




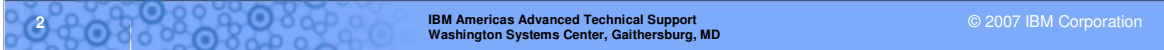
## Web Services Support on z/OS

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

© 2007 IBM Corporation



**This slide intentionally left blank**



2

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

© 2007 IBM Corporation

## Agenda of This Presentation

- **Deeper Look at Web Services**

Including a peek at the SOAP envelope, the WSDL file, and some background on various Web Services standards

- **WebSphere Application Server**

See how that product supports Web Services, and how you can access data in backend systems such as CICS, IMS and DB2 through a web service running in WebSphere.

☐ Hands on lab -- Simple Web Service deployment in WebSphere

- **Exposing Existing Applications as Web Services**

Leveraging your existing application base and making them available as a “service”

- CICS

- IMS

- DB2

We'll take a look at how each of these data systems supports the configuration of a “front-end Web Services interface”

☐ Hands on lab -- Exposing CICS application as Web Service

Web Services ...



3

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

© 2007 IBM Corporation

In this presentation we're going to take a closer look at Web Services and specifically how it's supported on the z/OS platform. We're going to go into three areas and have two hands-on labs.

- We'll go deeper into what Web Services, including a peek inside an example of the SOAP envelope. We'll also take an overview look at the standards development going on in the Web Services arena.
- We'll look at how WebSphere Application Server supports Web Services. We'll see how WebSphere as a Web Services front-end to traditional data stores such as IMS, DB2 and CICS is a very common way to expose existing data assets as Web Services. We'll also have a hands-on lab where you'll create a Web Service and deploy it into WebSphere to access a CICS application.
- Finally we'll look at how you can “expose” existing applications as Web Services in three of IBM's key data subsystems: CICS, IMS and DB2. Then we'll have a hands-on lab where you'll create a CICS web service using WebSphere Developer for zSeries (WD4Z).



A little deeper look at

# Web Services

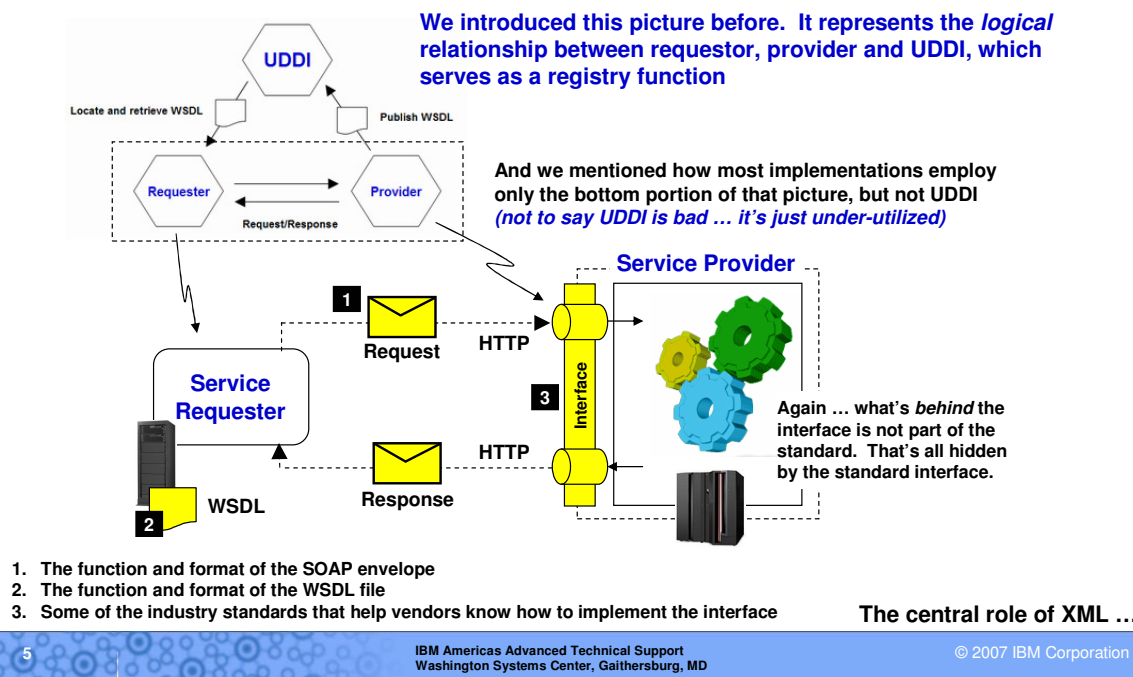
4

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

© 2007 IBM Corporation

## Reminder of What a “Web Service” Is

A system designed to support program-to-program interaction over a network.  
Its success is based on a deepening set of agreed-to standards:



We frame this discussion by reminding ourselves what a “Web Service” is. It is a mechanism to support program-to-program exchange of information, using standardized formats and protocols. Those formats and protocols are based on an expanding set of agreed-upon standards.

The two basic pieces of the puzzle are the “Service Requester” (sometimes known as the “consumer”) and the “Service Provider”. The requester asks the provider to supply some “service” -- typically in the form of a request for data. In the last unit we introduced the “three hexagon” chart that is typically used to illustrate Web Services. That picture is a logical representation of the relationship between the requester, producer and UDDI which serves as the registry function. We noted earlier that in most cases UDDI is not used, so the picture is usually just the bottom two hexagons.

We’ll now map onto that logical picture a more physical representation. We show a service requester in the form of a computer program somewhere, and a service provider in the form of something up on the z/OS platform. To understand the essentials, we need to dig into the following:

1. The function and the format of the SOAP envelope, which is the XML document passed between the requester and the provider. The input values the service needs are supplied by the requester and packaged in the XML file.
2. The function and format of the WSDL file, which is also XML and which supplies the necessary information about the service to the requester so it knows how to format up its SOAP envelope and knows where to send it.
3. The industry standards that are in place that allow vendors to know how to implement the Web Services interface so it abides by the standards.

What’s behind the interface is not part of the standard. As long as the Web Service interface adheres to the standard, then what takes place “behind the curtain” is not a concern to the requester.

## The Importance of XML in Web Services

You will see that XML is the common mechanism to exchange information in a web services environment. What is XML, and why is it valuable?

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <q0:DFHCOMMAREA>
      <CustNo>3</CustNo>
    </q0:DFHCOMMAREA>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Example of XML SOAP envelope  
we'll use in one of the labs

A series of "tags" that mark the beginning and end of blocks of XML

It holds both the data, as well as description of the data  
*<CustNo> provides an indicator of what the data is; "3" is the actual data.*

It is both machine readable and human readable, which makes things relatively easy to understand  
*Contrast with bit-format protocols, where bits within bytes meant certain things. Machine readable yes; human readable less so.*

Characters use "Unicode" encoding, which means it's universally understood  
*As opposed to the old EBCDIC vs. ASCII debates*

### It is "Self Describing"

Something called a "Schema Definition" (XSD) is used to tell a program what XML tags to expect.

The WSDL file (more in moment) has XSD information

### It can be "parsed"

If a program knows what tags to expect (the WSDL supplies this), then the program can "parse" (extract) information from the XML.

The SOAP Envelope and HTTP ...

6

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

© 2007 IBM Corporation

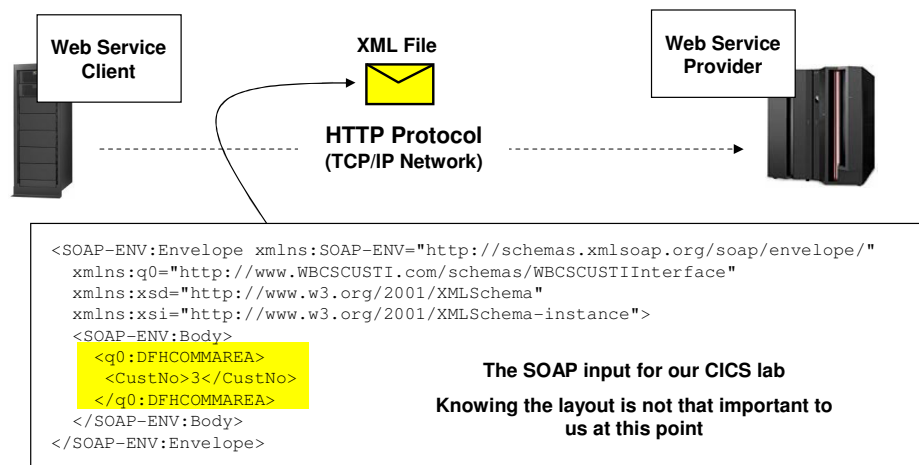
In the middle of this whole discussion is something called "XML," which stands for "eXtensible Markup Language." It plays a central role in Web Services. Here's what XML is and why it's so valuable:

- XML consists of a series of "tags," delimited in brackets < and >, that mark the beginning and end of data and blocks of data within a file. The example above is what we'll use in the CICS web service lab later. Look at the very center of that ... you see <CustNo>3</CustNo>. That digit 3 is the input to the Web Service we'll create. The tag <CustNo> indicates what the data is (a customer number). The "end tag" </CustNo> indicates the end of the data. The <q0:DFHCOMMAREA> marks the beginning of the Web Service input data; the </q0:DFHCOMMAREA> marks the end. In this example there is only one data input element -- <CustNo>, which a value of 3 supplied.
- The file is both machine-readable and human readable, which makes it much easier to work with for us humans than a packed binary format would be. The files use Unicode encoding, which means the requester and provider don't need to coordinate on character encoding.
- XML is "self describing," which means that computer programs can understand what XML tags to expect by reading a separate file, called an schema definition file. That tells what tags to expect, what kind of data will be included by each tag, which tags are required and which are optional. When a program reads the XSD, it then knows what to expect when it reads the actual XML. (The program of course needs to understand how to read the XSD ... which is one aspect of being a Web Service requester or provider.)
- The XML can be "parsed," which means a computer program can sweep through and extract the contained data by looking for the start and stop tags.

Let's look a bit more closely at the SOAP envelope, which is XML, and HTTP, which is one transport mechanism that can be used to send the XML from requester to provider.

## “SOAP over HTTP”

You’ll frequently hear this phrase. What it’s referring to is the passing of an XML document -- a SOAP “envelope” -- using the HTTP protocol



The key is that the client program knew what the provider expected -- what data elements and what XML tags to use. How did it know that? It had the WSDL file.

The Web Service Description Language (WSDL) file ...

7

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

© 2007 IBM Corporation

One of things you'll hear frequently is the phrase “SOAP over HTTP.” And what that means is that the standard request and response file format -- called SOAP (Standard Object Access Protocol). The XML that makes up the SOAP envelope needs to be sent from requester to provider. At the lowest level that goes over an IP network, but higher up in the protocol stack HTTP is frequently used to carry the XML. HTTP -- Hyper Text Transfer Protocol -- is the same protocol we all use to browse websites.

The layout of the SOAP envelope is an agreed-to standard, and both requester and provider must understand what it is and how to read it. Inside of the SOAP envelope is the input parameters the requester sends to the service. In our example it's the integer “3”, which is the customer number.

Knowing the exact layout of the SOAP envelope is not really important to us. What is important is that we know what a “SOAP envelope” is (an XML document); we understand that it is an industry standard format; and we understand that it's what goes back and forth between a Web Service requester and provider.

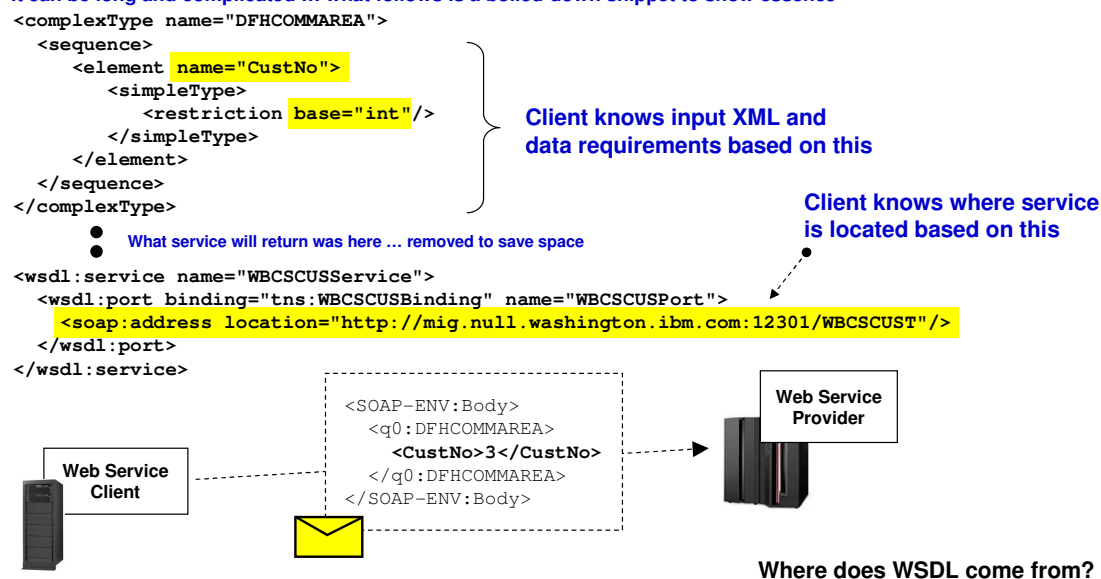
You can well imagine how difficult it would be to have a services architecture be widely adopted if requesters had no idea how to form up the basic unit of exchange with the provider. By defining a standard exchange format -- the SOAP envelope -- it facilitates a wider acceptance of Web Services.

Now let's look at the WSDL file -- the “Web Services Definition Language” file ...

## The WSDL File

WSDL contains information about the service -- where it's located, what parameters it takes as input, what it gives back as output, what XML tags to use, etc. It is sometimes known as a "bindings file".

It can be long and complicated ... what follows is a boiled-down snippet to show essence



8

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

© 2007 IBM Corporation

We have mentioned several times how important it is that the request know not only that a service exists, but also where that service is located, what that service's input requirements are, what can be expected in return. To provide that information the Web Services standard defines something called the WSDL file ... Web Services Definition Language ... which contains information about the service offered. The WSDL file uses XML, and is in Unicode encoding.

The example above is a shortened version of an actual WSDL file. It can be somewhat long and complicated. What's shown above is just a snippet.

In the upper portion of that example we see that the WSDL file is defining the input requirements of the offered service. The "complexType" is the set of input parameters defined by the service program. In this case the name is DFHCOMMAREA. (We're using as an example the WSDL created by WD4Z for the CICS Web Services lab we'll do in a bit.) This service has one input parameter -- CustNo. If it had three, for example, we'd see three <element> tags within the <complexType> block. But as it is we see only one <element>, and that's for "CustNo."

What kind of data is "CustNo"? We see that it's defined as an integer. So using this, the service requester knows that one input parameter is expected ("CustNo") and that it's an integer.

**Note:** a bit later we'll show how the WSDL also contains what the requester can expect in return. We didn't show it here because the chart would get too busy.

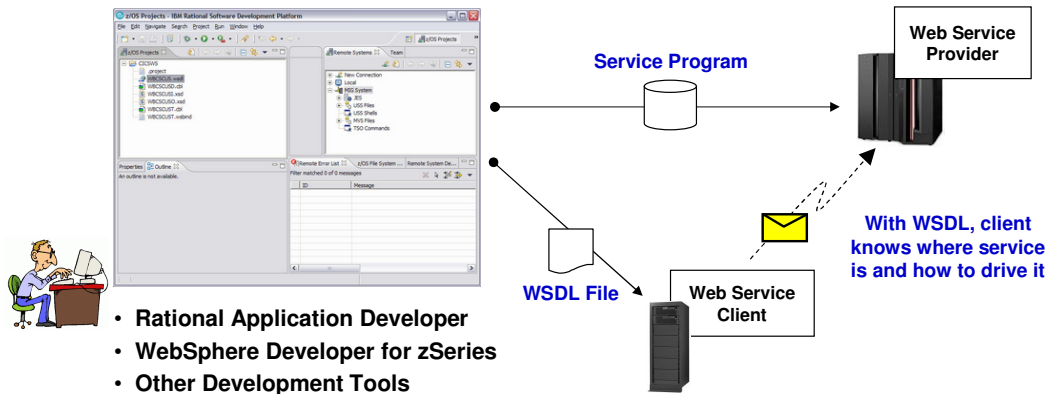
How about where to send the request? That's later in the WSDL. There we see the host and port where the service is located. We also see the "context root" of the service -- the text that follows the host and port and identifies *which* service is to be invoked.

The requester program reads this WSDL file to understand what the service expects, what can be expected in return and where the service is located and how it's invoked. But where does the WSDL file come from? That's next.



## WSDL File is a Product of Development Tool

You *could* hand-code the WSDL. More likely you'll use a development tool to create the web service, and that tool will produce the WSDL.



With only one service and one client, this is easy.

What about a hundred difference services and a thousand clients?

