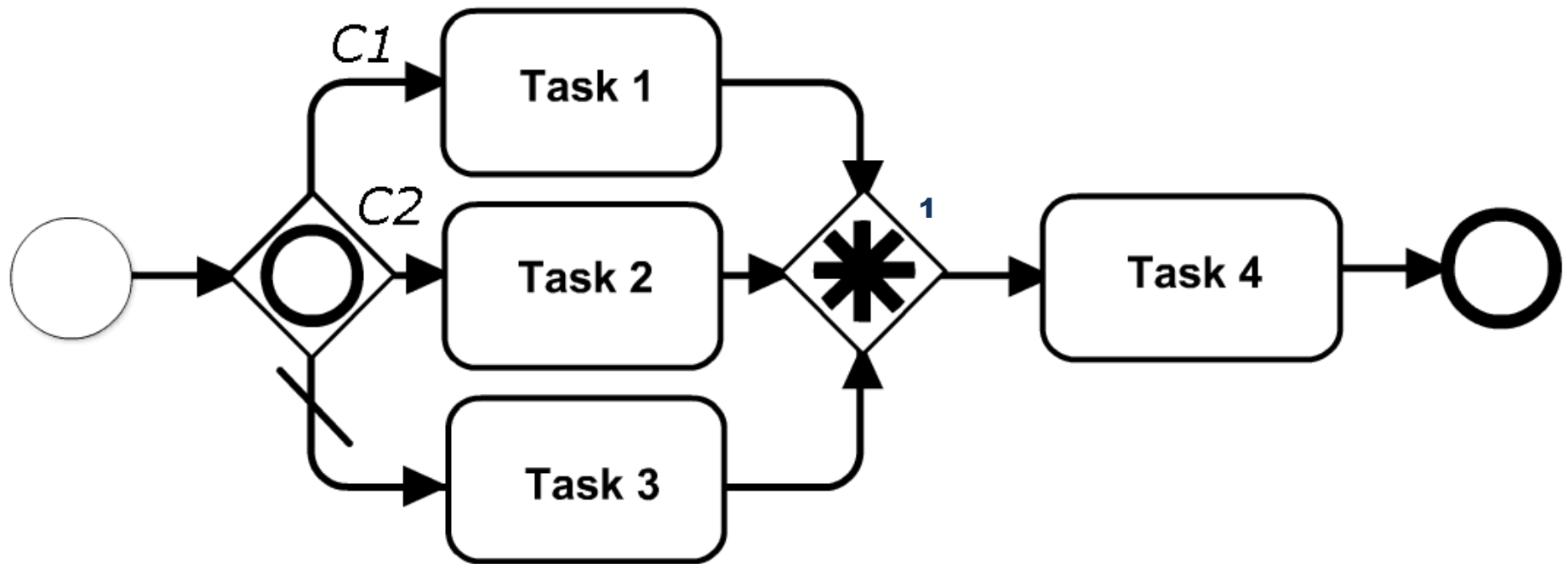


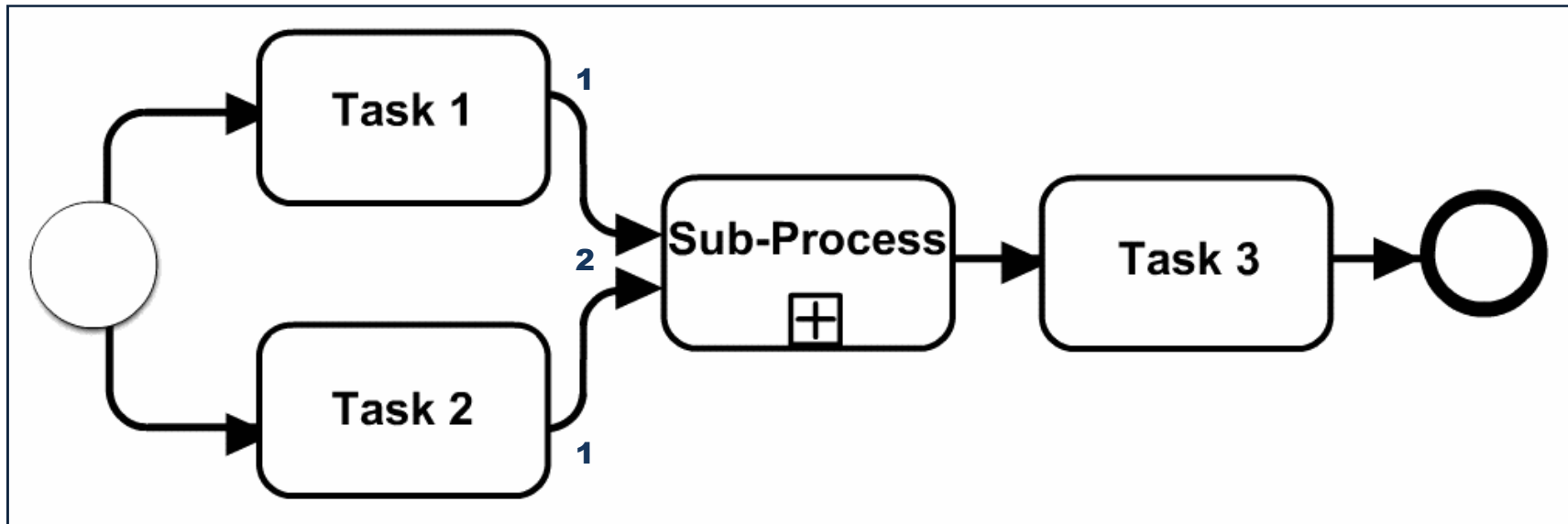
- The *Complex Gateway* was created to address complex cases which would require the combination of several other gateways. To avoid this, the behavior of the complex gateway can be scripted using an expression language.



