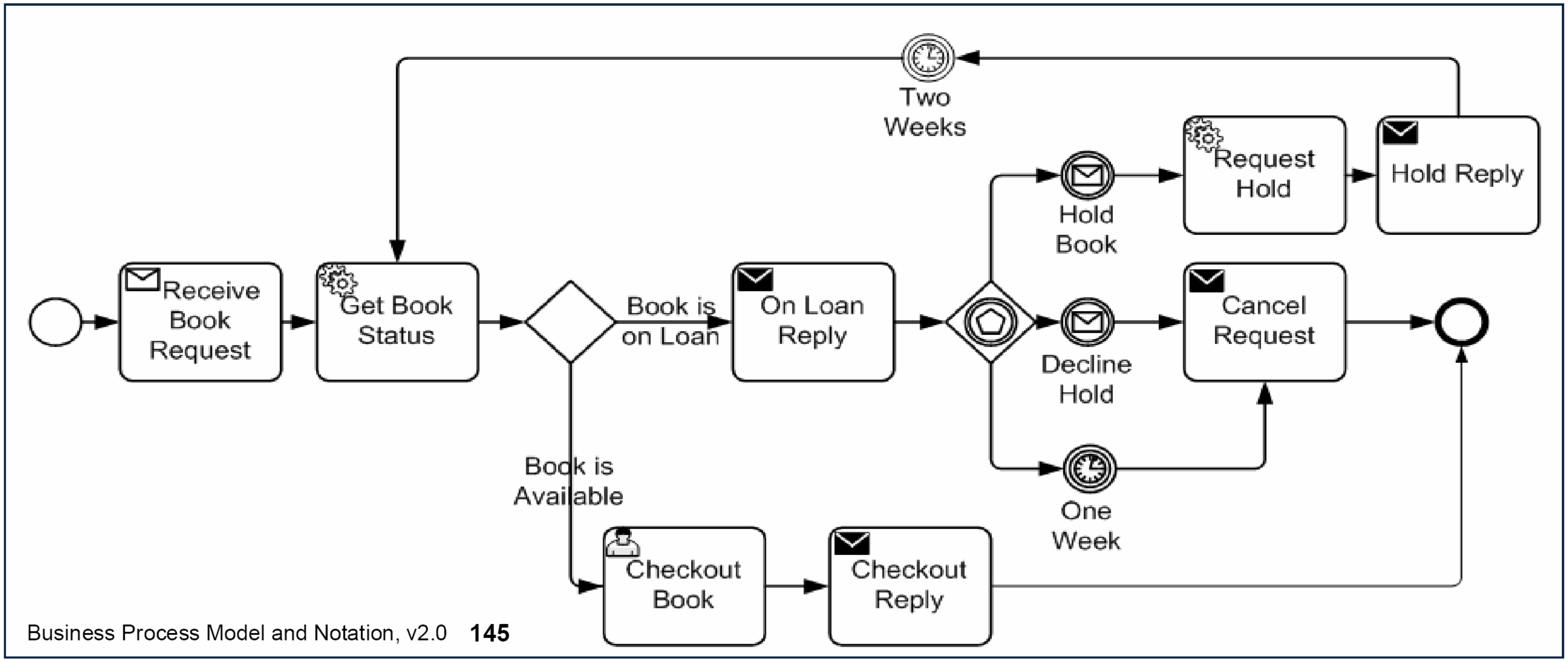
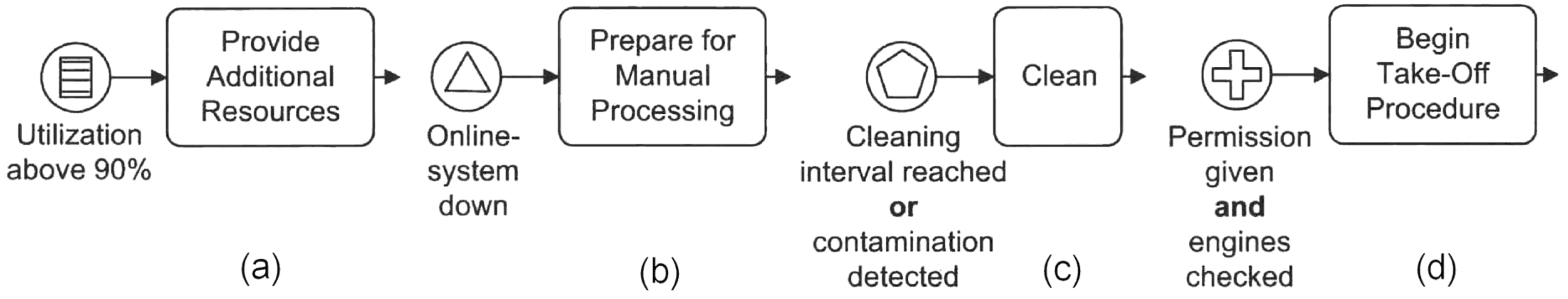
- Exercise: describe in natural language the following orchestration, concerning the book lending process managed by a library. Use both the BPMN poster and the BPMN specification to understand the meaning of unknown icons.



