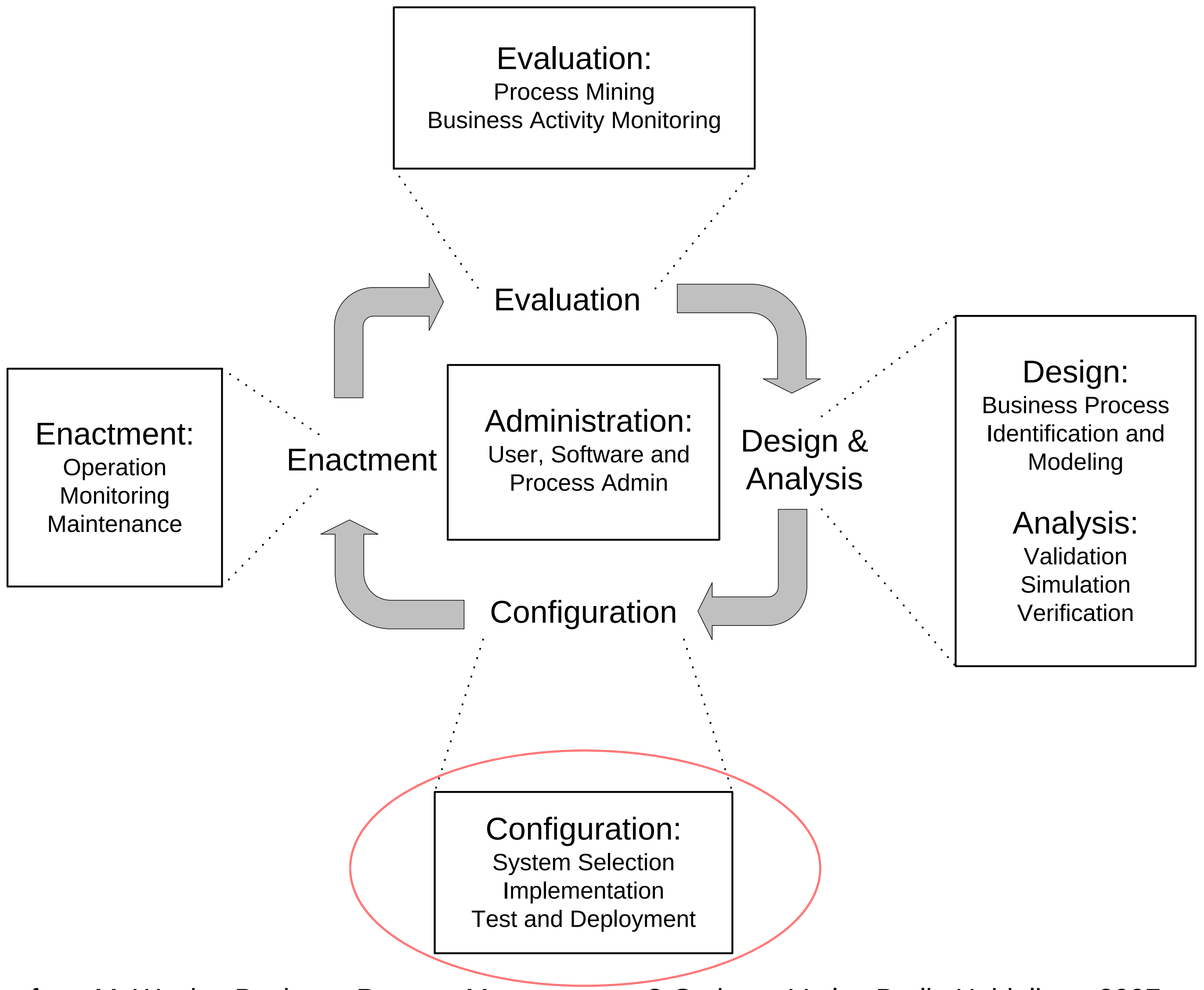
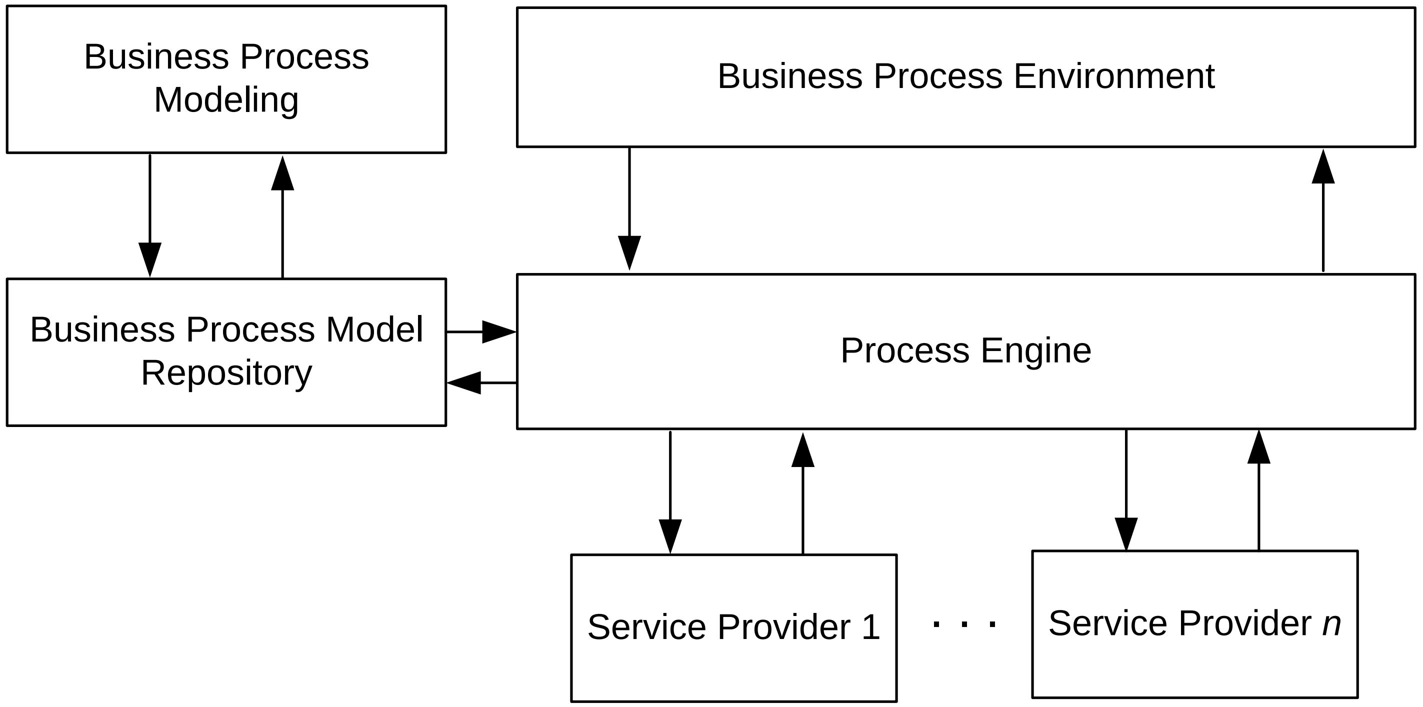
Topic 2: Workflow Systems Architectures & BPEL



from M. Weske: Business Process Management, © Springer-Verlag Berlin Heidelberg 2007

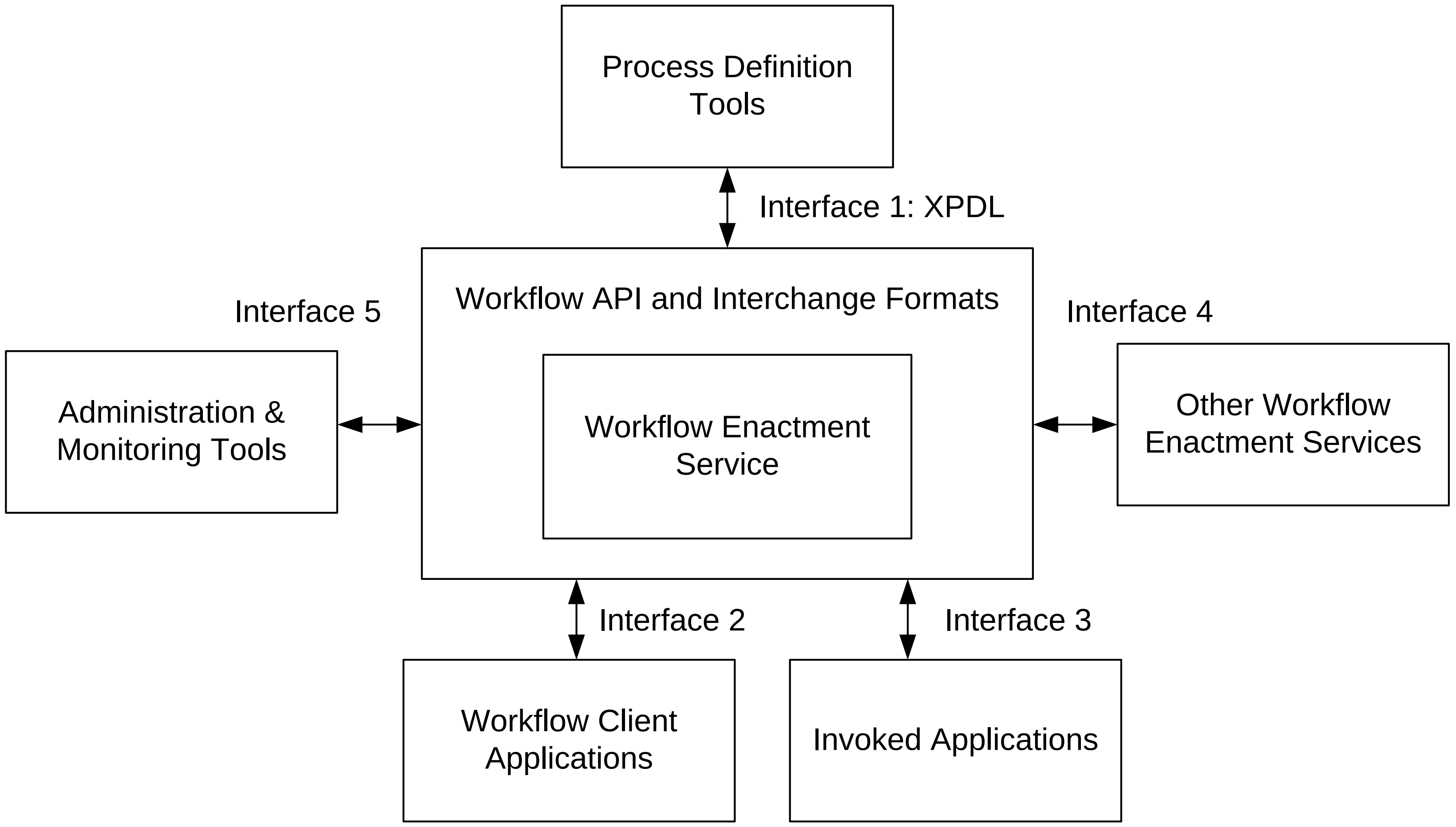
Workflow-Architectures


Business Process Modeling
Business Process Environment
Business Process Model Repository
Process Engine
Service Provider 1
. . .
Service Provider *n*

WfMC Reference Architecture

- Workflow Management Coalition
  - Interest group in Systems and application development
  - Major outcome: WfMC reference architecture (1990ies)
- Objective of WfMC reference architecture
  - Provide uniform interfaces in order to achieve interoperability between systems (and subsystems) of different manufacturers
    - Process modeling tool of provider A can be combined with runtime environment of provider B.
  - Exchange format XPDL: XML Process Definition Language
- Hint
  - WfMC has partially received its target

WfMC Reference Architecture


Process Definition Tools
Interface 1: XPDL
Workflow API and Interchange Formats
Interface 5
Interface 4
Administration & Monitoring Tools
Workflow Enactment Service
Other Workflow Enactment Services
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

from M. Weske: Business Process Management, © Springer-Verlag Berlin Heidelberg 2007

Service-based Architectures

- Objective
  - Re-usable and well-defined business functionality is provided by services (services)
  - Create new applications and adapt existing applications easily and inexpensively
- Requirements
  - Service descriptions must be accessible and sufficiently precise
  - Identification, specification and realization of business functionality through services (*service carving*)
  - Implementation of Services (*service enabling*)

