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
BPMN 1.0 (2004/06)

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 - Business Process Modeling Notation

**Gateways**
- 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: 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 sparingly as the semantics might not be clear.

**Activities**
- Multiple Instances: Multiple instances of the same activity are started in parallel or sequentially, e.g., for each line item in an order.
- Loop Activity: Throttled if a loop condition is true. The condition is either tested before or after the activity execution.
- Conditional Start Event: Defines the execution order of activities.
- Conditional Flow: Has a condition assigned that determines whether or not the flow is used.
- Default Flow: The default branch to be chosen if all other conditions evaluate to false.
- A Subprocess is a decomposable activity. It can be collapsed to hide the details.
- A Task is a unit of work, the job to be performed.

**Events**
- 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: An event is thrown when the end of the process is reached.
- Intermediate Event Throwing: An event is thrown and the process continues.
- Attached Intermediate Event: The activity is aborted once an event is caught.

**Transactions**
- A Transaction is a set of activities that logically belong together; it might follow a specified transaction protocol.

**Data**
- 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.

**Documentation**
- An arbitrary set of objects can be defined as a Group to show that they logically belong together.

**Swimlanes**
- 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 divide pools or other lanes hierarchically.
- The order of message exchanges can be specified by combining message flow and sequence flow.

**Collapsed Pools**
- A Collapsed Pool hides all internals of the contained processes.
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)
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.
BPMN 2.0 (2009/11)

FAQ

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.
Who is BPMN targeted at?

BPMN is targeted at a high level for business users and at a lower level for process implementers.

The former should be able to easily read and understand a BPMN diagram. The latter should be able to adorn a BPMN diagram with further details in order to represent the process in a physical implementation.

BPMN is targeted at users, vendors and service providers that need to communicate business processes in a standard manner.
Will there be a major rewrite?

Not for 2 or 3 years…

(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', ...

<table>
<thead>
<tr>
<th>Pool</th>
<th>Lane</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>Catching</th>
<th>Throwing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Plain</strong></td>
<td><img src="image" alt="Plain Icon" /></td>
<td><img src="image" alt="Plain Icon" /></td>
</tr>
<tr>
<td><strong>Message</strong></td>
<td><img src="image" alt="Message Icon" /></td>
<td><img src="image" alt="Message Icon" /></td>
</tr>
<tr>
<td><strong>Timer</strong></td>
<td><img src="image" alt="Timer Icon" /></td>
<td><img src="image" alt="Timer Icon" /></td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td><img src="image" alt="Error Icon" /></td>
<td><img src="image" alt="Error Icon" /></td>
</tr>
<tr>
<td><strong>Link</strong></td>
<td><img src="image" alt="Link Icon" /></td>
<td><img src="image" alt="Link Icon" /></td>
</tr>
<tr>
<td><strong>Terminate</strong></td>
<td><img src="image" alt="Terminate Icon" /></td>
<td><img src="image" alt="Terminate Icon" /></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.
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.
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>Set of standardized artefacts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data object</strong></td>
</tr>
<tr>
<td><strong>Group</strong></td>
</tr>
<tr>
<td><strong>Annotation</strong></td>
</tr>
</tbody>
</table>
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.

A message flow is represented by a dashed line with 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.

**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 created 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

Fig. 3.4 An example of the use of XOR gateways forked with an XOR-split. An XOR gateway is indicated with an empty diamond or with a diamond marked with an “X”. From now on, we will always use the “X” marker.

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” (see Fig. 3.5).

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.
**XOR + AND:**

**order fulfillment**

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: order fulfillment

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)

Event-Based This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process (see page 307) or Choreography (see page 375). The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the Alternatives will be chosen.

There are two options for receiving Messages:

- Tasks of Type Receive can be used (see figure top-right).
- Intermediate Events of Type Message can be used (see figure bottom-right).

Inclusive This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow (see page 300). In some sense it is a grouping of related independent Binary (Yes/No) Decisions. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. A Default Condition could be used to ensure that at least one path is taken.

There are two versions of this type of Decision:

- The first uses a collection of conditional Sequence Flow, marked with mini-diamonds (see top-right figure).
- The second uses an Inclusive Gateway (see bottom-right picture).
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

OR gateways, the diagram can ``scale'', but remember all the issues with unmatched OR-joins: they are still valid!