<http://www.iet.unipi.it/m.cimino/bpm/res/mov11.swf>

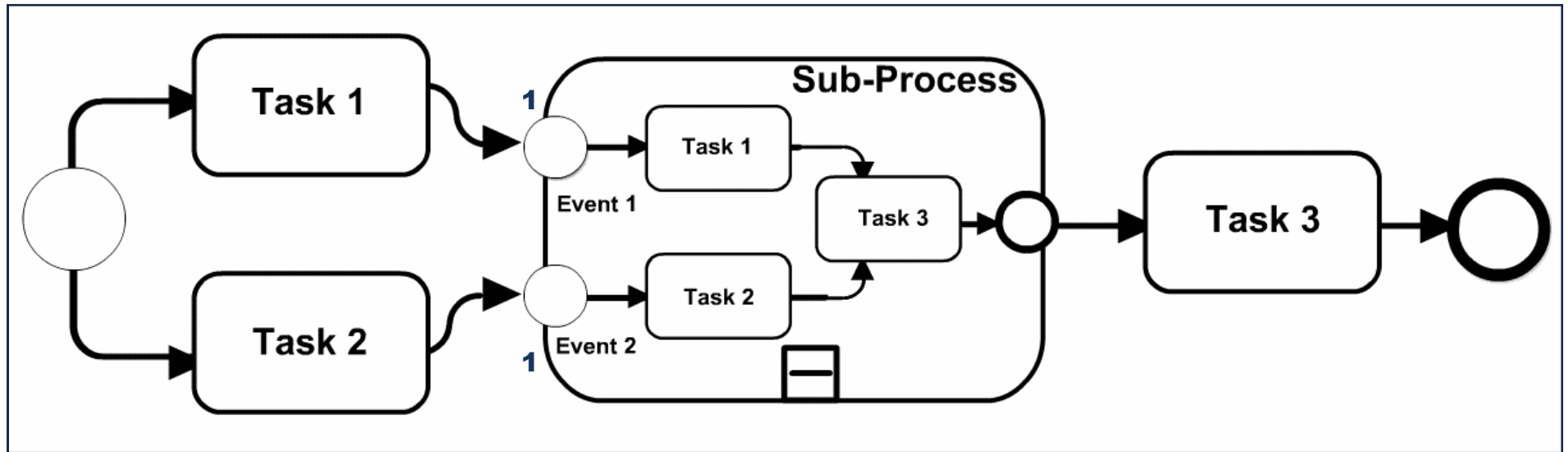
- As a result the complex gateway can be used to handle every situation. Yet, best practice is to avoid it since it makes the process models less readable.

- Merging alternative paths<sup>1</sup> can also be modeled without gateways: the alternative sequence flows directly go into the next activity<sup>2</sup> (*uncontrolled flow*).
- Each single incoming token is directly processed. The activity will not wait for any other tokens, and the tokens will not be merged.
- This behavior is appropriate for merging exclusive sequence flows where only one token can arrive.



- Two instances of the sub-process will be created because two execution points will reach it.

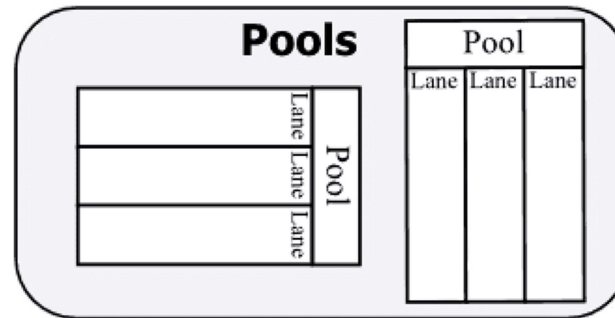
- Notice how the sub-process can be expanded and how the two incoming flows can point to different start events.



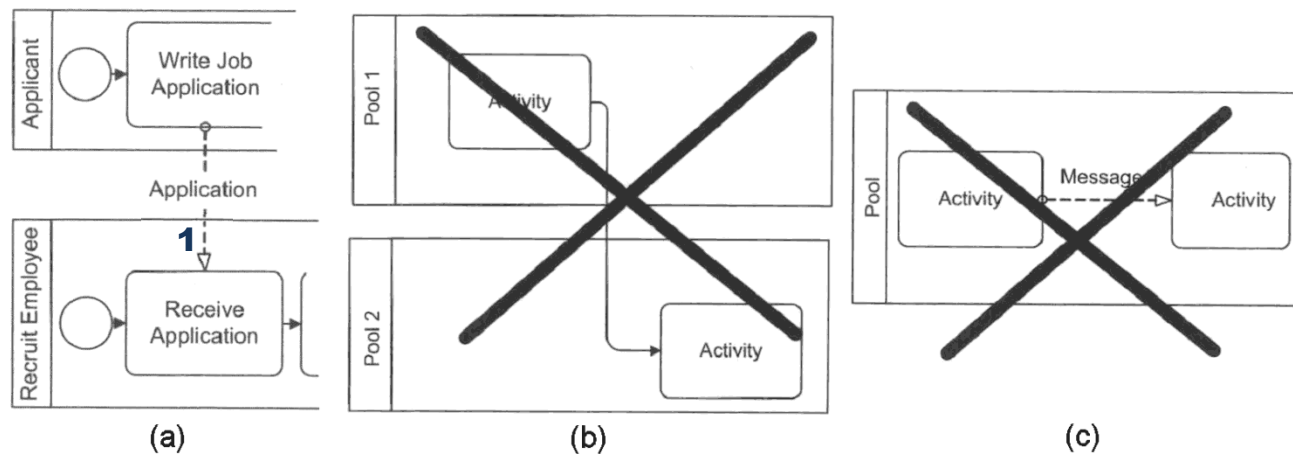
<http://www.iet.unipi.it/m.cimino/gpa/res/movie/mov12.swf>

- Modeling with (+) or without (-) gateways?
  - + it is not possible to model entirely without gateways
  - + gateways are more expressive and, sometimes, necessary
  - if gateways are omitted as far as possible, more compact models can be created
  - gateways with several inputs and several outputs may cause misunderstanding

- *Swimlanes* are used to organize activities and depict collaboration between partners. A business process is then organized inside a *pool* of swimlanes (business units).

