TTC 2013 Live Challenge: BPMN 2.0 Semantics

Christian Krause

As the live challenge of the Transformation Tool Contest 2013 we present a set of tasks for modeling the execution semantics of a subset of the Business Process Model and Notation (BPMN) 2.0 standard as in-place transformations. The challenge includes proper starting and termination of processes, execution of primitive tasks based on token games and the handling of parallel gateways. In addition to the rule-based modeling of the execution semantics of BPMN, we ask the contestants of the TTC to use their specification for (i) manual and batch execution of processes and (ii) a state space generation and basic analysis of temporal properties.

1 Business Process Modeling using BPMN

This year's live contest is devoted to the Business Process Model and Notation (BPMN) 2.0 standard [4]. BPMN is a control-flow oriented modeling language for specifying business processes. The BPMN standard defines a graphical syntax for process specifications and there exist a number of tools facilitating the modeling and the execution of BPMN processes.

An example of a simple BPMN process is shown in Figure 1. It contains designated start (green) and end (red) nodes which mark the entry and exit points of the process, respectively. The main elements are (atomic) *tasks* which are represented using labeled rectangles. In this example, there are specifically four tasks: Read Challenge, Specify Rules, Implement Example and Run Example. The control-flow of the process is indicated using directed edges between the nodes. In BPMN, so-called *gateways* are used to specify complex control-flow structures. In this challenge, we are focusing on so-called *parallel gateways*, which are drawn as "+"-labeled diamonds. The informal semantics of parallel gateways can be described as fork/join nodes.

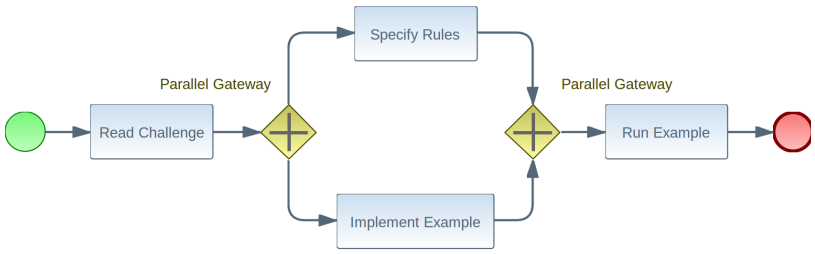


Figure 1: Example BPMN 2.0 Process.

The BPMN2 Modeler project [2] for Eclipse provides tooling for BPMN 2.0 including a graphical editor and an execution engine. Both of these components are based on a metamodel for BPMN 2.0 defined in the Eclipse Modeling Framework (EMF) [5, 3]. This metamodel is compatible with the BPMN 2.0 specification [4] proposed by the Object Management Group (OMG). A subset of this metamodel is shown in Figure 2. The Definitions class serves as root container for Processes. A Process is a special kind of *FlowElementsContainer* and can therefore store a set of *FlowElements*. There are two kinds of *FlowElements*: *FlowNodes* and SequenceFlows – the former represent the nodes and the