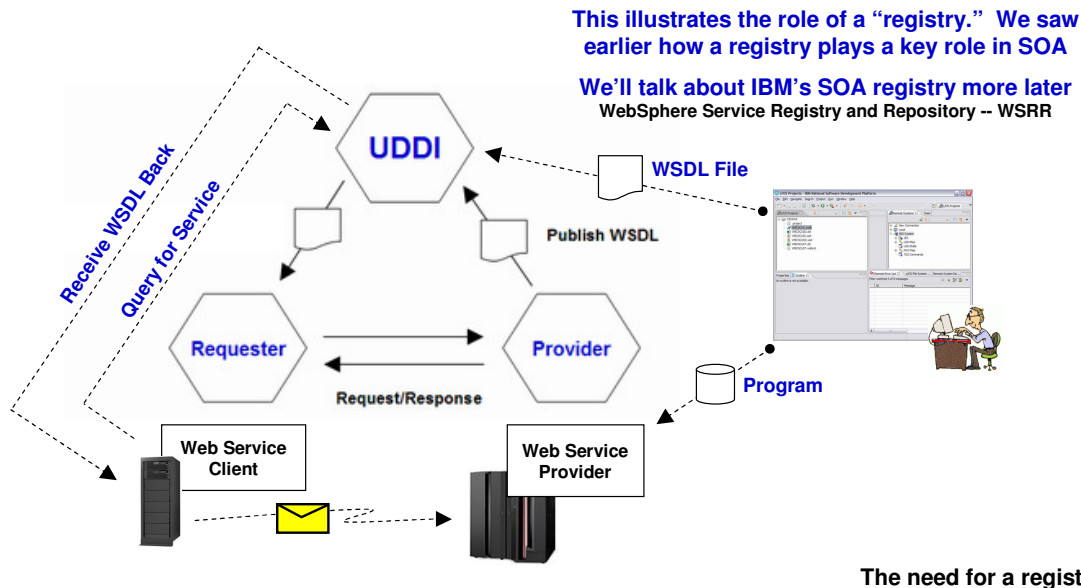
Revisit role of UDDI ...

The WSDL file doesn't magically appear. Something needs to create it. That something is typically the development program used to create the web service itself. For IBM, that development tool is increasingly coming to mean an "Eclipse" based tool, such as Rational Application Developer (RAD) or WebSphere Developer for zSeries (WD4Z). As part of the process of creating the code that implements the interface for the service, a WSDL file is also created. So what's required is for that WSDL file to be made available to the web service client so it can know how to form up its request SOAP envelope and where to send it to.

With only a handful of services and requesters, it's relatively easy to manage the manual process of making this file available to Web Service requesters. But when the number of services and requesters increases then the task becomes much more complicated. Ultimately some kind of "registry" is needed -- a place where service requesters can go get the most current WSDL file for a given service. With that we'll revisit the role of UDDI.

## UDDI, Revisited

UDDI is a server that acts as a registry for information about services that are available. Clients query the UDDI server and receive the requested service's WSDL and perhaps other information



The need for a registry ...

10

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

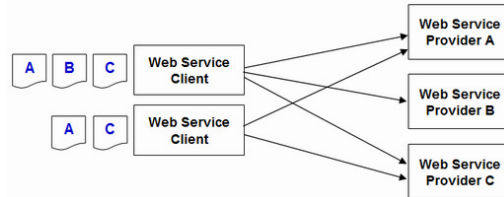
© 2007 IBM Corporation

Now we go back to the "three hexagon" picture and show why UDDI was part of that picture in the first place. The WSDL file is still produced by the development tool that develops the service itself. But rather than manually make it available to the requester, the WSDL file is "published" to the UDDI server. The UDDI server is a registry that can be queried and browsed, and the WSDL files contained there can be retrieved programmatically. The requester would go to the UDDI server, query for the WSDL based on the service name, get the most current WSDL back, and with that in hand it could then read it to determine the location and requirements of the service.

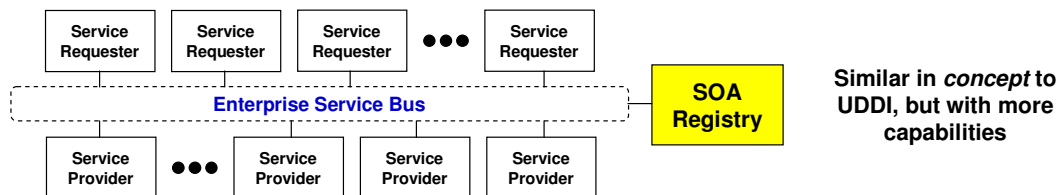
This concept of a registry is going to be a key part of SOA. IBM's SOA registry is called the WebSphere Services Registry and Repository (WSRR), which we'll cover in a bit more detail later.

## The Need for a Registry Grows as Services Grow

Up to recently, most people used a small handful of services, and maintaining the “static binding” between client and service wasn’t hard.



But the long range vision we’re painting here is much bigger ... maintaining static binding would quickly become a limiting factor in your SOA. Need a registry:



The SOAP envelope the service returns ...

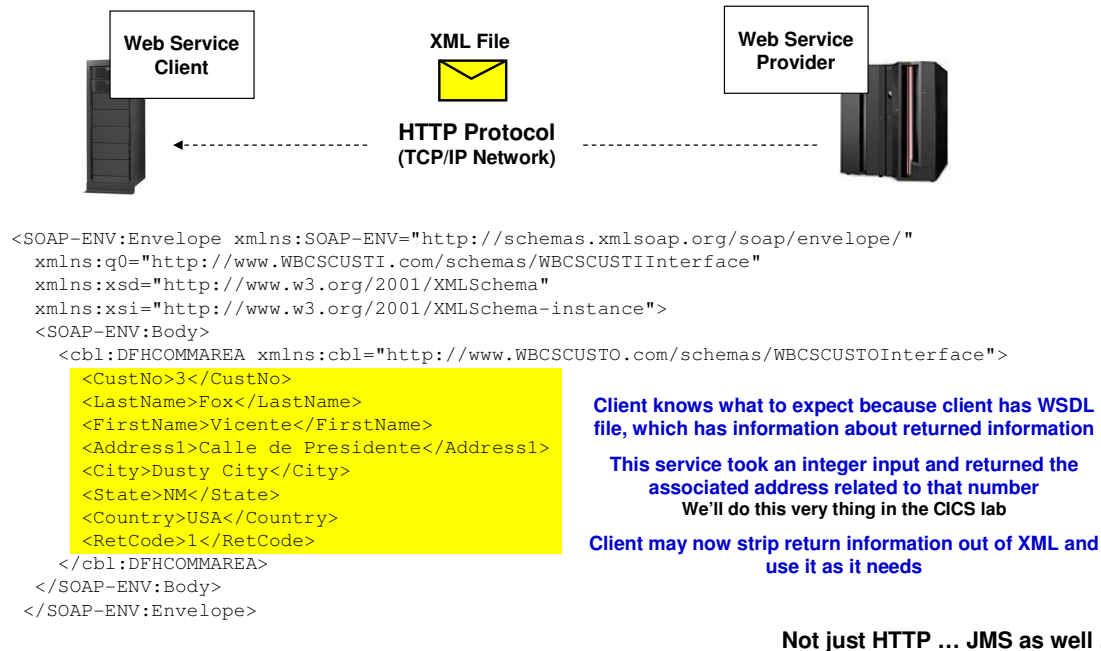
With only a handful of service providers and requesters, the need for a registry is relatively low. But as the number grows, then coordinating WSDL files gets to be more and more of a burden. The picture at the top shows an example of only two requesters and three services. If the first client needed access to all three services, then it would need all three WSDL files. If the second client needed access to only two, then it would need only two WSDL. Relatively simple.

But expand the picture to hundreds of requesters and hundreds of services. The numbers grow large very quickly. And thus a registry becomes increasingly important. That was the original intent behind UDDI, and the need is still there with SOA. The IBM SOA registry -- WebSphere Services Registry and Repository -- is similar in concept to UDDI but offers more. You can consider it like a superset of UDDI.

Now let’s cycle back to the SOAP envelope and show the results of the service invocation we illustrated earlier.

## The SOAP Response

Let's complete the picture



Not just HTTP ... JMS as well ...

12

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

© 2007 IBM Corporation

Let's complete the picture. We showed the SOAP envelope used for our request. Now let's look at the SOAP envelope we'd get back.

**Note:** this is taken from the actual we'll see in the CICS Web Services lab.

The application itself took a single input ("CustNo") and is designed to return back the full address. We saw earlier how the input SOAP envelope contained the single input held between `<CustNo>` and `</CustNo>`. The response comes back as eight values -- the customer number, the last name, first name, address, etc.

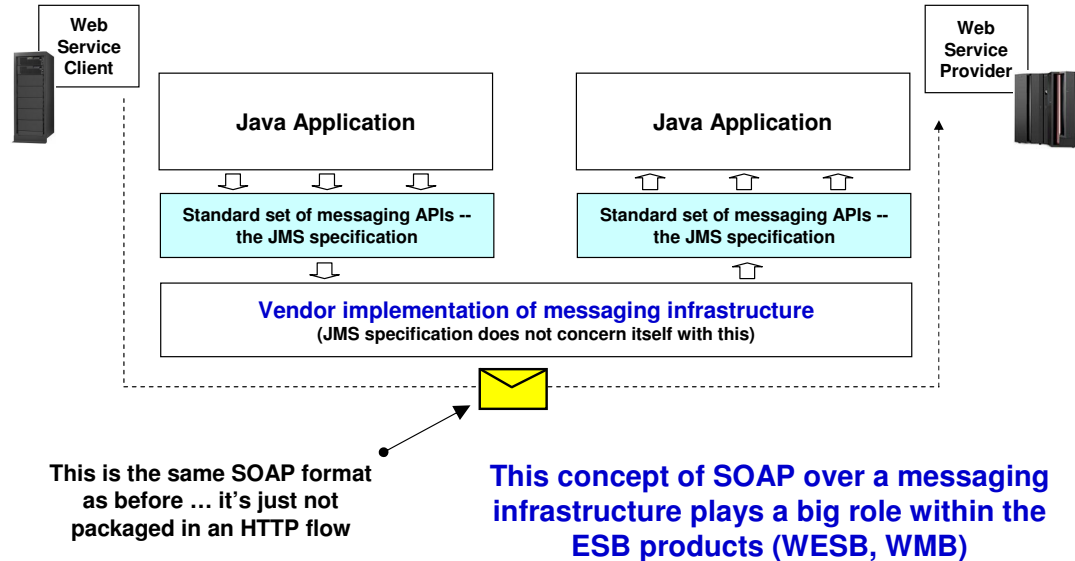
How does the requester know that it's going to get back those eight and know what the XML tags are that delimit each? Based on what it saw in the WSDL file. Thus we see once again the importance of the WSDL file -- as a means of *describing* the service being offered -- and we see once again the eventual importance of a *registry* where service descriptions will be housed.

With the response information housed in XML, and with the requester understanding what XML tags refer to what bits of information, it can now strip the key data out of the XML and use it as it needs to.

The cycle is complete: a requester invokes a remote service; the service packages up the response and sends it back. The requester uses the information it received.

## SOAP Over JMS Also Permitted

JMS stands for “Java Message Service” -- a Java specification for the *interface* to a messaging system. (In concept a lot like MQ)



The need for standards ...

13

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

© 2007 IBM Corporation

We've used HTTP as an example of the transport medium over which the SOAP envelope would be sent. But that's not the only transport medium. Another defined in the Web Services standard is JMS -- Java Message Service." JMS is a definition for the interface to a messaging system. If you're familiar with MQ, this is going to look very familiar to you.

The concept behind a messaging infrastructure is that applications will send information back and forth not across a real-time connection, but across a system that accepts a message and stores it in some centralized queue, then the receiver pulls the message off the queue. The elegance of this is that it means the requester and provider do not need to be active at the same time. The requester can send its request at any time, and the provider can pull the message when it comes back online. This kind of messaging is often referred to as *asynchronous* communications because it does not require a coordinated handshake between sender and receiver.

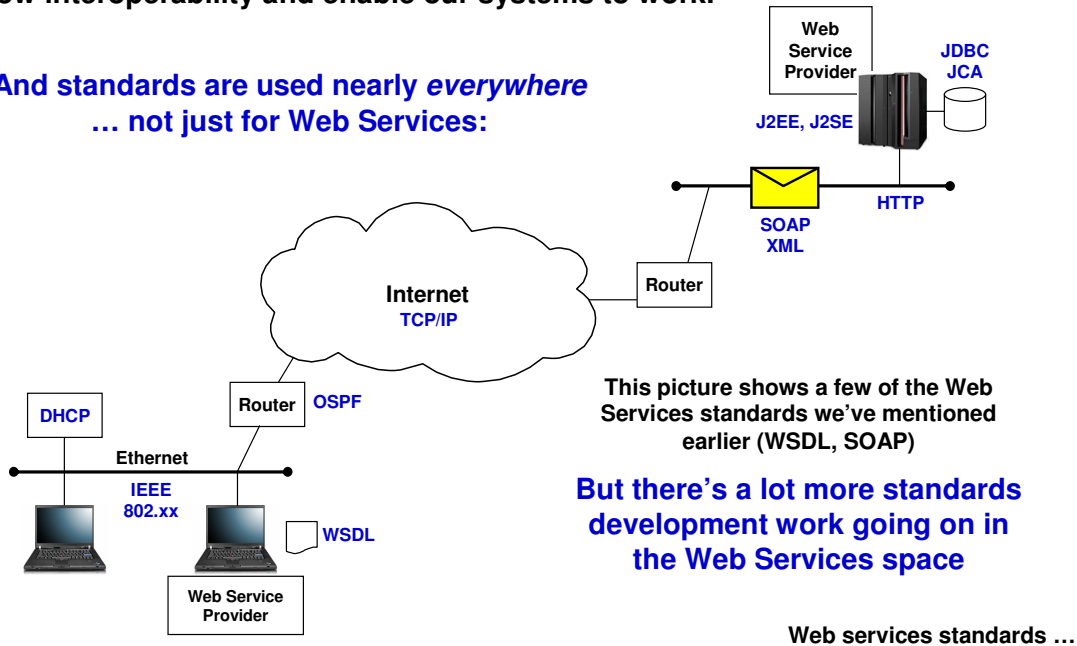
We say that JMS is an interface standard because the standard does not say how the messaging infrastructure under the covers is supposed to be implemented. All it says is that an API set be available, and however a vendor wishes to implement the nuts and bolts behind the interface is fine. One way IBM implements the messaging behind the JMS interface is to put the real WebSphere MQ back there. Another way is what WebSphere Application Server provides -- a pure Java implementation of a messaging infrastructure. That's sometimes called "platform messaging," which is meant to convey that the messaging is performed by the platform itself and not another system like MQ.

**Key Point:** this is just the transport under the covers. All the other stuff we told you about WSDL files and SOAP and such stays the same. What changes is the definition of where the service is located. With HTTP it's in the form of a host/port combination and a URL; with JMS it's in the form of something called a "destination," which is a fancy name for "queue."

## Without Standards None of This Would Be Possible

More and more we operate in a world where agreed-to standards are what allow interoperability and enable our systems to work.

And standards are used nearly *everywhere*  
... not just for Web Services:



14

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

© 2007 IBM Corporation

What holds this world of Web Services together is a set of agreed-to standards on things like the layout of the SOAP envelope, and the format of the WSDL file, and many other standards. The need for standards, and the proof that standards can work in the broader world, should be evident. We use standards every day for our most mundane things:

- Your PC communicates using one form of the 802.xx standard -- either 802.2 for wired Ethernet, or 802.11 for wireless.
- When you get an IP address it's issued up by a DHCP server. Another industry standard.
- Router to router communications are handled with something called OSPF.
- Of course the Internet is based on the mother of all standards, TCP/IP.
- We mentioned HTTP before -- that's a standard
- The acronyms you may be familiar with when working with WebSphere -- J2EE, J2SE, JDBC and JCA -- all are standards.
- And of course SOAP and WSDL.
- And there's more ... many more not shown on this chart.

There's a risk of thinking that SOAP and WSDL are the only standards related to Web Services, but that would be incorrect. There is in fact a wide variety of standards work underway for Web Services. Let's look at the scope of work underway.

## Web Services Standards

There are two primary standards bodies at work developing web services standards – W3C and OASIS:

**OASIS**  
www.oasis-open.org

Security  
Transactions  
Reliability  
Business Process  
Management

WS-Security  
WS-Transaction  
WS-ReliableMessaging  
WS-BPEL  
WS-DistributedManagement

**W3C** WORLD WIDE WEB  
consortium  
www.w3c.org

Description  
Messaging  
Publishing  
Discovery

WSDL  
SOAP  
UDDI  
UDDI

**IBM is heavily involved in both**

The architectural model ...

First, you should understand there are two primary standards bodies at work developing the Web Services standards -- W3C and OASIS. You can think of W3C doing work on the “lower level” of the architecture, and OASIS doing work on the higher levels.

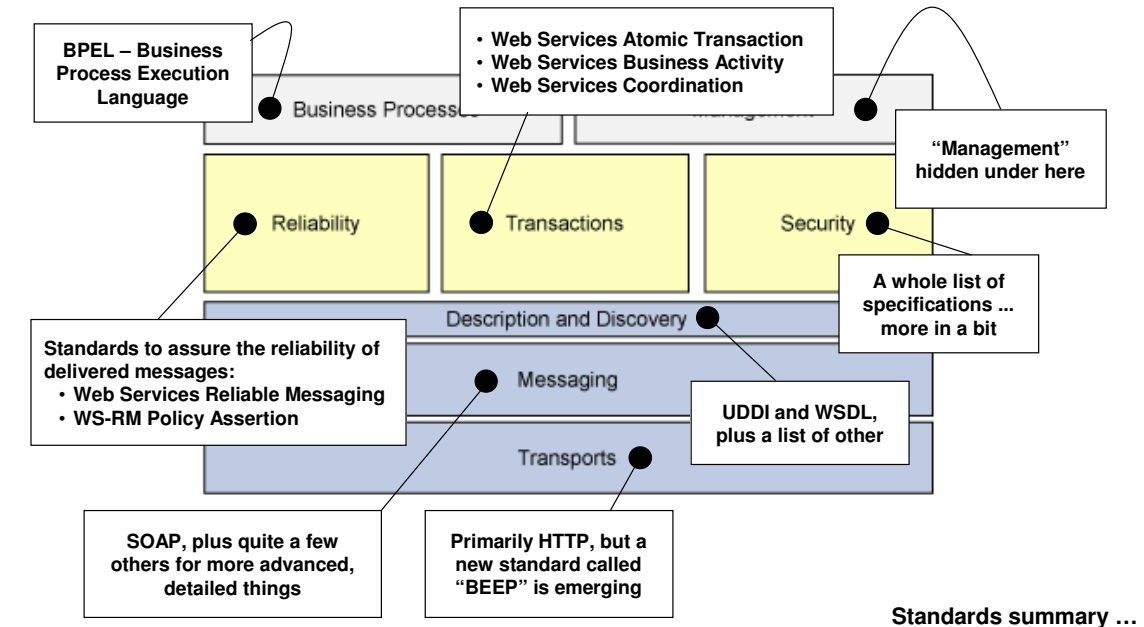
You should also be aware that IBM is heavily involved in both, and is making significant contributions to both.

