JavaOne℠

Dealing with Asynchronicity in Java™ Technology-Based Web Services

Gerard Davison
Manoj Kumar
Oracle

Learn how to invoke and implement Asynchronous Services using WS-*

(No not AJAX, Comet etc)

Agenda

- Introduction
- Client side asynchrony
- Server side asynchrony
- Implementation of clients and services
- Declarative asynchronous services
- Advanced Topics
- Q & A

Introduction
Synchronous Call

```
Service service = ...;
StockQuote quoteService =
  service.getStockQuote();

float quote =
  quoteService.getPrice(ticker);
```

Introduction
Asynchronous Call

```
Service service = ...;
StockQuote quoteService =
  service.getStockQuote();

quoteService.getPriceAsync(ticker);

// Do some other work

// How and when do we get the response?
```

Introduction
Why Asynchronous?

- The world is fundamentally asynchronous
- Improvement in ..
  - User's experience
  - Reliability in distributed applications
  - Scalability
- Only way to go in long running processes
- Implementation
  - Client side asynchrony
  - Server side asynchrony

Introduction
Alternatives to JAX-*

- BPEL
  - Integration of other operations
  - Can be re-exposed as an asynchronous web service
- Asynchronous 3.1 EJB™ architecture
  - Java only
- JMS, MQueue, Native Database Queues
  - Generally platform specific
- SMTP
  - Golden oldie
- SCA

Client Side Asynchrony
JAXWS 2.x API

- Calling synchronous services in asynchronous way
- Not really synchronous though
- WSDL to Java customization option
  - <jaxws:**enableAsyncMapping**>true</jaxws:**ena…**>
- Asynchronous Interface
  - Polling
  - Callback
- Control of executors to manage the number of threads used

Client Side Asynchrony Synchronous Interface

```
@WebService
public interface StockQuote {
  float getPrice(String ticker);
}
```

Client Side Asynchrony
Asynchronous Interface

```
@WebService
public interface StockQuote {
  float getPrice(String ticker);

  Response<Float>
     getPriceAsync(String ticker);
}
```

Client Side Asynchrony Polling Style

```
StockQuote quoteService =
  (StockQuote)service.getPort(portName);

Response<Float> response =
  quoteService.getPriceAsync(ticker);

// do something else while the
// stock quote is in flight

float quote = response.get();
```