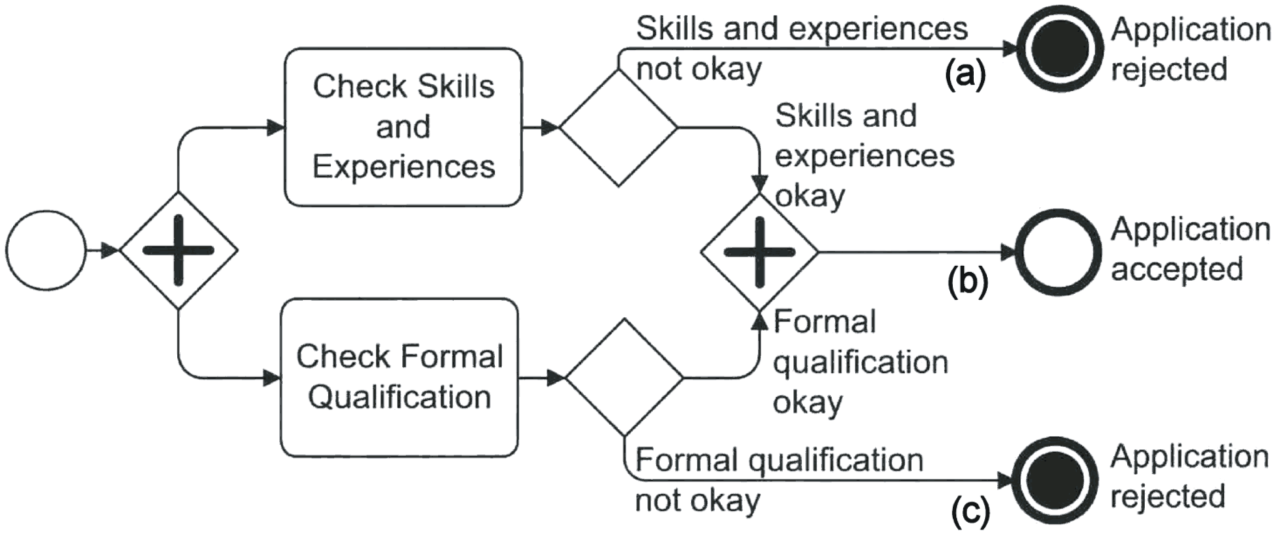
About events

- The *Conditional event* (a) starts a process when the related condition becomes true: "provide additional resources when the utilization of the current ones exceeds 90%".
- The *Signal event* (b) is triggered by the reception of a signal. In contrast to a *message* which is always sent to a specific receiver, a *signal* is broadcasted everywhere, from the same or a different pool: "when the computer system is not available, start a process for manually processing data".


Utilization above 90%
Provide Additional Resources
(a)
Online-system down
Prepare for Manual Processing
(b)
Cleaning interval reached or contamination detected
Clean
(c)
Permission given and engines checked
Begin Take-Off Procedure
(d)

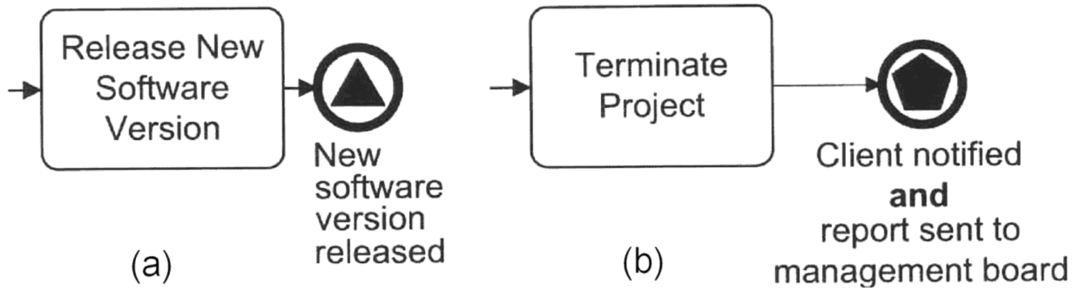
- The *Multiple event* (c) combines many events; if one of them occurs, the process is started: "a cleaning activity is started when either the cleaning interval has been reached, or a contamination has been detected".
- In the *Parallel multiple event* (d) all of the combined events must have occurred: " the take-off procedure can be started as soon as both the permission has been given and the engines have been checked".