To see the scope of what they’re doing, we need to look at an architectural map.

## The Web Services Architectural Model

The following picture illustrates the architectural layout of the standards:

Picture from: [www.ibm.com/developerworks/webservices/standards/](http://www.ibm.com/developerworks/webservices/standards/)



Standards summary ...

The pictures we presented earlier might leave one thinking Web Services is a simple “send a file, receive a response” type thing. But in fact its much more. The architectural picture shown above illustrates that Web Services is being seen as nothing less than a full-stack model, from the low level transports to the higher level business process and systems management level.

The URL at the top of the page is a pointer to an IBM “DeveloperWorks” page that describes in much more detail the standards work behind each of the boxes on that chart.



## Being a Web Services Provider

Some fundamental things ...



**Ability to take in (or somehow receive) the SOAP request**

Standard ways: SOAP/HTTP or SOAP/JMS

**Ability to read and understand the contents of the SOAP request**

XML parser along with implementation of the “WS-basic” standards

**Ability to act upon the request**

This is the “behind the interface” implementation we’ve shown before

**With that we now prepare to show how some of IBM’s key product systems support being a Web Services provider**

WebSphere Application Server, CICS, IMS and DB2

(WebSphere Message Broker, WebSphere Enterprise Service Bus and WebSphere Process Server come later)

**WebSphere Application Server ...**

17

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

© 2007 IBM Corporation

Up to now we’ve simply alluded to something acting as a Web Services provider. Let’s now explore what it takes to be a provider, which will then set the stage for our exploration of how key IBM systems such as WebSphere Application Server, CICS, IMS and DB2 act in this role.

The key things are:

- The ability to take in a SOAP envelope. A server box sitting alone without network connectivity won’t make a very good Web Services provider. Neither will a system that can’t communicate via HTTP or JMS.
- It needs to have a way to read and understand SOAP. That means it has to be able to “parse” XML (read through and separate tags from data). These first two bullets are part of the “WS-Basic” standard -- the basic things something must possess to play in the Web Services arena.
- Finally, there has to be something behind the interface ... something that will act upon the request and satisfy the requester. This does *not* need to be a standard implementation. Remember, the key is a standardized *interface*. What’s behind the interface is left up to the vendor.



# **Web Services and WebSphere Application Server for z/OS**



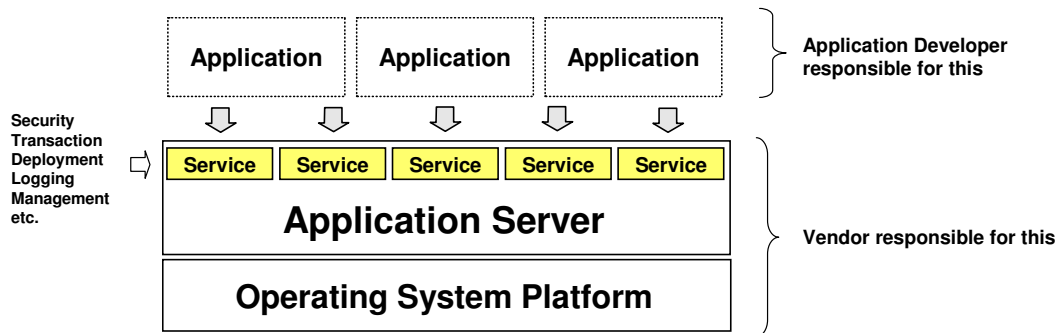
18

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

© 2007 IBM Corporation

## The Purpose of an “Application Server”

An “application server” provides a common and standard set of services that applications can use. Developers can focus on their business logic and not have to “reinvent the wheel” basic stuff.



**The concept is not new ... CICS is also an “application server” and has been around for almost 30 years! The difference is industry standards ...**

Survey of standards supported ...

WebSphere Application Server is ... wait for it ... an application server. But what does that mean? It means that it implements a series of common and standard services that applications can make use of, rather than having developers write the same stuff over and over again. That allows developers to stay focused on the business needs and pay less attention to the “plumbing” of a solution.

Examples of the “services” offered include security services, transaction services, logging services, connection services, etc.

WebSphere Application Server implements the J2EE specification to provide these services. WebSphere provides a great many industry standards as part of its overall platform.

This concept of providing an “application server” is not new. CICS is an example of an application server. The difference is that CICS started life in an era when industry standards didn’t generally exist. So it was developed using proprietary mechanisms. It worked wonderfully ... CICS has been an astonishingly successful application server and transaction manager.

**Note:** Be careful ... CICS now supports many industry standards, including Java and Web Services. The point about proprietary mechanisms was more about its origins rather than its present-day.

We’ll next explore the industry standards supported by WebSphere Application Server.

## WebSphere Specifications and APIs

The following URL provides a summarization of the specifications and APIs WebSphere Application Server for z/OS Version 6.1 supports:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rovr\\_specs.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rovr_specs.html)

- Java 2 Platform, Enterprise Edition (J2EE) specification -- 1.4
- Java 2 Standard Edition (J2SE) specification -- 5
- Java Servlet -- 2.3 and 2.4
- Java Server Page JSP -- 2.0
- Enterprise Java Bean EJB -- 2.0 and 2.1
- Java Message Service JMS -- 1.1
- SOAP -- 1.1
- WS-I Basic Profile WS-I Basic Profile -- 1.1
- Web Services Description Language (WSDL) -- 1.1
- Web Services Resource Framework (WSRF) WSRF -- 1.2
- WS-I Attachments -- 1.0
- Universal Description, Discovery and Integration UDDI -- 3.0
- WS-Addressing -- 1.0
- Web Services Atomic Transaction WS-AT -- 1.0
- Web Services Business Activity WS-BA -- 1.0
- Web Services Coordination WS-COOR -- 1.0
- Web Services Notification -- 1.3
- Web Services Security -- 1.0
- Web Services Reliable Messaging -- 1.1
- **more**

Not all standards supported are on this chart. Check the URL.

**Message is that IBM is heavily committed to open standards and WebSphere is an standard platform**

**Web Services is a big part of the WebSphere message**

The new Feature Pack ...

WebSphere Application Server is intended to be an open-standard platform. As such, it is frequently updated with the latest standards. The chart above shows some of the standards that are incorporated into WebSphere Application Server for z/OS Version 6.1.

**Note:** the chart is not exhaustive. It is a sampling. See the URL listed at the top of the chart for a better list of the supported standards.

The block highlighted in gray are those related to Web Services. Many of the standards we touched on in the previous discussion of W3C and Oasis are shown here. The message is that IBM is committed to open standards and WebSphere is one product that is implementing them in an aggressive fashion.

Of all the systems we're talking about in this section (WebSphere, CICS, IMS and DB2), WebSphere is perhaps the most robust in terms of its Web Services support.

In June of 2007 IBM released the "Feature Pack for Web Services" which took the Web Services support even further ...

## New - Feature Pack for Web Services

**“Feature Packs” are a way to provide net new function and allow customers to pick and choose what they wish to implement.**

**Compare to other method: including new function in maintenance stream**

### **Made GA for WebSphere z/OS on 6/29/2007:**

- **Web Services Reliable Messaging (WS-RM)**
- **Web Services Addressing (WS-Addressing)**
- **SOAP Message Transmission Optimization Mechanism (MTOM)**
- **Web Services Secure Conversations (WS-SC)**
- **New standards-based programming model support:**
  - **Java API for XML Web Services (JAX-WS 2.0)**
  - **Java Architecture for XML Binding (JAXB 2.0)**
  - **SOAP with Attachments API for Java (SAAJ 1.3)**
  - **Streaming API for XML (StAX 1.0)**

White Paper: [ibm.com/support/techdocs/atmsmastr.nsf/WebIndex/WP101084](http://ibm.com/support/techdocs/atmsmastr.nsf/WebIndex/WP101084)

**Message: as standards advance, IBM continues to incorporate into products**

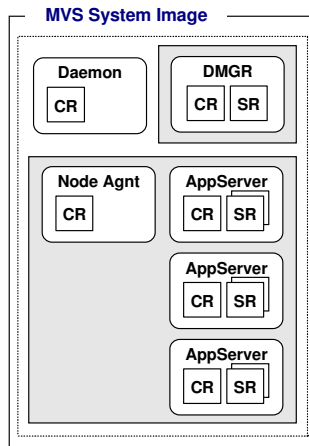
**Implemented as MVS started tasks ...**

At the end of June, 2007, IBM made available something called the “Feature Pack for Web Services” for WebSphere Application Server for z/OS. “Feature Packs” are a way for IBM to deliver new functionality for a product and allowing customers to pick and choose what they want, based on their needs. In the past all new function was delivered in the form of maintenance releases or new release or versions.

The Feature Pack for Web Services just released provides quite a bit of additional web services support. As we stated earlier, the world of industry standards is an evolving one, with new standards coming out all the time, and existing standards being advanced. IBM, as stated, is committed to open standards and implements chosen standards on an on-going basis.

## Implementing the Agreed-To Standards

The standards say what the interfaces and functions should be, not *how* vendors implement them behind the scenes.



- Implemented on z/OS as a series of started tasks
- Configured and built by submitted JCL batch jobs
- Started and stopped like any other started task

**There's more to it, of course. But the message is that your MVS knowledge is quite applicable to this environment**

**Simply run on z/OS, or exploit z/OS ...**

The people who developed WebSphere for z/OS had to find a way to implement the industry standards described for a J2EE server, and do so on the z/OS operating system. They chose to do this by using a series of MVS started tasks to represent the different server types that comprise what's called the "cell" -- which is a fancy term to mean the sum of all the servers over which the administrative application has control.

These started tasks are started and stopped like any other started task. The whole environment is built on z/OS by submitting a sequence of batch JCL jobs that construct the configuration HFS that houses all the information about the cell.

Our reason for showing this chart is to lend comfort to any MVS systems programmers who may wonder whether their existing skills can map to WebSphere. The message is they can for the most part.

There's much more to this, of course, and to find out more about WebSphere Application Server for z/OS you should attend the workshops listed at the start of this section.

WebSphere for z/OS runs on z/OS, but also exploits z/OS. That's an important distinction. More next.

## WebSphere for z/OS is ... z/OS Aware!

While WebSphere for z/OS has many elements common with the other platforms, it does exploit underlying z/OS and Parallel Sysplex strengths:

- **Parallel Sysplex** – CF for log, DB2 data sharing, DVIPA, Sysplex Distributor
- **WLM** – multiple servant regions based on goals; internal IOP routing based on WLM metrics
- **SAF** – security database for authentication, resource access, started task identities, keyrings and certificates
- **zAAP Engines** – z/OS JDK can exploit the zAAP engines for Java processing
- **TCP** -- Reduced TCP code path if communication within MVS image; use of XCF if communicating between members of Sysplex
- Use of **XCF** or **Hipersockets** for internal communications
- Value of close **proximity to data resources** (cross-memory communications)

WebSphere and inbound HTTP ...

23

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

© 2007 IBM Corporation

WebSphere Application Server for z/OS has most of its capabilities in common with WebSphere on other platforms. That's a good strategy ... it means that someone familiar with WebSphere on distributed can carry a large part of that skill up to the z/OS platform.

But we should note that the code is not entirely the same. If that were the case, then WebSphere would merely *run* on z/OS and not exploit it. But in fact there is a degree of exploitation, both direct and indirect. For example, the fact that DB2 can operate in a shared data configuration in a Sysplex, and WebSphere can take advantage of that for high availability configurations ... that's an *indirect* exploitation of z/OS. The fact that a great deal of WebSphere's internal communications routing is done based on knowledge of the environment from WLM is an example of *direct* exploitation.

In either case, the point is that WebSphere on z/OS can exploit the inherent strengths of the z/OS platform.

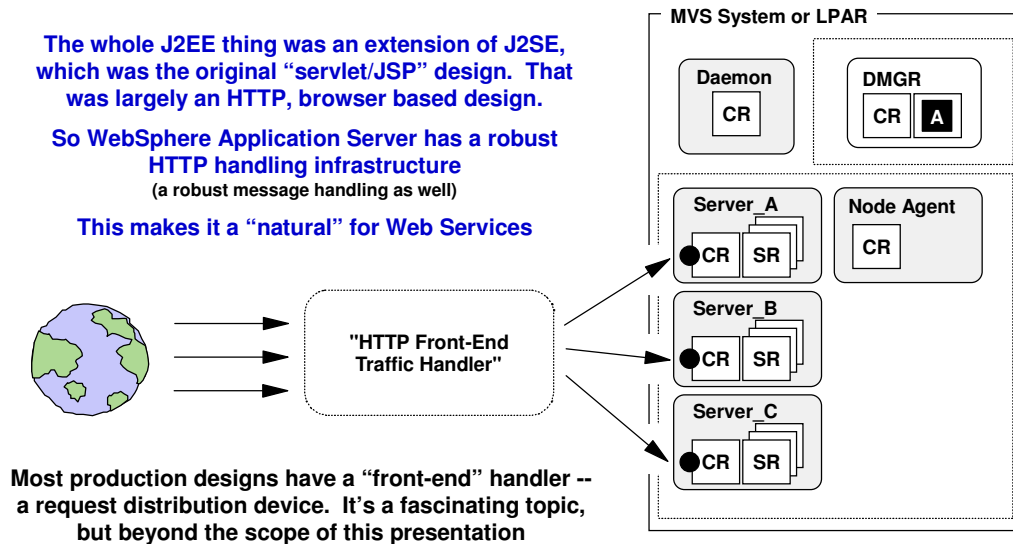
## WebSphere: A Natural HTTP Handler

The way WebSphere Application Server is architected, each of the application servers has its own HTTP listener:

The whole J2EE thing was an extension of J2SE, which was the original “servlet/JSP” design. That was largely an HTTP, browser based design.

So WebSphere Application Server has a robust HTTP handling infrastructure (a robust message handling as well)

This makes it a “natural” for Web Services



Most production designs have a “front-end” handler -- a request distribution device. It’s a fascinating topic, but beyond the scope of this presentation

Once into the server’s HTTP handler, then web service code can be invoked ...

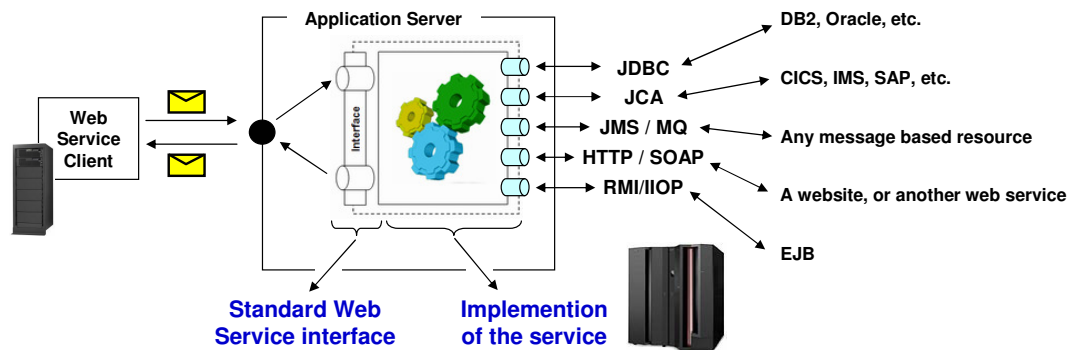
WebSphere is architected so that each application server has its own HTTP listener ports. What that means is that in most implementations there’s some kind of IP “sprayer” out front that handles the initial inbound flow, then routes it to the backend. But that’s not really the point of this chart. The point of this chart is that WebSphere has a very robust HTTP handling infrastructure. And we learned earlier that a common way to flow Web Services is over HTTP. Therefore, WebSphere has a natural affinity to doing Web Services. All that remains is the code in WebSphere to support Web Services. And we saw on an earlier chart that WebSphere supports a wide array of Web Services standards.

The key the is getting the “SOAP over HTTP” flow into WebSphere, then the web services code can take over.



## Web Service Inside of WebSphere Application Server

Is implemented as a Java program -- EJB or Javabeen. The interface provides the standard Web Service features; the rest may do whatever you wish to provide the service. And with WebSphere, many options exist:



### Two key points:

1. WebSphere Application Server can access a wide range of resources on z/OS
2. When WebSphere and the resources are both on z/OS, you can benefit from close proximity  
Cross memory speed; reduced TCP processing; potential reduced security complexity

Some examples ...