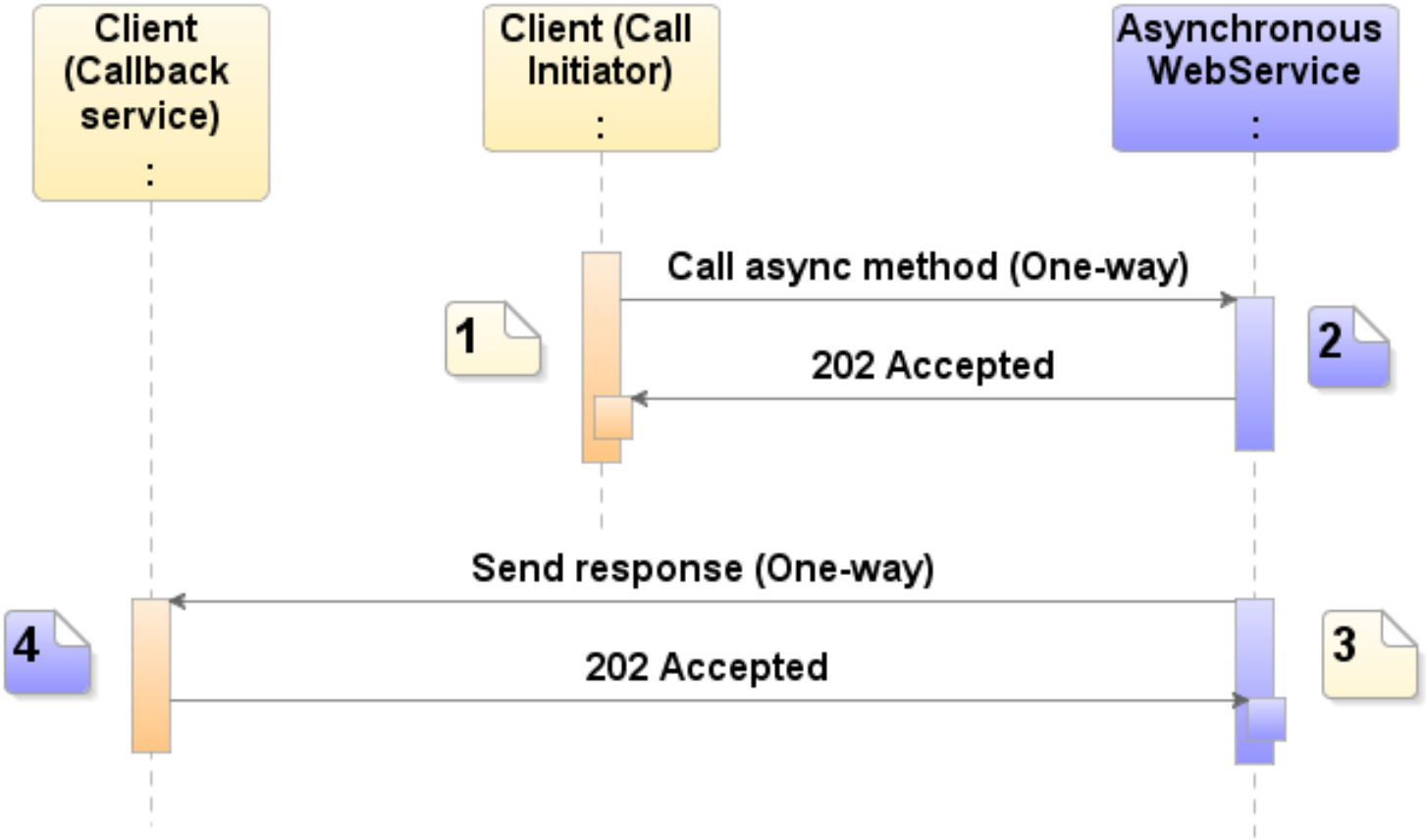
Client Side Asynchrony Asynchronous Interface

```
@WebService
public interface StockQuote {
  float getPrice(String ticker);

  Response<Float> getPriceAsync(String
      ticker);

  Future<?> getPriceAsync(String ticker,
      AsyncHandler<Float>
          responseReceiver);
}
```

Client Side Asynchrony Callback Style

```
PriceReceiver priceReceiver = new PriceReceiver();
quoteService.getPriceAsync(ticker, priceReceiver);
..
class PriceReceiver implements
      AsyncHandler<Float> {
  public void handleResponse(Response<Float>
                             response) {
      Float price = response.get();
      // do something with the result
  }
}
```

