Programming Models for Service Oriented Architecture

Paul Fremantle
Senior Technical Staff Member
IBM Hursley Park
Who Am I?

- pzf@uk.ibm.com
- Lead development architect for
  - IBM’s Web Services Gateway
  - C/C++ Web Services Toolkit
- This session will discuss how to successfully program applications as part of a wider Service Oriented Architecture
What Is a Service Oriented Architecture?

- An **approach** for **building** distributed systems that deliver application functionality as **services** to either end-user applications or other services.

- It defines:
  - An architecture that leverages open standards to represent software assets as services.
  - Provides a standard way of representing and interacting with software assets.
  - Individual software assets become building blocks that can be reused in developing other applications.
  - Shifts focus to application assembly rather than implementation details.
  - Used externally to integrate with applications outside of the enterprise.
Shift to SOA

From
- Function oriented
- Build to last
- Prolonged development cycles
- Application silos
- Tightly coupled
- Object oriented
- Known implementation

To
- Process oriented
- Build to change
- Incrementally built and deployed
- Orchestrated solutions
- Loosely coupled
- Message oriented
- Abstraction
Business Level SOA

- **Business Services**
  - are goods or services that a business component offers to other business components and/or to external parties
  - are what’s directly visible outside the component

A business process can be represented as collaboration among components.

- **A Business Component**
  - Discrete boundaries
  - Includes the resources, people, technology
  - Is ‘black box’
  - Provides logical ‘cleave points’
  - Has Attributes

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Market Segment Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>To analyze markets and derive targets</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
Current View of a Service Interaction

requester  wire  provider
Programming Models and Web Services

- Isn’t Web Services just about what’s on the wire?
  - Yes…. Today
  - But SOA is not just about the wire
    - Good boundaries between business logic
    - More efficient integration and re-use
What Is a Programming Model?

- The application programming interface (API)
- Service programming interfaces (SPI)
- Configuration and administration model
- Separation of concerns

- Differing views!
  - J2EE takes a prescriptive view (aiming for full portability)
  - Other people mean for example
    - Stub vs DII (Dynamic invocation interface)
What Are the Programming Models That Exist So Far?

- Java:
  - Apache SOAP
  - Apache Axis
  - JAX-RPC, Enterprise Web Services (109), J2EE
  - Glue

- C
  - gSOAP, Axis C/C++

- .NET
Examples – Axis

AddressBookService abs = new AddressBookServiceLocator();
AddressBook ab1 = abs.getAddressBook();
Address mine = ab1.lookup("pzf");
JAX-RPC and Enterprise Web Services (EWS)

- **JAX-RPC**

```java
ServiceFactory serviceFactory = ServiceFactory.newInstance();
Service service = serviceFactory.createService("my.wsdl", qName);
AddressService myProxy = (AddressService)
    service.getPort(AddressService.class);
myProxy.updateAddress(addressBean, 75005);
```

- **Enterprise Web Services + JAX-RPC**

```java
InitialContext ic = new InitialContext();
Service service = ic.lookup("java:comp/env/AddressService");
AddressService myProxy = (AddressService)
    service.getPort(AddressService.class);
myProxy.updateAddress(addressBean, 75005);
```
WSIF Dynamic Proxy

```java
WSIFService sq = ServiceFactory.newInstance().
    getService("http://my.com/svcs/stockquote.wsdl");
QuoteService quote =
    (QuoteService)sq.getStub("soap", QuoteService.class);
Float value = quote.getQuote("IBM");
```
Web Services Invocation Framework (WSIF) DII

```java
WSIFService sq = ServiceFactory.newInstance().
    getService("http://my.com/svcs/stockquote.wsdl")
WSIFPort defPort = sq.getPort();
WSIFOperation getQ =
    defPort.createOperation("getQuote");
WSIFMessage inMessage =
    getQ.createInputMessage();
inMessage.setStringPart("symbol", "IBM");
...
getQ.executeRequestResponse(inMessage, outMsg, fltMsg);
outMessage.getFloatPart("value");
```
.NET

- IFoo proxy = ProxyFactory.CreateProxy<IFoo>("service URI");
- int n = proxy.Bar(3);
Lessons Learnt...

- Axis – simple practical, but very Axis specific
  - Little abstraction from implementation classes
  - Portability only by running Axis somewhere else!

