Registry product

Other IBM Products and SOA ...

21 IBM Americas Advanced Technical Support Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

In an attempt to make this more “real” for you, here’s a picture of an actual IBM product that implements the “ESB” (as well as the “Service Broker” function). The product is WebSphere Message Broker (WMB). It’s *not* the only ESB product ... another called WebSphere Enterprise Server Bus (WESB) also provides the ESB function, though with slightly different capabilities. We’ll cover both in much more detail in the third unit of this workshop.

WebSphere Message Broker is a really an execution environment built on MQ. It runs a special kind of program called a “Message Flow” that determines the way a received request for a service is handled ... from its routing to the service, to any message content augmentation or transformation you desire, to switching between different protocols ... the message flow is code that runs inside WMB that does this.

We have a whole lot more to say about WMB (and WESB) later.

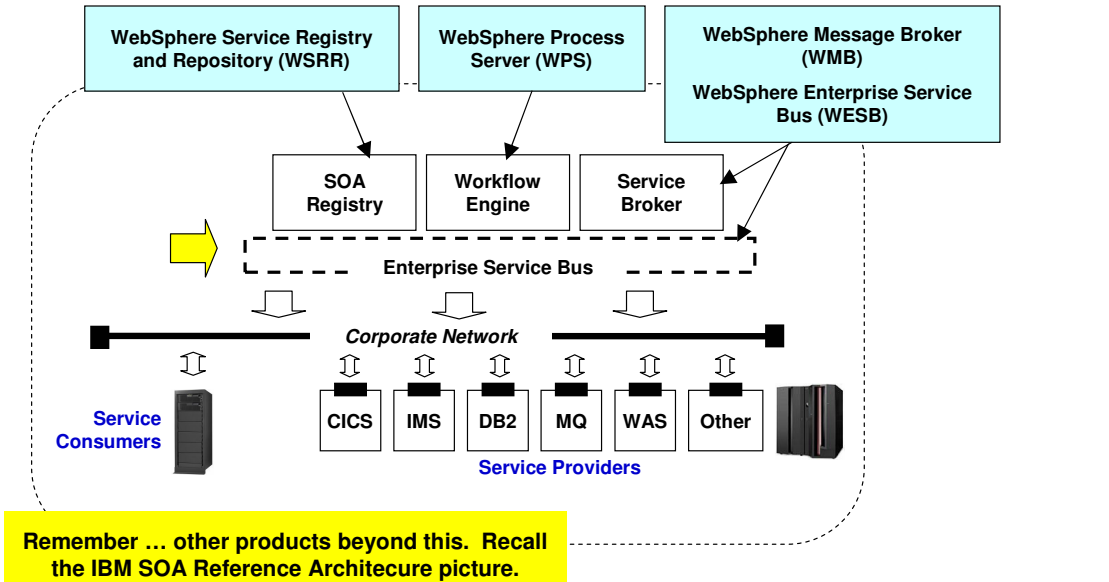
But for now, let’s go with a few key points:

- WMB is implemented as a series of z/OS address spaces. It’s based on MQ and has as part of its structure an MQ queue manager.
- Like we said, WMB is not the only IBM product to implement the ESB. WESB does as well.
- WebSphere Message Broker implements the essential ESB functions as well as the service broker function ... but not the Workflow Engine or the Registry function. Those are separate products which we’ll see in a bit. However, WMB *works with* the products that implement those functions.

Let’s look at what other IBM products are part of this picture ...

Other IBM Products and SOA/ESB

Back Again to our first picture ... this time with IBM products mapped onto the picture:



The role of standards in the SOA/ESB field ...

Going back to our original picture one more time and let's map other IBM products to it. We've already seen WebSphere Message Broker filling the role of the ESB and Service Broker. We mentioned WebSphere Enterprise Service Bus (WESB) doing the same. The Workflow Engine function is implemented with WebSphere Process Server (WPS). And the SOA Registry function is implemented with the WebSphere Service and Repository (WSRR) product.