Server Side Asynchrony



Server Side Asynchrony

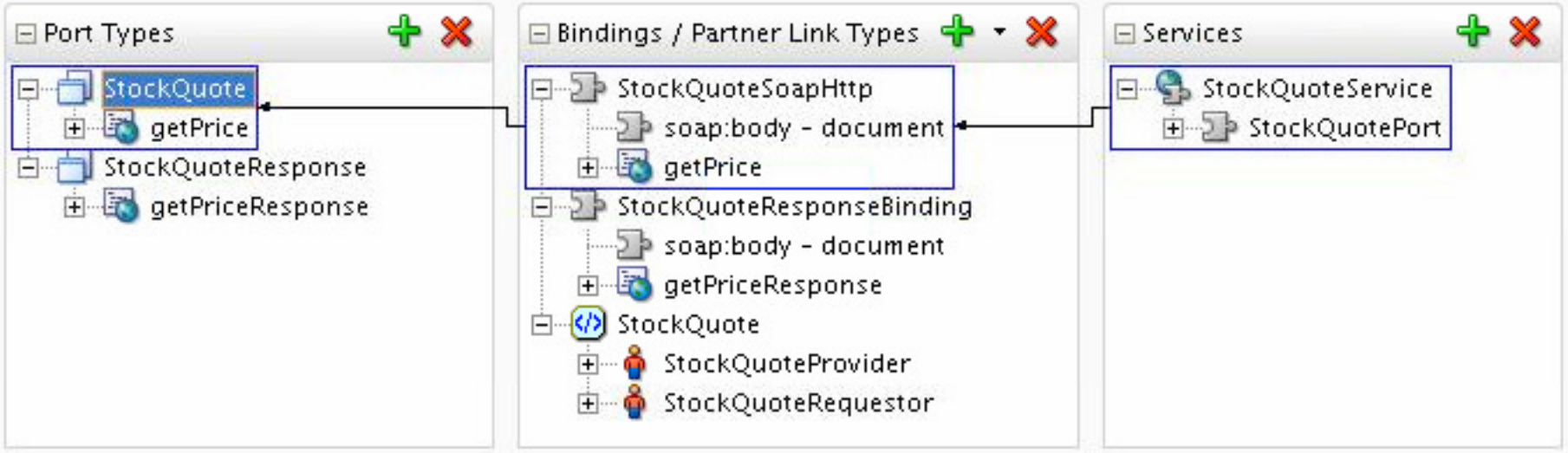
- Invocation and response separated
- A new connection for the call and the response
- The invoker and the receiver don't have to be on the same machine
- How it is implemented depends on the binding used

Server Side Asynchrony Web Service Bindings

- Using HTTP protocol
  - Can do polling but inefficient
  - Call and response are separated
  - But we need to layer something on top, WS-*
- Using JMS
  - Natively designed for asynchrony
  - Problems with firewalls
  - Only works for Java-2-Java
- Using SMTP

Server Side Asynchrony HTTP

- Generally has two port types, two binding; but one service



- Can use partnerLinkType to relate the two different ports
- Part of the BPEL specification; but an extension to WSDL

Server Side Asynchrony HTTP WS-Addressing

- Transport agnostic SOAP message delivery
- Endpoints references and message information headers
- Message information headers
  - Allow you to relate the message to a particular instance
- Endpoint references
  - Tell you where to send the message along with relevant metadata
- They can tell you where to reply to and send faults to, important for asynchronous services

Server Side Asynchrony HTTP WS-Addressing Reference Parameters

```
<soap:Envelope>
 <soap:Header>
  <wsa:MessageID>uuid:35f19ca8-c9fe</wsa:MessageID>
  <wsa:Action>http://lh:80/request/…</wsa:Action>
  <wsa:ReplyTo>
   <wsa:Address>http://lh:77/response</wsa:Address>
   <wsa:ReferenceParameters>
     <customHeader>correlationKey</customHeader>
   </wsa:ReferenceParameters>
  </wsa:ReplyTo>
  <wsa:To>http://lh:80/request</wsa:To>
 </soap:Header>
 <soap:Body>…</soap:Body>
</soap:Envelope>
```

Server Side Asynchrony HTTP WS-Addressing Reference Parameters

```
<soap:Envelope>
 <soap:Header>
  <wsa:To>http://lh:77/response</wsa:To>
  <wsa:Action>urn:response</wsa:Action>
  <wsa:MessageID>uuid:cb383139-cdf2</wsa:MessageID>
  <wsa:RelatesTo>uuid:35f19ca8-c9fe</wsa:RelatesTo>
  <customHeader wsa:IsReferenceParameter="1">
     correlationKey</customHeader>
 </soap:Header>
 <soap:Body>…</soap:Body>
</soap:Envelope>
```

Server Side Asynchrony Single Port

- You can using WS-Addressing do the same with a single port.
- This seen when using some .NET services
- For JAX-WS you end up having to hand edit the call back service
- http://kingsfleet.blogspot.com/2008/12/invoke-single-port-async-service-using.html

