Transformation of BPMN models to Petri nets

Motivation

- Check correctness of BPMN models
  - Develop new checking algorithms?
  - Be smart and benefit from what is already there for Petri nets
- Transformation
  - Map high-level and rich concepts of BPMN to P/T Petri nets
  - Consider only the required level of abstraction.

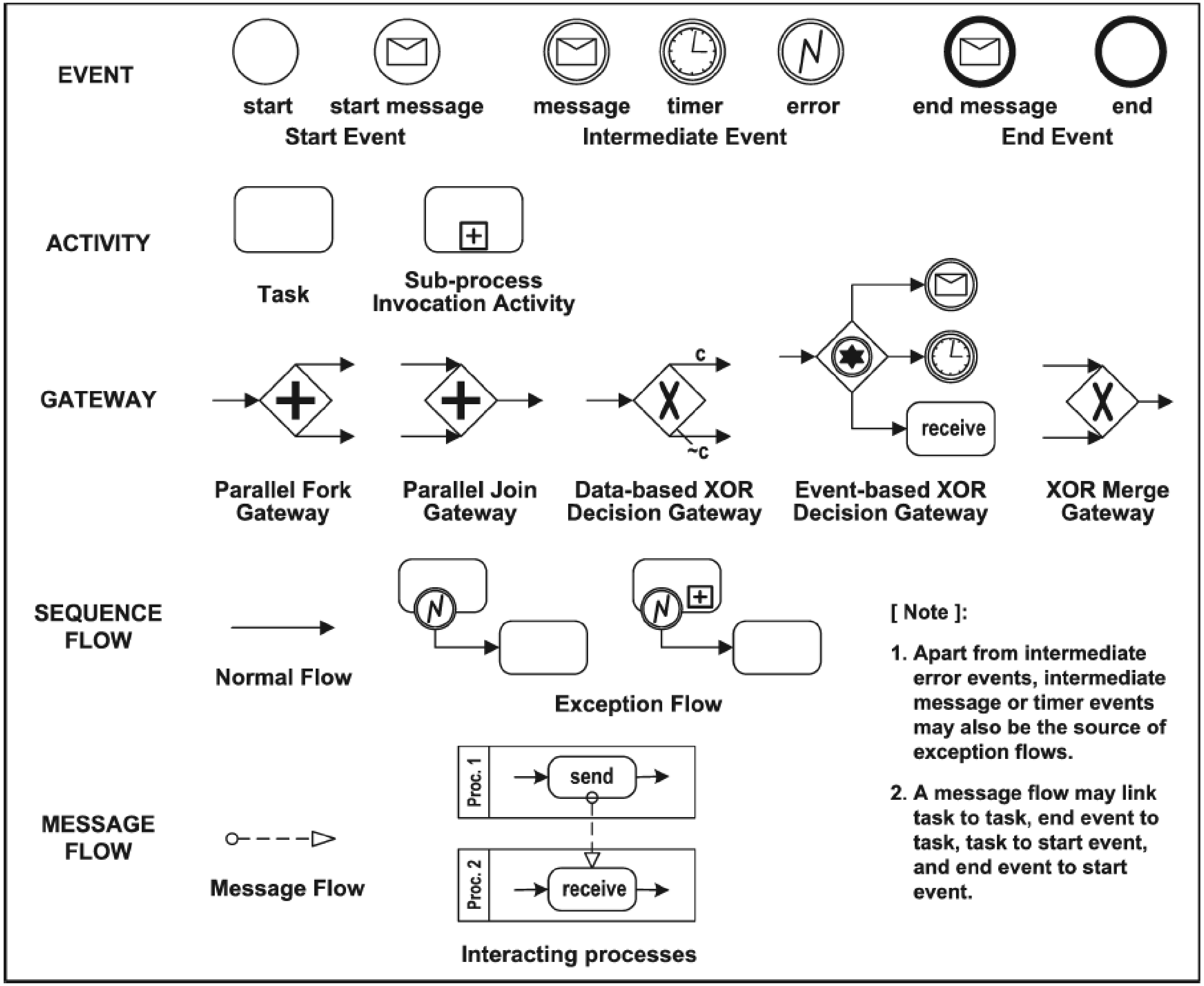
Analysis of BPMN Process models

- Idea
  - BPMN has no formally defined execution semantics
  - Use the mapping to Petri-nets for analysis of the resulting nets.
  - The resulting net is fed into an analysis tool, e.g., Woflan.
  - To recognize structural problems as well as deadlocks or improper termination.
- Tool Chain
  - BPMN -> Petri-Nets
  - Petri-Nets -> Woflan (Workflow Analyzer from TU Eindhoven)

Correctness of BPMN models