Note: this is *not* the complete list of IBM products that play in the SOA space. There's a whole bunch of products -- particularly in the realm of development, modeling, monitoring and governance -- that also play in this space. If you go back to the "IBM SOA Reference Architecture" picture from a few charts ago, we see that there's much more to this than just the ESB and the three functions we're highlighting. Again, we're limiting our picture so we can focus on the core of this.

The Role of Industry Standards in the SOA/ESB World

It's critical. It's what enables this whole thing to be possible. Lack of open standards -- and general resistance to standards in the past -- is what inhibited this earlier.

The diagram illustrates the SOA/ESB architecture. At the top, two 'Service Consumer' boxes are connected to an 'Enterprise Service Bus' (ESB) represented by a dashed box. Below the ESB, a horizontal line represents the bus, with four 'Service Provider' boxes (each containing a gear icon) connected to it. Arrows indicate the flow of data and services between consumers, the bus, and providers.

Many existing standards already in place:

- TCP/IP
- SOAP, HTTP
- Java, JMS, J2EE
- RMI/IIOP, JAX, JMX ... etc.

Many other standards under development

- WS-* standards from WC3 and OASIS standards bodies
- Others

Standards Development is an evolutionary process. We often implement standards into products as they develop ... update later as standard is updated.
(Example: WebSphere Application Server -- constantly implementing new standards as they're developed)

IBM is deeply involved in standards development and is committed to open standards
(Wasn't always the case, but it sure is now)

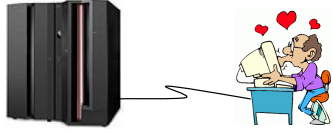
The role of System z in this ...

23 IBM Americas Advanced Technical Support Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

Industry standards play a **key** role in all this, all up and down the chain. You may not realize how many industry standards you rely upon on a daily basis -- TCP/IP, for example, is an industry standard that plays a crucial role as a unifying network protocol for the transfer of data. Higher up, we see such standards as SOAP (for Web Services ... more on that in the next unit), HTTP (as a web services exchange protocol) and Java (as a common programming language across platforms). Industry standards are what is allowing this SOA thing to open up and flourish.

There are many more standards, some in the process of development. It's important to realize that industry standards development is an evolutionary process. As standards get updated, IBM implements changes in their products. Also, please understand that IBM is deeply committed to open standards, and in fact is deeply involved in the development of many of those standards. There was a time, perhaps a decade or more ago, when IBM's commitment to open standards wasn't so solid. But IBM today has demonstrated over and over again in the way we're embracing standards all across our product line.

The Role of System z In the SOA/ESB Environment
Is there any reason System z *can't* play in this space?



	Yes	No
Capable of connection to IP Network?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Capable of running Java?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Capable of supporting open standards?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Proven and reliable platform for enterprise?	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• Products exist to implement SOA/ESB on System z?	<input type="checkbox"/>	<input type="checkbox"/>

This is the focus of this workshop -- the revealing of the products that exist on z/OS and how they work. It is our wish you come to see the value of SOA and ESB on z/OS by our revealing the “reality” of these solutions.

Web Services ...

24 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

This chart tries to lead you to a conclusion we hope is obvious -- that the System z platform is perfectly capable of hosting open standard SOA and ESB solutions. The platform is capable of doing the essential things related to this task. The question is really whether IBM has produced products for the platform. And that is the focus of the workshop.

The Internet and Program-to-Program Communications

The Internet provided a universal networking fabric. Browser to server communications are common. What about program to program?

What the industry settled on as the standard

HTML
HTTP
TCP/IP

Browser

Internet

HTTP Server

Program

Program

????
????
TCP/IP

A lot of possibilities here. In order to promote wide spread adoption, the industry knew that it had to create and adopt standards

SOAP
HTTP
TCP/IP

Bare-bones basics of "Web Services"

Web Services and the SOAP protocol ...