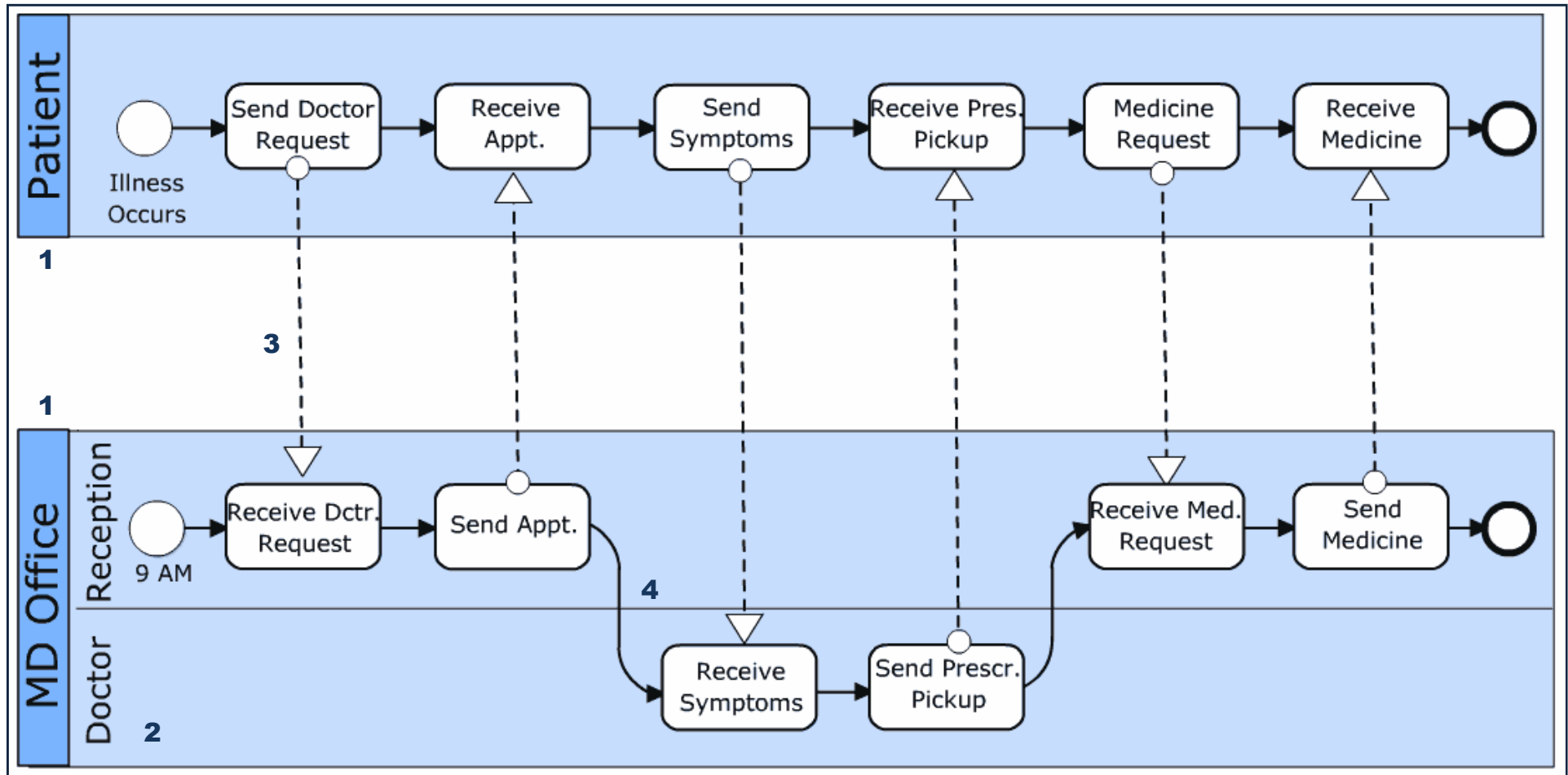


- *Private* business processes are those internal to a specific organization, i.e. contained in a pool, which does not interact with external pools (*orchestration*). A collaboration is a synchronized interaction of more processes without a central control (*choreography*), via *public* processes and message flows<sup>1</sup> (white dashed arrow) between pools (a).



- Control flow between pools (b) and message flow within a pool (c) are **forbidden**