- EWS/JAX-RPC
  - Good, but very focussed on “stub” model
  - Client-server model
  - No standard way to pass SOAP XML or Envelope

- WSIF DII + JROM
  - Excellent model of WSDL + XML
  - Perhaps a little too faithful!

- .NET
  - Easy to code
  - A little too much QoS in the code
What’s Changing This?

- Composability

- Business Process Execution Language

- WS-Addressing
  - WS-ReliableMessaging / WS-Reliability
  - WS-AtomicTransaction, WS-BusinessActivity
  - WS-Security

- WS-Policy
  - WS-MetadataExchange
# The Extended Web Services Platform

<table>
<thead>
<tr>
<th>Service Composition</th>
<th>WS-Notification</th>
<th>BPEL4WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality of Experience (QoX)</td>
<td>WS-Reliable Messaging</td>
<td>WS-Transaction</td>
</tr>
<tr>
<td></td>
<td>WS-Security</td>
<td>WS-Resource Lifetime</td>
</tr>
<tr>
<td>Description</td>
<td>WS-Resource Properties</td>
<td>WS-Metadata Exchange</td>
</tr>
<tr>
<td></td>
<td>XSD</td>
<td>WSDL</td>
</tr>
<tr>
<td>Messaging</td>
<td>XML</td>
<td>SOAP</td>
</tr>
<tr>
<td>Transports</td>
<td>HTTP/HTTPS</td>
<td>SMTP</td>
</tr>
</tbody>
</table>
What’s REALLY Changing This

- Enable “dynamic” e-business
  - In IBM-speak “On Demand Computing”
- Quickly host new business processes
- Incorporate new partners as needed
- Add the right quality of service
  - (reliable, transactional, secure)
  - As needed
Virtualisation

- The ability
  - to swap implementations as simply as possible
    - Change a constant in a program = NOT BAD
    - Redeploy a program with a new config = OK
    - Change a UDDI entry and restart = GOOD
    - Administer a change to take place at 00:01 on Weds morning – GREAT
  - to change quality of service
  - to use the same service in different business and technical contexts
How?

- Re-target an existing service or process to use new providers
  - Intermediaries and directories allow new service providers to be added
  - BPEL allows new partner links to be created dynamically
  - WS-MetadataExchange allows services to be queried for the WSDL and WS-Policy
Composability

Transactions

Reliable Messaging

Security

Basic Transport
Composable Headers

```xml
<S:Envelope ...>
  <S:Header>
    <wsa:ReplyTo>
    </wsa:ReplyTo>
    <wsa:To>http://fabrikam123.com/Traffic</wsa:To>
    <wssec:Security>
      <wssec:BinarySecurityToken
        ValueType="wssec:X509v3"
        EncodingType="wssec:Base64Binary">
        dWJzY3JpYmVyLVBlc.....eFw0wMTEwMTAwMD
      </wssec:BinarySecurityToken>
    </wssec:Security>
    <wsrm:Sequence>
      <wsu:Identifier>http://fabrikam123.com/seq1234</wsu:Identifier>
      <wsrm:MessageNumber>10</wsrm:MessageNumber>
    </wsrm:Sequence>
  </S:Header>
  <S:Body>
    <app:TrafficStatus
      xmlns:app="http://highwaymon.org/payloads">
      <road>520W</road><speed>3MPH</speed>
    </app:TrafficStatus>
  </S:Body>
</S:Envelope>
```
WS Policy

- Assertions
  - An individual capability, requirement etc
    - e.g. `<wsse:SecurityToken TokenType="wsse:Kerberosv5TGT" wsp:Usage="wsp:Required" wsp:Preference="100"/>`
  - Assertions are defined in “plug-in” languages
    - such as WS-SecurityPolicy
- Assertions are grouped into Statements
  - `<wsp:ExactlyOne>
    <assertion1>
    <assertion2>
  </wsp:ExactlyOne>`
- Policy Attachment defines how to link from
  - A WSDL document or UDDI entry to a policy
WS Policy (Continued)

- Operators
  - ExactlyOne, OneOrMore, All

- For example
  - *WS-CoffeePolicy™ – in order to talk to me in the morning I require*
  - Double Cappuccino with an extra shot of espresso ☻ ☻
    - or
  - Double Cappuccino ☻
    - Or
  - Large Mocha Java
    - but not
  - Instant, Motel filter, Tea, Decaffeinated......
Lesson #1 – The Right Contracts Matter

