Business Process Modelling and Implementation
BPEL: Business Process Execution Language

Helen Paik

School of Computer Science and Engineering
University of New South Wales

References:

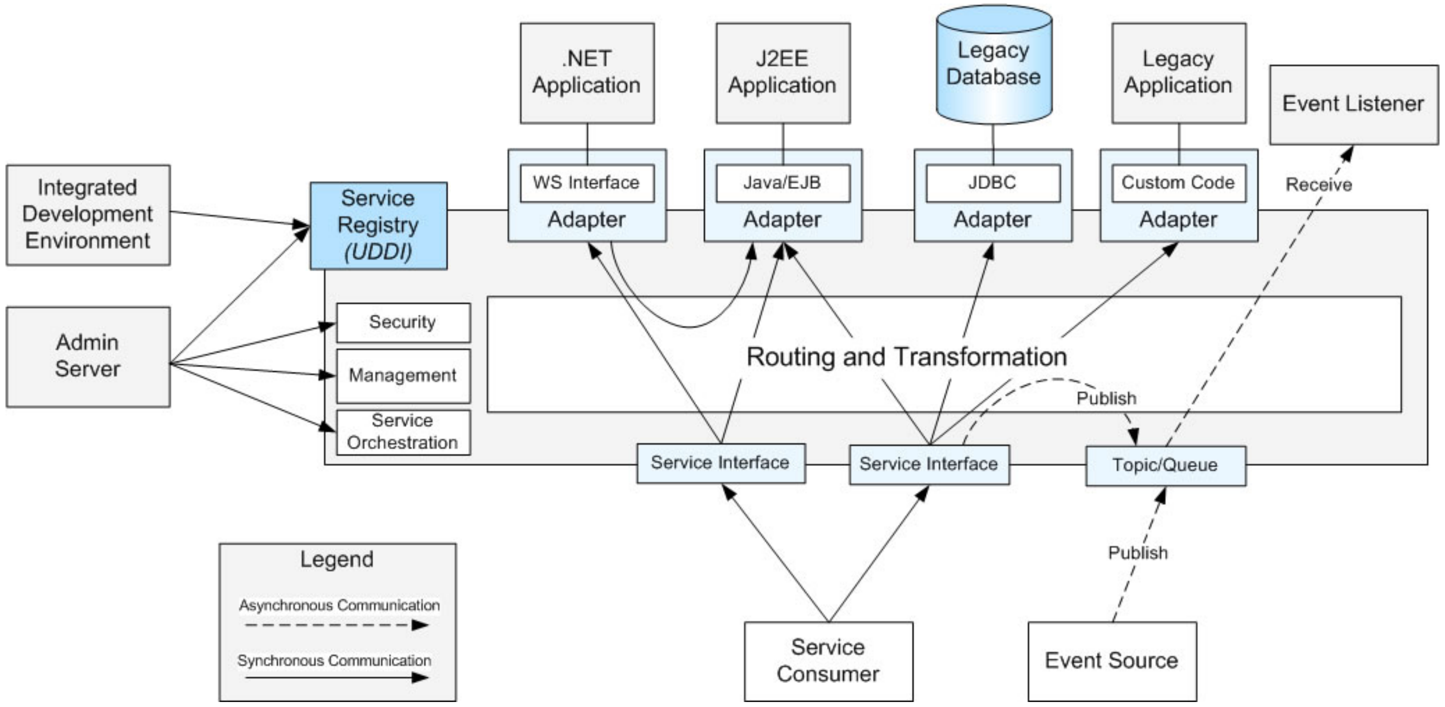
- BPEL 2.0: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
- Essential Business Process Modelling, Michael Havey, O'Reilly, 2005

Acknowledgement: The slides are adapted from COMP9322 lectures in previous sessions, which were originally prepared by Dr. Helen H-Y Paik, Dr. Sherif Sakr. This semester, we also refer to lecture materials created by Prof. Fabio Casati (University of Trento, Italy) and Dr. Marcello La Rosa (QUT, Brisbane) for their respective SOA courses.

Week 4

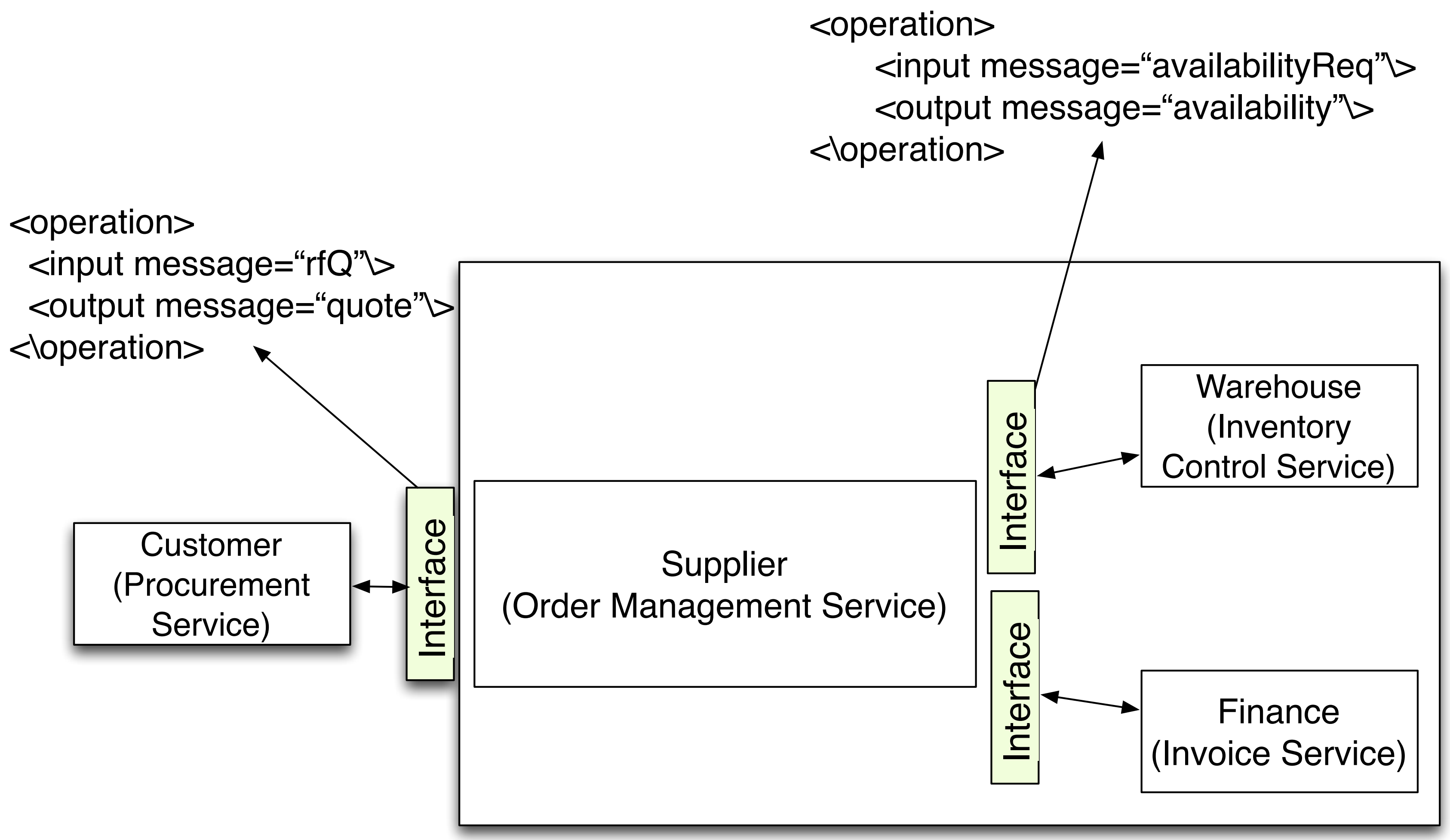
So far through the Web services topics ...

- Web Services as a standard way of invoking remote components (SOAP, WSDL, UDDI and implementation platforms like Apache CXF)
- Application integration requires more than the ability to conduct simple interactions via invoking remote operations. It requires coordinating complex interactions amongst the services involved → *coordination logic*



Business Processes and Web Services

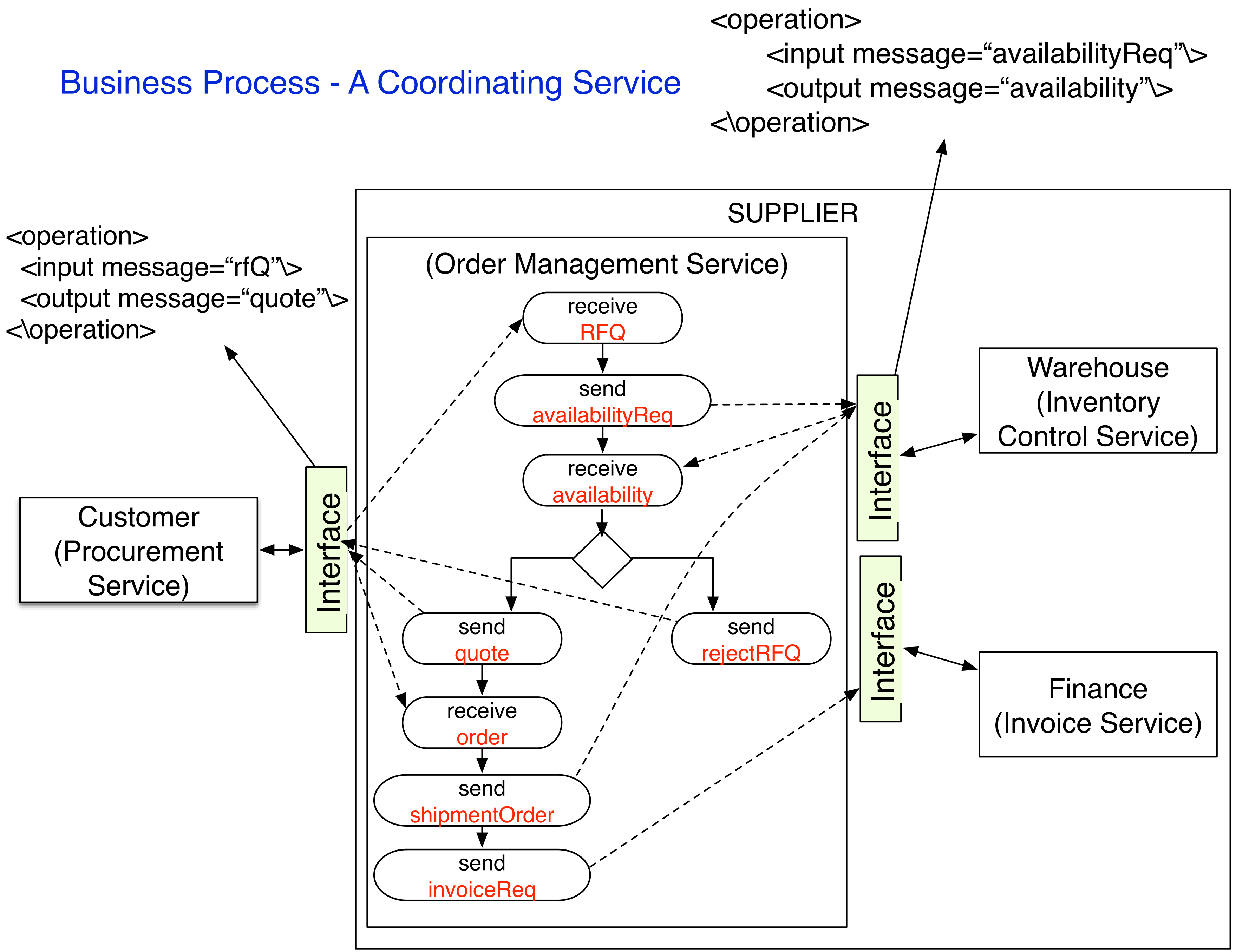
A business process acts as a ‘coordinating service’ for the Web services. It coordinates all necessary service calls according to some process logic (i.e., a path in a workflow).



<operation>
<input message=“availabilityReq”\>
<output message=“availability”\>
<\operation>
<operation>
<input message=“rfQ”\>
<output message=“quote”\>
<\operation>
Customer (Procurement Service)
Interface
Supplier (Order Management Service)
Interface
Warehouse (Inventory Control Service)
Interface
Finance (Invoice Service)

Figure from Dr. Marcello La Rosa’s course note, QUT, Brisbane

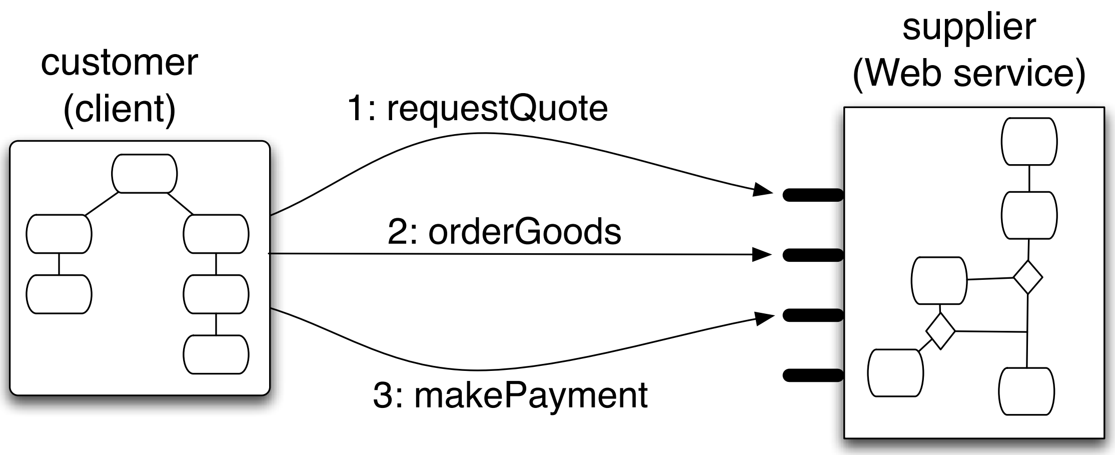
Business Process - A Coordinating Service



Interacting with a Service Interface

Let's leave the BPM aside for a moment and consider the concept of interacting with a Web service - or a *service interface*.

Generally speaking, an interaction with a service interface could be more complex than calling a single operation.
e.g., A simple buying process:



The interface description - WSDL doesn't tell you *how* the client should interact with the service

Interacting with a Service Interface

Note that these interactions have to be performed in a given order. This order is referred to as 'conversation'.

The conversation affects how the client is implemented, i.e., the business logic of the client must support the conversation (e.g., allowed to skip a quote?)