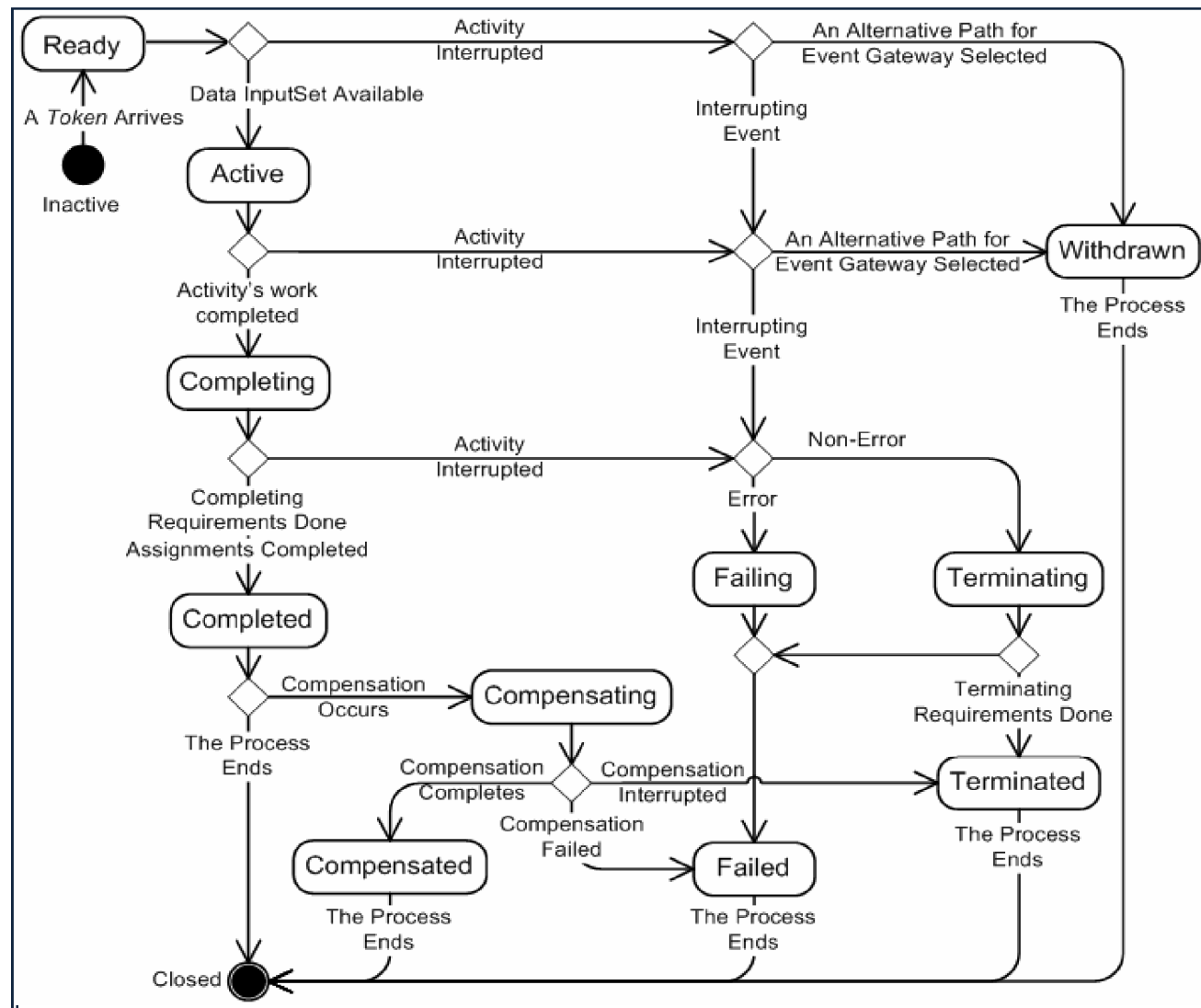
- An example of collaboration between a patient and medical office. The processes within pools are abstracted to hide complexity or private information to partners. Note: pool (1), lane (2), message flow (3) and control flow (4).



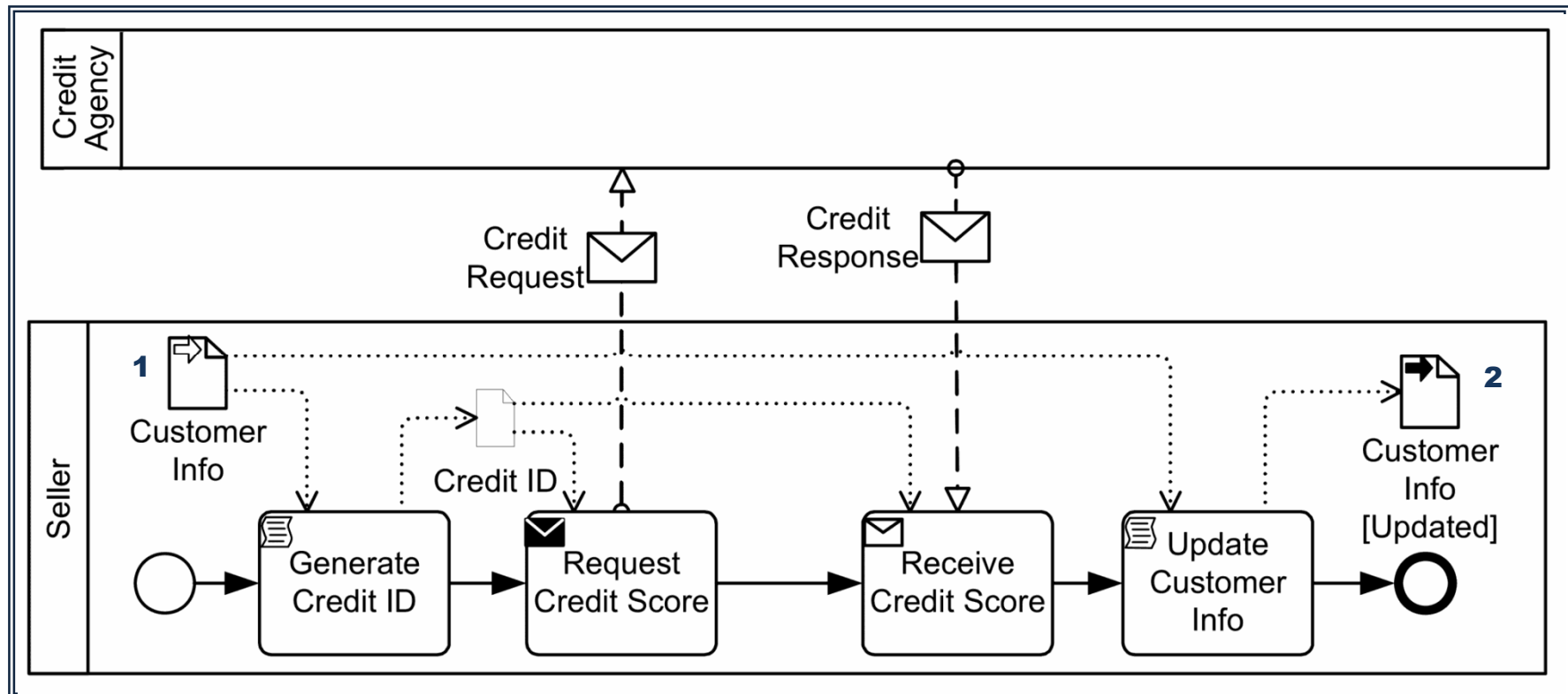
<http://www.iet.unipi.it/m.cimino/bpm/res/mov13.swf>