Submitted to:
TTC 2013

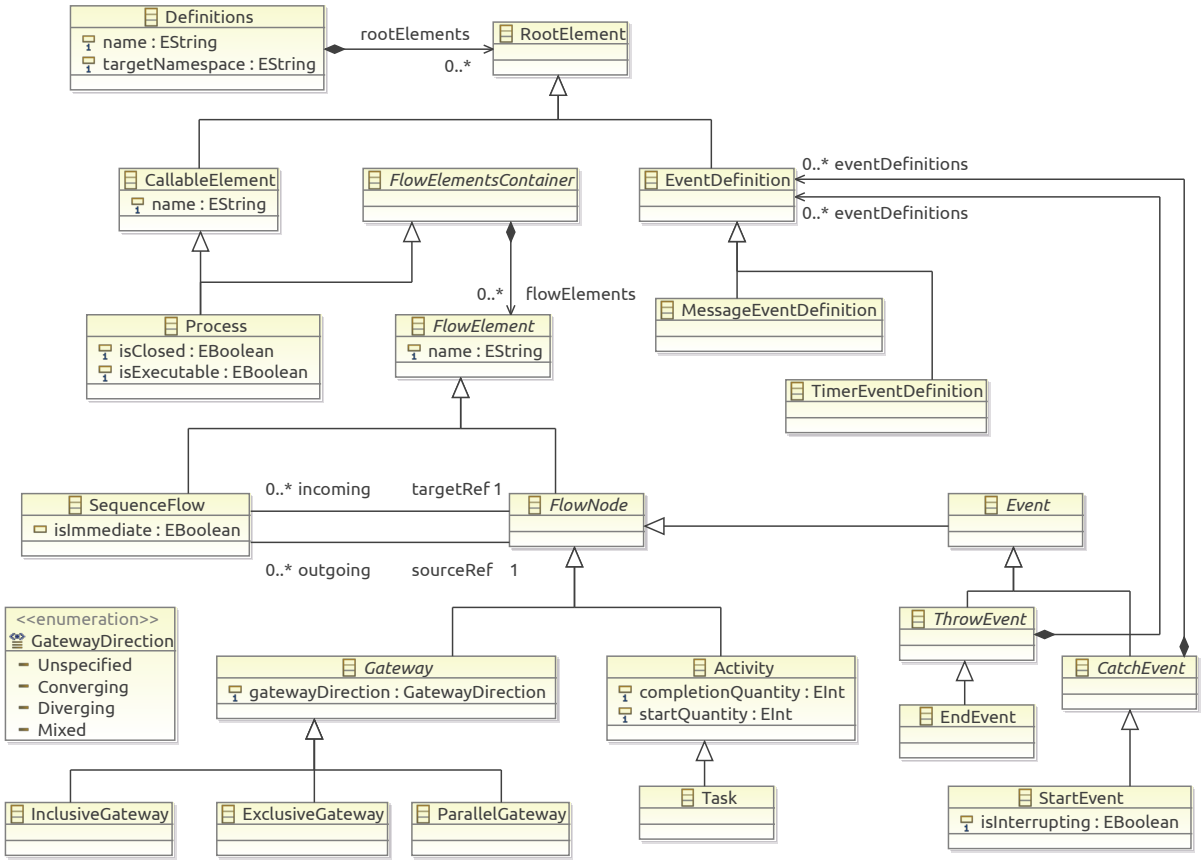


Figure 2: Subset of the EMF-based metamodel for BPMN 2.0 [2].

latter the edges in a process graph like the one in Figure 1. We distinguish between a number of different flow nodes. Tasks are flow nodes which are used as the basic executable elements in a process, e.g. the Read Challenge task in the process in Figure 1. *Gateways* are flow nodes for control-flow structures. We focus here only on the ParallelGateway, as used in the example process. Moreover, StartEvent and EndEvent are flow nodes for marking the entry and the exit points of processes. Both of them can have a number of EventDefinitions.

2 Execution Semantics

The BPMN 2.0 standard contains an informal description of the execution semantics of business processes. The semantics of BPMN 2.0 is based on so-called *token games* (similar to the operational semantics of Petri nets). In the context of this challenge, your task is to implement the execution semantics of the BPMN 2.0 language using in-place transformations. Since the time for the live challenge is limited, we focus on a restricted subset of BPMN 2.0. Before describing the tasks in detail, we first introduce a metamodel which is required for realizing the execution.

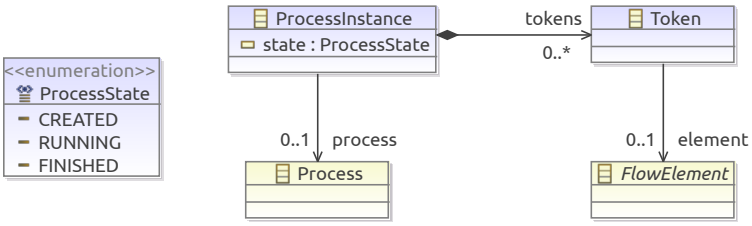


Figure 3: Minimal execution metamodel for BPMN.

Execution Metamodel

The BPMN 2.0 metamodel does not include the necessary concepts for modeling process executions. Therefore, we define the minimal execution metamodel shown in Figure 3, which you can use to specify the semantics of BPMN 2.0. The execution metamodel consists of the enum ProcessState and the two classes ProcessInstance and Token. A ProcessInstance refers to a currently executed Process of a BPMN 2.0 model, has a state given by an ProcessState enum value, and it carries a number of Tokens. Every Token refers to a FlowElement in a BPMN 2.0 process. A process instance together with its tokens can be compared to a *marking* of a Petri net. You will use this minimal execution metamodel in the following to define an interpreter semantics for the BPMN 2.0 metamodel.

TASK 1: Process Instantiation and Termination

Process Instantiation Define a rule which matches a Process and creates a corresponding ProcessInstance for it and sets its state attribute to RUNNING. The rule should also create a Token at the StartEvent of the process. The process instantiation should be allowed only if the StartEvent either has no EventDefinitions or if all its EventDefinitions are TimerEventDefinitions. Note that the BPMN 2.0 metamodel contains in fact more types of EventDefinitions than shown in Figure 2.

Normal Termination Define one or more rules that handle normal termination of a ProcessInstances. The rule should apply only to process instances where the state attribute is set to RUNNING. Moreover, one of the following two conditions must be fulfilled:

- the process contains an EndEvent and all tokens of the process instance are in an EndEvent, or
- the process contains no EndEvent and all tokens are in flow nodes without any outgoing arcs.

Note that an outgoing arc of a FlowNode is represented by a SequenceFlow object with a set sourceRef property. If one of the above two conditions are fulfilled, the rule(s) set the state attribute of the ProcessInstance to FINISHED and delete *all* tokens of the process instance.

For all of the folowing tasks, we assume that matched process instance is in the state RUNNING.

TASK 2: Starting and Ending

Starting Define a rule which moves a Token located at a StartEvent of the current process instance to one of its outgoing SequenceFlow objects.

Ending Define a rule which moves a Token of the current process instance located at a SequenceFlow with a EndEvent as target to this EndEvent.

TASK 3: Entering and Leaving Tasks

Entering Tasks Define a rule for entering Tasks. For an active ProcessInstance, the rule should match a Token of this process instance located at a SequenceFlow object. If the target of the SequenceFlow is a Task, the rule should delete the Token's reference to the SequenceFlow and create a reference to the target Task.