- **Web Service Coordination Protocols**: a collection of *valid conversations* supported by a service interface is called *'co-ordination protocol'*

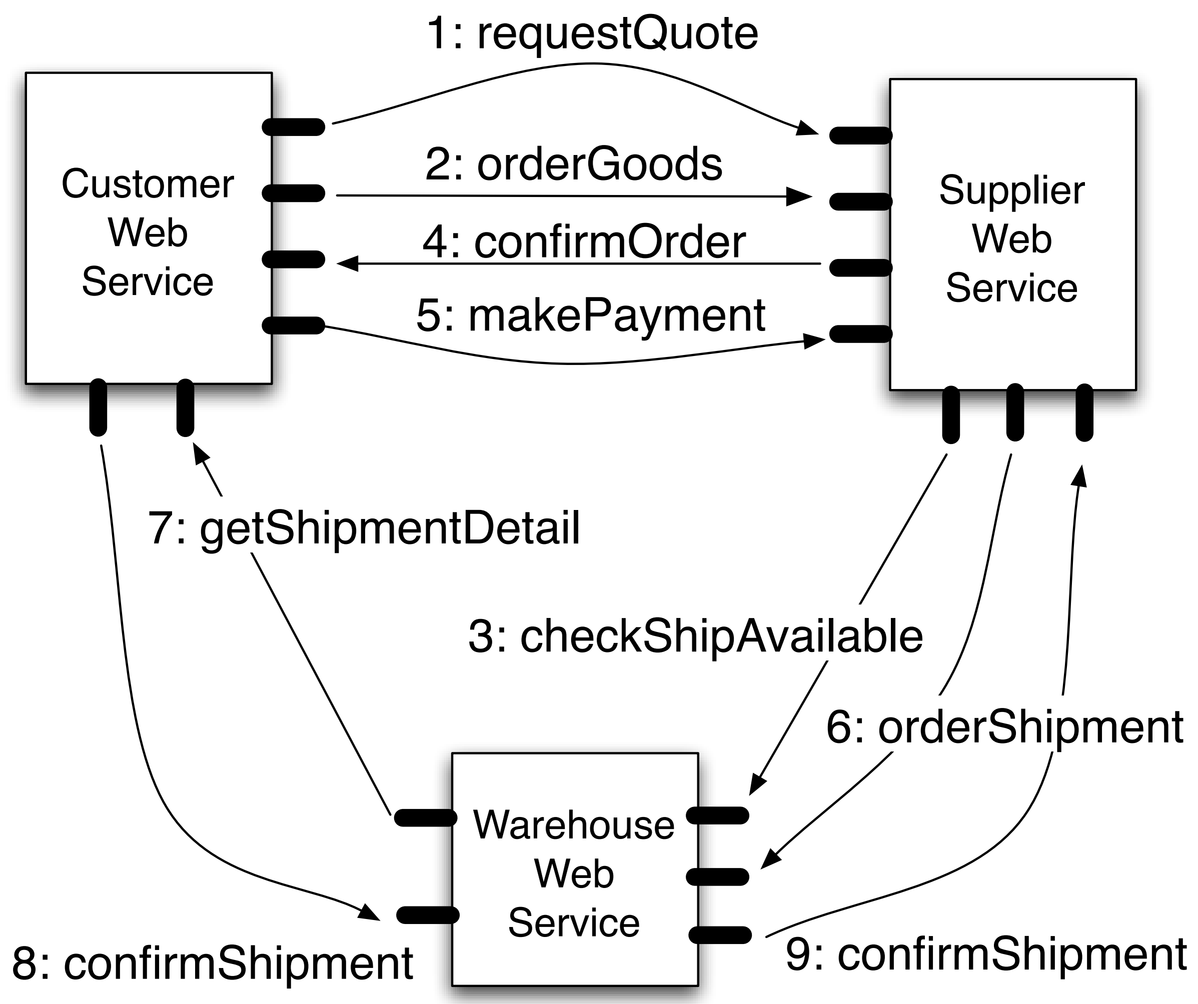
  A co-ordination protocol detail can be given to the client beforehand ... the client can work out which conversations are allowed

- Two types of Web service coordinations: *orchestration* and *choreography*

To discuss this, let's look at a conversation involving multiple Web services.

Conversations among Multiple Web Services

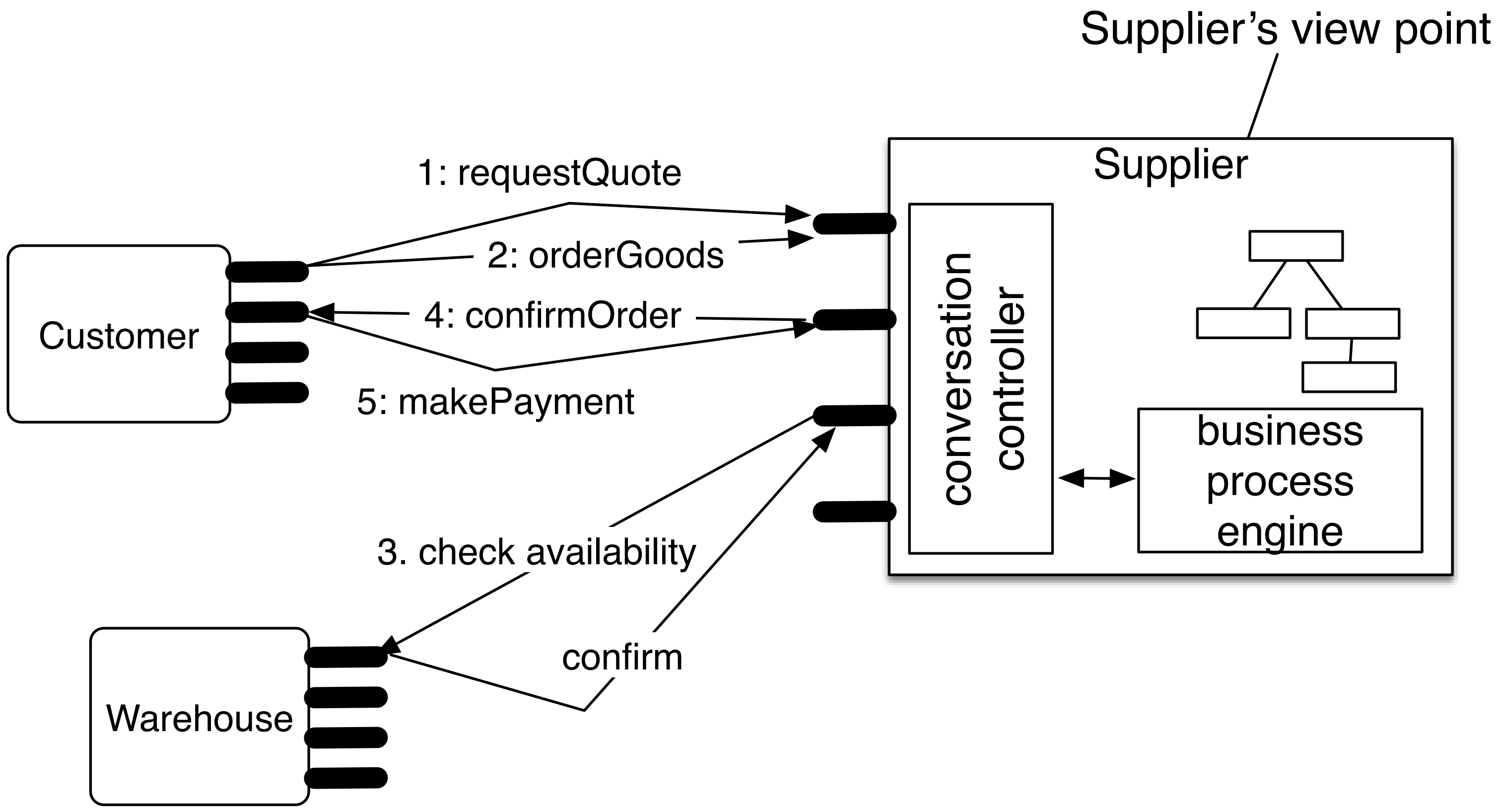
eg. Procurement Protocols - global view



choreography: multi-party collaboration

Two view points of coordinations: external vs. internal

eg. Procurement Protocols - one/single controller view



orchestration: single executable processes – Web service composition

WS Orchestration vs. Choreography

- **Orchestration** describes how Web services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint (single party).
- **Choreography** is associated with the public (globally visible) message exchanges, rules of interaction and agreements that occur between multiple business process endpoints.
- **Choreography** tracks the sequence of messages that may involve multiple parties and multiple sources, and described from the perspectives of all parties (common view).

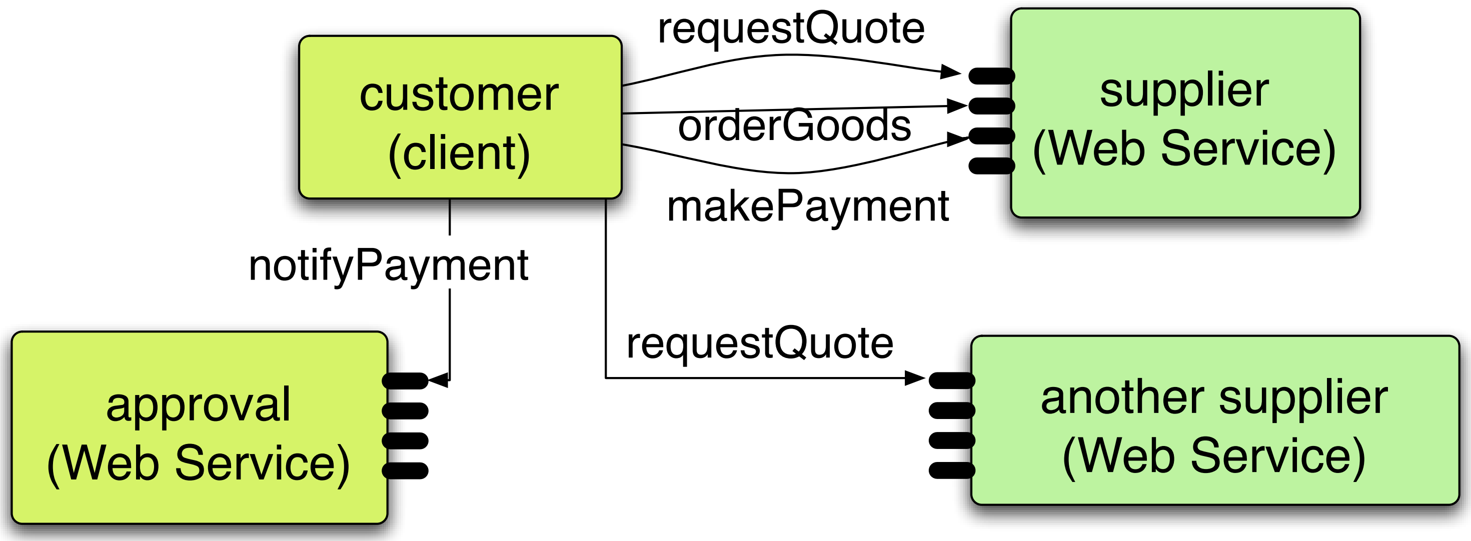
orchestration: single executable processes – Web service composition
choreography: multi-party collaboration – industry B2B standards ...

Web Service Composition

Web service composition is a coordination/*orchestration* implementation technique.

eg., A procurement scenario. A client;

1. Sends requestQuotes to two different suppliers.
2. Chooses the supplier that offers a better deal.
3. Obtains an approval from its finance department
4. Sends an order and then makes the payment



Web Service Composition

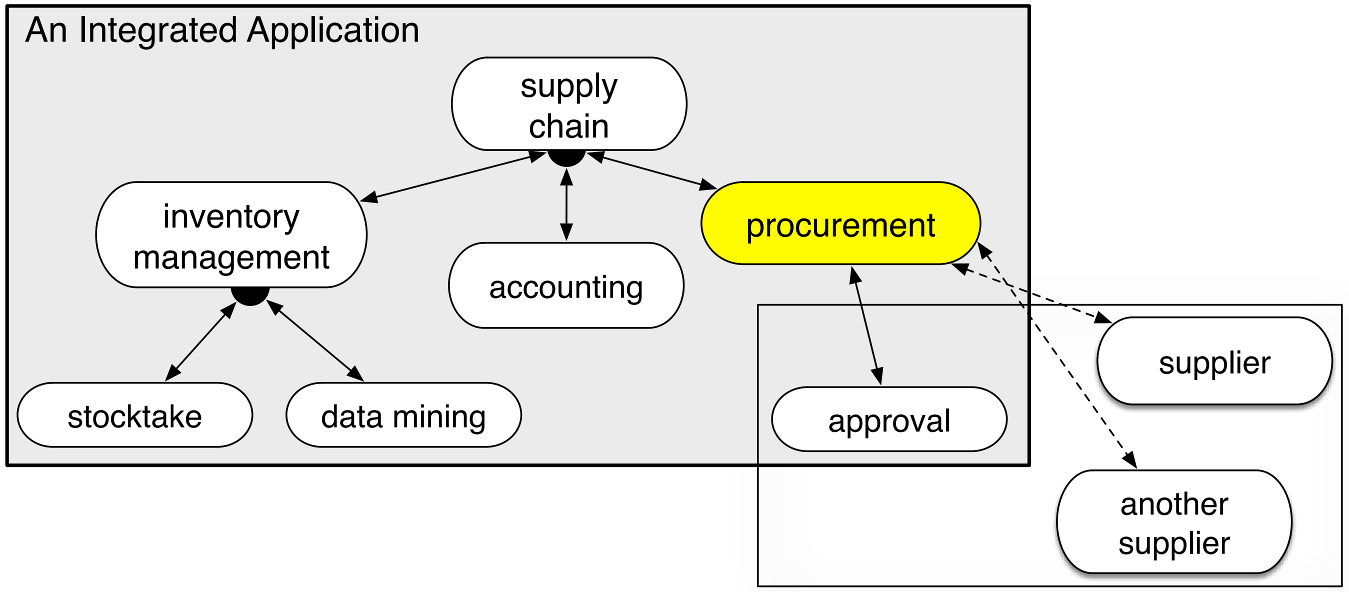
Important observations:

- The suppliers are Web services, as is the internal application that performs approval.
- Hence, the business logic of the client is realised by ‘composing’ multiple services (i.e., orchestrating the interaction among multiple services)
- This is the core concept of Web service composition: *Write a program by calling on other services* (i.e., a new application implemented by integrating other applications)

Web Service composition can be iterated $\rightarrow$ Consider Web Services as building blocks that can be assembled. It allows building of a complex applications by progressively aggregating components.

Web Service Composition

e.g., the client program itself can be a procurement 'Web Service' to other clients



Once it becomes a web service, it may be integrated into a bigger application as a component that provides procurement functionality.

This allows to maintain higher levels of abstraction (i.e., the 'composer of Web services' does not necessarily know the "inside" of each service

Web Service Composition

Things to note about Web Service Composition:

- Web services are not like application libraries where you have to be compiled and linked as part of an application.
  - The basic components (individual services) remains separated (ie., exist independently) from the composite service.