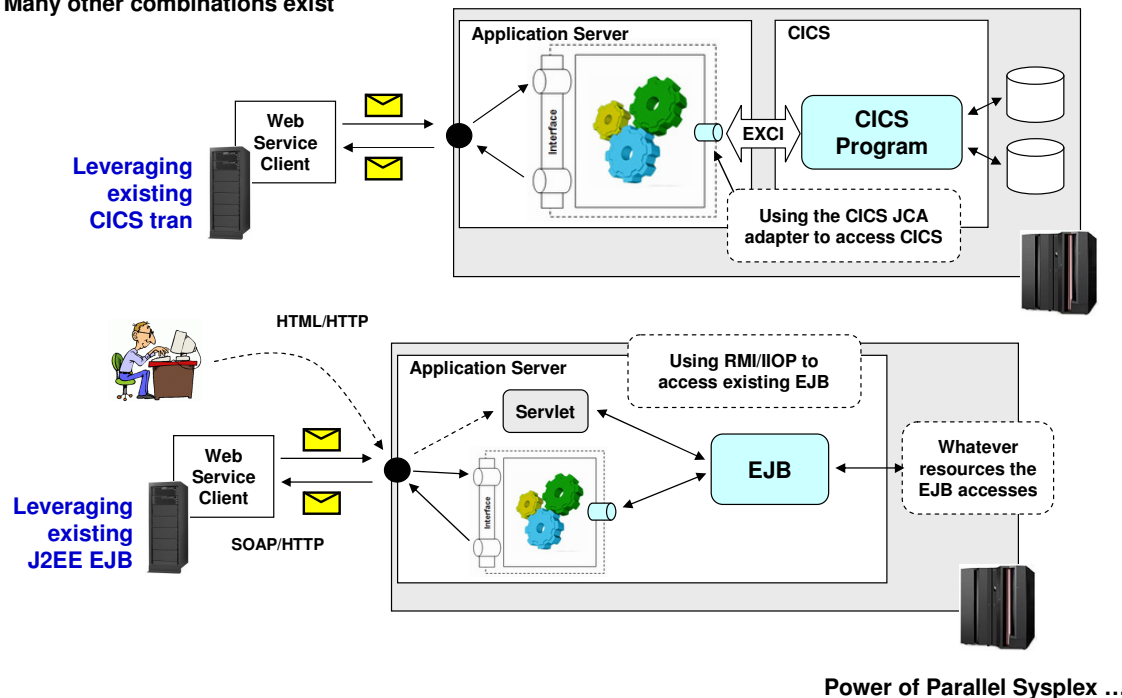
We mentioned how WebSphere is an excellent and natural HTTP handler. That means it can host a Web Service. That service will be written in Java as an EJB or a Javabeen. It will provide the interface for the service. The implementation of the service -- what does the actual work -- is whatever you create behind the interface.

That implementation has available to it all the resources WebSphere Application Server itself provides. This includes the full range of "data connector" technologies that permit access to data outside of WebSphere Application Server. The chart above shows you what those include.

The final point listed on the chart above is one that should be noted. If the data resource and the WebSphere Application Server web service is co-located on the same MVS image, there are benefits related to that proximity. In many cases cross-memory speed can be realized accessing the data. Or, if the connection is TCP, then a reduced code-path TCP connection service can be used that greatly speeds up communications.

## Two Examples of Leveraging Existing Assets

Many other combinations exist



Power of Parallel Sysplex ...

26

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

© 2007 IBM Corporation

Here are two examples of leveraging existing assets using a Web Service hosted by WebSphere Application Server. There are many examples beyond this as you'll see.

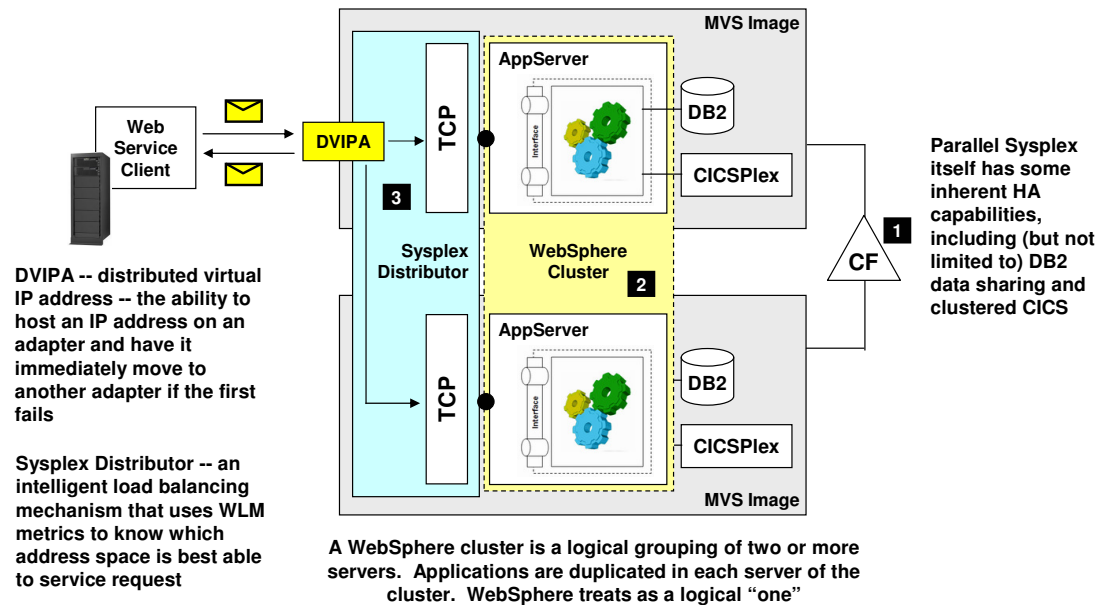
The first example shows the Web Service hosted by WebSphere, with the use of the CICS Transaction Gateway JCA adapter to access CICS. The "existing asset" is a CICS transaction program. The implementation of the Web Service would receive the SOAP request, strip out the input information, then turn and drive CICS through the EXCI interface.

**Note:** there are ways to host a Web Service directly inside of CICS, eliminating WebSphere Application Server from this equation. We'll see that in a bit.

The second examples shows an existing EJB application that you wish to expose as a Web Service. Let's say today you have some access to this EJB asset through a browser, where a front-end servlet acts as the interface between the user (a human at a browser) and the EJB. Here you can code up the Web Service interface, and the implementation of the Web Service can turn and drive the EJB using RMI. Now you've implemented a program-to-program Web Service while maintaining the existing browser interface and re-using the existing asset.

## A Highly Available Web Service

WebSphere on top of Parallel Sysplex offers some unique advantages:



WebSphere as foundation for other solutions ...

27

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

© 2007 IBM Corporation

Once again we cycle back to how the power of the System z platform can be exploited by the Web Service infrastructure that runs on top of it. In this case, we're focusing on how WebSphere Application Server can take advantage through its clustering capabilities.

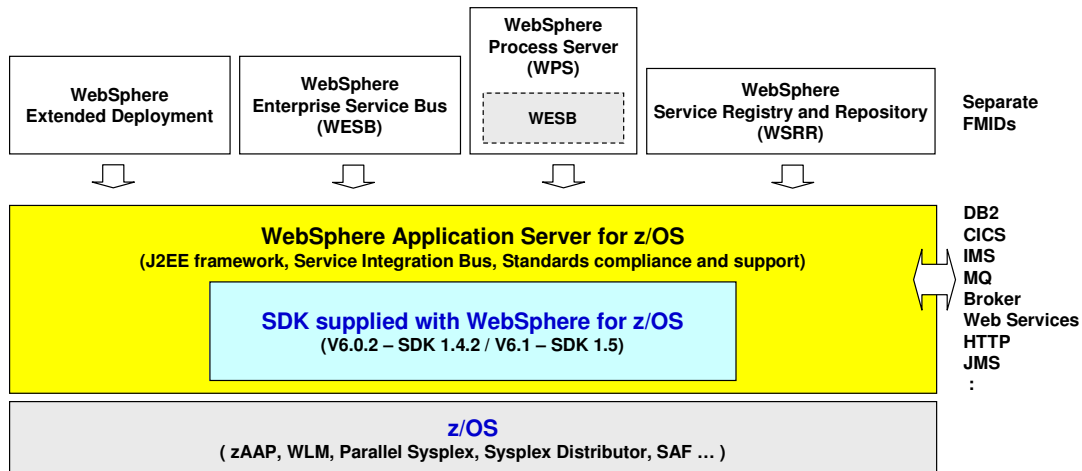
This picture is a bit complex. But the underlying message is quite elegant.

- WebSphere Application Server itself has the ability to cluster its servers. The Web Service would run as an application in a server, and if it was a cluster then the Web Service would actually run concurrently in multiple locations, ideally on different MVS images.
- Behind the WebSphere cluster you have the advantage of Parallel Sysplex. That brings the data sharing capabilities inherent in that, which includes things like DB2 data sharing, MQ shared queues and CICSplex. If your Web Service accesses a backend data store that can take advantage of shared data, then the clustered service combined with the shared data provides a powerful high-availability story.
- In front of the WebSphere cluster you again have the power of Parallel Sysplex, this time in the form of Sysplex Distributor and DVIPA. That provides a way to intelligently balance the incoming requests across the clustered servers, and in the event of an outage that removes the hosting IP adapter, the address can be immediately re-hosted.

When people speak of System z's ability to "always be up," they're referring to the ability to configure Parallel Sysplex in a way that provides for things to "always be up," even if a piece of the system goes down, either because of failure or because of scheduled maintenance.

## WebSphere Application Server as “Foundation” for Other Function

We’ve made this point before ... we’ll make it again: WebSphere Application Server is increasingly being used as a “foundation” for other functionality:

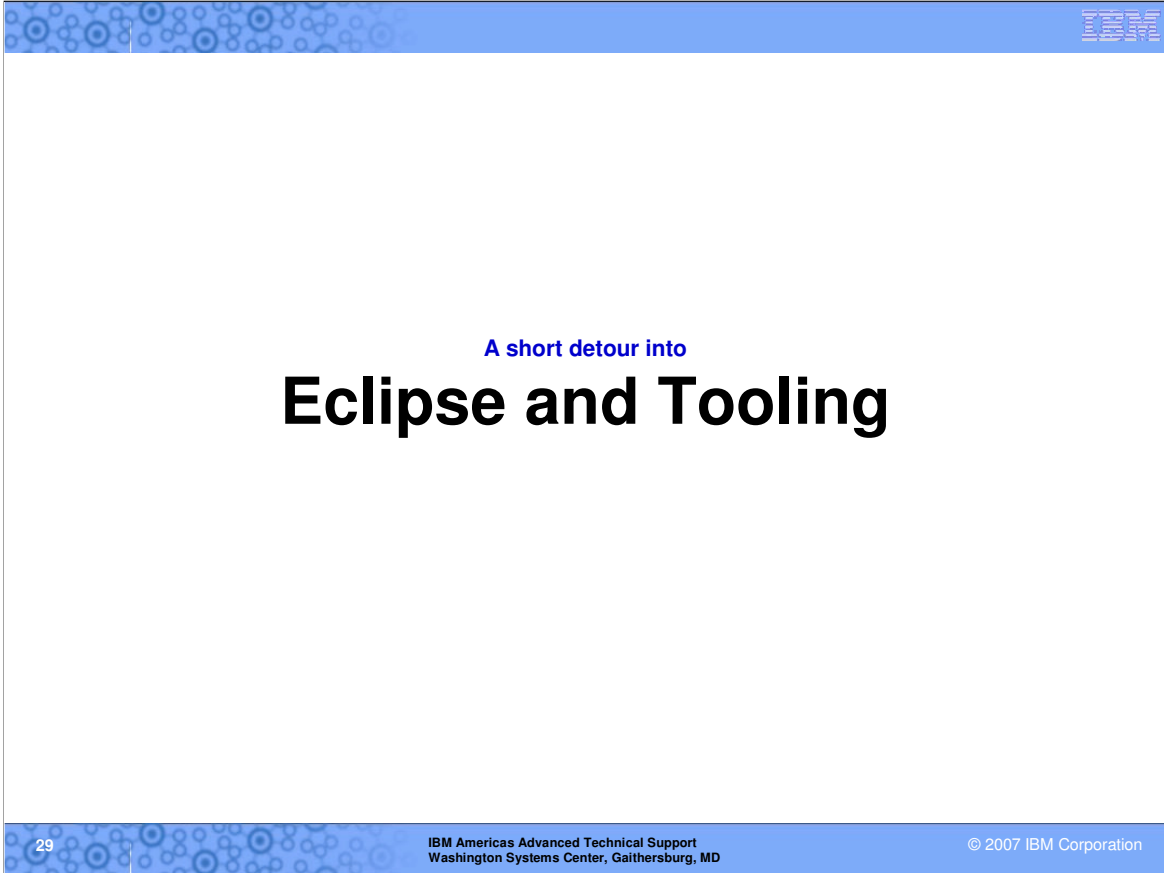


Makes good sense – WebSphere is a proven platform. Why not re-use the function by building on top, rather than gutting WebSphere function and baking it into each product? Avoids separate maintenance streams, functional drift over time, etc.

Detour into tooling ...

IBM is making extensive use of WebSphere Application Server as a foundation for other solutions. These other solutions require the kind of thing WebSphere Application Server provides -- a fully compliant J2EE environment with a rich framework for management and access to external data resources. IBM could seek to gut the function out of WebSphere and incorporate it as part of these products, but that introduces a whole slew of other complications, most notably related to maintaining consistent function and managing maintenance streams. It makes a lot more sense to simply pre-req the Application Server base and build on that with the additional function.

Now onto some brief comments about the tooling, then onto the first lab.



IBM  
Web  
Services  
Support

A short detour into

# Eclipse and Tooling

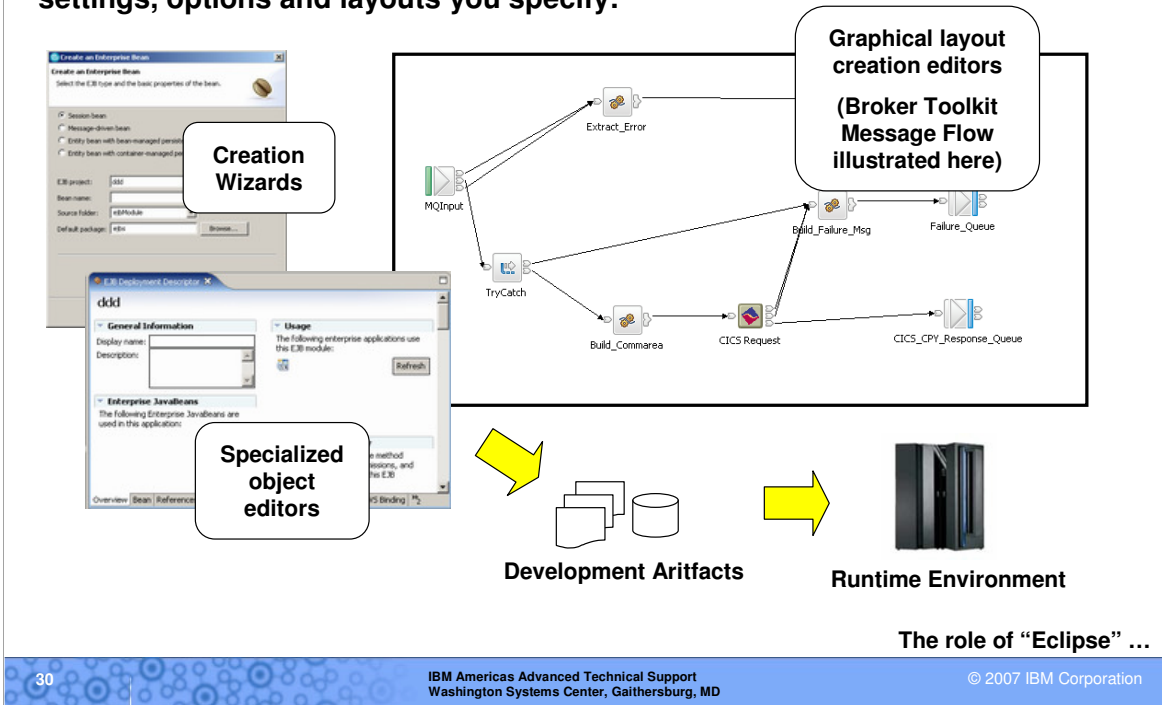
29

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

© 2007 IBM Corporation

## Basic Concept Behind Use of Workstation-Based Tooling

Use the graphical nature of workstation tool to generate code based on settings, options and layouts you specify:



If you're already familiar with workstation tooling, this will be nothing new. But if you're not, then the basic concept behind the tooling is to provide various development aids -- wizards, object properties editors, graphical layouts -- to speed the development of the final code. Ultimately what gets generated is code, either Java or some other programming language. But rather than offering a simple editor and making the developer know all the syntax and calls, most of that is hidden behind a set of things like we're showing here.

And of course, when the development artifacts are created, they are then deployed to the runtime environment. That runtime environment could be CICS, WebSphere Application Server, or one of the ESB products we've mentioned and will discuss in more details in the next unit.

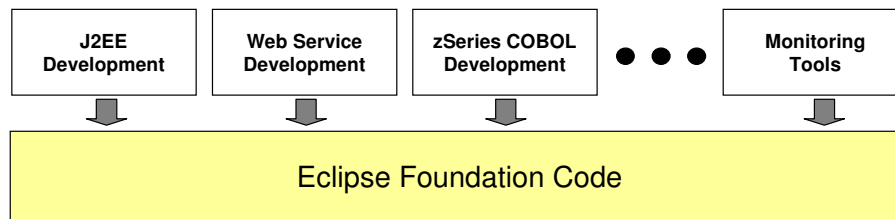
Underlying IBM's tooling is something called "Eclipse," which is a tool framework on which most of IBM's tooling now runs.

## Eclipse-based Tooling

Eclipse is an open-source community organization that provides the tooling base used by IBM. IBM is very involved with the Eclipse organization.