- The *Terminate event* (a)(c), like an un-typed end event, deletes the arriving token when it arrives, but it also terminates the entire process, i.e., any other tokens in the entire process are also removed at the same time.
- Example: skills and experiences of a job applicant are checked, as well as his formal qualification is also checked in parallel. If the results of both checks are positive, the two tokens are joined together to one token which eventually is deleted (b).


Skills and experiences not okay
(a)
Application rejected
Check Skills and Experiences
Skills and experiences okay
Application accepted
(b)
Check Formal Qualification
Formal qualification okay
Formal qualification not okay
(c)
Application rejected

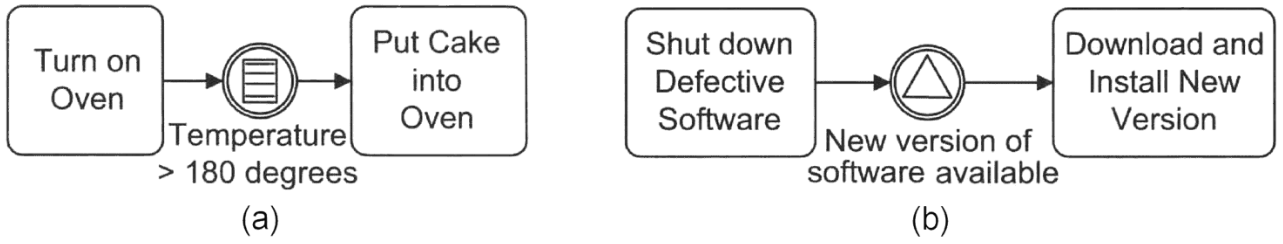
- If one of the checks has a negative result, the result of the other checks is irrelevant, because the application will be rejected anyway. Thus the process can be immediately terminated (a) (c).
- Replacing the terminate events by un-typed end event would cause a problem: if the result of one of the checks would be positive and the other negative, the token from the positive check would be stuck at the parallel gateway.

- The *Signal end event* (a) sends a signal, which can be received anywhere in the same process, as well as in other processes: “once a new software version has been released, different processes may react. Therefore, a signal is sent to everyone”.



Release New Software Version
New software version released
(a)
Terminate Project
Client notified and report sent to management board
(b)

- The *Multiple end event* (b) is used when the end of a process has several consequences. When it is reached, all the combined partial events occur: “one project has been terminated, several messages are sent”.
- In contrast to start and end events, which can only be catching and throwing, respectively, intermediate events can either be catching or throwing. (a) *conditional intermediate event*: “first the oven is turned on; the process then waits until the temperature exceeds 180 °C, before the cake is put into the oven”. (b) *signal intermediate event*: “defective sw application is shut down; then the process waits for the signal “new version of sw available”, before the next activity can be carried out.



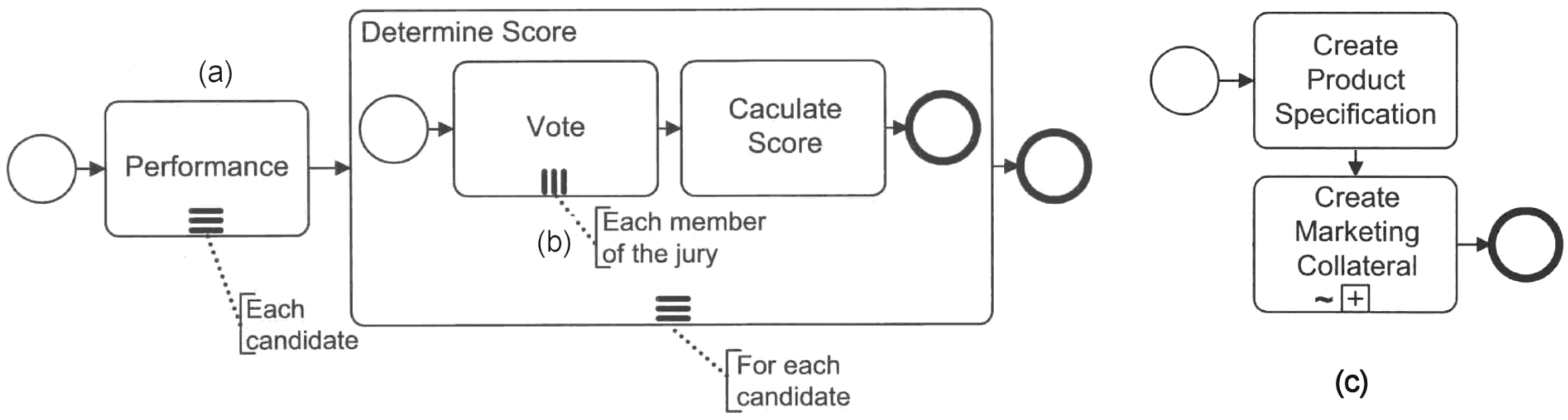
Turn on Oven
Temperature > 180 degrees
Put Cake into Oven
(a)
Shut down Defective Software
New version of software available
Download and Install New Version
(b)