- A composition of web services mainly involves specifying which services need to be invoked, in what order and how to handle exceptional situations, etc.
  - can be seen as Web service-based workflow

Web Service Composition Development

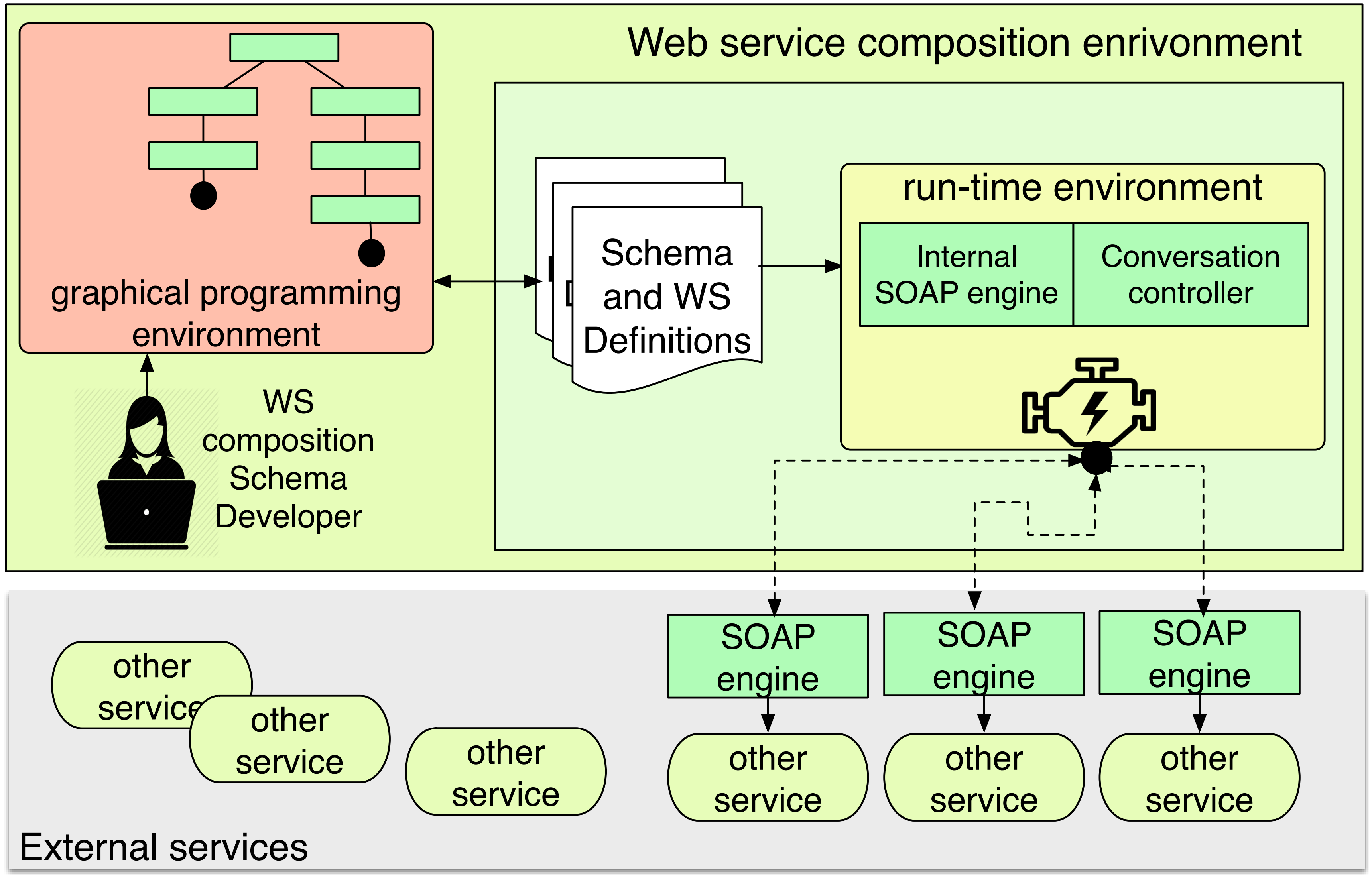
A Web service composition development environment $\rightarrow$ BPM technology gives a method to model and execute Web service composition.

- **composition model and langauge**: enabling the specification of the services to be combined, the order in which the different services are to be invoked, and the way in which service invocation parameters are determined. The specification is referred to as *composition schema*.

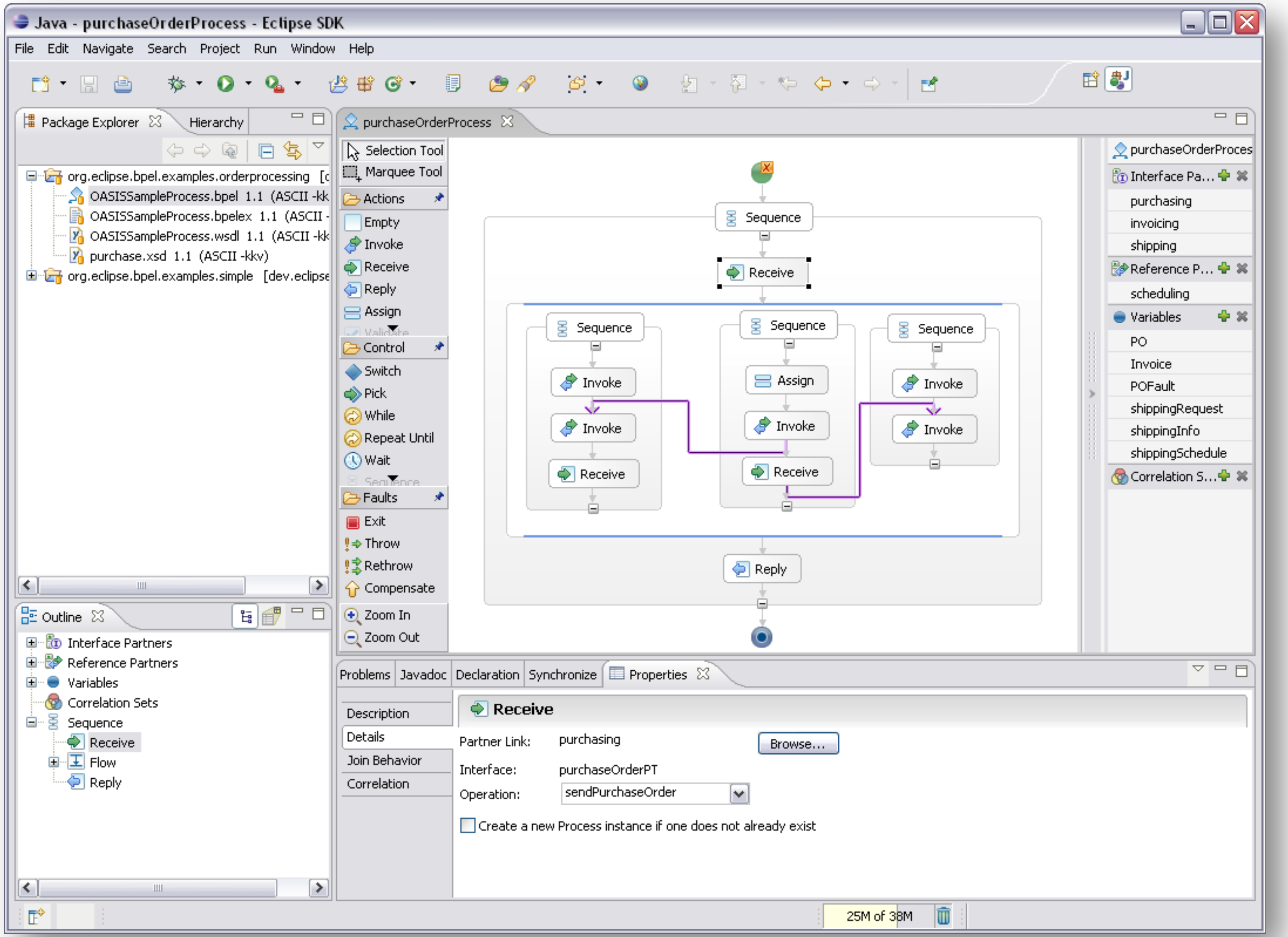
- **graphical user interface**: an interface through which designers can specify a composition schema by dragging and dropping Web services into a canvas. The graphs and other descriptive information are them translated into textual specifications (ie., the composition schema)

- **run-time environment** composition engine that executes the business logic of the composite service.

Web Service Composition Development



A Graphical Development Env. (e.g., Eclipse plugin)



Receive

Partner Link: purchasing

Interface: purchaseOrderPT

Operation: sendPurchaseOrder

☐ Create a new Process instance if one does not already exist

Business Process Execution Language (BPEL)

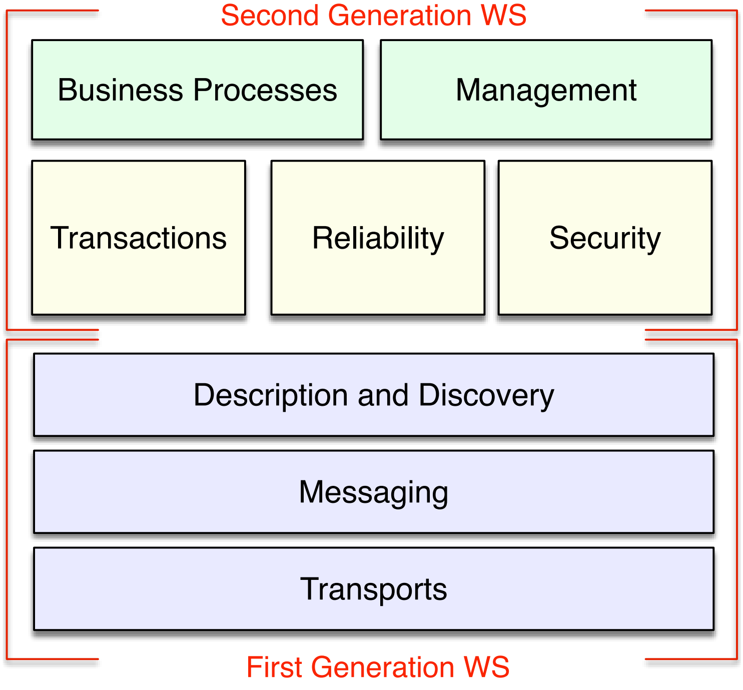
BPEL is one of the most popular standards for Web Service composition (referred to as "business process" in this language). BPEL Specification: OASIS standard - http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

BPEL is an XML programming language. Two types of files:

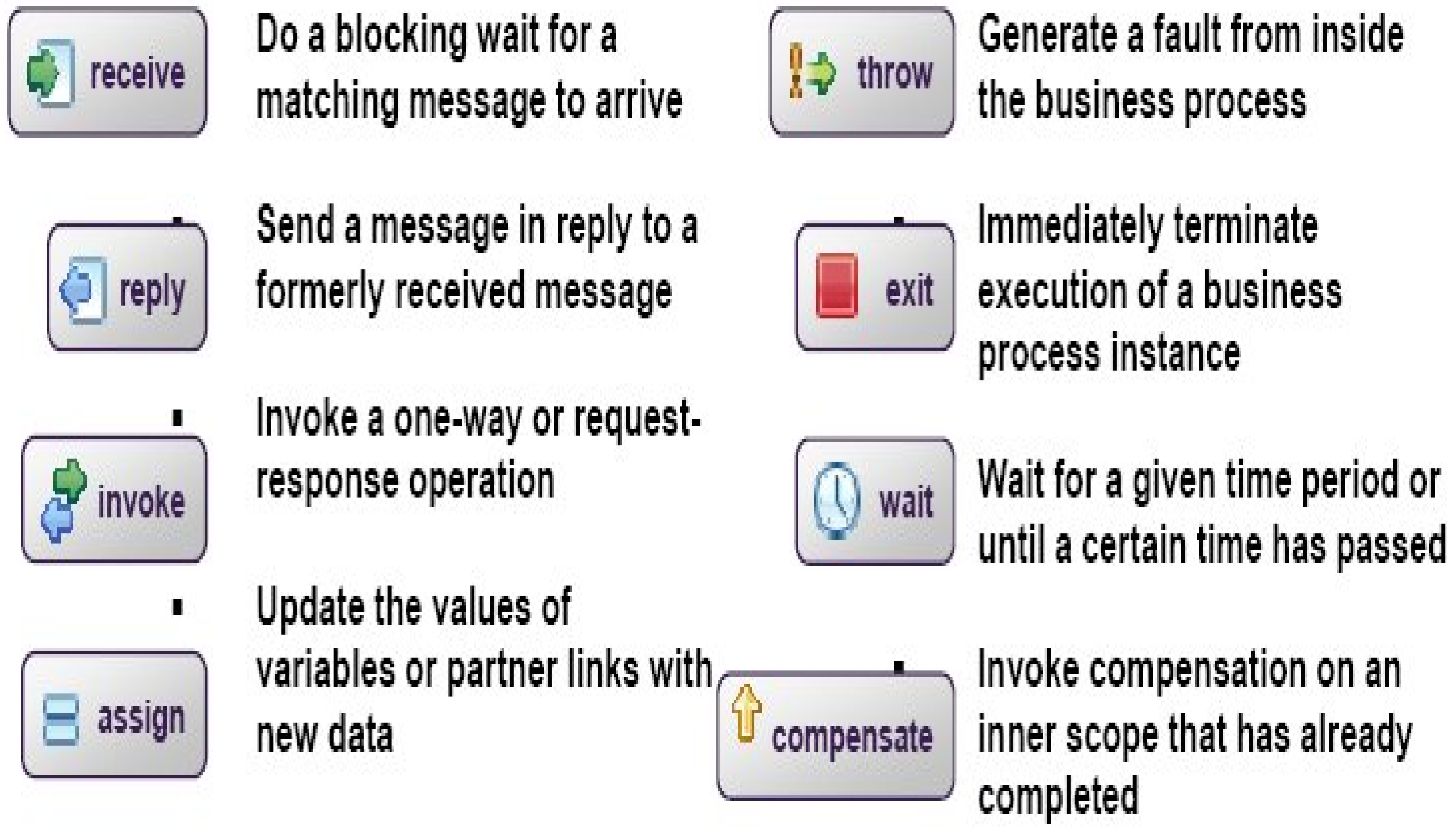
- WSDL files: specifying interfaces of the services implemented by and called by the process
- BPEL files: each encodes, in XML form, the definition of a process including activities, routing options, etc.