- Lifecycle of a BPMN 2 activity/process:
  - An activity is *ready* for execution if the required number of tokens is available to activate the activity.
  - When some data input becomes available, the activity changes from *ready* to *active* state.
  - An activity, if *ready* or *active*, can be *withdrawn* from being able to complete, for tasks that are attached after an event-based exclusive gateway (race condition): the first completed task causes all other tasks to be withdrawn.
  - If an activity fails during execution, it changes from the state *active* to *failing*. If an activity execution is interrupted, e.g. by an interrupting event or an error, its state changes to *terminating* or *failing*, respectively.
  - After all terminating requirements of a terminating activity have been fulfilled, the state of the activity changes to *terminated*. Otherwise it changes to *failing* and then to *failed*.
  - If an activity execution ends without anomalies, its state changes to *completing*. After all completion dependencies of a *completing* activity have been fulfilled, the state of the activity changes to *completed*. The outgoing sequence flows becomes active and a certain number of tokens is placed on it. Upon completion, also data output are available.

- Only completed activities can change their state to *compensating* until either compensation finishes successfully (state *compensated*), an exception occurs (state *failed*), or termination is triggered (state *terminated*).
- An UML state diagram



- BPMN allows the explicit modeling of data transfer, via *data objects*, *messages* and *data store*. Data objects only exist within a process, whereas data store represent persistent data. UML class diagrams or technical terms diagrams could be used to refine data objects in process models → process view is integrated with data view.
- A directed data association is drawn as a dotted line to model which activity outputs and takes as input a certain data object. Data input<sup>1</sup> and data output<sup>2</sup> can represent a form of dependency between activities.

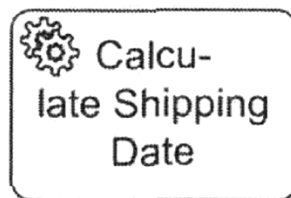


- Exercise: identify the type of the first activity with BPMN poster and specification.

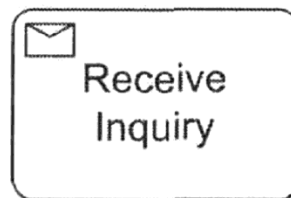


- Types of tasks and icons

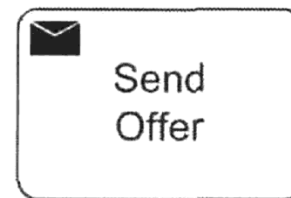
- ✓ a *service task* is an automated function;
- ✓ a *receive/send task* receives/sends a message;
- ✓ a *user task* expects input by a user via a UI;
- ✓ in a *business rule task* some business rules are applied (e.g. via a business rule management system launched by the process engine) to produce a result;
- ✓ a *script task* contains statements processed directly by the process engine;
- ✓ a *manual task* is carried out without IT support;
- ✓ in an *abstract task* no type is defined.