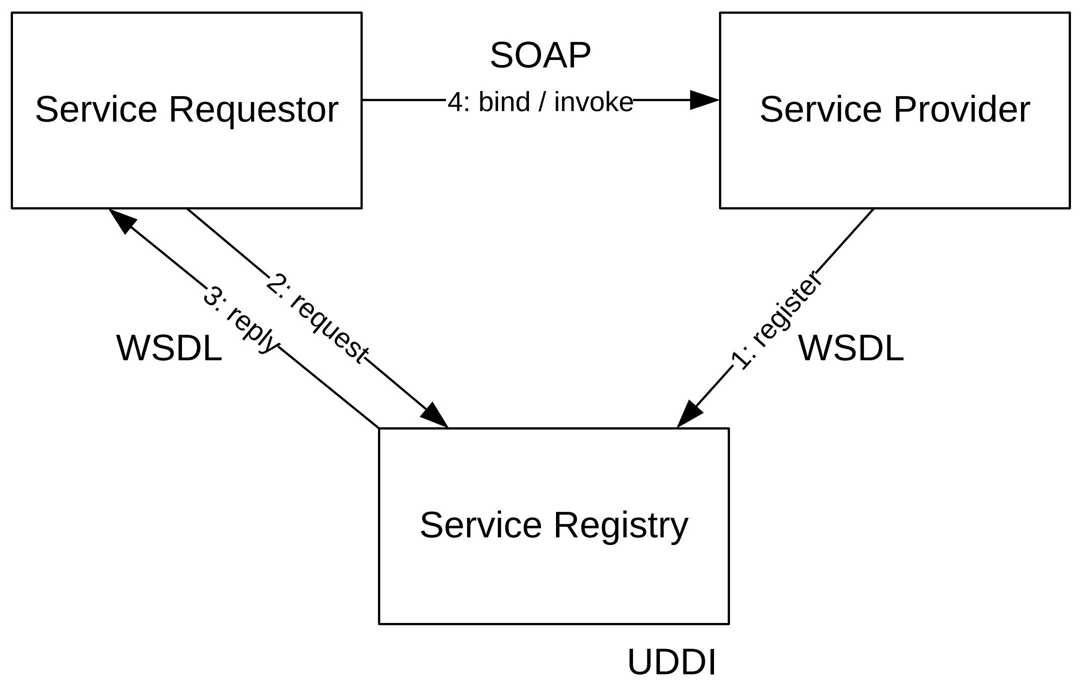
W3C Web-Services

- Current implementation of service oriented architectures
- Characterization
  - *Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the web.*
  - *Web services perform functions, which can be anything from simple requests to complicated business processes.*
  - *Once a web service is deployed, other applications (and other web services) can discover and invoke the deployed service.*
  - *XML messaging is used to interact with a web service.*

Web-Services

- Central Standards
  - *SOAP:* XML Message formatting
  - *Web Service Description Language WSDL*: Format for the specification of services and their methods, and input message and the output message
    - Logical and physical aspects are described
  - *Universal Description, Discovery, and Integration (UDDI):* Structured storage of service descriptions and descriptions of service providers and request functionality
- Hint
  - UDDI is in contrast to SOAP and WSDL not widely accepted today

Web Service Triangle



from M. Weske: Business Process Management, © Springer-Verlag Berlin Heidelberg 2007

Web-Service Composition

- Idea
  - System workflows in service-oriented environments are realized by composing Web services
  - Concept is recursive, that is a service composition can in turn be described as a service using WSDL and be part of a higher service composition
- Industry standard: WSBPEL, Business Process Execution Language for Web Services
  - Combination of WSFL (Web Services Flow Language) from IBM and XLANG from Microsoft
  - Very powerful language with support for complex control flow
  - OASIS-Standard, 2007

OASIS Advancing open standards for the information society

WSBPEL Design Goals

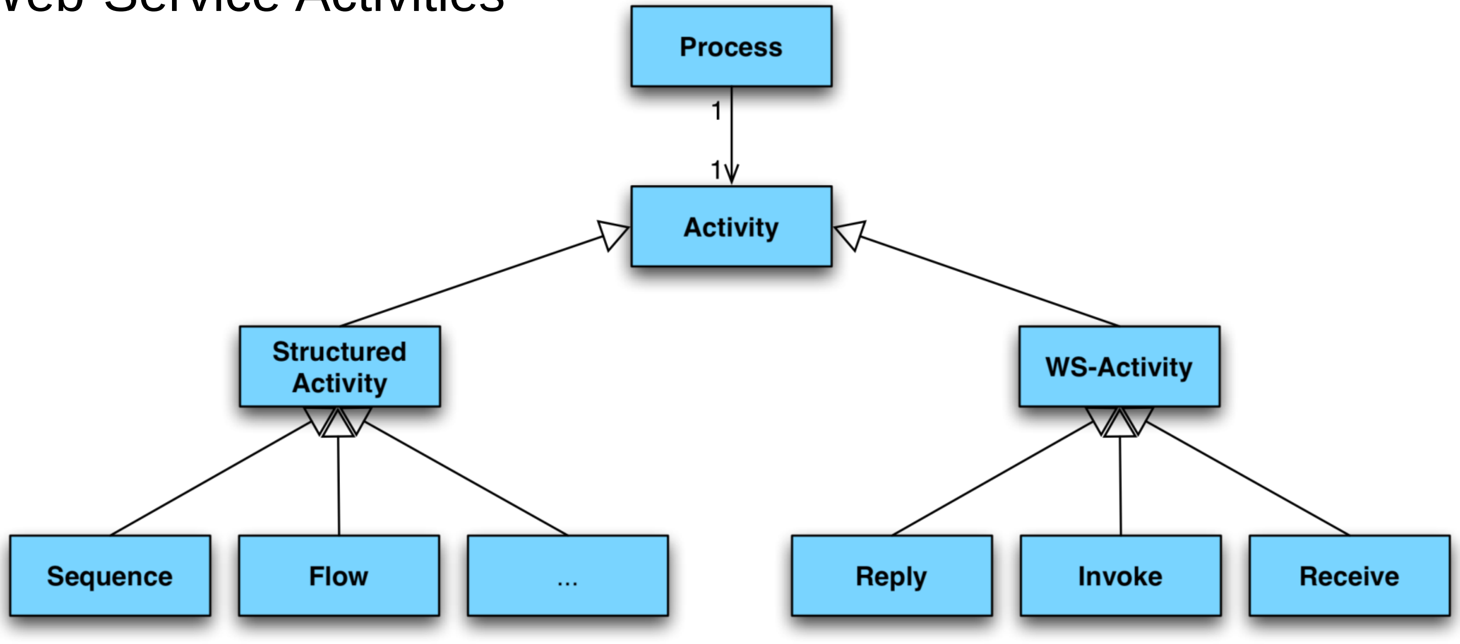
- WSBPEL bases on W3C Web-Services
  - WSBPEL processes interact with Web services, which were described by WSDL
  - Structure of these processes is described by a corresponding XML schema definition
  - WSBPEL processes have no graphical representation
  - WSBPEL "inherits" from
    - XLANG (block structure with special control flow constructs)
    - WSFL (graph structure with transition condition)
  - WSBPEL allows these two views to combine the modeling of processes together

WSBPEL Design Goals

- Data Management
  - Data-dependent control flow can be defined
  - Data in web services messages are used to analyse conditions affecting control flow.
- Correlation
  - Process instances have unique identifiers
  - Partner-Organisations can use different process instance identifier
  - Correlation is defined by properties of messages
- Modularization
  - WSBPEL-Process can be a service of its own that is described by a WSDL file and thus can be part of another service-composition.

WSBPEL Concepts

- Each process consists of exactly one <process> element that can contain other activities.
- Types of Activities
  - Structured activities
  - Web-Service Activities



Web-Services Activities

- Invoke: An operation of a Web service is invoked. This may possibly have an answer
- Receive: Awaiting receipt of a message
  - *createInstance=„yes“* signals process instantiation
- Reply: Send a reply in response to receipt of a message
- Wait: A defined period of time to wait
- Assign: Assignment of data values, for example of a received message to a process variable
- Throw: Show errors for exception handling
- Terminate: terminate the whole process instance.

WSBPEL Control flow, block structured

- Sequence

```
<sequence>
  <!-- activities -->
</sequence>
```

- Switch / Case

```
<switch>
  <case condition = “condition”> <!-- activity --> </case>
  <case condition = “condition”> <!-- activity --> </case>
</switch>
```

- While

```
<while condition = “condition”>
  <!-- activity -->
</while>
```

WSBPEL Control flow, block structured

- Pick: Waiting for an event from a set of possible events (deferred choice, event-based XOR split)

```
<pick>
  <onMessage .../>
  <onAlarm .../>
</pick>
```

- If: conditional branch

```
<if condition = “condition”> activity
<elseif condition = “condition”>activity</elseif>
<else> activity</else></if>
```

- Flow: concurrent execution

```
<flow>
  <!-- activities -->
</flow>
```

WSBPEL Control flow, block structured

– *Scope:* Allows defining a notion of sub-process. In scopes, you can define variables, messages, other control flows with their exception handling etc.

```
<scope>
  <partnerLinks>
    <!-- Partner link definitions local to scope. -->
  </partnerLinks>
  <messageExchanges>
    <!-- Message exchanges local to scope.-->
  </messageExchanges>
  <variables>
    <!-- Variable definitions local to scope. -->
  </variables>
  <correlationSets>
    <!-- Correlation sets local to scope.-->
  </correlationSets>
  <faultHandlers>
    <!-- Fault handlers local to scope. -->
  </faultHandlers>
activity
</scope>
```

```
<scope>
  <faultHandlers>
    <catch faultName="emp:WrongEmployeeName" >
      <!-- Perform an activity --></catch>
    <catch faultName="emp:TravelNotAllowed"
           faultVariable="Description" >
      <!-- Perform an activity →</catch>
   <catchAll>
     <!-- Perform an activity -->
   </catchAll>
  </faultHandlers>
  <invoke partnerLink="employeeTravelStatus"
          portType="emp:EmployeeTravelStatusPT"
          operation="EmployeeTravelStatus"
          inputVariable="EmployeeTravelStatusRequest"
          outputVariable="EmployeeTravelStatusResponse" >
  </invoke>
</scope>
```

WSBPEL control flow, graph structured

- Link defines an execution order between activities
- Anchor: Naming of links

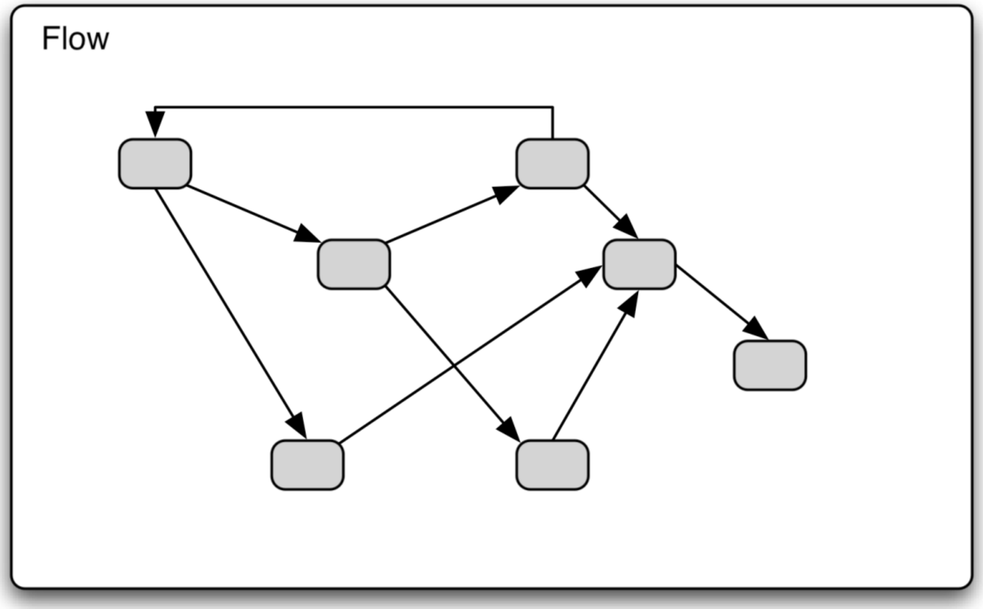
  <links>
  <link name="link1"/>
  </links>

