More than a Buzzword: Extending the Reach of Your MQ Messaging with Web 2.0

Simon Gormley (sgormley@uk.ibm.com)
IBM Hursley Park

August 10th, 2011
Session 9360

Introduction: Extending the Reach of Applications Using WMQ

- **New distributed programming models have become widely adopted**
  - SOA Web services
    - Promises greater re-use and interoperability through standards e.g. WS-*
    - Mixed environments: platforms, qualities of service and communications transports.
  - Programmable web
    - Ad-hoc web services – simple, lightweight, pragmatic e.g. REST

- **IBM is extending WMQ to take advantage of these new models…**

- **Goals:**
  - Easy access to WMQ from HTTP and the World Wide Web
  - To make WMQ easier to use in Service Oriented Architectures and web services

# Extending the Reach of Applications Using MQ

N O T E S

- Recent years have seen a growth of interest and adoption of distributed programming styles, led primarily by the success of standards based Web services. The success of Web services standards based on SOAP (WS-*) has been based on the promise of greatly improved code re-use and component interoperability through the development of composite applications, built from services; using a variety of qualities of service; and accessible over a range of communication transports.

- The success of Web services has been dramatic, but they are not seen as the best solution to *all* problems. The specifications are geared towards recreating many of the traditional Enterprise qualities of service (transactionality etc) over interoperable XML protocols. This makes the specifications complicated, and the XML-based protocols are relatively heavyweight. Web services implementations rely heavily on tools to hide this complexity which pushes up the cost-of-entry in terms of skills and software product requirements.

- For simpler scenarios, a section of the software development community is advocating more basic APIS and protocols – typically based on HTTP so that they can be invoked directly from Rich Internet (Web 2.0) Applications. These APIs are geared towards keeping simple things simple, and so deliberately steer clear of the complexities of Enterprise qualities of service, leading to a lower cost-of-entry than SOAP and WS-*,. Many of these APIs follow the REST best practices discussed later in this presentation, and use plain old XML (POX) or JavaScript Object Notation (JSON) message formats.

- WebSphere MQ is very relevant in these environments and so is being extended to take advantage of these new programming models with the goals of providing: Much simpler access from Web and HTTP clients, so enterprise date can be much more readily accessed and closer integration with Service Orientated Architectures so applications can be more easily developed, managed and re-used in this environment.

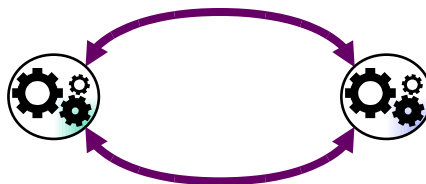# WMQ Web Services Themes
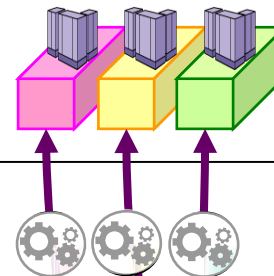
1. Easy Access from the Web

- Enrich Rich Internet Apps with data from core systems
- Extends reach and ROI of existing enterprise apps
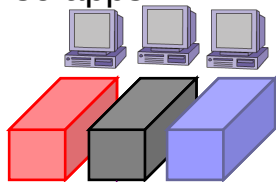
Web 2.0
REST, AJAX, JSON

3. Managing WMQ Apps as Services

- Reduces time and effort to develop Enterprise apps
- Enables Re-use, Governance etc in SOA without re-working interfaces

Enterprise Messaging Backbone

WebSphere MQ

2. Robust Async SOAP Transport

- Decouple client and service availability
- Quality of service: more tangible, traceable and reliable then HTTP.
- Transactionality; Scalability
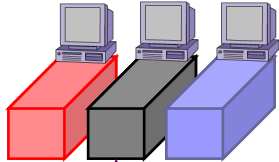
MQ Web Services Themes

- To support these goals, WebSphere MQ is being developed under three main themes:

- The theme supports the first goal of providing easier access from HTTP and the Web. As mentioned previously, this is all about providing much simpler access to enterprise data from 'lightweight' Web style clients, allowing low QoS "Zero footprint" client applications to access MQ over HTTP, eliminating the need to deploy MQ client libraries when client applications are deployed.

- The two remaining themes are in support of the second goal of more closely integrating WMQ with Service Orientated architectures and web services.

- The second theme is enable the reliable and asynchronous qualities of MQ to be taken advantage of for carrying SOAP messages between standard web service clients and services, as an excellent compliment to the HTTP's less reliable synchronous protocol.

- The third theme is about being able to manage MQ applications as services. Enabling native MQ applications to be catalogued as a first-class part of an SOA, and consumed in composite applications, in the same was as any other web service.

- A majoroity of what you'll be seeing today is fairly newly in the product and all are still focus areas for development, so any feedback is greatly appreciated.

MQ Web Services Themes
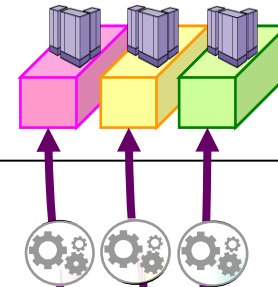
1. Easy Access from Web Clients

➤ *WebSphere MQ Bridge for HTTP*
  • Web *Client* side access to WMQ

➤ *JSP/Servlets & "SAM" PHP API*
  • Web *Server* side access to WMQ

Web 2.0
REST, AJAX, JSON

3. Managing WMQ Apps as Services

➤ *WMQ Service Definitions*
  • WSDL for WMQ Applications

Enterprise Messaging Backbone

WebSphere MQ

2. Robust Async SOAP Transport

➤ *SOAP over JMS, IBM & Standard*

➤ *WMQ Channel for Windows Communication Foundation*

MQ Web Services Themes

1. Easy Access from Web Clients

➢ ***WebSphere MQ Bridge for HTTP***
  - Web *Client* side access to WMQ

➢ ***JSP/Servlets & "SAM" PHP API***
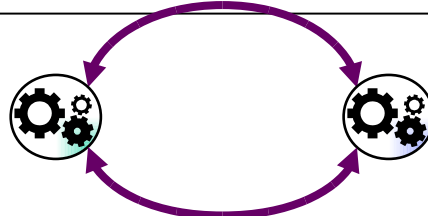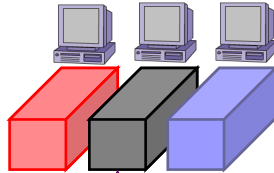 - Web *Server* side access to WMQ

Web 2.0
REST, AJAX, JSON

3. Managing WMQ Apps as Services

➢ ***WMQ Service Definitions***
  - .wsdl for WMQ Applications

Enterprise Messaging Backbone
WebSphere MQ
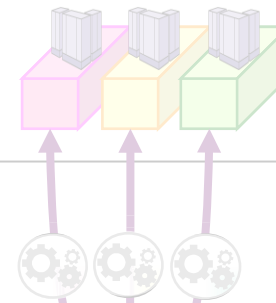
2. Robust Async SOAP Transport

➢ ***SOAP / JMS Standard***

➢ ***WMQ Channel for Windows Communication Foundation***

Part 1a – Web Client Access
MQ-HTTP Bridge

Allows applications to connect to WMQ over HTTP.

Two main goals:

1. Simplifies access to MQ Apps from browser-based Rich Internet Applications

- Gives AJAX and Rich Internet applications access to the Enterprise
  - Submit data direct to queues & topics from Browser

2. Non-browser-based "Zero Footprint Client" - Enable MQ Application Connectivity from *any* Platform or Language with HTTP capabilities

- Significantly increase range of supported platforms
  - e.g. – Linux distros, POS terminal running Windows Services for Unix environment, RFID reader, Mobile devices
- Reduce cost of MQ Client deployment and maintenance
  - No client library installation required
- Lightweight (low qualities of service) messaging

HTTP-MQ Goals

- Web Browsers provide a near ubiquitous platform for deploying client applications
- HTTP is the pervading transport protocol

- The first goal of the MQ HTTP feature is to extend the reach of MQ applications to these environments over HTTP
- This will give Rich Internet Applications simplifed access to the Enterprise
- For example

  - web-based clients will be able to submit data (e.g. forms) as MQ messages directly to enterprise batch applications
  - Req/resp MQ applications can be fronted with web-based clients
  - Pub sub – data flowing through network

- Another goal is to provide access to MQ from many more environments
- Eliminating the MQ client will reduce the cost of application deployment
- Though of course this is not a complete replacement for the MQ client

  - It is missing many MQI features and does not offer transactionality, assured delivery etc.
  - But in many cases where applications have resend logic and check for duplicates it will be good enough

Two Implementation Options Available
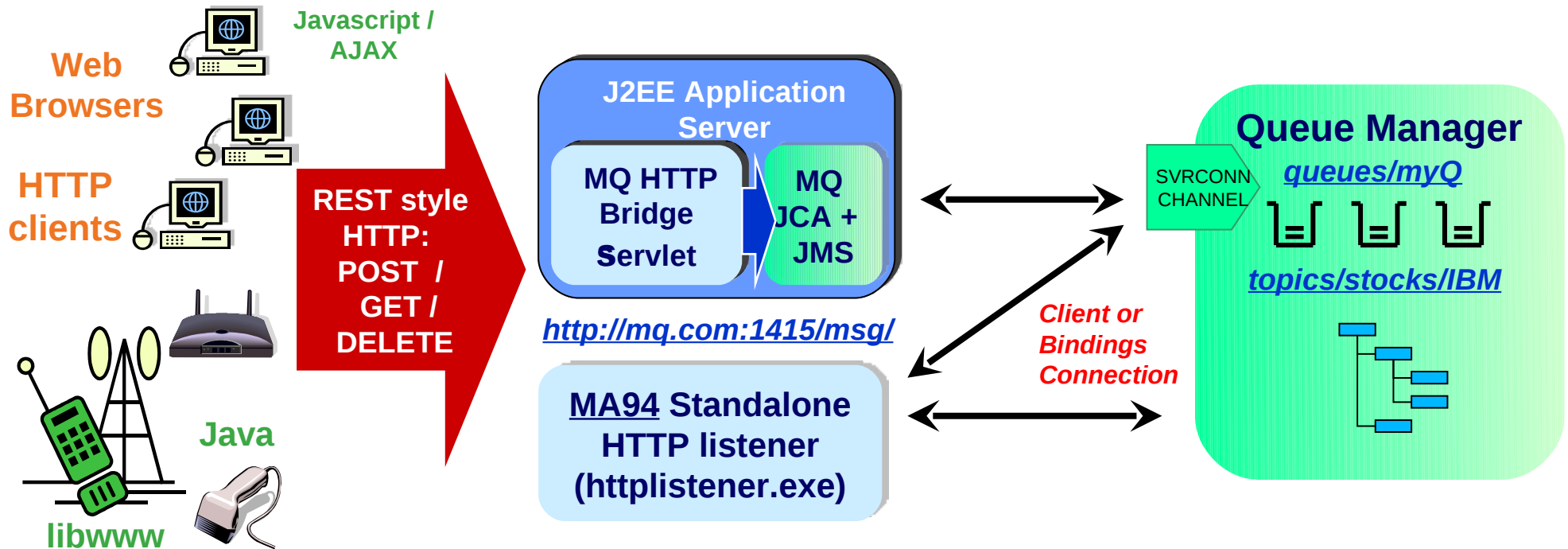
1. SupportPac MA0Y & WMQ V7.0: HTTP Bridge implemented as a Servlet
   - Suitable for deployment to existing J2EE app. servers (WAS, Geronimo, JBoss)
   - Back end connectivity uses the WMQ JCA resource adapter and WMQ JMS
   - JMS ConnectionFactory configuration determines if MQ connection uses client mode or bindings