Leaving Tasks Define one or more rules for leaving a Task again. The rules can apply only if there is at least one outgoing SequenceFlow object. The leaving semantics is realized by a so-called *AND-split* workflow pattern [1]. That means, the the token at the Task object is deleted and at every outgoing SequenceFlow object a new Token object is created for the current process instance. Ideally, the leaving of tasks should be realized as an atomic step, i.e., by a single rule application.

TASK 4: Parallel Gateways

Entering Parallel Gateways Define one or more rules for entering a ParallelGateway. The rules can be applied only if there is at least one incoming SequenceFlow object. All incoming SequenceFlows must carry a token. For each of the incoming SequenceFlows a token should be deleted from the current process instance. At the same time, a new token should be created at the ParallelGateway. Ideally, the entering of a parallel gateway should be realized as an atomic step, i.e., by a single rule application.

Leaving Parallel Gateways Define one or more rules for leaving a ParallelGateway again. There must be at least one outgoing SequenceFlow object. The token at the ParallelGateway is deleted and at every outgoing SequenceFlow object of the current process instance a new token is created. Ideally, the leaving of a parallel gateway should be again realized as an atomic step, i.e., by a single rule application.

3 Manual and Automatic Execution

TASK 5: Use your rules to execute the provided example process models. There exist two types of nondeterminism in the execution semantics. Specifically, in any given state two types of choices can be necessary: (i) which rule should be applied, and (ii) which match should be used. Please execute the provided process models using the following strategies for resolving the nondeterminism during the execution:

- **Interactive resolution**. Whenever a choice has to be made, the execution should be interrupted. The user is offered a list of applicable rules and their matches and can choose between them. The user interactions could be realized on the console or using a debugger for transformations or using a tailored graphical user interface for executing and animating BPMN process executions.
- **Random resolution**. The transformation engine flips a fair coin to decide which rule and which match to use next.
- **Prioritized tasks**. If the rule for entering tasks is enabled, it gets priority over all other rules. If the rule can be applied to more than one task, the alphabetical order of the names of the task should be used to decide which task to be entered next. All other types of nondeterministic choices should be resolved randomly.

4 State Space Generation and Analysis

TASK 6: Generate the full state space for the process in statespace_example.bpmn. How many states are there? Are all generated states valid markings of the BPMN process?

TASK 7: We define a state as *terminal* if the state attributes of all ProcessInstances of the state are set to FINISHED. Now, use the (generated) state space to validate the following properties.

- **State Invariants.** In every state, all tokens have a reference to a FlowElement.
- **Deadlock freedom.** The system is deadlock-free, except for the terminal states. This means that in all non-terminal states at least one rule is applicable.
- **Termination.** In every non-terminal state it is possible to reach a terminal state.
- **Causality.** Task X is always executed before task Y.

References

[1] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski & Alistair P. Barros (2003): *Workflow Patterns*. *Distributed and Parallel Databases* 14(1), pp. 5–51. Available at `http://dx.doi.org/10.1023/A:1022883727209`.

[2] Eclipse Foundation: *BPMN2 Modeler*. `http://www.eclipse.org/bpmn2-modeler`.

[3] Eclipse Foundation: *Eclipse Modeling Framework*. `http://www.eclipse.org/modeling/emf`.

[4] OMG: *Business Process Model And Notation (BPMN) Version 2.0*. `http://www.omg.org/spec/BPMN/2.0/PDF`.

[5] Dave Steinberg, Frank Budinsky, Marcelo Paternostro & Ed Merks (2009): *EMF: Eclipse Modeling Framework*, 2. edition. Addison-Wesley.

Supplementary Material

You can find a quick start guide to BPMN 2.0 in Appendix 1. This guide was kindly provided by Louis Rose. A graphical representation of the state space of the example process is shown in Appendix 2.

A Guide to Business Process Model and Notation

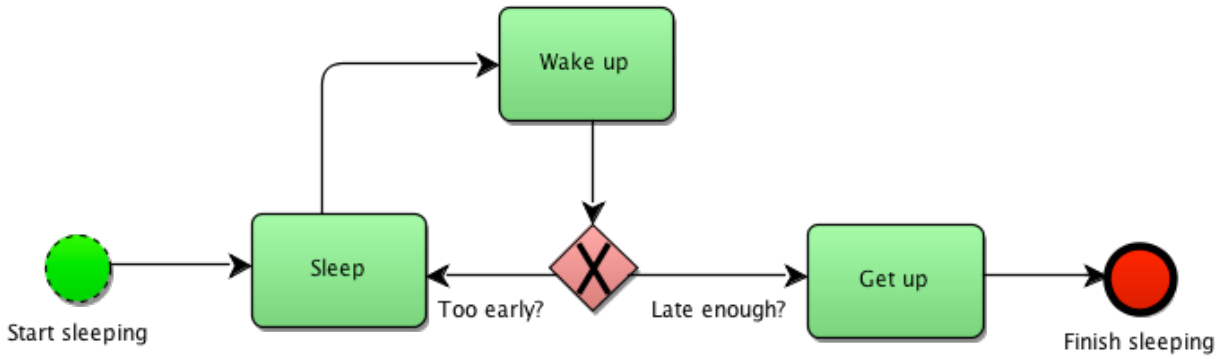
This is a brief guide to BPMN, based on “Introduction to BPMN 2” by Jim Arlow and Ila Neustadt and on BPMN 2.0 By Example by the Object Management Group.