BPEL in WS-* Stack

http://www.ibm.com/developerworks/webservices/standards/

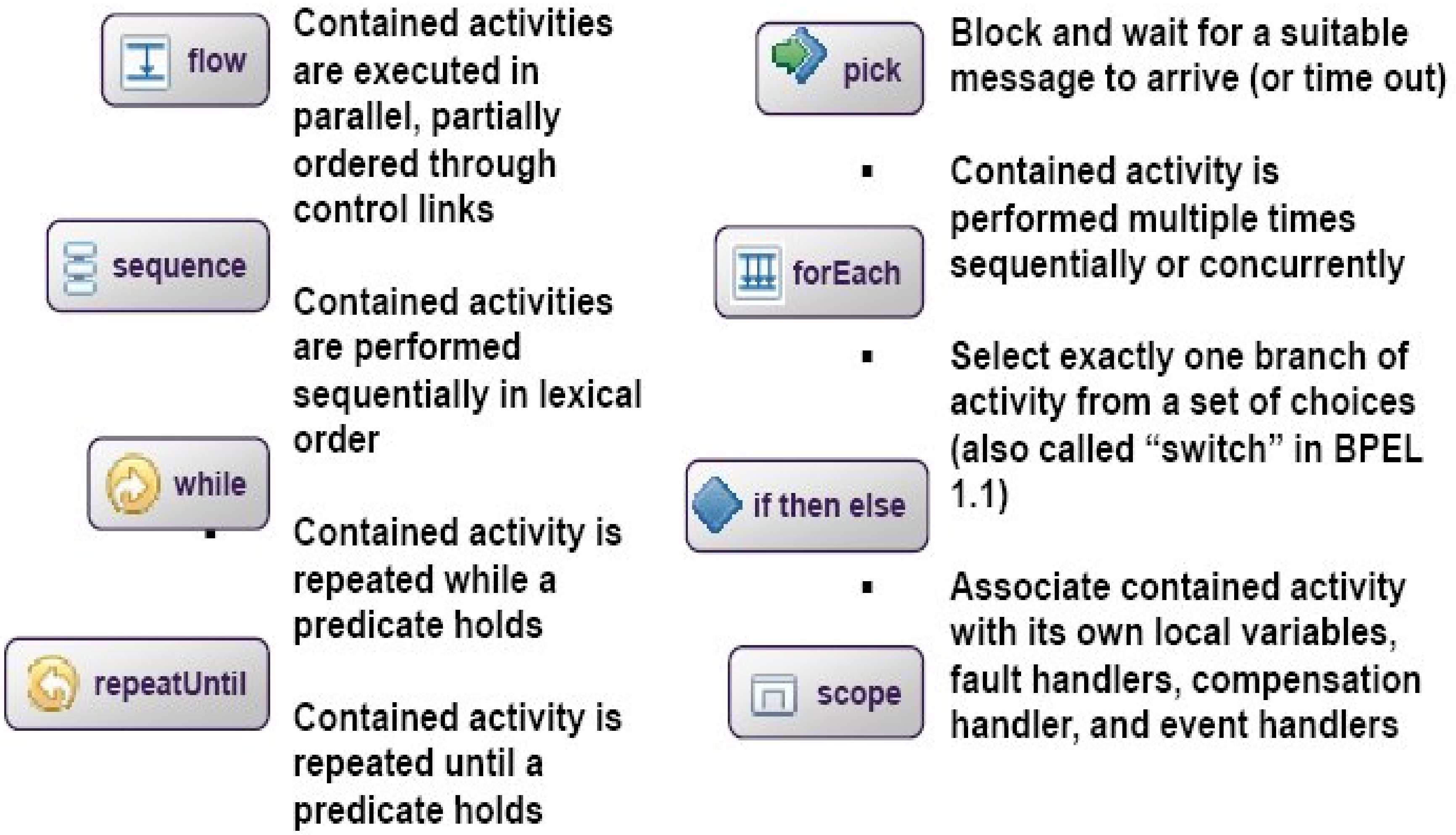


BPEL : Basic Activities



receive	Do a blocking wait for a matching message to arrive
reply	Send a message in reply to a formerly received message
invoke	Invoke a one-way or request-response operation
assign	Update the values of variables or partner links with new data
throw	Generate a fault from inside the business process
exit	Immediately terminate execution of a business process instance
wait	Wait for a given time period or until a certain time has passed
compensate	Invoke compensation on an inner scope that has already completed

BPEL : Structured Activities



flow — Contained activities are executed in parallel, partially ordered through control links

sequence — Contained activities are performed sequentially in lexical order

while — Contained activity is repeated while a predicate holds

repeatUntil — Contained activity is repeated until a predicate holds

pick — Block and wait for a suitable message to arrive (or time out)

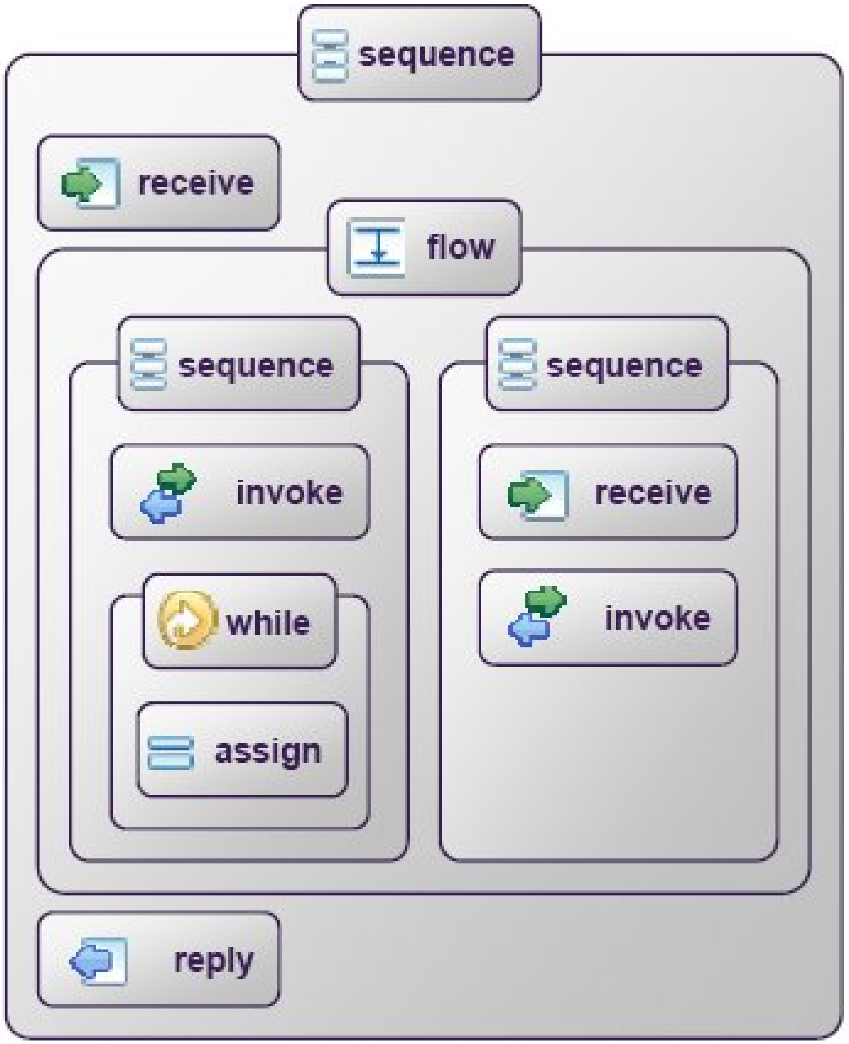
forEach — Contained activity is performed multiple times sequentially or concurrently

if then else — Select exactly one branch of activity from a set of choices (also called “switch” in BPEL 1.1)

scope — Associate contained activity with its own local variables, fault handlers, compensation handler, and event handlers

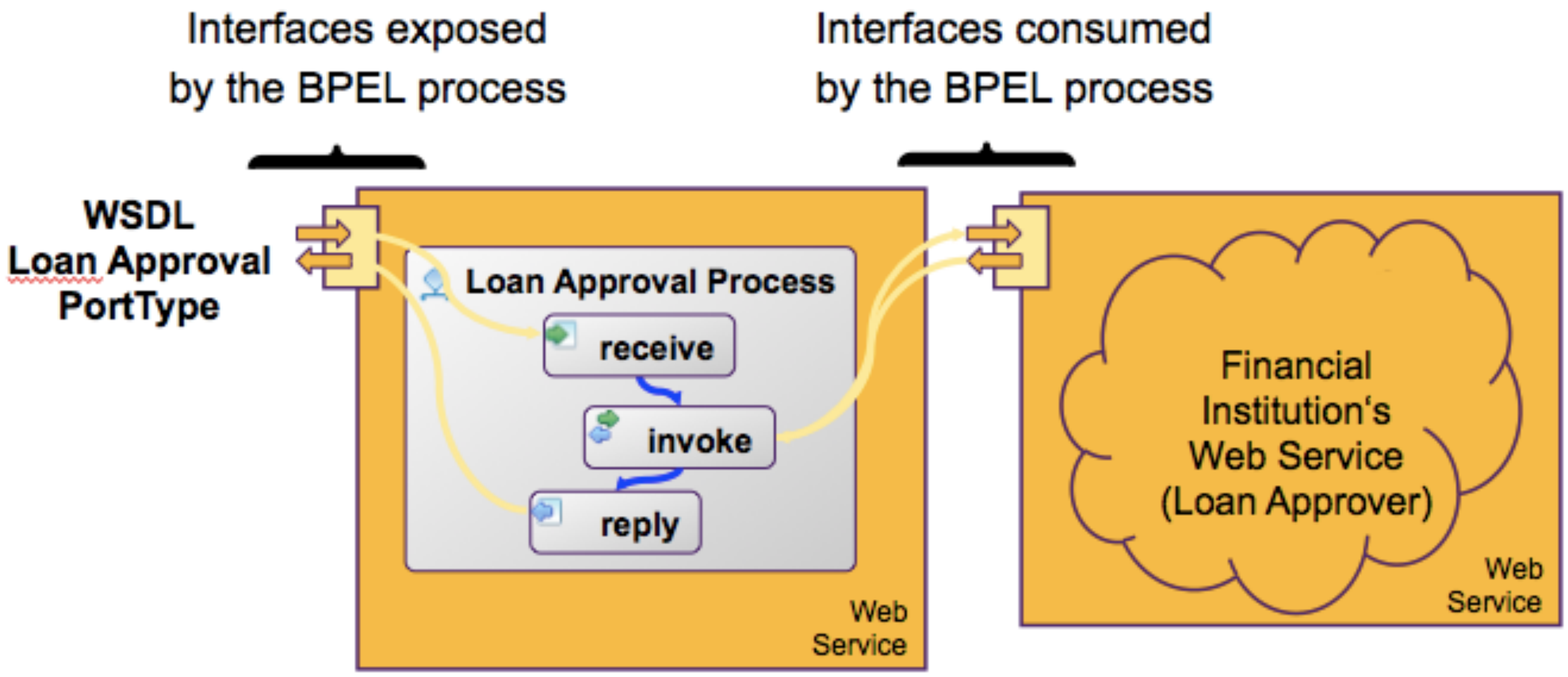
BPEL : Structuring Activities (Nesting)

```
<sequence>
   <receive .../>
   <flow>
      <sequence>
         <invoke .../>
         <while ... >
            <assign>...</assign>
         </while>
      </sequence>
      <sequence>
         <receive .../>
         <invoke ... >
      </sequence>
   </flow>
   <reply>
</sequence>
```



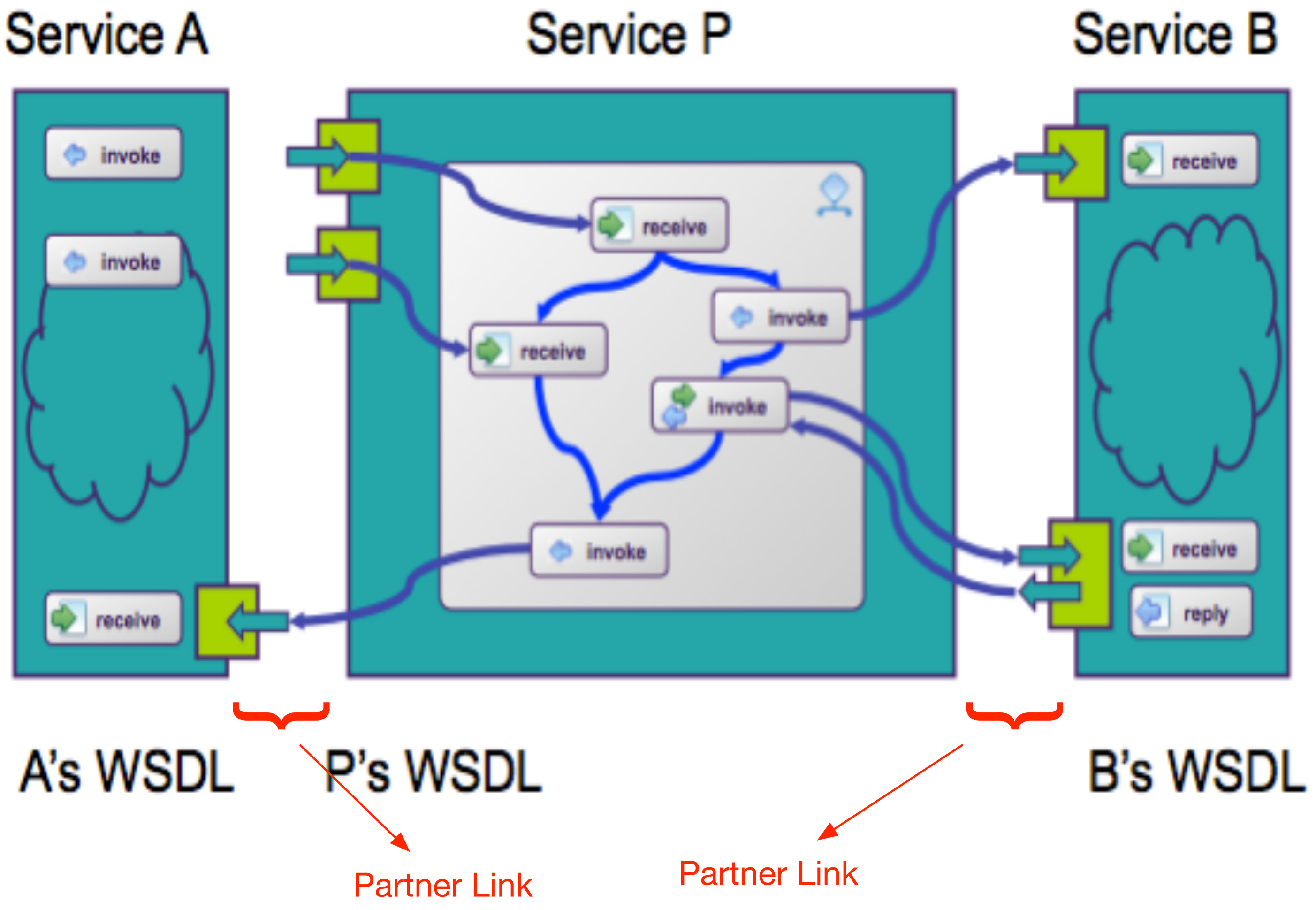
BPEL concepts: .bpel (process) and .wsdl (interfaces)

- The result of a composition (business process) using BPEL is recursively published as WSDL.
- BPEL processes interact with WSDL services exposed by business partners


Interfaces exposed by the BPEL process
Interfaces consumed by the BPEL process
WSDL Loan Approval PortType
Loan Approval Process
receive
invoke
reply
Web Service
Financial Institution's Web Service (Loan Approver)
Web Service

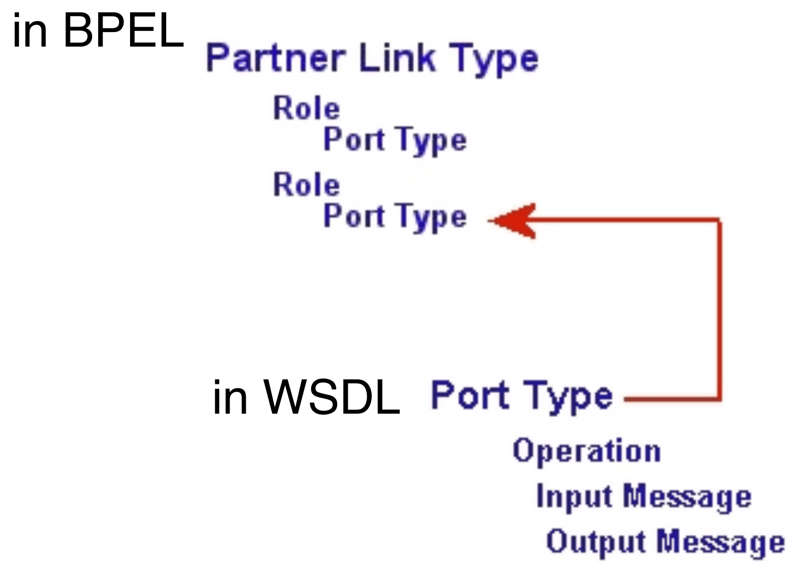
BPEL concepts: partners and partner link types

BPEL, as the controller in the orchestration, is 'consumed' by services and also 'consumes' services.

