Business Processes Modelling

MPB (6 cfu, 295AA)

Roberto Bruni
http://www.di.unipi.it/~bruni

22 - Business process modelling notation
Object

We overview BPMN and their analysis based on Petri nets

Ch.4.7, 5.7 of Business Process Management: Concepts, Languages, Architectures
Ch.3, 4 of Fundamental of Business Process Management. M. Dumas et al.
Standardisation

The development of BMPN is an important step in reducing the fragmentation that existed with myriad of process modelling tools and notations exploiting experiences with many divergent proposals to consolidate the best ideas supporting the adoption of inter-operable business process management systems
The Business Process Management Initiative Initiative was an independent organization devoted to the development of open specifications for the management of e-Business processes that span multiple applications, corporate departments, and business partners, behind the firewall and over the Internet.
The membership of the BPMI Notation Working Group represents a large segment of the BP modelling community.
The Business Process Management Initiative (BPMI.org) and the Object Management Group™ (OMG™) decided to merge their activities on Business Process Management (BPM) to provide thought leadership and industry standards for this vital and growing industry.

The combined group has named itself the Business Modeling & Integration Domain Task Force (BMI -DTF)
Business process diagram

BPMN defines a standard for Business Process Diagrams (BPD) based on flowcharting technique tailored to graphical models of business process operations.

Four basic categories of elements:
- Swimlanes
- Flow objects
- Artefacts
- Connecting objects
Main goal:

provide a **notation** that is **readily understandable by all business users**

from the **business analysts** who create initial drafts of the processes

to the **technical developers** responsible for implementing the technology that will perform those processes

to the **business people** who will manage those processes
BPMN Versioning

BPMN 1.0 approved 2006
BPMN 1.1 approved 2007
BPMN 1.2 approved 2009

BPMN 2.0 Beta 1 proposed 2009
BPMN 2.0 Beta 2 proposed 2010
BPMN 2.0 Final delivered 2011
Disclaim

Formal rigor and conciseness are not primary concerns
(over 100 symbols, shorthands and alternative constructs are often available)

The large number of object types and their continuous evolution
makes it hard to define mappings and to prove their consistency under all contexts

Inconsistencies and ambiguities in BPMN standard are present but hard to detect
BPMN 2.0 vs 1.0

Updated (new markers):
Tasks/SubProcesses
Events
Gateways
Artefacts

Added:
Choreographies
Full metamodel
XML Serialization
Diagram Interchange
BPMN Execution Semantics (verbal)
Conversazioni
Una comunicazione definisce un insieme di scambi di messaggi collegati logicamente. Se annotati con un simbolo, indica una conversazione interna ad un’altra conversazione.

Un forked conversation link connette le comunicazioni ed i partecipanti.

Diagramma di coreografie

Coreografie
Un Task di coreografia rappresenta un’interazione/cambio di messaggi tra due o più partecipanti.

Il simbolo MultiPort Participants denota un insieme di partecipanti della stessa tipologia.

Diagramma di conversazione

Classifikasi

Pool (compreso)

Con Attività

Un task è una unità di lavoro, cioè il lavoro da eseguire. Quando si annetta con il simbolo di un task, si indica un insieme di messaggi che possono essere eseguiti in parallelo.

Un forked conversation link connette le comunicazioni ed i partecipanti.

Diagramma di coreografie

Conversazioni
Una comunicazione definisce un insieme di scambi di messaggi collegati logicamente. Se annotati con un simbolo, indica una conversazione interna ad un’altra conversazione.

Un forked conversation link connette le comunicazioni ed i partecipanti.

Diagramma di conversazione

Coreografie
Un Task di coreografia rappresenta un’interazione/cambio di messaggi tra due o più partecipanti.

Il simbolo MultiPort Participants denota un insieme di partecipanti della stessa tipologia.

Diagramma di coreografie

Simboli per attività
I seguenti simboli indicano il comportamento di esecuzione delle attività:

- Sottoprocesso: un'attività in parallelo
- Loop: ricerca di un modello in un insieme di attività
- Esecuzione sequenziale: sequenza di attività
- Ad hoc: regole di business
- Service: esegue un servizio
- Script: esegue un script

Tipologie di task
Le tipologie specificano la natura dell’azione da eseguire:

- Task di invio: l’attività è inviata
- Task di ricezione: l’attività è attesa
- Utente: l’attività è eseguita da un modello di business
- Task manuale: l’attività è eseguita da un modello di business
- Regole di business: l’attività è eseguita da un modello di business
- Service: l’attività è eseguita da un modello di business
- Script: l’attività è eseguita da un modello di business

Flusso sequenziale/parallelo
Il flusso sequenziale definisce l’ordine di esecuzione delle attività.

- Flusso sequenziale: un’attività successiva eseguita in sequenza
- Flusso parallelo: un’attività successiva eseguita in parallelo

Flussi di messaggi
Flussi di messaggi rappresentano il flusso di interazione di messaggi. Un flusso di messaggi può essere unito ad altri flussi di messaggi.

Swimlanes
Swimlanes (partecipanti) e Laneconnectors rappresentano la responsabilità per le attività in un processo. Esse possono essere un’organizzazione, un ruolo o un sistema. Le lane subdividono i pool o altre coreografie.

Data
Un Data Input è un input esterno esterno all’interno del processo. Può essere letto da un’attività.

Un Data Output è un elemento variabile che viene trasmesso da un’attività.

Un Collection Data Object rappresenta una collezione di informazioni, come ad esempio una lista di elementi ordinati.

Un Data Store è un luogo dove il processo può leggere oppure scrivere dati, ad esempio un database. Esso si mantiene oltre la durata dell’istanza del processo.

Un messaggio è usato per rappresentare i contenuti di una comunicazione tra due partecipanti.
BPMN 2.0 (2009/11)
FAQ

What is BPMN?

BPMN is a graphical notation that depicts the steps (end to end flow) in a business process.

The notation has been specifically designed to coordinate the sequence of processes and the messages that flow between different participants in a related set of activities.
Why is BPMN important?

The world of business processes has changed dramatically over the past few years. Processes can be coordinated from behind, within and over organizations boundaries. A business process now spans multiple participants and coordination can be complex.

Until BPMN, there has not been a standard modelling technique developed that addresses these issues. BPMN provides users with a royalty free notation.

This will benefit users in a similar manner in which UML standardised the world of software engineering. There will be training courses, books and a body of knowledge that users can access in order to better implement a business process.
BPMN is targeted at users, vendors and service providers that need to communicate business processes in a standard manner.
BPMN 2.0 (2009/11)

FAQ

Will there be a major rewrite?

Not for 2 or 3 years…

(2018 and still no revision is planned)
**Strong points of BPMN**

**Simplicity**: A small set of basic symbols

**Extensibility**: many decorations available (new ones can be added in the future)

Graphical design: **intuitive**

**Generality**: orchestration + choreography

Tool availability: **exchange format**
Weaknesses of BPMN

over 100 graphical elements

verbose description (500 pages)

difficult to learn comprehensively:
different reading of the same diagram are possible

different BPMN vendors implement the execution of BPMN diagrams in different ways (and for different subsets)
BPMN basics:
Swimlanes
(pools, lanes)
Swimlanes

Many process modelling methodologies utilise the concept of a **swimlane** as a mechanism to **organise activities into separate visual categories** in order to illustrate different capabilities or responsibilities.

BPMN supports two main swimlane objects:
- **Pool**
- **Lane**
Pools and Lanes

A pool represents a participant (or role) in a process
A pool is represented as a rectangle with a name

A lane is a hierarchical sub-partition within a pool
that is used to organise and categorise activities

A lane is an inner rectangle to the pool
that extends to the entire length of the pool
Swimlanes

Pools and lanes are used to represent organizations, roles, systems and responsibilities. Examples: 'University', 'Sales division', 'Warehouse', 'ERP system', ...

- A Pool MUST contain 0 or 1 business process.
- A Pool can contain 0 or more lanes.
- Two pools can only be connected with message flows.

A Pool represents a participant in a process. It contains a business process and is used in B2B situations.

A Lane is a sub-partition within a pool used to organize and categorize activities.

- Pools and Lanes represent responsibilities for activities in a process. A pool or a lane can be an organization, a role, or a system. Lanes sub-divide pools or other lanes hierarchically.

- Collapsed Pools hide all internals of the contained processes.
Naming conventions

Process models:
a noun possibly preceded by an adjective

the label is often obtained by `nominalizing` the verb
that describe the main action in the process
(e.g., claim handling, order fulfillment)

Avoid long labels
Articles are often omitted
BPMN basics: Flow Objects (events, activities, gateways)
Flow objects

Theory:
fix a small set of core elements
so that modellers do not have to learn and recognise
a large number of different shapes:
Events
Activities
Gateways

Practice:
use different border styles and internal markers
to add many more information
(this way the notation is extensible)
Events

An event is something that “happens” during the course of a business process.

The type of an event is one among: start, intermediate, end

An event is represented as a circle its type depends on the style of the border (thin, double, thick)

An event can have a cause (trigger) or an impact (result)
Internal markers denote the trigger or result
Naming conventions

Events:
the label should begin with a noun and end with a verb in past participle form to indicate something that just happened (e.g., Invoice emitted)

the noun can be preceded by an adjective (e.g., Urgent order sent)

Avoid long labels
Articles are often omitted
<table>
<thead>
<tr>
<th></th>
<th>Start</th>
<th>Intermediate</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Catching</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plain</td>
<td><img src="image" alt="Circle" /></td>
<td><img src="image" alt="Circle" /></td>
<td><img src="image" alt="Circle" /></td>
</tr>
<tr>
<td>Message</td>
<td><img src="image" alt="Envelope" /></td>
<td><img src="image" alt="Envelope" /></td>
<td><img src="image" alt="Envelope" /></td>
</tr>
<tr>
<td>Timer</td>
<td><img src="image" alt="Clock" /></td>
<td><img src="image" alt="Clock" /></td>
<td><img src="image" alt="Clock" /></td>
</tr>
<tr>
<td>Error</td>
<td><img src="image" alt="Wave" /></td>
<td><img src="image" alt="Wave" /></td>
<td><img src="image" alt="Wave" /></td>
</tr>
<tr>
<td>Link</td>
<td><img src="image" alt="Link" /></td>
<td><img src="image" alt="Link" /></td>
<td><img src="image" alt="Link" /></td>
</tr>
<tr>
<td>Terminate</td>
<td></td>
<td></td>
<td><img src="image" alt="Circle" /></td>
</tr>
</tbody>
</table>

**Catching**

- **Start Event**: Catching an event starts a new process instance.
- **Intermediate Event (catching)**: The process can only continue once an event has been caught.