- Source

  <someActivity name = "X">
  <source linkName="link1"/>
  </someActivity>

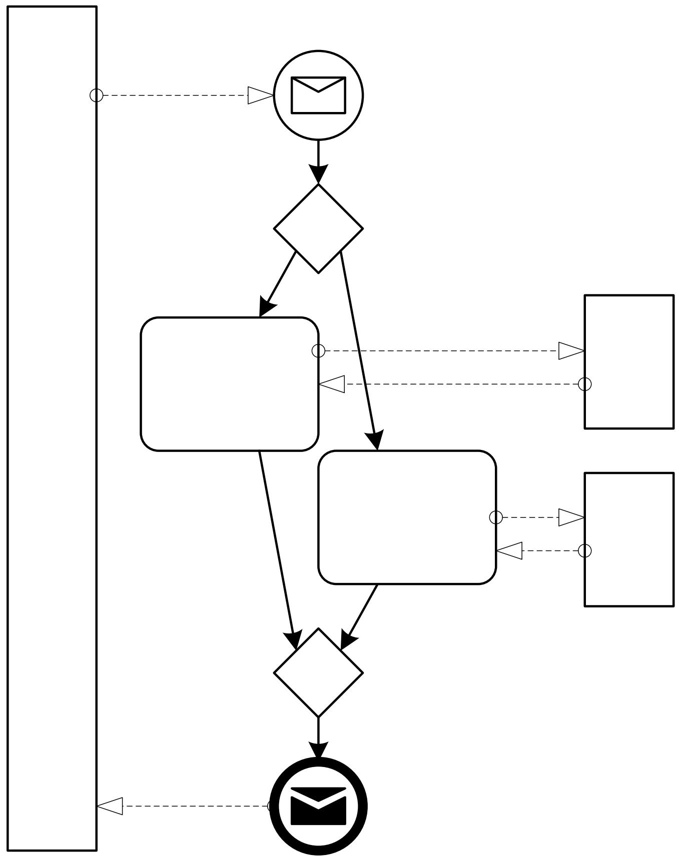
- Target

  <someActivity name = "Y">
  <target linkName="link1"/>
  </someActivity>



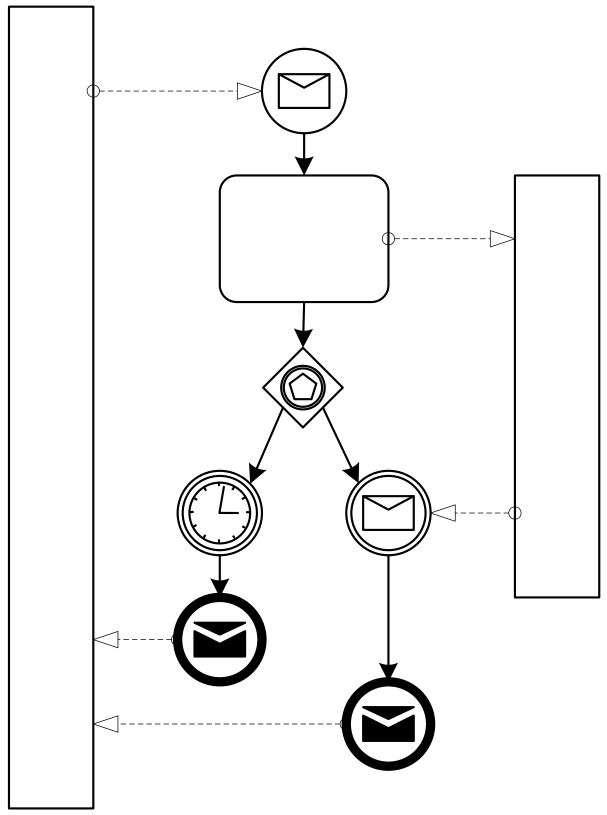
Control flow example 1

```
<process>
    <sequence>
        <receive createInstance=„yes“ />
        <if>
            <invoke />
            <else>
                <invoke />
            </else>
        </if>
        <reply />
    </sequence>
</process>
```



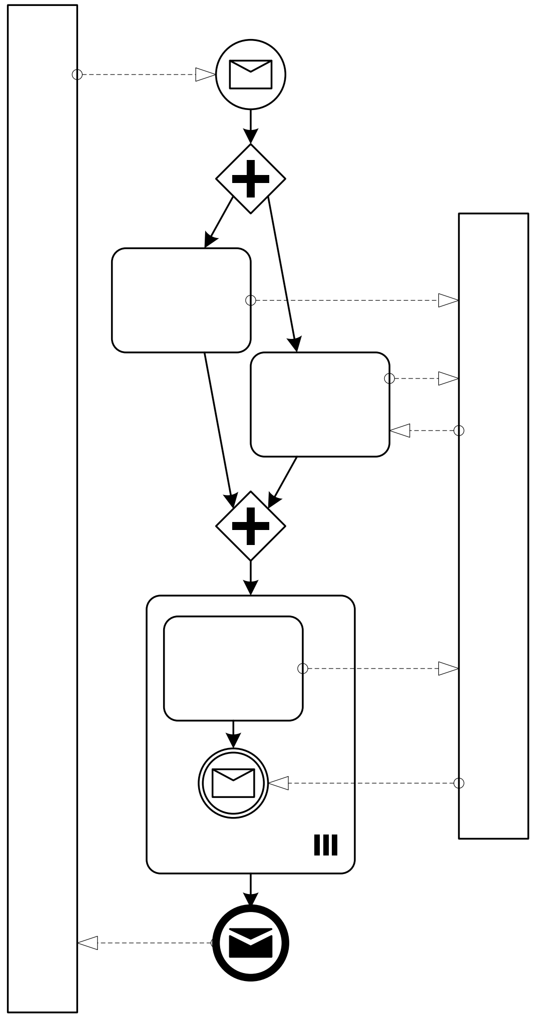
Control flow example 2

```
<process>
  <sequence>
    <receive createInstance=„yes“ />
    <invoke />
    <pick>
      <onMessage>
        <reply />
      </onMessage>
      <onAlarm>
        <reply />
      </onAlarm>
    </pick>
  </sequence>
</process>
```



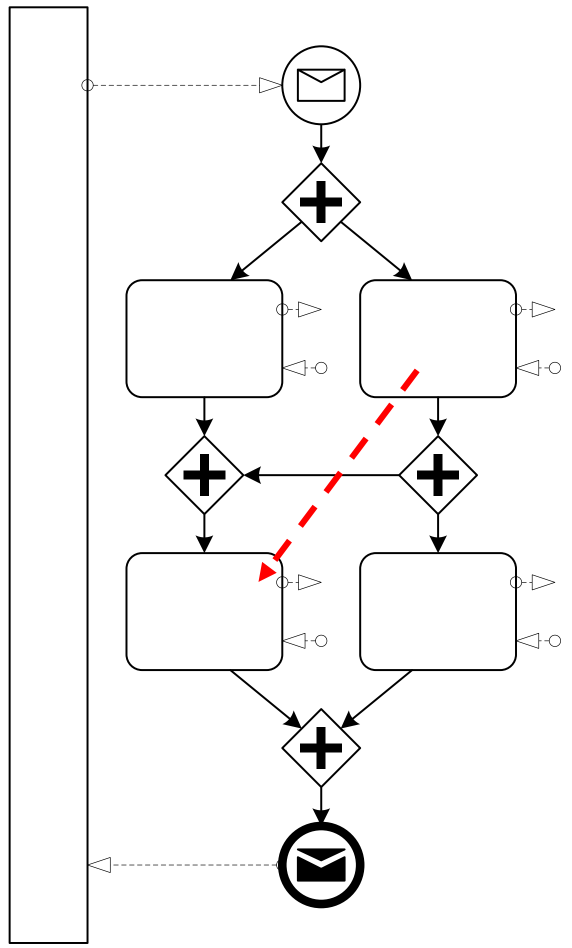
Control flow example 3

```
<process>
  <sequence>
    <receive createInstance=„yes“ />
    <flow>
      <invoke />
      <invoke />
    </flow>
    <forEach>
      <scope>
        <sequence>
          <invoke />
          <receive />
        <sequence>
      </scope>
    </forEach>
    <reply />
  </sequence>
</process>
```

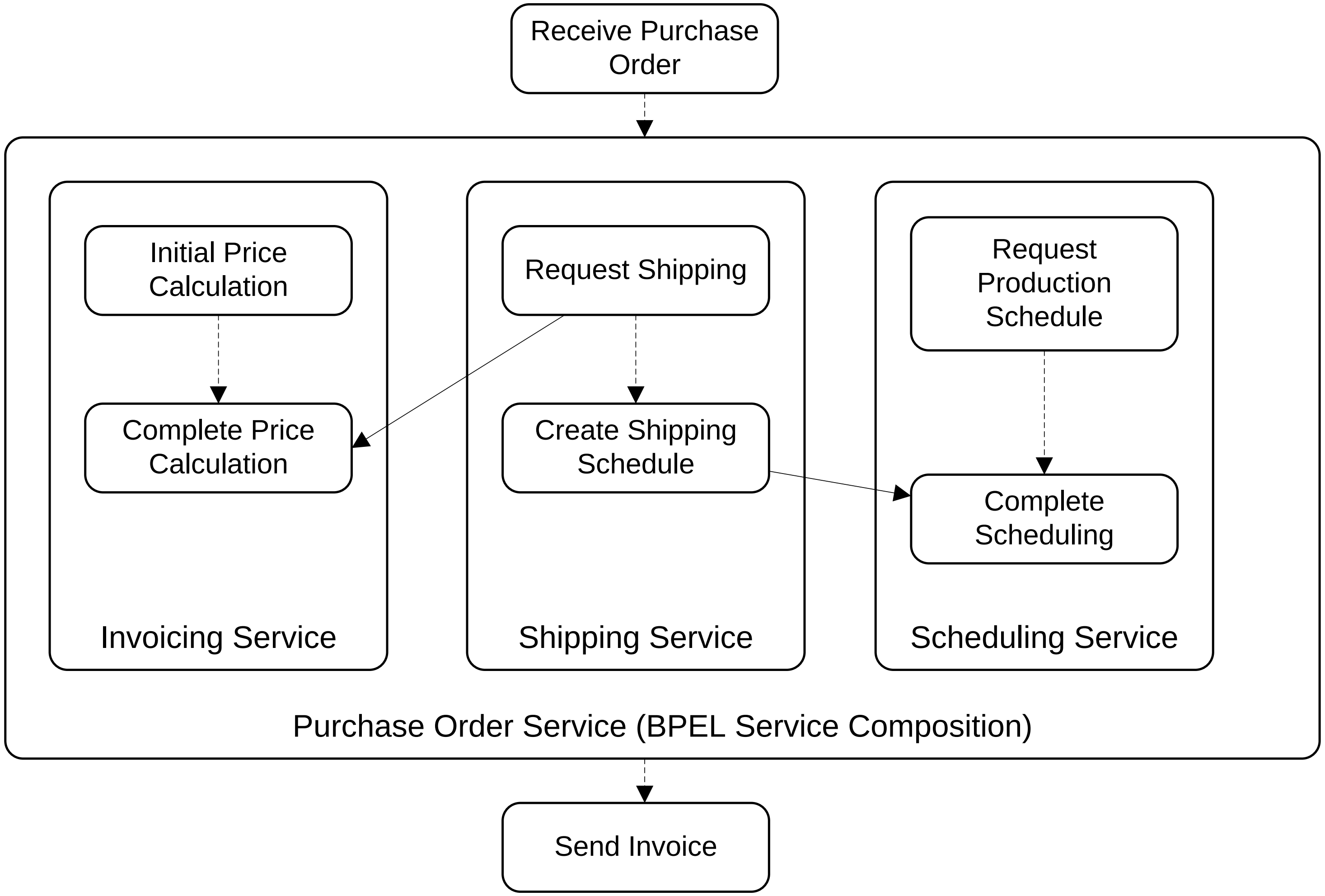


Control flow example 4

```
<process>
    <sequence>
        <receive createInstance=„yes“ />
        <flow>
            <links><link name=„l1“ /></links>
            <sequence>
                <invoke />
                <invoke>
                    <targets><target linkName=„l1“ /></targets>
                </invoke>
            </sequence>
            <sequence>
                <invoke>
                    <sources><source linkName=„l1“ /></sources>
                </invoke>
                <invoke />
            </sequence>
        </flow>
        <reply />
    </sequence>
</process>
```