Use OR-gateways only when strictly necessary
XOR + AND + OR:
order fulfillment

Better if gateways are balanced
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 proper condition for a 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)
Information artifacts: order fulfillment

Artifacts provide additional information, at the price of increased complexity.
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” to “Ship product”.

Information artifacts: order fulfillment
Information artifacts: order fulfillment

Data stores (for persistent data objects)
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 shipment address" to activity "Ship product".
Resources as lanes: order fulfillment
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:

**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).

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

**Question time**

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

- **default?**
  - which symbol? XOR
  - which implicit gateway? XOR
Identify sub-processes: order fulfillment

Identifying sub-processes in the order fulfillment process of Fig. 3.12...

Thus these activities, and their connecting gateways, can be encapsulated in a sub-process. In other words, they can be seen as the internal steps of a macro-activity called "Acquire raw materials". Similarly, the two parallel branches for shipping and invoicing the order can be grouped under another sub-process activity called "Ship and invoice". Figure 4.1 illustrates the resulting model, where the above activities have been enclosed in two sub-process activities. We represent such activities with a larger rounded box which closes the internal steps. As we observe from Fig. 4.1, we add data events and data objects inside each sub-process activity, to explicitly indicate when the sub-process starts and completes.

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 (the order fulfillment process has only six activities now), 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).
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 sub-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 text.
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 nested within another 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

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

Readability: processes tend to be smaller

Reusability: define once, use many time

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:
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.
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.
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.
Exercise: loan application 4

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.
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

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?

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?
Fig. 4.7 Obtaining quotes from five suppliers

To do so, we added a task to retrieve the list of suppliers, and passed this list to a multi-instance task, which contacts the various suppliers. You would have noticed that in this example we have also marked the data object Suppliers list with the multi-instance symbol. This is used to indicate a collection of similar data objects, like a list of order items, or a list of customers. When a collection is used as input to a multi-instance activity, the number of items in the collection determines the number of activity instances to be created. Alternatively, we can specify the number of instances to be created via an annotation on the multi-instance activity (e.g. “15 suppliers”, or “as per suppliers database”).

Let us come back to our example. Assume the list of suppliers has become quite large over time, say there are 20 suppliers in the database. As per our organizational policies, however, five quotes from five different suppliers are enough to make a decision. Thus, we do not want to wait for all 20 suppliers to reply back to our request for quote. To do so, we can annotate the multi-instance activity with the minimum number of instances that need to complete before passing control to the outgoing arc (e.g. “complete when five quotes obtained”, as shown in Fig. 4.8).

When the multi-instance activity is triggered, 20 tokens are generated, each marking the progress of one of the 20 instances. Then, as soon as the first five instances complete, all the other instances are canceled (the respective tokens are destroyed) and one token is sent to the output arc to signal completion.

Let us take the order fulfillment example in Fig. 4.2, and expand the content of the sub-process for acquiring raw materials. To make this model more realistic, we can use a multi-instance sub-process in place of the structure delimited by the two OR gateways, assuming that the list of suppliers to be contacted will be determined...
Multi-instance activities: procurement process

the multi-instance symbol annotation denotes a collection of data

the list of instances is determined dynamically

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

the annotation says that as soon as five instance terminate we cancel the pending ones
Ad-hoc sub-processes: customer process

we can use annotations to specify loop conditions

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

as we pointed out in Chap. 3, events are used to model something that happens spontaneously 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. at ok 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 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  
Increment Tally

the envelope annotation denotes an intermediate message event: it signals the receipt of a message
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.
Event-based decisions (also deferred choice)

Event-based Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process (see page 307) or Choreography (see page 375). The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the Alternatives will be chosen.

There are two options for receiving Messages:

- Tasks of Type Receive can be used (see figure top-right).
- Intermediate Events of Type Message can be used (see figure bottom-right).

Inclusive Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow (see page 300). In some sense it is a grouping of related independent Binary (Yes/No) Decisions. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. A Default Condition could be used to ensure that at least one path is taken.

There are two versions of this type of Decision:

- The first uses a collection of conditional Sequence Flow, marked with mini-diamonds (see top-right figure).
- The second uses an Inclusive Gateway (see bottom-right picture).