- Idea
  - In order to decide properties about BPMN diagrams formalization is necessary
  - More complex than, e.g., Workflow nets, because
    - Less limited structure of the diagrams
    - Multiple-instance activities and subprocesses
    - Exception handling Intermediate Events
    - Message flow among processes
- Approach [Dijkmann, Dumas, Ouyang 2007]
  - Formalization of a subset of BPMN by mapping to Petri nets and analyzing these Petri nets

Subset of BPMN



[Note]:

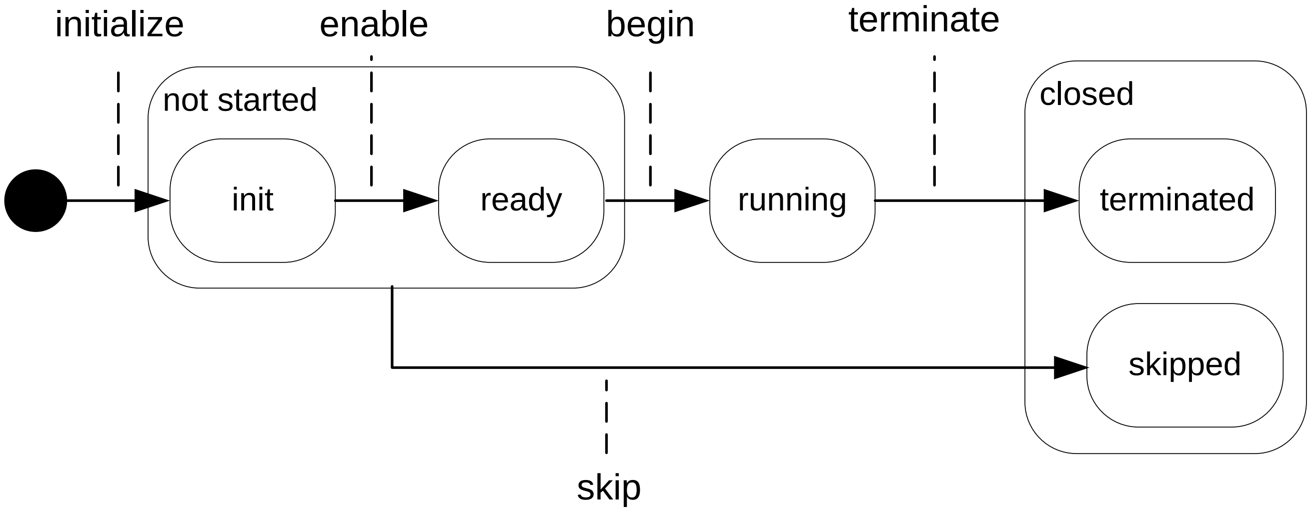
1. Apart from intermediate error events, intermediate message or timer events may also be the source of exception flows.

2. A message flow may link task to task, end event to task, task to start event, and end event to start event.

Figure 1. A core subset of BPMN elements.