**Throwing**

- **End Event**: An event is thrown when the end of the process is reached.
- **Intermediate Event (throwing)**: An event is thrown and the process continues.
Activities

An activity is some “unit of work” (job) to be done during the course of a business process.

An activity can be atomic (task) or compound (sub-process).

An activity is represented as a rounded box, suitable markers are used to indicate the nature of the action to be performed (task type) and the execution behaviour (activity marker).
Sub-processes

Process models tend to be too large to be understood at once.

Hiding certain parts within sub-processes we improve readability.

A sub-process is a self-contained, composite activity that can be broken into smaller units of work.
A **Task** is a unit of work, the job to be performed.

A **Subprocess** is a decomposable activity. It can be collapsed to hide the details.

An **Expanded Subprocess** contains a valid BPMN diagram.
Activity types and markers

Activity Markers
Markers indicate execution behavior of activities:

- Sub-Process Marker
- Loop Marker
- Parallel MI Marker
- Sequential MI Marker
- Ad Hoc Marker
- Compensation Marker

Task Types
Types specify the nature of the action to be performed:

- Send Task
- Receive Task
- User Task
- Manual Task
- Business Rule Task
- Service Task
- Script Task
Naming conventions

Activities:
verb in the imperative form followed by a noun
(e.g., Approve order)

the noun can be preceded by an adjective
(e.g., Issue driver license)

the verb may be followed by a complement
(e.g., Renew driver license via offline agencies)

Avoid long labels
Articles are often omitted
Events vs Activities

Events are instantaneous

Activities take time (have a duration)
Multiple Instances of the same activity are started in parallel or sequentially, e.g. for each line item in an order.

Loop Activity is iterated if a loop condition is true. The condition is either tested before or after the activity execution.

Ad-hoc Subprocesses contain tasks only. Each task can be executed arbitrarily often until a completion condition is fulfilled.

Attached Intermediate Event: The activity is aborted once an event is caught.

A Call Activity is a wrapper for a globally defined Sub-Process or Task that is reused in the current process.
Gateways

A *gateway* is used to control the splitting and joining of paths in the sequence flow (conditional, fork, wait)

A gateway is represented as a diamond shape. Suitable markers are used to indicate the nature of behaviour control.
Gateway markers

Data-based Exclusive Gateway
When splitting, it routes the sequence flow to exactly one of the outgoing branches based on conditions. When merging, it awaits one incoming branch to complete before triggering the outgoing flow.

Event-based Exclusive Gateway
Is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.

Parallel Gateway
When used to split the sequence flow, all outgoing branches are activated simultaneously. When merging parallel branches it waits for all incoming branches to complete before triggering the outgoing flow.

Inclusive Gateway
When splitting, one or more branches are activated based on branching conditions. When merging, it awaits all active incoming branches to complete.

Complex Gateway
It triggers one or more branches based on complex conditions or verbal descriptions. Use it sparingly as the semantics might not be clear.
BPMN basics:
Artefacts
(data-objects, groups, text annotations)
Artefacts

BPMN is designed to allow modellers and modelling tools some flexibility in extending the basic notation.

Any number of artefacts can be added to a diagram as appropriate for the specific context of the business process being modelled.

BPMN includes three pre-defined types of artefacts:

- Data object
- Group
- Text annotation
Data object

A **data object** specifies the data that are required or produced by an activity.

A data object is often represented by the usual file icon.

A **Data Input** is an external input for the entire process. It can be read by an activity.

A **Data Output** is a variable available as result of the entire process.

A **Data Object** represents information flowing through the process, such as business documents, e-mails, or letters.

A **Collection Data Object** represents a collection of information, e.g., a list of order items.

A **Data Store** is a place where the process can read or write data, e.g., a database or a filing cabinet. It persists beyond the lifetime of the process instance.

A **Message** is used to depict the contents of a communication between two Participants.
Group

An arbitrary set of objects can be defined as a group to show that they logically belong together.

A group is represented by rounded corner rectangles with dashed lines.

A group is represented by rounded corner rectangles with dashed lines.
Annotation

Any object can be associated with a **text annotation** to provide any additional information and documentation that can be needed.

A text annotation is represented as a dotted-line call-out.

Text Annotation Allows a Modeler to provide additional information.
Artefacts
Artefacts are used to provide additional information about the process. If required, modellers and modelling tools are free to add new artefacts. Examples of data objects: 'A letter', 'Email message', 'XML document', 'Confirmation',...

<table>
<thead>
<tr>
<th>Data object</th>
<th>Group</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[state]</td>
<td>Grouping can be used for documentation or analysis purposes. Groups can also be used to identify the activities of a distributed transaction that is shown across Pools. Grouping of activities does not affect the Sequence or Message Flow.</td>
<td>Text Annotations are a mechanism for a modeller to provide additional information for the reader of a BPMN Diagram.</td>
</tr>
</tbody>
</table>

Data objects provide information about what activities are required to be triggered and/or what they produce. They are considered as Artefacts because they do not have any direct effect on the Sequence Flow or Message Flow of the Process. The state of the data object should also be set.
BPMN basics: Connecting objects
(sequence flow, message flow, association)
Connecting objects

The Flow objects are connected together in a diagram to create the basic skeletal structure of a business process.

Three connecting objects can be used:

- Sequence flow
- Message flow
- Association
Sequence flow

A sequence flow is used to show the order in which activities are to be performed.

Note: connected objects must reside in the same pool (but they can be in different lanes). The term “control flow” is generally avoided in BPMN.

A sequence flow is represented by a solid line with a solid arrowhead.
**Sequence Flow** defines the execution order of activities.

**Conditional Flow** has a condition assigned that defines whether or not the flow is used.

**Default Flow** is the default branch to be chosen if all other conditions evaluate to false.

read as ```otherwise```
Message flow

A message flow is used to show the flow of messages between two separate process participants (business entities or business roles) that send and receive them.

Note: the participants reside in separate pools.

```
 o---------▷
```

A message flow is represented by a dashed line with a open arrowheads (see above).
Sequence Flow and Message Flow rules
Only objects that can have an incoming and/or outgoing Sequence Flow / Message Flow are shown in the Tables Below.

<table>
<thead>
<tr>
<th>From:</th>
<th>To:</th>
</tr>
</thead>
<tbody>
<tr>
<td>📝</td>
<td>📝</td>
</tr>
<tr>
<td>📝</td>
<td>📝</td>
</tr>
<tr>
<td>📝</td>
<td>📝</td>
</tr>
<tr>
<td>📝</td>
<td>📝</td>
</tr>
<tr>
<td>📝</td>
<td>📝</td>
</tr>
</tbody>
</table>

**Message Flow** symbolizes information flow across organizational boundaries. Message flow can be attached to pools, activities, or message events.

The order of message exchanges can be specified by combining message flow and sequence flow.
An **association** is used to associate data, text, and other artefacts with flow objects.

Note: in particular, input and output of activities.

An association is represented by a dotted line with a line arrowhead.
A **Data Object** represents information flowing through the process, such as business documents, e-mails or letters.

Attaching a data object with an **Undirected Association** to a sequence flow indicates hand-over of information between the activities involved.

A **Directed Association** indicates information flow. A data object can be read at the start of an activity or written upon completion.

A **Bidirected Association** indicates that the data object is modified, i.e. read and written during the execution of an activity.
Graphical connecting objects

There are three ways of connecting **Flow objects (Events, Activities, Gateways)** with each other or with other information – using sequence flows, message flows or associations.

<table>
<thead>
<tr>
<th>Graphical connecting objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal sequence flow</td>
<td>A Sequence Flow is used to show the order in which the activities in a process will be performed.</td>
</tr>
<tr>
<td>Conditional sequence flow</td>
<td>A Sequence Flow can have condition expressions which are evaluated at runtime to determine whether or not the flow will be used.</td>
</tr>
<tr>
<td>Default sequence flow</td>
<td>For Data-Based Exclusive Decisions or Inclusive Decisions, one type of flow is the Default condition flow. This flow will be used only if all other outgoing conditional flows are NOT true at runtime.</td>
</tr>
<tr>
<td>Message flow</td>
<td>A Message Flow is used to show the flow of messages between two participants that are prepared to send and receive them. In BPMN, two separate Pools in a Diagram can represent the two participants.</td>
</tr>
<tr>
<td>Association</td>
<td>An Association (directed, non-directed) is used to associate information with Flow Objects. Text and graphical non-Flow Objects can be associated with Flow objects.</td>
</tr>
</tbody>
</table>
A few patterns
In this chapter we will become familiar with the core set of symbols provided by BPMN. As stated earlier, a business process involves events and activities. Events represent things that happen instantaneously (e.g. an invoice has been received) whereas activities represent units of work that have a duration (e.g. an activity to pay an invoice). Also, we recall that in a process, events and activities are logically related. The most elementary form of relation is that of sequence, which implies that one event or activity A is followed by another event or activity B. Accordingly, the three most basic concepts of BPMN are event, activity, and arc. Events are represented by circles, activities by rounded rectangles, and arcs (called sequence flows in BPMN) are represented by arrows with a full arrow-head.

Example 3.1
Figure 3.1 shows a simple sequence of activities modeling an order fulfillment process in BPMN. This process starts whenever a purchase order has been received from a customer. The first activity that is carried out is confirming the order. Next, the shipment address is received so that the product can be shipped to the customer. Afterwards, the invoice is emitted and once the payment is received the order is archived, thus completing the process.