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

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.

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)
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).
4.5 Handling Exceptions

Fig. 4.19 Error events model internal exceptions. 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 at ok semantics, when the intermediate message event is triggered, 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.

the lightning annotation denotes an error-catching event
Recovery from faults: image manipulation

Legend
- Start event
- End event
- Error exception event – Cancel Region
- Task
- XOR-split – Exclusive Choice
- XOR-join – Simple Merge
- AND-split – Parallel Split
- AND-join – Synchronization
- Sequence flow from – Sequence

Fig. 3. The example process as a BPMN process transformation itself. The section focuses on BPMN version 1.0, because at the time that the transformation was developed that was the current version. Therefore, comments on BPMN apply to version 1.0 only.

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

Figure 2: A Segment of a Process with more Details

Swimlanes

Many process modeling methodologies utilize the concept of swimlanes as a mechanism to organize activities into separate visual categories in order to illustrate different functional capabilities or responsibilities. BPMN supports swimlanes with two main constructs. The two types of BPMN swimlane objects are:

- **Pool**
  - A Pool represents a Participant in a Process. It also acts as a graphical container for partitioning a set of activities from other Pools (see the figure to the right), usually in the context of B2B situations.

- **Lane**
  - A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally (see the figure to the right). Lanes are used to organize and categorize activities.

Pools are used when the diagram involves two separate business entities or participants (see Figure 3) and are physically separated in the diagram. The activities within separate Pools are considered self-contained Processes. Thus, the Sequence Flow may not cross the boundary of a Pool. Message Flow is defined as being the mechanism to show the communication between two participants, and, thus, must connect between two Pools (or the objects within the Pools).

Marker showing that Sub-Process Loops

Intermediate Event for a Time Out

Time Limit Exceeded
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 Conversation is a set of Message exchanges (Message Flow) that share the same Correlation. A Conversation is associated with a set of name-value pairs, or a Correlation Key (e.g., "Order Identifier," "Delivery Identifier"), which is recorded in Messages that are exchanged. In this way, a Message can be routed to the specific Process instance responsible for receiving and processing the Message.

Figure 11.1 shows a simple example of a Conversation diagram.
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.
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.

Sequence Flow will define the order of Choreography elements.

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)
A conversation

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 "*") 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
Figure 12.3 - A Collaboration diagram logistics example

The scenario modeled in Figure 12-4 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.
Semantics and analysis of business process models in BPMN

Remco M. Dijkman a, Marlon Dumas b,c, Chun Ouyang c,*

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

Abstract

The Business Process Modelling Notation (BPMN) is a standard for capturing business processes in the early phases of systems development. The mix of constructs found in BPMN makes it possible to create models with semantic errors. Such errors are especially serious, because errors in the early phases of systems development are among the most costly and hardest to correct. The ability to statically check the semantic correctness of models is thus a desirable feature for modelling tools based on BPMN. Accordingly, this paper proposes a mapping from BPMN to a formal language, namely Petri nets, for which efficient analysis techniques are available. The proposed mapping has been implemented as a tool that, in conjunction with existing Petri net-based tools, enables the static analysis of BPMN models. The formalisation also led to the identification of deficiencies in the BPMN standard specification.

Keywords: Business process modelling and analysis; BPMN; Petri nets

1. Introduction

The Business Process Modelling Notation (BPMN) [17] is a standard notation for capturing business processes, especially at the level of domain analysis and high-level systems design. The notation inherits and combines elements from a number of previously proposed notations for business process modelling, including the XML Process Definition Language (XPDL) [21] and the Activity Diagrams component of the Unified Modelling Notation (UML) [16]. BPMN process models are composed of: (i) activity nodes, denoting business events or items of work performed by humans or by software applications and (ii) control nodes capturing the flow of control between activities. Activity nodes and control nodes can be connected by means of a flow relation in almost arbitrary ways.