25

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

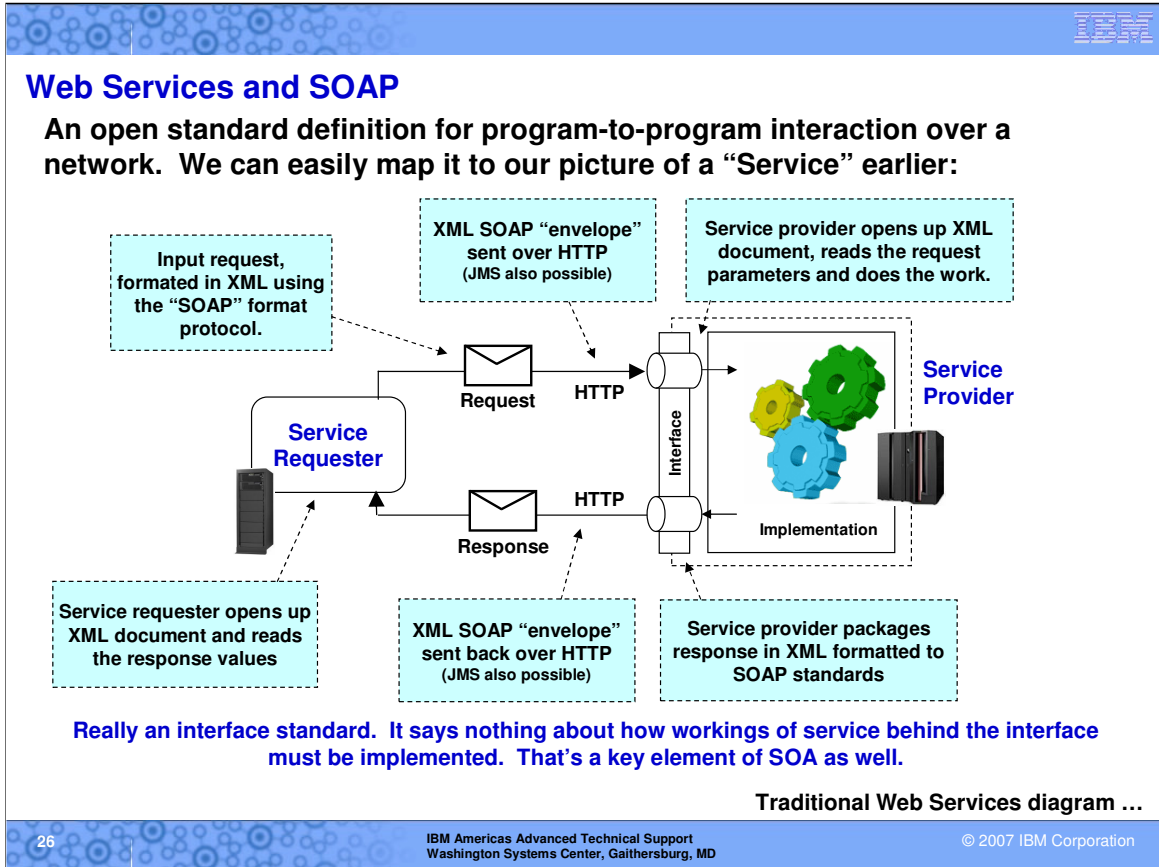
© 2007 IBM Corporation

The adoption of IP as the standard internet-network communication protocol created a universal networking fabric (meaning nearly everyone had access to everything). One of the very first things people did with this was transfer files back and forth. (The FTP protocol maps on top of the IP network.) The next big thing to come along was web browsing, which made use of an emerging standard called HTML. HTML defined the page layout and the content. HTML is carried back and forth using the HTTP protocol, which defines how a browser interacts with the web server. All that is then transported at the network layer using TCP/IP, the universal network fabric.

Program to program communications can also take advantage of the universal network fabric. And there are a lot of different ways one program can invoke the services of another. To prevent a fragmented industry, the various players got together and set about crafting a set of standards for the interaction of program to program over the common TCP/IP fabric. The standard is something called "Web Services" and it encompasses a broad range of standards. We'll focus on a subset that gets to the heart of the matter.

The lower left of the picture shows the very barest of essentials about Web Services. SOAP -- Simple Object Access Protocol -- defines a layout of XML for one program to communicate with another. That SOAP "envelope" (think "file") is sent over the HTTP protocol. (JMS is also defined as acceptable.) And all that rides on the TCP/IP network.