[www.eclipse.org](http://www.eclipse.org)

### Additional function added by adding to the Eclipse foundation



- This is what provides a consistent look-and-feel between the different development environments
- For this class we'll be using several: Rational Application Developer, WebSphere Developer for zSeries, WebSphere Message Broker Toolkit

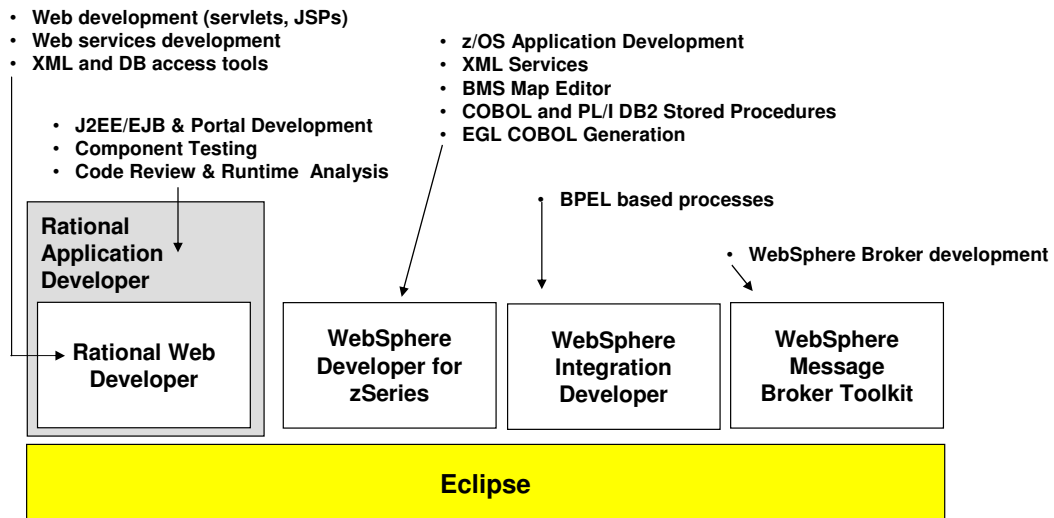
What tools do what things ...

At one point in time Eclipse was a tooling environment owned by IBM, but IBM then turned it over to the open source community and it now is managed by the Eclipse organization. Eclipse is a set of software that provides a common framework for other tools that “plug into” the base Eclipse, or foundation. The value of this is that it provides a consistent look and feel between the different development environments.

**Note:** that used not to be the case. It used to be that one tool was quite a bit different look and feel from another, and that made for some difficult learning going from one tool to another.

## Architectural Picture of Eclipse-Based Web Services Development

This is how the various tools relate to one another and functions performed:



**Message is that different tools provide specific development functions, all integrated into Eclipse-based foundation**

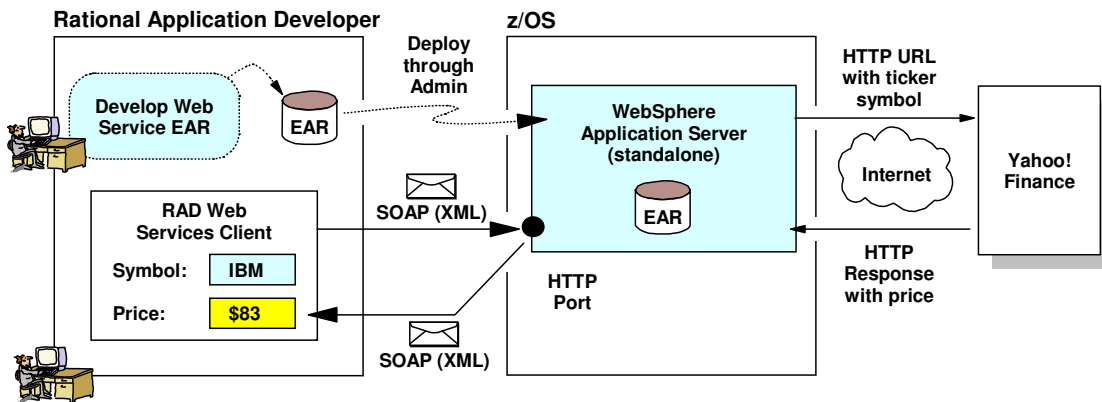
First lab ...

This is the architectural layout of the different Eclipse-based products from IBM and what they do.



## Our First Lab

The first lab will have us using the J2EE capabilities of RAD to build a very simple web service.

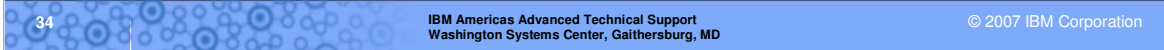



- Our client will be the integrated test client of RAD
- Our service will run on WebSphere for z/OS
- Our “data source” is Yahoo! Finance
- All will share same WebSphere instance  
Pay attention to your unique team number

This will introduce us to RAD and simple concepts of web services

In this first lab what we're going to do is create a very simple Web Service that'll run in WebSphere Application Server. This Web Service will accept a SOAP request from the test client that runs inside of RAD, and that Web Service will then turn and fetch a stock price based on the ticker symbol you supply. That means the Web Service is going to turn and drive Yahoo! Finance using the standard URL method (in other words, not a Web Service). But the interaction between your client running in RAD and the Web Service running in WebSphere will very definitely be the standard “SOAP over HTTP” methodology.

For this lab we're all going to be sharing the same WebSphere Application Server instance. That means you'll each deploy your developed EAR into the same running copy of WebSphere. That's okay, but it means we'll need to be careful and insure uniqueness. The lab instructions provide that uniqueness.



IBM  
Web  
Services  
Support

Web Services and

# CICS TS 3.1 and 3.2

Web Services prior to CICS TS 3.1 generally not recommended

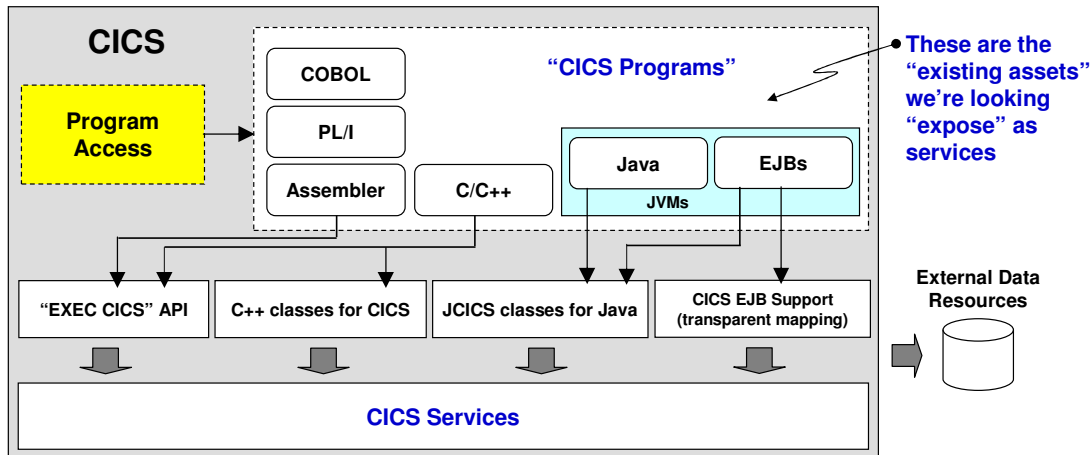
34

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

© 2007 IBM Corporation

## CICS is an Application Server

CICS is a system that hosts applications, and provides a rich set of “services” which the applications may make use of:



There are many ways to access programs running in CICS -- 3270 terminal, EXCI or EPI, RMI/IOP, MQ, HTTP. Our focus here is going to be accessing via Web Services.

Several ways to expose CICS resources as Web Service ...

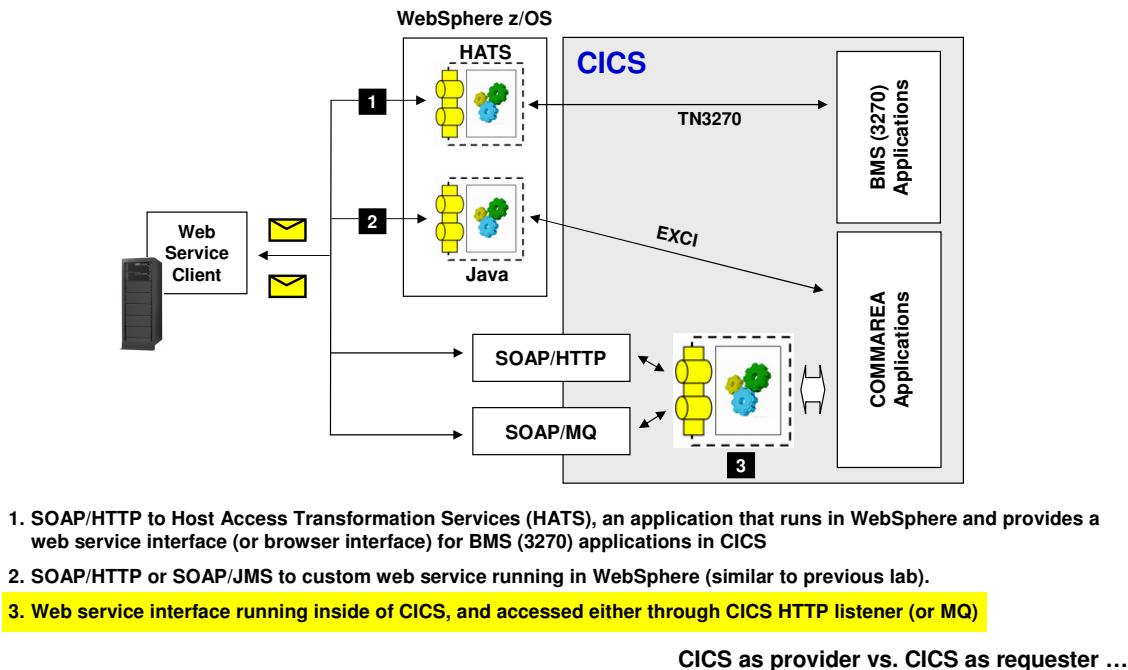
The first thing to put on the table is that CICS is in many ways an application server -- it hosts the running of applications within it, and it provides a rich set of “services” that applications can make use of rather than having to write their own. It provides a way to access backend data such as DB2 or IMS.

Programs written to run in CICS may be written in the traditional languages, COBOL, PL/I or Assembler; C or C++; as well as Java, in either “POJO” format (Plain Old Java Object) or EJB (Enterprise Java Bean). Each of these can gain access to the services provided by CICS, but through different means based on the programming language.

When it comes to accessing programs hosted by CICS, there's a myriad of ways. Later on we'll have a picture that shows the options available to you. In this workshop our interest is showing how to access Web Services ... and specifically, Web Services that are exposing an existing CICS application as a Web Service.

## Different Architectural Approaches to CICS and Web Services

This represents a few, and provides an illustration of what we'll focus on:



36

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

© 2007 IBM Corporation

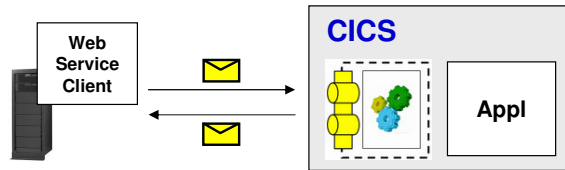
Suppose you have some existing applications that are in CICS, and you want to expose them as a Web Service. There's a couple of different ways you can do that:

1. If your application is a BMS 3270 application, you can expose it to the web using an IBM product called HATS (Host Access Transformation Services). HATS provides a way to make the 3270 application available either to browsers (it provides a terminal-to-HTML mapping facility), or to Web Service clients. HATS runs in WebSphere Application Server
2. You can develop your own Web Services application that runs in WebSphere Application Server. The implementation of the service behind the Web Service interface could make use of the JCA connector to access CICS via EXCI. This is accessible as SOAP over HTTP or SOAP over JMS.
3. You can develop a Web Service that runs inside of CICS itself. This Web Service would be accessible via either SOAP over HTTP or SOAP over JMS. The Web Service would then turn and invoke the CICS application using function within CICS designed to do this. Note: this is what we're going to do in lab.

**Note:** please do not think this picture is the definitive list of all possible ways to access existing CICS applications using Web Services. There are other variations on this theme that provide slightly different approaches. This picture represents three key ones we wished to highlight at this point in time.

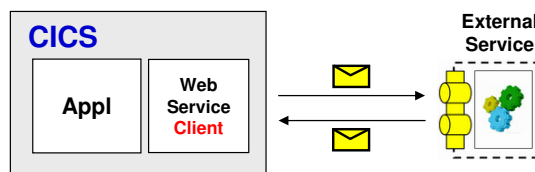
## Provider vs. Requester -- CICS Can Do Both

We typically consider CICS as a *provider* of web services:



The case where existing (or new) CICS applications are exposed as reusable services.

But it can also be a consumer (or requester) of web services:



This web service could be anywhere accessible to CICS -- inside your company or outside

**We'll focus on the top one for the most part. The concepts you'll see are mostly applicable to both environments. See "CICS Web Services Guide" (SC34-6458) for more.**

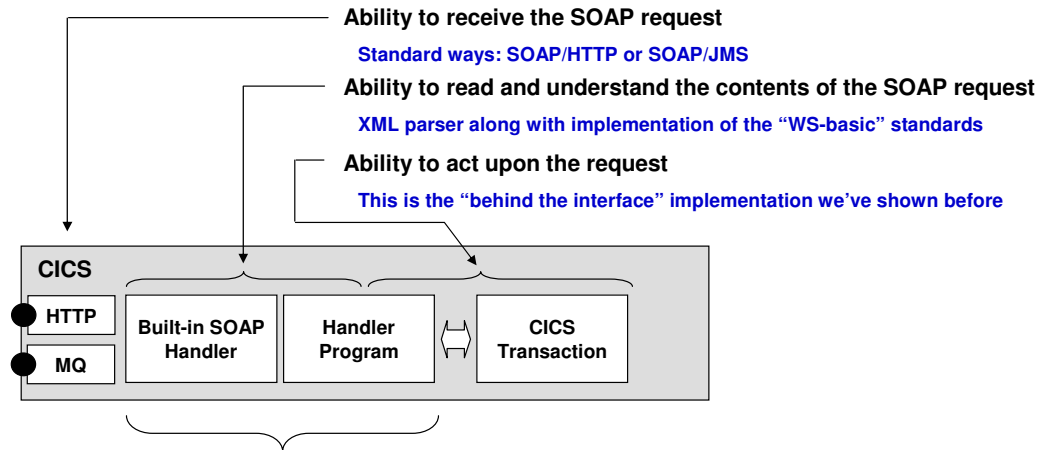
**CICS as a Web Service Provider ...**

We need at this point to make sure a key concept is communicated. And it is this: CICS can be a Web Services provider (the typical scenario, and the one we'll focus on most in this workshop), and it can be a Web Services requester. That is, a program in CICS can call out and invoke a Web Service outside of CICS. This would be done, for example, if the CICS program needed some piece of data that resides in a service outside of CICS.

As we said, we're going to focus on the top scenario. Our focus is going to be exposing existing CICS resources as Web Services. But we wish you to know that CICS applications can call out to a Web Service as well.

## CICS as a Web Services Provider

Recall the three basic requirements of being a Web Services provider:



This is defined within something called a “Pipeline,” which is a structure within CICS that invokes your customized handler program(s).

This is what does the mapping of XML to application data structure and invokes the CICS transaction.

Understanding the Pipeline ...

38

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

© 2007 IBM Corporation

Here we provide a look at the basic way in which CICS handles a web service request. Go back to our statement about the three basic requirements of being a web services provider: the ability to receive the SOAP request; the ability to read and understand the SOAP request; and the ability to act upon the request.

**Receiving the request** -- CICS has a built-in HTTP listener, so it can take in SOAP over HTTP. It also has the ability to receive SOAP that was sent over JMS. This would be done over MQ via the CICS MQ Bridge. In either case, the SOAP request is received by the CICS server.

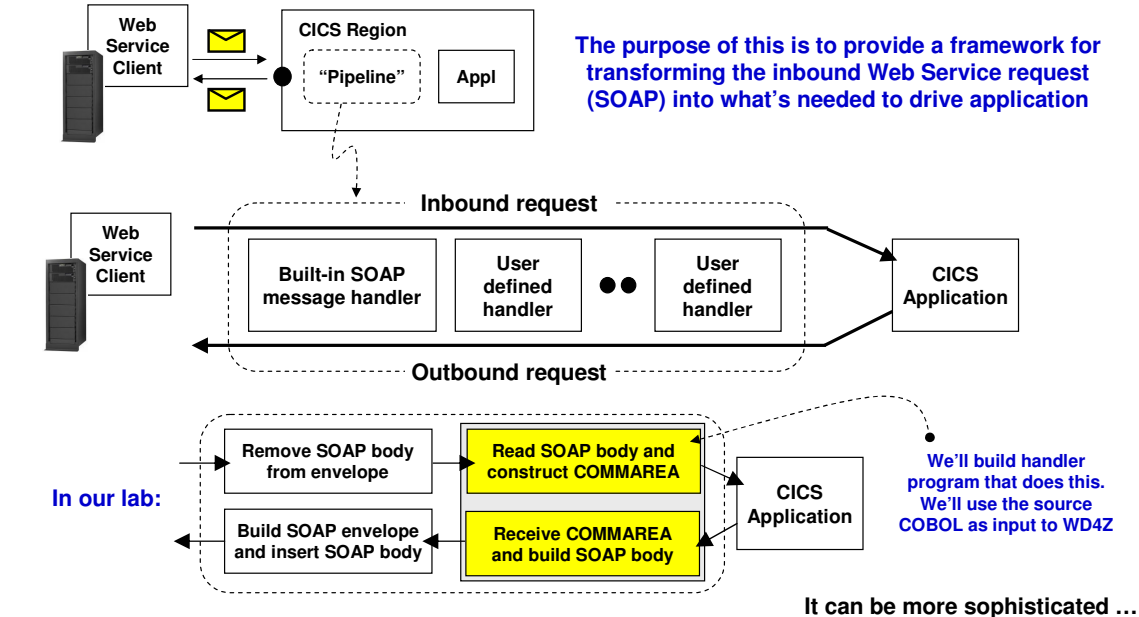
**Ability to read and understand the SOAP request** -- this ability is split across two things: a built-in SOAP handler function of CICS (which understands the SOAP envelope components), and your custom handler program that is written to understand the request held within the envelope. The request portion of the SOAP envelope will have XML tags specific to the CICS tran in the background being invoked. Your custom handler is what understands those tags and formats up the native CICS transaction invocation.

**Ability to act upon the request** -- this is done by a combination of your custom handler program and the existing CICS transaction. Your custom handler turns the XML request into COMMAREA and drives the CICS tran. This is all work done “behind” the web services interface.