2. SupportPac MA94: Standalone HTTP Listener
   - Available for Windows, AIX, and Linux
   - Client or bindings mode, Security exits

Architectural Overview

- The functionality is in two SupportPacs – MA0Y which is a servlet based implementation which can be deployed in any J2EE app server, or MA94 which is a standalone executable HTTP listener which runs on Windows, AIX, and Linux.
- Front end connectivity is over HTTP(S) into the Application Server, or HTTP into the Listener, enabling any HTTP capable device or scripting language to send and receive messages. This could include browser based web 2.0 applications using AJAX to dynamically refresh data, server side applications, or mobile/point of sale devices.
- For the MA0Y, backend connectivity is supported by using the WMQ JCA adapter or WMQ JMS, the choice of which depends on the application server. Either way, the connection mode that's used to connect to MQ can be configured using a JMS ConnectionFactory, both Bindings and clients modes are supported. Authentication and access control to URI's must be configured in the application server.
- MA94 also supports Client or Bindings connections to the queue manager, is secured by a security exit, and includes some additional features like a "memory" of messages delivered to topics
- The HTTP interface is modelled on REST

Background – HTTP-MQ Loosely Modelled on REST

REpresentational State Transfer (Roy Fielding)

o Everything is modeled as a Resource

o Every resource is identified by an address (URI)

o Resources have state (representation)

o HTTP is used to transfer state to networked application

o HTTP verbs operate on the resource

➢ **GET** → retrieves a resource's state representation

➢ **POST** → Updates resource (or other processing)

➢ **DELETE** → deletes resource

➢ **PUT** → Creates / updates resource state

Server		
Resource	State Representation	HTTP GET
	State Representation	HTTP POST
State	State Representation	HTTP PUT
		HTTP DELETE
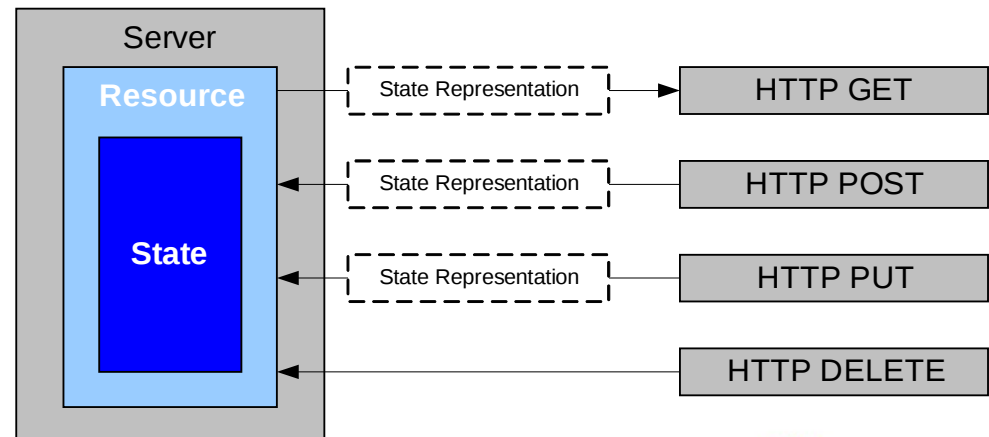
Background – HTTP-MQ Loosely Modelled on REST

N
O
T
E
S

- REST was originally described by Roy Fielding (principle author of HTTP spec) in his 1999 dissertation.
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- Has been described as the "Architecture of the WEB" – essentially a set of architectural best practices for highly scalable scalable distributed applications. HTTP and the Web is the de-facto standard implementation of REST. It represents a Client-Server Protocol with a uniform interface across resources (HTTP Methods POST, GET, DELETE, etc). Resources are identified by URIs, connections are stateless etc.

- So for example, a REST interface for a online catalogue would have a separate URI for each item in the catalogue. Items could then be added by using a PUT, removed by using a DELETE and enquired by using a GET.

- REST therefore offers a different integration style to WS-* standards based web services. Qualities of service are sacrificed for simplicity and scalability to keep barriers-to-entry low. REST API's are typically simple and so don't require an SOA expert, or heavyweight tools to implement. They can be used spontaneously **and** incrementally – for example in Web 2.0 mash-ups.

- "Good Enough" is often Good Enough

HTTP-MQ API: Verb / Resource Mapping

- **Defines an IRI to identify the address of the queue (or topic)**

- **API is a simple mapping of HTTP verbs to the MQI…**

		HTTP verb mapping			
Resource	**Sample URIs**	**GET**	**POST**	**PUT**	**DELETE**
Messages	**http://host/msg/queue/qname/** **http://host/msg/topic/topic_path/**	MQGET w. browse	MQPUT	-	MQGET

- **Message header fields (MQMD) are conveyed in HTTP headers**
  - All HTTP headers prefixed with "x-msg-"
- **Message Body is passed in HTTP entity body**
- **Message type is conveyed in HTTP Content-Type**
  - HTTP media-types are used :
  - "text/plain" or "text/html" equate to MQ string messages (MQFMT_STRING)
  - All other media types map to MQ Binary messages (MQFMT_NONE)
- **No plan to supply client-side libraries – apps code direct to HTTP**

HTTP-MQ Verb / Resource Mapping

- The HTTP/MQ API is largely based on REST, though it has some quirks. For example MQ/HTTP transfers message representations, but messages are not ideal REST resources
  - They do not necessarily have a unique identifier, and so cannot be addressed individually
  - Not generally amenable to caching etc. because they must be delivered only once
  - They are very transient
- The API defines a URI to address messages on queues and topics
- It is a stateless / connectionless API with one HTTP verb -> one MQ operation
  - HTTP headers = Message headers
  - request headers (get and put options) – wait, requires-headers
  - entity headers (MQMD options) – priority, expiry, timestamp, persistence, msgId, correlId, replyTo, encoding
  - HTTP request payload = Message body
- Message Body
  - Passed as the HTTP entity body - can be text or binary
  - The MQ message type is conveyed in the HTTP Content-Type header
- We don't supply client libraries – applications code directly to HTTP verbs
- The API is not finalised !!! - e.g. some firewalls may block DELETE and extended HTTP headers, so In the future we may allow other resource types in the URI
  - For instance 'status' resources to allow Queries on the status of MQ channels, 'admin' resources to allow administration alternatives may be required (possibly passing additional data as IRI query parameters)

Deconstructed HTTP Flow - POST (= MQPUT)

Put to destination

Request:

```
POST /msg/queue/requestQ/ HTTP/1.1
Host: www.mqhttpsample.com
Content-Type: text/plain
Content-Length: 60
x-msg-replyTo: /msg/queue/replyQ/
x-msg-require-headers: msgID, priority, timestamp
Message body which will appear on the queue as an MQSTR
```

Type and length of message (60 char string)

reply Queue

Headers to include on reply

Message Data

Response:

```
HTTP/1.1 200 OK
x-msg-msgID: 1234567890
x-msg-timestamp: Thu, 22 Mar 2007 08:49:37 GMT
x-msg-priority: 4
```

Response code

Required Headers returned

Deconstructed HTTP Flow - POST (= MQPUT)

The previous slide shows the HTTP flow on the wire for putting an MQ request message to a queue.

Request

- The first line is the request line which identifies the HTTP verb (i.e. POST - mapped to an MQPUT)
- The request line is followed by the HTTP headers are passed as colon separated name-value pairs
- The Content-Type header = text/plain tells the MQ HTTP gateway to put the message to the queue using MQMD.Format = MQFMT_STRING.
- The Content-Length header tells the MQ HTTP gateway that the message is 60 bytes long
- "x-msg-replyTo" is an extension header which specifies the reply queue and queue manager to go in the MQMD.ReplyToQ and ReplyToQmgr fields
- "x-msg-requiresHeaders" is an extension header which specifies the header fields which the MQ HTTP header should return in the HTTP response.
- The HTTP headers are followed by the HTTP entity body which contains the message data

Response

- The response to a POST shows whether the message was put successfully to the queue or not.
- The first line of the HTTP response is the status line – which includes the response code (in this case it is 200 - "OK")
- The headers which follow the status line are the headers requested by "x-msg-requiresHeaders" i.e. msgID, priority, and timestamp

N
O
T
E
S

Deconstructed HTTP Flow - DELETE (=MQGET)

Request:

```
DELETE /msg/queue/replyQ/ HTTP/1.1
Host: www.mqhttpsample.com
x-msg-wait: 10
x-msg-correlID: 1234567890
x-msg-require-headers: correlID
```

Get from destination

Get wait time (ms)

Get using this correlID

Headers to include on reply

Response:

```
HTTP/1.1 200 OK
Content-Length: 60
Content-Type: text/plain
x-msg-correlID: 1234567890
Response message body which will appear on the queue as an MQSTR
```

Response code

Message length

Message data type

Retrieved Message

Correlation ID

Retrieved Message Data

JavaScript XHR Example: Web Page to put a message

```javascript
<head>
<script language="javascript">
 function doPOST() {
    xmlDoc = new XMLHttpRequest();
    xmlDoc.onreadystatechange = processPOSTResponse;
    xmlDoc.open("POST", "http://localhost:50667/msg/queue/myQueue/", true);
    xmlDoc.setRequestHeader("Content-Length    11);
    xmlDoc.setRequestHeader(" x-msg-requiresHeaders ", "msgID, timestamp ");
    xmlDoc.send("Hello World");
 }

 function processPOSTResponse() {
    if ( xmlDoc.readyState == 4 ) {
       if (xmlDoc.status == 200) {
          window.status = "POST succeeded";
       } else {
          window.status = "POST failed: " + xmlDoc.status;
       }
    }
 }
</script>
</head>
<body onload="doPOST() " >
  This page sent a message when it was loaded.
</body>
```