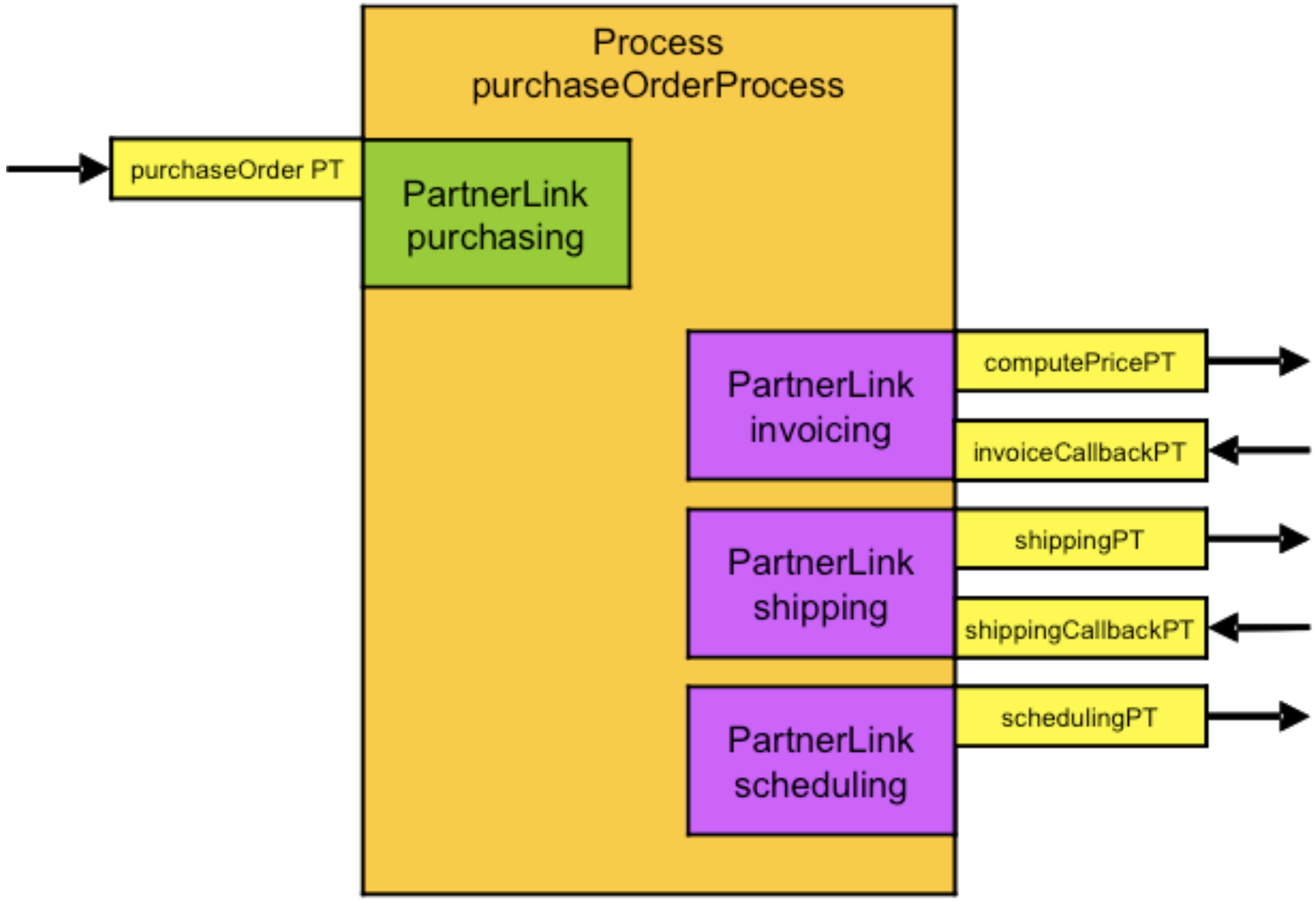


BPEL concepts: partners and partner link types

- *partners* are the actors in a composition who play roles.
- **partner link types** characterizes the conversational relationship between two services by defining the "roles" played by each of the services
- Each role in a partner link type is linked with a WSDL port type
- Web services playing the role are required to support such operations

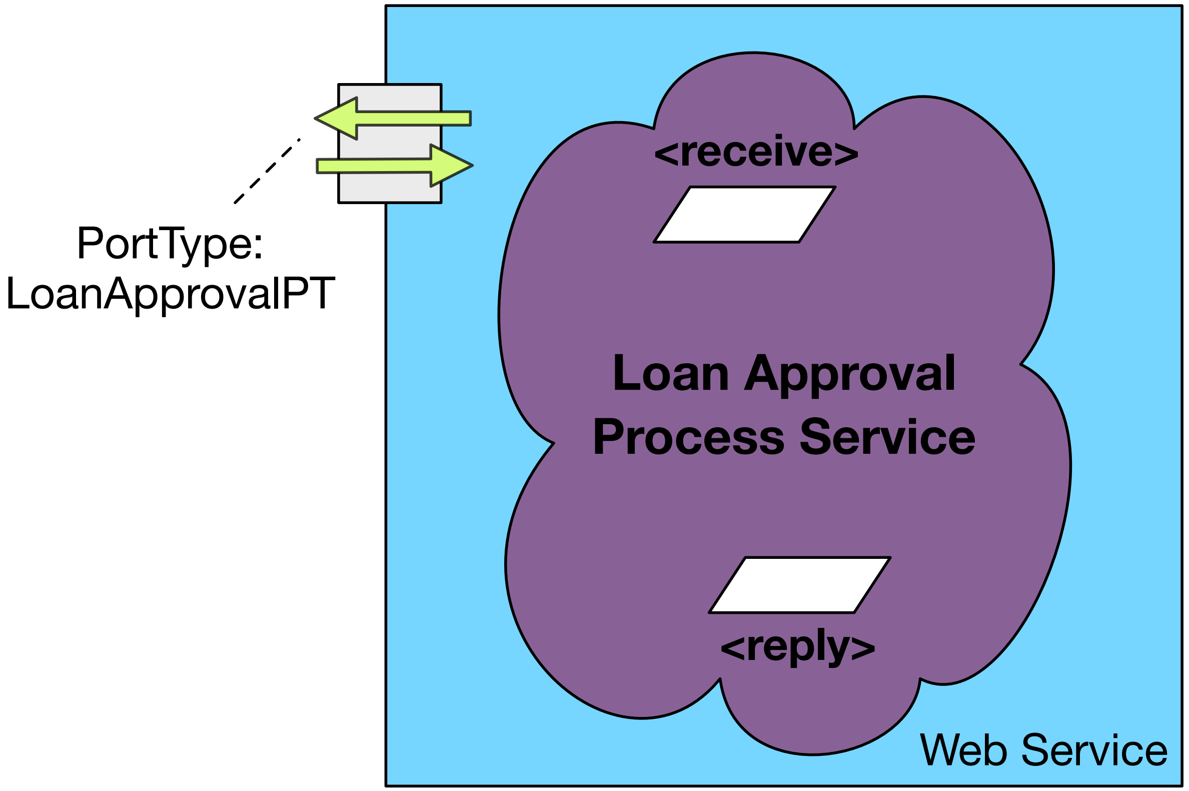


BPEL Design - I declare what I need and offer



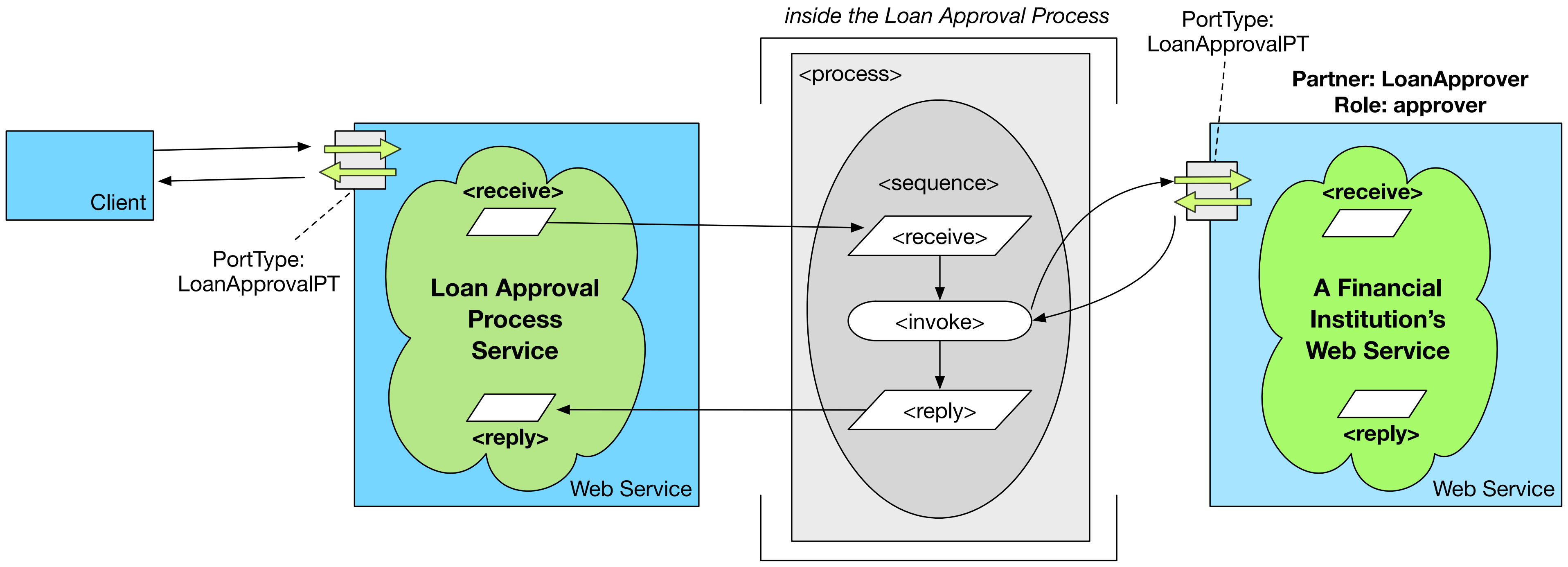
BPEL Basics via a Loan Approval Process Example

Loan Approval service is a composed Web service. Here is the external view of the loan approval process



BPEL Basics via a Loan Approval Process Example

Internal view of the loan approval process



First, the WSDL files involved ...

This WSDL declares the messages used in the service. It is declared separately so that it can be shared by other WSDLs.

loandefinition.wsdl:

```
<definitions
  targetNamespace="http://tempuri.org/services/loandefinitions"
  xmlns:tns="http://tempuri.org/services/loandefinitions"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="creditInformationMessage">
    <part name="firstName" type="xsd:string"/>
    <part name="name" type="xsd:string"/>
    <part name="amount" type="xsd:integer"/>
  </message>
  <message name="loanRequestErrorMessage">
    <part name="errorCode" type="xsd:integer"/>
  </message>
</definitions>
```

First, the WSDL files involved ...

This is the WSDL for the Loan Approver Service (loanapprover.wsdl) - i.e., the partner

```
<definitions
  targetNamespace="http://tempuri.org/services/loanapprover"
  xmlns:tns="http://tempuri.org/services/loanapprover"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:loandef="http://tempuri.org/services/loandefinitions"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
   <import namespace="http://tempuri.org/services/loandefinitions"
        location="http://localhost:8080/bpws-samples/loanapproval/
                  loandefinitions.wsdl"/>
   <message name="approvalMessage">
     <part name="accept" type="xsd:string"/>
   </message>
   <portType name="loanApprovalPT">
     <operation name="approve">
       <input message="loandef:creditInformationMessage"/>
       <output message="tns:approvalMessage"/>
       <fault name="loanProcessFault"
              message="loandef:loanRequestErrorMessage"/>
     </operation>
   </portType>
 <binding ...> ... </binding>
 <service name="LoanApprover">....</service>
 </definitions>
```

First, the WSDL files involved ...

This is the WSDL for the Loan Approval Process Service (loanapproval.wsdl) - i.e., the service provider itself

```
<definitions
       targetNamespace="http://loans.org/wsdl/loan-approval"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
       xmlns:lns="http://loans.org/wsdl/loan-approval"
       xmlns:apns="http://tempuri.org/services/loanapprover">
   <import namespace="http://tempuri.org/services/loanapprover"
           location="http://localhost:8080/bpws-samples/loanapproval/
                     loanapprover.wsdl"/>
   <import namespace="http://tempuri.org/services/loandefinitions"
           location="http://localhost:8080/bpws-samples/loanapproval/
                     loandefinitions.wsdl"/>

   <plnk:partnerLinkType name="loanApprovalLinkType">
     <plnk:role name="approver">
       <plink:portType name="apns:loanApprovalPT"/>
     </plnk:role>
   </plnk:partnerLinkType>
   <service name="loanapprovalServiceBP"/>
</definitions>
```

Creating the BPEL process definition: loanapproval.bpel

- **Create the BPEL process: (Start)**

```
<process name="loanApprovalProcess"  // all namespaces
            targetNamespace="http://acme.com/simpleloanprocessing"
            xmlns:loandef="http://tempuri.org/services/loandefinitions"
            xmlns:apns="http://tempuri.org/services/loanapprover" >
```

- **Define Partners:**

```
 <partners>
     <partner name="customer"
                 partnerLinkType="lns:loanApproveLinkType"
                 myRole="approver"/>
     <partner name="approver"
                 partnerLinkType="lns:loanApprovalLinkType"
                 partnerRole="approver"/>
  </partners>
```

- **Define variables:**

```
<variables>
     <variable name="request"
                messageType="loandef:CreditInformationMessage"/>
     <variable name="approvalInfo"
                messageType="apns:approvalMessage"/>
</variables>
```

Activities in the process

Interactions: Receive, Invoke and Reply ...