Languages that follow a similar paradigm, known as graph-oriented process definition languages, have been previously studied from a formal perspective (e.g., the work on task structures [2]). It is known that models defined in this family of languages may exhibit a range of semantic errors, including deadlocks and livelocks. Such errors are especially problematic at the levels of domain analysis and high-level systems design, because errors at these levels are among the hardest and most costly to correct. BPMN even increases the types of semantic errors with respect to traditional graph-oriented languages, because it combines graph-oriented features with other features, drawn from a range of sources including Workflow Patterns [5] and Business Process Execution Language (BPEL) [12], a standard for defining business processes at the implementation level. These features include the ability to define: (i) subprocesses that may be executed multiple times concurrently; (ii) subprocesses that may be interrupted as a result of exceptions; and (iii) message flows between processes. The interactions between these features are an additional source of semantic errors.

For these reasons the ability to statically analyse BPMN models is likely to become a desirable feature for tools supporting process modelling in BPMN. Anecdotal evidence...
Overview

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

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
- Proc. 1 send
- Proc. 2 receive

[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.
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, ...)
**Zoom in: start, intermediate**

<table>
<thead>
<tr>
<th>BPMN Object</th>
<th>Petri-net Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start s</strong></td>
<td>![Diagram of Start s]</td>
</tr>
<tr>
<td><strong>Message E</strong></td>
<td>![Diagram of Message E]</td>
</tr>
</tbody>
</table>

**Fig. 3. Mapping task, events, and gateways onto Petri-net modules.**
3.2. Activity looping and multiple instances

In BPMN, an activity may have attributes that specify terms of decision and merge nodes as shown in [5]. Since we deliberately restrict the proposed mapping to produce plain Petri nets, we have chosen not to deal with multi-instance activities. For an OR-split gateway, the race condition between events or receive tasks is only calculated at runtime, we need to ensure that the entire multi-instance activity does not deadlock. This is why our tool implementation does not deadlock to ensure that the entire multi-instance activity does not deadlock. This is why our tool implementation does not deadlock.
Zoom in: parallel gateway

<table>
<thead>
<tr>
<th>Fork F1</th>
<th>P (x, F1)</th>
<th>P (F1, y1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y1</td>
<td>t_F1</td>
</tr>
<tr>
<td></td>
<td>y2</td>
<td>P (F1, y2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Join J1</th>
<th>P (x1, J1)</th>
<th>t_J1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>y1</td>
<td>P (J1, y)</td>
</tr>
<tr>
<td>x2</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P (x2, J1)</th>
<th>P (J1, y)</th>
</tr>
</thead>
</table>

Fig. 3. Mapping task, events, and gateways onto Petri-net modules.
### Zoom in: choice gateway

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Decision Diagram" /></td>
<td><img src="image" alt="Petri Net Diagram" /></td>
</tr>
<tr>
<td>(Data-based) Decision D1</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Merge Diagram" /></td>
<td><img src="image" alt="Petri Net Diagram" /></td>
</tr>
<tr>
<td>Merge M1</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram Explanation**

- **Decision D1** is represented by a diamond shape with conditions `c` and `~c` leading to branches `y1` and `y2`, respectively.
- **Merge M1** is represented by a diamond shape with branches `x1` and `x2` leading to `y`, indicating a merge point.

**Petri Net Equivalents**

- For Decision D1:
  - Token transition `t(D1, y1)` leads to place `P(D1, y1)`.
  - Token transition `t(D1, y2)` leads to place `P(D1, y2)`.

- For Merge M1:
  - Token transitions `t(M1, x1)` and `t(M1, x2)` lead to corresponding places `P(M1, y)` and `P(M1, y)`.
**Zoom in: event based gateway**

<table>
<thead>
<tr>
<th>BPMN Object</th>
<th>Petri-net Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BPMN Object</strong></td>
<td><strong>Petri-net Module</strong></td>
</tr>
<tr>
<td><img src="image" alt="BPMN Diagram" /></td>
<td><img src="image" alt="Petri-net Diagram" /></td>
</tr>
<tr>
<td><strong>BPMN Object</strong></td>
<td><strong>Petri-net Module</strong></td>
</tr>
<tr>
<td>Event-based gateway</td>
<td>Event-based gateway</td>
</tr>
<tr>
<td>Message E1</td>
<td>Message E1</td>
</tr>
<tr>
<td>Decision V1</td>
<td>Decision V1</td>
</tr>
<tr>
<td>Receive task T1</td>
<td>Receive task T1</td>
</tr>
<tr>
<td><img src="image" alt="BPMN Diagram" /></td>
<td><img src="image" alt="Petri-net Diagram" /></td>
</tr>
<tr>
<td><strong>BPMN Object</strong></td>
<td><strong>Petri-net Module</strong></td>
</tr>
<tr>
<td><strong>BPMN Object</strong></td>
<td><strong>Petri-net Module</strong></td>
</tr>
<tr>
<td><img src="image" alt="BPMN Diagram" /></td>
<td><img src="image" alt="Petri-net Diagram" /></td>
</tr>
<tr>
<td><strong>BPMN Object</strong></td>
<td><strong>Petri-net Module</strong></td>
</tr>
<tr>
<td><img src="image" alt="BPMN Diagram" /></td>
<td><img src="image" alt="Petri-net Diagram" /></td>
</tr>
</tbody>
</table>