Java Example: Method to PUT a message over HTTP

```java
public void doPOST() {
  try {
    // Set URI to point to queue
    URL postURL = new URL("http://localhost/msg/queue/myQueue/" );

    // Open HTTP connection to URI
    HttpURLConnection connection =
            (HttpURLConnection)postURL.openConnection();
    connection.setRequestMethod("POST"); // Set HTTP method
    connection.setDoOutput(true);

    // Send message
    PrintWriter out = new
                      PrintWriter(connection.getOutputStream());
    out.println("Hello from a Java app");
    out.flush();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Current Status & Summary

MA0Y & WMQ V7.0 – JEE Servlet-based HTTP Bridge

- Windows SupportPac (MA0Y) includes a bundled copy of WAS CE
    - Pre-configured with samples
    - Pre-configured with JCA
    - Suitable for WMQ V6.0.2.1 onwards

MA94 – Standalone listener – no JEE pre-requisites

- Runs on Windows, Linux, AIX
- Ajax Samples, and Sample security exit

Benefits

- No client libraries required – use stand-alone or in a Browser
- Simple API mapping HTTP to MQ
- Low cost of entry
    - develop in notepad, ad-hoc service descriptions / message formats

Limitations

- As (un)reliable as HTTP
- No transactional capabilities
- Subset of MQI

Current Status
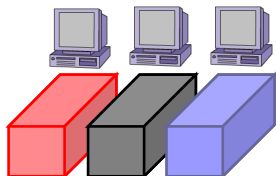
- The feature is delivered in the (unsupported) Category 2 SupportPacs mentioned and the (supported) WMQ V7.0
- Longer term we intend to add a tightly integrated HTTP listener and channels as a native part of WMQ

- Limitations
  - REST is not good for everything – for instance event-based low latency systems do not work well in the REST model (a separate 'server-push' API over HTTP is under investigation to address that problem). REST is also silent on many of the quality of service standards typified by WS-* like reliable messaging, duplicate elimintion, end-to-end security etc. (but this contributes to it's simplicity so can be considered a plus point).

- Longer term
  - Could become a new listener type integrated within the Qmgr - e.g.
    - No application server pre-requisite – HTTP clients connect direct to MQ
    - A new HTTP channel type and an HTTP listener
    - No access by default, secured by exits

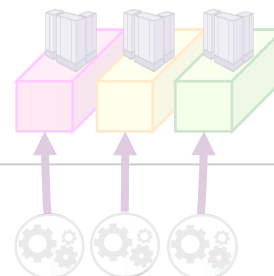# MQ Web Services Themes

1. Easy Access from Web Clients

➢ ***WebSphere MQ Bridge for HTTP***
  • Web <u>Client</u> side access to WMQ

➢ ***JSP/Servlets & "SAM" PHP API***
  • Web <u>Server</u> side access to WMQ

Web 2.0
REST, AJAX, JSON

3. Managing WMQ Apps as Services

➢ ***WMQ Service Definitions***
 • .wsdl for WMQ Applications

Enterprise Messaging Backbone
WebSphere MQ

2. Robust Async SOAP Transport

➢ ***SOAP / JMS Standard***

➢ ***WMQ Channel for Windows Communication Foundation***

Part 1b – Web Server Access JSP/Servlets

- Servlets
  - Java class that outputs text, such as HTML
  - Can run any Java code, such as JMS
  - Code run on server, text returned to web client
- Java Server Pages (JSP)
  - HTML-like document
  - Tags to embed Java code, such as JMS
  - JSP page compiled into Servlet
  - Run on server, text returned to web client
- Both technologies part of JEE application servers
  - For example, WebSphere Application Server
  - Standalone web container servers available, such as Tomcat

Web client/browser

HTTP Get/Post

JEE Server

JSP/Servlet

JMS

WebSphere MQ JMS

MQ
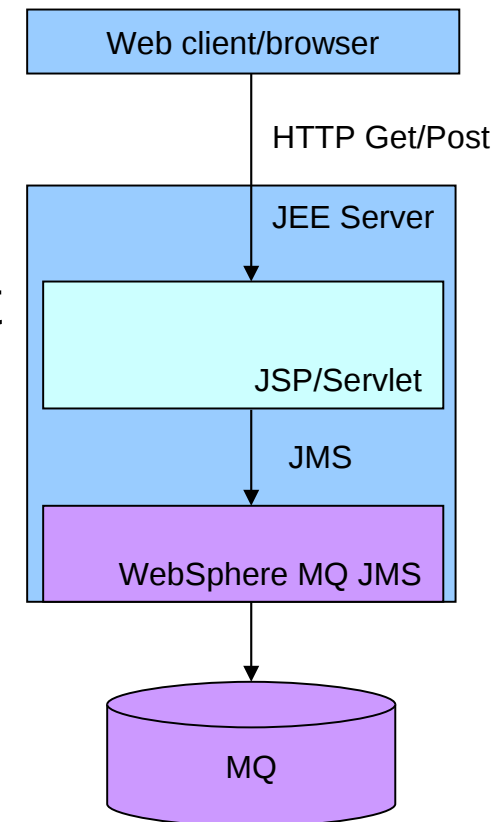
Part 1b – Web Server Access JSP/Servlets

- JSP

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Page Title</title></head>
<body>
<%
   Connection conn = connectionFactory.createConnection();
%>
</body>
</html>
```

- Servlet

```
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
   …
   response.getWriter().println(message.getText());
}
```

Part 1b – Web Server Access
Simple Asynchronous Messaging (SAM)

- Languages like PHP simplify creating dynamic web pages
  - They are not over engineered but good enough
  - The languages and runtimes are largely open-source
  - They are lightweight and require no tooling
- Aims of SAM:
  - Develop a simplified messaging API for PHP
  - Can be used uniformly across the WebSphere messaging family
  - Deliver as an open-source extension

Web Server

PHP Engine

PHP Scripts

HTTP

SAM Extension

XMS C API

Brokers WMQ, WAS, MicroBroker etc…
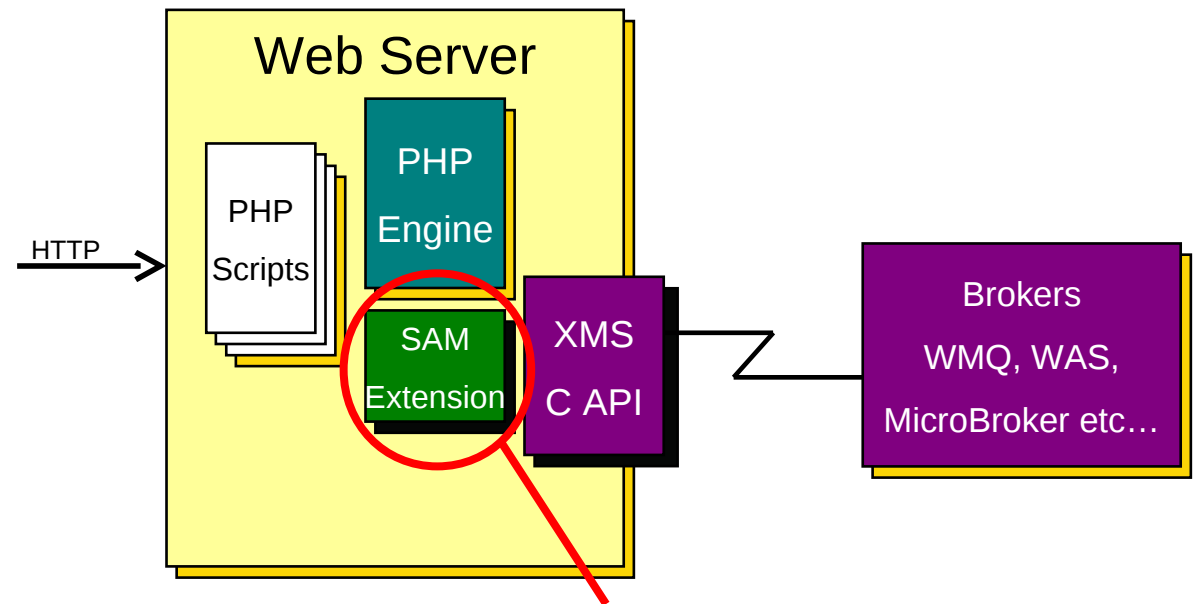
This is the SAM deliverable

SAM Home Page: http://project-sam.awardspace.com

SAM – Simple Asynchronous Messaging

WebSphere MQ provides a number of messaging API to support the wide range of messaging environments and requirements. This includes:

Message Queue Interface (MQI) - Fully functional low level API focussing on performance and functionality in C, C++, Java, .NET, COBOL and PL/1

Java Message Service (JMS) - Support for Java in the J2EE environment.

Message Service Clients (XMS) - Extends the reach of JMS style messaging to the C, C++ and .NET languages

However, these do not provide an ideal match for integration with PHP or similar languages where simplicity is the primary focus.

The SAM extension for PHP is a framework that provides a very simple API that can be used uniformly across the range of WebSphere middleware messaging systems.

Currently the package includes built in support for MQTT (MQ Telemetry Transport) but can also communicate with WebSphere Message Brokers, WebSphere MQ and WebSphere Application Server through the use of additional pre-req software.

SAM is designed to be readily extended to support other messaging systems and extension modules may be written in C or PHP.

API style – send and receive example

```php
// Create a connection to a broker
$conn = new SAMConnection();
$conn->connect(SAM_WMQ, array(SAM_BROKER =>
'MyQueueManager'));


//Create a message
$msg = new SAMMessage('Simple text message');


// set a reply queue identity in the message header
$msg->header->SAM_REPLY_TO = 'queue://receive-test';


//Send it and capture the correlation id that is returned…
$correlid = $conn->send('queue://send-test', $msg);


// receive the answer but only wait for 5 seconds
$answer = $conn->receive('queue://receive-test',
        array(SAM_WAIT => 5000, SAM_CORRELID =>
$correlId) );
```

API style –send and receive example

Demonstrates the script required for a typical send and receive operation (with error handling removed for clarity).

A request message is sent to an WebSphere MQ queue destination. To enable the application processing the message to route its response message back to us, a reply destination is also specified in the request.

The receive call is then made, providing the correlation id of the sent message, this ensures that we receive the correct reply for our request.

Architecture overview

SAM is provided as a PHP extension. Some optional pre-reqs are also required depending which messaging server is being used. These are as follows:

WebSphere Message Brokers - MQ Telemetry Transport (MQTT) protocol:
- No additional software.

WebSphere Message Brokers - Realtime Transport (RTT) protocol:
- IBM Message Service Client for C/C++ (XMS C/C++)

WebSphere MQ – Client or bindings connections
- IBM Message Service Client for C/C++ (XMS C/C++)
- IBM WebSphere MQ (a local queue manager or clients package)

WebSphere Application Server
- IBM Message Service Client for C/C++ (XMS C/C++)

Interested in taking a look?