Overview

- BPMN is a standardised language for describing business processes
- BPMN evolved from flowcharts and UML activity diagrams.
- It’s less technical than process execution languages (such as WS-BPEL, which includes concepts that are particular to web services)
- Used for modelling:
  - *Processes*: a sequence of activities and events that constitute a business process
  - *Collaborations*: interactions (e.g. messages) between different business processes
  - *Choreographies*: a collaboration that focuses on interactions between participants and hides process details.
- There are no diagram types in the BPMN specification (unlike UML), although tool vendors might recommend / enforce their own diagram types.
- BPMN is useful for modelling how an organisation functions at a relatively high-level of abstraction.
- BPMN doesn’t typically describe schedules (when?) or business goals (why?)

Processes

A process comprises *events* (things that happen), *activities* (work that is performed), *gateways* (things that control flow). Events, activities and gateways are example of *flow objects*. *Sequence flows* are used to connect two flow objects.



Below, we see a process that describes my morning routine. There are two events (a start event and an end event), three activities (sleep, wake up and get up), an exclusive gateway, and several sequence flows. Exclusive gateways represent decisions.

A process can be *instantiated* many times. Instantiation normally occurs each time the start event is triggered. For example, we might instantiate the above process once for each person every night.

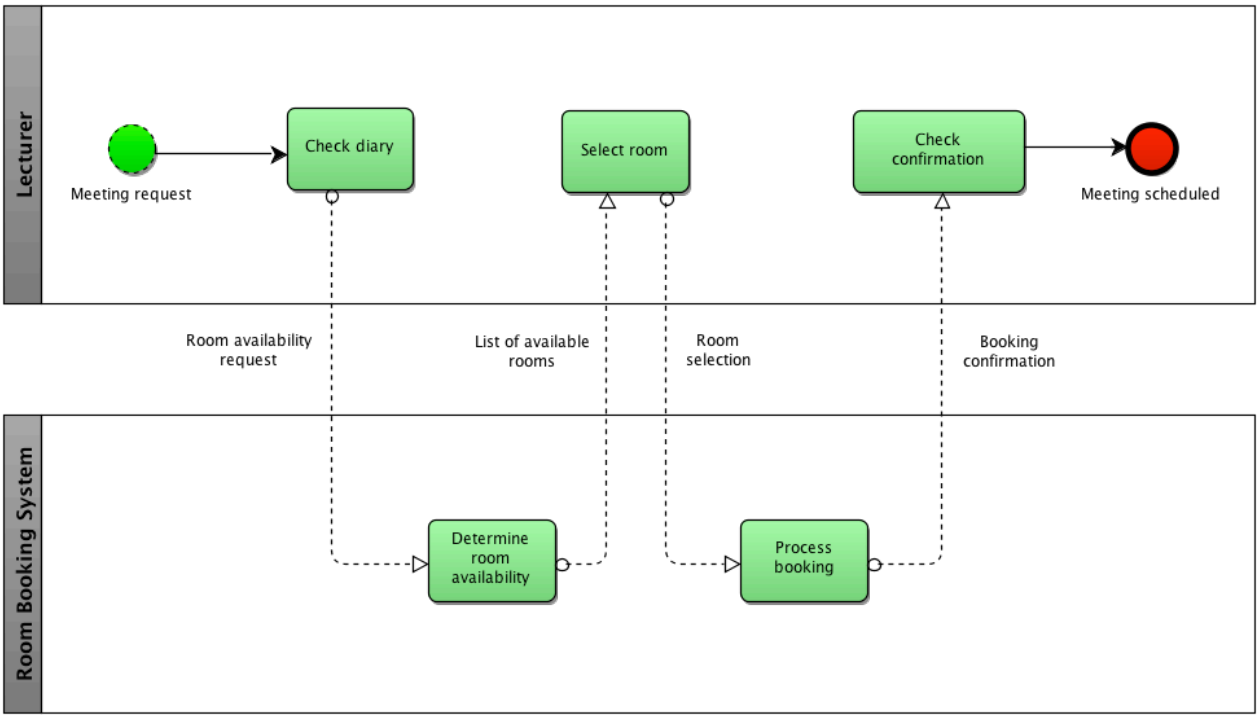
When a process is instantiated, its current state is represent by a *token.* Each flow object (i.e. an event, activity or gateway) consumes tokens, produces tokens, or some combination of both. For example:

- A start event produces a token.
- An end event consumes a token.