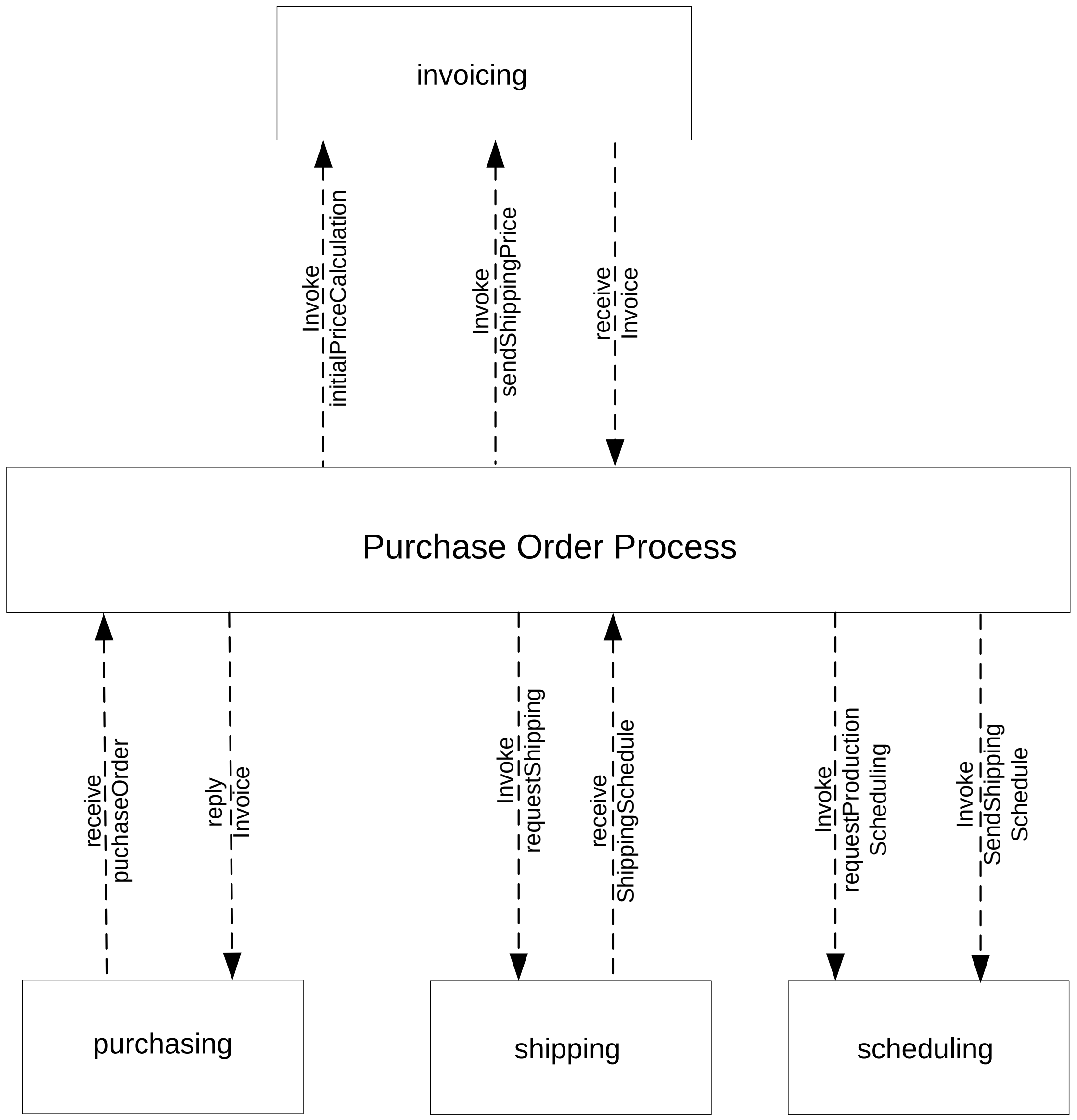


Example: Service Composition

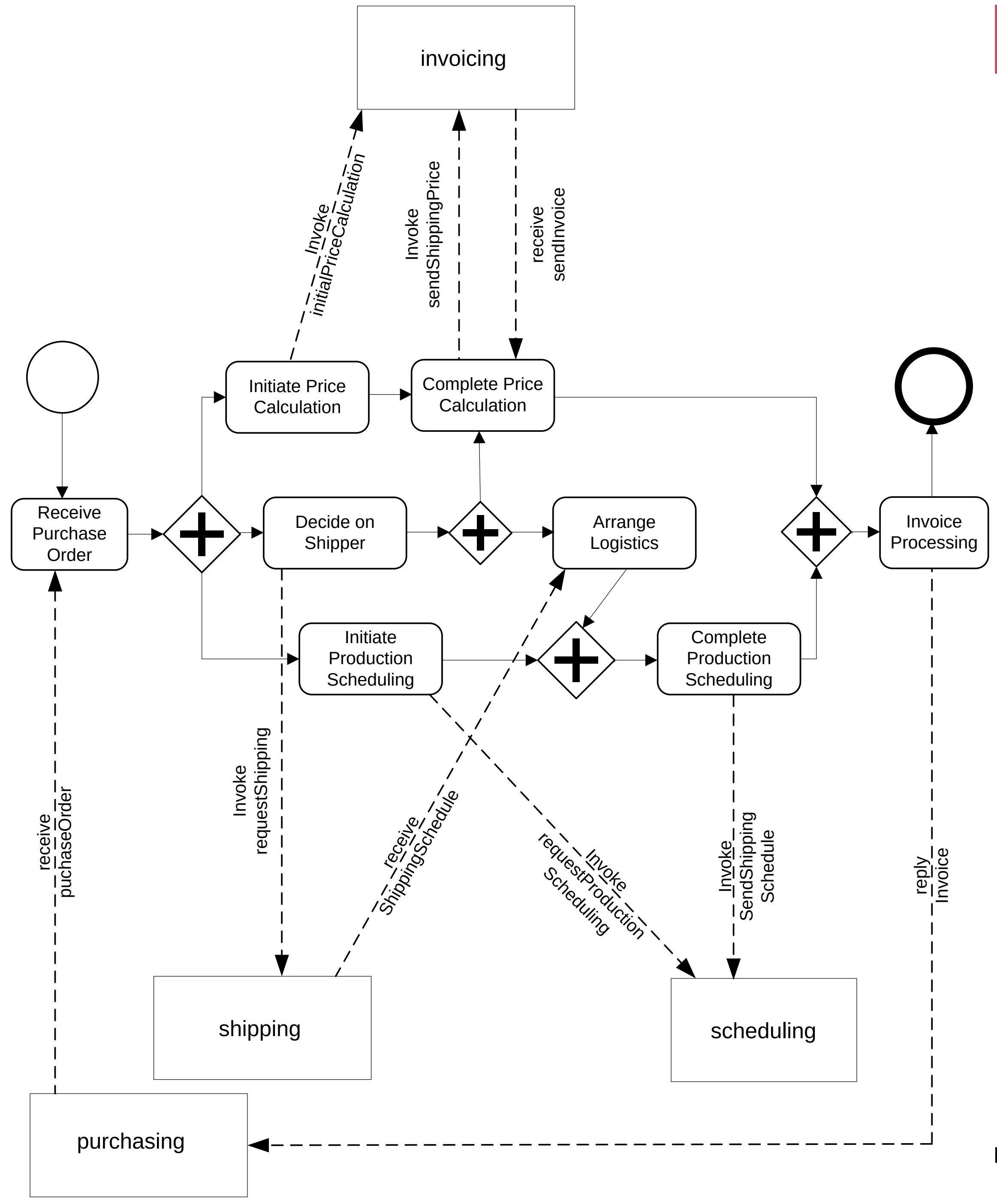


from M. Weske: Business Process Management, © Springer-Verlag Berlin Heidelberg 2007

Communication Behavior



Overall picture service composition



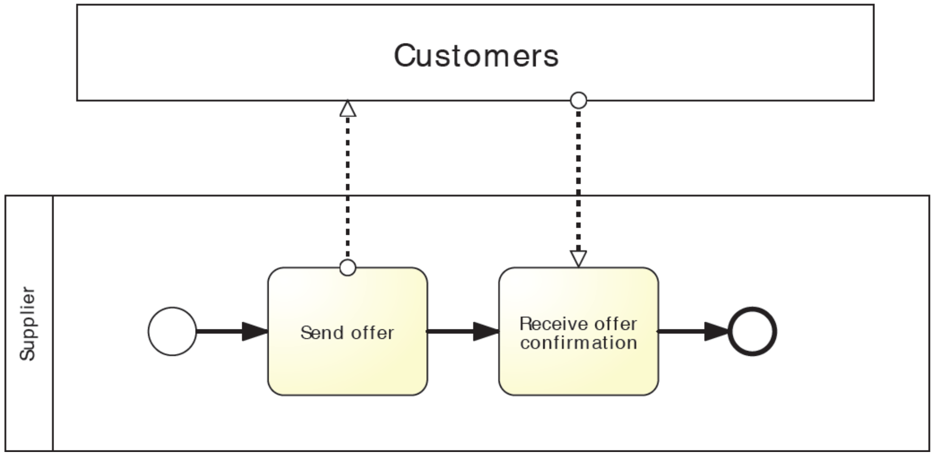
WSBPEL-representation (Simplified)

```
sequence
receive PO // Receive Purchase Order activity
   (from partner link purchasing, port type purchaseOrderPT,
    operation sendPurchaseOrder)
flow
   sequence
      assign $PO.CustomerInfo to $shippingRequest.customerInfo
      invoke requestShipping(in: shippingRequest, out:shippingInfo)
      // Decide on Shipper activity
         (partner link shipping, port type shippingCallbackPT)
         source linkName ship-to-invoice
      receive shippingSchedule // Arrange Logistics activity
         (from partner link shipping, port type shippingCallbackPT,
          operation sendSchedule)
   sequence
      invoke initialPriceCalculation (in: PO)
```

```
        // Initial Price Calculation activity
           (partner link invoicing, port type computePricePT)
        invoke sendShippingPrice (in: shippingInfo)
        // Complete Price Calculation activity
           (patner link invoicing, port type computePricePT)
           target ship-to-invoice
        receive Invoice
           (from partner link invoicing, port type invoiceCallbackPT,
            operation SendInvoice)
     sequence
        invoke requestProductionScheduling (in: PO)
        // Initiate Production Scheduling
           (partner link scheduling, port type scheduling PT)
        invoke sendShippingSchedule (in: shippingSchedule)
        // Complete Production Scheduling
           (partner link scheduling, port type scheduling PT)
           target ship-to-scheduling
  reply

    Invoice // Invoice Processing Activity
       (partner link purchasing, port type purchaseOrderPT,
        operation sendPurchaseOrder)
```

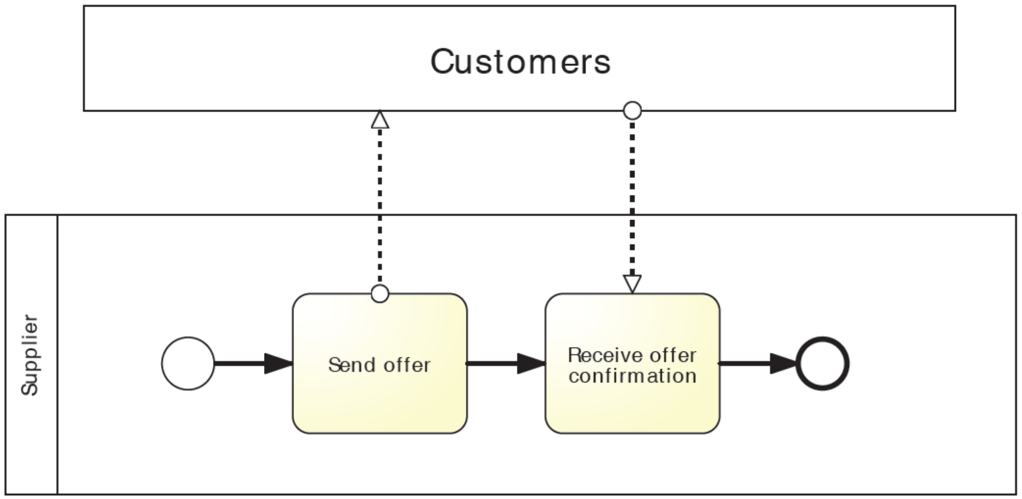
Correlation

- Idea
  - Process Engine sends out many messages of the same type and receives many messages of the same type
  - Question: How does a message finds its way to the right <receive> ?
- Example
  - Offers are sent
  - Confirmations are received
- Approach
  - Solution: Send „Order ID“ as part of the message
  - The <receive> activity registers itself for the matching Order IDs only.