Server Side Asynchrony Client Implementation

- Create a proxy for the service
- Implement the callback binding to receive the response
- Deploy the callback web service

- Instantiate a proxy for the initiation endpoint.
- Set the MessageId property to a new unique value
- Set the To; ReplyTo; and FaultTo URI
- Invoke the web service

- In the callback web service use the "RelatesTo" header or reference parameters to correlate the response

Server Side Asynchrony Initiator Implementation with the RI

```
StockQuote sq = service.getStockQuote();

// Populate headers

AddressingVersion av = AddressingVersion.W3C;
WSBindingProvider wsbp = (WSBindingProvider)sq;
WSEndpointReference replyTo = new WSEndpointReference(
  "http://lh:77/response",av);
String uuid = "uuid:" + UUID.randomUUID();

wsbp.setOutboundHeaders(
  new StringHeader(av.messageIDTag, uuid),
  new StringHeader(av.toTag,"http://lh:88/…"),
  replyTo.createHeader(av.replyToTag);

sq.getPrice("ORCL");
```

Server Side Asynchrony Callback Implementation with the RI

```
// Deal with response

@Resource WebServiceContext wsContext;

public void getPriceResponse(String _return)
{
 HeaderList hl = wsContext.getMessageContext().get(
  JAXWSPRopertiers.INBOUND_HEADER_LIST_PROPERTY);
 Header h = hl.get(AddressingVersion.W3C.relatesToTag());
 String relatesToMessageId = h.getStringContents();

 …
}
```

Server Side Asynchrony FaultTo Implementation

- This should be easy right?
- Just create a @WebServiceProvider?
- There was a bug in JAX-WS RI (Issue 585, fixed in 2.1.5)
- You might end up having to use a Servlet if on an older version.

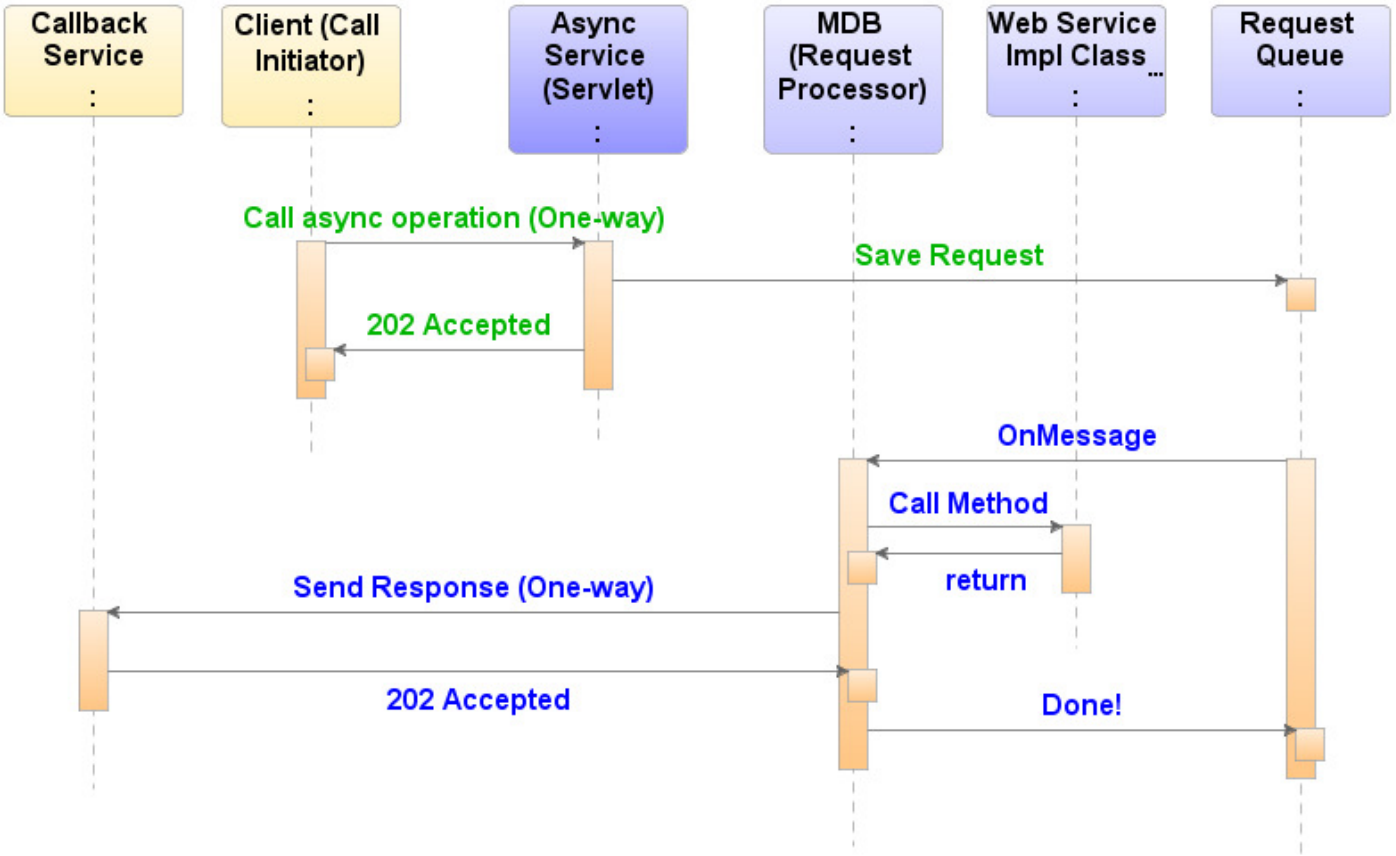
Server Side Asynchrony Service Implementation