About triggers

- Two aspects are considered in modeling events: the *trigger* (*cause*) of the event, and its effect in the process. Different strategies to forward the trigger to catching events: *publication*, *direct resolution*, *propagation*, *cancellations*, and *compensations*.
- Typically a *trigger* carries information out of the scope where the throw event occurred into the scope of the catching events. A scope is the context in which execution happens, consisting of the set of data objects available as well as events available for catching or throwing triggers.
- With *publication* a trigger may be received by catching events in any system scope. For instance: (i) messages are triggers generated outside of their pool; (ii) signals are triggers generated in their pool and broadcasted across processes, pools, diagrams.
- A trigger that is *propagated* is forwarded to the innermost enclosing scope instance with an attached event able to catch it. For instance: (i) *error* triggers are critical and suspend execution at the location of throwing; (ii) *escalations* are non-critical and execution continues at the location of throwing. If no catching event is found for an error or escalation trigger, this trigger is *unresolved*.
- *Compensation* of a successfully completed activity triggers directly its compensation handler. *Cancellation* terminates all running activities and compensates all successfully completed activities in the sub-process it is applied to. If the sub-process is a transaction, the transaction is rolled back.

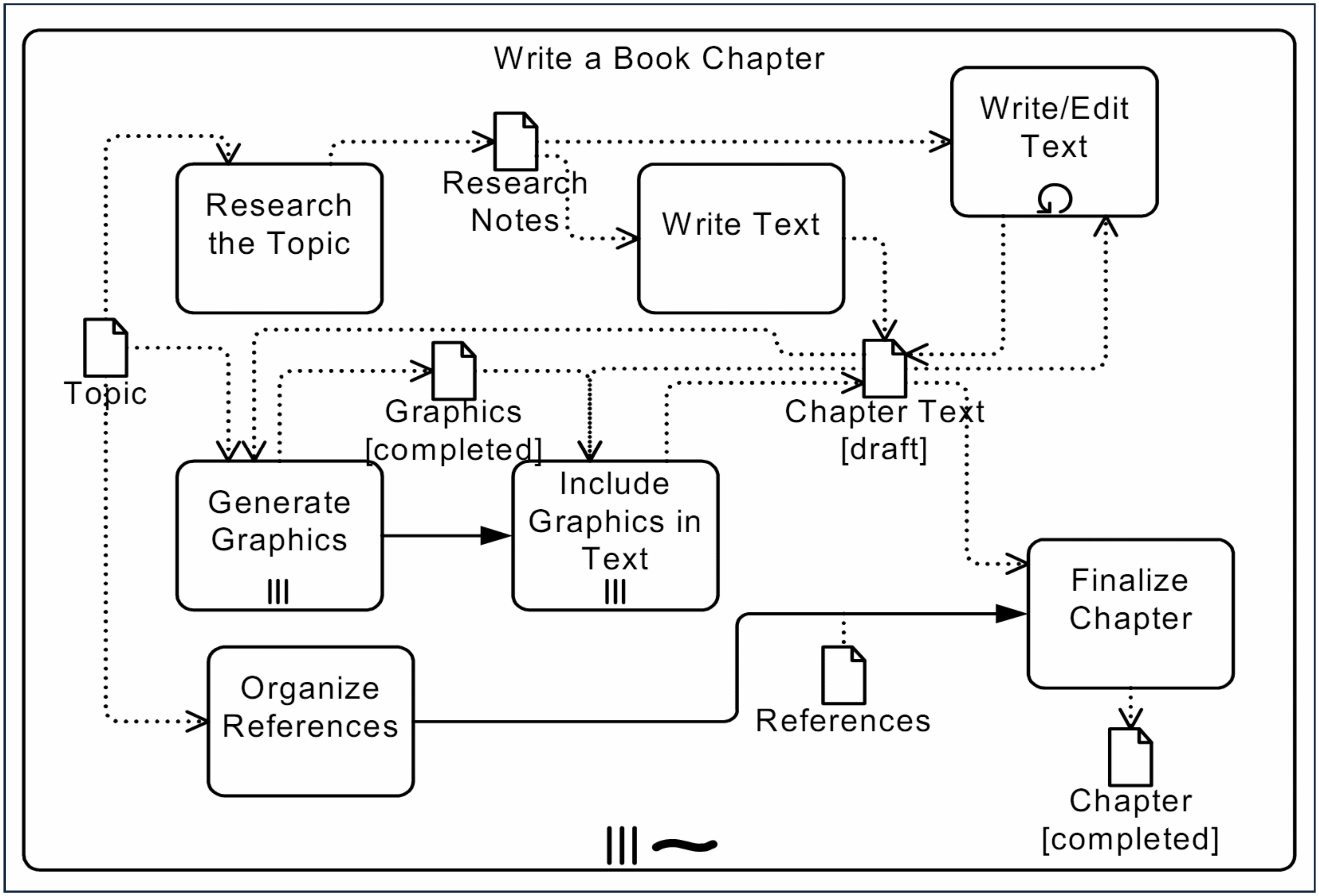
About activities

- *Multi-instance activities* are different from loop activities: the number of instances is known in advance (i.e., the size of the collection of objects), and the different objects can be processed in any *sequential* order (a), and also in *parallel* (b).
- In a process of a talent show in tv, first each candidate gives a performance; the group of candidates is known in advance. After the last performance, the determination of score is also sequential for each candidate. But the voting process is carried out in parallel by each member of the jury (b).



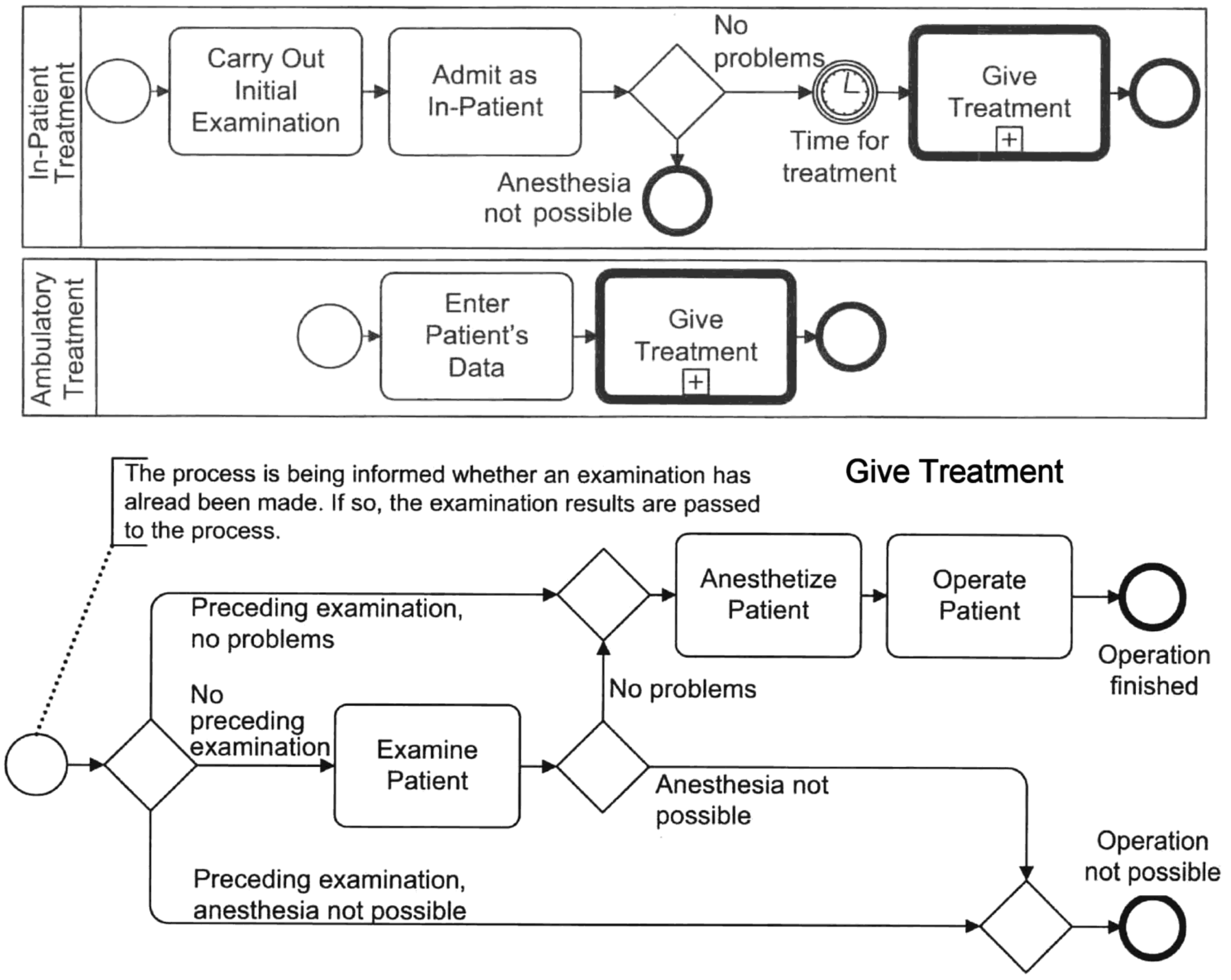
- In an *ad-hoc* (sub-)process (c), marked with a tilde, internal activities are determined, but their order is not known in advance. For instance, creative and knowledge-intensive processes, such as computer code development (at a low level), sales support, writing a book chapter.