From the example above we notice that the two events are depicted with two slightly different symbols. We use circles with a thin border to capture start events and circles with a thick border to capture end events. Start and end events have an important role in a process model: the start event indicates when instances of the process start whereas the end event indicates when instances complete. For example, a new instance of the order fulfillment process is triggered whenever a purchase order is received, and completes when the order is fulfilled. Let us imagine that the order fulfillment process is carried out at a seller’s organization. Every day this organization will run a number of instances of this process, each instance being independent of the others. Once a process instance has been spawned, we use the notion of token to identify the progress (or state) of the instance. Tokens are recreated in a start event, flow throughout the process model until they are destroyed in an end event. We depict tokens as colored dots on top of a process model. For example Fig. 3.2 shows the state of three instances of the order fulfillment process: one instance has just started (black token on the start event), another is shipping the product (red token on activity “Ship product”), and the third one has received the payment and is about to start archiving the order (green token in the sequence flow between “Receive payment” and “Archive order”).

While it comes natural to give a name (also called label) to each activity, we should not forget to give labels to events as well. For example, giving a name to each start event allows us to communicate what triggers an instance of the process.
Exclusive decisions: invoice checking process

Example 3.2:
Invoice checking process. As soon as an invoice is received from a customer, it needs to be checked for mismatches. The check may result in either of these three options: i) there are no mismatches, in which case the invoice is posted; ii) there are mismatches but these can be corrected, in which case the invoice is re-sent to the customer; and iii) there are mismatches but these cannot be corrected, in which case the invoice is blocked. Once one of these three activities is performed the invoice is parked and the process completes.

To model this process we start with a decision activity, namely “Check invoice for mismatches” following a start event “Invoice received”. A decision activity is an activity that leads to different outcomes. In our example, this activity results in three possible outcomes, which are mutually exclusive; so we need to use an XOR-split after this activity to fork the flow into three branches. Accordingly, three sequence flows will emanate from this gateway, one towards activity “Post invoice”, performed if there are no mismatches, another one towards “Re-send invoice to customer”, performed if mismatches exist but can be corrected, and a third flow towards “Block invoice”, performed if mismatches exist which cannot be corrected (see Fig. 3.4).

From a token perspective, an XOR-split routes the token coming from its incoming branch towards one of its outgoing branches, i.e. only one outgoing branch can be taken. When using an XOR-split, make sure each outgoing sequence flow is annotated with a label capturing the condition upon which that specific branch is taken. Moreover, always use mutually exclusive conditions, i.e. only one of them can be true every time the XOR-split is reached by a token. This is the characteristic of the XOR-split gateway. In our example an invoice can either be correct, or contain mismatches that can be fixed, or mismatches that cannot be fixed: only one of these conditions is true per invoice received.

It is important to annotate branches with the conditions under which they are taken.
Parallel activities: airport security check

Fig. 3.5. An example of the use of AND gateways undergoing the required security checks. After the first activity, and before the last one, we need to perform two activities which can be executed in any order, i.e. which do not depend on each other: "Pass personal security screening" and "Pass luggage screening". To model this situation we use an AND-split linking activity "Proceed to security check" with the two screening activities, and an AND-join linking the two screening activities with activity "Proceed to departure level".

The AND-split splits the token coming from activity "Proceed to security check" into two tokens. Each of these tokens independently flows through one of the two branches. This means that when we reach an AND-split, we take all outgoing branches (note that an AND-split may have multiple outgoing arcs). As we said before, a token is used to indicate the state of a given instance. When multiple tokens of the same color are distributed across a process model, e.g. as a result of executing an AND-split, they collectively represent the state of an instance. For example, if a token is on the arc emitting from activity "Pass luggage screening" and another token of the same color is on the arc incident to activity "Pass personal security screening", this indicates an instance of the security check process where a passenger has just passed the luggage screening.

The AND-join of our example waits for a token to arrive from each of the two incoming arcs, and once they are all available, it merges the tokens back into one. The single token is then sent to activity "Proceed to departure level". This means that we proceed when all incoming branches have completed (note again that an AND-join may have multiple incoming arcs). This behavior of waiting for a number of tokens to arrive and then merging the tokens into one is called synchronization.

Example 3.4: Let us extend the order fulfillment example of Fig. 3.1 by assuming that a purchase order is only confirmed if the product is in stock, otherwise the process completes by rejecting the order. Further, if the order is confirmed, the shipment address is received and the requested product is shipped while the invoice is emitted and the payment is received. Afterwards, the order is archived and the process completes.

The resulting model is shown in Fig. 3.6.
Multiple end events are often considered as a convenient notation (they are mutually exclusive in the example)

BPMN adopts implicit termination semantics: a case ends only when each "token" reaches the end
Multiple start events are often considered as a convenient notation (they capture mutually exclusive triggers to start a process instance)
Omitting gateways

An **AND-gateway** can be omitted when it **follows** an activity or event.

Similarly, a **XOR-gateway before** an activity or event can be omitted.
Inclusive decisions
(one, many)
Inclusive decisions: order distribution

Only XOR / AND gateways, but the diagram is convoluted! What if we had three or more warehouses? (does not scale)
Inclusive decisions: order distribution

Only XOR / AND gateways, the diagram can "scale", but is it correct? (also the case no-warehouse is now possible)
Inclusive decisions: order distribution

What is the problem with this second solution? The example scenario allows three cases: the products are in Amsterdam only, in Hamburg only, or in both warehouses, while this solution allows one more case, i.e. when the products are in neither of the warehouses. This case occurs when the two empty branches of the two XOR-splits are taken and results in doing nothing between activity "Check order line items" and activity "Register order". Thus this solution, despite being more compact than the first one, is wrong.

To model situations where a decision may lead to one or more options being taken at the same time, we need to use an inclusive (OR) split gateway. A an OR-split is similar to the XOR-split, but the conditions on its outgoing branches do not need to be mutually exclusive, i.e. more than one of them can be true at the same time. When we encounter an OR-split, we thus take one or more branches depending on which conditions are true. In terms of token semantics, this means that the OR-split takes the input token and generates a number of tokens equivalent to the number of output conditions that are true, where this number can be at least one and at most as the total number of outgoing branches. Similar to the XOR-split gateway, an OR-split can also be equipped with a default flow, which is taken only when all other conditions evaluate to false.

Figure 3.10 shows the solution to our example using the OR gateway. After the sub-order has been forwarded to either of the two warehouses or to both, we use an OR-join to synchronize the flow and continue with the registration of the order. An OR-join proceeds when all active incoming branches have completed. Waiting for an active branch means waiting for an incoming branch that will ultimately de-
XOR + AND + OR: order fulfillment

Fig. 3.12 The order fulfillment process diagram with product manufacturing

3.2 Branching and Merging

3.2.4 Rework and Repetition

So far we have seen structures that are linear, i.e. each activity is performed at most once. However, sometimes we may require to repeat one or several activities, for instance because of a failed check.

Example 3.7

In the treasury minister's office, once a ministerial inquiry has been received, it is first registered into the system. Then the inquiry is investigated so that a ministerial response can be prepared. The finalization of a response includes the preparation of the response itself by the cabinet officer and the review of the response by the principal registrar. If the registrar does not approve the response, the latter needs to be prepared again by the cabinet officer for review. The process finishes only once the response has been approved.

To model rework or repetition we first need to identify the activities, or more in general the fragment of the process, that can be repeated. In our example this consists of the sequence of activities "Prepare ministerial response" and "Review ministerial response". Let us call this our repetition block. The property of the repetition block is that the last of its activities must be a decision activity. In fact, this will allow us to decide whether to go back before the repetition block starts, or to continue with the rest of the process. As such, this decision activity should have two outcomes. In our example the decision activity is "Review ministerial response" and its outcomes are: "response approved" (in this case we continue with the process) and "response not approved" (we go back). To model these two outcomes, we use an XOR-split with two outgoing branches: one which allows us to continue with the rest of the process (in our example, this is simply

Better if gateways are balanced
XOR + AND + OR: order fulfillment

3.2 Branching and Merging

Fig. 3.12 The order fulfillment process diagram with product manufacturing

3.2.4 Rework and Repetition

So far we have seen structures that are linear, i.e., each activity is performed at most once. However, sometimes we may require to repeat one or several activities, for instance because of a failed check.

Example 3.7

In the treasury minister’s office, once a ministerial inquiry has been received, it is first registered into the system. Then the inquiry is investigated so that a ministerial response can be prepared. The finalization of a response includes the preparation of the response itself by the cabinet officer and the review of the response by the principal registrar. If the registrar does not approve the response, the latter needs to be prepared again by the cabinet officer for review. The process finishes only once the response has been approved.

To model rework or repetition we first need to identify the activities, or more in general the fragment of the process, that can be repeated. In our example this consists of the sequence of activities “Prepare ministerial response” and “Review ministerial response”. Let us call this our repetition block. The property for repetition block is that the last of its activities must be a decision activity. In fact, this will allow us to decide whether to go back before the repetition block starts, or to continue with the rest of the process. As such, this decision activity should have two outcomes. In our example the decision activity is “Review ministerial response” and its outcomes are: “response approved” (in this case we continue with the process) and “response not approved” (we go back). To model these two outcomes, we use an XOR-split with two outgoing branches: one which allows us to continue with the rest of the process (in our example, this is simply)

Better if gateways are balanced
XOR + AND + OR: order fulfillment

Better if gateways are balanced
Rework and repetition: ministerial correspondence

A repetition block starts with a XOR-join and ends with a decision gateway (XOR-split)
80 3 Essential Process Modeling

Fig. 3.14 The order fulfillment example with artifacts from activity "Obtain raw materials from Supplier 1" to Raw materials, indicates that Raw materials is an output object for this activity. To avoid cluttering the diagram with data associations that cross sequence flows, we may repeat a data object multiple times within the same process model. However, all occurrences of a given object do conceptually refer to the same artifact. For example, in Fig. 3.14 Purchase order is repeated twice as input to "Check stock availability" and to "Confirm order" since these two activities are far away from each other in terms of model layout.