---

**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 [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...
Task, events and gateways as nets

<table>
<thead>
<tr>
<th>BPMN Object</th>
<th>Petri-net Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start s y</td>
<td>P (s, y)</td>
</tr>
<tr>
<td>Message E</td>
<td>P (x, E1)</td>
</tr>
<tr>
<td>Fork F1 y1</td>
<td>P (x, F1)</td>
</tr>
<tr>
<td>(Data-based) Decision D1</td>
<td>P (x, D1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BPMN Object</th>
<th>Petri-net Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task T y</td>
<td>P (x, T1)</td>
</tr>
<tr>
<td>Join J1 y</td>
<td>P (x2, J1)</td>
</tr>
<tr>
<td>Merge M1 y</td>
<td>P (x2, M1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BPMN Object</th>
<th>Petri-net Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message E1 y'1</td>
<td>P (E1, y1)</td>
</tr>
<tr>
<td>Receive task T1 y2</td>
<td>P (T1, y2)</td>
</tr>
</tbody>
</table>

[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

In BPMN, exception handling is captured by exception flows. An exception flow originates from an error event attached to the boundary of an activity. For presentation purposes, it is convenient to distinguish the case where the activity is a single task, from the case where it is a subprocess.

Fig. 8 shows the mapping of an error event associated with a task. Given that the execution of task \( T \) is atomic, the occurrence of exception \( E_x \) may only interrupt \( T \) when \( T \) is enabled and has not yet completed. In Petri net terms, this means that the occurrence of exception \( E_x \) can "steal" the input token that would normally be consumed by the transition corresponding to task \( T \).

In the case of an exception flow associated to a subprocess, the occurrence of the exception (i.e., the error event) will cancel the execution of the subprocess assuming that this latter has started but has not yet completed. The mapping is complicated by the fact that it needs to capture the cancellation of the running subprocess at any point when the exception occurs. This means that when the transition corresponding to the error event fires, all the tokens left in the Petri net fragment corresponding to the subprocess need to be removed. However, due to the local nature of Petri net transitions, it is cumbersome to model a "vacuum cleaner" that would remove all tokens from a given fragment of a net [3].
Finally, we note that if a subprocess $P$ is nested within another subprocess $P_0$, the execution of $P$ may be cancelled due to the cancellation of $P_0$, regardless of the reason why $P_0$ is cancelled. Accordingly, each task or event in $P$ needs to check the OK status of both $P$ and $P_0$ to ensure that once $P_0$ is cancelled the execution of $P$ stops as well.

3.5. Message flow

A message flow describes the interaction between processes. It can be mapped to a place with an incoming arc from the transition modelling a send action and an outgoing arc to the transition modelling a receive action. A special case is the mapping of a message flow to a start event where the process is instantiated each time a message is received. In this case, the message flow is directly mapped to an arc linking the transition that models sending the message to the place that signals triggering the start event (e.g., place $p_s$ in the mapping of start event $s$ shown in Fig. 3, which we refer to as the "trigger place" of start event $s$).

Fig. 10 shows four mapping rules, each capturing a case for a message sent by a task or an end event and received by a task or a start event. Note that a task may be replaced by an intermediate message event without changing the rule. The above mapping is restricted to tasks that either send or receive messages but not both (such as user task and service task). This restriction does not limit the expressive power of BPMN, because successively sending and receiving a message can be represented by two tasks such as a send followed by a receive.