- In writing a book chapter, there is no explicit process structure, but some sequence and data dependencies can be added. Data objects as input into a task act as an additional constraint, because the task cannot start without the appropriate input.



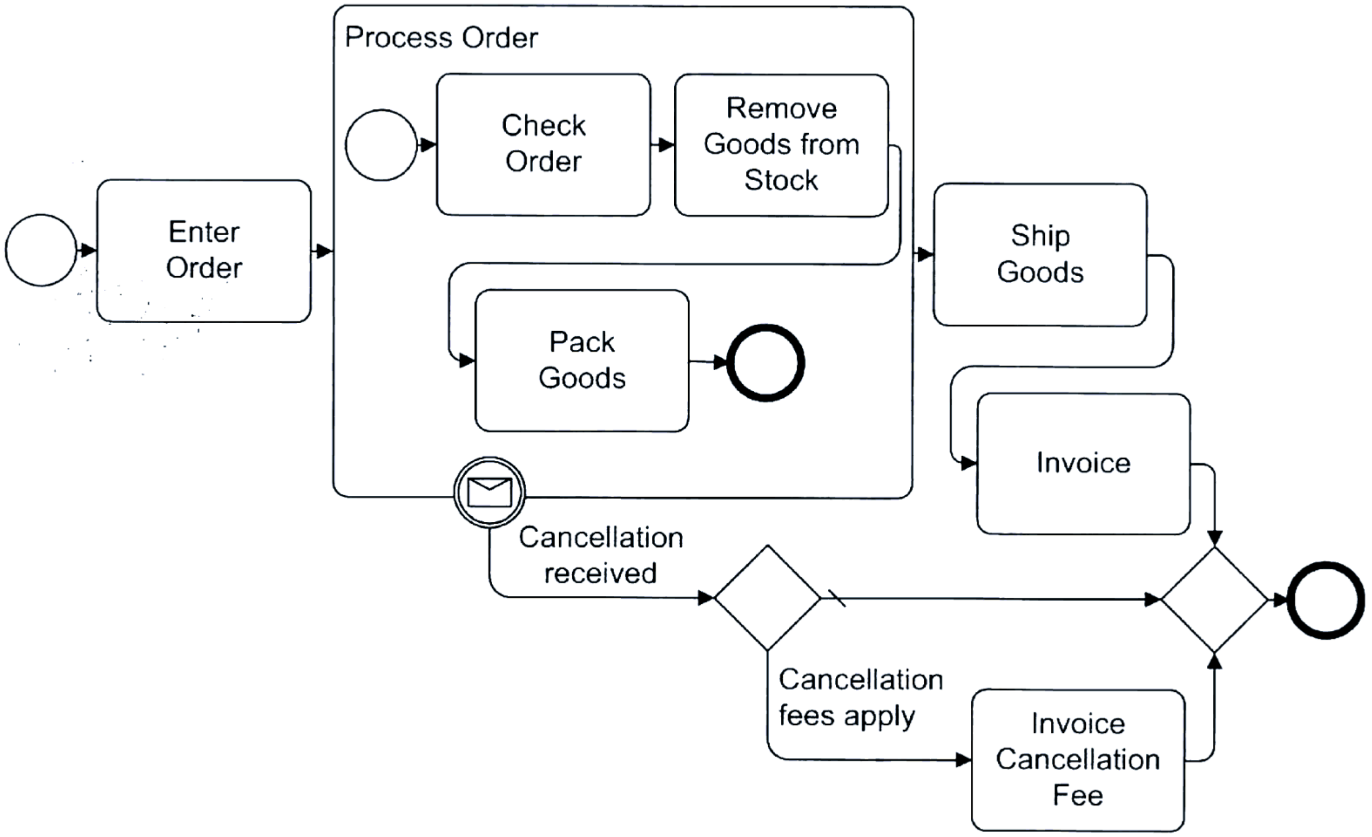
- Sub-processes and tasks are part of the process and cannot be used separately. A *call activity* (with thick border) can be reused in processes. Call activities cannot access the calling process, to be independent. Needed information must be provided.