Activity instances

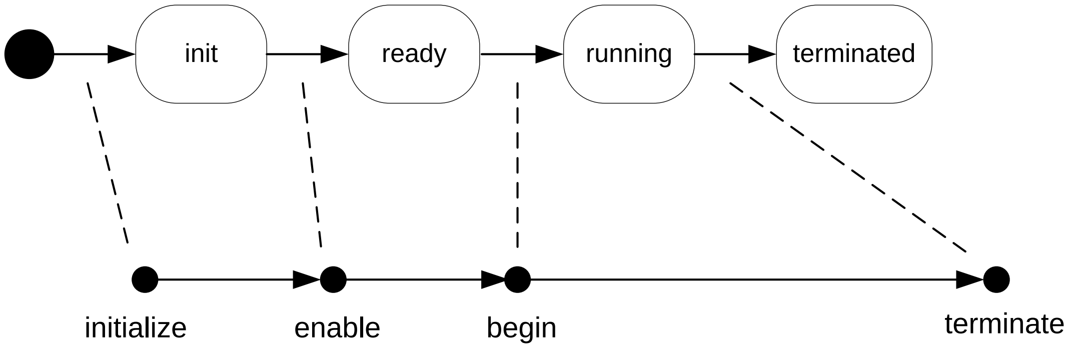
- Idea
  - Each instance goes through a series of states
  - At any time it is in exactly one state
  - States and transitions are represented by a diagram (state machine)
  - State transitions occur as response to events



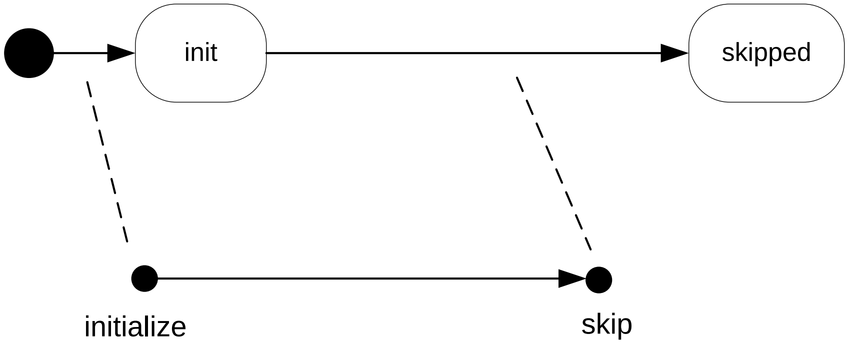
from M. Weske: Business Process Management, © Springer-Verlag Berlin Heidelberg 2007