- Accept the asynchronous request and save it (where?)
- Return 202 Accepted

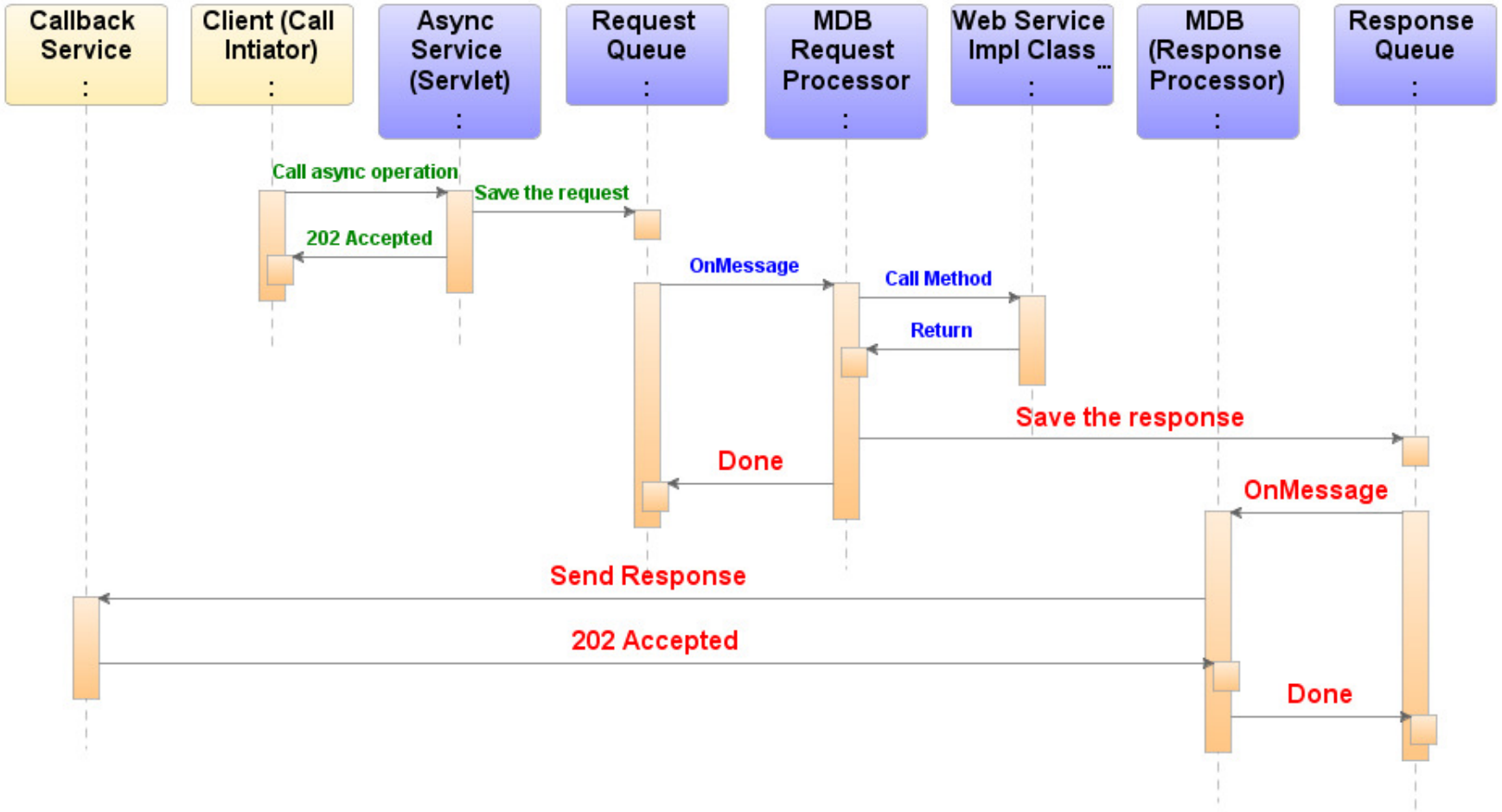
- Pick the request and process it
- Invoke the callback service and pass the result