- For example: the process for a treatment in hospital (in-patient) or ambulatory.



About exception handling

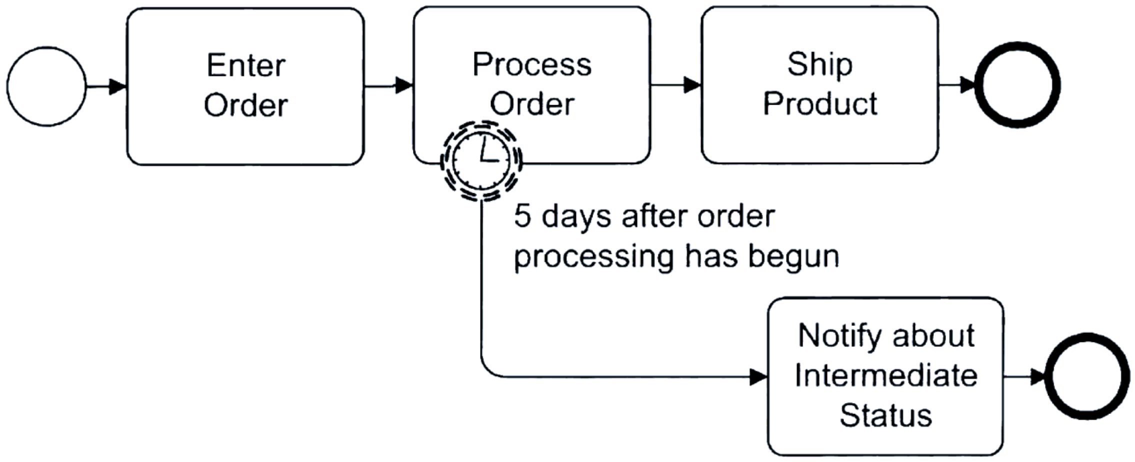
- While an activity / a process is carried out, sometimes an event occurs that results in an early abortion of the activity / process.
- The sub-process "Process Order" is aborted when a cancellation is received, i.e., all token will be removed from within that sub-process, and a token will be emitted via the exception flow at the interrupting event.



- If the order has been completely processed, no cancellations are possible anymore. If a cancellation arrives anyway, it will be ignored.