Events and States

- State transitions occur as response to events
- Representation by event charts



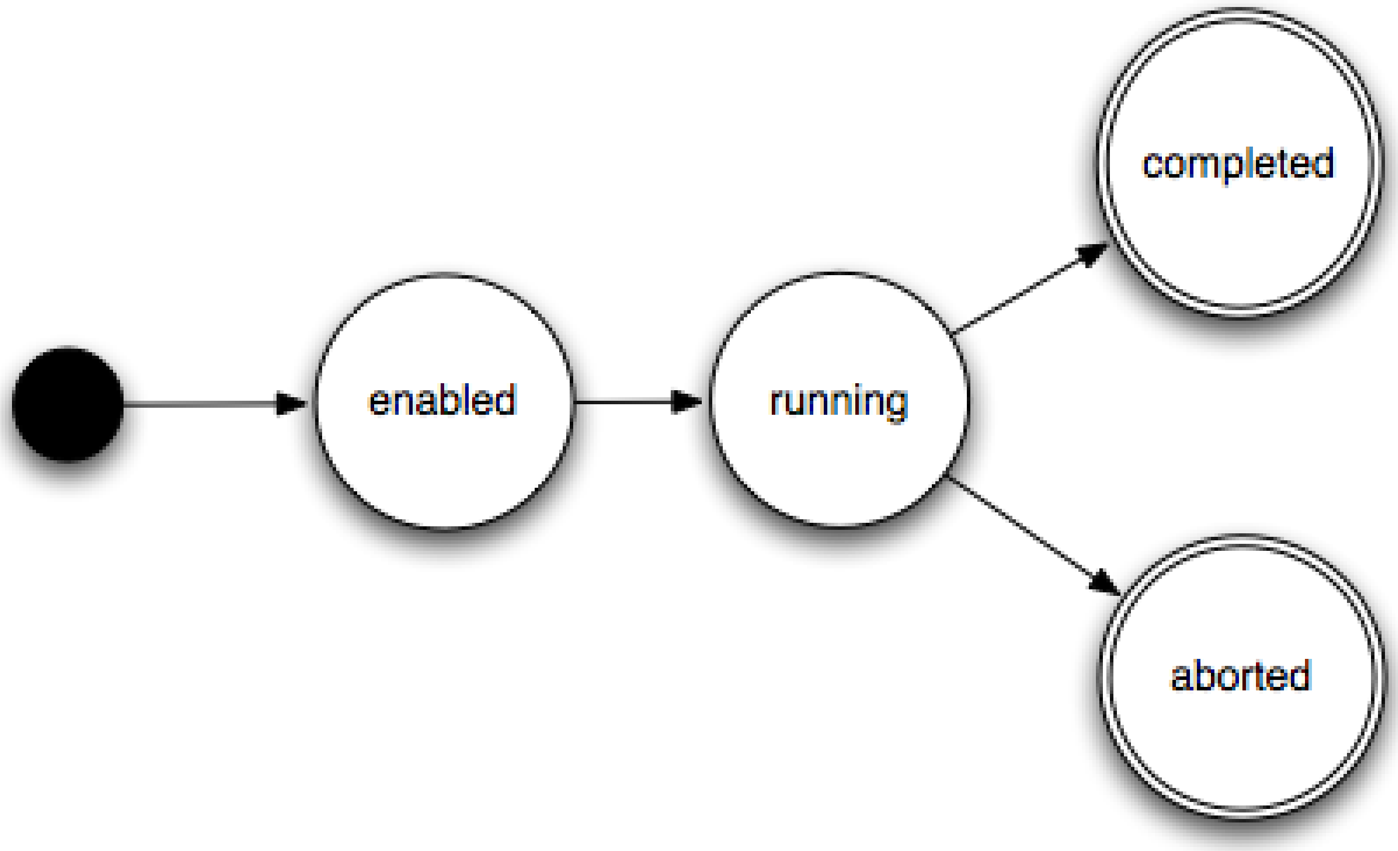
(a)



(b)

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

Activity instance with exception



Well formed Process Diagram

- Defined as
  - Number of incoming / outgoing edges for
    - Start Events
    - Intermediate Events
    - End Events
  - Successor node of event-based XOR gateways
  - For Data-based XOR Gateway
    - Order of evaluation
    - Default condition is the last
  - Structural Soundness