- **Downloads**

  - Available for PHP 4 and 5

  - Source available from PECL: http://pecl.php.net/package/SAM

  - Binaries (limited binaries available for Windows from the SAM home page)

- **Latest Version: 1.1.0**
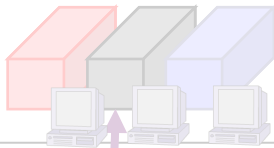
- **Further info**

  - SAM Home Page: http://project-sam.awardspace.com

  - Google Group: http://groups.google.co.uk/group/phpsam

….Feedback welcome

MQ Web Services Themes

1. Easy Access from Web Clients

➢ *WebSphere MQ Bridge for HTTP*
  • Web *Client* side access to WMQ

➢ *JSP/Servlets & "SAM" PHP API*
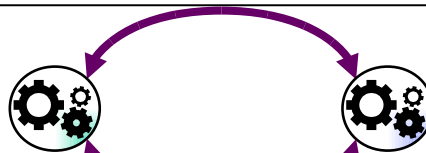 • Web *Server* side access to WMQ

3. Managing WMQ Apps as Services

➢ *WMQ Service Definitions*
  • .wsdl for WMQ Applications

Web 2.0
REST, AJAX, JSON

Enterprise Messaging Backbone

WebSphere MQ
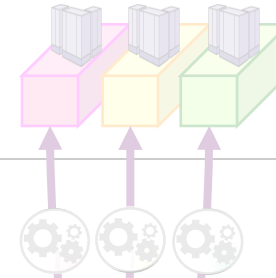
2. Robust Async SOAP Transport

➢ *SOAP over JMS Standard*

➢ *WMQ Channel for Windows Communication Foundation*

Part 2a - WMQ SOAP Transport Feature

- WMQ is an ideal transport for WS-* Web services
  - Offers enhanced qualities of service compared with Web services over HTTP, e.g.:
    - Asynchronous invocation – decoupled in time, multi-hop, reroute responses
    - Queuing offers more reliable scaling characteristics
    - Better manageability – more tangible messages
      - *Use MQ infrastructure, track messages, find bottlenecks etc.*
- WMQ includes a 'SOAP Transport' feature
  - Enables WMQ to carry WS requests as a pluggable replacement for HTTP.
  - Made available in WMQ v5.3 as SupportPac MA0R - "MQ Transport for SOAP"
  - Fully integrated into product in WMQ v6.0
- IBM interoperable way of carrying SOAP messages in WMQ
  - Enables SOAP requests to be interpreted by IBM products WAS, CICS
  - Named 'SOAP over JMS' specifies how SOAP message is carried as the payload of a JMS message

Part 2 - WMQ SOAP Transport Feature

**N
O
T
E
S**

WMQ offers a number of advantages over HTTP as a web services transport

- The first advantage is in improved scalabilty for birectional data exchanges. HTTP Caches offer amazing scalability when data is static, but are no use for scaling unique req/resp. and HTTP infrastructures need to be designed to be able to handle the peak number of concurrent requests. Queues are inherently better at variable workloads.

- The second advantage is manageability

  - WMQ / JMS messages are more tangible and traceable than HTTP requests

- The third advantage is asynchrony

  - Limited (short term) async has existed in .NET for some time, and it has been added to Java JAX-WS. This does not offer any guidance on how to persist and correlate requests over a long time. True queued async adds resilience – (both parties don't need to be running at the same time), and requests can be longer running than typical HTTP request.

- To allow exploitation of WMQ as a SOAP transport, WMQ has been offering SOAP/JMS support since v5.3 with SupportPac MA0R.
See SupportPacs site for download…

- http://www-1.ibm.com/support/docview.wss?rs=171&q1=mA1J&uid=swg24006280&loc=en_US&cs=utf-8&lang=en

- This SupportPac allows services to be created and deployed to MQ – so MQ can be used as a service hosting environment, but we expect most services in production to be hosted in other – more functional managed environments like CICS or WAS.

- The most important thing about this SupportPac is that it defined a standard way for carrying SOAP service invocations over MQ – thus ensuring interoperability between CICS, MQ, and WAS.

- Any message format could have been chosen, but JMS fits in best for compatibility with WAS.
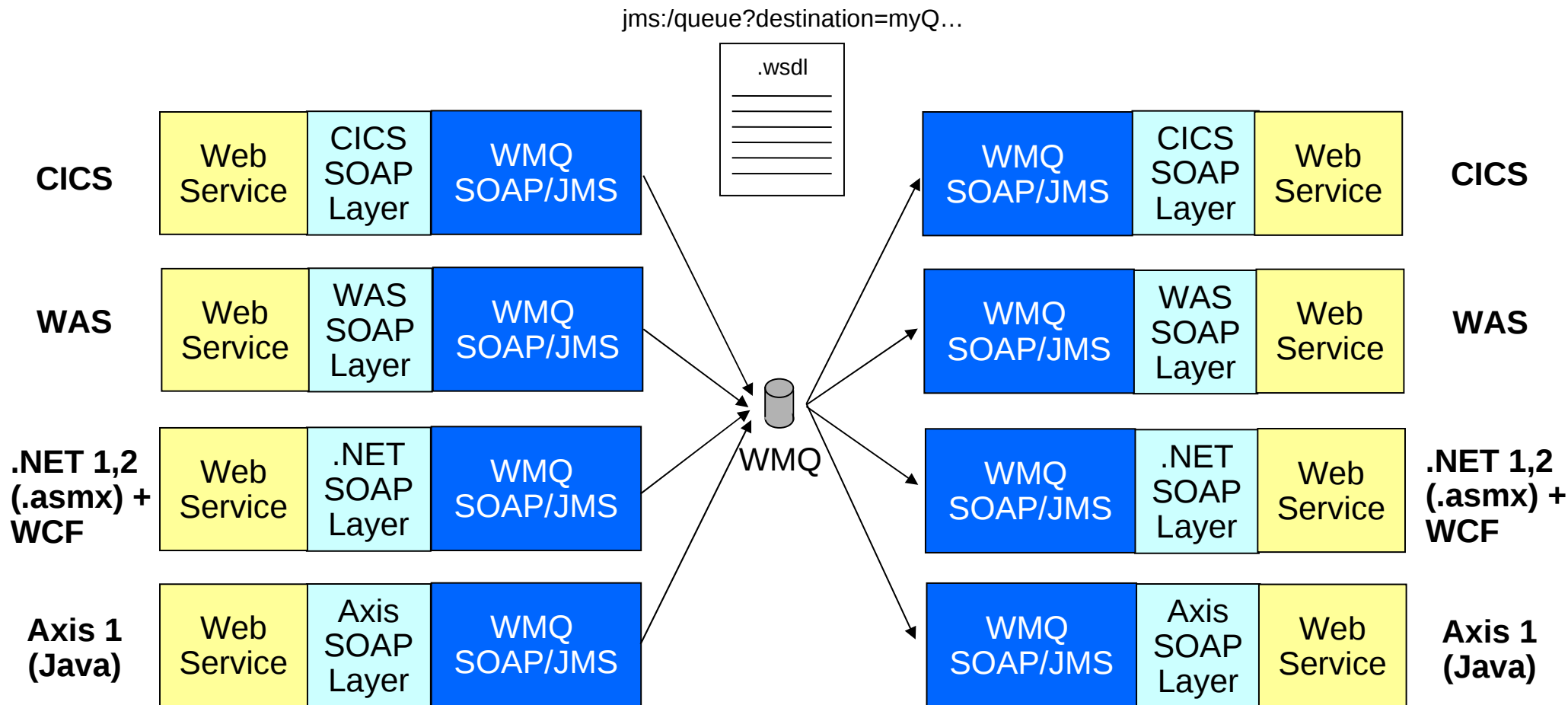
SOAP Transport for WMQ - Architecture

Typical Deployment of Web Service using HTTP
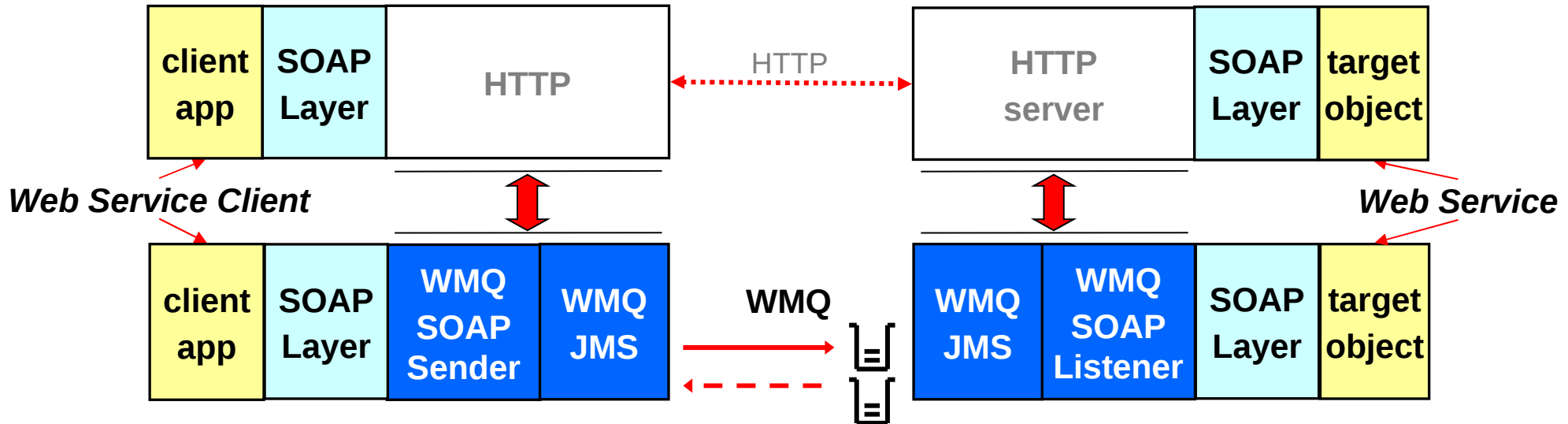
Deployment of Web Service using WMQ

- **Directly interchangeable with HTTP for supported SOAP Layers**

- **Message sent using 'SOAP over JMS' format**

  - JMSBytes or JMSText message types (with RFH2 etc.) with SOAP request carried as message payload.

  - Additional properties (e.g. 'SOAPAction') carried in JMS message props.

- **Services accessed using JMS URI**

  - URI of form '**jms:queueName**…'

  - Options to control other WMQ details such as queue manager, persistence etc
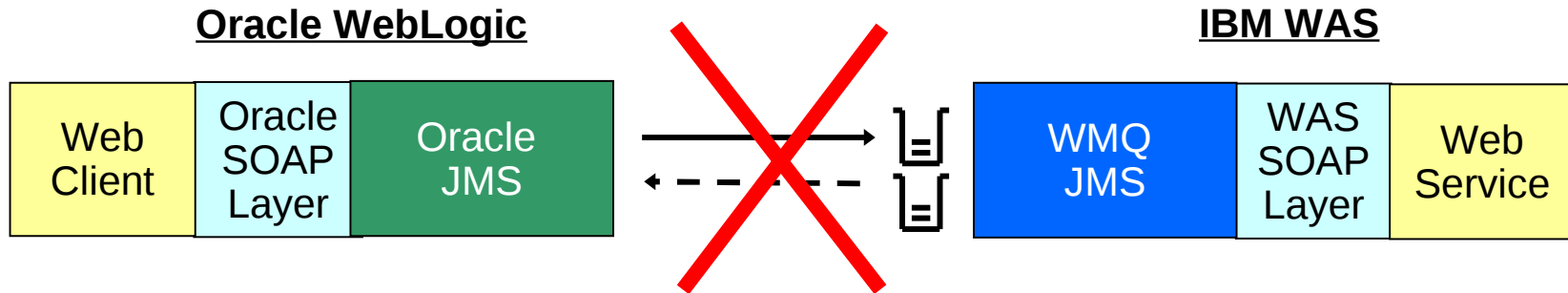
SOAP Transport for WMQ - Interops

jms:/queue?destination=myQ…

.wsdl

	Web Service	CICS SOAP Layer	WMQ SOAP/JMS
CICS			

	Web Service	WAS SOAP Layer	WMQ SOAP/JMS
WAS			

	Web Service	.NET SOAP Layer	WMQ SOAP/JMS
.NET 1,2 (.asmx) + WCF			

	Web Service	Axis SOAP Layer	WMQ SOAP/JMS
Axis 1 (Java)			