3.6. Initial marking configuration

The initial state of a BPMN model can be specified by the initial marking of the corresponding Petri net model. The basic idea for configuring the initial marking is to mark the trigger places for each of the start events that do not have any incoming message flows and that the processes they belong to are top-level processes. A message flow that has as a target the start event of a process, will create an instance of the process upon message delivery. So, the mapping should ensure that the trigger place of each start event with an incoming message flow does not contain a token in the initial marking, because the process can only be instantiated as a consequence of this event when a message has arrived. A special case is that each top-level process is instantiated by another process via an
Exception handling: single task

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

accounts for multiple instances

Call subprocess $P$

1-safe

Fig. 10. Mapping of message flows between BPMN processes.

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

1288

accounts for multiple instances

Fig. 10. Mapping of message flows between BPMN processes.

Fig. 9. Mapping of a subprocess with an exception flow.
Example: Order process
Example: Order process

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

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

Tasks Events XOR

Example: Order process
Example: Order process

[Diagram of an example order process model in BPMN and its corresponding CSP process model]

- **Check credit card**
- **Prepare products**
- **Ship products**

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, which uses Communicating Sequential Processes (CSP) as the target formal model. In their work, a BPMN model is mapped to a set of CSP processes. Each task is mapped to a CSP process while the flow relations between task objects are captured through CSP events. The object is mapped to a CSP process while the flow relations among tasks are captured through CSP events. The execution of a task generates event occurrences that may then combine with other event occurrences to be enabled. When a task completes, it generates event occurrences that may then combine with other event occurrences to be enabled.
Example: Order process

- Check credit card
- Prepare products
- Ship products

Experiments show however that this approach does not scale beyond relatively small BPMN models (less than 10 occurrences that may then combine with other event occurrences to be enabled. When a task completes, it generates event combinations of CSP events that need to occur for the task conditions for initiation of a task are encoded as possible between task objects are captured through CSP events. The object is mapped to a CSP process while the flow relations produced by this tool can be used to check the soundness of BPMN models using existing reasoning tools based on \textit{p}-calculus, in particular using the Mobility Workbench.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{Model No.} & \textbf{BPMN model} & \textbf{Petri net model} & \textbf{Processing times (ms)} \\
\hline
1 & 1292 & 8 & 2 & 2 & 4 & 11 & 12 & 13187 & 797 \\
2 & 13828 & 7 & 5 & 12 & 4 & 5 & 3 & 25 & 13828 & 1265 \\
3 & 13781 & 6 & 4 & 8 & 4 & 4 & 24 & 20 & 13781 & 1218 \\
4 & 15375 & 5 & 3 & 2 & 2 & 2 & 12 & 11 & 15375 & 734 \\
5 & 13109 & 4 & 4 & 2 & 2 & 10 & 10 & 13109 & 703 \\
6 & 14703 & 3 & 9 & 8 & 3 & 2 & 2 & 35 & 39 & 14703 & 2031 \\
7 & 13875 & 2 & 7 & 4 & 4 & 4 & 23 & 21 & 13875 & 1297 \\
8 & 1281 & 2 & 1 & 1 & 38 & 43 & 14657 & 2016 \\
\hline
\end{tabular}
\caption{Evaluation results of BPMN2PNML}
\end{table}
Example: Order process

[Diagram of a BPMN process model showing the following steps:
1. Message: Check credit card.
2. Decision: OK? Yes/No.
3. Activities: Prepare products, Ship products.
4. Final state: Completion.]
Example: Order process

![Diagram of an order process model in BPMN and its corresponding Petri net representation.]
Example: Order process

![Diagram of Order process]

- **Check credit card**
- **Prepare products**
- **Ship products**

For further explanation, please refer to the text.
Example: Order process
Example: Order process
Example: Order process
Example: Travel itinerary
Example: Travel itinerary
Example: Travel itinerary

[Diagram of a travel itinerary process with decision points and tasks labeled Draft itinerary, Confirm itinerary, Discuss itinerary with client, and Change itinerary.]
Example: Travel itinerary

This includes both data-based and event-based gateways. To the best of our knowledge, the only other attempt to present the foundations of a tool for static analysis of BPMN process models. This tool relies on a mapping from a subset of BPMN to tools such as FDR. However, this mapping only covers a small subset of various types of errors. Also, Wong and Gibbons show how the CSP semantics can be used to detect variants.