This notion of a custom handler relates to something called a “Pipeline,” which is a structure within CICS that is invoked when a request is received. The Pipeline contains information about the handlers to invoke, which in turn invoke the CICS Transaction.

## The CICS Web Service “Pipeline”

The “pipeline” is a definition within CICS that defines what “handlers” will be invoked upon receipt of the request:



39

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

© 2007 IBM Corporation

A “pipeline” is a kind of message handling thing implemented inside of CICS. The purpose of the pipeline is to provide a framework in which you can define how you want the received SOAP message processed prior to invoking the CICS application. The upper-most picture in the chart shows the position of the pipeline relative to the HTTP port and the CICS application.

The middle picture shows some detail -- the pipeline has within it a built-in SOAP message handler, which means it's capable of reading into the SOAP envelope and stripping out the body of the message. Once the body is extracted, the message is handed to a user-defined program, or “handler.” There can be multiple handlers defined to the pipeline, and if that's the case then the pipeline processes them in sequence. In the case of this picture, the processing is done left to right as the message moves from the SOAP handler towards the CICS application. When the response comes back, the message will flow through the pipeline again, only this time in reverse.

But you don't need to have more than one handler ... you may wish to keep things simple and have only one. For example, in our lab we're going to define only one message handler. Its purpose will be to take the received XML and construct a COMMAREA that will be passed to the application. Upon return, the CICS application will return data in the COMMAREA. The handler, operating in reverse, will take the COMMAREA and form up the response XML. The built-in SOAP handler out front will either strip the SOAP headers off (inbound) or build them back up (outbound).

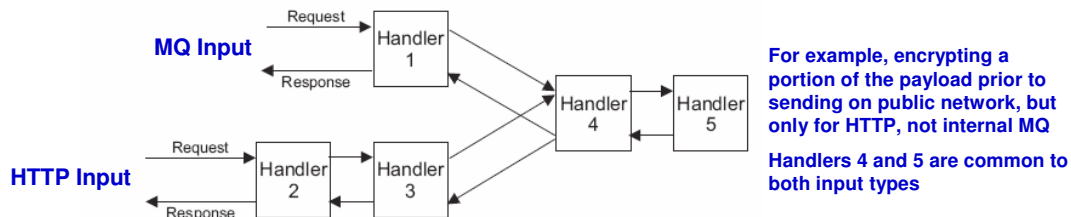
This processing can be quite sophisticated. The next chart illustrates this.

## Some More Sophisticated Examples of Pipeline Processing

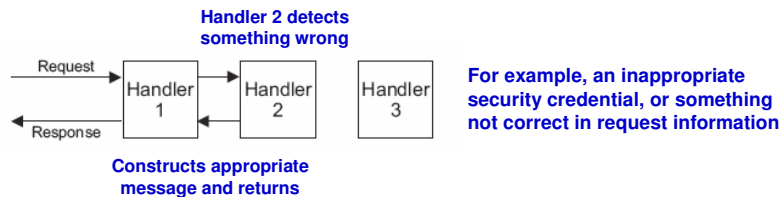
Just to give you a flavor that it's not just a simple linear processor, here are two scenarios that point out some of the flexibility of this thing.

(Both examples come from "CICS Web Services Guide," Chapter 4)

### Initial Handling Separated by Input Type



### Interrupting the Flow and Processing Error Message



Tools to develop these ...

The processing can be more than just simple linear processing. The pictures above came from the CICS Web Services Guide, chapter 4, which outlined a couple of more sophisticated examples:

- In the first example, two different input methods are defined -- HTTP and MQ. Depending on which channel the message came in on, either handler 1 or handlers 2 and 3 would be processed. Then the two flows converge onto handler 4 and 5. In reverse it would go: 5, then 4, then either 1 or 3 and 2, depending on whether the output was HTTP or MQ.
- In the second example we show a linear flow again, but offer a twist: error handling. Here the flow comes in on handler 1, then proceeds to handler 2. But handler 2 detects something wrong. So it processes an error message and turns the flow back around, sending it back to 1 and then back to the requester. Handler 3 never sees the message, and the CICS application definitely never processes anything.

So the basic concept here is that the pipeline defines a framework in which your custom message handler programs are processed when they are received. The next chart discusses how these things are developed.



## CICS Web Services Development Tools

We'll discuss two primary tools used to develop CICS web services:

### CICS Web Service Assistant

Consists of a set of JCL batch utilities that generate program components

- DFHLS2WS – Transforms a language structure into a Web Service Binding File and a Web Service Description (WSDL). Use this to put a web service front end on an existing application.
- DFHWS2LS - Generates a Web Service binding file from a Web Service description (WSDL). This utility also generates a language structure that you can use in your application programs. Use this to create a new CICS application based on a WSDL, or to enable CICS to be a web service requester

### WebSphere Developer for zSeries (WD4Z)

An Eclipse-based tool for zSeries development (not *just* web services), it does what CICS Web Service Assistant does with additional flexibility and capabilities.

**Generally speaking, WD4Z is the more powerful alternative.  
CICS Web Service Assistant is useful for more basic web  
services enablement and when Eclipse expertise is lacking**

**We'll use WD4Z in the lab**

Lab exercise artifacts ...

There are two basic approaches available to you when it comes to developing the Web Services handler programs:

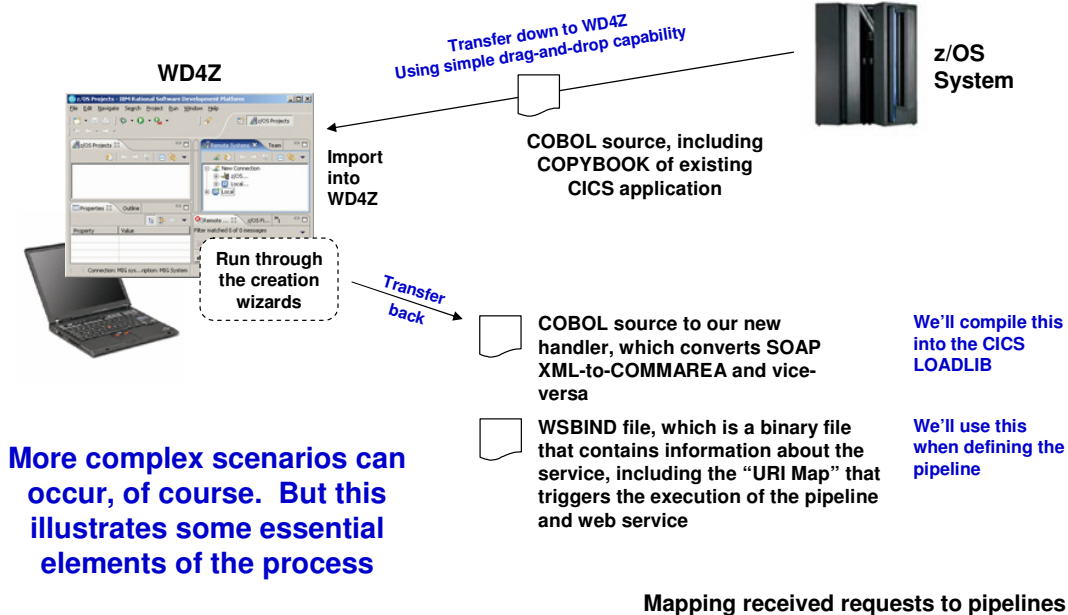
- One is to use the "CICS Web Service Assistant," which is a set of batch JCL jobs that render handler programs without requiring a graphical development environment like Eclipse. Two batch jobs are available -- DFHLS2WS and DFHWS2LS. The difference between them is what's used as input to the process.
- WebSphere Developer for zSeries, which is an Eclipse-based tool for lots of different z/OS-related development, not just Web Services. You can consider WD4Z a superset of what Cics Web Service Assistant offers.

If you were looking for the most powerful of the two, WD4Z would be it. CICS Web Service Assistant is made available when skills in using an Eclipse-based environment like WD4Z are lacking, or if it's been determined that CICS Web Service Assistant provides all that's required for your development needs.

We'll use WD4Z in the lab. And what we'll create is discussed next.

## What's Produced by WD4Z for Our Web Service Lab

We'll use the lab we'll do in a short while to illustrate the process of creating and defining a Web Service to CICS



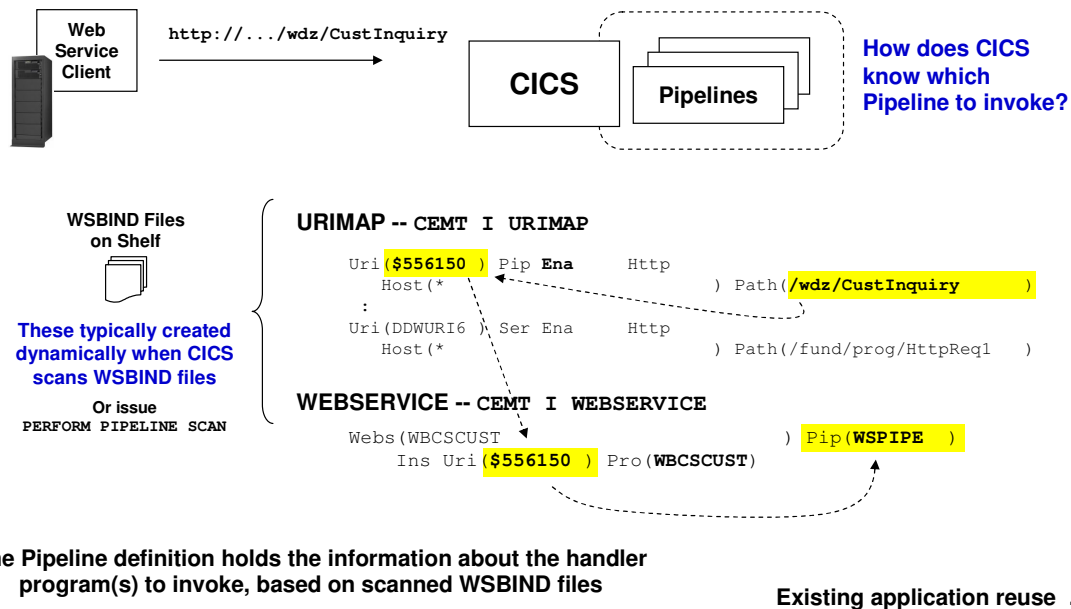
Here's a brief illustration of what we're going to do in lab. It'll help demonstrate what's produced by the tooling to create the Web Service handler in the pipeline.

- We're going to bring down to the WD4Z tool the COBOL source to the WBCSCUST application that's currently up in our CICS region. By doing that we're going to give WD4Z visibility to the language structure of the application -- the input and output fields.
- We'll then run through the Web Service creation wizards to create the deployment artifacts. This will consist of two files:
  1. A source COBOL file that will be, when compiled into the CICS load libraries, the handler program in the pipeline. This COBOL is not the same thing as the COBOL you downloaded to start the process. This is COBOL that implements the handler that turns XML to COMMAREA, and COMMAREA back into XML.
  2. A WSBIND file, which is a binary-format file that contains information about this handler, including the "URI" associated with the pipeline and the handler program module(s) that are part of the pipeline definition.

You'll transfer both back up to the z/OS system using the transfer capabilities of WD4Z..

## Mapping of URIs to Pipeline Handlers

CICS is capable of supporting many separate pipelines. How does it relate a received web service request to a given pipeline? With URIMAP information:

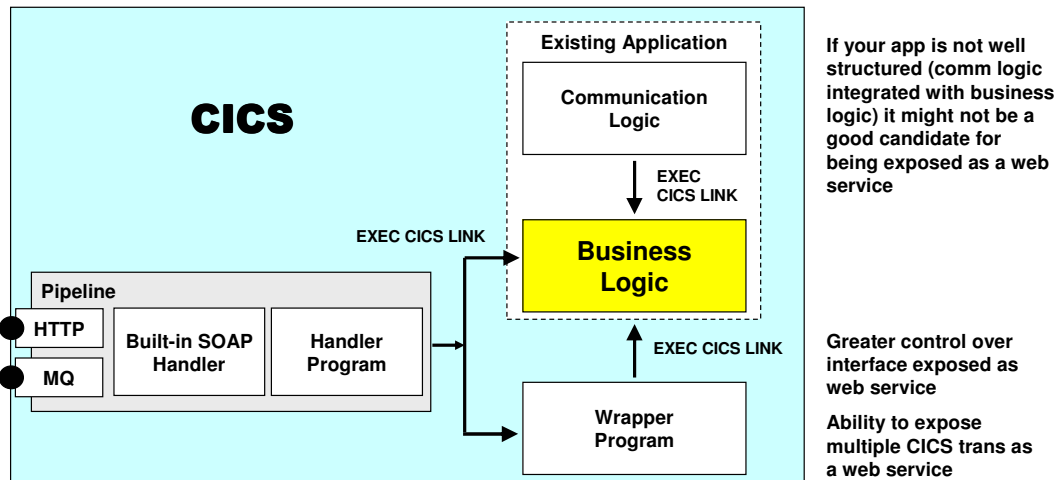


When multiple pipelines are defined, and each has its own handler program included, how does CICS know how to relate a received web service request to a given pipeline so the proper handler can be invoked? It's done through CICS' knowledge of the "URIMAP" information contained in each of the WSBIND files. CICS scans those at startup (and periodically thereafter) and reads into an internal table the relationship of the "Path" to the pipeline.

Consider the example above. CICS has scanned the WSBIND files defined to it, and it creates a list of URIMAP definitions. It also creates a series of WEBSERVICE definitions. The URIMAP has the "Path," which is like the "context root" in WebSphere -- it is the portion of the URL that follows the host and port. The URIMAP definition has a CICS-created number that ties the definition to a WEBSERVICE definition. That definition has the PIPELINE information. The pipeline is then invoked.

## Reusing Existing Application Resources

You can do this either directly or indirectly using a wrapper program:



This is listed as a “best practice” in the following redbook:



Application Development for CICS Web Services  
SG24-7126

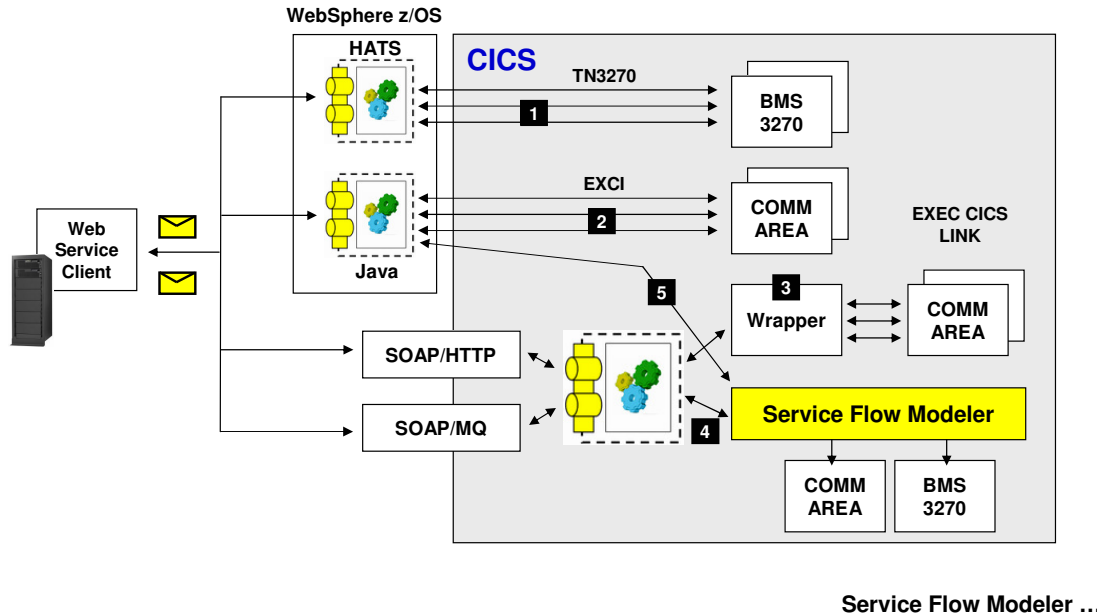
Application Aggregation ...

If you plan to expose existing applications as web services, you can do that in two different ways: you can either invoke from the Pipeline handler directly, or you can provide a wrapper program which gets invoked by the Pipeline handler and the wrapper turns and invokes the existing application. The advantage of the latter is that it gives you additional flexibility in how the existing application is exposed -- your wrapper can determine what data fields are used and the format of those fields. The redbook SG24-7126 cites this as a “best practice,” but allows for the direct invocation as well.

Both are predicated on the idea that your existing application is well structured ... that the communication and presentation logic is properly separated from the business logic. If that's not the case, then the existing application may not be a good candidate for exposure as a web service using this methodology ... perhaps HATS might be a better choice if it's a 3270 application and the presentation logic is buried with the business logic.

## CICS Application Aggregation

Imagine you have several CICS transactions you wish to “combine” and expose as a single web service. How can that be done?



Service Flow Modeler ...