- -RI does understand WS-Addressing, but response sent before request returns
- -RI AsyncProvider only suitable for short duration

Server Side Asynchrony
Using JMS Queue for incoming request



Server Side Asynchrony
Using response queue



Declarative Asynchronous Services

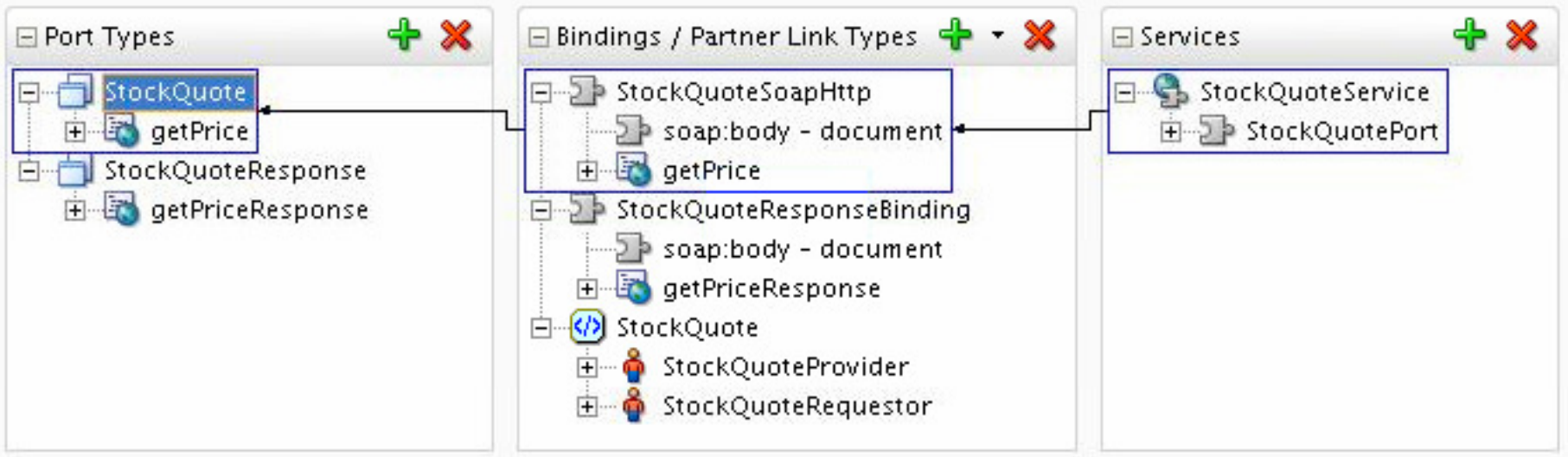
- Implementing an asynchronous service is not trivial
- But the basic concepts are….
- …. so how about a declarative model?
- We have been working on something in Oracle iAS 11, based on the JMS model