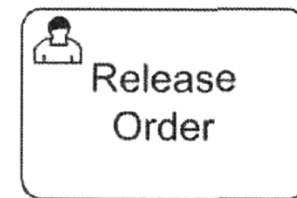
Service Task



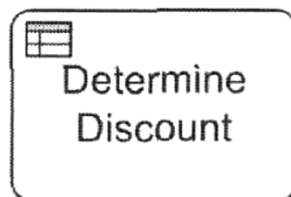
Receive Task



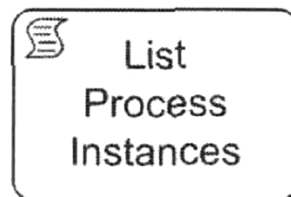
Send Task



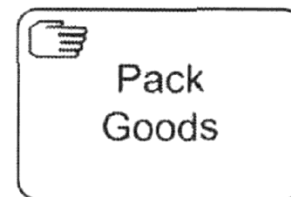
User Task



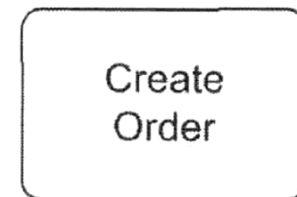
Business Rule Task



Script Task

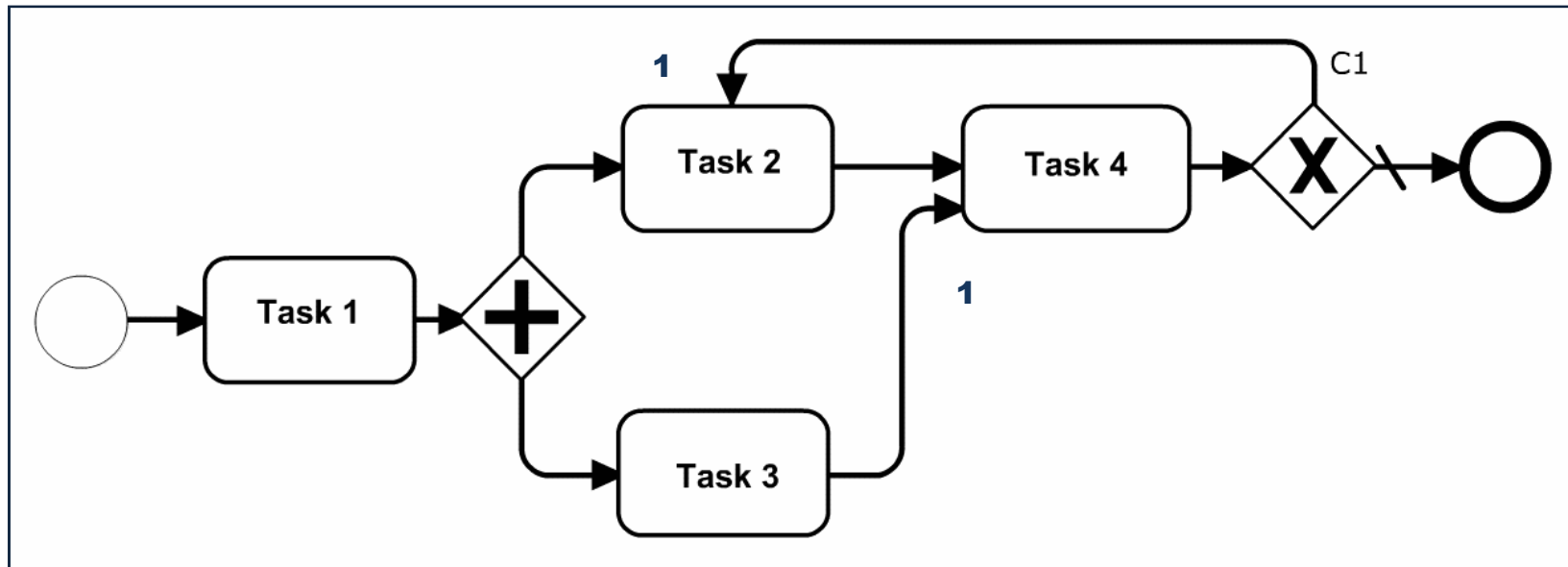


Manual Task



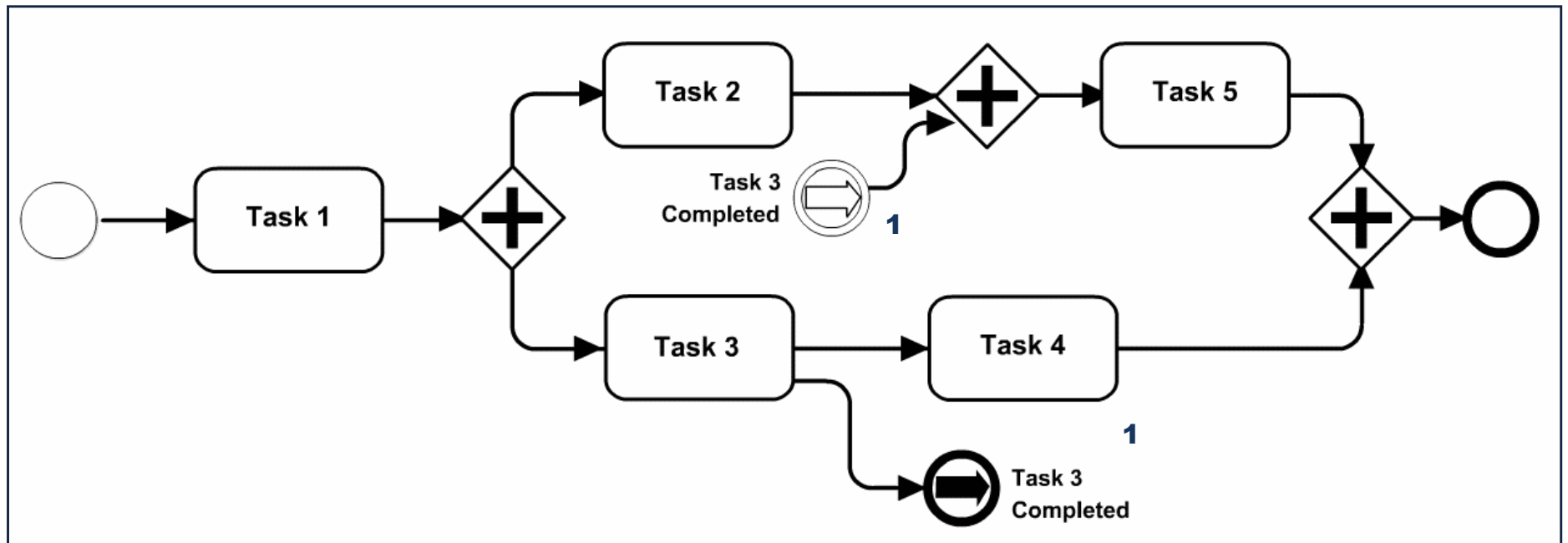
Abstract Task

- *Workflow patterns* are models of recurrent aspects of processes control flows, as identified by the Workflow Pattern Initiative ([www.workflowpatterns.com](http://www.workflowpatterns.com)).
- e.g., wcp-10: unstructured loop (known as *arbitrary cycles*): the ability to represent cycles in a process model that have more than one entry or exit point. Block structured looping operators (while, for,...) cannot implement this pattern.
- Example: two entry points<sup>1</sup>



