Asynchronous Web Services: From JAX-RPC to BPEL

Jonathan Maron
Oracle Corporation
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
Tightly Coupled Systems

- Make many assumptions about each other
- Suited for intra-application communication
- Suited for communication where you have control over both sides of interaction
- Generally more efficient, easier to develop and debug
Loosely Coupled Systems

- Less assumptions about each other
- Suited for communication where you do not control all systems
- Systems vary independently
- Generally less efficient and more difficult to develop and debug
Remote Procedure Call (RPC)

- Extension of local programming
- Allows invocation of method in remote applications
- Details of invocation hidden from developer
- Examples: RMI, IIOP, .NET Remoting

GetCreditRating() -> 740
RPC Dangers

- Tightly coupled
- Approach ignores:
  - Latency
  - Lack of shared memory access
  - Lack of control over remote systems

“...objects that interact in a distributed system need to be dealt within ways that are intrinsically different from objects that interact in a single address space.”

_A Note On Distributed Computing_, Jim Waldo _et al_, November 1994
Asynchronous Messaging

- Destinations have logical addresses
- Data encapsulated in messages using platform independent format (XML)
- “Client” application sends message and continues processing (“fire-and-forget”)
- Messaging infrastructure queues message until receiver is ready to consume it
Messaging Advantages

- Loose coupling and reliability
- Messages are self contained
- Removes:
  - Location dependencies
  - Temporal dependencies
  - Technology dependencies
  - Data format dependencies
Synchronous vs Asynchronous
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
## JAX-RPC 1.1

- Java API for XML Remote Procedure Calls
- Supports both RPC and Document-based Web services
- Supports one-way invocations
- No built-in support for asynchronous message exchanges!
Demonstration

JAX-RPC 1.1 One-Way invocation
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
Reliability

- Two existing standards
  - WS-Reliability 1.1
    - Sun, Oracle, Fujitsu, Hitachi, etc.
    - OASIS standard, November 2004
  - WS-ReliableMessaging
    - IBM, Microsoft
- Effort to merge underway
  - WS-ReliableExchange
WS-Reliability

- Addresses:
  - Message persistence
  - Message acknowledgement and resending
  - Elimination of duplicate messages
  - Ordered delivery of messages
  - Delivery status awareness for sender and receiver applications
Web Services Reliability

- At least once semantics
- At most once semantics

- Guaranteed message ordering
- Exactly once semantics
Reliability Client Code

```java
ReliabilityClient reliabilityClient =
    ReliabilityClientFactory.getClientFactory().getClient();
ReliabilityContext reliabilityContext =
    reliabilityClient.createContext();
ReliabilityListener listener = new MonitoringReliabilityListener();
reliabilityContext.setAcknowledgmentType(
    ReliabilityContext.ACKNOWLEDGMENT_TYPE_CALLBACK);
reliabilityContext.setReplyToURL( "http://127.0.0.1:9999"
reliabilityContext.addReliabilityListener( listener );

// invoke service request
String groupId = reliabilityContext.getGroupId();
String sequenceNumber = Long.toString ( reliabiltyContext.getSequenceNumber() );

// leverage to check on ack
```
class MonitoringReliabilityListener implements ReliabilityListener
{
    Map events = new HashMap();

    public boolean hasEvent (String groupId, String sequenceNumber )
    {
        return events.containsKey (groupId + sequenceNumber );
    }

    public void responseReceived ( ReliabilityEvent event )
    {
        String key = event.getGroupId() + Long.toString ( event.getSequenceNumber () );
        events.put (key, event);
    }
}
Demonstration

WS-Reliability
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
Orchestration Requirements

Connectivity
- Heterogeneous Back Ends
- Silos of API and mechanisms
- Opaque/heterogeneous data definitions
- Synchronizing multiple data stores

Flow Control
- Asynchrony, Flow Coordination, Data Transformation, Compensation, Version Control, Auditing

Scalability
- Unpredictable loads
- Asymmetric performance capabilities

Management and Security
- Access control, Encryption, Logging, Metering
- Independent of the service

Interaction/Access
- Catalog, Customization, Access
What Is BPEL?