Well-formedness

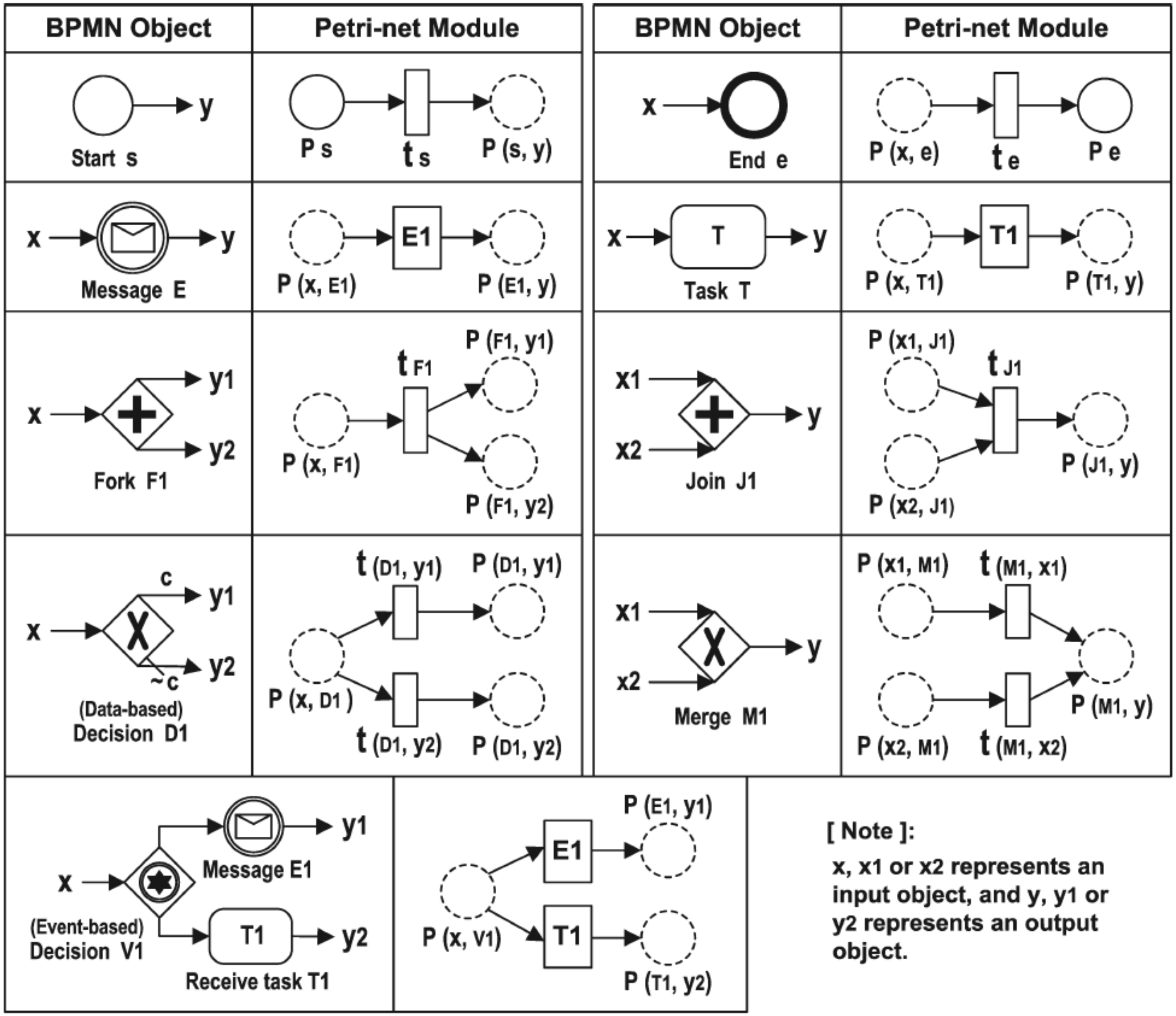
Definition 2 (Well-formed core BPD). *A core BPD is well formed iff relation $\mathcal{F}$ satisfies the following requirements:*

- $\forall\ s \in \mathcal{E}^S,\ \mathsf{in}(s) = \varnothing \wedge |\mathsf{out}(s)| = 1$, *i.e., start events have an indegree of zero and an outdegree of one,*
- $\forall\ e \in \mathcal{E}^E,\ \mathsf{out}(e) = \varnothing \wedge |\mathsf{in}(e)| = 1$, *i.e., end events have an outdegree of zero and an indegree of one,*
- $\forall\ x \in \mathcal{T} \cup \mathcal{E}^I,\ |\mathsf{in}(x)| = 1$ *and* $|\mathsf{out}(x)| = 1$, *i.e., tasks and intermediate events have an indegree of one and an outdegree of one,*
- $\forall\ g \in \mathcal{G}^F \cup \mathcal{G}^D \cup \mathcal{G}^V,\ |\mathsf{in}(g)| = 1 \wedge |\mathsf{out}(g)| > 1$, *i.e., fork or decision gateways have an indegree of one and an outdegree of more than one,*
- $\forall\ g \in \mathcal{G}^J \cup \mathcal{G}^M,\ |\mathsf{out}(g)| = 1 \wedge |\mathsf{in}(g)| > 1$, *i.e., join or merge gateways have an outdegree of one and an indegree of more than one,*
- $\forall\ g \in \mathcal{G}^V,\ \mathsf{out}(g) \subseteq \mathcal{E}^I \cup \mathcal{T}^R$, *i.e., event-based XOR decision gateways must be followed by intermediate events or receive tasks,*
- $\forall\ g \in \mathcal{G}^D$, $\exists$ *an order* $<_g$ *which is a strict total order over the set of flows* $\{g\} \times \mathsf{out}(g)$, *and for* $x \in \mathsf{out}(g)$ *such that* $\neg\ \exists_{f \in \{g\} \times \mathsf{out}(g)}((g,x) <_g f)$, (g,x) *is the default flow among all the outgoing flows from* g,[6]
- $\forall\ x \in \mathcal{O},\ \exists\ s \in \mathcal{E}^S,\ \exists\ e \in \mathcal{E}^E,\ s\mathcal{F}^* x \wedge x\mathcal{F}^* e$,[7] *i.e., every object is on a path from a start event to an end event.*