Let's look at how this works in relation to our earlier definition of a "service."



Now we're going to turn our attention to Web Services, which is an important element of SOA. A Web Service is a “service” just like we described before, but Web Services is built on an open standard definition.

We can use the same picture of a “service” we used before. But this time we're going to map Web Services definitions onto it.

- The service requester passes to the provider a standard-format message with its request inside. That message is formatted in something called SOAP (Simple Object Access Protocol) and is sent over standard HTTP. (JMS is also possible; more later.)
- The service provider receives the SOAP envelope (which is standard XML) on its HTTP port. Because the service provider is implemented as a standard Web Service, it knows how to open up the SOAP XML envelope and read the request information contained inside.
- The service provider then takes the request -- whatever that may be: ZIP code lookup, foreign currency exchange, credit application check, account balance update -- and does the work behind the scenes. Remember the key to services is the hiding of the implementation details behind a defined interface. *How* the service does its thing is not of interest to the client.
- When the answer is ready to go back to the client, the provider packages the answer up in a response SOAP envelope. It then sends the message back to the client. The client, which is serving as a web services client, knows how to crack open the SOAP XML and get its answer.

So in reality Web Services is really an interface standard. It defines the format of the request/response message (the SOAP envelope) used by the requester and the provider.

You may be more familiar with the more traditional way of explaining Web Services ...

The Standard “Web Services” Diagram

Perhaps you’re more familiar with this diagram, which is commonly provided to explain “Web Services”:

Univesal Description, Discovery, Integration
A repository for WSDL XML files

Notion being that when service deployed, WSDL published to UDDI. Clients browse UDDI and retrieve WSDL for info on service

Locate and retrieve WSDL

Publish WSDL

Requester

Provider

Request/Response

The area shaded in gray is the same thing as the picture on the previous page

Truth is, most implementations don’t make use of UDDI. Most implementations employ a “static binding” ... WSDL supplied directly to client.

The vision of SOA seeks to make this far more dynamic

WebSphere Service Registry and Repository (WSRR) has a broader data model, and is a fuller-function repository

Relationships ...

27

IBM Americas Advanced Technical Support
 Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

This is the more traditional diagram illustrating “Web Services”. This diagram has the requester and provider just like we’ve already shown you, but it includes something called UDDI -- Universal Description, Discovery and Integration. It’s a standard for the storing and retrieving of information about Web Services, including the storing and retrieving of the WSDL file we discussed. The picture is saying that when a service is made available, the WSDL is “published” (or stored) in the UDDI server. Then requesters can query the UDDI server to get the WSDL, read it, and thus know how to connect to the service.

The reason why this picture is a bit challenging to grasp is that very few people actually use a UDDI server. The function does exist ... it’s just that most people have implemented Web Services with what’s called “static bindings” -- that means the requester has been configured with the WSDL file manually, and that’s how it knows about the service’s requirements. If the service were to change in some way it would mean the WSDL file would need to be updated. Static binding is exactly that -- static. The vision for the future is for a more dynamic binding mechanism ... something more along the lines of what UDDI is doing.

For IBM’s SOA picture, the role of a “repository” is going to be filled by something called WebSphere Services Registry and Repository (WSRR). We’ll go into that in a bit more detail in a later unit. UDDI will continue to exist ... it’s just that WSRR is going to do that function plus much more.

Web Services, SOA and ESB ... All the Same Thing?

No, not really. It's possible to have one and not the other. Recall the old "Venn Diagram" tool from high school math:

Caution!
This is a thought exercise and discussion tool
This is not some precise statement of SOA doctrine
In other words, it's open to some debate and disagreement

A Web Services without being SOA
Very possible. SOA is a architecture and mindset. You could have a few web services apps without really being SOA.

B Web Services and SOA without ESB
Early on this may well be the case. You may commit to the concept of SOA and be working in that direction but may not yet have implemented an ESB.

C Web Services, SOA and ESB
This is the goal we've been describing ... though this is not necessarily "better" than D.