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 simple sequence of BPMN activities is not translated as a sequence of processes.
Example: Travel itinerary
Example: Travel itinerary

Diagram of a travel itinerary process model in BPMN. The process starts with a draft itinerary task. If the draft itinerary is OK, it moves to the confirm itinerary task. If not, it goes to a decision task to discuss the itinerary with the client. If the discussion is OK, the itinerary is confirmed. If not, it is changed. The diagram also shows transitions and gates (AND, XOR) for handling different conditions and events.
Example: Travel itinerary

![Travel itinerary diagram]

1. **Draft itinerary**
   - **OK?**
     - **yes** 
       - **Confirm itinerary**
     - **no** 
       - **Discuss itinerary with client**
       - **OK?**
         - **yes** 
           - **Change itinerary**
         - **no** 

2. **Draft itinerary**
   - Connect to other tasks as needed.
Example: Travel itinerary

![Diagram of travel itinerary process]
Example: Travel itinerary

![Travel itinerary diagram]

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. In their work, a BPMN model is mapped to a set of CSP processes. Each task is mapped to a CSP process while the flow relations between task objects are captured through CSP events. The conditions for initiation of a task are encoded as possible combinations of CSP events that need to occur for the task to be enabled. When a task completes, it generates event occurrences that may then combine with other event occurrences to be enabled. Experiments show however that this approach does not show how the CSP semantics can be used to detect various types of errors. To this end, this paper presents the foundations of a CSP-calculus, in particular using the Mobility Workbench, for static analysis of BPMN process models. This tool can be used to check the soundness of produced models. However, this mapping only covers a small subset of BPMN. Puhlmann and Weske [20] present the foundations of a CSP-calculus, in particular using the Mobility Workbench, for static analysis of BPMN process models. This tool can be used to check the soundness of produced models. However, this mapping only covers a small subset of BPMN. Wong and Gibbons [22] do not show how the CSP semantics can be used to detect various types of errors.

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>8</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

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. In their work, a BPMN model is mapped to a set of CSP processes. Each task is mapped to a CSP process while the flow relations between task objects are captured through CSP events. The conditions for initiation of a task are encoded as possible combinations of CSP events that need to occur for the task to be enabled. When a task completes, it generates event occurrences that may then combine with other event occurrences to be enabled. Experiments show however that this approach does not show how the CSP semantics can be used to detect various types of errors. To this end, this paper presents the foundations of a CSP-calculus, in particular using the Mobility Workbench, for static analysis of BPMN process models. This tool can be used to check the soundness of produced models. However, this mapping only covers a small subset of BPMN. Puhlmann and Weske [20] present the foundations of a CSP-calculus, in particular using the Mobility Workbench, for static analysis of BPMN process models. This tool can be used to check the soundness of produced models. However, this mapping only covers a small subset of BPMN. Wong and Gibbons [22] do not show how the CSP semantics can be used to detect various types of errors.

![Diagram of BPMN process models and transformations to Petri nets]
Example: Travel itinerary

Tasks Events XOR

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>11 11 2 9 2 31 34 16828 1234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7 5 12 4 5 31 25 13828 1265</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6 4 8 4 4 24 20 13781 1218</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5 3 2 2 2 12 11 15375 734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4 4 2 2 10 10 13109 703</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>8</td>
<td>8 2 2 4 11 12 13187 797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3 9 8 3 2 2 35 39 14703 2031</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Experiments show however that this approach does not produce the structure of the BPMN model. For example, a simple sequence of BPMN activities is not translated as a single event in CSP processes, whereas our approach can cope with models at least three times larger without being a sequential process.

Puhlmann and Weske also show that the CSP processes can be used to check the soundness of the model. In their work, a BPMN model is translated into a CSP process, which can then be checked for consistency.

Experiments show however that this approach does not show how the CSP semantics can be used to detect various types of errors. Wong and Gibbons define a comprehensive formal semantics of BPMN that relies on a mapping from a subset of BPMN to Sequential Processes (CSP) as the target formal model. This tool for static analysis of BPMN process models. This tool does not show how the CSP semantics can be used to detect various types of errors.
Example: Travel itinerary
Example: Travel itinerary
Exercise

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