<http://www.iet.unipi.it/m.cimino/bpm/res/mov14.swf>

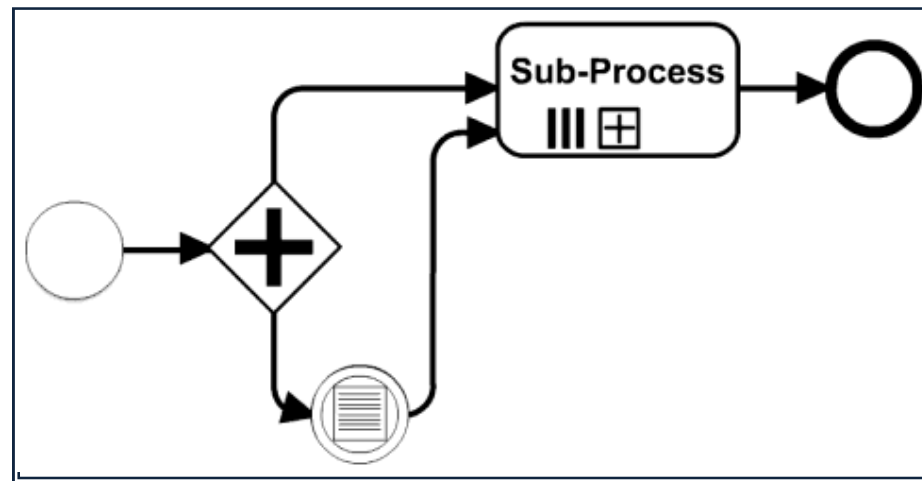
- e.g., wcp-18: *milestone*, a task is only enabled when its process is in a specific state, which corresponds to a specific execution point. When this execution point is reached the nominated task can be enabled.
- If the process instance has progressed beyond this state, then the task cannot be enabled now or at any future time (i.e. the deadline has expired).



<http://www.iet.unipi.it/m.cimino/bpm/res/mov15.swf>

- example: most budget airlines allow the routing of a booking to be changed providing the ticket has not been issued.


- e.g., wcp-15: *multiple instances without a priori run-time knowledge*, multiple instances of a task can be created. The required number of instances may depend on a number of runtime factors, and is not known until the final instance has completed. Once initiated, these instances are independent of each other and run concurrently. At any time, whilst instances are running, it is possible for additional instances to be initiated. It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered.



<http://www.iet.unipi.it/m.cimino/bpm/res/mov16.swf>

- The dispatch of an oil rig from factory to site involves numerous transport shipment tasks. These occur concurrently, and it is always possible for additional tasks to be initiated if there is a shortfall in transportation requirements. Once the whole oil rig has been transported, and all transport shipment tasks are completed, the next task (assemble rig) can commence.

- Each BPMN element has a structure made of attributes and element associations, as defined in the BPMN specification via UML classes and XML schemas.
- Example: *text annotation*, used for adding explanations, with no execution effect.



Text Annotation allows a modeler to provide additional information

**Text Annotation XML schema**

```

<xsd:element name="textAnnotation"
  type="tTextAnnotation" substitutionGroup="artifact"/>
<xsd:complexType name="tTextAnnotation">
  <xsd:complexContent>
    <xsd:extension base="tArtifact">
      <xsd:sequence>
        <xsd:element ref="text"
          minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="textFormat"
        type="xsd:string" default="textplain"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="text" type="tText"/>
<xsd:complexType name="tText" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##any"
      processContents="lax" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>


```

**Text Annotation attributes**

Attributes	Description
<b>text:</b> string	Text is an attribute that is text that the modeler wishes to communicate to the reader of the Diagram.
<b>textFormat:</b> string	This attribute identifies the format of the text. It MUST follow the mime-type format. The default is "text/plain."

- Example: *timer event*, which waits until (i) an absolute point of time, (ii) the end of a certain time span, or (iii) a time cycle for a recurring time interval.

**Timer Event**



The `TimerEventDefinition` element inherits the attributes and model associations of `BaseElement`

**TimerEventDefinition model associations**

Attribute Name	Description/Usage
<code>timeDate</code> : Expression [0..1]	If the <i>trigger</i> is a <code>Timer</code> , then a <code>timeDate</code> MAY be entered. Timer attributes are mutually exclusive and if any of the other <code>Timer</code> attributes is set, <code>timeDate</code> MUST NOT be set (if the <code>isExecutable</code> attribute of the <b>Process</b> is set to <code>true</code> ). The return type of the attribute <code>timeDate</code> MUST conform to the ISO-8601 format for date and time representations.
<code>timeCycle</code> : Expression [0..1]	If the <i>trigger</i> is a <code>Timer</code> , then a <code>timeCycle</code> MAY be entered. Timer attributes are mutually exclusive and if any of the other <code>Timer</code> attributes is set, <code>timeCycle</code> MUST NOT be set (if the <code>isExecutable</code> attribute of the <b>Process</b> is set to <code>true</code> ). The return type of the attribute <code>timeCycle</code> MUST conform to the ISO-8601 format for recurring time interval representations.
<code>timeDuration</code> : Expression [0..1]	If the <i>trigger</i> is a <code>Timer</code> , then a <code>timeDuration</code> MAY be entered. Timer attributes are mutually exclusive and if any of the other <code>Timer</code> attributes is set, <code>timeDuration</code> MUST NOT be set (if the <code>isExecutable</code> attribute of the <b>Process</b> is set to <code>true</code> ). The return type of the attribute <code>timeDuration</code> MUST conform to the ISO-8601 format for time interval representations.