Correlations

```
<correlationSets>
    <correlationSet name=“OfferCorr“
      properties="cor:Id"/>
</correlationSets>

<invoke inputVariable=“Offer">
    <correlations>
        <correlation set=“OfferCorr" initiate=“yes" />
    </correlations>
</invoke>
<receive variable=“Offer">
    <correlations>
        <correlation set=“OfferCorr" initiate=“no">
    </correlations>
</receive>
```



Variables

```
<Variables>
<variable name=“x” messageType=“namespace:tag”/>
</Variables>
```

Variables are used to contain data in BPEL. A variable can either contain an XSD value or a WSDL message. In the example above, a variable called ‘x' is declared as a container for WSDL messages of type ‘namespace:tag'. Instead of the 'messageType' attribute, the variable could have had a 'type' attribute which would specify some xsd simple or complex type like 'xsd:string' or 'xsd:integer'. Variables are used to pass data in and out of web service endpoints

Variable Assignment

```
<Assign>
    <Copy>
          <from><literal>Hello</literal></from>
          <to>$x.value</to>
    </Copy>
</Assign>
```

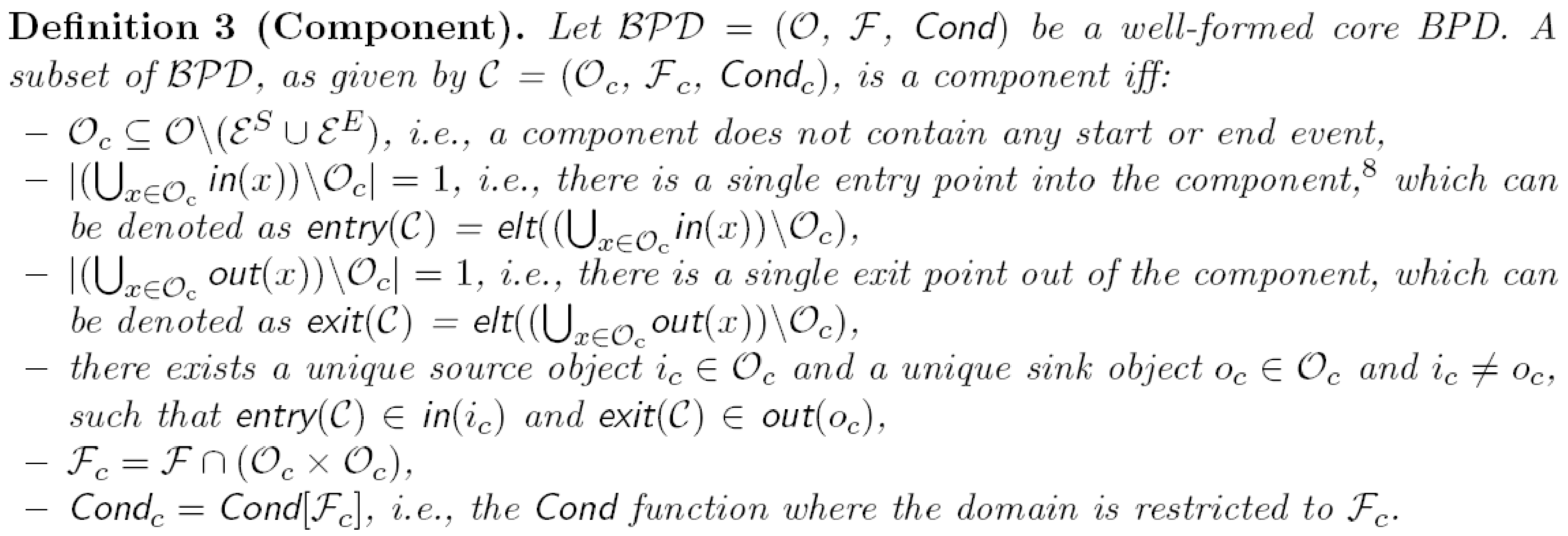
Variables are manipulated in BPEL either through use via web service endpoints or by assignment. The example above shows a literal string value being assigned into the variable ‘x'. The variable ‘x' in this case
is a WSDL message with a part called 'value'. The part called 'value' is an 'xsd:string' type. It can therefore have other 'xsd:string's assigned into it, including literal strings'

Mapping from BPMN to BPEL

- Idea
  - BPMN for Modeling of Processes
  - Added value through automatic mapping to executable languages
    - BPEL for Orchestration of Web-Services
- Remark
  - BPMN allows arbitrary graph structures, while BPEL is block structured principle (link allowed)
- Approach
  - Identifying block structured part of BPD, so-called components that can be translated directly into BPEL code
  - These parts are combined in an incremental process

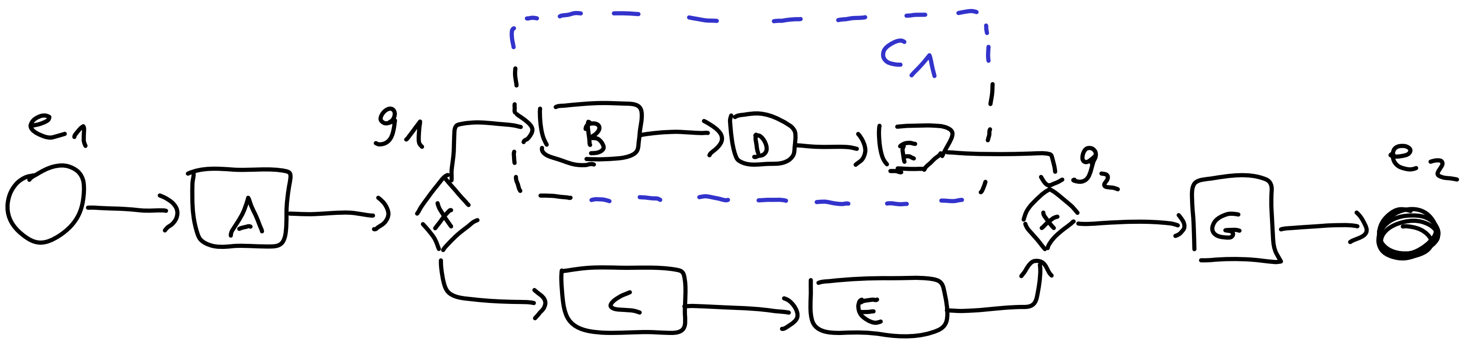
Definition *Component*

- It is based on well-formed business process diagrams BPD
- *Components* are block-structured parts of a BPD that:
  - Do not have start or end events
  - Have exactly one entry and one exit node
  - Have exactly one entry and one exit edge.



Definition 3 (Component). *Let* $\mathcal{BPD} = (\mathcal{O}, \mathcal{F}, \mathsf{Cond})$ *be a well-formed core BPD. A subset of* $\mathcal{BPD}$*, as given by* $\mathcal{C} = (\mathcal{O}_c, \mathcal{F}_c, \mathsf{Cond}_c)$*, is a component iff:*

- $\mathcal{O}_c \subseteq \mathcal{O} \backslash (\mathcal{E}^S \cup \mathcal{E}^E)$, *i.e., a component does not contain any start or end event,*
- $|(\bigcup_{x \in \mathcal{O}_c} \mathsf{in}(x)) \backslash \mathcal{O}_c| = 1$, *i.e., there is a single entry point into the component,*[8] *which can be denoted as* $\mathsf{entry}(\mathcal{C}) = \mathsf{elt}((\bigcup_{x \in \mathcal{O}_c} \mathsf{in}(x)) \backslash \mathcal{O}_c)$,
- $|(\bigcup_{x \in \mathcal{O}_c} \mathsf{out}(x)) \backslash \mathcal{O}_c| = 1$, *i.e., there is a single exit point out of the component, which can be denoted as* $\mathsf{exit}(\mathcal{C}) = \mathsf{elt}((\bigcup_{x \in \mathcal{O}_c} \mathsf{out}(x)) \backslash \mathcal{O}_c)$,
- *there exists a unique source object* $i_c \in \mathcal{O}_c$ *and a unique sink object* $o_c \in \mathcal{O}_c$ *and* $i_c \neq o_c$, *such that* $\mathsf{entry}(\mathcal{C}) \in \mathsf{in}(i_c)$ *and* $\mathsf{exit}(\mathcal{C}) \in \mathsf{out}(o_c)$,
- $\mathcal{F}_c = \mathcal{F} \cap (\mathcal{O}_c \times \mathcal{O}_c)$,
- $\mathsf{Cond}_c = \mathsf{Cond}[\mathcal{F}_c]$, *i.e., the* Cond *function where the domain is restricted to* $\mathcal{F}_c$.



$C_1 = (\sigma_1, \mathcal{F}_1, Cond_1)$

$\sigma_1 = \{B, D, F\}$

$entry(C_1) = g_1$

$exit(C_1) = g_2$

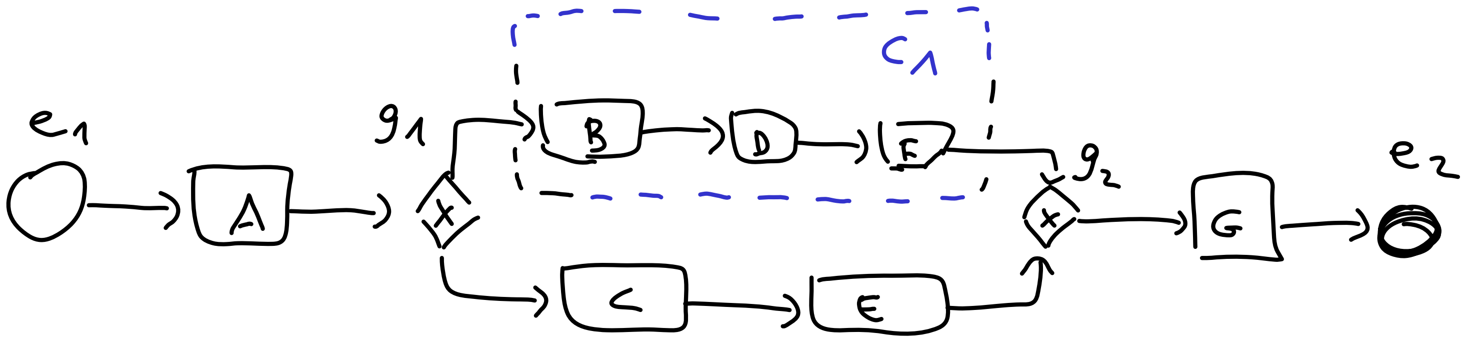
source object : B

sink object : F

Folding of the Components

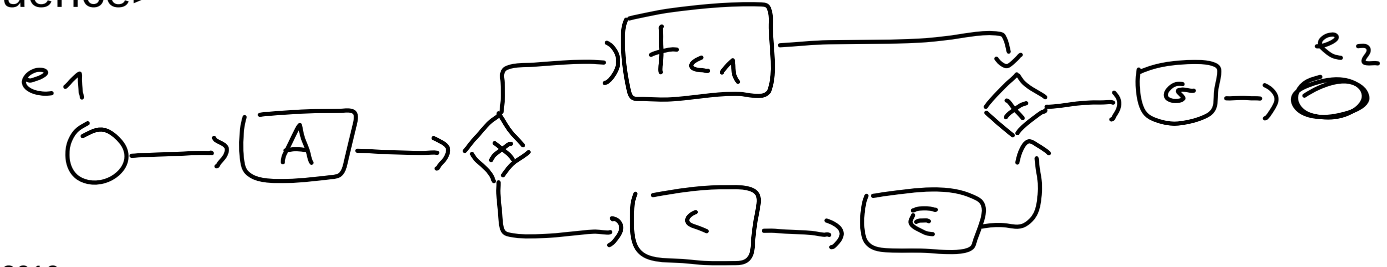
- Approach
  - Component C is replaced by a task tc, which is connected to BPEL description of C, mapping (tc) associated
  - Repeated identifying components and folding of the components into tasks results in BPEL specification, which represents the structure of BPD
- Basic mappings
  - Service Task in BPMN: <invoke>-Activity in BPEL
  - Receive Task in BPMN: <receive>-Activity in BPEL