45

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

© 2007 IBM Corporation

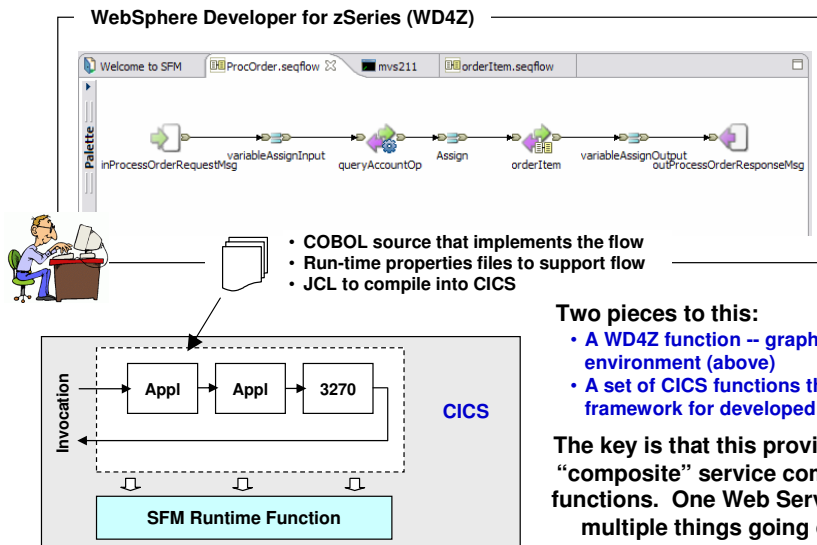
Suppose you want to expose multiple CICS transactions as web services, but ideally you'd like to expose them as a single web service, rather than separate individual ones. How can that be done? In several ways:

- 1** - HATS can be used to string together multiple 3270 screens to form a single web service. The limitation here is that HATS can only work with 3270 screens, not COMMAREA applications. But if your application is 3270 based, HATS is a very good solution.
- 2** - You could write a web service that runs in WebSphere that turns and drives the various CICS trans in the order you desire. In this scenario WebSphere can act as the transaction manager for the multi-part web service, rolling back the updates if some piece of the overall flow fails.
- 3** - You can write a CICS wrapper program that then turns and drives the other CICS trans in the order you desire. Here CICS itself would be the transaction manager. Access to the wrapper program would be through the CICS web service interface support of CICS. (Or you could drive the CICS wrapper program from WebSphere.)
- 4** - You can use Service Flow Modeler, which we explore next. Without going into too much detail, suffice to say SFM handles both COMMAREA and 3270 applications. It can be invoked from the CICS web service interface with a Pipeline and handler program.
- 5** - You can also drive a SFM flow as a normal CICS tran from WebSphere.

Let's look a little closer at Service Flow Modeler ...

## Service Flow Modeler

Service Flow Modeler is a function of CICS V3.1 and above (with support in WebSphere Developer for zSeries) to construct an aggregated application flow



Similar in concept to what you'll see in the WebSphere Message Broker Toolkit

Two pieces to this:

- A WD4Z function -- graphical development environment (above)
- A set of CICS functions that provides supporting framework for developed flow

The key is that this provides a way to construct a "composite" service comprised of multiple CICS functions. One Web Service invocation results in multiple things going on behind the scenes.

Can invoke with either Pipeline handler, EXCI or EXEC CICS LINK

Summary ...

This feature consists of two pieces -- a development side feature that is part of WD4Z, and a set of runtime functions that provides the service up in CICS. The development side of this consists of a graphical environment where you draw out the flow you desire, assigning properties to each element in the flow, which then results in deployable code. The concept is similar to what you're going to see in WebSphere Message Broker Toolkit later.

What SFM provides is a way to expose a single Web Service that is, in reality, a composite application behind the scenes. And the composition of the application is based on how you modeled it in the development environment.

## CICS TS 3.2 Web Services Enhancements

<http://www.ibm.com/software/http/cics/tserver/v32/appcon/>

### New in CICS TS 3.2 with respect to Web Services:

- WSDL 2.0
- MTOM and XOP
- WS-I Basic Profile 1.1 and Simple SOAP Binding Profile 1.0
- WS-Trust specification in WS-Security

### In addition to what was in CICS TS 3.1 and carried forward:

- HTTP 1.0 and 1.1
- SOAP 1.1 and 1.2
- WS-Coordination
- WS-AtomicTransaction
- WS-Security

**As well as other enhancements not directly related to Web Services.**

**Check out the URL for more information.**

**Summary ...**

CICS TS 3.2 recently became available, and with it came some enhancements to the web services support -- these are enhancements in addition to the support that was already there in TS 3.1. Also, TS 3.2 brings other enhancements to the table as well ... so check out the URL at the top of the chart for a better review of the merits of this new release of CICS.

## Reference Information

For more complete information about CICS and CICS Web Services:



CICS Transaction Server for z/OS



### CICS Web Services Guide

Version 3 Release 1

**SC34-6458**

<http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zswpdf/cicstszos31.html>



**The complete z/OS CICS TS 3.1 library in PDF form, including the SC34-6458 book**

**IMS ...**

The CICS Web Services Guide is the definitive guide to Web Services on CICS. You can access that book, and the whole CICS library, through the URL shown on the chart. Now onto IMS.





**Web Services and**  
**IMS**

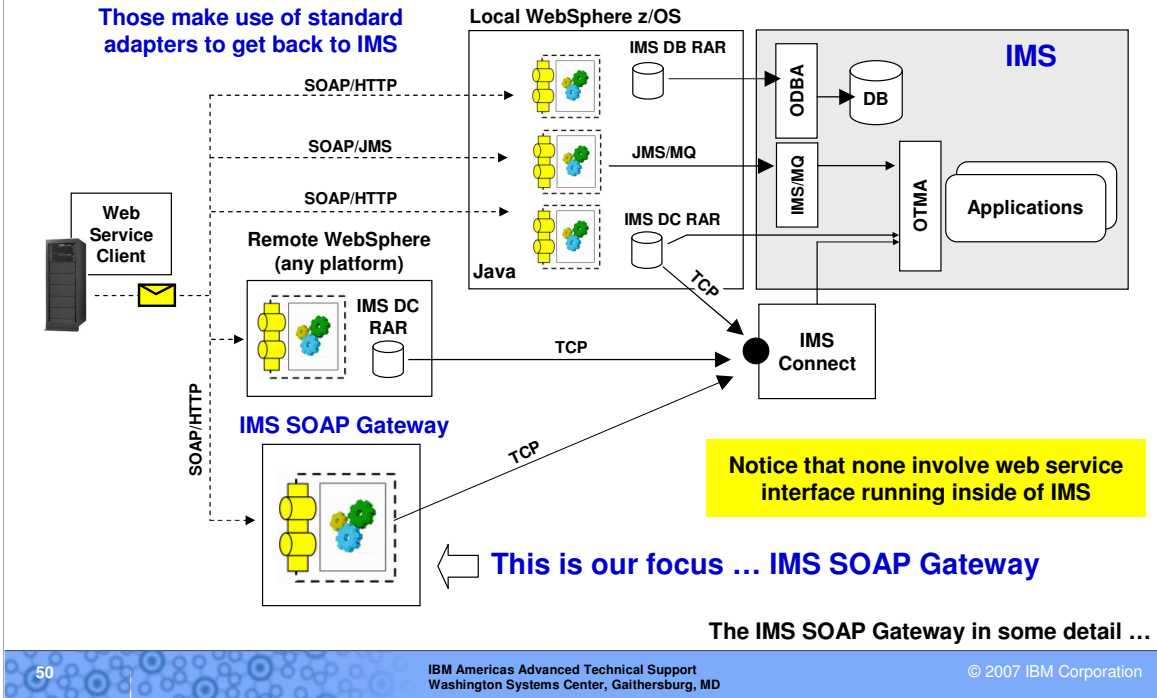
49

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

© 2007 IBM Corporation

## A Few Ways to Get Web Service Request into IMS

Several of the ways involve front-ending IMS with a web service in WebSphere



There are many ways one can access IMS data as a Web Service, but none of them involve accessing IMS *directly* with HTTP. In all cases there's something between the client and IMS. This picture illustrates five different scenarios:

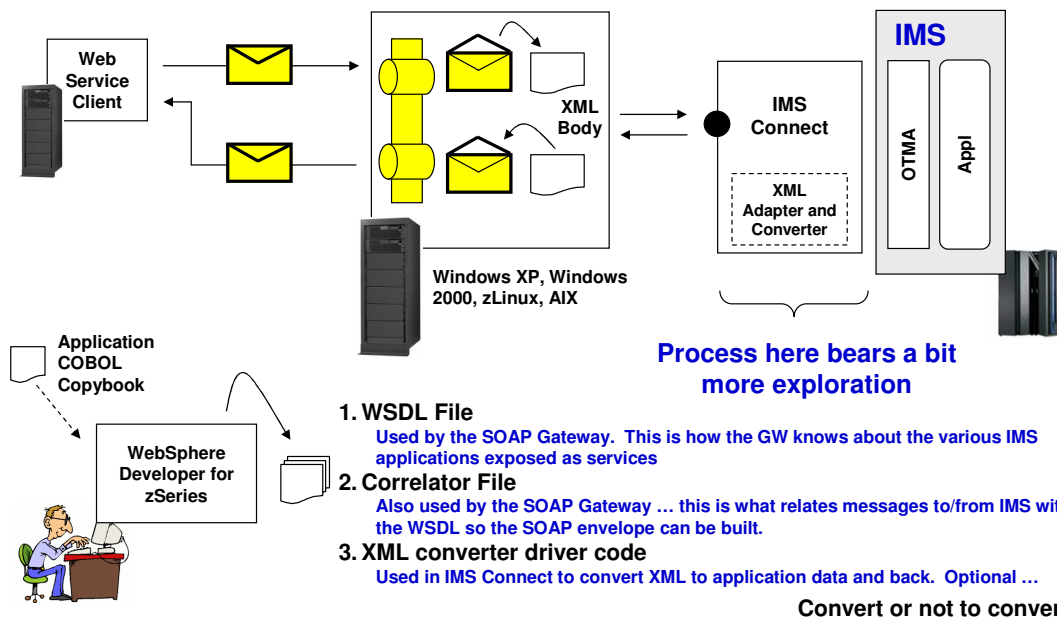
- The top-most box illustrates WebSphere Application Server for z/OS running on the same MVS image as the IMS region. We learned earlier that WebSphere itself is an excellent host of a Web Service interface, and from there a long list of data connection options is available. Two of those are data connectors to IMS. One is called "IMS DB RAR," which is really a JDBC implementation to access the IMS database. This can only be used when the application using the files in the RAR is local to the IMS region. The second is something called "IMS DC RAR," which is a way to access IMS applications using OTMA. If the application is "local" (same MVS image) then it can directly access without need of IMS Connect. But even if it is local you may opt to connect via TCP through the IMS Connect function. You can also access your Web Service through JMS, then go MQ back to the IMS/MQ bridge and into IMS.
- If WebSphere is "remote" (not on the same MVS image) -- either WebSphere z/OS or WebSphere on any platform -- then IMS connect is required as the communication from the application to IMS will be over TCP. (**Note:** not shown is flowing JMS/MQ to IMS/MQ Bridge. It's possible, but the picture was getting too cluttered.)
- The final means is through something called the "IMS SOAP Gateway," which is function implemented on a distributed box (it is not available for z/OS). The SOAP gateway is capable of handling the SOAP envelope and communicating with IMS Connect to execute the service.

The focus of this discussion will be on the IMS SOAP Gateway. The other methods are really more related to WebSphere and traditional data connector access.

**Note:** Why would someone choose the IMS SOAP Gateway over WebSphere? One reason -- if there's not at present any built-up skills in WebSphere. If WebSphere already exists then going that route would be relatively easy. But building up WebSphere skills just to handle SOAP might be more effort than one desires.

## IMS SOAP Gateway

Runs on a non-z/OS platform, it acts as a front-end SOAP handler. It provides the web services interface and forwards XML body back to IMS.



51

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

© 2007 IBM Corporation

The SOAP Gateway function is something that runs on a non-z/OS platform (Windows XP or 2000, AIX or z/Linux). Its role is to act as an HTTP and SOAP handler out front of IMS. It receives the SOAP request, strips the request body out of the SOAP envelope and passes that back to IMS Connect for processing. The message is still in XML format at this time, so more handling is required.

IMS Connect receives the message and then prepares to pass it back to IMS (via OTMA). Here a decision needs to be made -- should the XML be converted by IMS connect? Or passed back to the application for conversion? If converted by the IMS Connect function then the XML Adapter and Converter function needs to be employed. If passed back to the application, then the application needs to be prepared to do XML conversion. We'll look at this process in a bit more detail next.

To make this work you need some key components that are produced by WebSphere Developer for z/Series:

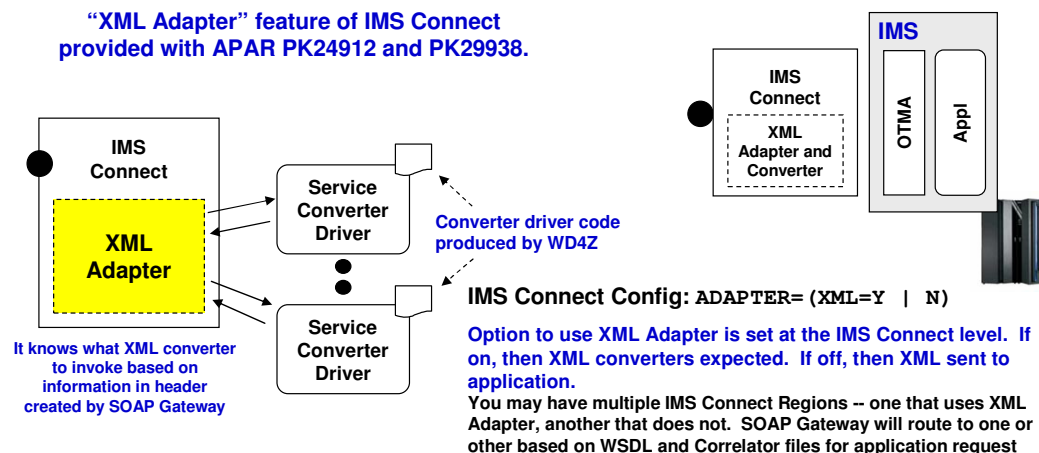
- The WSDL file -- this is used by the SOAP Gateway to understand this exposed web service. Any single instance of the SOAP Gateway can handle multiple services in the backend, so having all the WSDL files allows the SOAP gateway to understand what services it supports.
- The Correlator File -- also used by the SOAP Gateway, this allows the SOAP Gateway to relate messages it sees coming back from IMS to the specific web service it came from.
- The XML converter driver code -- if used (it is optional), it is what gets invoked by IMS Connect to convert the data contained in XML to the format that can be handled by IMS via OTMA. The same happens in reverse -- when the data comes back from IMS, the converter driver re-formats the XML response and passes it back to the SOAP Gateway.

As mentioned, the XML converter function is optional. Let's explore that component a bit more closely.

## XML Adapter and the XML Converter Driver

In all cases the Gateway is going to pass XML back to IMS Connect. Two choices: XML handled by converter, or XML handled by the application.

“XML Adapter” feature of IMS Connect provided with APAR PK24912 and PK29938.



So you have two options:

- XML Adapter/Converter -- *no changes to your application*
- XML handled by your application -- *may require changes to application*

The big picture ...

52

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

© 2007 IBM Corporation

The SOAP Gateway will, in all cases, pass XML back to IMS Connect. The SOAP envelope will be stripped out, but the XML body will be passed back. That XML needs to be handled by someone, and the choice is yours -- have it handled by a new feature called the XML Adapter and Converter, or have it handled by your application.

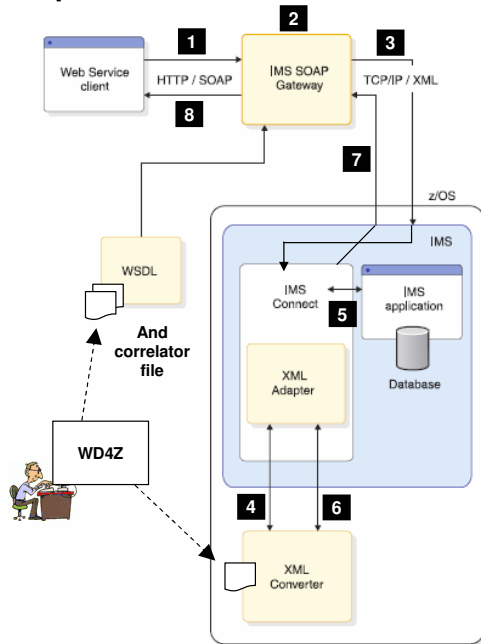
The choice to enable or not enable the XML converter is done at the IMS Connect level. If you turn it on (ADAPTER= (XML=Y)) then IMS Connect expects to invoke some converter driver code. If not enabled, then IMS Connect expects to pass XML back to your application.

Let's say you have it enabled. Then IMS Connect is going to look to invoke XML converter code. That code is something produced by WD4Z and is compiled into load libraries accessible by IMS Connect. IMS Connect knows *which* converter driver to invoke based on information in the header created by the SOAP Gateway, which is then passed back to IMS Connect. IMS Connect reads the converter driver information and then passes the XML to that converter. When the conversion is done, the data is then passed back to your IMS application through OTMA. On the way back, the response is again passed through the XML converter driver, which formats up the XML and passes it back to the SOAP Gateway, which wrappers it in a SOAP envelope and passes it back to the client.

Why would you use one vs. the other? It really has to do with the impact to the application in the background. If you employ the XML converter you don't need to touch your application -- it remains unchanged. But if you don't use the XML converter, then your application is going to be expected to handle the XML. So you'd have to make sure your application is prepared to do that.

## The Big Picture Flow (from SOAP Gateway PDF ... info next chart)

To put it all into context ...



1. The Web service client application sends a SOAP message to IMS SOAP Gateway which contains input to the IMS application in an XML format.
2. IMS SOAP Gateway processes the SOAP header (XML) and retrieves the appropriate correlation and connection information for the input request.  
*Artifacts produced by WD4Z and placed in GW directory*
3. IMS SOAP Gateway sends the input XML data to IMS Connect using TCP/IP after *adding the appropriate IMS Connect header. Header is what allows IMS Connect to know what to do with XML coming in.*
4. IMS Connect calls the XML Adapter which in turn calls the XML Converter to perform the XML to IMS application format transformation.  
*If adapter function enabled. Steps 4, 6 skipped if off.*
5. IMS Connect then sends the message for further processing. From this point on, the processing is the same as a normal transaction flow. The transaction gets executed and the output is queued.
6. IMS Connect calls the XML Adapter in order to perform the transformation of the IMS application format back to XML.
7. IMS Connect sends the output XML message back to IMS SOAP Gateway using TCP/IP.
8. IMS SOAP Gateway wraps a SOAP header on the output message and sends it back to the client application.