WMQ

WMQ SOAP/JMS	CICS SOAP Layer	Web Service	
			CICS

WMQ SOAP/JMS	WAS SOAP Layer	Web Service	
			WAS

WMQ SOAP/JMS	.NET SOAP Layer	Web Service	
			.NET 1,2 (.asmx) + WCF

WMQ SOAP/JMS	Axis SOAP Layer	Web Service	
			Axis 1 (Java)

▪ Environments which currently support using WMQ as a SOAP transport

SOAP Transport for WMQ - Interops

N

O

T

E

S

- SOAP/JMS shipped in WMQ v6
- Scripts allow Java or .NET objects to be deployed as a SOAP/JMS web service over MQ
  - Utilities generate WSDL, client proxies, Request & Response Queues etc.
  - A WMQ Web service is identified by URI prefix "jms:" E.g.
    - jms:/queue?destination=myQ&connectionFactory=()&targetService=myService &initialContextFactory=com.ibm.mq.jms.Nojndi

- Client applications call SOAP/JMS services as if they are HTTP services - using generated proxy code
  - One additional line of code is required to register the "jms" extension
    - Java : com.ibm.mq.soap.Register.extension ();
    - C# : IBM.WMQSOAP.Register.Extension();
    - Visual Basic : IBM.WMQSOAP.Register.Extension()
- On the Server – a new MQ SOAP/Listener process
  - waits for messages to arrive on the Q
  - Hands off the request to the appropriate SOAP stack (.NET or Axis 1)
  - The SOAP stack de-martials the SOAP, invokes the service, and queues the response
- Interoperates with WAS and CICS SOAP/JMS implementations allowing
  - MQ services to be called from WAS, or CICS
  - WAS or CICS services to be called using MQ as the transport

SOAP Transport for WMQ–Vendor Interops

Oracle WebLogic

Web Client	Oracle SOAP Layer	Oracle JMS

IBM WAS

WMQ JMS	WAS SOAP Layer	Web Service

- **Different vendors implementations do not interoperate**
  - Each vendor has their own SOAP/JMS implementation
  - Progress, TIBCO, Oracle, Axis…
- **Interoperability is required**
  - Web services and clients hosted on different vendors J2EE Application Servers should interoperate.
  - Vendor solutions can not interoperate on the wire (JMS is a standard API but not a standard wire format)
- **…*should* be able to interoperate at the JMS API level**

Oracle WebLogic

client app	Oracle SOAP Layer	Oracle SOAP/ JMS	WMQ JMS

WMQ

IBM WAS

WMQ JMS	WAS SOAP/ JMS	WAS SOAP Layer	target object
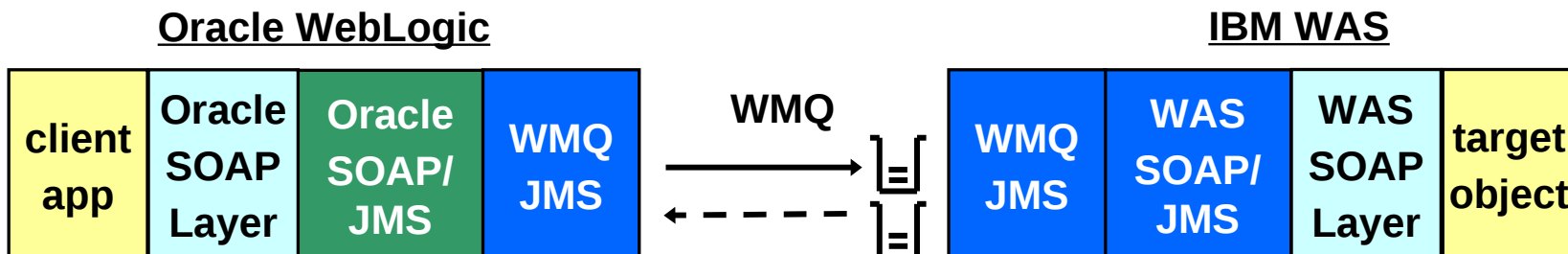
SOAP/JMS – Vendor Interoperability

- A number of other vendors have also implemented SOAP/JMS in their Applications Servers, ESB's, etc.

- Customers have heterogeneous environments
  - For example using MQ as the JMS provider for BEA Weblogic
  - Using a mixture of WAS and other application servers
- …and would like interoperability when invoking services using SOAP over JMS
  - E.g. Consuming a service deployed in in WAS or CICS from a client running in BEA WebLogic

- Different vendor JMS solutions can not interoperate on the wire (because JMS is a standard API but not a standard wire format), but if

- Ideally it should be possible for different SOAP/JMS implementations to interoperate.
  - Sonic, TIBCO, BEA, Axis
- …but *should* be able to interoperate at the API level – e.g. WAS -> WebLogic

SOAP/JMS Standardisation

- ## A SOAP/JMS standardisation effort is underway

  - ### W3C SOAP-JMS Binding Working Group: http://www.w3.org/2002/ws/soapjms/

    - Participants include IBM, Oracle (BEA), TIBCO, WSO2, Progress, Software AG, Sun, Cisco
    - In "Candidate Recommendation" stage – hopes to finalise in 2010
      - *Awaiting two interoperable implementations before completion*
      - *Already implemented in WAS 7*

- ## Standardisation will make it possible to have:

  - ### JMS provider-neutral deployment of SOAP/JMS services

  - ### Bidirectional HTTP - SOAP Bridging

    - *Maintaining integrity of SOAP message and protocol metadata*

SOAP/JMS Standardisation

- A SOAP/JMS standardisation effort between IBM, BEA, Sonic, and TIBCO has been underway for some time. In Oct '06 the specifications reached v1.0 RC1 and were distributed to five more vendors and various customers for feedback. WebMethods subsequently joined the core team.

- http://mail-archives.apache.org/mod_mbox/ws-axis-dev/200701.mbox/%3c80A43FC052CE3949A327527DCD5D6B27020FB65C@MAIL01.bedford.progress.com%3e

- Standardisation will make it possible to mix and match JMS providers and SOAP transport handlers, and will allow HTTP SOAP requests to be bridged to Message oriented middleware transports without losing metadata like SOAPAction

- There is growing interest in this standard – for instance at a recent W3C workshop (http://www.w3.org/2006/10/wos-ec-cfp.html) standardisation of SOAP/JMS was the feature most requested by the participants

- Standardisation will be a step towards WS transport independence – something which so far has been lacking in WS-* - an issue identified by Paul Downey in his - BT Position Paper "Services on the Web and Web Services"
  "Description of Transports
  In spite of Web services advocating transport independence, there are currently no well supported means to describe the WSDL endpoint for many commonly used paths such as IBM MQSeries or Sonic MQ, which are often abstracted using the Java Messaging Service (JMS). This is quite baffling, given how many of these transports are under the direct control of individual companies who advocate Web services." http://www.w3.org/2007/01/wos-papers/bt

SOAP/JMS Specification Details

Based on IBM SOAP/JMS but some minor differences

Consists of 2 specification documents

1. SOAP Binding Specification
   - Message Format (BytesMessage)
   - Properties that MUST / SHOULD flow in JMS messages
   - SOAP/JMS WSDL bindings
   - Request / Response & One-Way MEP State Machines
   - **Member submission Submitted to W3C - http://www.w3.org/Submission/2007/05/Comment**

2. URI Scheme Specification – jms:
   - "Internationalized Resource Identifier" (updated version of URI - RFC3987)
   - e.g. jms:news?targetService=current-affairs &connectionFactory=SOAPJMSFactory &deliveryMode=2 &priority=8 replyToName=interested
   - **The scheme is under review on the IETF review list: http://www.ietf.org/internet-drafts/draft-merrick-jms-uri-01.txt**

- SOAP/JMS enables interoperable access to services over MQ
   - It isn't restricted to Java (e.g. .NET support in MQ v6)
   - Non-Java Applications can construct JMS messages to invoke services
      - MQ JMS message formats are all published, so e.g. COBOL apps can build JMS messages

(left margin vertical text: N O T E S)

SOAP/JMS Specification Details

The SOAP/JMS standard is being delivered in 2 specifications.

1. The SOAP Binding Specification which defines how JMS messages should be constructed. This specifies the message type which should be used (BytesMessage), and the properties that MUST or SHOULD flow in JMS service invocation messages. It also specifies how to describe a SOAP/JMS service in WSDL bindings, and defines State Machines for the supported Message Exchange Patterns (Request / Response and One-Way).
   - Now submitted to W3C who have proposed a Working Group be chartered
2. The SOAP/JMS IRI Scheme defines the syntax of the jms: IRI (an "Internationalized Resource Identifier" is a address (URI) which can support internationalised character sets as defined in RFC3987)
   - Submitted to IETF URI review list: http://www1.ietf.org/mail-archive/web/uri-review/current/index.html

- Inevitably there are some slight differences between the IBM SOAP/JMS support implemented in WMQ v6, and the standardised version of SOAP/JMS.
  - These are diffences in the IRI and the message formats
  - Once the SOAP/JMS standard has been published, IBM products will need enhancements to implement the standard

- It's important to note that, despite its' name, the SOAP/JMS specification is not restricted to Java and JMS. It enables <u>interoperable</u> access to services over MQ from and to any languages
  - Because MQ JMS message formats are all published, non-Java Applications can construct JMS messages to invoke services by building the correct RFH2 headers
    - e.g. COBOL apps building JMS messages, .NET support in v6

Part 2b - WCF Custom channel for MQ

- Windows Communication Foundation (WCF) underpins Web services and Messaging in .NET 3

  - Built-in Transports e.g. MSMQ, HTTP(S), Named Pipes, TCP/IP, etc.

  - Transports can be extended with 'custom channels'

  - Allows alternative transports (like MQ) to be slotted into WCF seamlessly

- IBM has now released a custom channel for MQ

  - Initially released on AlphaWorks June 2007

  - Now available in WMQ v7.0.1

  - Uses IBM SOAP/JMS message formats

    - For interoperability with WAS, CICS SOAP/JMS services

WCF Architecture

- Primary focus of WCF is for service orientated architectures
  - Simplifies integration by separating the roles of transporting and encoding data
    - Transport channels (HTTP, TCP/IP…)
    - Message encoders (Binary, XML,SOAP, MTOM…)
    - Protocols (WS-*…)
  - WMQ is integrated as a transport channel

WCF Client Application
WCF Services Layer
WCF Channel Stack
Protocols..
Message Encoder
WMQ Transport

WMQ

WCF Service Application
WCF Services Layer
WCF Channel Stack
Protocols..
Message Encoder
WMQ Transport

WCF Architecture

3 → WCF Custom Channel for WMQ

2 → XMS .NET

1 → NMQI (used to be WMQ .NET Base Classes)

| local MQ bindings | MQ client | .NET unmanaged connections | .NET managed client connections |

MQ Queue Manager

Usage Pattern 1

- Interface to web services hosted over WMQ (SOAP/JMS)

jms:/queue?destination=myQ…

.wsdl

WCF Application

WCF Services Layer

WCF Channel Stack

Protocols..