Component Folding

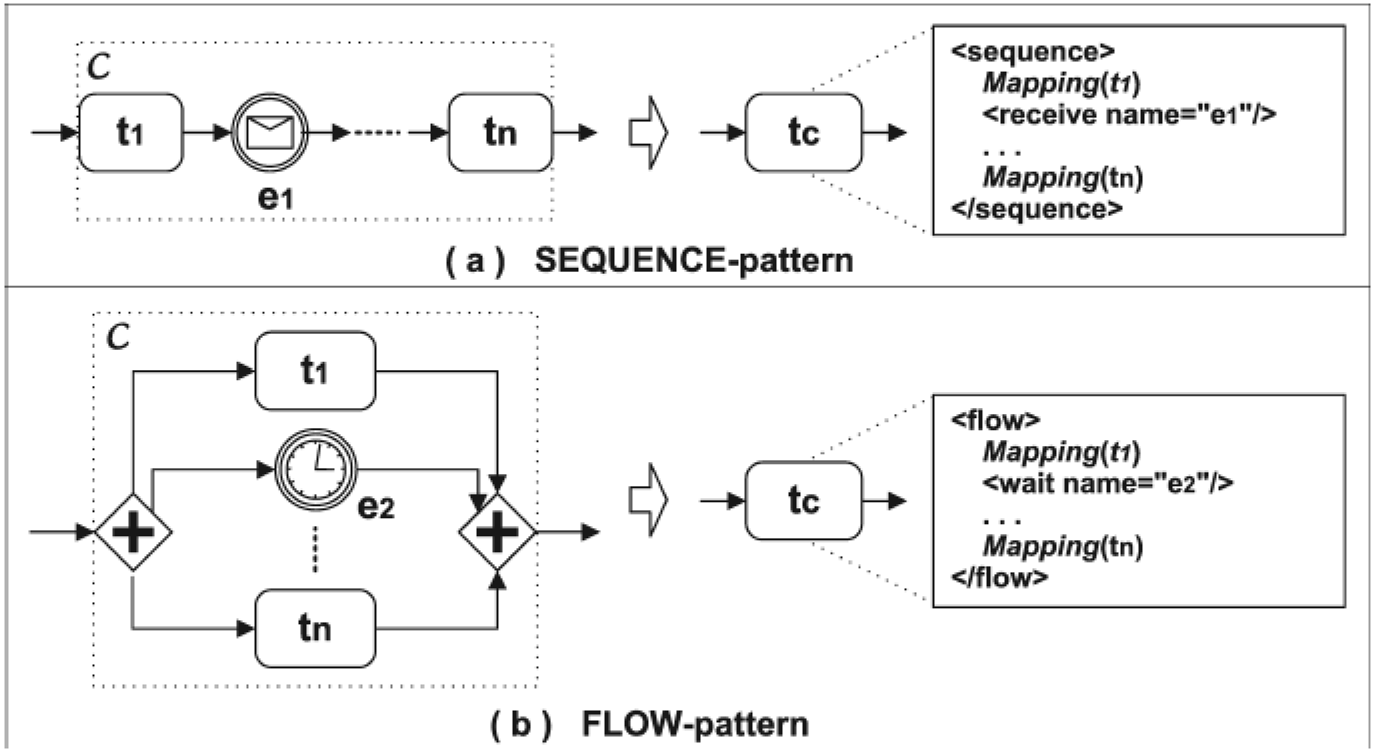


- BPEL-Representation of C1
  Mapping(t_{C1})

```
<sequence name=„tc1“>
    <invoke name=„Activity_B“ … >
    <invoke name=„Activity_D“ … >
    <invoke name=„Activity_F“ … >
</sequence>
```



Mapping Rules



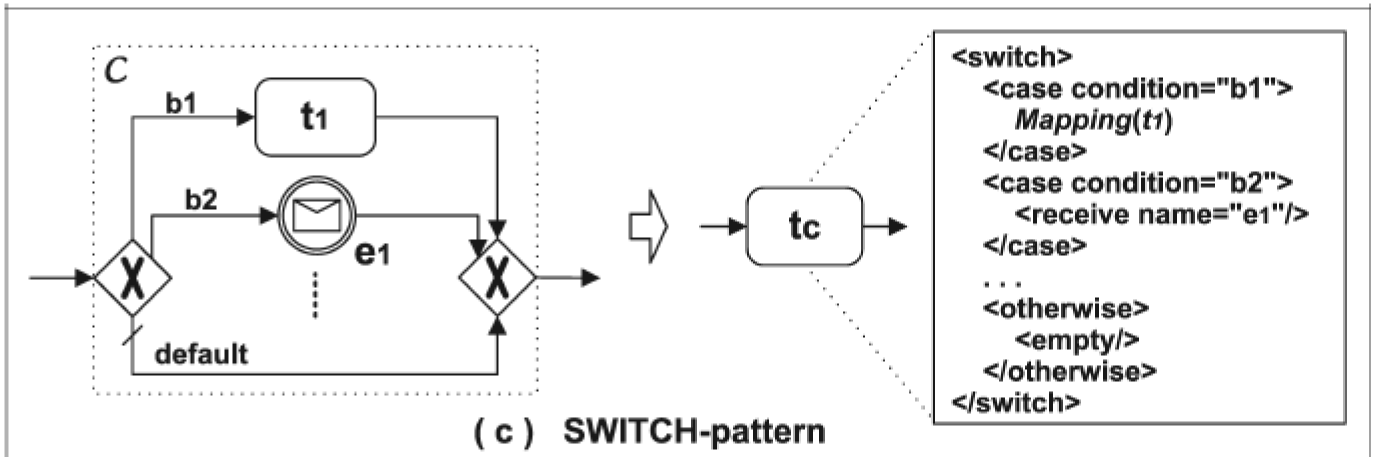
```
<sequence>
   Mapping(t1)
   <receive name="e1"/>
   . . .
   Mapping(tn)
</sequence>
```

(a) SEQUENCE-pattern

```
<flow>
   Mapping(t1)
   <wait name="e2"/>
   . . .
   Mapping(tn)
</flow>
```

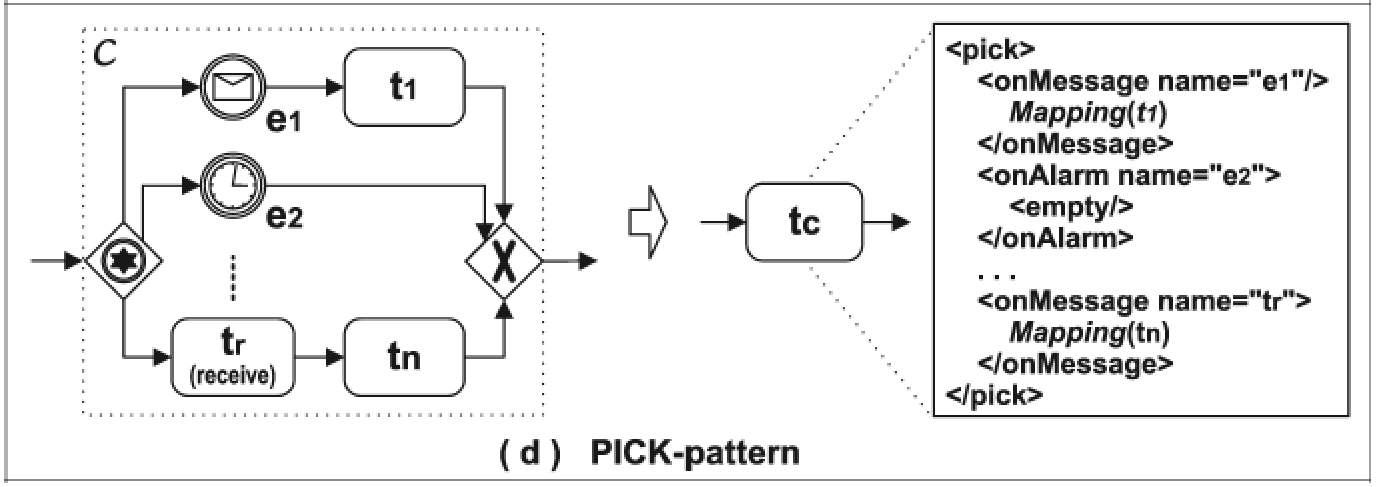
(b) FLOW-pattern

Mapping Rules



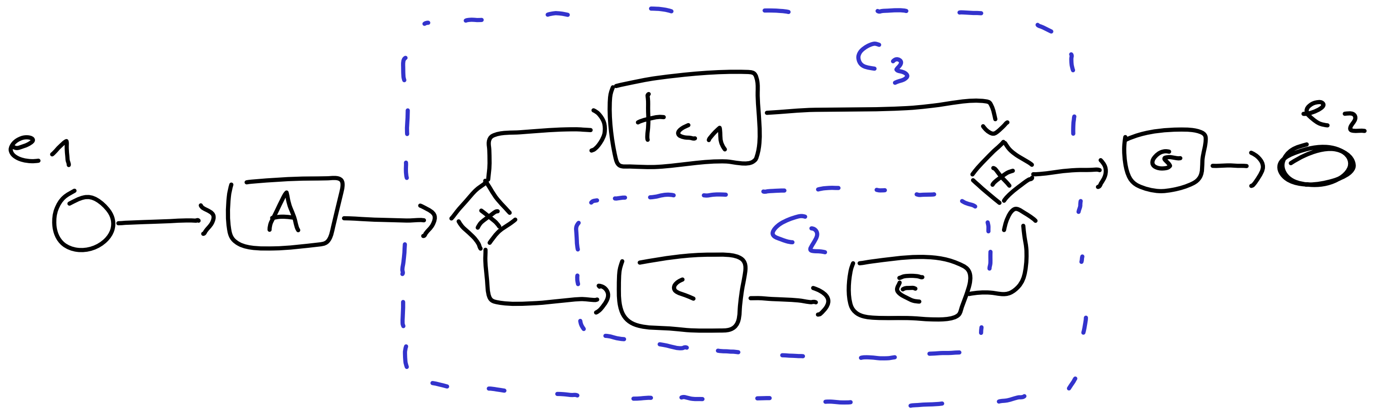
```
<switch>
   <case condition="b1">
      Mapping(t1)
   </case>
   <case condition="b2">
      <receive name="e1"/>
   </case>
   . . .
   <otherwise>
      <empty/>
   </otherwise>
</switch>
```