D ESB and SOA without Web Services
Other protocols, such as JMS or native MQ may be the underlying messaging format you use. WebSphere Message Broker can take those as input and transform to a different output protocol as needed.

E ESB without SOA
ESB implemented as application integration without any other consideration of SOA.

WebSphere as foundation ...

28 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

The question always comes up ... "What's the relationship of SOA, ESB and Web Services?" To answer that, we're going to go back to high school math class and resurrect the "Venn Diagram." In the picture we show, the outer box (yellow, if you have this in color) is SOA. Web Services is represented in light blue and the ESB in green. Under the picture you see arrows indicating points of intersection. Let's walk through them.

- **A - Web Services without being SOA** ... very possible. It could be a matter of degree, meaning only a few web services are deployed but in general not much of a service orientation. Or it could be a lot of web services, but all point-to-point and statically bound. Some might argue that is SOA, but many would argue it is not.
- **B - Web Services and SOA** ... the likely case early on in the adoption cycle. At this point there may not be any ESB in the picture, but the organization is thinking along those lines and the up-front work is being done to position the organization for SOA.
- **C - Web Services, SOA and ESB** ... this is one of the goals we've been describing. If an organization is committed to open standards, and Web Services is the services implementation that will be used, then this spot -- C on the chart -- is where they're going to end up.
- **D - SOA and ESB, but no Web Services** ... very possible. This is what WebSphere Message Broker is designed to do -- provide SOA for non-Web Services protocols and formats. Be careful -- Message Broker can also do Web Services, so we're not trying to imply its role is just non-standard stuff. Most likely people will have a bit of both -- Web Services and non-standard protocols being handled by WMB.
- **E - ESB but no SOA** ... the Message Broker product has been around for a while. It started life as a way to integrate divergent applications. Some are still using it for that purpose. It fills that role very well ... but that by itself is not SOA.

WebSphere Used as a “Foundation” for SOA/ESB Products

More and more we are seeing WebSphere Application Server being used as the basic foundation for additional function offered by IBM

The diagram illustrates a layered architecture. At the top, four boxes represent WebSphere products: WebSphere Extended Deployment, WebSphere Enterprise Service Bus (WESB), WebSphere Process Server (WPS) containing a dashed box for WESB, and WebSphere Service Registry and Repository (WSRR). These are labeled as 'Separate FMIDs'. Below these, a large box represents 'WebSphere Application Server for z/OS' (J2EE framework, Service Integration Bus, Standards compliance and support), which contains a 'SDK supplied with WebSphere for z/OS' (V6.0.2 – SDK 1.4.2 / V6.1 – SDK 1.5). This layer is supported by 'z/OS' (zAAP, WLM, Parallel Sysplex, Sysplex Distributor, SAF ...). To the right, a list of protocols and services is shown: DB2, CICS, IMS, MQ, Broker, Web Services, HTTP, JMS, and a colon indicating more. A double-headed arrow connects this list to the WebSphere Application Server layer.

Makes good sense – WebSphere is a proven platform. Why not re-use the function by building on top, rather than gutting WebSphere function and baking it into each product?

Avoids separate maintenance streams, functional drift over time, etc.

Tooling ...

29 IBM Americas Advanced Technical Support Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

One of the things you’re going to see much more of is the use of WebSphere Application Server as a kind of “foundation” for more and more solutions. WebSphere itself provides a now proven platform for constructing solutions on top of it. WebSphere on z/OS bring the inherent power of the z/OS platform along with the standards-compliance and J2EE framework, along with a considerable breadth of connectivity options to other data stores. So solutions such as Extended Deployment, WebSphere Enterprise Solutions Bus (WESB), WebSphere Process Server and the new WebSphere Services Registry and Repository -- all are constructed with the prereq of WebSphere Application Server as its base.

This makes good sense. If the WebSphere function was included inside each of these products, it would create a more complex maintenance structure, and would very likely result in multiple instances of WebSphere on the same MVS image when one would have sufficed.

Development Tools for SOA/ESB

The different IBM tools all operate on the Eclipse foundation. They provide different things. You acquire what you need ...