Often the output from an activity coincides with the input to a subsequent activity. For example, once Raw materials have been obtained, these are used by activity "Manufacture product" to create a Product. The Product in turn is packaged and sent to the customer by activity "Ship product". Effectively, data objects allow us to model the information flow between process activities. Bear in mind, however, that data objects and their associations with activities cannot replace the sequence flow. In other words, even if an object is passed from an activity A to an activity B, we still need to model the sequence flow from A to B. A shorthand notation for passing an object from an activity to the other is by directly connecting the data object to the sequence flow between two consecutive activities via an undirected association. See for example the Shipment address being passed from activity "Get shipment address".

Information artifacts: order fulfillment

artifacts provide additional information, at the price of increased complexity
Information artifacts: order fulfillment

Fig. 3.14 The order fulfillment example with artifacts from activity "Obtain raw materials from Supplier 1" to Raw materials, indicates that Raw materials is an output object for this activity. To avoid cluttering the diagram with data associations that cross sequence flows, we may repeat a data object multiple times within the same process model. However, all occurrences of a given object do conceptually refer to the same artifact. For example, in Fig. 3.14, Purchase order is repeated twice as input to "Check stock availability" and to "Confirm order" since these two activities are far away from each other in terms of model layout.

Often the output from an activity coincides with the input to a subsequent activity. For example, once Raw materials have been obtained, these are used by activity "Manufacture product" to create a Product. The Product in turn is packaged and sent to the customer by activity "Ship product". Effectively, data objects allow us to model the information flow between process activities. Bear in mind, however, that data objects and their associations with activities cannot replace the sequence flow. In other words, even if an object is passed from an activity A to an activity B, we still need to model the sequence flow from A to B. A shorthand notation for passing an object from an activity to the other is by directly connecting the data object to the sequence flow between two consecutive activities via an undirected association. See for example the Shipment address being passed from activity "Get products from warehouse" to "Ship product".
Information artifacts: order fulfillment

80 3 Essential Process Modeling

Fig. 3.14

The order fulfillment example with artifacts from activity "Obtain raw materials from Supplier 1" to Raw materials, indicates that Raw materials is an output object for this activity. To avoid cluttering the diagram with data associations that cross sequence flows, we may repeat a data object multiple times within the same process model. However, all occurrences of a given object do conceptually refer to the same artifact. For example, in Fig. 3.14 Purchase order is repeated twice as input to "Check stock availability" and to "Confirm order" since these two activities are far away from each other in terms of model layout.

Often the output from an activity coincides with the input to a subsequent activity. For example, once Raw materials have been obtained, these are used by activity "Manufacture product" to create a Product. The Product in turn is packaged and sent to the customer by activity "Ship product". Effectively, data objects allow us to model the information flow between process activities. Bear in mind, however, that data objects and their associations with activities cannot replace the sequence flow. In other words, even if an object is passed from an activity A to an activity B, we still need to model the sequence flow from A to B. A shorthand notation for passing an object from an activity to the other is by directly connecting the data object to the sequence flow between two consecutive activities via an undirected association. See for example the Shipment address being passed from activity "Get product from Warehouse" to "Ship product".

Information artifacts:

- order fulfillment
- data stores
- (for persistent data objects)
Information artifacts: order fulfillment

Data objects (for convenience, the same object can be repeated several times)

State of the object
Resources as lanes: order fulfillment

Fig. 3.15 The order fulfillment example with resource information
Placing items

**events**: must be placed in the proper lane

**activities**: must be placed in the proper lane

**data-objects**: placement is irrelevant

**gateways**:

**(X)OR-splits**: same lane as preceding decision activity

**AND-split, joins**: placement is irrelevant
Some remarks

Lanes are often used to separate activities associated with a specific company function or role.

Sequence flow may cross the boundaries of Lanes within the same Pool.

Message flow may not be used between Flow objects in Lanes of the same Pool.
Connecting Objects

The Flow Objects are connected together in a diagram to create the basic skeletal structure of a business process. There are three Connecting Objects that provide this function. These connectors are:

1. **Sequence Flow**
   - A Sequence Flow is represented by a solid line with a solid arrowhead (see the figure to the right) and is used to show the order (the sequence) that activities will be performed in a Process. Note that the term “control flow” is generally not used in BPMN.

2. **Message Flow**
   - A Message Flow is represented by a dashed line with an open arrowhead (see the figure to the right) and is used to show the flow of messages between two separate Process Participants (business entities or business roles) that send and receive them. In BPMN, two separate Pools in the Diagram will represent the two Participants.

3. **Association**
   - An Association is represented by a dotted line with a line arrowhead (see the figure to the right) and is used to associate data, text, and other Artifacts with flow objects. Associations are used to show the inputs and outputs of activities.

For modelers who require or desire a low level of precision to create process models for documentation and communication purposes, the core elements plus the connectors will provide the ability to easily create understandable diagrams (see Figure 1).

For modelers who require a higher level of precision to create process models, which will be subject to detailed analysis or will be managed by Business Process Management System (BPMS), additional details can be added to the core elements and shown through internal markers (see Figure 2).

**Figure 1: An Example of a Simple Business Process**

**Figure 2: A More Complex Business Process**

---

**Question time**

- default?
- which symbol?
- which implicit gateway?
Question time

Connecting Objects

The Flow Objects are connected together in a diagram to create the basic skeletal structure of a business process. There are three Connecting Objects that provide this function. These connectors are:

- **Sequence Flow**: A Sequence Flow is represented by a solid line with a solid arrowhead (see the figure to the right) and is used to show the order (the sequence) that activities will be performed in a Process. Note that the term “control flow” is generally not used in BPMN.

- **Message Flow**: A Message Flow is represented by a dashed line with an open arrowhead (see the figure to the right) and is used to show the flow of messages between two separate Process Participants (business entities or business roles) that send and receive them. In BPMN, two separate Pools in the Diagram will represent the two Participants.

- **Association**: An Association is represented by a dotted line with a line arrowhead (see the figure to the right) and is used to associate data, text, and other Artifacts with flow objects. Associations are used to show the inputs and outputs of activities.

For modelers who require or desire a low level of precision to create process models for documentation and communication purposes, the core elements plus the connectors will provide the ability to easily create understandable diagrams (see Figure 1).

For modelers who require a higher level of precision to create process models, which will be subject to detailed analysis or will be managed by Business Process Management System (BPMS), additional details can be added to the core elements and shown through internal markers (see Figure 2).

**Question:**

- Which symbol? XOR
- Which implicit gateway? XOR
Identify sub-processes: order fulfillment

In the order fulfillment process of Fig. 3.12, materials from Supplier 1(2) lead to the acquisition of raw materials. These activities, along with their connecting gateways, can be encapsulated in a sub-process called "Acquire raw materials." Similarly, the two parallel branches for shipping and invoicing the order can be grouped under another sub-process called "Ship and invoice." Figure 4.1 illustrates this model, where the above activities have been enclosed in sub-process activities.

Recall that our initial objective was to simplify a process model. Once we have identified the boundaries of the sub-processes, we can simplify the model by hiding the content of its sub-processes, as shown in Fig. 4.2. This is done by replacing the macro-activity representing the sub-process with a standard-size activity. We indicate that this activity hides a sub-process by marking it with a small square with a plus sign (+) inside (like we could expand the content of an activity by pressing the plus button). This operation is called collapsing a sub-process. By collapsing a sub-process, we reduce the total number of activities, thus improving the model readability. In BPMN, a sub-process which hides its internal steps is called a collapsed sub-process, as opposed to an expanded sub-process which shows its internal steps (as in Fig. 4.1).
Hiding sub-processes: order fulfillment

Exercise 4.1
Identify suitable sub-processes in the process for assessing loan applications modeled in Exercise 3.5.

Hint: Use the building blocks that you created throughout Exercises 3.1–3.4.

Collapsing a sub-process does not imply losing its content. The sub-process is still there, just defined at an abstraction level below. In fact, we can nest sub-processes in multiple levels, so as to decompose a process model hierarchically. An example is shown in Fig. 4.3, which models a business process for disbursing home loans. In the first level we identified two sub-processes: one for checking the applicant's liability, the other for signing the loan. In the second level, we factored out the scheduling of the loan disbursement within the process for signing loans into a separate sub-process.

As we go down the hierarchical decomposition of a process model, we can add more details. For example, we may establish a convention that at the top level we only model core business activities, at the second level we add decision points, and so on all the way down to modeling exceptions and details that are only relevant for process automation.

Question: When should we decompose a process model into sub-processes?

We should use sub-processes whenever a model becomes too large that is hard to understand. While it is hard to precisely define when a process model is “too large”, since understandability is subjective, it has been shown that using more than approximately 30 flow objects (i.e. activities, events, gateways) leads to an increased probability of making mistakes in a process model (e.g. introducing behavioral issues). Thus, we suggest to use as few elements as possible per each process model level, and in particular to decompose a process model if this has more than 30 flow objects.

Reducing the size of a process model, for example by collapsing its sub-processes, is one of the most effective ways of improving a process model's readability. Other structural aspects that affect the readability include the density of the...
Fig. 4.3 A process model for disbursing home loans, laid down over three hierarchical levels via process model connections, the number of parallel branches, the longest path from a start to an end event, as well as cosmetic aspects such as the layout, the labels style (e.g. always use a verb-noun style), the colors palette, the lines thickness, etc. More information on establishing process modeling guidelines can be found in Chap. 5.

We have shown that we can simplify a process model by first identifying the content of a sub-process, and then hiding this content by collapsing the sub-process activity. Sometimes, we may wish to proceed in the opposite direction, meaning that when modeling a process we already identify activities that can be broken down in smaller steps, but we intentionally under-specify their content. In other words, we do not link the sub-process activity to a process model at a lower level capturing its content (like if by pressing the plus button nothing would happen). The reason for doing this is to tell the reader that some activities are made up of sub-steps, but that disclosing the details of these is not relevant. This could be the case of activity “Ship product” in the order fulfillment example, for which modeling the distinction between its internal steps for packaging and for shipping is not relevant.

4.2 Process Reuse

By default a sub-process is embedded within its parent process model, and as such it can only be invoked from within that process model. Often, when modeling a business process we may need to reuse parts of other process models of the same organization. For example, a loan provider may reuse the sub-process for signing home loans:
Global sub-processes: home / student loans

suppose the "Sign loan" process is defined as a separate model: it can be reused
4.2 Process Reuse 101

Fig. 4.4

The process model for disbursing student loans invokes the same model for signing loans used by the process for disbursing home loans, via a call activity.