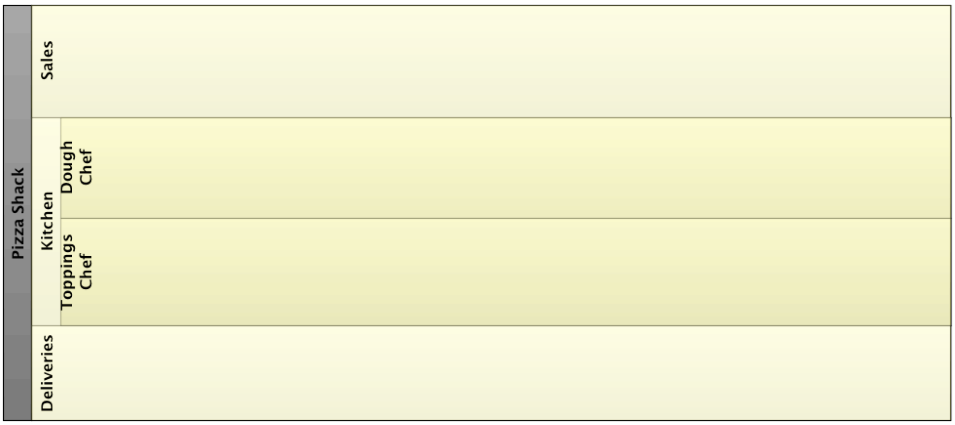
- An activity consumes a token, does some work and then produces a token.
- An exclusive gateway consumes a single token from **any** of its inputs, and then produces a single token on **one** of of its outputs.

Collaborations
A collaboration typically shows the interactions between several processes. As there are no diagram types in BPMN, we merely use additional language constructs to represent collaborations, including *pools* and *lanes* (which represent different participants in the process) and *message flows* (which indicate how information is exchanged between different processes).

Below, we see a collaboration that describes how to schedule a meeting. The collaboration involves two participants (the lecturer and the room booking system), which are modelled using separate pools. Inside a pool, we can use object flows (such as events, activities and gateways). Communication between pools is represented using a message flow. We see that the "check diary" activity of the lecturer causes a message to flow to to the "determine room availability" activity of the room booking system. Message flows can be named to indicate the type of information that is flowing between participants.

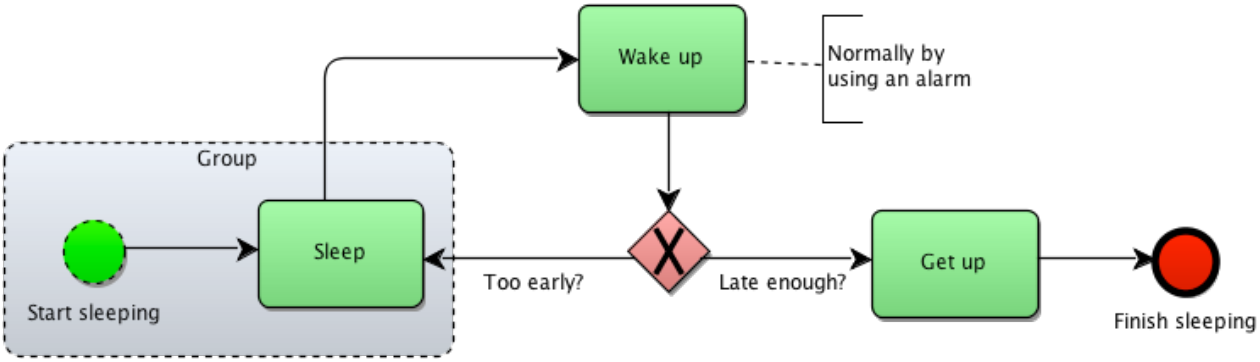


Pools can be divided into lanes, as shown below. Pools and lanes can be used to model the logical groupings that make sense for your problem. For example, you can use a pool (or lane) to represent an entire organisation (e.g. university), part of an organisation (e.g. department of Computer Science), a role (e.g. head of department), or a person (e.g. Louis). Lanes can be nested.

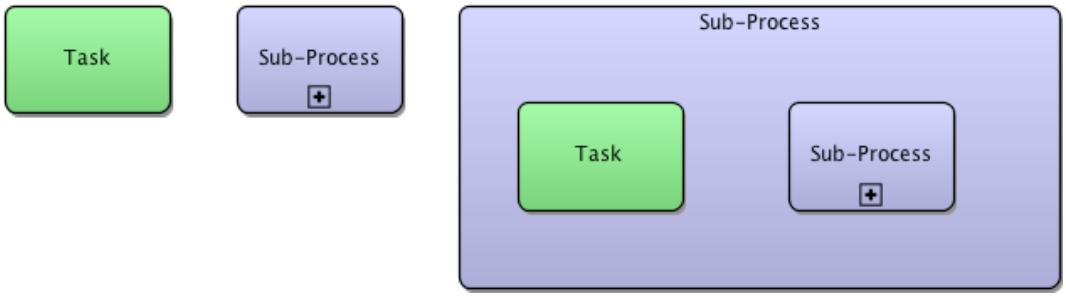


Groups and Annotations
There are a couple of language constructs in BPMN that you can use to clarify your diagrams, *groups* and *annotations*. Neither constructs affects the way in which processes are instantiated or executed.