Message Encoder

WMQ Transport

WMQ

| WMQ SOAP/JMS | SOAP Layer | Web Service | CICS |

| WMQ SOAP/JMS | SOAP Layer | Web Service | WAS |

| WMQ SOAP/JMS | SOAP Layer | Web Service | .NET 1/2 (.asmx)* |

| WMQ SOAP/JMS | SOAP Layer | Web Service | Axis (Java) |
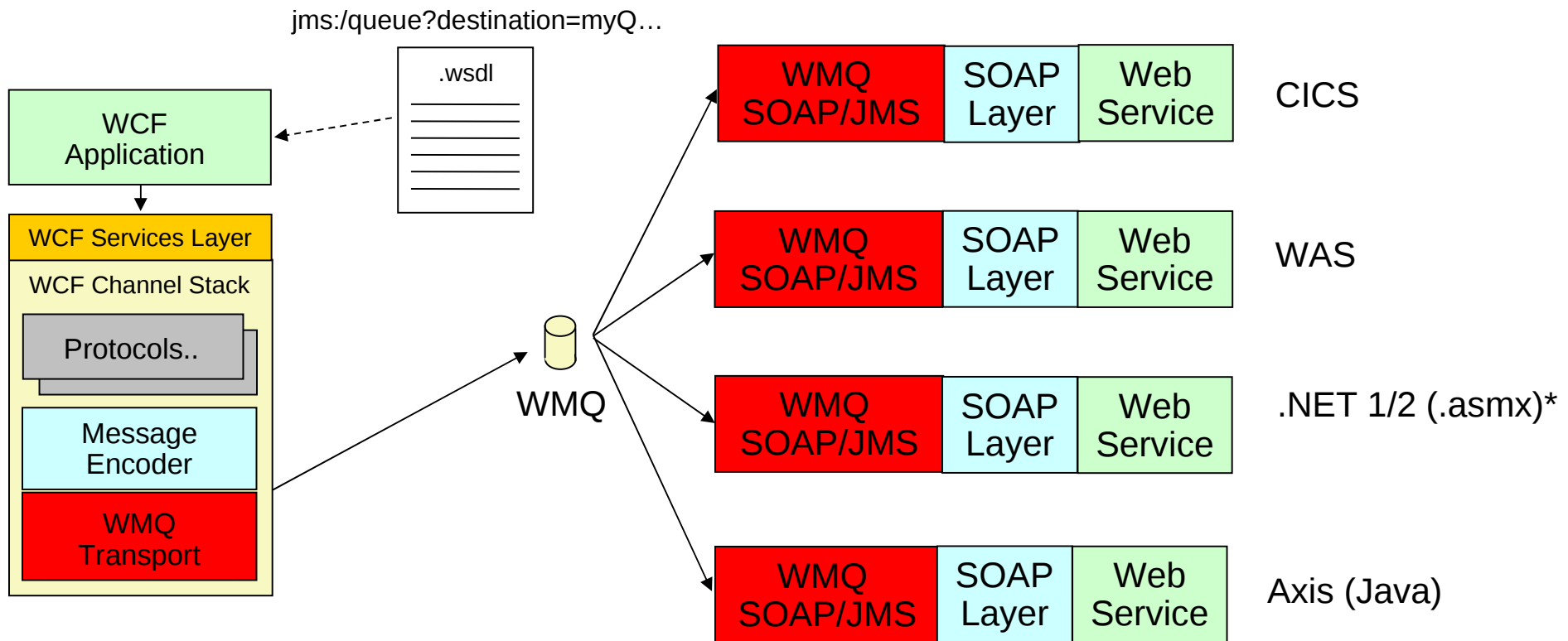
▪ Extending to other application servers with adoption of W3C SOAP/JMS
  standard (BEA, Sonic, TIBCO, Axis)

Usage Pattern 2

- Interface to native WMQ applications which are described as services
  - WMQ Service Definition

WMQ Service Definition

wmq:/msg/queue/myQ?priority=4&…

.wsdl

WCF Application

WCF Services Layer

WCF Channel Stack

Protocols..

Message Encoder

WMQ Transport

WMQ

Native WMQ Application
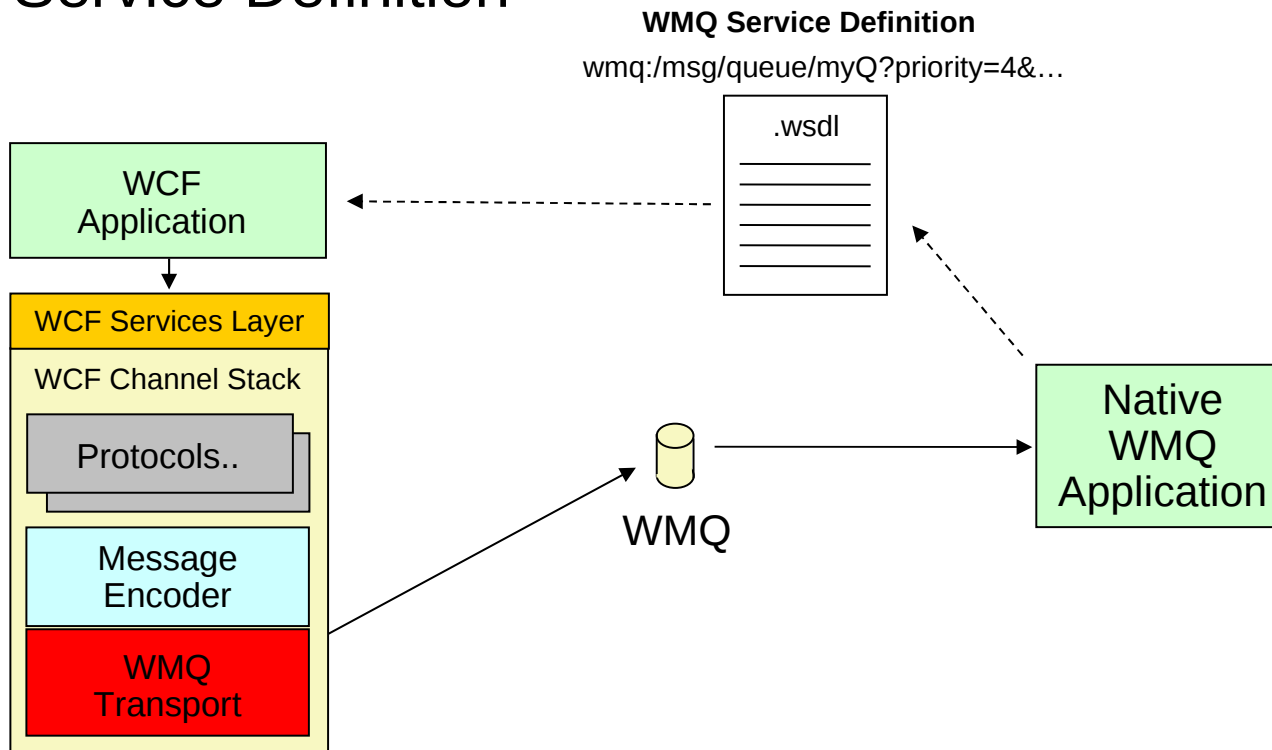
Usage Pattern 3

- As a standalone messaging API
  - CreateChannel(), Send(), Receive().
- WCF Channel stack used directly by applications (not through the services layer)
- Likely to require an additional message encoder
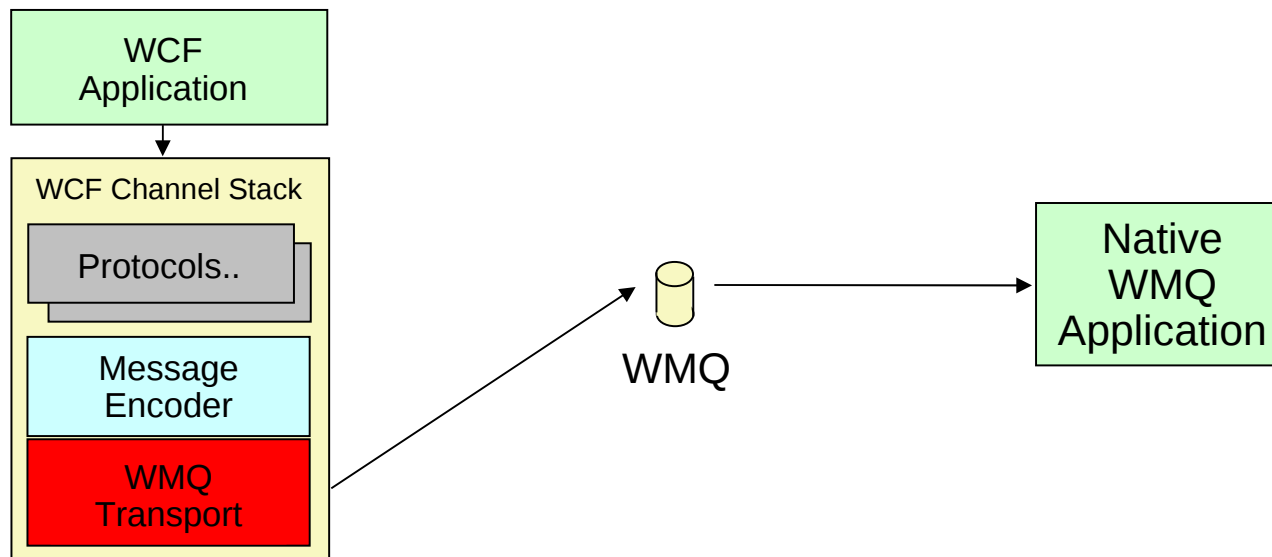
WCF Custom channel

- The work IBM is doing on using MQ for the Windows Communication Foundation is an example of how SOAP/JMS can be used outside Java

  - The Windows Communication Foundation (aka Indigo) provides the core for all .NET connectivity
    - Available for Windows XP
    - An integral part of Vista
  - Microsoft supply a number of built-in transports
    - MSMQ, HTTP(S), Named Pipes, TCP/IP, etc.
    - Can be used with .NET bindings to invoke .NET services
    - Or with SOAP bindings for interoperability with other services
    - The communication framework is extensible

- We have requirements to provide a WMQ transport for WCF
  - Allowing seamless WMQ integration in Windows

- This will allow .NET to .NET communication using WMQ as a Web services transport, and, because it uses the SOAP/JMS message formats and IRI, it will also allow WCF applications to communicate with SOAP/JMS services deployed to CICS, WAS etc.