Declarative Asynchronous Services Simple Return Case


```
package com.stock;

import …;

@WebService
@AsyncWebService
@Addressing
public class StockQuote {
    public float getPrice(String ticker) {
        return 100;
    }
}
```

Declarative Asynchronous Services Simple Return Case

- The resultant WSDL, as before



Declarative Asynchronous Services Callback Case

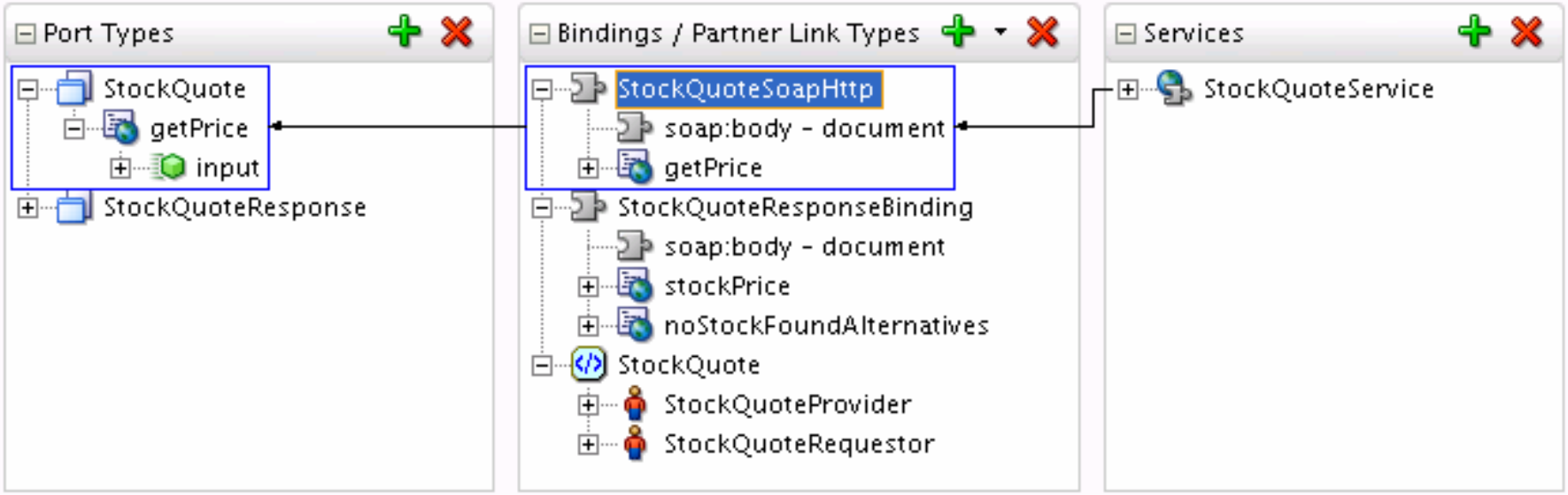
```
@WebService
@AsyncWebService
@Addressing
@CallbackInterface(StockQuoteResponse.class)
public class StockQuote {

  @CallbackRef StockQuoteResponse response;

  @OneWay public void getPrice(String ticker) {
    if ("ORCL".equals(ticker))
      response.stockPrice(100);
    else
      response.alternatives(new String[]
        {"ORCL", "JAVA"});
  }
}
```