Mapping to Petri nets

- Idea
  - Activities, events and gateways are mapped to Petri net modules
- Notes
  - Activities with attached intermediate event need special handling
  - Intermediate event timer will be shown analogous intermediate message event
  - Introduction of non-visible (silent) transitions, e.g. XOR-split

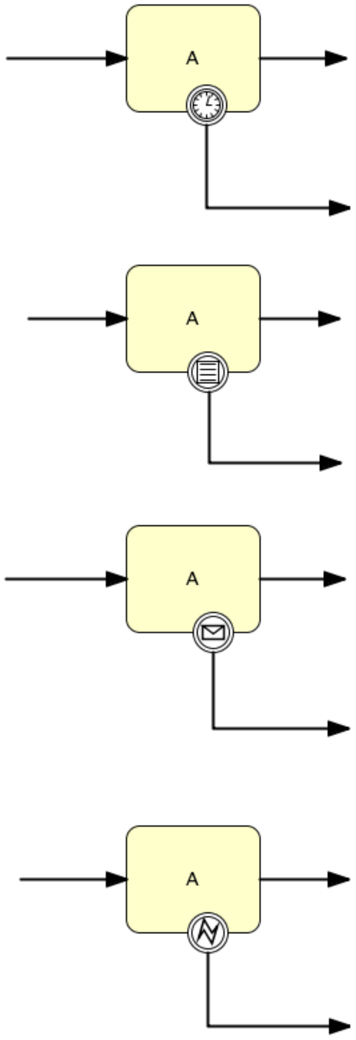
Activities, Events and Gateways

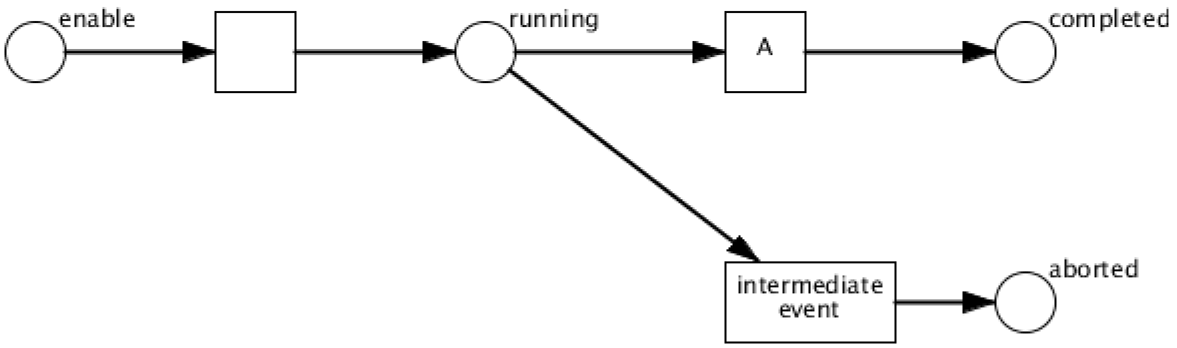


[Note]:

x, x1 or x2 represents an input object, and y, y1 or y2 represents an output object.