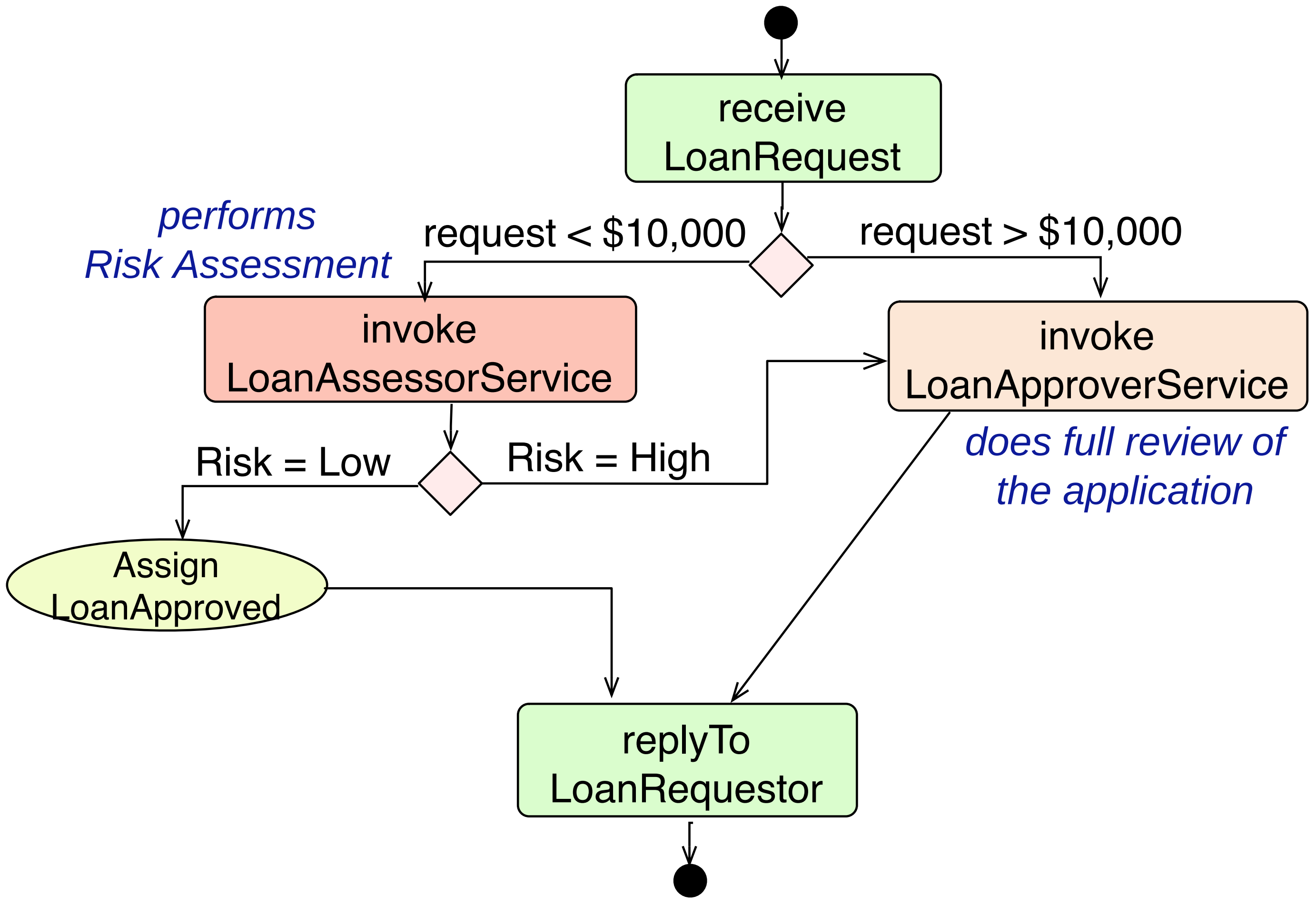
```
<sequence>
     <receive name="receive1" partner="customer"
                 portType="apns:loanApprovalPT"
                 operation="approve" variable="request"
                 createInstance="yes">
     </receive>

     <invoke name="invokeapprover"
                partner="approver"
                portType="apns:loanApprovalPT"
                operation="approve"
                inputVariable="request"
                outputVariable="approvalInfo">
     </invoke>

     <reply name="reply" partner="customer"
            portType="apns:loanApprovalPT"
            operation="approve" variable="approvalInfo">
     </reply>
  </sequence>
</process>
```

BPEL through Examples

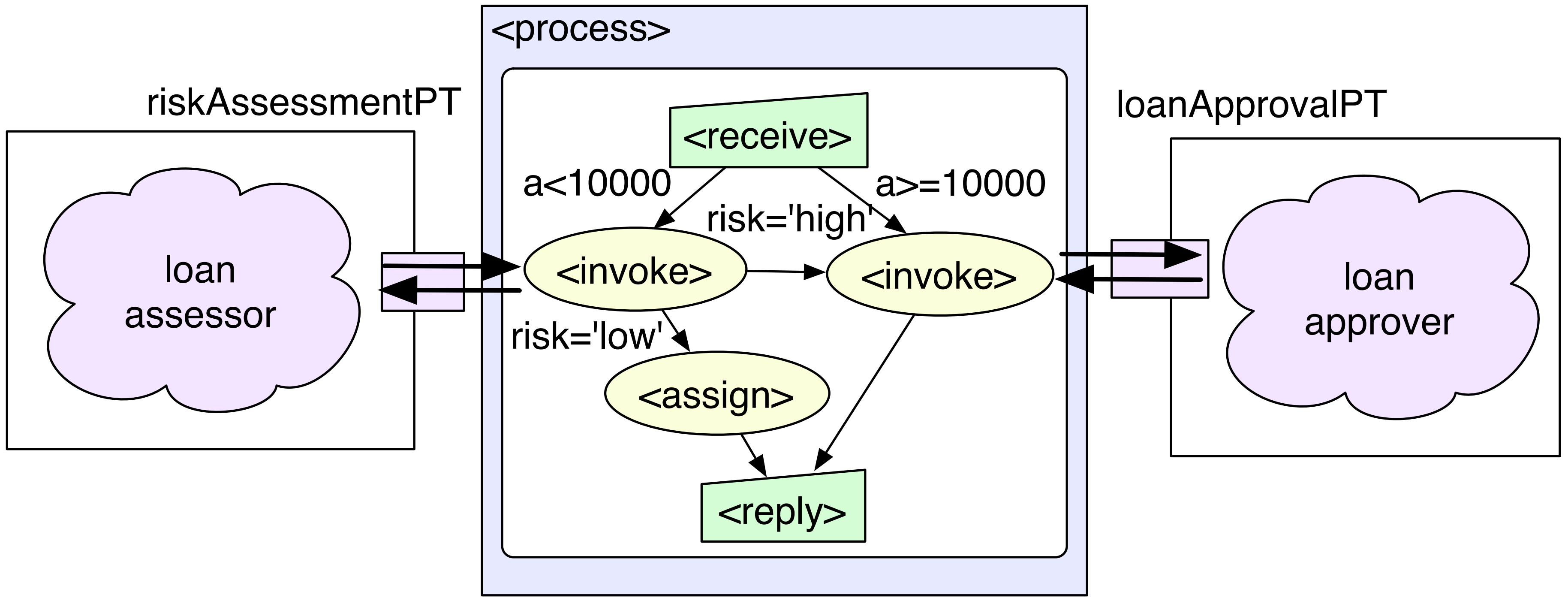
Example 1: (To learn: flow, links, conditions and assign activity:) Let us add an extra business logic and an additional service to the simple loan approval example.



www-128.ibm.com/developerworks/webservices/library/ws-bpelcol5/

BPEL Example 1

Internal view of the composition:



BPEL Example 1

A new service: Loan Assessor (loanassessor.wsdl)

```
<definitions   targetNamespace="http://tempuri.org/services/loanassessor"
     xmlns:tns="http://tempuri.org/services/loanassessor"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:loandef="http://tempuri.org/services/loandefinitions"
     xmlns="http://schemas.xmlsoap.org/wsdl/">
<import namespace="http://tempuri.org/services/loandefinitions"
        location="http://localhost:8080/bpws4j-samples/loanapproval/
        loandefinitions.wsdl"/>
<message name="riskAssessmentMessage">
  <part name="risk" type="xsd:string"/>
</message>
<portType name="riskAssessmentPT">
   <operation name="check">
     <input message="loandef:creditInformationMessage"/>
     <output message="tns:riskAssessmentMessage"/>
     <fault name="loanProcessFault"
            message="loandef:loanRequestErrorMessage"/>
   </operation>
</portType>
<binding ...> ... </binding>
   <service name="LoanAssessor">....</service>
</definitions>
```

BPEL Example 1

The new service's role and port type: assessor, riskAssessmentPT

```
<plnk:partnerLinkType name="riskAssessmentLinkType">
    <plnk:role name="assessor">
       <portType name="asns:riskAssessmentPT"/>
    </plnk:role>
</plnk:partnerLinkType>
```

(Added into loanapproval.wsdl)

In Loan Approval BPEL specification (loanapproval.bpel):

- adding the new partner
- defining a new variable to store the riskAssessment result.

BPEL Example 1

```
<process name="loanApprovalProcess"
            targetNamespace="http://acme.com/simpleloanprocessing"
            xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
            xmlns:lns="http://loans.org/wsdl/loan-approval"
            xmlns:loandef="http://tempuri.org/services/loandefinitions"
            xmlns:apns="http://tempuri.org/services/loanapprover"
            xmlns:asns="http://tempuri.org/services/loanassessor">
 <partners>
     <partner name="customer"
                 partnerLinkType="lns:loanApproveLinkType"
                 myRole="approver"/>
     <partner name="approver"
                 partnerLinkType="lns:loanApprovalLinkType"
                 partnerRole="approver"/>
     <partner name="assessor"
                 partnerLinkType="lns:riskAssessmentLinkType"
                 partnerRole="assessor"/>
  </partners>
<variables>
     <variable  name="request"
                messageType="loandef:CreditInformationMessage"/>
     <variable name="riskAssessment"
                messageType="asns:riskAssessmentMessage"/>
</variables>
<!-- activities to follow next -->
</process>
```

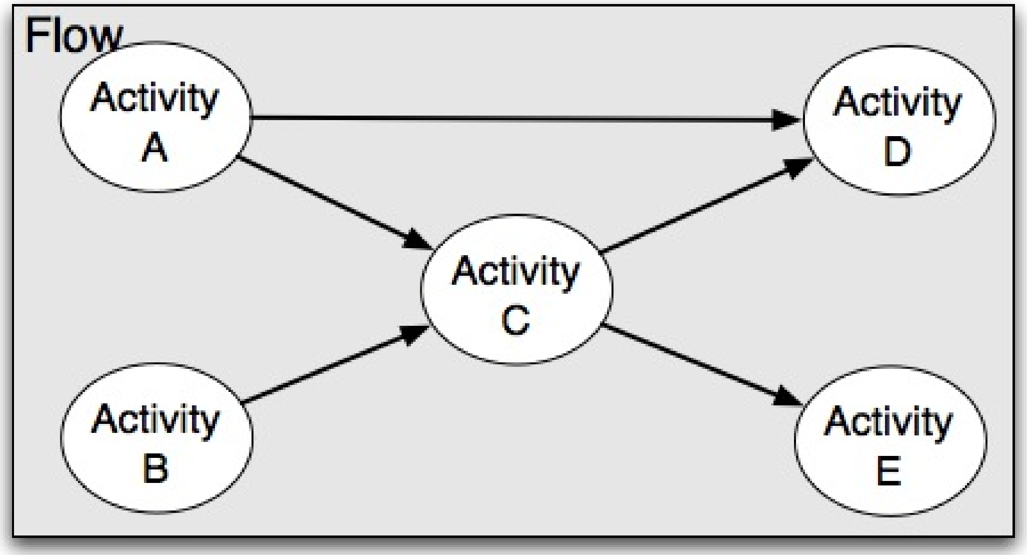
BPEL Example 1

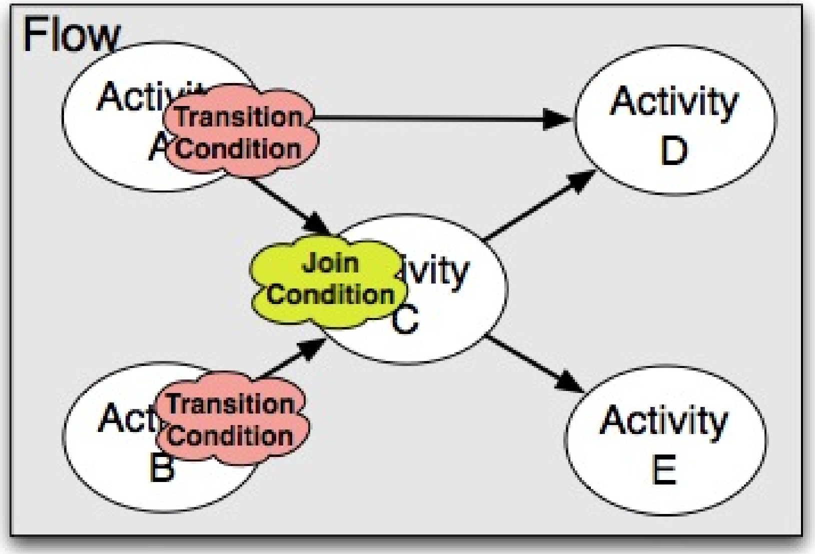
Now define the order of the activities:

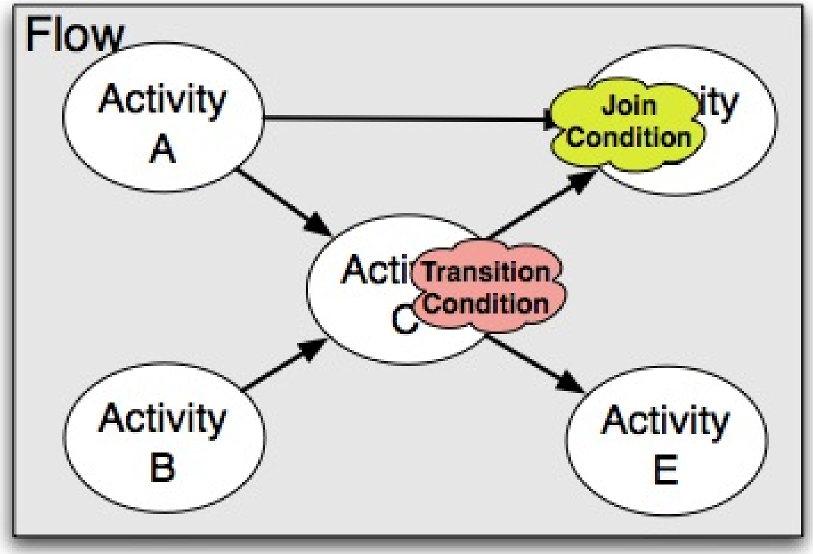
Flow and links:

- By default, Flow lets the activities to be run concurrently
- Flow may contain "links" that are designed to control the flow of activity executions
- <links> connects activities together and consists of one or more <link> element
- An individual <link> connects two activities.
- <link> and activities named in the link are enclosed by <flow>
- Definition of each activity contained in a link needs to identify whether it is 'source' or 'target' of a link.

BPEL: Flow and Links







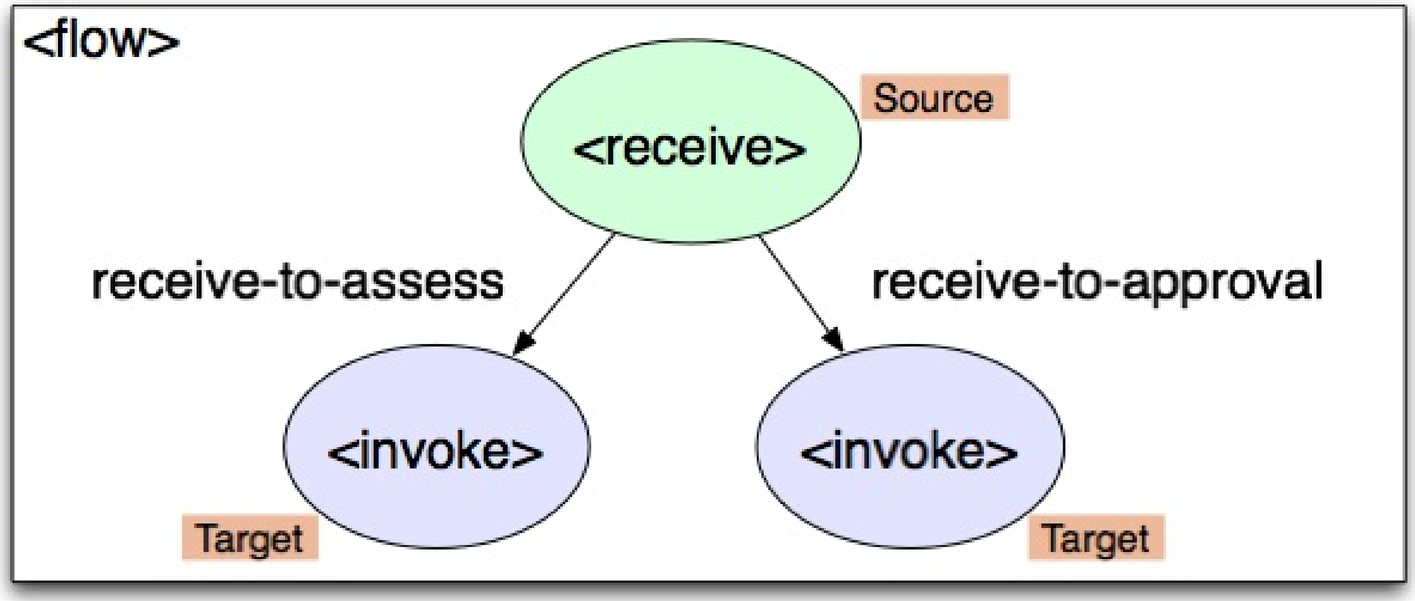
BPEL Example 1

```
<flow>
   <links>
      <link name="receive-to-assess"/>
      <link name="receive-to-approval"/>
   </links>
  <receive name="receive1" partner="customer"
         portType="apns:loanApprovalPT"
         operation="approve" variable="request" createInstance="yes">
    <source linkName="receive-to-assess"  .../>
    <source linkName="receive-to-approval"  .../>
  </receive>
  <invoke name="invokeAssessor" partner="assessor"
        portType="asns:riskAssessmentPT"
        operation="check" inputVariable="request"
        outputVariable="riskAssessment">
    <target linkName="receive-to-assess"/>
    <source linkName="assess-to-setMessage"  .../>
    <source linkName="assess-to-approval"  .../>
  </invoke>
  <invoke name="invokeapprover" partner="approver"
        portType="apns:loanApprovalPT"
        operation="approve"
        inputVariable="request"
        outputVariable="approvalInfo">
    <target linkName="receive-to-approval"/>
 </invoke>
</flow>
```

BPEL Example 1

Linking activities ...

