Service Composition & Mediation

Orchestration vs. Choreography

Technology: BPeL

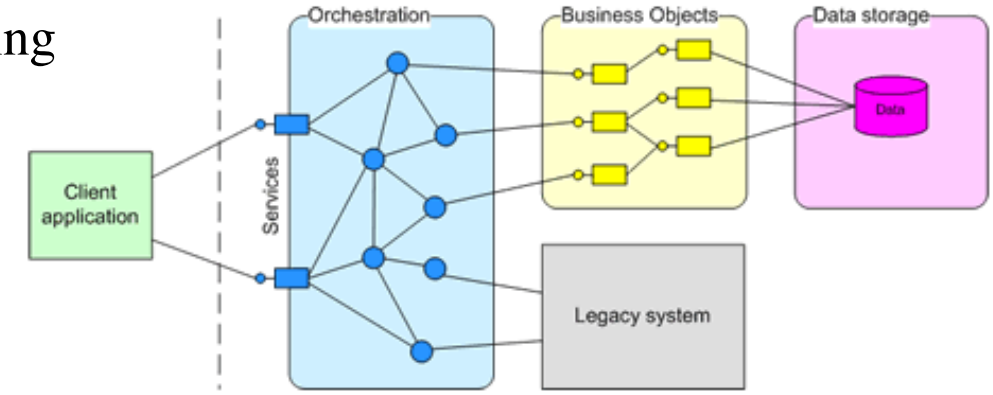
Architecture: ESB

SERVICE COMPOSITION

Service Composition

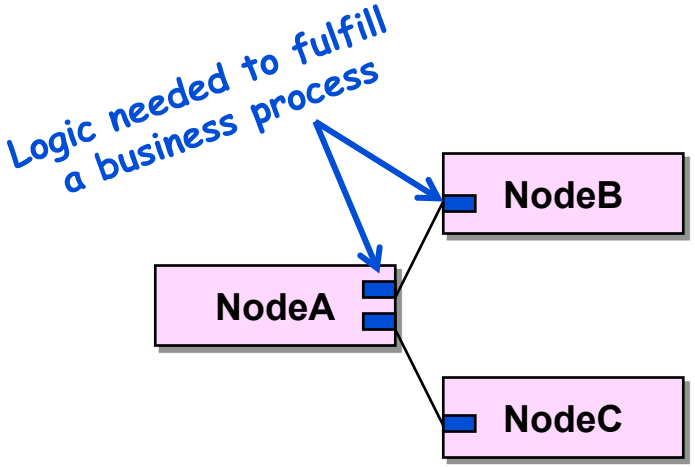
Systems behavior created by existing and/or newly created services

- Executes defined biz processes & workflow
- Defines workflow & routing of information to services
- Consumers only concerned about messages and their response



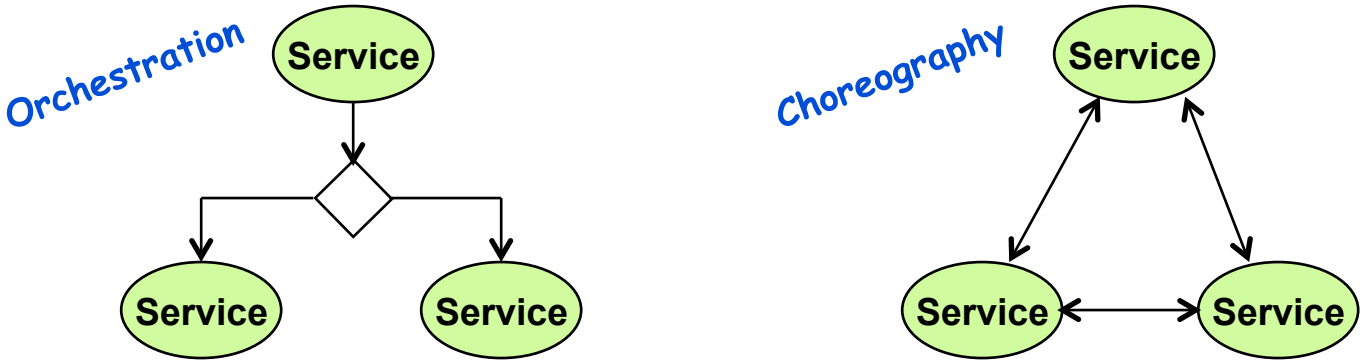
* From "At Your Service", IBM DeveloperWorks

The Question is: do we assemble services by having each know about the other (distributed workflow), or do we use a central service?