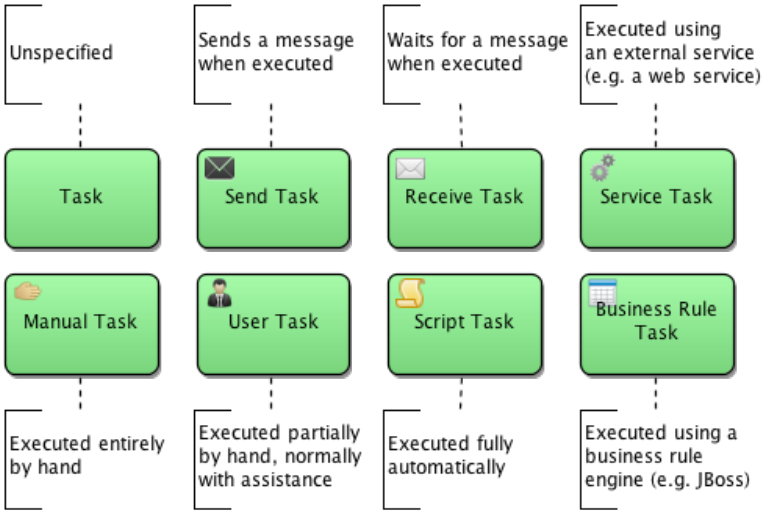
In the diagram below, we see that a group has been used to indicate the object flows in which I am unconscious. (NB: due to a bug in my BPMN editor, I've not been able to label this group). Also, a text annotation has been attached to the wake up activity.



Activities
Activities are a richer construct than we have explored so far. Until now, we have actually been considering only a specific type of activity known as a *task* (an atomic piece of work). A further type of activity is a *sub-process*. A sub-process contains other activities (including other sub-processes).

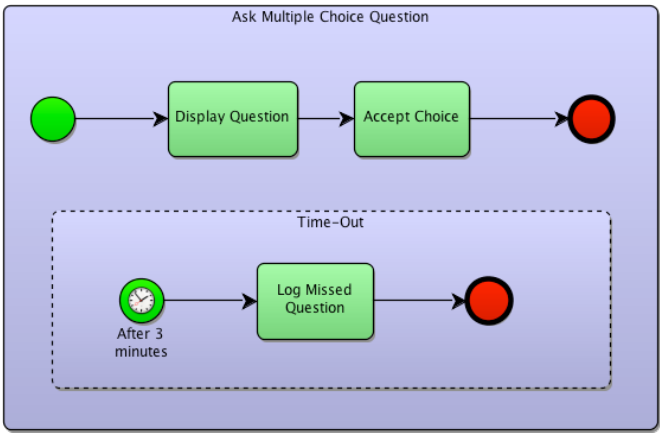


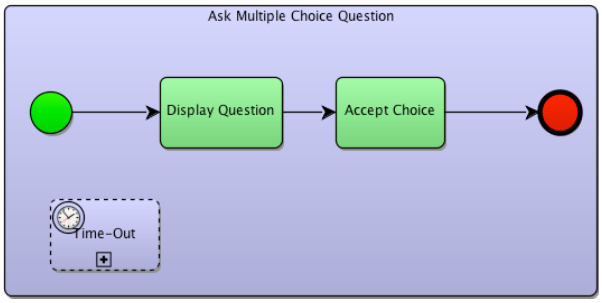
Often, it is helpful to indicate the nature of task: does a task involve automation or sending a message? BPMN provides several *types* of task for this purpose, as shown here.



There are also different types of sub-process. The most common types of sub-process are *embedded* (introduced above) and *event*. Event sub-processes have a dashed border and are triggered with a message, timer, condition, etc. (More on these kinds of triggers in the following section).

Below, we see the use of an event sub-process to specify that the parent process will time-out after 3 minutes. Here, there is a normal flow (show in the top of the process) and an interrupting flow that will halt the normal flow (shown in the bottom of the process). The figure on the right-hand side shows a collapsed syntax, which can be used to hide detail.





Events

The way in which events are executed is a little more sophisticated than we have seen so far. Events are either *throwing* (consume a token and cause a trigger) or *catching* (produce a token and wait for a trigger).

Start events are catching. In the Ask Multiple Choice Question example, there are two start events: one has an unspecified trigger, the other has a time trigger (it is fired after 3 minutes have elapsed).

End events are throwing. They always consume a token, and may optionally cause a trigger (such as sending a message).

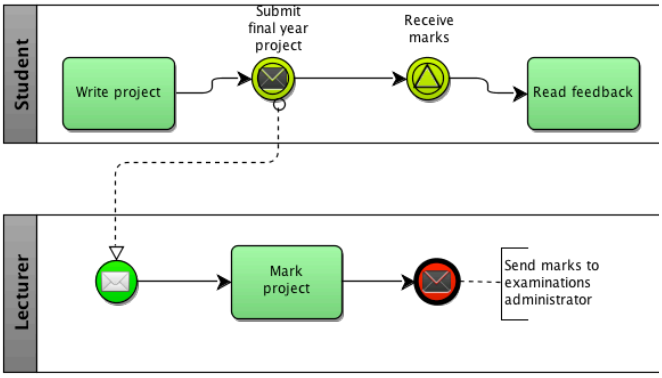
Below we see the different types of catching and throwing events supported by BPMN.

Message events involve sending or receiving a point-to-point (1-to-1) communication using a message flow.

Signal events involve sending or receiving a broadcast (many-to-many) communication. Message flows are not used between signal events.

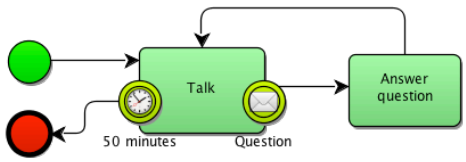
Time events occur at a point in time, after a duration, or after a repeating time condition (e.g. every Thursday).

Conditional events occur when a Boolean expression evaluates to true.
In addition to start and end events, there are *intermediate* events (which are used to highlight milestones partway through a process) and *boundary* events (which can be used to modify an activity).