In BPMN, we can define the content of a sub-process outside its parent process, by defining the sub-process as a global process model. A global process model is a process model that is not embedded within any other process model, and such can be invoked by other process models within the same process model collection. To indicate that the sub-process being invoked is a global process model, we use the collapsed sub-process activity with a thicker border (this activity type is called call activity in BPMN). Coming back to the loan disbursement example of Fig. 4.3, we can factor out the sub-process for signing loans and define it as a global process model, so that it can also be invoked by a process model for disbursing student loans (see Fig. 4.4).

Question: Embedded or global sub-process?

Our default choice should be to define sub-processes as global process models so as to maximize their reusability within our process model collection. Supporting processes such as payment, invoicing, HR, printing, are good candidates for being defined as global process models, since they are typically shared by various business processes within an organization. Besides reusability, another advantage of using global process models is that any change made to these models will be automatically propagated to all process models that invoke them. In some cases, however, we may want to keep the changes internal to a specific process. For example, an invoicing process used for corporate orders settlement would typically be different.

Call activities:

- home / student loans

83

thick borders denote call activities (to global sub-processes)
Global processes: advantages

Readability: processes tend to be smaller

Reusability: define once, use many times

Sharing: any change made to a global process is automatically propagated to all models that invoke it
Exercises

Model the following fragments of business processes for assessing loan applications:
Exercise: loan application 1

Once a loan application has been approved by the loan provider, an acceptance pack is prepared and sent to the customer.

The acceptance pack includes a repayment schedule which the customer needs to agree upon by sending the signed documents back to the loan provider.

The latter then verifies the repayment agreement:
- if the applicant disagreed with the repayment schedule, the loan provider cancels the application;
- if the applicant agreed, the loan provider approves the application.

In either case, the process completes with the loan provider notifying the applicant of the application status.
Exercise: loan application 2

A loan application is approved if it passes two checks: (i) the applicant’s loan risk assessment, which is done automatically by a system, and (ii) the appraisal of the property for which the loan has been asked, carried out by a property appraiser.

The risk assessment requires a credit history check on the applicant, which is performed by a financial officer.

Once both the loan risk assessment and the property appraisal have been performed, a loan officer can assess the applicant’s eligibility.

If the applicant is not eligible, the application is rejected, otherwise the acceptance pack is prepared and sent to the applicant.
Exercise: loan application 3

A loan application may be coupled with a home insurance which is offered at discounted prices. The applicant may express their interest in a home insurance plan at the time of submitting their loan application to the loan provider.

Based on this information, if the loan application is approved, the loan provider may either only send an acceptance pack to the applicant, or also send a home insurance quote.

The process then continues with the verification of the repayment agreement.
Once a loan application is received by the loan provider, and before proceeding with its assessment, the application itself needs to be checked for completeness.

If the application is incomplete, it is returned to the applicant, so that they can fill out the missing information and send it back to the loan provider.

This process is repeated until the application is complete.
Exercise: loan application 5

Put together the four fragments of the loan assessment process that you created in previous Exercises.

Then extend the resulting model by adding all the required artifacts.

Moreover, attach annotations to specify the business rules behind: (i) checking an application completeness, (ii) assessing an application eligibility, and (iii) verifying a repayment agreement.
Exercise: loan application 6

 Extend the business process for assessing loan applications that you created in previous exercises by considering the following resource aspects.

 The process for assessing loan applications is executed by four roles within the loan provider:
 a **financial officer** takes care of checking the applicant’s credit history;
 a **property appraiser** is responsible for appraising the property;
 an **insurance sales representative** sends the home insurance quote to the applicant if this is required.
 All other activities are performed by the **loan officer** who is the main point of contact with the applicant.
Exercises: refactoring

Can the process model below execute correctly? If not, how can it be fixed without affecting the cycle, i.e., such that F, G, and E all remain in a cycle?
Semantics annotations

The graphical syntax is not expressive enough to model exactly all interesting situations.

In many cases part of the behaviour is moved to decorations and annotations (without them no implementation is possible).
Loop annotation: ministerial correspondence

we can use annotations to specify loop conditions

the loop-symbol decoration marks the possible repetition of the sub-process activity

Exercise 4.3
1. Identify the entry and exit points that delimit the unstructured cycles in the process models shown in Solution 3.4 and in Exercise 3.9. What are the repetition blocks?

the loop-symbol decoration marks the possible repetition of the sub-process activity

Question
Loop activity or cycle?

The loop activity is a shorthand notation for a structured cycle, i.e. a repetition block delimited by a single entry point to the cycle, and a single exit point from the cycle, like in the example above. Sometimes there might be more than one entry and/or exit point, or the entry/exit point might be inside the repetition block. Consider for example the model in Fig. 4.6. Here the repetition block is composed of activities "Assess application", "Notify rejection" and "Receive customer feedback"; the cycle has an entry point and two exit points, of which one inside the repetition block. When an unstructured cycle has multiple exit points, like in this case, a loop activity cannot be used, unless additional conditions are used to specify the situations in which the cycle can be exited, which will render the model more complex.
Parallel repetition: procurement process

the larger the number of suppliers
the larger the model!

we must revise the model if
the suppliers change!
Multi-instance activities: procurement process

The multi-instance symbol annotation denotes a collection of data. The list of instances is determined dynamically.

The annotation says that as soon as five instance terminate, we cancel the pending ones.

The multi-instance symbol annotation indicates an activity that is executed multiple times concurrently (e.g., repeated activity for multiple entries or data-items).

Fig. 4.8 Obtaining quotes from multiple suppliers, whose number is not known a priori.

Fig. 4.9 Using a multi-instance pool to represent multiple suppliers on the fly from a suppliers database (the updated model is shown in Fig. 4.9). By the same principle, we replace the two pools “Supplier 1” and “Supplier 2” with a single pool, namely “Supplier”, which we also mark with the multi-instance symbol. A multi-instance pool represents a set of resource classes, or resources, having similar characteristics.
Ad-hoc sub-processes: customer process

The ad-hoc symbol annotation denotes an uncontrolled repetition of activities: they may be repeated multiple times with no specific order or not occur at all, until a condition is met.

We can use annotations to specify loop conditions.

As we pointed out in Chap. 3, events are used to model something that happens instantaneously in a process. We saw start events, which signal how process instances start (tokens are created), and end events, which signal when process instances complete (tokens are destroyed). When an event occurs during a process, for example an order confirmation is received after sending an order out to the customer and before proceeding with the shipment, the event is called intermediate. A token remains trapped in the incoming sequence flow of an intermediate event until the event occurs. Then the token traverses the event instantaneously, i.e. events cannot retain tokens. An intermediate event is represented as a circle with a double border.

4.4.1 Message Events

In the previous chapter, we showed that we can mark a start event with an empty envelope to specify that new process instances are triggered by the receipt of a message (cf. Fig. 3.16). Besides the start message event, we can also mark an end event and an intermediate event with an envelope to capture the interaction between our process and another party. These types of event are collectively called message events.

An end message event signals that a process concludes upon sending a message. An intermediate message event signals the receipt of a message, or that a message has just been sent, during the execution of the process. Intermediate and end message events represent an alternative notation to those activities that are solely used to send or receive a message. Take for example activities "Return application to applicant" and "Receive updated applications" in Fig. 4.11a, which is an extract of the loan assessment model of Solution 3.7. It is more meaningful to replace the former activity with an intermediate send message event and the latter activity with an intermediate receive message event, as illustrated in Fig. 4.11b, since these activities...
Message annotated events and activities

A start event can be annotated with a white-envelope: a process instance is created when a certain message is received.

An end event can be annotated with a black-filled envelope: the process concludes by sending a message.

Intermediate events and activities can be annotated with both kinds of envelope: (white = receipt of a message, black = the sending of a message).
Process break (event waiting)

Announce Issues for Vote → Voting Response → Increment Tally

the envelope annotation denotes an intermediate message event: it signals the receipt of a message
Fig. 4.12

Timer events: small claims tribunal

the clock annotation denotes a timer start event:
an instance of the process is created when some temporal event happens

the clock annotation denotes a timer intermediate event:
the process is blocked until a time-out expires

**Example 4.3**

Let us consider the following process at a small claims tribunal.

In a small claims tribunal, callovers occur once a month, to set down the matter for the upcoming trials. The process for setting up a callover starts three weeks prior to the callover day, with the preparation of the callover list containing information such as contact details of the involved parties and estimated hearing date. One week prior to the callover, the involved parties are contacted to determine if they are all ready to go to trial. If this is the case, the callover is set, otherwise it is deferred to the next available slot. Finally, on the callover day, the callover material is prepared and the callover is held.

This process is driven by three temporal events: it starts three weeks prior to the callover date, continues one week prior to the callover date, and concludes on the day of the callover. To model these temporal events we need one start and two intermediate timer events, as shown in Fig. 4.12.
Event-based decisions (also deferred choice)

Event-based split gateway can be used to select a branch based on some external event.
Deferred choice

A race condition between an incoming message and a timer
Exceptions: rainy-days vs sunny-days

Exceptions are events that deviate a process from its normal course.

They include: business faults (e.g., out of stock), technology faults (e.g., database crash).

Exceptions provoke the interruption or abortion of the running process instance.

Before adding exceptions it is important to have the sunny-day scenario well understood.
Process abortion: home loan

- **End terminate event:** causes the immediate cessation of the current process instance (and of any sub-process, but not of the parent process if any)

---

**Fig. 4.18** Using a terminate event to signal improper process termination with a full circle inside, causes the immediate cessation of the process instance at its current level and for any sub-process.

In the example of Fig. 4.18—a variant of the home loan that we already saw in Fig. 4.3—a home loan is rejected and the process is aborted if the applicant has debts and/or low liability. From a token semantics, the terminate event destroys all tokens in the process model and in any sub-process. In our example, this is needed to avoid the process to deadlock at the AND-join, since a token may remain trapped before the AND-join if there is high liability and debts or low liability and no debts.