- Markup language for composing a set of discrete services into an end-to-end process flow
- 10+ years of research and development from Microsoft (XLANG) and IBM (WSFL, FDML)
- The best integration solution for XML and Web services but also Java, JCA and JMS.
- Rich support for async interactions, parallel processing and exception management.
- Composability: A process flow is automatically a service.

“Gartner believes that BPEL will emerge as the leading industry standard for Web service orchestration and coordination of business processes.”
- David Smith, Research Vice President and fellow, Gartner

“BPEL is the future of the integration space in my view… Why? Because the value is so much higher when you provide not only a way to integrate applications, but also a way to create services from them and put them into business processes.”
- John Rymer, Vice President, Forrester Research, Inc.
BPEL by Example

<variable>

Credit Rating

<partnerLink>

United Loan

<partnerLink>

<process>

10:00am

Get Rating

<invoke>

Credit Rating

<switch>

Select Lowest Offer

03:00pm

<flow>

<receive>

Send Loan Application

<partnerLink>

<partnerLink>

Send Loan Application

<faultHandlers>

Handle Negative Credit Exception

<invoke>

<flow>

<partnerLink>

<partnerLink>

Receive Loan Offer

<partnerLink>

Send Loan Application

<partnerLink>

Receive Loan Offer

<partnerLink>
BPEL Interaction Patterns

1. Sync Request Response
2. Fire and Forget
3. Async. Request Response
4. Partial Processing
5. Progress Observer
6. Change Handler
BPEL Based Applications

BPEL Process

Web Application

JAX-RPC

JSR 227

WSDL

BPEL Process

Partner Links
Workflow

One top level Human Workflow Service which interacts with the BPEL engine and has pluggable services and encapsulates Notification, Assignment Service, TaskManager, WorklistManager, Identity Service

(1) assign tasks to a user/role
(2) wait for task completion as part of an end to end process flow
Demonstration
BPEL
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
JAX-WS 2.0

- Successor to JAX-RPC 1.1
- Retains RPC programming model
- Improves on predecessor
  - Integration with JAXB
  - Asynchronous Web Services!
  - Transport and protocol neutrality
  - New handler framework
  - WS Security
Asynchronous Web Services

- Request-response operations are mapped as either synchronous or asynchronous
- Two asynchronous modes on the client:
  - Polling (with and without timeout)
  - Callback
- Server-side asynchrony deferred
Asynchronous with Polling

- Client initiates operation
- Method returns immediately
  ➢ Returns Response object
- Client continues with other work
- When client is ready it retrieves value out of Response
- Alternatively, cancels request if it no longer cares
Example: Async with Polling

```java
public interface CreditRatingService extends Remote {

    // sync operation
    Score getCreditScore (Customer customer)
        throws RemoteException;

    // async operation w/ polling
    Response<Score> getCreditScoreAsync (Customer customer);

}
```
Example: Async with Polling

CreditRatingService creditSvc = ...;

Response<Score> response = svc.getCreditRatingAsync (customerFred);

// client does other work

// deal with response
Score score = response.get ();

// or Score score = response.get (10L,
// TimeUnit.SECONDS);
Example: Async with Callback

```java
public interface CreditRatingService extends Remote {

    // sync ops . . .

    // async w/ callback
    Future<?> getCreditRatingAsync (Customer customer,
                                    AsyncHandler<Score> handler);

}
```
Example: Async with Callback

CreditRatingService svc = ...  

Future<?> invocation = 
   svc.getCreditScoreAsync (customerFred, 
   new AsyncHandler<Score>() { 
      public void handleResponse 
         (Response<Score> response) { 
            Score score = response.get();  
            // more work based on score
         }
   });

   // could cancel with invocation.cancel(true);
Demonstration
JAX-WS 2.0 Early Access
Agenda

- Loose *versus* Tight Coupling
- Asynchronous Web Services Today
- Asynchronous Web Service Standards
  - WS-Reliability/WS-ReliableMessaging
  - WS-Events/WS-Notification
- Business Process Execution Language
- Asynchronous Web Services Tomorrow
  - JAX-WS 2.0
Absolutely Shameless Self Promotion

- From amazon.com reviews:
  - “Required Enterprise Transactions Reading”
  - “All J2EE developers should read this book”