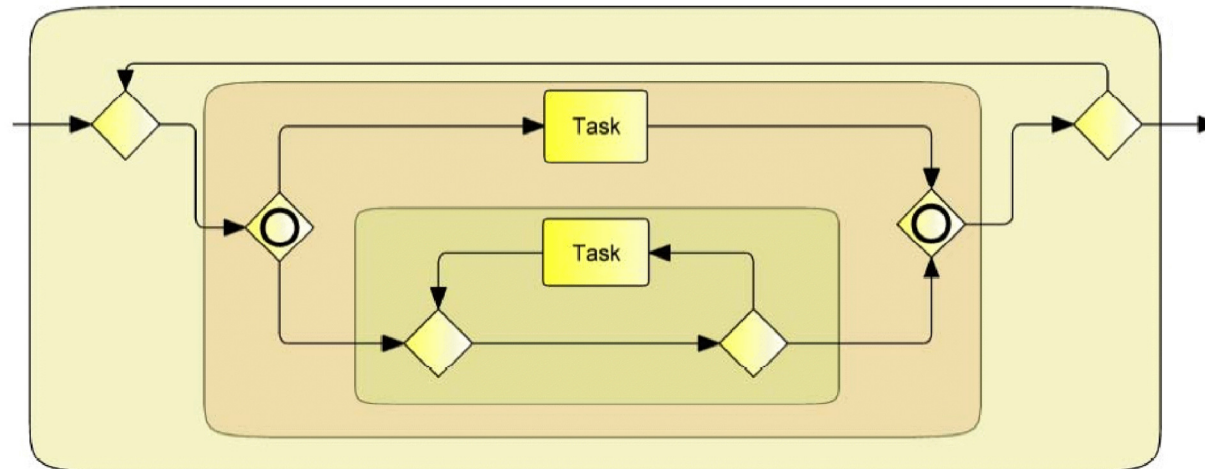
**TimerEventDefinition XML schema**

```

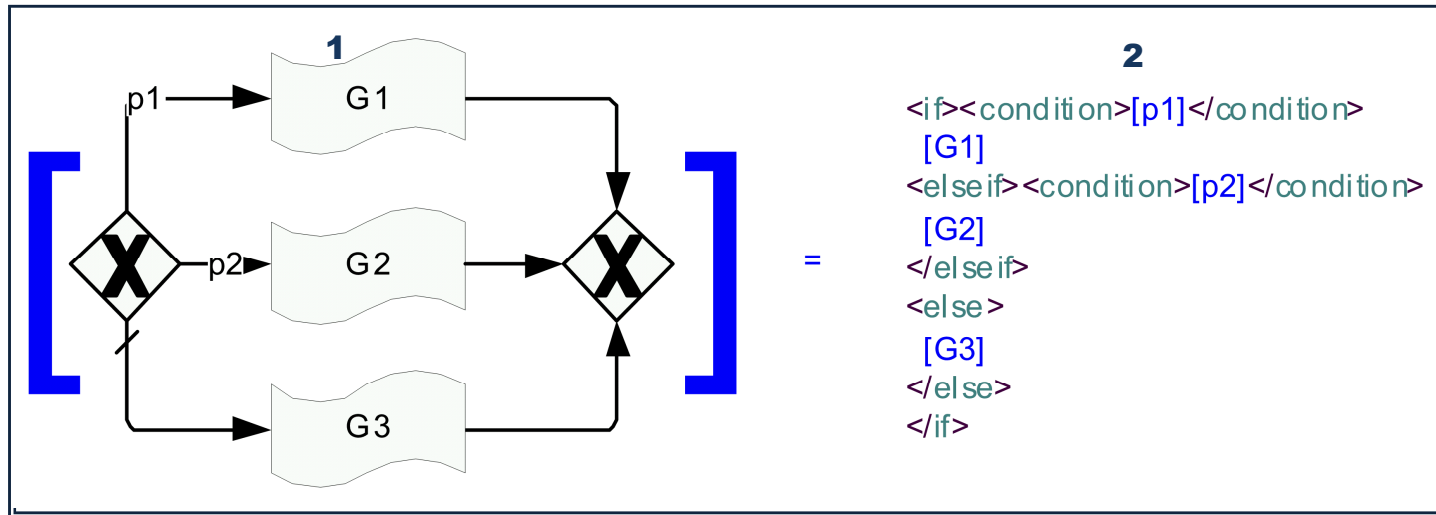
<xsd:element name="timerEventDefinition" type="tTimerEventDefinition" substitutionGroup="eventDefinition"/>
<xsd:complexType name="tTimerEventDefinition">
  <xsd:complexContent>
    <xsd:extension base="tEventDefinition">
      <xsd:choice>
        <xsd:element name="timeDate" type="tExpression" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="timeDuration" type="tExpression" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="timeCycle" type="tExpression" minOccurs="0" maxOccurs="1"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

- Roughly speaking, a BPMN orchestration process (i.e., within a Pool) may be mapped to an individual WS-BPEL process (executable on a SOA environment). A BPMN choreography can be mapped to WSDL calls (web services interfaces).
- Not all BPMN orchestration processes can be mapped to WS-BPEL, because BPMN allows the modeler to draw almost arbitrary graphs to model control flow, whereas in WS-BPEL there are certain restrictions. For example, an unstructured loop cannot directly be represented in WS-BPEL (there is no *goto* like statement, there are *while*, *for*, ... structured cycles)
- Further, to map a BPMN orchestration process to WS-BPEL it must be *sound* (i.e., no *deadlock*, no *lack of synchronization*, etc., van der Aalst 2003).
- A *block* of a BPMN diagram is a sub-diagram that is connected to the rest of the diagram only through exactly one entering sequence flow and one leaving sequence flow.



- Example of a data-based exclusive choice, controlled by two predicates p1 and p2, containing three BPMN blocks G1, G2, and G3<sup>1</sup>, mapped to WS-BPEL<sup>2</sup>. The “waved rectangle” symbol is used to denote BPMN blocks.



- Example of a mapping of a simple BPMN interface, with a scalar data item definition typed by an XML schema definition, to SW-BPEL using WSDL.