To the left, we see part of a collaboration that shows how student projects are conducted. The pool representing the student's participation contains two intermediate events. The first is a throwing event that sends a message directly to the lecturer. The second intermediate event is a catching event that waits for a signal is received. I've used a signal rather than a message to indicate that notification that marks are available is a signal received by more than one type of participant (perhaps lecturers also need to respond to this signal).

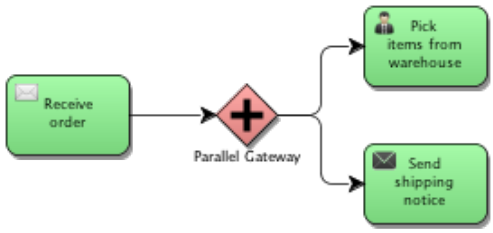
Boundary events modify an activity. To the right, we see the use of two boundary events. The first is triggered after 50 minutes has elapsed, and causes the process to end. The second boundary event is triggered when a message is received, and causes execution of the process to proceed to the "Answer Question" activity.

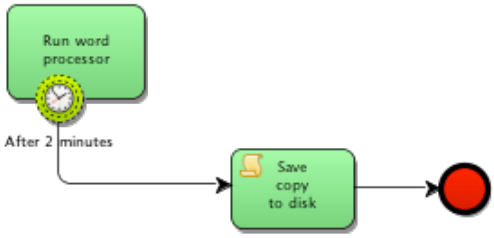


Parallelism
Until now, we have assumed that when a process is executed only one token is used to represent the current state of the process. In fact, there are several constructs that can be

used to introduce more than one token into the execution of a process. Two of the most useful constructs are the *parallel gateway* and the *non-interrupting boundary event*. They are both variations on constructs that we have explored before.

To the right, we can see the use of a parallel gateway. The parallel gateway symbol (+) is unfortunately similar to an exclusive gateway (X), but semantically the two are quite different. A parallel gateway waits for a token on **all** of its inputs and emits a token on **each** of its outputs. (An exclusive gateway waits for a token on **any** of its inputs and emits a token on **one** of its outputs). Consequently, a parallel gateway can be used to fork the flow of execution. In this example, after receiving an order, both the “Pick items from warehouse” and the “Send shipping notice” tasks are executed.





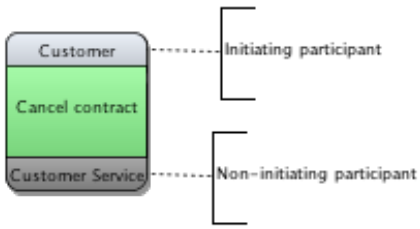
To the left, we see the use of a non-interrupting boundary event, which forks the “Run word processor” activity to execute the “Save copy to disk activity.” Note that after saving to disk, that thread of execution finishes (i.e. its token is consumed by an end event). The “Run word processor” continues to execute even after the end event has been executed.

Note that the *terminate event* is used to halt all activities (i.e. consume every token active in the process). In the example above, if the end event were replaced with a *terminate event*, the word processor task would be halted after the “Save copy to disk” event completed.

Choreographies

Choreographies are a fairly new addition to BPMN and are not yet very widely used. They do, however, provide a concise summary of larger collaborations. Furthermore, they focus on interactions rather than the participants themselves. I imagine that choreographies could be used quite effectively to model value streams.

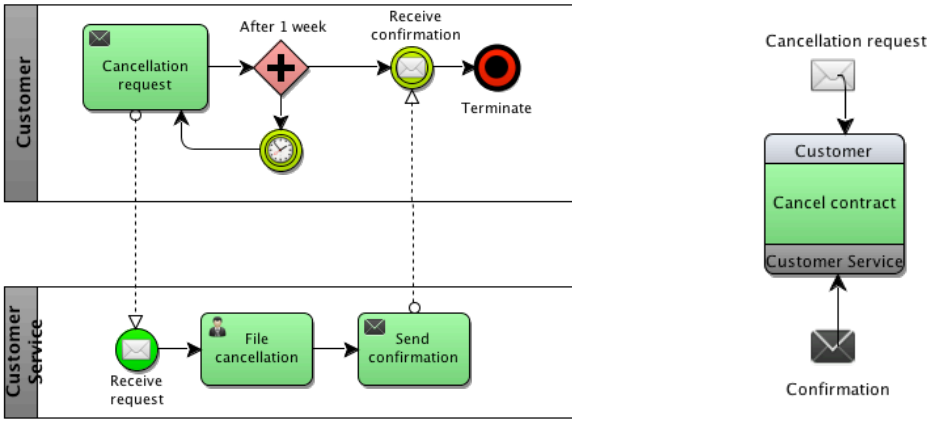
Choreography activities are a special kind of activity that provide a compact representation of a collaboration. As shown below, a choreography activity comprises an initiating participant, a non-initiating participant and the name of the activity.



Choreography activities are normally connected to gateways and events (just like tasks). In the manner, choreographies that summarise substantial parts of an organisation can be specified.

Finally, we can see below that when we are not concerned about the details of a collaboration, a choreography provides a more compact representation. On the left, we

see the details of the collaboration between a customer and a customer service department. On the right, we see the an overview of the collaboration, represented as a single choreography activity.