Service Composition

- Create a service from composition & coordination of lower-level services
  - Orchestration defines centrally controlled process flow (e.g., ESB architectures)
  - Choreography coordinates distributed services



- *The key difference between orchestration and choreography is whether there is a central service coordinating the process or whether the process logic is distributed among the participants.*
- *In either case, the logic is external to the app-level services*
- *BPEL (Business Process Execution Language) can be used for either orchestration or choreography*

BPEL Provides Behavior to Web Services

Web Services Technologies	
Interface	WSDL
Message	SOAP
Type	XML Schema
Data	XML
Behavior	BPEL

- XML based language used to integrate web services into a business process
- BPEL provides constructs for managing behavioral logic

Primitive :
- <invoke>
- <receive>
- <reply>
- <assign>
- <throw>
- <wait>
- <terminate>

Structured :
- <sequence>
- <flow>
- <switch>/<case>
- <while>
- <pick>

BPEL Example

```
<?xml version="1.0" encoding="utf-8"?>
<process name="insuranceSelectionProcess"
         targetNamespace="http://packtpub.com/bpel/example/"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:ins="http://packtpub.com/bpel/insurance/"
         xmlns:com="http://packtpub.com/bpel/company/" >

   <partnerLinks>
      <partnerLink name="client"
                   partnerLinkType="com:selectionLT"
                   myRole="insuranceSelectionService"/>

      <partnerLink name="insuranceA"
                   partnerLinkType="ins:insuranceLT"
                   myRole="insuranceRequester"
                   partnerRole="insuranceService"/>

      <partnerLink name="insuranceB"
                   partnerLinkType="ins:insuranceLT"
                   myRole="insuranceRequester"
                   partnerRole="insuranceService"/>
   </partnerLinks>
   <variables>
      <!-- input for BPEL process -->
      <variable name="InsuranceRequest"   messageType="ins:InsuranceRequestMessage"/>
      <!-- output from insurance A -->
      <variable name="InsuranceAResponse" messageType="ins:InsuranceResponseMessage"/>
      <!-- output from insurance B -->
      <variable name="InsuranceBResponse" messageType="ins:InsuranceResponseMessage"/>
      <!-- output from BPEL process -->
      <variable name="InsuranceSelectionResponse"
                messageType="ins:InsuranceResponseMessage"/>
   </variables>
```

Define web service that will be invoked

Process declarations

Process invoked by a client role will select the best price between insuranceA and insuranceB

Specify variable types used by services in the process

Example from http://www.theserverside.com/articles/content/BPELJava/article.html

BPEL Example

```
<sequence>
     <!-- Receive the initial request from client -->
     <receive partnerLink="client"
              portType="com:InsuranceSelectionPT"
              operation="SelectInsurance"
              variable="InsuranceRequest"
              createInstance="yes" />
     <!-- Make concurrent invocations to Insurance A and B -->
     <flow>
         <invoke partnerLink="insuranceA"
                 portType="ins:ComputeInsurancePremiumPT"
                 operation="ComputeInsurancePremium"
                 inputVariable="InsuranceRequest"
                 outputVariable="InsuranceAResponse" />
         <invoke partnerLink="insuranceB"
                 portType="ins:ComputeInsurancePremiumPT"
                 operation="ComputeInsurancePremium"
                 inputVariable="InsuranceRequest"
                 outputVariable="InsuranceBResponse" />
     </flow>
     <switch>
       <case condition="bpws:getVariableData('InsuranceAResponse',
                        'confirmationData','/confirmationData/Amount')
                     <= bpws:getVariableData('InsuranceBResponse',
                        'confirmationData','/confirmationData/Amount')">
          <assign>
            <copy>
              <from variable="InsuranceAResponse" />
              <to variable="InsuranceSelectionResponse" />
            </copy>
          </assign>
       </case>
```