(c) SWITCH-pattern

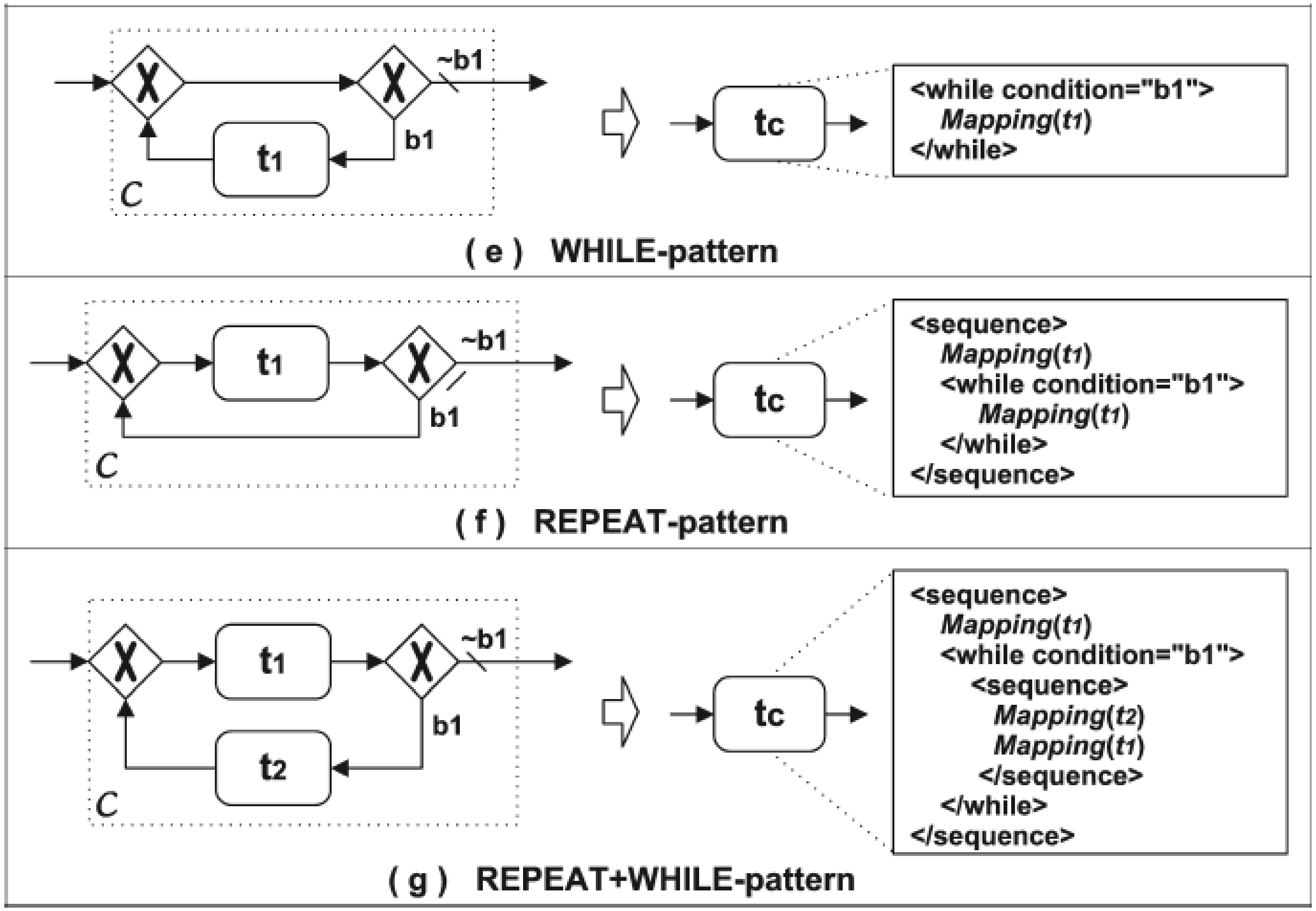


```
<pick>
   <onMessage name="e1"/>
      Mapping(t1)
   </onMessage>
   <onAlarm name="e2">
      <empty/>
   </onAlarm>
   . . .
   <onMessage name="tr">
      Mapping(tn)
   </onMessage>
</pick>
```

(d) PICK-pattern


e1
A
tc1
C3
C2
C
E
G
e2

Mapping Rules



```
<while condition="b1">
   Mapping(t1)
</while>
```

```
<sequence>
   Mapping(t1)
   <while condition="b1">
      Mapping(t1)
   </while>
</sequence>
```

```
<sequence>
   Mapping(t1)
   <while condition="b1">
      <sequence>
         Mapping(t2)
         Mapping(t1)
      </sequence>
   </while>
</sequence>
```

Example

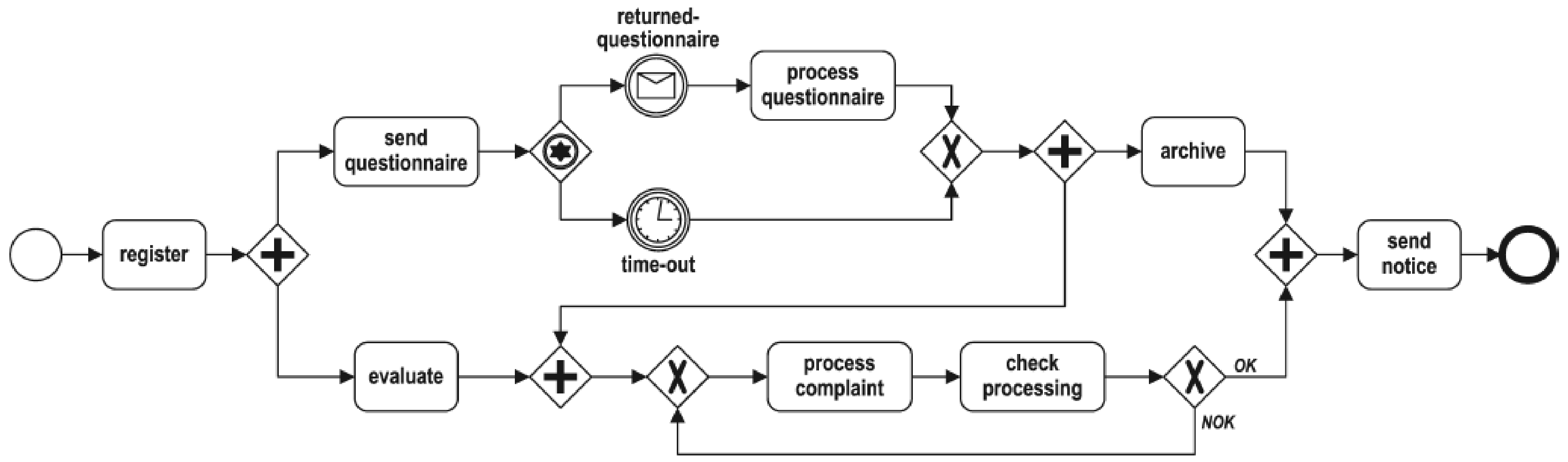
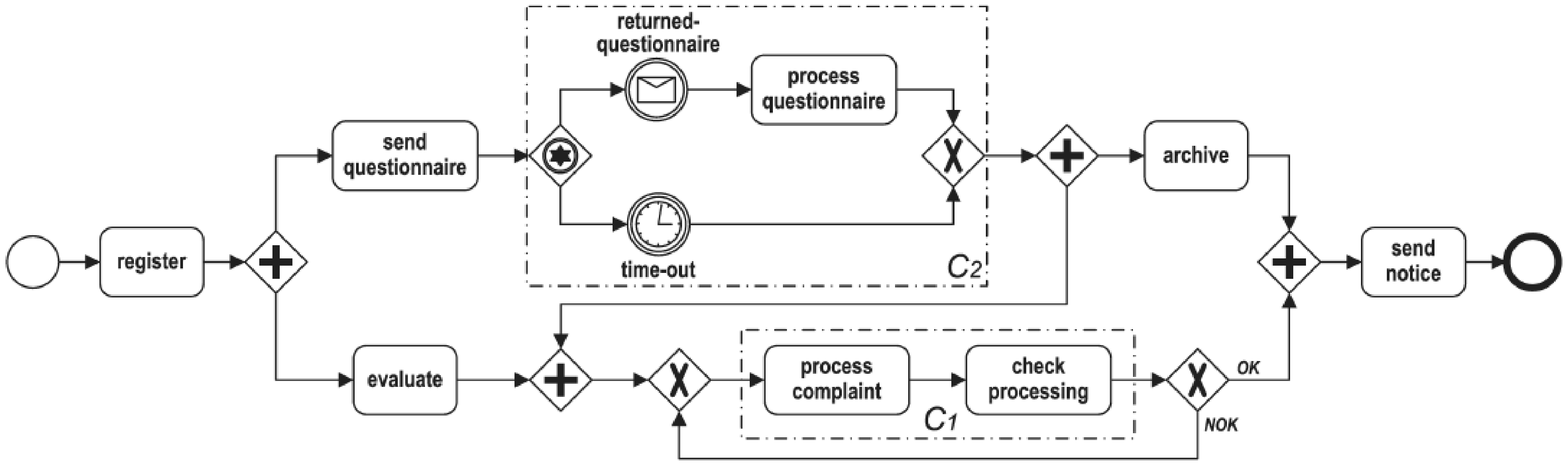


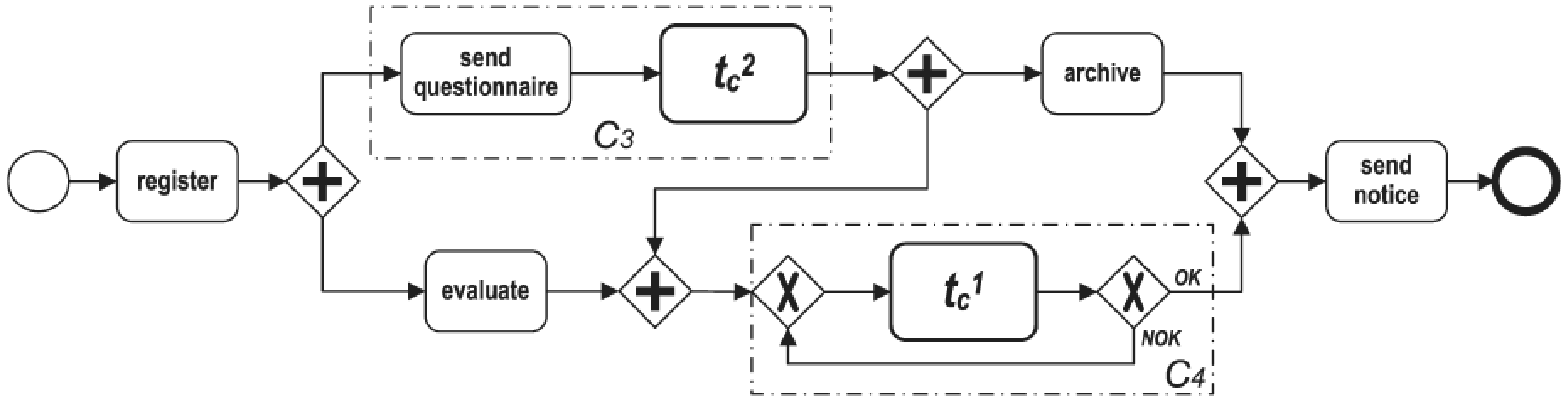
Figure 6. A complaint handling process model.

- Scenario
  - Query processing, with
    - AND, XOR Split/Join
    - Deferred Choice
    - Intermediate Events


returned-questionnaire
process questionnaire
send questionnaire
archive
register
time-out
C_2
send notice
evaluate
process complaint
check processing
C_1
OK
NOK

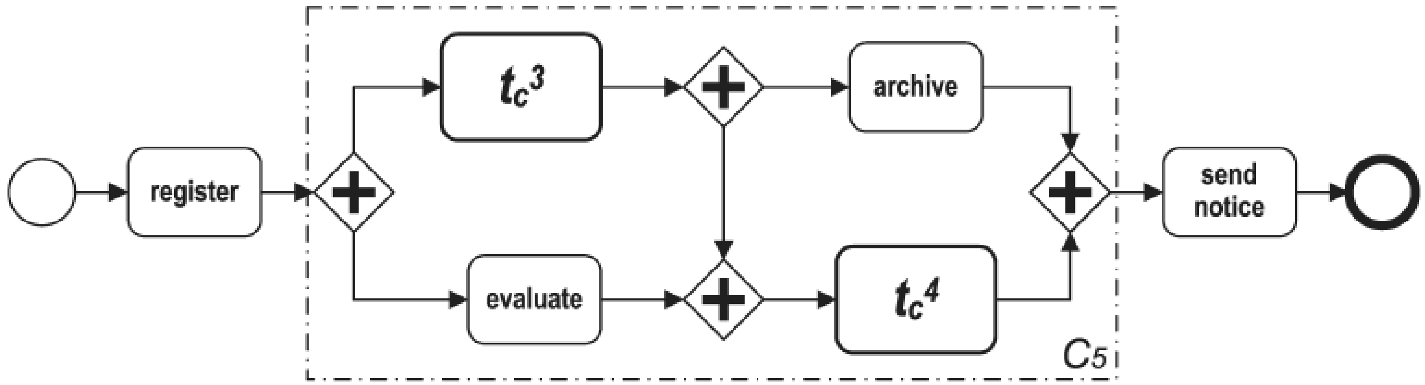
```
<pick name="$t_c^2$">
   <onMessage name="returned-questionnaire">
      <invoke name="process questionnaire"/>
   </onMessage>
   <onAlarm name="time-out">
      <empty/>
   </onAlarm>
</pick>
```

```
<sequence name="$t_c^1$">
   <invoke name="process complaint"/>
   <invoke name="check processing"/>
</sequence>
```