Declarative Asynchronous Services Callback Case

```
package com.stock;

import …;

@WebService
public interface StockQuoteResponse {

    public void stockPrice(float price);
    public void alternatives(String alternatives[]);

}
```

Declarative Asynchronous Services Callback Case

- The resultant WSDL

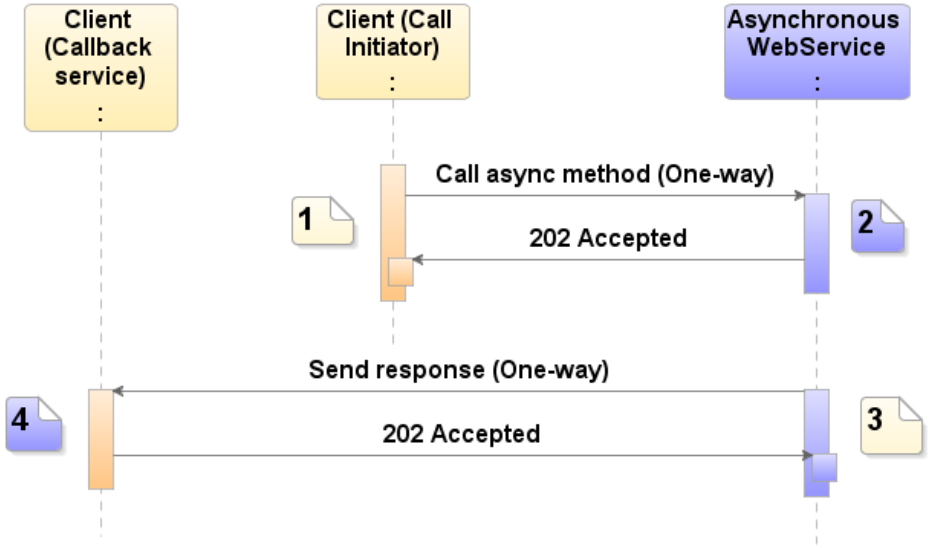


Demo Asynchronous Services

Oracle JDeveloper/iAS

Advanced Topics
Policy for Asynchronous Services

- Asynchronous Service
  - Server side policy for incoming request (2)
  - Callback client side policy to send response (3)
- Asynchronous Client
  - Client side policy to invoke asynchronous operation (1)
  - Callback server side policy to receive response (4)



Advanced Topics
Policy for Asynchronous Services

- Multiple ad hoc clients for service
  - Use PKI for web of trust, servers trust servers
    - Store incoming client public key for encrypting response where available
    - Or pass keys using WS-Addressing headers and WS-Identity
  - Use SAML, callback reuses identity of invoker
  - M:N configuration, nightmare

Advanced Topics
Transaction and Reliability

- Make piece wise transactional and reliable
  - Propagate WS-TX back to client
- Ensure that request and response do not get lost
- Make MDBs run in transaction
  - Container managed transaction demarcation
  - Bean managed transaction demarcation
- Setup JMS queue for retries
- Setup error queues for request and response queues

Summary

- Asynchronous APIs are powerful but harder to work with
- By using WS-Addressing you can create a cross platform asynchronous messaging system
- Client model programming model could be improved.
- Implementing a service can be fiddly
- We put forward a proposal for declaratively making a POJO service asynchronous
- Policy should be considered from the start

Q & A

- See Also
  - http://jax-ws.dev.java.net
  - http://www.oracle.com/technology/products/jdev/
  - http://www.oracle.com/technology/products/ias/index.html
  - http://edocs.bea.com/wls/docs/103/index.html
  - http://kingsfleet.blogspot.com

Thank You

Gerard Davison
gerard.davison@oracle.com
http://kingfleet.blogspot.com

Manoj Kumar
manoj.k.kumar@oracle.com