- Implementing flexible QoS means
  - separating the QoS code from the business logic

```
requester code | requester stack | wire | provider stack | provider code
A               | B                | C    | D                
```
Lesson #2

- Two different models:
  - Java → WSDL type of people
    - Basically using a stub model to do distributed computing
    - Using Java / C / C++ / perl / CLR types
  - XML type of people
    - Using existing XML schemas or DTDs
      - e.g. OAGIS
    - Business process and integration focused
Lesson #2a

- Contract first
  - Start with WSDL and schema

- Improves interoperability
  - (if you use a fairly simple subset of XSD)
  - See lesson #5!

- Makes for better design
  - More re-usable

- Can be a refactoring of code-first
Lesson #3 – Don’t Hardcode URLs

- **EWS**
  
  ```java
  Service s = ic.lookup("service-ref");
  ```

- **UDDI lookup**
  
  ```java
  Stub.setProperty(
      ENDPOINT_ADDRESS_PROPERTY, myUDDIHelper.getURLforKey(
          "C6410450-9976-11D8-A122-000629DC0A53")
  )
  ```

- **Inversion of Control Pattern**
  
  ```java
  public interface businessService { ... }
  public class businessApp {
      private businessService = null;
      public void setBusinessService(businessService bs) {
          this.businessService = bs;
      }
      ....
  }
  ```
Lesson #3a – Do hardcode URLs

- Hardcode a “virtual URL”

```
Stub.setProperty(ENDPOINT_ADDRESS_PROPERTY,
    “http://router.fremantle.org/StockQuoteService”);
```

![Diagram showing the flow from App to router.fremantle.org to StockQuote service]
Lesson #4 – Split Your Logic between Application and Handlers

- The SOAP message maps directly into your logic:
  - `<S:BODY>` is the domain of your business logic
  - `<S:HEADER>` is the domain of the middleware – enhanced by handlers

- What can you do in handlers?
  - Add/read/modify headers
  - Modify the target URL of the request (e.g. “target.url”) in WebSphere
  - Log the message
  - Look at the SOAPMessage
Lesson #5 – If You Are Doing XML, Do XML

- Taking existing XML schemas and DTDs and trying to map them to Java types does NOT work!
  - Most stacks support a way of passing an XML object to the service invocation
    - See Axis Message example
    - WebSphere has a Call.invoke(SOAPEnvelope) — unsupported so far
    - WSIF can use DOM objects
    - Service Data Objects and XMLBeans offer nicer interfaces than SOAPElement / DOM
  - If you are really stuck you can do it in a handler
Using XML

- App
  - SOAP data
  - SOAP Element
  - XMLHelper

- Call object
- SOAP handlers
- SOAP/HTTP
WS-Addressing

- There are two main aspects:
  - #1
  - Assertion that each service may have “instances”, distinguished by EndpointReferences
  - For example, same WSDL but two URLs
    - http://my.com/StockQuote
    - http://fremantle.org/FreeStockQuote
  - An example EndpointReference
    ```
    <wsa:EndpointReference>
        <wsa:Address>http://www.fremantle.org/FreeStockQuote</wsa:Address>
        <wsa:PortType>stock:StockQuoteInterface</wsa:PortType>
    </wsa:EndpointReference>
    ```
#2. SOAP messages may have more complex transport patterns than a single HTTP connection:

```xml
<S:Envelope>
  <S:Header>
    <wsa:MessageID>uuid:aabb-cc-dd-ee-fgf</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>
        http://requester.example/client1
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:To S:mustUnderstand="1">
      mailto:joe@fabrikam123.example
    </wsa:To>
  </S:Header>
  <S:Body>...
```
Lesson #6 – Watch Out for Instances

- Beware instances!
  - What is the behaviour of the middleware?
  - At what point is the service instance selected?
    - Deployment
    - Lookup
    - Invocation
  - Affinity / Context
    - By the middleware – *e.g.* WebSphere Activity Service
    - Or with help from the programmer
Lesson #7

- Use intermediaries
  - Pass context as headers
  - Virtualize services via routers
  - Log, audit, filter, meter, monitor
  - Transform
  - Takes handler logic out of clients and servers
    - Making it easier to support .NET, Java, J2EE, Perl, etc.
  - Leaves running deployed applications untouched
Affinity and Context