WCF Custom channel Features & Status

- Available in WMQ 7.0.1
- Features :
  - Can call a service using One-Way (Fire and forget) and Request-Reply MEPS
  - SOAP/JMS or POX messages
- Samples for:
  - Calling Request-Response, and One-way WCF services
  - Calling a sample Axis service hosted by WMQ v6
  - Calling a sample .NET service hosted by WMQ v6
  - Calling a sample service hosted in WAS
- Dependencies
  - .NET Framework v3 or 3.5 runtime & SDK
- Limitations:
  - Does not support distributed transactions

MQ Web Services Themes

1. Easy Access from Web Clients

➢ *WebSphere MQ Bridge for HTTP*
  • Web *Client* side access to WMQ

➢ *JSP/Servlets & "SAM" PHP API*
  • Web *Server* side access to WMQ

Web 2.0
REST, AJAX, JSON

3. Managing WMQ Apps as Services

➢ *WMQ Service Definitions*
  • .wsdl for WMQ Applications

Enterprise Messaging Backbone

WebSphere MQ
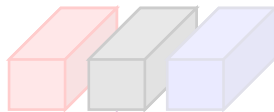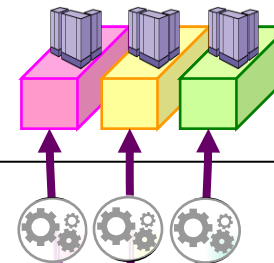
2. Robust Async SOAP Transport

➢ *SOAP / JMS Standard*

➢ *WMQ Channel for Windows Communication Foundation*

Reusing and Governing MQ Applications

- Requirement for a standard to describe MQ apps as SOA assets

  - Allowing stand-alone WMQ applications to be described as services:

    - To be inventoried, and catalogued in Service Registry
    - To be re-used as services in composite SOA applications
    - To be managed and traced with SOA tools

- IBM has specified the MQ Service Definition and SOAP binding

  - URI for MQ Queues and Topics ("wmq:")

  - WSDL bindings

  - Defines applications properties, and where and how to send messages

    - E.g. the Message Exchange Pattern; Request queue; Response queue; Correlation style; Message format; Message persistence, priority etc.

- This is a published specification

  - WMQ SupportPac MA93

WMQ Service Descriptions

- MQ users have requested guidance from IBM on how they should describe their MQ applications as services for use in service oriented architectures.

- There has been particular interest in applying this to unmanaged native WMQ applications (i.e. those coded to the MQI - not JMS - and those running outside of an application server / CICS etc.)
- This will allow applications to:
  - Be inventoried, and catalogued in Service Registry. For example, the WSDL description of an application can be stored in WSRR
  - Be managed and traced with SOA tools. which - for example – will be able to monitor the queues associated with a service
  - Be re-used in composite applications. For example, once the MQ service definition has been implemented by web services tools, it will be possible to drop an MQ application into a composite Web services application, and the tools will generate the code required to invoke the MQ application

- IBM has creating the MQ service definition specification to address this
- This consists of two documents.
  - An **IRI specification**, which defines :
    - The address of a WMQ message destinations i.e. *Queues* or *Topics* for use by messaging applications
    - The address of other WMQ resources i.e. *Qmgrs*, *Queues*, *channels*, *channel status* etc. for use by admin tools
  - **A Bindings Specification**, which defines :
    - Properties which may be used to describe and connect to a WMQ app.
    - The mapping of properties to message headers for the construction and interpretation of SOAP and non-SOAP messages
    - Supported message exchange patterns
    - A WSDL binding for SOAP/WMQ and non-SOAP/WMQ
    - Examples of IRIs, Messages, and WSDL documents

- This is published in SupportPac MA94
  - No immediate plans to add tooling support in the base MQ product, but it will provides a standard to be implemented by IBM and vendor tooling products
  - Expect this to be picked up first by Message Broker and WebSphere Service Registry and Repository

Why use WMQ service definitions?

- Catalogue MQ applications existing within your organisation
- Increase application re-use
- Reduce the effort required to manage and govern MQ applications and resources
- Identify which resources your MQ applications are using

  - Who will be affected if I:
    - Delete a queue?
    - Shutdown a queue manager?

Using WMQ service definitions

Architect

Writes service definitions
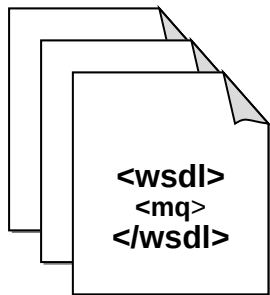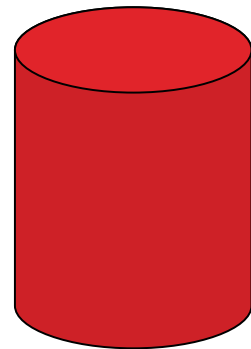
Service Developer

Writes services based on service definitions

Application Developer

Looks up existing services, connection details etc. to develop new applications

```
<wsdl>
  <mq>
</wsdl>
```

Repository

Integration Administrator

Manages existing applications, performs impact analysis, generates application statistics etc.

MB Flow

Runtime Clients & Message Broker Flows

Look up connection details at runtime, allowing services to be re-located without re-writing client apps

SHARE
Technology · Connections · Results

SHARE
in Orlando
2011
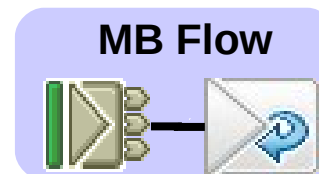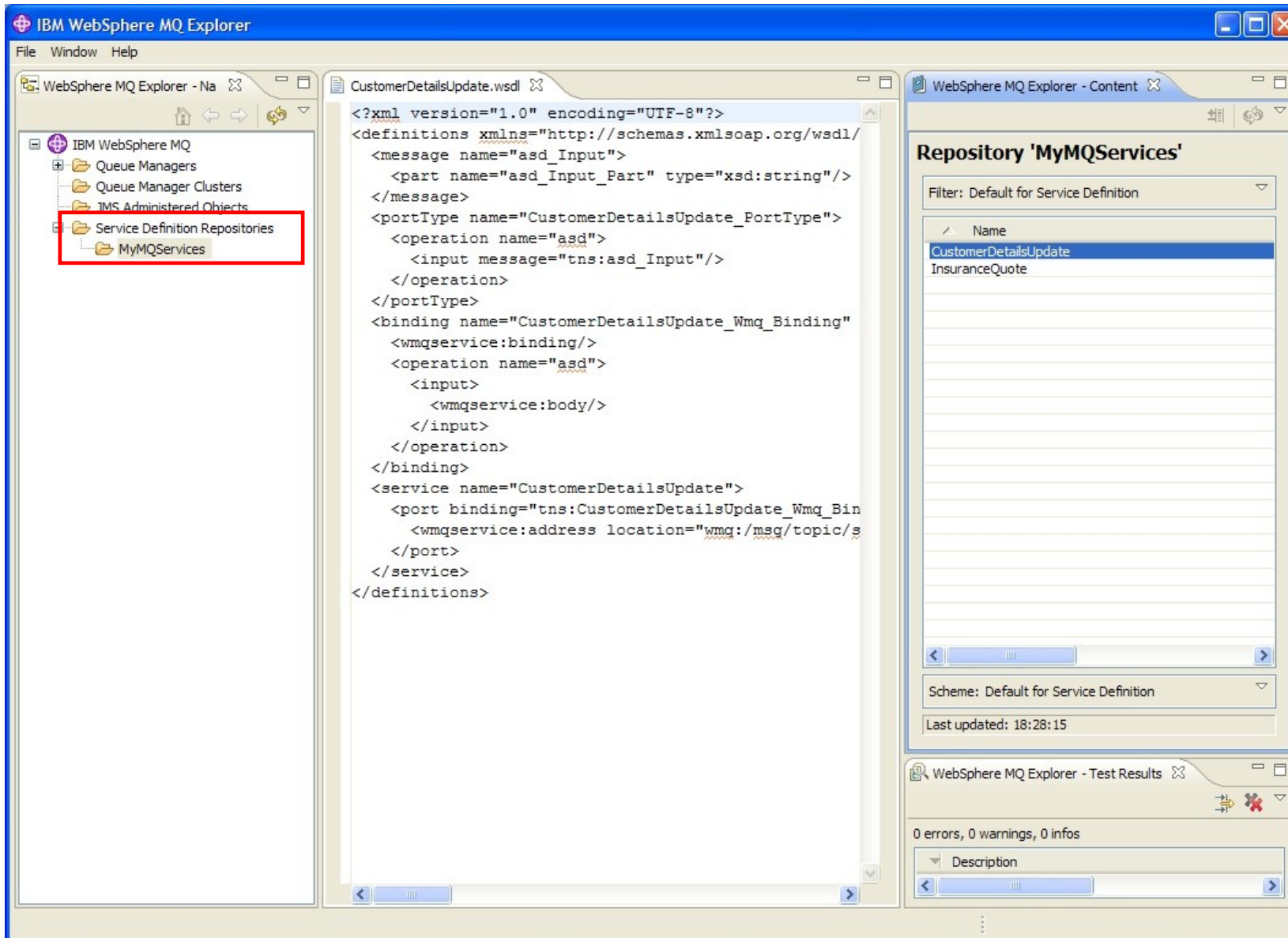
Notes

•Service definitions provide a centralised point which can be used to decouple all aspects of the software development lifecycle.

•Typically, this would start with the architect developing a set of interface definitions for the services needed to meet a particular requirement.

•These would then be picked up by the development team to provide an implementation. In parallel, the application developer, can start to bring the together the individual services into a composite application based on the published interface information and endpoint/connection details. This could be by using high level tooling to create process flows, such as WebSphere Message Broker, or more directly into Java/.NET applications.

•In more advanced scenario, the binding of services into the composite application could be performed at runtime, where for example, WebSphere Message Broker could be used to look up the endpoint information at runtime to determine which instance of a service to bind to – providing a greater level of flexibility and agility in managing systems.

•The final aspect is then the monitoring and control that can take place. By using the published data, impact analysis can be performed, such as what services would be effected if I took this server out for maintenance. Statistics can also be collected to help with monitoring utilisation, performance or accounting.