Relationship to UML
UML is a broader language than BPMN. Much of BPMN can be achieved in UML via activity diagrams. However, BPMN is often more concise and more expressive than UML: complicated process can be described succinctly with BPMN. As far as I know, UML doesn't provide any choreography constructs.

Further reading
Although I've tried to describe many of the common uses of BPMN in this guide, there are additional BPMN constructs that I've not introduced. For more information, or a more detailed look at the semantics of BPMN constructs, try the following resources

- BPMN Specifications, Object Management Group, http://www.omg.org/spec/BPMN/
  Definitive references for every version of BPMN.

- Introduction to BPMN 2, Jim Arlow and Ila Neustadt, Mountain Way Ltd, 2011
  with complementary slides: http://www.slideshare.net/jimarlow/introductiontobpmn005
  The best distillation of the BPMN 2.0 specification that I've found.

- BPMN 2.0 Reference Card: http://www.bpmb.de/images/BPMN2_0_Poster_EN.pdf,
  A handy "cheat sheet" for BPMN 2.0.

- BPMN 2.0 by Example, Object Management Group: http://www.omg.org/spec/BPMN/20100601/10-06-02.pdf
  If you'd like more detailed or larger examples of using BPMN, this is a great place to start.

Appendix 2

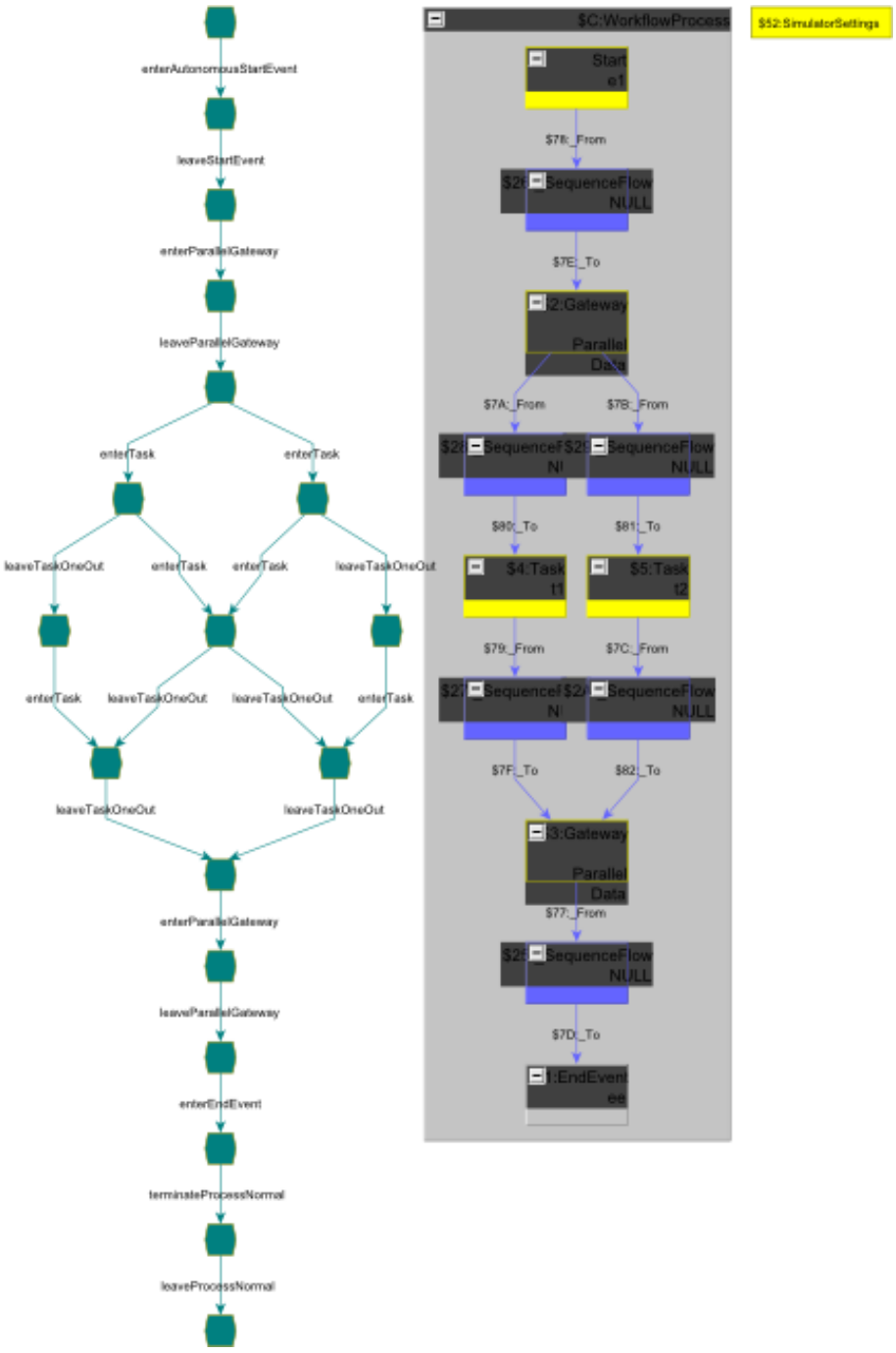


Figure 4: State space of the example process.