Process flow of events

Insurance request is input into the process

Invoke the services

Select the best offer and construct the response

Select Insurance A

BPEL Example

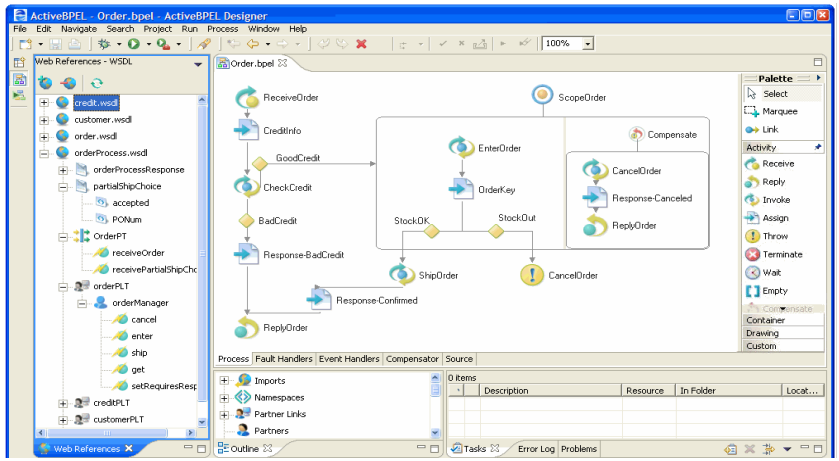
```
<!- The default case is called "otherwise" -->
       <otherwise>
          <!-- Select Insurance B -->
          <assign>
            <copy>
              <from variable="InsuranceBResponse" />
              <to variable="InsuranceSelectionResponse" />
            </copy>
          </assign>
       </otherwise>
     </switch>
 <!-- Send a response to the client -->
     <reply partnerLink="client"
            portType="com:InsuranceSelectionPT"
            operation="SelectInsurance"
            variable="InsuranceSelectionResponse"/>
  </sequence>
</process>
```

Process information from the Web Service calls

Generate process response, the insurance selection value

Or just model the business process & generate BPEL!!!

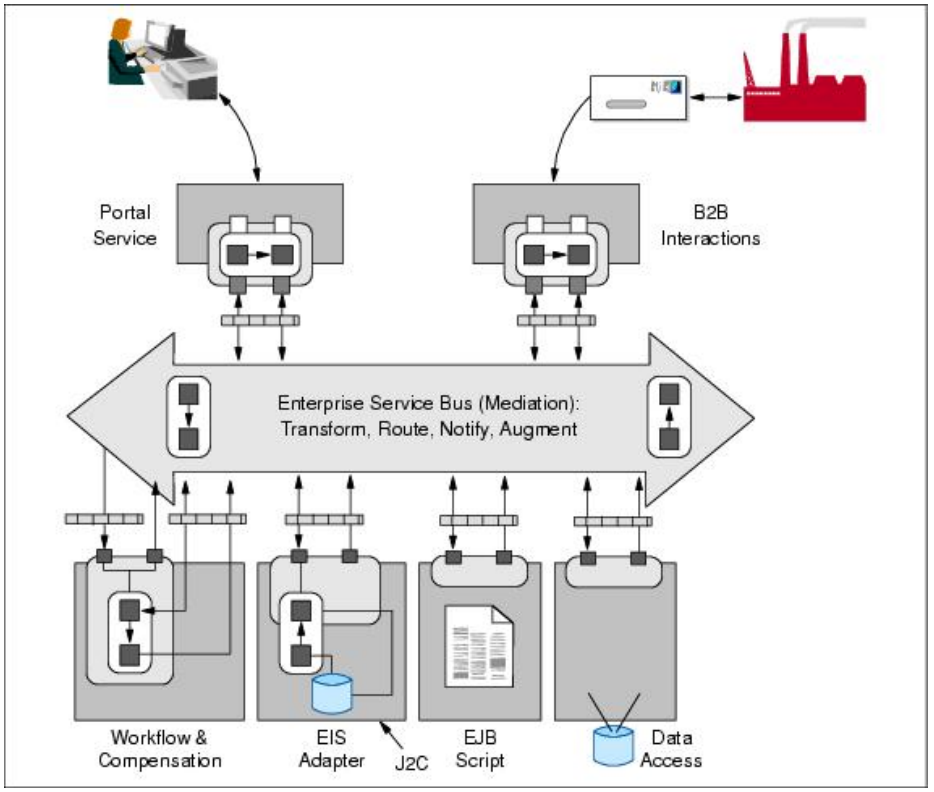
- Vendors support modeling solutions based on UML Activity Diagrams