- One way of ensuring affinity is to place a token into a header
  - Use the token to ensure affinity
  - `stub.setProperty("com.my.affinityToken", UUIDGenerator.getUUID());`
  - Affinity Handler grabs property and populates Header
  - Routing Handler may be in same stack or intermediary
    - Uses affinity header to ensure consistent routing
Reliable Messaging

- Three extra qualities of service
  - At most once
  - At least once
  - In order
- Plus – ExactlyOnce = AtMostOnce + AtLeastOnce
- Associated with the destination
RM + One-way Messaging

- Can be “composed” with existing application logic

```
createSequence
    Send #1
    Ack #1-#1
    Send #2
    Send #3
    Ack #1-#1,#3-#3
    Send #2
    Ack #1-#3
```
RM + Request-Response

- Cannot be composed with existing middleware in general
- Either timeout $= \infty$ or a statistical improvement: greater chance of a message working
Lesson #8 – Be Prepared for Reliability

- Think business process, not stub
  - Until the stub programming models get better 😊

- Decompose request response into two messages:
  - With some correlation information
    - either in a header,
    - or in the body
Decomposing Request Response

1. Create two WSDLs – each one way
2. Generate Java stubs and skeletons for each
3. Create handlers that can add / read the correlation information at each end
Client Request Handler

handleRequest(MessageContext mc) {
    if (mc instanceof SOAPMessageContext) {
        SOAPMessageContext smc = (SOAPMessageContext)mc;
        SOAPMessage m = smc.getMessage();
        SOAPHeader h = m.getSOAPPart().getEnvelope().getHeader();
        String messageID = smc.getProperty("message.id");
        h.addChildElement(WSA_URI, MESSAGEID_TAG)
            .addTextNode(messageID);
    }
}

Provider Request Handler

handleRequest(MessageContext mc) {
    if (mc instanceof SOAPMessageContext) {
        SOAPMessageContext smc = (SOAPMessageContext)mc;
        SOAPMessage m = smc.getMessage();
        SOAPHeader h = m.getSOAPPart().getEnvelope().getHeader();
        Iterator it = h.getChildElements();
        while (it.hasNext()) {
            SOAPElement thisEl = (SOAPElement)it.next();
            if (thisEl.getElementName.getLocalName().equals(MESSAGEID_TAG) &&
                thisEl.getElementName.getURI().equals(WSA_URI)) {
                smc.setProperty("message.id", thisEl.getValue());
            }
        }
    }
}
SessionEJB – provider1

- J2EE 1.4 adds
  - SessionContext.getMessageContext

- Now... we can extract the message ID, and then add it onto the next invocation back

- A third handler turns this into a header:
  
  <wsa:RelatesTo
   RelationshipType="wsa:Reply">uuid:aa-bb-cc
  </wsa:RelatesTo>
Overview of the Interaction

“Client”
- Creates uuid
- Creates entry
- setProperty

Persistent Object

Stub1

Session Bean2
- Implements Response logic
- Store uuid→busData

Handler1
- Property
- Header
  <wsa:MessageID>

SOAP

Handler4
- Header
- Property

Handler4
- Property
- Header
  <wsa:relatesTo>

Session Bean1
- Implements business logic
Lesson #8a – MEPs As Well

Message Exchange Patterns

http://www.w3.org/TR/wsdl20-extensions

- In-Only
- Robust In-Only
- In-Out
- In-Optional-Out
- Out-Only
- Robust Out-Only
- Out-In
- Out-Optional-In
Lesson #9

- Services are NOT objects
  - Coarse grained
  - Pass by value
  - Fundamentally data exchange

- Sounds simple, but this one takes a while to sink in!
Lesson #10

- Learn your own lessons...
  - These are some lessons that have been learnt from real projects, and from exploring and implementing the standards
  - Getting the value out of SOA is more about architecture than <s:xml>
  - Don’t be afraid to challenge existing programming practice to build more loosely coupled systems
Resources

- IBM developerWorks WebServices Zone
  - All the IBM / MS standards

- “The Hidden Impact of WS-Addressing”

- Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI (2nd Edition)
  - Publisher: Pearson Education
  - ISBN: 0672326418