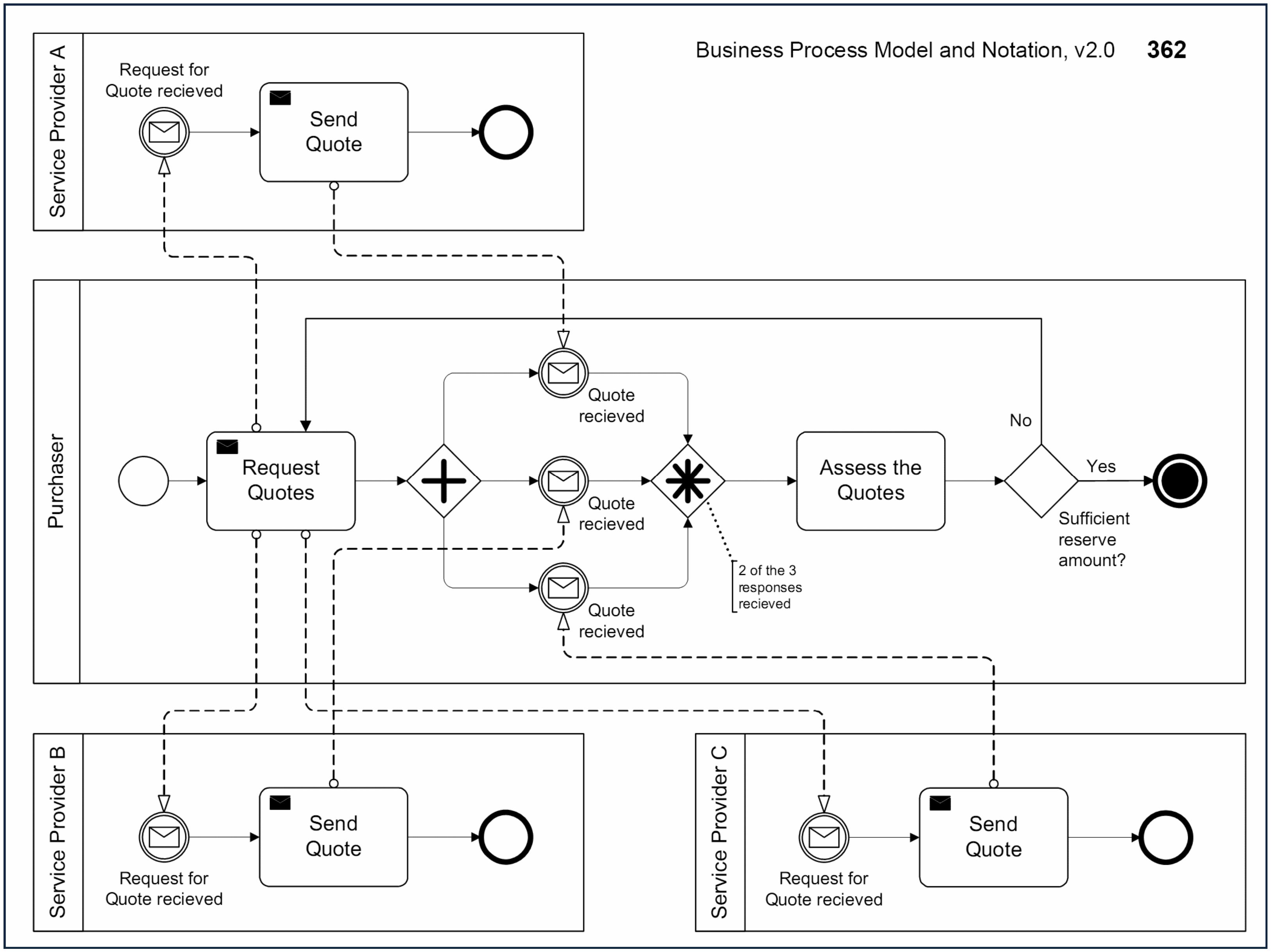
- Sometimes, it is desired to react to the event, but still to continue the ongoing activity. This can be done by using *non-interrupting* intermediate events, characterized by dashed borders.

- If the event "5 days after order processing has begun" is reached before the activity has been finished, a newly created token will be passed via the exception flow to the task "notify about intermediate status".


Enter Order
Process Order
Ship Product
5 days after order processing has begun
Notify about Intermediate Status

- At the same time, the original token still remains in the activity "Process Order", and is moved via the regular sequence flow.

- Exercise: describe in natural language the following process diagram, concerning an *e-tendering*.


Business Process Model and Notation, v2.0
362
Service Provider A
Request for Quote recieved
Send Quote
Purchaser
Request Quotes
Quote recieved
Quote recieved
Quote recieved
2 of the 3 responses recieved
Assess the Quotes
No
Yes
Sufficient reserve amount?
Service Provider B
Request for Quote recieved
Send Quote
Service Provider C
Request for Quote recieved
Send Quote

- Consider an e-tender that sends a request for quote to multiple service providers (e.g., warehouse storage) in a marketplace.
- The e-tender process sends out requests to each service provider and anticipates their response through three choreography activities.
- The response branches merge at a complex gateway to model the requirement that when 66% responses have arrived, an assessment of the tender can proceed.
- The assessment occurs after the complex gateway. If the assessment reports that the reserve amount indicated by the customer cannot be met, a new iteration of the tender is made.
- A key issue is to ensure that the responses should not be mixed across tender iterations. A terminate end event ensures that all activities are terminated, when a tender has been successful.