INTEGRATION ARCHITECTURE : ESB

Enterprise Service Bus

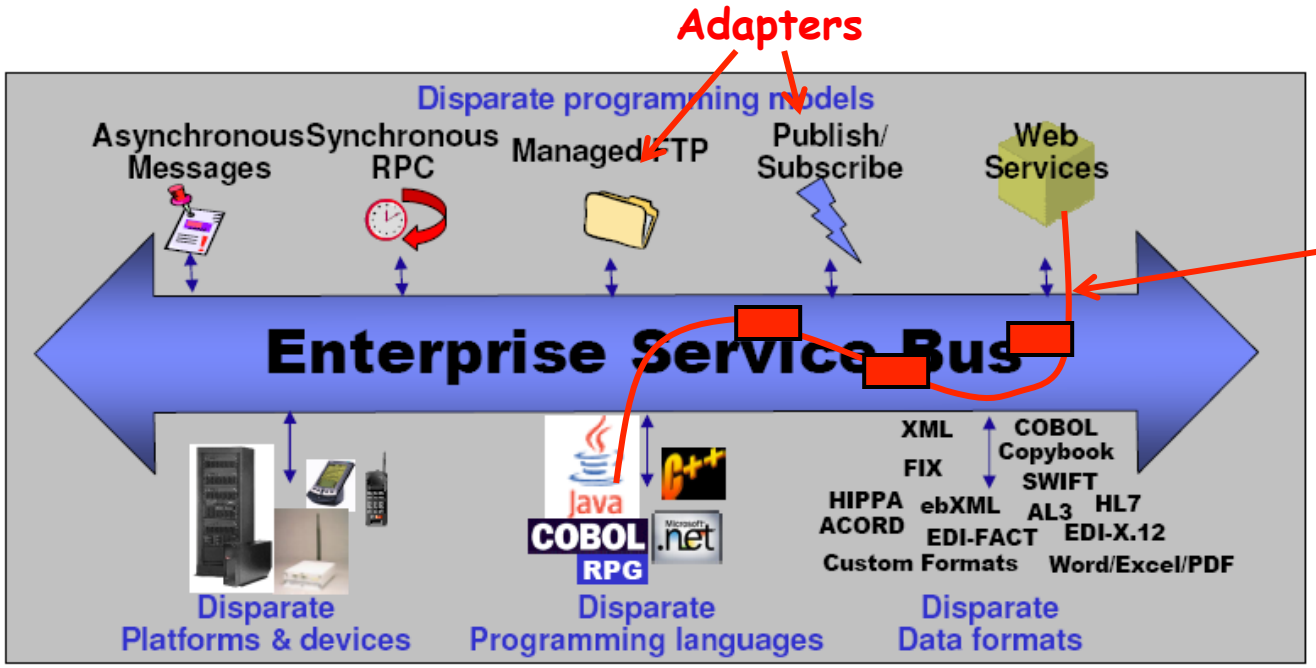
- SOAs use a broker pattern to provide communication
  - Services consumers and providers not directly connected
- Message/Service/Information Bus:
  - Marshals service data into normalized form
  - Delivers normalized messages to end points
  - Routes request to correct service provider
  - Mediates transport protocols



*** From "Patterns: Service-Oriented Architecture and Web Services", IBM DeveloperWorks**

ESBs and Integration

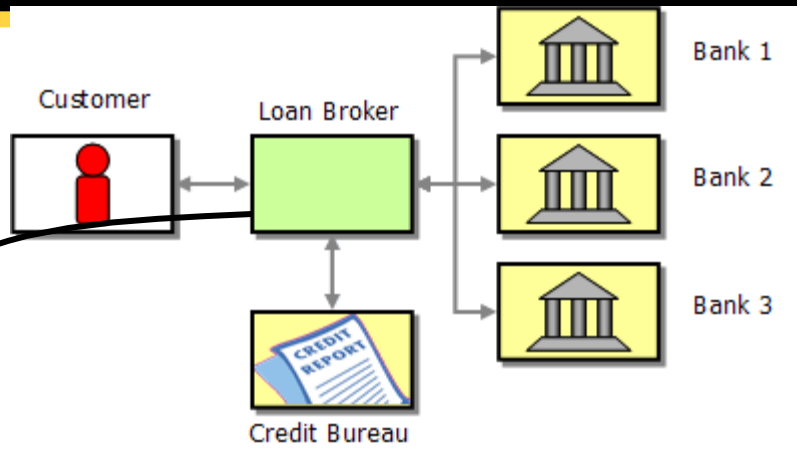
- Adapters facilitate integration "***without writing code***"
  - ESB provides adapters that inject messages into or out of the bus
  - Based on common technologies – databases, web services, JMS, etc.
- Message chain can be declared, not coded
  - Modify integration without developers ($) or a new software release (time)