•The WMQ service definition enables WMQ applications to be managed in exactly the same way as any other service above.
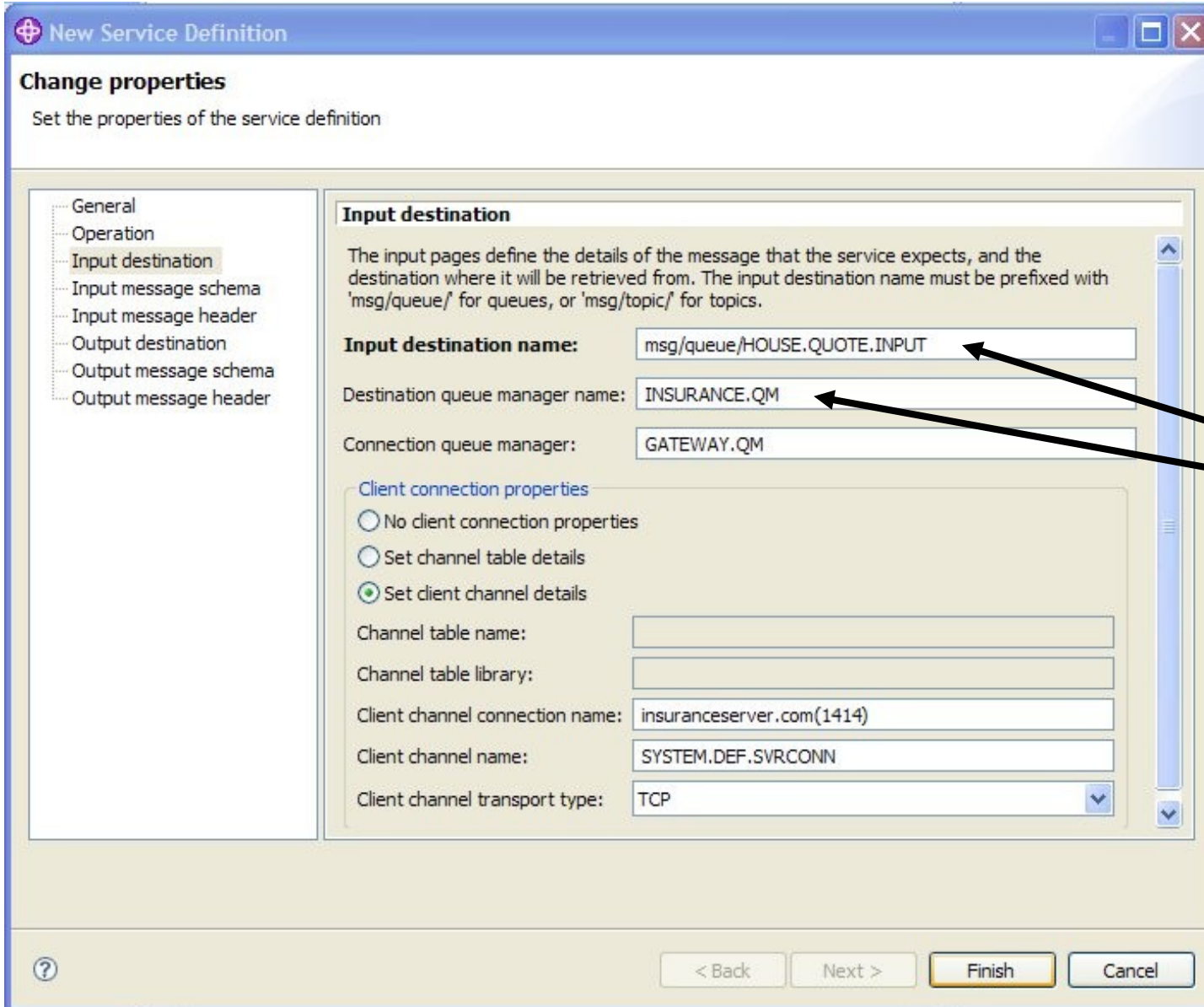
WebSphere MQ Explorer Service Definitions

Service Definition Wizard

Enter queue name, queue manager details etc.

Notes

N
O
T
E
S

- Service Definition feature available with WebSphere MQ Explorer since WMQ v7.0.0.1
    - Provides a wizard which allows you to generate WMQ service definition WSDLs describing one-way and request-reply services
    - WSDLs produced can be consumed by WebSphere Service Registry and Repository 6.2.0.2
    - Precludes the user from having to know anything about WSDL

Example – Native WMQ Request-Response MEP (WSDL)

```
<definitions ... xmlns:wmqservice="http://www.ibm.com/2007/02/service/bindings/WMQ/">
 <!-- Message and portType definitions are all standard WSDL -->

 <!- Binding – WMQ Native-->
  <binding name="mq_insurance_bindings" type="tns:portType1">
    <wmqservice:binding/>
    <operation name="getQuote">
       <input>
          <wmqservice:body/>
       </input>
       <output>
          <wmqservice:body/>
       </output>
    </operation>
  </binding>
  <service name="InsuranceServices">
    <port name="mq_ports" binding="tns:mq_insurance_bindings">
      <wmqservice:address location=" wmq:/msg/queue/INS.QUOTE.REQUEST"/>

      <wmqservice:persistence>MQPER_NON_PERSISTENT</wmqservice:persistence>
     <wmqservice:format>MQSTR</wmqservice:format>
     <wmqservice:reportOptions>MQRO_PASS_MSG_ID</wmqservice:reportOptions>
     <wmqservice:replyTo>msg/queue/INS.QUOTE.REPLY</wmqservice:replyTo>
    </port>
  </service>
</definitions>
```

3

1

2

Example – Native WMQ Request-Response MEP (WSDL)

- This example shows a native MQ request/response application which processes an insurance quote request and place a response on a reply queue
- The service Interface (portType) and Message definitions are not shown in this example because the MQ Service Definition specification does not extend or change WSDL's intrinsic XML Schema-based type system.
  - Note, this means that WMQ messages encoded in non-XML data – e.g. COBOL copybooks – must be described in WSDL using XML schema. A standard set of schema annotations and rules - called Data Format Description Language (DFDL) - is being developed to extend XML schema to non-XML formats and should be used once ratified, but until it is ratified, proprietary schema annotation can be used – (e.g. the ones used by WebSphere Message Broker).
- Looking at the WSDL in more detail:
- The service definition (1) contains the addresses of the request queue (`wmq:/msg/queue/INS.QUOTE.REQUEST`) and response queue (`msg/queue/INS.QUOTE.REPLY`) and defines the bindings to use (`mq_insurance_bindings`). The bindings section (2) is identified as an MQ transport binding by the presence of the <wmqservice:binding/> element. It specifies that the application must be accessed using the qualities of service defined by the elements shown in (3) – i.e. Persistent messages, with an MQ String format. It also specifies that the responding MQ Service Application correlates the response message to the request message by copying the request MsgId into the Response MsgId (MQRO_PASS_MSG_ID)

- The Application can either be described using the WSDL shown on the previous page, or alternatively it can be completely described using the following IRI.

  wmq:/msg/queue/INS.QUOTE.REQUEST?
  &replyTo=msg/queue/INS.QUOTE.REPLY
  &format=MQSTR
  &persistence=MQPER_NON_PERSISTENT
  &reportOptions=MQRO_PASS_MSG_ID

- The IRI definition is a more compact version of the WSDL definition, but they are equivalent.
- The presence of the *replyTo* parameter tells the invoking implementation to expect a response (In the IRI this MEP could also be indicated by a "msgType=MQMT_REQUEST" parameter, in which case the replyTo parameter could be omitted and the implementation would be expected to provide information about the replyTo destination – perhaps by using a temporary queue)
- The *format* and *persistence* parameters specify that the messages are non-persistent text messages.
- The *reportOptions* parameter specifies that the responding application should set the messageID of the response to the messageID of the request message (in addition to the default behaviour of copying the request messageId to the correlId of the response).

Impact Analysis for INS.HOUSE.QM
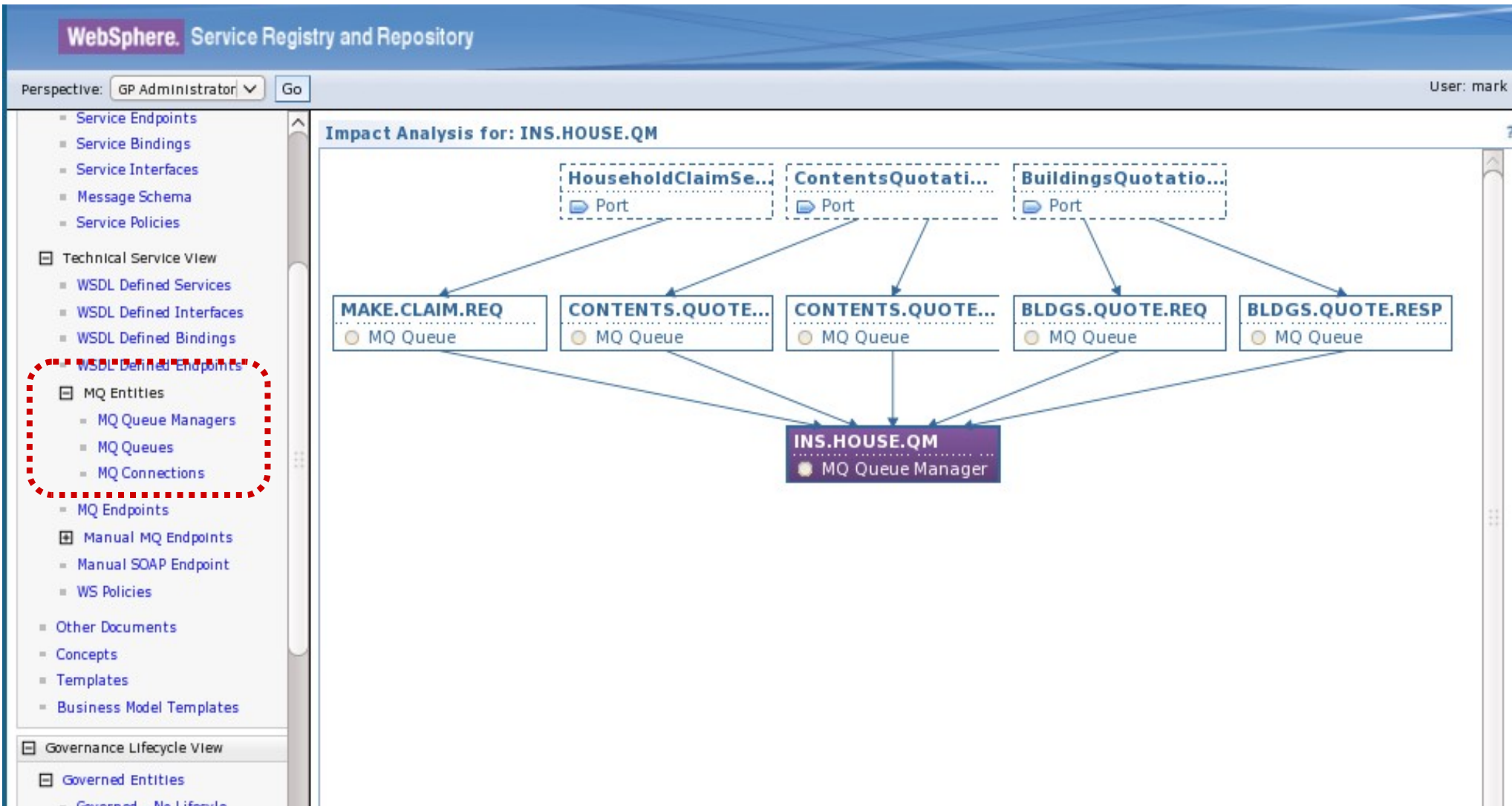
Impact Analysis

By defining MQ applications as services using a standard WSDL spec, it is possible to view the dependencies between abstract services and concrete components of your infrastructure

 - Which queues does a given service depend on?
 - Which services are hosted by a particular queue manager?

These are difficult questions to answer if applications aren't described and catalogued in a consistent manner

Summary – MQ, the Web, and Web Services

- **MQ can be used as a Web 2.0 enabler**

  - Extending the reach of Enterprise applications to the Web
- **MQ HTTP can be used for "Zero Footprint Clients"**

  - Server, Client , and Device applications can communicate over HTTP

  - No need to deploy client libraries

  - Extends MQ to many more client environments
- **MQ can be used as a Web services transport**

  - Using SOAP over JMS standard, or by sending SOAP messages in raw MQ messages

  - As a integrated custom channel in the Windows Communication Foundation
- **WMQ based applications can be described and re-used in Service Oriented Architectures**

  - Describe WMQ applications in a standard way like other Web service assets using WSDL and IRI's

  - Store in Service Registry, and use tools to access

Copyright and Trademarks