- Web development (servlets, JSPs)
- Web services development
- XML and DB access tools
- J2EE/EJB & Portal Development
- Component Testing
- Code Review & Runtime Analysis
- z/OS Application Development
- XML Services
- BMS Map Editor
- COBOL and PL/I DB2 Stored Procedures
- EGL COBOL Generation
- BPEL based processes
- WebSphere Broker development

Rational Application Developer

Rational Web Developer

WebSphere Developer for zSeries

WebSphere Integration Developer

WebSphere Message Broker Toolkit

Eclipse

All within a consistent look-and-feel framework

We'll see a bit more about IBM's Eclipse based tools later

Governance ...

30

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

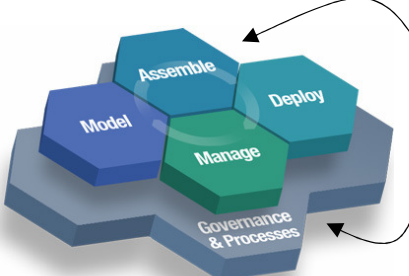
A great deal of the SOA story is going to be told in the development tools, and for IBM those are all based on the Eclipse framework. Eclipse is now an open-source graphical interface that allows multiple “plugins” (or features) to be incorporated onto Eclipse. The benefit of this is that it provides a consistent look and feel across tools designed to do very different things. The learning curve to learn the different tooling is diminished because of this, and it provides some degree of cross-tool synergy.

The picture above shows a piece of the architectural picture. More tools than just this exist, but these are the ones we're going to focus on.

Peek at “Governance”

Governance is the systematic and disciplined planning and implementation of a Service Oriented Architecture. This is to avoid a poorly structured SOA.

There's nothing about SOA that guarantees a well structured environment ... it takes good planning and execution



- **Solution design and implementation lifecycle**
Based on business need, experience ... aided by technology
- **Process management and oversight lifecycle**
Based on experience, aided by technology

What this is suggesting is that SOA involves same key things that *any* project involves:

- Proper planning and consideration of the relevant issues
- Actual development of solution and implementation of the solution
- On going management and monitoring, along with updates and improvements
- An understanding of who the decision makers are, and where lines of responsibility fall between different organizations
- Commitment to the effort, support of key sponsors, discipline

SOA is a new concept, but the need for careful planning and discipline is not new or unique to SOA

Original Objectives ...

31 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD © 2007 IBM Corporation

We have a whole unit on “Governance” at the end of the workshop. Here we give you a peek -- and you’ll see that at its heart, governance is about *careful, intentional, disciplined planning and execution*. As the chart says, there’s nothing about “SOA” that magically guarantees a well structured and successful SOA environment. In fact, there are many things that can go wrong ... mostly having to do with poor planning and poor execution.

IBM makes various tools and technologies to assist in this process. But ultimately it comes down to you and the commitment of the organization to plan and implement in a disciplined fashion. That’s the best recipe for success.

In the delivery of the workshop the topic of governance comes up over and over again ... so much so that when we get to the last unit it’s usually a somewhat redundant presentation. That’s how integral governance is to SOA.

Revisit the Objectives

We have six high-level objectives:

The diagram illustrates a Service-Oriented Architecture (SOA) with six high-level objectives. Objective 1 is to understand the "big picture" concepts. Objective 2 is to appreciate the "value proposition" of SOA. Objective 3 is to understand the major functional components of SOA, which include the SOA Registry, Workflow Engine, and Service Broker. Objective 4 is to understand IBM products used to construct the Enterprise Service Bus (ESB). Objective 5 is to understand how you can "expose" applications as services. Objective 6 is to see how System z plays a role in this architecture. The architecture shows Service Consumers on the left, a Corporate Network in the middle, and Service Providers (CICS, IMS, DB2, MQ, WAS, Other) on the right. The ESB is positioned above the Corporate Network, and System z is shown as a server rack on the right side of the Service Providers.

32

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

© 2007 IBM Corporation

We cycle back and look at our original six objectives. Did this introductory presentation serve its role and answer -- at the high level intended -- these things?

End of Unit