Activities with Attached Intermediate Events





Sub-processes

- Idea
  - Sub-processes are invoked
  - Call and return are modeled by transitions

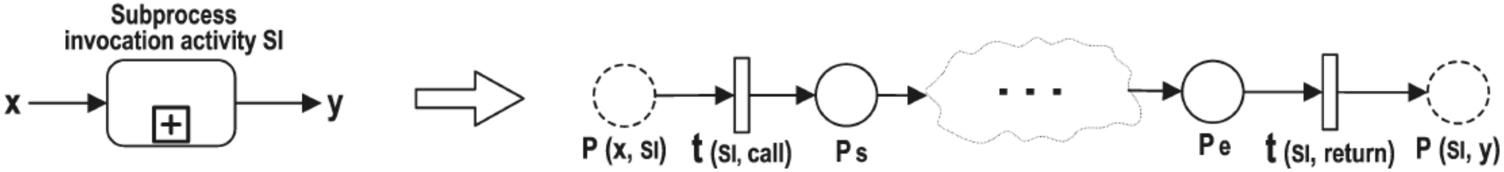
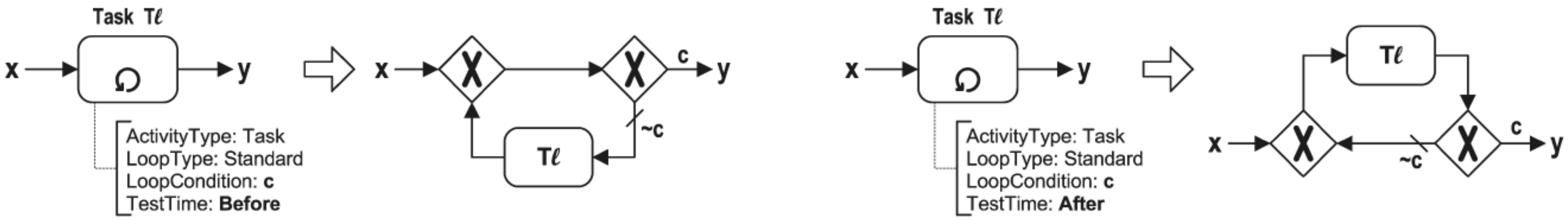


Figure 4. Calling a subprocess $\mathcal{P}$ via a subprocess invocation activity SI.

Loops

- Idea
  - Two types of loops are possible in BPMN
  - By "test time" before or after "while" or "repeat-until" loops are represented
- Sorry, there is an error in the following figure …



(a) "while-do" loop

(b) "do-until" loop

Figure 10. Macros for structured activity looping.

MI-Activity

- Remark
  - Just considered: Number of instances statically known
  - In this case, with (MI_Ordering = parallel) it can be represented by an AND-spilt.

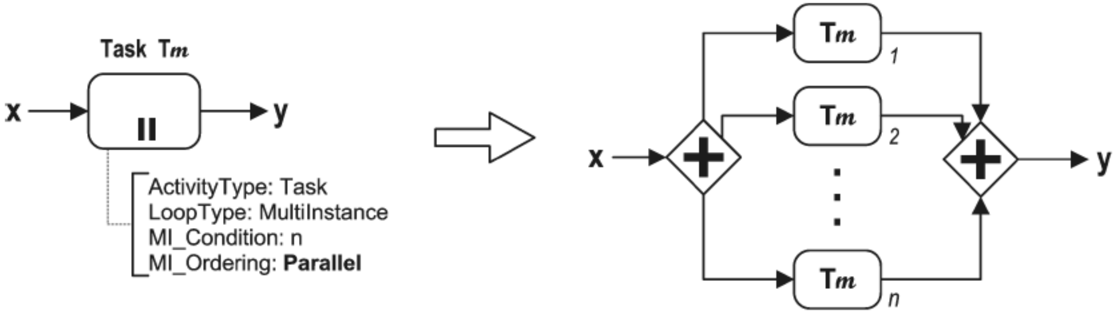
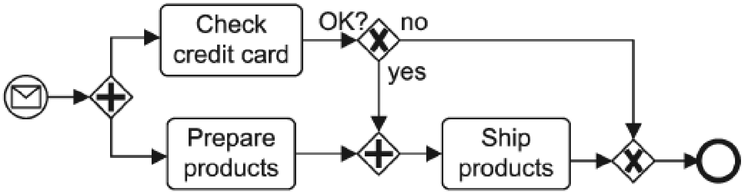