Observe that if a terminate event is triggered from within a sub-process, it will not cause the abortion of the parent process but only that of the sub-process, i.e. the terminate event is only propagated downwards in a process hierarchy.

**Exercise 4.10** Revise the examples presented so far in this chapter, by using the terminate event appropriately.

---

**4.5.2 Internal Exceptions**

Instead of aborting the whole process, we can handle an exception by interrupting the specific activity that has caused the exception. Next, we can start a recovery procedure to bring the process back to a consistent state and continue its execution, and if this is not possible, only then, abort the process altogether. BPMN provides the **error event** to capture these types of scenario. An end error event is used to interrupt the enclosing sub-process and throw an exception. This exception is then caught by an intermediate catching error event which is attached to the boundary of the same sub-process. In turn, this **boundary event** triggers the recovery procedure through an outgoing branch which is called **exception flow**.
Handling exceptions: rainy-days vs sunny-days

We can handle exceptions of sub-processes by interrupting the activity that caused the exception and moving the control flow to another process.

The recovery procedure can try to bring the process back to a consistent state.

Error end events are used to interrupt the execution.

Boundary events trigger the recovery procedure (called exception flow).
Throwing and catching: order fulfillment

The error event is depicted as an event with a lightning marker. Following the BPMN conventions for throwing and catching events, the lightning is empty for the catching intermediate event and full for the end throwing event.

An example of error events is shown in Fig. 4.19 in the context of our order fulfillment process. If there is an out of stock exception, the acquisition of raw materials is interrupted and the recovery procedure is triggered, which in this case simply consists of a task to notify the customer before aborting the process. In terms of token semantics, upon throwing an end error event, all tokens are removed from the enclosing sub-process (causing its interruption), and one token is sent through the exception flow emanating from the boundary error event. There is no restriction on the modeling elements we can put in the exception flow to model the recovery procedure. Typically, we would complete the exception flow with an end terminate event to abort the process, or wire this flow back to the normal sequence flow if the exception has been properly handled.

4.5.3 External Exceptions

An exception may also be caused by an external event occurring during an activity. For example, while checking the stock availability for the product in a purchase order, the Seller may receive an order cancellation from the customer. Upon this request, the Seller should interrupt the stock availability check and handle the order cancellation. Scenarios like the above are called unsolicited exceptions since they originate externally to the process. They can be captured by attaching a catching intermediate message event to an activity's boundary, as shown in Fig. 4.20. From token semantics, when the intermediate event triggers, the token is removed from the enclosing activity, consequently causing the activity interruption, and sent through the exception flow emanating from the boundary event, to perform the recovery procedure.
Recovery from faults: image manipulation

Fig. 3. The example process as a BPMN process

3.1 Language
BPMN is a rich language that provides the modeler with a large collection of object types to represent various aspects of a business process, including the control-flow, data, resources and exceptions. BPMN is mainly meant for modeling business processes at a conceptual level, meaning that it is mainly intended for drawing process models that will be used for communication between stakeholders in the processes. As a consequence, formal rigor and conciseness were not primary concerns when developing the BPMN specification.

The three types of BPMN objects that can be used to represent the control-flow aspect of a process are activities, events, gateways. Many subtypes of these objects exist. Control-flow objects can be connected by sequence flows, which are directed arcs that represent the flow of control from one object to the next. Figure 3 illustrates some of these objects, by representing the example process in BPMN and by relating the object types to the workflow patterns explained in Sect. 2.2.

3.2 Transformation Challenges
Due to the large number of object types that constitute BPMN it is hard to define a mapping and show (or prove) that the mapping works for all possible combinations of these object types. Especially, because the mapping of a composition of object types is not the same as the composition of the mapping of those object types. This complicates, for example, defining mapping rules for interruptions of sub-process invocations.
Intermediate time out and a loop

![Diagram showing a process with intermediate time out and a loop.](image)

- Any Suppliers?
  - Yes: Repeat for Each Supplier
    - Send RFQ
    - Receive Quote
    - Add Quote
  - No: Send “No Suppliers”
- Intermediate Event for a Time Out
- Time Limit Exceeded
- Marker showing that Sub-Process Loops
- Find Optimal Quote
Compensations

if the compensable activities have been already completed, then they must be compensated

the receipt of an order
cancellation request triggers
the start of a compensation
Exercises

Model the following process fragment:

After a car accident, a statement is sought from two witnesses out of the five that were present, in order to lodge the insurance claim. As soon as the first two statements are received, the claim can be lodged with the insurance company without waiting for the other statements.
Conversations, choreographies, and collaborations
A Conversation is the logical relation of (correlated) Message exchanges
A **Communication** defines a set of logically related message exchanges. When marked with a [+ symbol it indicates a Sub-Conversation, a compound conversation element.

A **Conversation Link** connects Communications and Participants.

A **Forked Conversation Link** connects Communications and multiple Participants.
Choreography

The behaviour of different Conversations is modelled through separate Choreographies

A Choreography defines the sequence of interaction between participants

A choreography does not exist in a pool and it is not executable

It describes how the participants are supposed to behave
Choreography task

A **Choreography task** is an activity in a choreography that consists of a set (one or more) Message exchanges.

A choreography task involves two or more participants that are displayed in different bands.

Top/bottom band positioning is inessential.
Sequence flow in a choreography

Sequence Flow are used within Choreographies to show the sequence of the Choreography Activities, Events, and Gateways.

Figure 12.5 - The use of Sequence Flow in a Choreography

There are two additional variations of Sequence Flow:

• Conditional Sequence Flow: Conditions can be added to Sequence Flow in two situations:
  • From Gateways: Outgoing Sequence Flow have conditions for Exclusive and Inclusive Gateways. The data referenced in the conditions must be visible to two (2) or more Participants in the Choreography. The data becomes visible if it is part of a Message that had been sent (previously) within the Choreography. See pages 375 and 383 for more information about how Exclusive and Inclusive Gateways are used in Choreography.
  • From Choreography Activities: Outgoing Sequence Flow may have conditions for Choreography Activities. Since these act similar to Inclusive Gateways, the Conditional Sequence Flow can be used in Choreographies. The conditions have the same restrictions that apply to the visibility of the data for Gateways.

• Default Sequence Flow: For Exclusive Gateways, Inclusive Gateways, and Choreography Activities that have Conditional Sequence Flow, one of the outgoing Sequence Flow may be a Default Sequence Flow. Because the other outgoing Sequence Flow will have appropriately visible data as described above, the Participants would know if all the other conditions would be false, thus the Default Sequence Flow would be selected and the Choreography would move down that Sequence Flow.

In some applications it is useful to allow more Messages to be sent between Participants when a Choreography is carried out than are contained the Choreography model. This enables Participants to exchange other Messages as needed without changing the Choreography. There are two ways to specify this:

• If the isClosed attribute of a Choreography has a value of false or no value, then Participants MAY send Messages to each other without additional Choreography Activities in the Choreography. Unmodeled messaging can be restricted on particular Sequence Flow in the Choreography, see next bullet. If the isClosed attribute of a Choreography has a value of true, then Participants MAY NOT send Messages to each other without additional Choreography Activities in the Choreography. This restriction overrides any unmodeled messaging allowed by Sequence Flow in the next bullet.

the initiator of the second interaction must be involved in the previous one
Collaboration

A Collaboration contains two or more Pools, representing the Participants in the Collaboration

A Pool may be empty or show a Process within

The Message exchange is shown by a Message Flow that connects Pools or the objects within the Pools. The Messages associated with the Message Flow may also be shown.

Choreographies may be shown “in between” the Pools as they bisect the Message Flow.
Examples
(a taste of BPMN)
Figure 11.3, above, depicts 13 distinct Conversations between collaborating Participants in a logistics domain. As examples, Retailer and Supplier are involved in a Delivery Negotiations Conversation, and Consignee converses with Retailer and Supplier through Delivery/Dispatch Plan and Shipment Schedule Conversations respectively. More than two participants may be involved in a Conversation, e.g., Consignee, Consolidator and Shipper in Detailed Shipment Schedule. The association of Participants to a Conversation are constrained to indicate whether one or many of Participants are involved. For example, one instance of Retailer converses with one instance of Supplier for Deliver Negotiations. However, one instance of Shipper converses with multiple instances of Carrier (indicated by the multiplicity symbol "\*" near Carrier) for Carrier Planning. Note, multiplicity in constraints of Conversation diagrams means one or more (not zero or more).

The behavior of different Conversations is modeled through separate Choreographies, detailing the Message exchange sequences. In practice, Conversations which are closely related could be combined in the same Choreography models – e.g., a Message exchange in the Delivery Negotiation leads to Shipment Schedule, Delivery Planning and Delivery/Dispatch Conversations and these could be combined together in the same Choreography. Alternatively, they could be separated in different models.
A choreography

12.2 Data

A Choreography does not have a central control mechanism and, thus, there is no mechanism for maintaining any central Process (Choreography) data. Thus, any element in a Process that would normally depend on conditional or assignment expressions, would not have any central source for this data to be maintained and understood by all the Participants involved in the Choreography.

As mentioned above, neither Data Objects nor Repositories are used in Choreographies. Both of these elements are used exclusively in Processes and require the concept of a central locus of control. Data Objects are basically variables and there would be no central system to manage them.

Data can be used in expressions that are used in Exclusive Gateways, but only that data which has been sent through a Message in the Choreography.

12.3 Use of BPMN Common Elements

Some BPMN elements are common to both Process and Choreography diagrams, as well as Collaboration; they are used in these diagrams. The next few sections will describe the use of Messages, Message Flow, Participants, Sequence Flow, Artifacts, Correlations, Expressions, and Services in Choreography.

The key graphical elements of Gateways and Events are also common to both Choreography and Process. Since their usage has a large impact, they are described in major sections of this chapter (see page 369 for Events and page 375 for Gateways).
A collaboration with two pools

Figure 3: An Example of a BPD with Pools

Lanes are more closely related to the traditional swimlane process modeling methodologies. Lanes are often used to separate the activities associated with a specific company function or role (see Figure 4). Sequence Flow may cross the boundaries of Lanes within a Pool, but Message Flow may not be used between Flow Objects in Lanes of the same Pool.