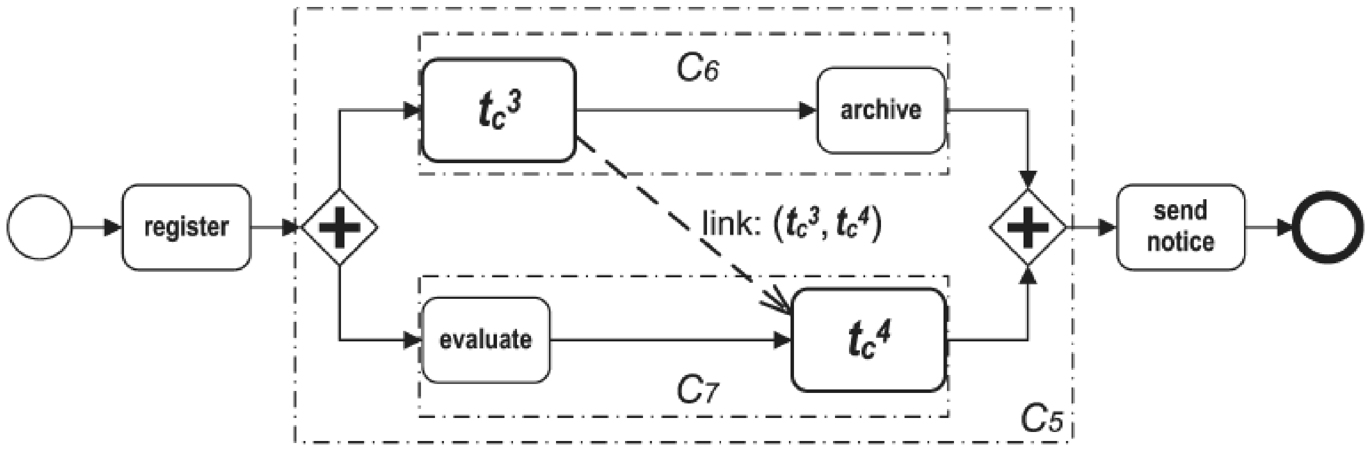

send questionnaire
t_c^2
C_3
archive
register
send notice
evaluate
t_c^1
OK
NOK
C_4

```
<sequence name="$t_c^3$">

  <invoke name="send questionnaire"/>
  <pick name="$t_c^2$">
    <onMessage name="returned-questionnaire">
      <invoke name="process questionnaire"/>
    </onMessage>
    <onAlarm name="time-out">
      <empty/>
    </onAlarm>
  </pick>
</sequence>
```

```
<sequence name="$t_c^4$">

  <sequence name="$t_c^1$">
    <invoke name="process complaint"/>
    <invoke name="check processing"/>
  </sequence>
  <while condition="NOK">
    <sequence name="$t_c^1$">
      <invoke name="process complaint"/>
      <invoke name="check processing"/>
    </sequence>
  </while>
</sequence>
```

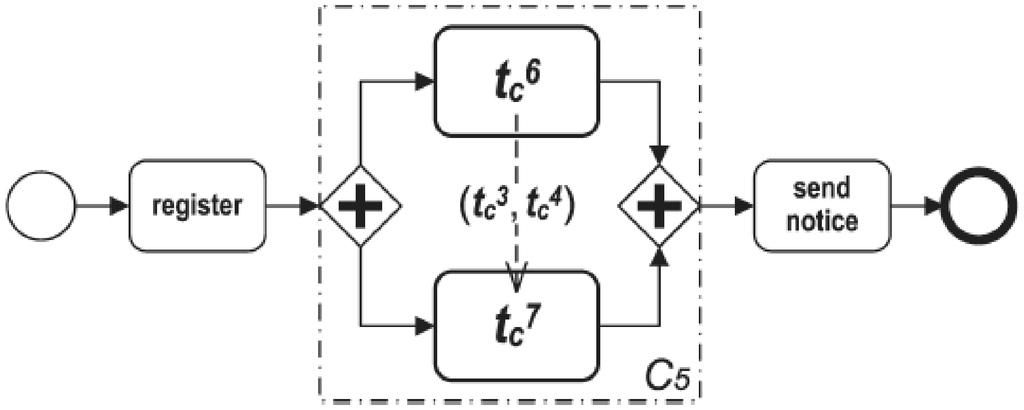


- Problem
  - AND synchronization between concurrent activities
  - Component C4 can begin only when evaluate and C3 complete
- Solution
  - Link between these components


register
t_c^3
C_6
archive
link: (t_c^3, t_c^4)
evaluate
t_c^4
C_7
C_5
send notice

```
<sequence name="$t_c^3$">
  <source linkName="t3TOt4"/>
  <invoke name="send questionnaire"/>
 <pick name="$t_c^2$">
   <onMessage name="returned-questionnaire">
     <invoke name="process questionnaire"/>
   </onMessage>
   <onAlarm name="time-out">
     <empty/>
   </onAlarm>
 </pick>
</sequence>
```

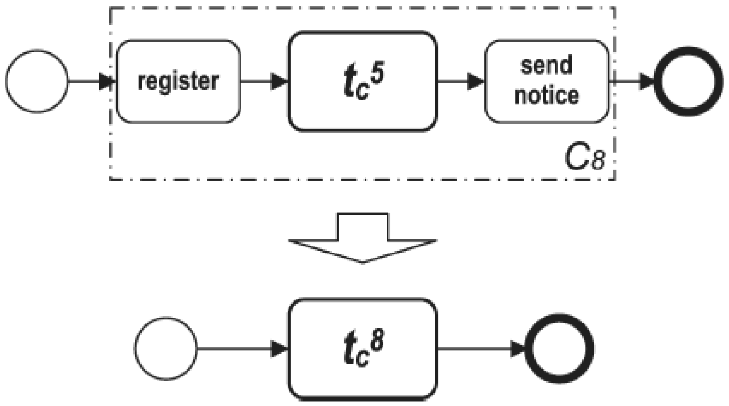
```
<sequence name="$t_c^4$">
  <target linkName="t3TOt4"/>
  <sequence name="$t_c^1$">
    <invoke name="process complaint"/>
    <invoke name="check processing"/>
  </sequence>
  <while condition="NOK">
    <sequence name="$t_c^1$">
      <invoke name="process complaint"/>
      <invoke name="check processing"/>
    </sequence>
  </while>
</sequence>
```


register
t_c^6
(t_c^3, t_c^4)
t_c^7
send notice
C_5

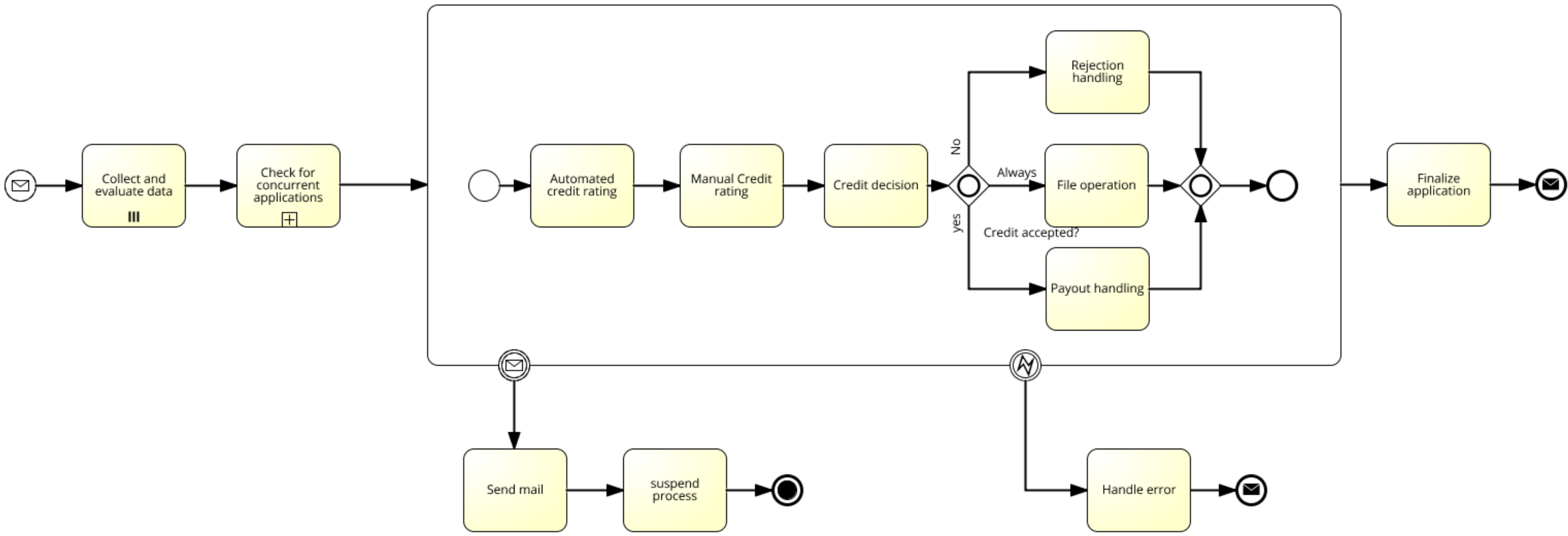
```
<sequence name="$t_c^6$">
  <sequence name="$t_c^3$">
    <source linkName="t3TOt4"/>
    <invoke name="send questionnaire"/>
    <pick name="$t_c^2$">
      <onMessage name="returned-questionnaire">
        <invoke name="process questionnaire"/>
      </onMessage>
      <onAlarm name="time-out">
        <empty/>
      </onAlarm>
    </pick>
  </sequence>
  <invoke name="archive"/>
</sequence>
```

```
<sequence name="$t_c^7$">
  <invoke name="evaluate"/>
  <sequence name="$t_c^4$">
    <target linkName="t3TOt4"/>
    <sequence name="$t_c^1$">
      <invoke name="process complaint"/>
      <invoke name="check processing"/>
    </sequence>
    <while condition="NOK">
      <sequence name="$t_c^1$">
        <invoke name="process complaint"/>
        <invoke name="check processing"/>
      </sequence>
    </while>
  </sequence>
</sequence>
```

```
<process>
  <links>
    <link name="t3TOt4"/>
  </links>
  <sequence name="$t_c^8$">
    <invoke name="register"/>
    <flow name="$t_c^5$">
      <sequence name="$t_c^6$">
        <sequence name="$t_c^3$">
          <source linkName="t3TOt4"/>
          <invoke name="send questionnaire"/>
          <pick name="$t_c^2$">
            <onMessage name="returned-questionnaire">
              <invoke name="process questionnaire"/>
            </onMessage>
            <onAlarm name="time-out">
              <empty/>
            </onAlarm>
          </pick>
        </sequence>
        <invoke name="archive"/>
      </sequence>
      <sequence name="$t_c^7$">
        <invoke name="evaluate"/>
        <sequence name="$t_c^4$">
          <target linkName="t3TOt4"/>
          <sequence name="$t_c^1$">
            <invoke name="process complaint"/>
            <invoke name="check processing"/>
          </sequence>
          <while condition="NOK">
            <sequence name="$t_c^1$">
              <invoke name="process complaint"/>
              <invoke name="check processing"/>
            </sequence>
          </while>
        </sequence>
      </sequence>
    </flow>
    <invoke name="send notice"/>
  </sequence>
</process>
```


register
t_c^5
send notice
C_8
t_c^8

Map the following Process to BPEL



```
<process>
<sequence>
<receive createinstance="yes">
</receive>
<flow>
<invoke name="Collect and evaluate data"/>
<invoke name="Collect and evaluate data"/>
<invoke name="Collect and evaluate data"/>
</flow>
<invoke name="Check for concurrent application"/>
```

```
<scope>
<faultHandler>
<catch>
<sequence>
<invoke name="Handle error"/>
<reply/>
</seqeunce>
</catch>
</faultHandler>
```

```
<eventHandler>
<onMessage>
<seqeunce>
<invoke name="send mail"/>
<invoke name="suspend process"/>
<terminate/>
</seqeunce>
</onMessage>
</eventHandler>
```

```
<sequence>
<invoke name="automated credit rating"/>
<invoke name="Manual credit rating"/>
<invoke name="credit decision"/>
<flow>
<invoke name="file operation"/>
<if>
<condition>decision=yes</condition>
<invoke name="Payout handling"/>
```

```
<elseif>
<condition>decision=no</condition>
<invoke name="Rejection handling"/>
</elseif>
</if>
</flow>
</seqeunce>
</scope>
<invoke name="Finalize application"/>
<reply/></sequence></process>
```