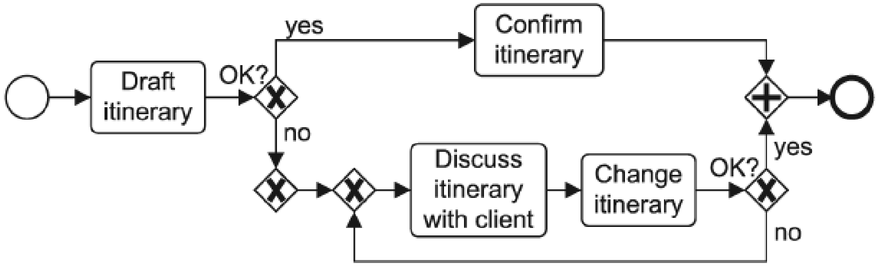
Figure 11. Macro for multi-instance activity of which the number of multiple instances is known at design time.

Example



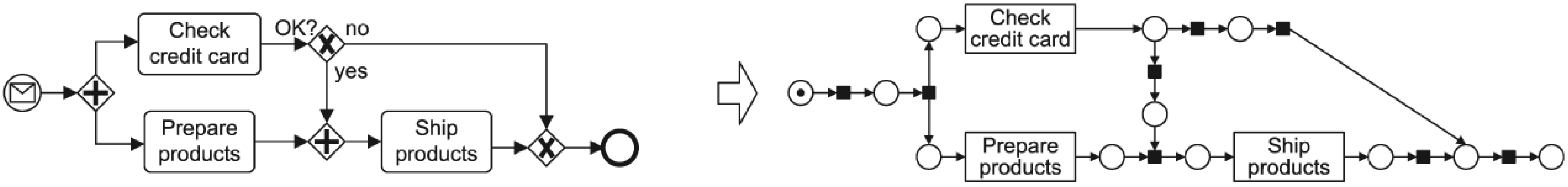
(a) Order process

Example

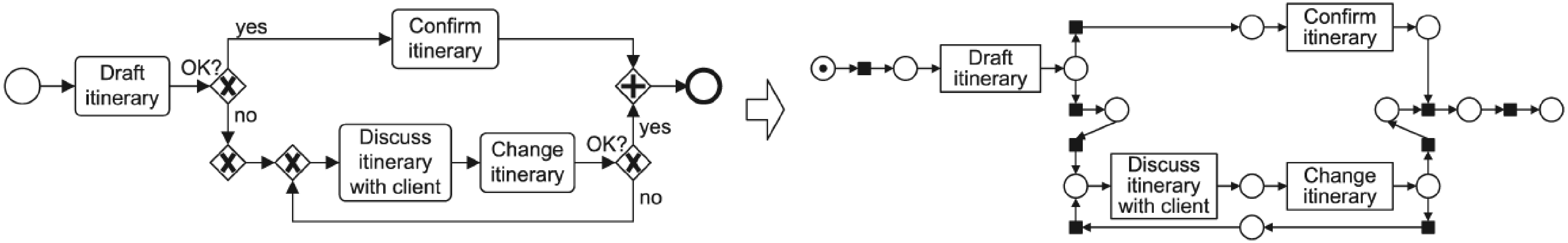


(b) Travel itinerary process

Example



(a) Order process



(b) Travel itinerary process