Figure 4: A Segment of a Process with Lanes
The scenario modeled in Figure 12.3 entails shipment planning for the next supply replenishment variations: the Supplier confirms all previously accepted variations for delivery with the Retailer; the Retailer sends back a number of further possible variations; the Supplier requests to the Shipper and Consignee possible changes in delivery; accordingly, the Retailer interacts with the Supplier and Consignee for final confirmations.

A problem with model interconnections for complex Choreographies is that they are vulnerable to errors – interconnections may not be sequenced correctly, since the logic of Message exchanges is considered from each partner at a time. This in turn leads to deadlocks. For example, consider the PartnerRole of Retailer in Figure 12.4 and assume that, by error, the order of Confirmation Delivery Schedule and Retailer Confirmation received (far right) were swapped. This would result in a deadlock since both, Retailer and Consignee would wait for the other to send a Message. Deadlocks in general, however, are not that obvious and might be difficult to recognize in a Collaboration.

Figure 12.4 shows the Choreography corresponding to the Collaboration of Figure 12.3 above.
A collaboration diagram: order fulfillment
Multi-instance pools: order fulfillment

the multi-instance symbol annotation
denotes
a set of resources
with similar characteristics

multi-instance sub-process
Exercises: loan application

Extend the loan application model by representing the interactions between the loan provider and the applicant.
From BPMN to Petri nets

Semantics and analysis of business process models in BPMN

Remco M. Dijkman\textsuperscript{a}, Marlon Dumas\textsuperscript{b,c}, Chun Ouyang\textsuperscript{c,*}

\textsuperscript{a} Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, 5600 MB, The Netherlands
\textsuperscript{b} Institute of Computer Science, University of Tartu, J Liivi 2, Tartu 50409, Estonia
\textsuperscript{c} Faculty of Information Technology, Queensland University of Technology, G.P.O. Box 2434, Brisbane, Qld 4001, Australia
Overview

**EVENT**  
- start  
  - Start Event  
- start message  
  - Intermediate Event  
- message  
- timer  
- error  
- end message  
- end  
  - End Event

**ACTIVITY**  
- Task  
- Sub-process Invocation Activity  
- Activity Looping  
- Multiple Instance

**GATEWAY**  
- Parallel Fork Gateway  
- Parallel Join Gateway  
- Data-based XOR Decision Gateway  
- XOR Merge Gateway  
- Event-based XOR Decision Gateway  
- OR Decision Gateway

**SEQUENCE FLOW**  
- Normal Flow  
  - Exception Flow

**MESSAGE FLOW**  
- Message Flow  
  - Interacting processes

[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.

---

**Fig. 1. Overview of BPMN.**

**Fig. 2. Sample workflow net in two different states.**
Simplified BPMN

a start / exception event has just one outgoing flow and no incoming flow

an end event has just one incoming flow and no outgoing flow

all activities and intermediate events have exactly one incoming flow and one outgoing flow

all gateways have either one incoming flow (and multiple outgoing) or one outgoing flow (and multiple incoming)
Simplified BPMN

The previous constraints are no real limitation:

- events or activities with multiple incoming flows: insert a preceding XOR-join gateway
- events or activities with multiple outgoing flows: insert a following AND-split gateway
- gateways with multiple incoming and outgoing flows: decompose in two gateways
- insert start / end event if needed
Simplified BPMN

No link events
they are just a notational convenience
to spread a model into several pages
(no effect on the semantics)

No transactions and compensations

Limited form of sub-processing

no OR-split
(can be expressed in terms of AND-split and XOR-split)

no OR-join
Roughly

A place for each arc
one transitions for each event
one transition for each activity
one or two transitions for each gateway

... with some exceptions!
(start event, end event, event-based gateways, loops, ...)

131
only the fact that one of the conditions will hold true when the gateway is reached. In the case of an event-based gateway, the race condition between events or receive tasks is captured by having the corresponding transitions compete for tokens in the place corresponding to the gateway’s input flow (but without introducing silent transitions as we do for decision gateways). For an OR-split gateway, since its behaviour can be captured through a combination of AND-split and XOR-split gateways [5], the mapping, which is not shown in Fig. 3, can be achieved accordingly.

Finally, places, which are drawn in dashed borders, indicate that their usage is not unique to one module. They are used to link the Petri-net modules of two connecting BPMN objects and thus are identified by both objects. Generally, any sequence flow is mapped onto a place except for event-based decision gateways.

3.2. Activity looping and multiple instances

In BPMN, an activity may have attributes that specify special behaviour such as repetition (i.e., the activity is executed multiple times sequentially) and multiple instantiation (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity repetition: one corresponding to a "while" loop and the other corresponding to a "repeat-until" loop. From a control-flow perspective these repetition constructs can be seen as "macros," in the sense that they can be expanded in terms of decision and merge nodes as shown in Fig. 4. Note that the value of attribute "TestTime" determines whether the repeated activity corresponds to a "while" loop or a "repeat-until" loop.

Activities with a "multiple instantiation" attribute, hereinafter called multi-instance activities, are executed in multiple instances (i.e., copies) with each of these instances running concurrently and independently of the others. The number of instances \( n \) may be determined at design time or at runtime. If \( n \) is known at design time, the "multiple instantiation" construct can be regarded as a macro. Indeed, a multi-instance activity of this type can be replaced by \( n \) identical copies of the activity enclosed between an AND-split and an AND-join as shown in Fig. 5. On the other hand, if \( n \) is only calculated at runtime, we need to synchronise an a priori unknown number of instances of the activity. This type of synchronisation can be expressed using high-level Petri net features such as those found in Coloured Petri nets or YAWL [4]. Since we deliberately restrict the proposed mapping to produce plain Petri nets, we have chosen not to deal with multi-instance activities where \( n \) is only determined at runtime. Nonetheless, if the purpose of the mapping is to check for deadlocks in the process model, we can treat a multiple instance activity as a single-instance one. Indeed, because the multiple instances (or copies) of the activity are executed independently, it is sufficient to check that one instance does not deadlock to ensure that the entire multi-instance activity does not deadlock. This is why our tool implementation offers the option of mapping multiple instance activities with an a priori unknown instance parameter \( n \), with the assumption that \( n = 1 \).

3.3. Subprocess

A subprocess may be viewed as a standalone process. Fig. 6 shows the mapping of a subprocess without exception handling and with a single start and end event. A BPMN process model may have multiple start or end events. The behaviour of such a process is however not clear in the BPMN specification (see Section 4 for a detailed discussion). Hence, we have restricted the mapping to subprocesses with a single start event and a single end event only. This restriction could be lifted if a clear semantics for multiple (sub-)processes with multiple start and end events was given.

Fig. 7 depicts the mapping of calling a subprocess \( P \) via a subprocess invocation activity \( SI \). Two places drawn in dashed borders capture, respectively, the incoming and outgoing flows of activity \( SI \). There are two new...
### Zoom in: end event, task

<table>
<thead>
<tr>
<th>BPMN Object</th>
<th>Petri-net Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x</strong> → <strong>End e</strong></td>
<td><strong>P (x, e)</strong> → <strong>t_e</strong> → <strong>P e</strong></td>
</tr>
<tr>
<td><strong>x</strong> → <strong>Task T</strong> → <strong>y</strong></td>
<td><strong>P (x, T1)</strong> → <strong>T1</strong> → <strong>P (T1, y)</strong></td>
</tr>
</tbody>
</table>
3.2. Activity looping and multiple instances

Activities with a "multiple instantiation" attribute, hitherto known at design time, the "multiple instantiation" loop or a recursion can be achieved accordingly.

On the other hand, if the assumption that all activities for multiple (sub-)processes with multiple start and end events was given, the purpose of the mapping is to check for deadlocks in the process model, we can treat a multiple instance activity as a single-instance one. Indeed, because the multiple instances (or copies) of the activity are executed independently, it is sufficient to check that one instance does not deadlock to ensure that the entire multi-instance activity synchronises an a priori unknown number of instances of the activity. This type of synchronisation can be expressed using high-level Petri net features such as those found in Coloured Petri nets or YAWL.

Generally, any sequence flow is mapped onto a place. The gateway is reached. In the case of an event-based decision gateway, the mapping, can be achieved accordingly. We do for decision gateways. For an OR-split gateway, identical copies of the activity enclosed between an AND-split and an AND-join as shown in (5). Note that the mapping, is only determined at runtime. Nonetheless, if the activity. This type of synchronisation can be expressed via a subprocess invocation activity. A subprocess may be viewed as a standalone process.

Fig. 3 shows the mapping of a subprocess without exception handling and with a single start and end event. A subprocess may be viewed as a standalone process.
3.2. Activity looping and multiple instances

Activities with a "multiple instantiation" assumption that their usage is not unique to one module. They are known at design time, the "multiple instantiation" construct can be regarded as a macro. Indeed, a BPMN process model may have multiple start or end events handling and with a single start and end event. A subprocess may be viewed as a standalone process. Finally, places, which are drawn in dashed borders, indicate the incoming and outgoing transitions of a Petri net. There are two new variants of sequential activity (i.e., the activity is executed multiple times concurrently and independently of the others. The number of instances (i.e., copies) with each of these instances running concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity (i.e., the activity is executed multiple times concurrently).
Only the fact that one of the conditions will hold true when the gateway is reached. In the case of an event-based gateway, the race condition between events or receive tasks is captured by having the corresponding transitions compete for tokens in the place corresponding to the gateway's input flow (but without introducing silent transitions as we do for decision gateways). For an OR-split gateway, since its behaviour can be captured through a combination of AND-split and XOR-split gateways, the mapping, which is not shown in Fig. 3, can be achieved accordingly.

Finally, places, which are drawn in dashed borders, indicate that their usage is not unique to one module. They are used to link the Petri-net modules of two connecting BPMN objects and thus are identified by both objects.

Generally, any sequence flow is mapped onto a place except for event-based decision gateways.