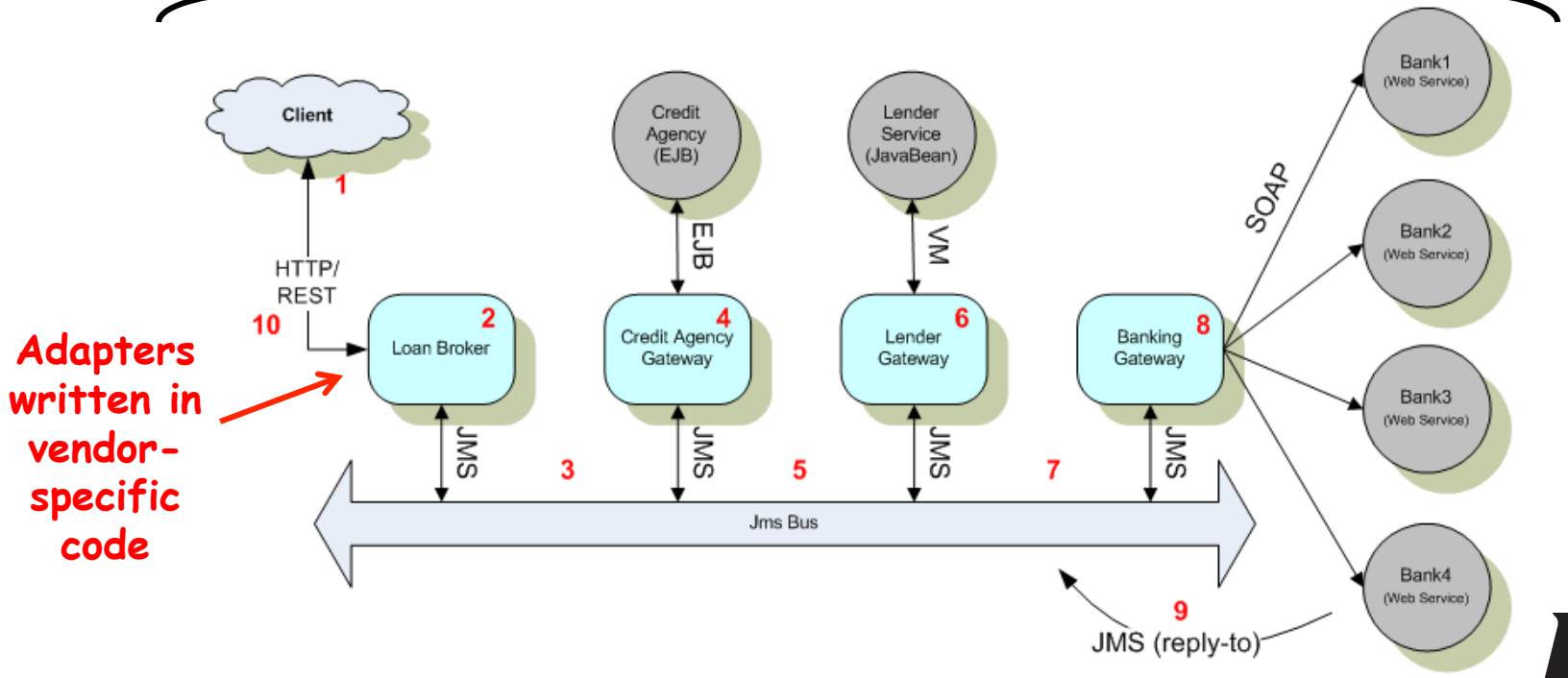


Message chain:
- Input adapter
- Mediation components
- Output adapter

ESB Example

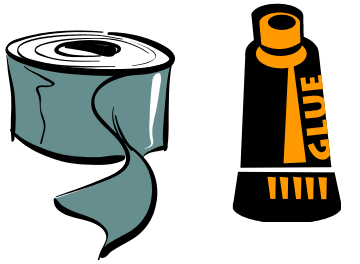
- ESBs use internal message structure (XML, JMS, etc.)
- Components and adapters written in vendor-specific code





ESB Discussion

- ESB Strengths
  - Many binding adapters provided by ESB container
  - No code added to services
    - Simple communication patterns can be declared in the ESB
    - Programmed clients can connect to ESB which provides access to services (connection, transformations, location, etc)
- ESB Weaknesses
  - Composite data (collections) are challenging
  - Standards exist, but not well adopted and most vendors provide proprietary extensions
  - Complexity of specifying adapters/routing rules and deployment
    - However, is it more complex than modifying the applications being integrated?



ASU