An activity `A` may have outgoing links (when `A` is the source of the links) and incoming links (when `A` is the target of the links)



At the beginning, all links are inactive and only those with no dependencies (those that are not declared as target) can execute.

BPEL Example 1

The transition conditions on source (`transitionCondition`):

Consider this:

```
<receive name="receive1" partner="customer"
         portType="apns:loanApprovalPT"
         operation="approve" variable="request" createInstance="yes">
  <source linkName="receive-to-assess"
    transitionCondition="getVariableData('request','amount')<10000"/>
  <source linkName="receive-to-approval"
    transitionCondition="getVariableData('request','amount')>=10000"/>
</receive>
```

- Once the activity completes (i.e., `receive`), its outgoing links (ie., two sources) get evaluated using the transition condition
- the transition condition is an XPath expression
- A special XPath function:

  getVariableData ('variableName', 'partName','locationPath' ?)
- The evaluation results in `true` or `false`.

BPEL Example 1

According to `loanapproval.bpel`, '`request`' in the function `getVariableData` is mapped to the `CreditInfomationMessage` messsageType:

```
<variables>
  <variable name="request" messageType="loandef:CreditInformationMessage"/>
  ...
</variable>
```

which has:

```
<message name="creditInformationMessage">
   <part name="firstName" type="xsd:string"/>
   <part name="name" type="xsd:string"/>
   <part name="amount" type="xsd:integer"/>
</message>
```

Hence, the XPath function will return a number (ie., `amount`).

Depending on the result of the transition condition, the target activity of the link may (or may not!) become active.

BPEL Example 1

Data assignment: <assisgn> activity.

Consider this:

```
<invoke name="invokeAssessor" partner="assessor"
        portType="asns:riskAssessmentPT"
        operation="check" inputVariable="request"
        outputVariable="riskAssessment">
    ...
  <source linkName="assess-to-setMessage"
    transitionCondition="getVariableData('riskAssessment', 'risk')='low'"/>
  <source linkName="assess-to-approval"
    transitionCondition="getVariableData('riskAssessment', 'risk')!='low'"/>
</invoke>
```

- The result of 'check' is stored in the 'riskAssessment' message ('low' or 'high').
- If the message says 'low', we would like to set the correct response for 'approvalInfo' message (ie., yes).

BPEL Example 1

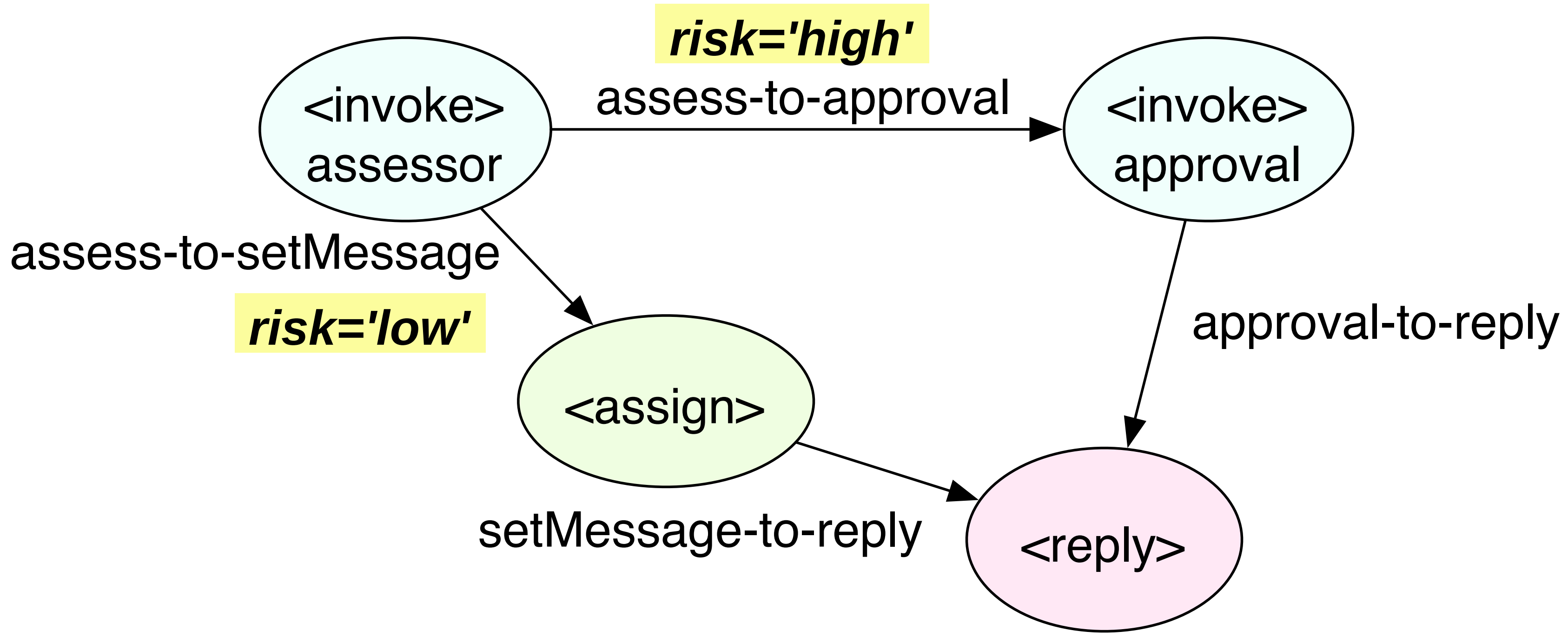
Define an <assign> activity as follows:

```
<assign name="assign">
   <target linkName="assess-to-setMessage"/>
   <source linkName="setMessage-to-reply"/>
   <copy>
      <from expression="'yes'"/>
      <to variable="approvalInfo" part="accept"/>
   </copy>
</assign>
```

Any general XPath expression is allowed in <from> element, as long as it returns a valid XPath value type (ie., string, number or boolean).

The result of <copy> is that string 'yes' is copied to the 'accept' part of the 'approvalInfo' message.

BPEL Example 1



The assign activity is the target of 'assess-to-setMessage' link, which means it is activated after 'assessor' is done.

After <assign> is completed, the outgoing link (source) is evaluated (`true`, in this case.)

BPEL Example 1

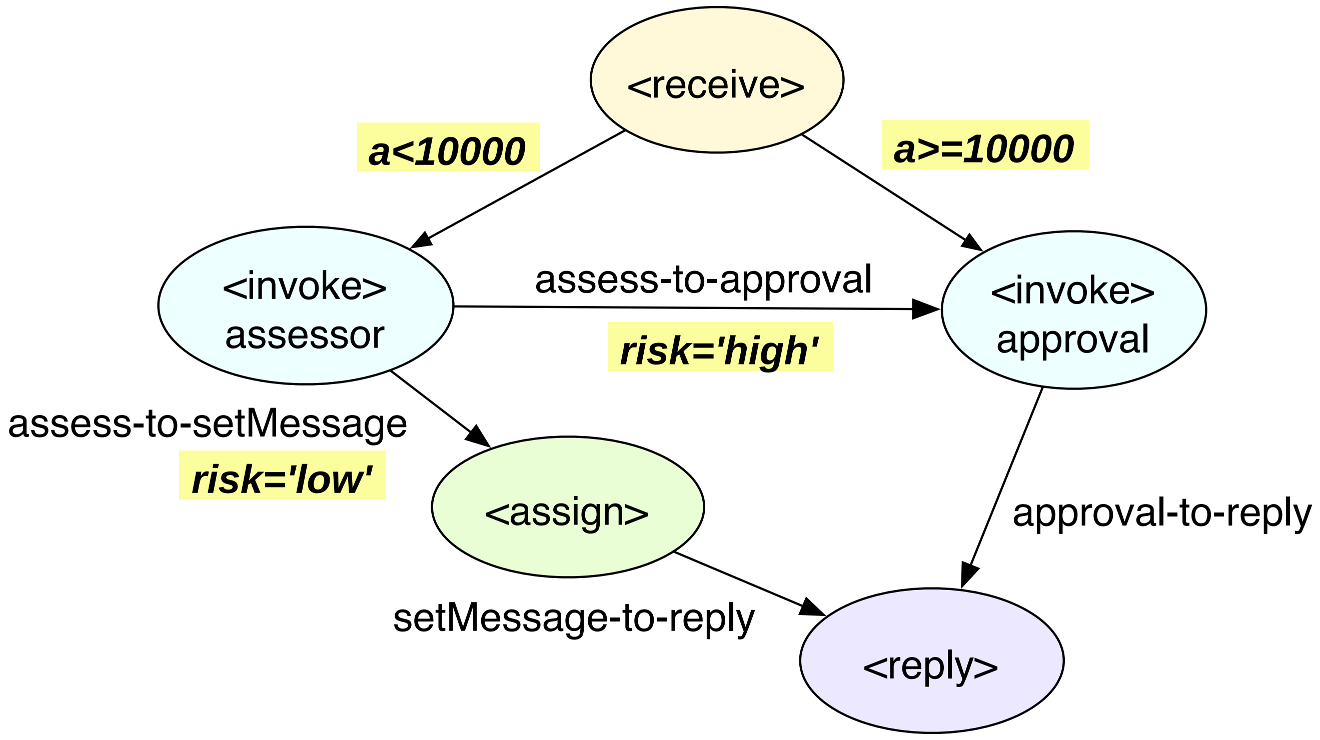
Consider that there is more than one target (i.e., more than one incoming links). How do we synchronising activities?

```
<invoke name="invokeapprover"
        partner="approver" portType="apns:loanApprovalPT"
        operation="approve"
        inputVariable="request"
        outputVariable="approvalInfo">
   <target linkName="receive-to-approval"/>
   <target linkName="assess-to-approval"/>
   <source linkName="approval-to-reply" />
</invoke>
<reply name="reply" partner="customer"
       portType="apns:loanApprovalPT"
       operation="approve" variable="approvalInfo">
   <target linkName="setMessage-to-reply"/>
   <target linkName="approval-to-reply"/>
</reply>
```

Each activity that is the target of a link has an implicit/explicit `joinCondition` attribute that is used to evaluate the state.

BPEL Example 1

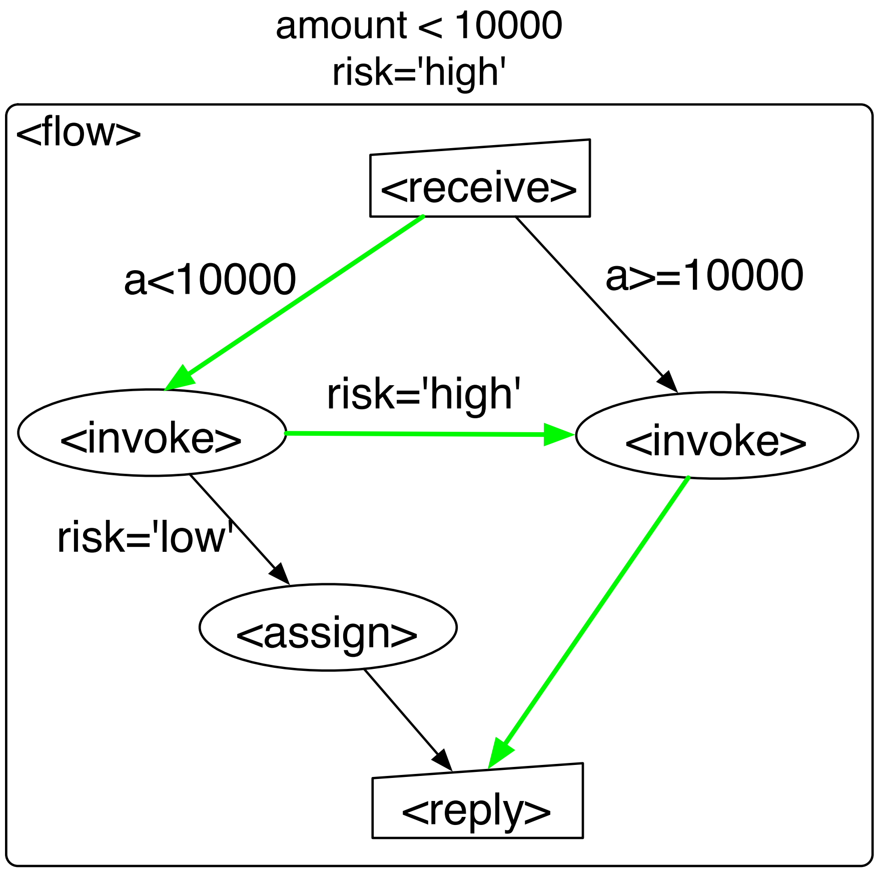
The default semantics of the condition (ie., implicit join condition): wait until the status of all 'incoming' links has been determined. If at least one of the 'incoming' links is true, the activity (ie., `invoke` and `receive`) itself is activated.