3.2. Activity looping and multiple instances

In BPMN, an activity may have attributes that specify special behaviour such as repetition (i.e., the activity is executed multiple times sequentially) and multiple instantiation (i.e., the activity is executed multiple times concurrently). There are two variants of sequential activity repetition: one corresponding to a "while" loop and the other corresponding to a "repeat-until" loop. From a control-flow perspective these repetition constructs can be seen as "macros," in the sense that they can be expanded in terms of decision and merge nodes as shown in Fig. 4. Note that the value of attribute "TestTime" determines whether the repeated activity corresponds to a "while" loop or a "repeat-until" loop.

Activities with a "multiple instantiation" attribute, hitherto called multi-instance activities, are executed in multiple instances (i.e., copies) with each of these instances running concurrently and independently of the others. The number of instances \( n \) may be determined at design time or at runtime. If \( n \) is known at design time, the "multiple instantiation" construct can be regarded as a macro. Indeed, a multi-instance activity of this type can be replaced by \( n \) identical copies of the activity enclosed between an AND-split and an AND-join as shown in Fig. 5. On the other hand, if \( n \) is only calculated at runtime, we need to synchronise an a priori unknown number of instances of the activity. This type of synchronisation can be expressed using high-level Petri net features such as those found in Coloured Petri nets or YAWL. Since we deliberately restrict the proposed mapping to produce plain Petri nets, we have chosen not to deal with multi-instance activities where \( n \) is only determined at runtime. Nonetheless, if the purpose of the mapping is to check for deadlocks in the process model, we can treat a multiple instance activity as a single-instance one. Indeed, because the multiple instances (or copies) of the activity are executed independently, it is sufficient to check that one instance does not deadlock to ensure that the entire multi-instance activity does not deadlock. This is why our tool implementation offers the option of mapping multiple instance activities with an a priori unknown instance parameter \( n \), with the assumption that \( n = 1 \).

3.3. Subprocess

A subprocess may be viewed as a standalone process. Fig. 6 shows the mapping of a subprocess without exception handling and with a single start and end event. A BPMN process model may have multiple start or end events. The behaviour of such a process is however not clear in the BPMN specification (see Section 4 for a detailed discussion). Hence, we have restricted the mapping to subprocesses with a single start event and a single end event only. This restriction could be lifted if a clear semantics for multiple (sub-)processes with multiple start and end events was given.

Fig. 7 depicts the mapping of calling a subprocess \( P \) via a subprocess invocation activity \( SI \). Two places drawn in dashed borders capture, respectively, the incoming and outgoing flows of activity \( SI \). There are two new...
Task, events and gateways as nets

1. BPMN Object | Petri-net Module
--- | ---
Start s | y
Message E | y
Fork F1 | y1, y2
(Data-based) Decision D1 | y1, y2

2. BPMN Object | Petri-net Module
--- | ---
Task T | y
Task D1, y1 | P (D1, y1)
Join J1 | P (J1, y)
Merge M1 | P (M1, y)

3. BPMN Object | Petri-net Module
--- | ---
Start s | y
Message E | y
Receive task T1 | y2
(Event-based) Decision V1 | T1

[Note]:
x, x1 or x2 represents an input object, and y, y1 or y2 represents an output object.
Activity looping

(a) “while-do” loop

(b) “do-until” loop
Multiple instances (design-time bounded)

ActivityType: Task
LoopType: MultiInstance
MI_Condition: n
MI_Ordering: Parallel
Sub-processes

Sub-processes are...
Message flow

(a) task to task
(b) end event to task
(c) task to start event
(d) end event to start event
Exception handling: single task

Fig. 8. Mapping of a task with an exception flow.
Exception handling: sub-processes

Call subprocess $P$

accounts for multiple instances

Fig. 9. Mapping of a subprocess with an exception flow.

Fig. 10. Mapping of message flows between BPMN processes.
Example: Order process
Example: Order process

![Diagram of order process]

1. **Check credit card**
2. **Prepare products**
3. **Ship products**

**Table 1: Evaluation results of BPMN2PNML**

<table>
<thead>
<tr>
<th>Model No.</th>
<th>BPMN model</th>
<th>Petri net model</th>
<th>Processing times (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11 12 13 14</td>
<td>11 12 13 14</td>
<td>14703 2031</td>
</tr>
<tr>
<td>2</td>
<td>2 3 4 5</td>
<td>2 3 4 5</td>
<td>13875 1297</td>
</tr>
<tr>
<td>3</td>
<td>3 4 5 6</td>
<td>3 4 5 6</td>
<td>13781 1218</td>
</tr>
<tr>
<td>4</td>
<td>4 5 6 7</td>
<td>4 5 6 7</td>
<td>13828 1265</td>
</tr>
<tr>
<td>5</td>
<td>5 6 7 8</td>
<td>5 6 7 8</td>
<td>13187 797</td>
</tr>
<tr>
<td>6</td>
<td>6 7 8 9</td>
<td>6 7 8 9</td>
<td>12810 641</td>
</tr>
<tr>
<td>7</td>
<td>7 8 9 10</td>
<td>7 8 9 10</td>
<td>12474 603</td>
</tr>
<tr>
<td>8</td>
<td>8 9 10 11</td>
<td>8 9 10 11</td>
<td>12047 561</td>
</tr>
<tr>
<td>9</td>
<td>9 10 11 12</td>
<td>9 10 11 12</td>
<td>11620 521</td>
</tr>
<tr>
<td>10</td>
<td>10 11 12 13</td>
<td>10 11 12 13</td>
<td>11200 481</td>
</tr>
</tbody>
</table>

**6. Related work**

To the best of our knowledge, the only other attempt to define a comprehensive formal semantics of BPMN is that of Wong and Gibbons. However, this mapping only covers a small subset of BPMN. In particular, it does not take into account error recovery in the description of BPMN. Experiments show, however, that this approach does not scale beyond relatively small BPMN models (less than 10 nodes), whereas our approach can cope with models at least three times larger without being a factor in efficiency. Moreover, Wong and Gibbons do not show how the CSP semantics can be used to detect various types of errors.
Example: Order process
Example: Order process
Example: Order process

The diagram illustrates an example of an order process, showing the flow of activities such as checking credit card, preparing products, and shipping products. The process uses decision nodes (diamonds) to determine the outcome of each step, leading to different paths for order processing.

The bottom part of the diagram represents the equivalent Petri net model for the BPMN process. Petri nets are a mathematical modeling tool used in the field of computer science to describe and analyze concurrent systems.

This example demonstrates how BPMN models can be translated into Petri nets, which is useful for formal verification and analysis of process models.
Example: Order process
Example: Order process
Example: Order process

Fig. 13. Examples of BPMN process models and transformations to Petri nets.

Table 1

<table>
<thead>
<tr>
<th>Model No.</th>
<th>BPMN model</th>
<th>Petri net model</th>
<th>Processing times (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. Related work

This includes both data-based and event-based gateways.

Example: Order process

![Diagram of an order process workflow]

This includes both data-based and event-based gateways. To the best of our knowledge, the only other attempt to define a comprehensive formal semantics of BPMN is that of Wong and Gibbons. Experiments show however that this approach does not do so. In their work, a BPMN model is presented as a p-calculus expression. The use of Communicating Sequential Processes (CSP) for static analysis of BPMN process models was first presented by Puhlmann and Weske. However, this mapping only covers a small subset of BPMN models. Puhlmann and Weske also show that the handling, which is a key feature of BPMN, relies on a mapping from a subset of BPMN to p-calculus. This tool for static analysis of BPMN process models is a tool for the Mobility Workbench. The produced by this tool can be used to check the soundness of p-calculus. Experiments show however that this approach does not.
Example: Order process
Example: Order process
Example: Travel itinerary

This includes both data-based and event-based gateways. To the best of our knowledge, the only other attempt to define a comprehensive formal semantics of BPMN is that of Wong and Gibbons [22], which uses Communicating Sequential Processes (CSP) as the target formal model.

However, this mapping only covers a small subset of various types of errors. Puhlmann and Weske also show that the handling, which is a key feature of BPMN. Puhlmann [

Experiments show however that this approach does not show how the CSP semantics can be used to detect various types of errors.

In their work, a BPMN model is translated as a simple sequence of BPMN activities is not translated as a CSP process. Also, Wong and Gibbons [

Table 1: Evaluation results of BPMN2PNML

<table>
<thead>
<tr>
<th>Model No.</th>
<th>BPMN model</th>
<th>Petri net model</th>
<th>Processing times (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11 6 4 2 2 19 16 13891 1125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10 5 2 2 11 9 15516 750</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9 8 3 2 2 35 39 14703 2031</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4 4 2 2 10 10 13109 703</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3 9 8 3 2 2 35 39 14703 2031</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2 7 4 4 4 23 21 13875 1297</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 11 2 9 2 31 34 16828 1234</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 2 2 4 11 12 13187 797</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 13. Examples of BPMN process models and transformations to Petri nets.
Example: Travel itinerary

![Diagram of a travel itinerary process model in BPMN and its corresponding Petri net transformation.](Image)
Example: Travel itinerary

![Diagram of a travel itinerary process model in BPMN]
Example: Travel itinerary
Example: Travel itinerary

Tasks: Draft itinerary, Confirm itinerary, Discuss itinerary with client, Change itinerary

Events: XOR

AND Subprocesses

Messages

Exceptions

Places

Transitions

Total Transformation

Evaluation results of BPMN2PNML
Example: Travel itinerary

- Draft itinerary
  - OK?
    - yes: Confirm itinerary
    - no: Discuss itinerary with client
  - no: Change itinerary
    - OK?
      - yes: Confirm itinerary
      - no: Change itinerary

- Draft itinerary
  - OK?
    - yes: Confirm itinerary
    - no: Discuss itinerary with client
Example: Travel itinerary
Example: Travel itinerary
Example: Travel itinerary

A BPMN process model for a travel itinerary is shown, with decision points for confirming the itinerary, discussing it, and changing it. The model includes a draft itinerary stage, followed by a decision point to confirm the itinerary. If confirmed, it proceeds to the discussion stage, while if not confirmed, it returns to the draft stage. The discussion stage includes a decision to change the itinerary, and if not changed, the itinerary is confirmed again.
Example: Travel itinerary
Example: Travel itinerary
Example: Travel itinerary
Exercise

Translate the BPMN collaboration diagram to nets and discuss problematic issues.