Reference material ...

So here's the big picture, as provided in the SOAP Gateway PDF we'll reference on the next chart. The text on the chart explains everything in fair detail. You can go to the PDF for more detailed information.

## Reference Information

<http://www.ibm.com/software/data/ims/soap/>

**IMS Family**  
Information Management System

**Technology**

**IMS Soap Gateway**

The IMS SOAP Gateway is a Web service solution that integrates IMS assets in a Service-Oriented Architecture (SOA) environment.

IMS SOAP Gateway enables IMS applications to inter-operate outside of the IMS environment through SOAP to provide and request services independent of platform, environment, application language, or programming model. It assists you to enable your IMS application to become a Web service. Different types of client applications, such as Microsoft .NET, Java, and third-party applications, can submit SOAP requests into IMS to drive the business logic of the COBOL applications.

The IMS SOAP Gateway assists an organization in the following areas:

- Enterprise modernization
- Application development
- Business-to-Business (B2B) integration
- Service-oriented architecture (SOA) implementation

IMS SOAP Gateway Version 9.2.1 is now available at no additional charge to all IMS Version 9 customers.

**Information Management software**

**Download**

→ IMS SOAP Gateway Version 9.2.1 is now available

**Documentation**

→ IMS SOAP Gateway Documentation

→ IMS SOAP Gateway Documentation in PDF format

→ Get Adobe® Reader®

**IBM**

**IMS SOAP Gateway Version 9.2**


**DB2 ...**

54

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

© 2007 IBM Corporation

We've covered only the highlights of the Web Services solution to IMS. To get more information about the SOAP Gateway you should go to the IMS documentation library, and specifically the PDF found there that covers the SOAP gateway. It has much more detailed information about configuration and setup, as well as security issues we've not covered here.



## Web Services and **DB2**

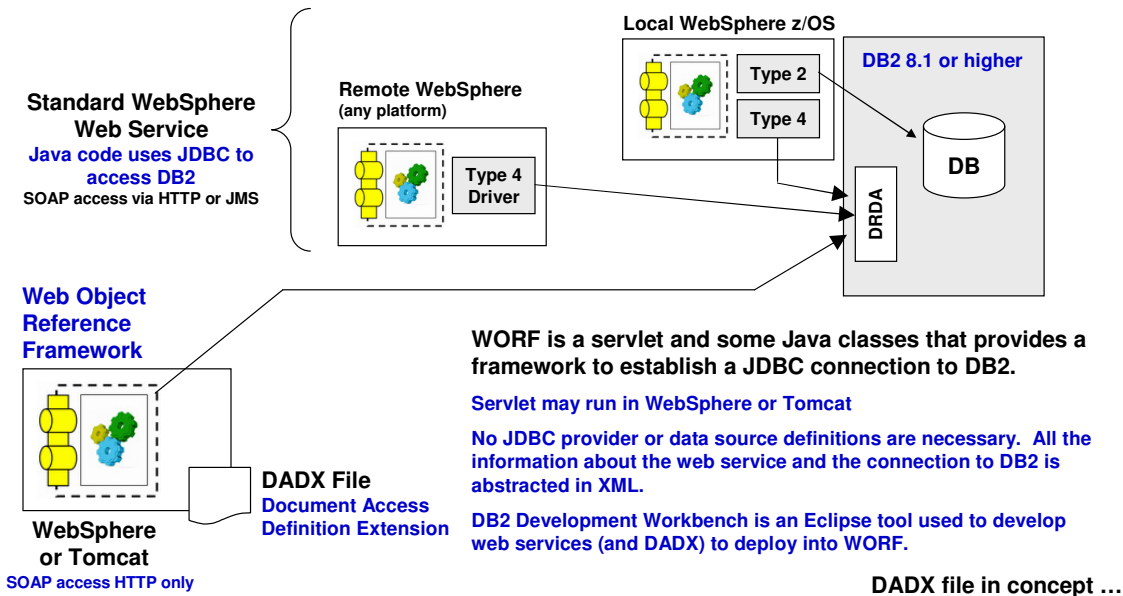
55

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

© 2007 IBM Corporation

## Web Service Interface for DB2

As before, we start the discussion by determining where the Web Service interface can be hosted. Unlike CICS and IMS (with SOAP GW), DB2 does not have an HTTP listener. So we have to put *something* out front.



56

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

© 2007 IBM Corporation

We see a common thread here ... the first question seems always to be: "Where can the SOAP interface be hosted?" We saw in CICS that it has its own HTTP (or MQ) listener. IMS provided it in the SOAP Gateway. WebSphere is a natural because it's made to be an HTTP, MQ and JMS listener. But what about DB2? It has no native HTTP listener. So something has to go out front to host the interface. For DB2, that something is called "WORF" -- Web Object Reference Framework.

WORF is really a specialized servlet that provides the SOAP interface and provides the connection back to DB2. Because it's a servlet it can run in either WebSphere or Tomcat. It uses something called a DADX file to provide the definitions of how the request is to be handled. We'll see more on that in a bit.

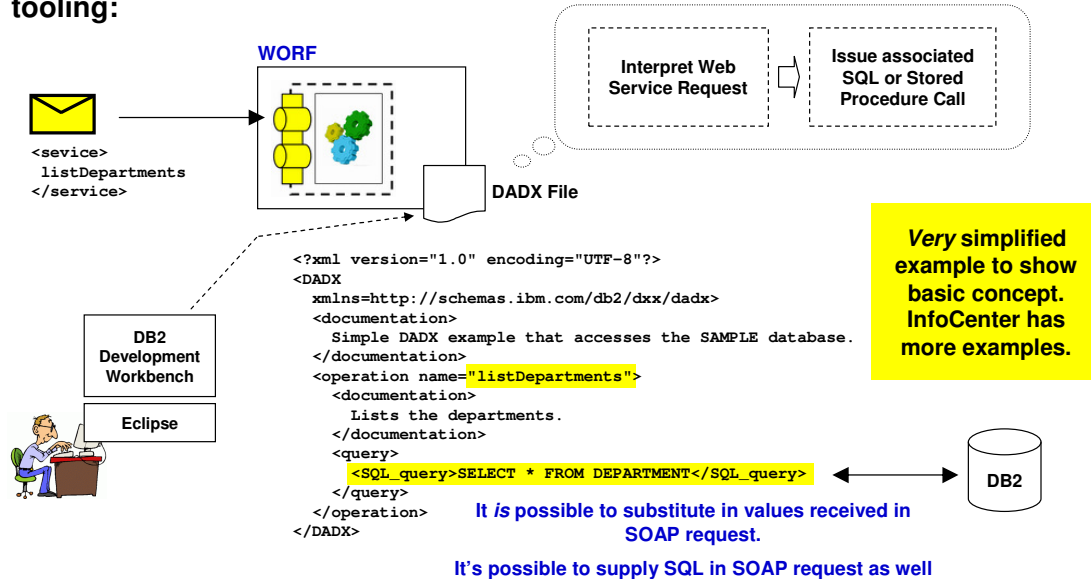
DB2 can act as a provider or requester. In either case, the access is via SOAP over HTTP. We'll see how it can be a requester in a little bit.

In addition to using WORF, you may also code up a Web Service and host it in WebSphere, either on the same z/OS system or a remote one. The development of the Web Service would be done with Rational Application Developer, and the connectivity to DB2 in the background would be through the normal JDBC support provided by WebSphere.



## DADX File in Concept

The DADX file is what defines how the Web Service request is converted to the appropriate SQL or Stored Procedure calls. DADX produced by Eclipse-based tooling:



Web Service *Provider* Operations via DADX file ...

57

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

© 2007 IBM Corporation

The DADX file acts as a kind of “bridge” between the SOAP request and the action against DB2 in the background. In that sense it’s conceptual role is to interpret the SOAP request and then perform the specified actions. Those actions may be SQL calls or Stored Procedure calls.

The chart shows a very simple example of a DADX file. Here the defined request is something called listDepartments, and the DADX file translates that request to an actual `SELECT` statement that will be issued against DB2. The DADX file can be generated using an Eclipse-based tool called the DB2 Development Workbench.

**Note:** it is possible to take parameters from the SOAP request and substitute it into the SQL, and it is possible to supply the entire SQL in the SOAP request. So you do have some flexibility here. Let’s explore what’s possible ...

## Web Service Provider Operations via DADX File

Here's a reference list of the operations that can be performed:

### SQL operations: non-dynamic

Non-dynamic operations are those that are predefined within the DADX file. There are three elements that make up the predefined SQL operations type:

- <query> - Queries the database ★ [Example from previous chart](#)
- <update> - Inserts into a database, deletes from a database, or updates a database
- <call> - Calls stored procedures that can return 0 or more result sets

### SQL operations: dynamic

Dynamic operations are those that are generated in a SOAP message with no predefined SQL operations.

The following elements are dynamic operations:

- <getTables> - Retrieves a description of available tables.
- <getColumns> - Retrieves a description of columns.
- <executeQuery> - Issues a single SQL statement.
- <executeUpdate> - Issues a single INSERT, UPDATE, DELETE.
- <executeCall> - Calls a single stored procedure.
- <execute> - Issues a single SQL statement.

Web Services requester  
supplies the SQL in its  
SOAP message

Which you use depends on how much control the Web Services requester has over what will be queried. Non-dynamic limits the control; dynamic opens it up.

Web Service Provider or Requester ...

We separate this chart into two categories: non-dynamic and dynamic SQL operations. The difference is where the SQL is specified -- in the DADX or passed into the DADX from the original SOAP request. The operations are shown on the chart.

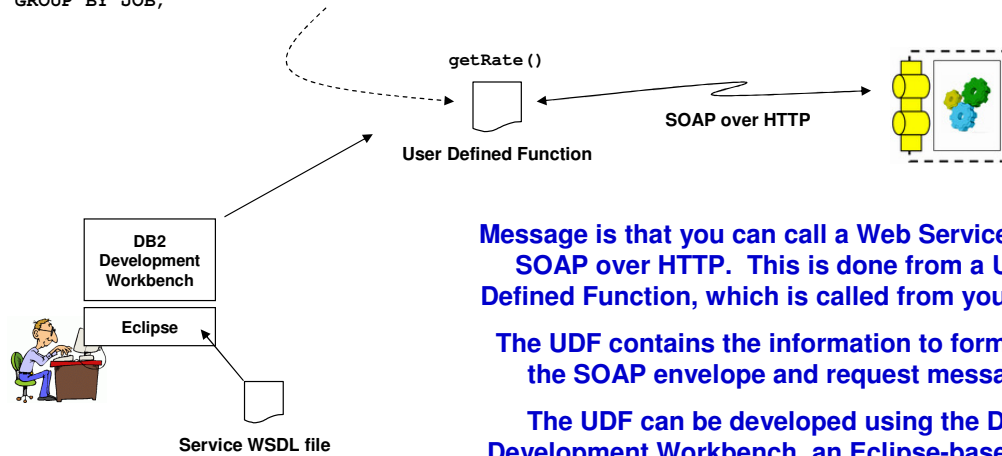
The difference seems to relate to how you want the service exposed. If you want the SQL hidden behind a service name, then you'd use the non-dynamic operations shown above. That would be like the example we showed on the previous chart. But if you want the WOF implementation to be a kind of "SQL proxy" for services, then you can use the dynamic SQL operations to do that.

The reference information we'll give in a moment will provide a pointer to more specific information on how this is done. For now we're staying pretty conceptual.

## DB2 as Web Service *Requester* (or “Consumer”)

We saw how DB2 can be a provider. But it can also serve as requester. The request to the service is done from a UDF. Your SQL calls the UDF.

```
SELECT;
AVERAGE (SALARY), getRate("Canada"), JOB FROM STAFF;
GROUP BY JOB;
```



**Message is that you can call a Web Service using SOAP over HTTP. This is done from a User Defined Function, which is called from your SQL.**

**The UDF contains the information to format up the SOAP envelope and request message**

**The UDF can be developed using the DB2 Development Workbench, an Eclipse-based tool**

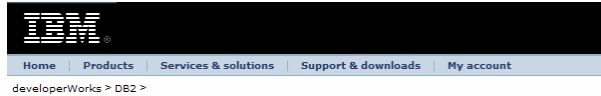
**Reference ...**

DB2 is capable of acting as a Web Services requester (or “consumer” if you prefer that term). The way this is done is to code up a User Defined Function (UDF) that executes the SOAP over HTTP request to the external service. The creation of the SOAP envelope and the processing of the XML is all done within the UDF. The UDF is created using a tool like DB2 Development Workbench. You would supply the WSDL file from the target server to the development tool so it knows how to invoke the service, and what to expect back.

The UDF you create is then called from SQL. The example above provides an illustration.

## Reference Information

<http://www.ibm.com/developerworks/db2/zones/webservices/worf/>



### Web Services Object Runtime Framework for DB2

#### Resources

- The [DB2 Infocenter](#) is an important resource for the latest information on all DB2 UDB topics.
- The [Web services section](#) of the DB2 Infocenter provides basic information about DB2 and Web services.
- The article [DB2 and Web services](#) (*IBM Systems Journal*, Vol. 41, No. 4, 2002) explains why Web services are important to DB2 and how DB2 has been extended to provide optimized support for Web services.
- The article [DB2 Web services for DB2 UDB practitioners](#) provides an in-depth guide to understanding how to use Web services with DB2.
- The white paper [Dynamic e-Business with DB2 and Web Services](#) explains what is meant by a Web service and introduces the architecture.
- The [Web services and UDDI](#) page explains the IBM Universal Description, Discovery, and Integration (UDDI) business registry, a database that facilitates the rapid participation of business in the e-commerce and business-to-business marketplaces.
- Find more resources for DB2 and Web Services on the [developerWorks DB2 and Web Services page](#).
- Purchase [Web services books at discounted prices](#) in the Web services section of the Developer Bookstore.

Links for  
documentation and  
download of Worf  
framework files  
(Free of charge)

Summary ...

The URL provided here provides an excellent summary of where other information on Worf and Web Services with DB2 can be found.



# Overall Summary

## Web Services and Support in Key IBM Products

- “Web Services” is an industry standard method of providing a message-based interface for program-to-program communications
- It is a key building block of SOA. SOA does not necessarily require the use of Web Services, but in most cases Web Services will be involved  
*At least to some degree.*
- IBM is heavily invested in Web Services, and is actively involved in the development/refinement of standards.

- We saw how key IBM systems support Web Services.

**WebSphere Application Server -- a natural host for Web Services; it has a rich application environment with connectors to access virtually any backend data store**

Expose CICS, IMS, DB2 or any other backend data as Web Service through WebSphere

**CICS -- built in Web Services Support allows direct invocation of Web Services through CICS HTTP listener.**

Excellent way to expose existing CICS applications as Web Services assets

**IMS -- supports Web Services through IMS SOAP Gateway.**

**DB2 -- supports Web Services through WOF**

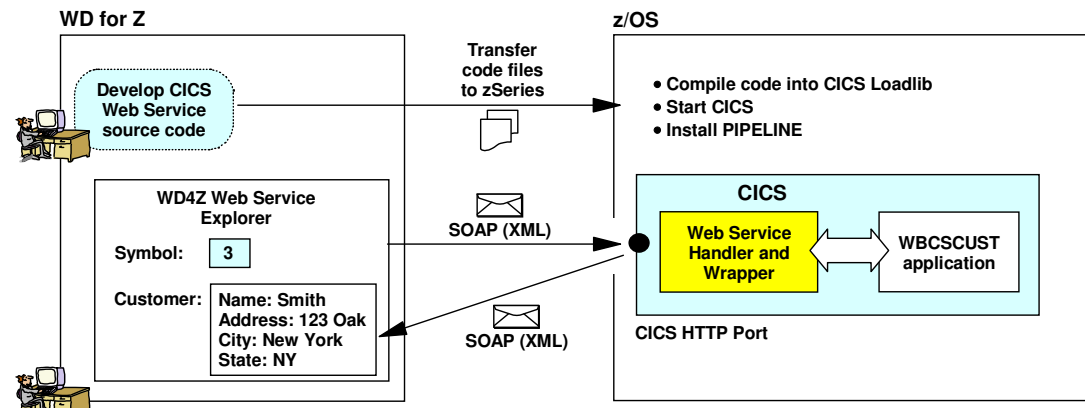
**In all cases except IMS, system can be Web Service *requester* as well.**

Though our focus was much more on being a Web Service provider.

Throughout this presentation we've explained what a “Web Service” is and how it operates. Web Services do not themselves make for SOA, but Web Services can make for a good foundation for the development of an overall SOA. IBM is a strong player in the Web Services game, and in fact we've shown how key IBM systems can be either a Web Services provider or requester.

## CICS Web Service Lab

The following picture illustrates at a high level what we'll be doing in the upcoming lab.



The lab you'll do involves using WD4Z to develop a Web Service that'll be deployed into CICS. We'll bring down the COBOL source for the WBCSCUST application and use that to define the Web Service interface and the pipeline handler program. We'll then transfer those artifacts up to z/OS and do a little green screen work -- compiling the handler into the CICS load libraries, starting CICS and then defining and installing the pipeline.

To test the new Web Service we'll use the client that's part of the WD4Z product. This test client will present a GUI form into which we'll provide an integer -- between 1 and 10 -- that will be passed to the CICS application. That integer will be packaged up into a SOAP envelope and passed up to CICS, where it'll invoke our pipeline handler chain (consisting of only one handler). The handler will strip the integer out of the XML, format up the COMMAREA, and drive the application. The result is going to be an address -- one associated with the integer we supply. That address is going to come back as a COMMAREA, so our handler will format it up as XML, wrapper it in a SOAP envelope and pass it back to WD4Z.

In so doing, you'll get a feel for how to develop a Web Service -- for CICS at any rate.

**End of Unit**