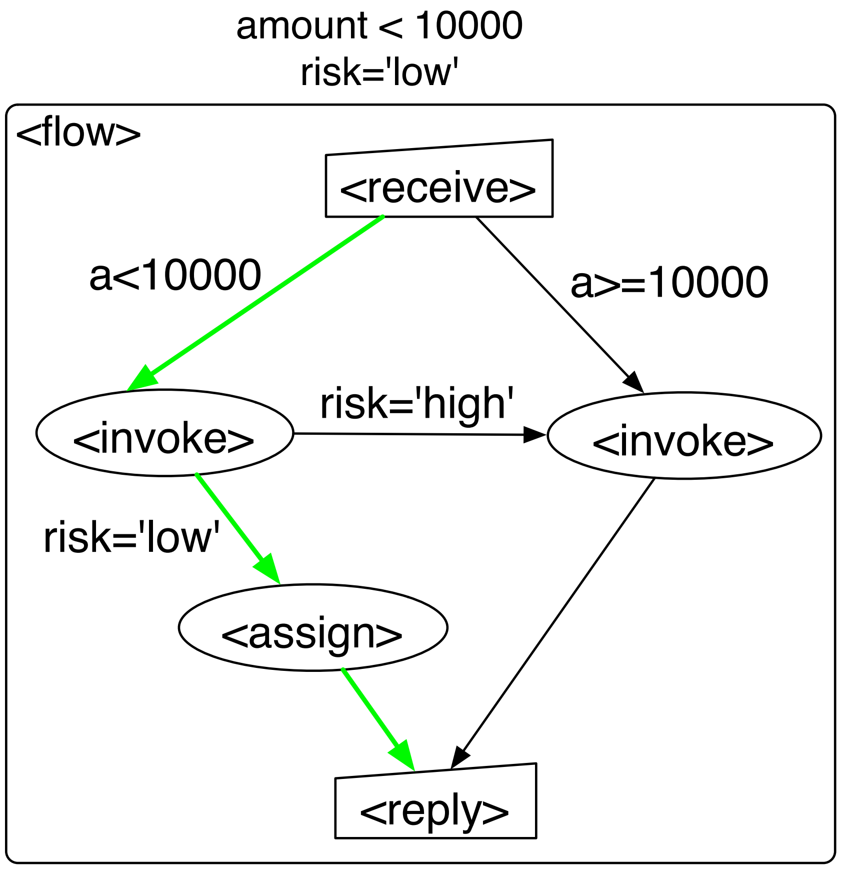


For explicit join condition, you can use bpws:getLinkStatus.

BPEL Example 1: Recap

Green links (or pale coloured in b/w) are evaluated with `true` and black lines are `false` in each run.





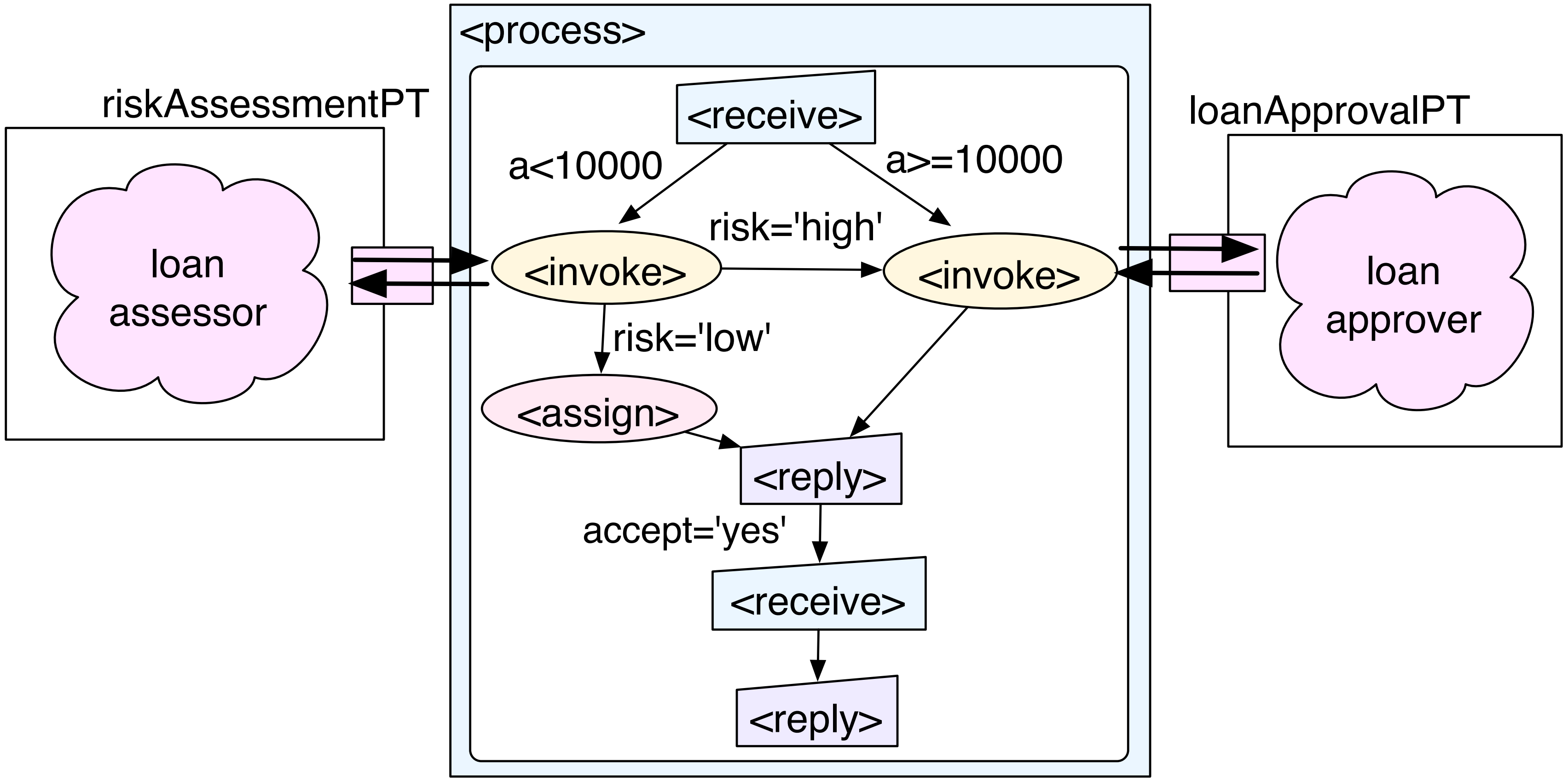
BPEL through Examples

Example 2: (To learn: message correlation) Add an extra business logic that allows the customers to actually obtain the loan that they have been approved for. We'll add a `receive` activity to accept the request to obtain the loan, and a `reply` activity to acknowledge it has been obtained.

Lifecycle of a BPEL process

- BPEL specification is a template for creating business process instances
- Every BPEL process must have at least one 'start activity' that is marked by 'createInstance="yes"'
- 'Message correlation set' is used to route messages to the correct instances
- A process terminates when all activities are complete, or fault reaches the process scope, or terminate activity is defined.

BPEL Example 2



- To achieve this, whenever a message arrives, you need to identify which instance of the business process it belongs to (ie., stateful conversation)

BPEL Example 2

So now added in `loanapproval.wsdl`:

```
<portType name="loanApprovalPT">
   <operation name="obtain">
     <input message="loandef:creditInformationMessage"/>
     <output message="apns:approvalMessage"/>
   </operation>
</portType>
<plnk:partnerLinkType name="loanApprovalLinkType">
   <slnk:role name="approver">
     <portType name="apns:loanApprovalPT"/>
     <portType name="lns:loanApprovalPT"/>
   </plnk:role>
</plnk:partnerLinkType>
<variables>
    <variable name="acceptanceRequest"  // new variable
        messageType="loandef:creditInformationMessage"/>
 ...
```

BPEL Example 2

What we need here is to make sure that the request to obtain the loan arrives at the same process instance that provided the initial approval. (ie., enabling a 'stateful' conversation)

In BPEL, this is called – **Message Correlation**. It refers to routing of a message from a known customer to the correct instance of a business process.

A BPEL instance is identified by a unique `correlationSet`.

- one or more sets of key data fields within the exchanged messages
- eg., an order number
- eg., customer's lastname and date of birth.
- Use XPath expressions to identify the key data fields in the messages

BPEL Example 2

Defining a correlation set:

First, define what WSDL 'properties' we correlated on (ie., what will be used as a unique identifier?)

In this example, we assume that customer names are unique. In `loanapproval.wsdl`, add the following two WSDL *'properties'*:

```
<bpws:property name="applicantFirstName" type="xsd:string"/>
<bpws:property name="applicantLastName" type="xsd:string"/>
```

Then define how the 'properties' are extracted:

```
<bpws:propertyAlias propertyName="lns:applicantFirstName"
                  messageType="loandef:creditInformationMessage"
                  part="firstName"
                  query="/firstName"/>

<bpws:propertyAlias propertyName="lns:applicantLastName"
                  messageType="loandef:creditInformationMessage"
                  part="name"
                  query="/name"/>
```

BPEL Example 2

Now define a correlation set:

```
<correlationSets>
  <correlationSet name="loanIdentifier"
           properties="lns:applicantFirstName lns:applicantLastName"/>
</correlationSets>
```

The set needs to be initialised before the second `receive` activity (eg., the first `receive` activity is an obvious choice).

Initialisation makes the tie between a process instance and the set.

BPEL Example 2

```
<receive name="receive1" partner="customer"
     portType="apns:loanApprovalPT"
     operation="approve" variable="request"
     createInstance="yes">
 <source linkName="receive-to-assess"
   transitionCondition="bpws:getVariableData('request','amount')<10000"/>
 <source linkName="receive-to-approval"
   transitionCondition="bpws:getVariableData('request','amount')>=10000"/>
 <correlations>
    <correlation set="loanIdentifier" initiation="yes"/> // initialised
 </correlations>
</receive>
```

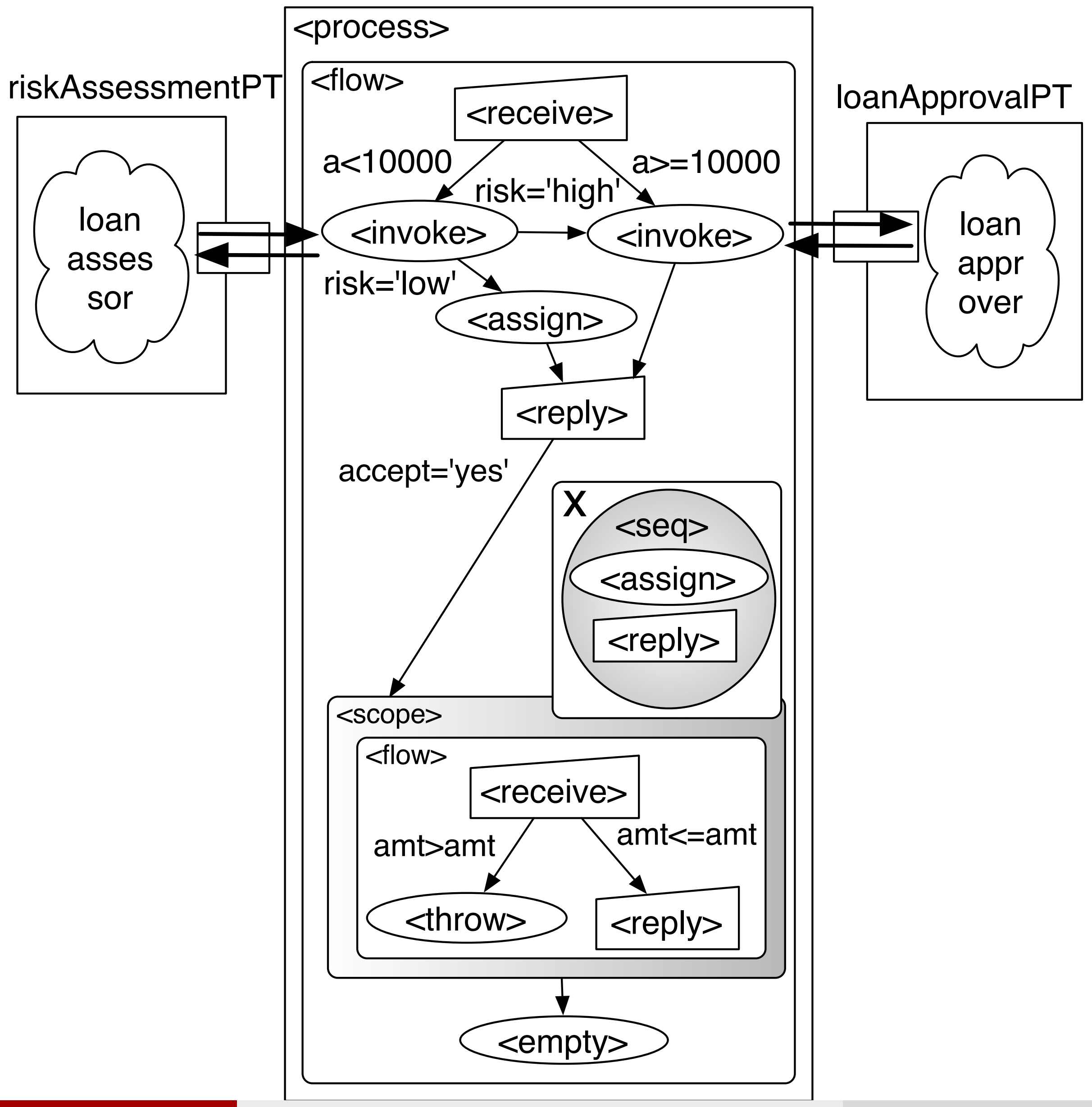
When `receive` needs to run, the incoming message is checked (ie., values of the fields specified in the correlation set against the initialised values for the set).

BPEL through Examples

Example 3: (To learn error handling in BPEL; scope, faultHandlers, throw elements:) Add a fault handler which notifies the customers when they try to obtain a loan that is higher than the one they have been approved for.

- <faultHandlers>: When an error occurs, BEPL provides a mechanism to catch the error and handle it by executing another activity defined by a *fault handler*.
- An error handling usually affects a set of activities associated with each other. In BPEL, this is done by enclosing them in a $< scope >$.
- <scope> can be considered as a possibly recoverable, compensatable unit of work.

BPEL Example 3



BPEL Example 3

The fault handler definition and error throwing:

```
<scope name="new-scope">  // scope for error handling
  <source linkName="scope-to-empty"/>
  <faultHandlers>
   <catch faultName="lns:loanProcessFault"
          faultVariable="error">
    <sequence name="fault-sequence">
      <assign>
        <copy>
          <from expression="'invalid request: amount too high'"/>
          <to variable="approvalInfo" part="accept"/>
        </copy>
      </assign>
      <reply partner="customer" portType="lns:loanApprovalPT"
             operation="obtain" variable="approvalInfo"
             faultName="lns:loanProcessFault"/>
    </sequence>
   </catch>
  </faultHandlers>
```

BPEL Example 3

```
<flow name="inner-flow">
   <receive name="acceptance-receive" partner="customer"
           portType="lns:loanApprovalPT"
          operation="obtain" variable="acceptanceRequest">
     <target linkName="reply-to-receive"/>
     <source linkName="receive-to-grant"
             transitionCondition=
             "bpws:getVariableData('acceptanceRequest','amount')
             <=bpws:getVariableData('request', 'amount')"/>
     <source linkName="receive-to-fail"
             transitionCondition=
             "bpws:getVariableData('acceptanceRequest', 'amount')
             >bpws:getVariableData('request', 'amount')"/>
     <correlations>
       <correlation set="loanIdentifier"/>
     </correlations>
  </receive>
```

BPEL Example 3

```
    <reply name="grant-reply" partner="customer"
           portType="lns:loanApprovalPT" operation="obtain"
           variable="approvalInfo">
     <target linkName="receive-to-grant"/>
    </reply>
    <throw name="grant-failure" faultName="lns:loanProcessFault">
        <target linkName="receive-to-fail"/>
    </throw>
  </flow>
 </scope>
 <empty name="the-last-one">
    <target linkName="scope-to-empty"/>
 </empty>
</flow>
```

There are also other activities that are useful: $<$ *switch* $>$, $<$ *pick* $>$

References:

- http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm
- Apache ODE project (ode.apache.org/)
- Eclipse BPEL Designer Project - www.eclipse.org/bpel/