Business Processes Modelling

**MPB (6 cfu, 295AA)**

**Roberto Bruni**

[http://www.di.unipi.it/~bruni](http://www.di.unipi.it/~bruni)

21 - BPMN analysis
We overview the main challenges that arise when analysing BPMN diagrams with Petri nets.
BPMN Diagrams
Business process diagrams

BPMN defines a standard for Business Process Diagrams (BPD) based on flowcharting technique

Four categories of elements

<table>
<thead>
<tr>
<th>swimlanes</th>
<th>flow objects</th>
<th>connecting objects</th>
<th>artefacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lane</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>□</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ → ○</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ → ○ ↓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Store</td>
<td></td>
</tr>
</tbody>
</table>

(ET) - Error - compensation
Wrong use of flows in/between Pool A

Multiple Rule Cancel Message define message flows between Pools.

are incorrectly used to connect pools. flows substitute sequence flows. often missing, because it is wrongly presumed that message sequence flow

Starting Task D Event X Document Y a document is a task itself. This task is redundant. The act of receiving Message Task B X Intermediate Document X Additionally, sequence flows ...
...

flows and start/end events are Events Performing Task C Gateways Gateways are connected only with events cannot be decision

Data based exclusive decision or Decision Exception X Exception X

EPC (roughly)

Wrong model Warning or error in the BPMN model

BPMN vs EPC (roughly)

Pool Lane Lane Lane

swimlanes event
event activity function connector
gateway control flow

sequence flow

message flow

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)

BPMN vs EPC (roughly)
BPMN 2.0 poster

Activities
- **Task**
- **Conversation**
- **Event**
- **Sub-Process**
- **Call Activity**

Activity Markers
- **Sub-Process Marker**
- **Loop Marker**
- **Parallel Marker**
- **Ad Hoc Marker**
- **Compensation Marker**

Task Types
- **Send Task**
- **Receive Task**
- **User Task**
- **Manual Task**
- **Business Rule Task**
- **Service Task**
- **Script Task**

Sequence Flow
- **Default Flow**
- **Conditional Flow**

Conversations
- A **Communication** defines a set of logically related message exchanges. When marked with a **symbol it indicates a Sub-Process, an activity that can be refined.**
- A **Transaction** is a set of activities that logically belong together; it might follow a specific transaction protocol.
- An **Event** is placed into a **Process or Sub-Process.** It is activated when its start event gets triggered and can interrupt the higher level process context or run in parallel (non-interrupting) depending on the start event.
- A **Call Activity** is a wrapper for a globally defined Sub-Process or Task that is reused in the current process.

Event Markers
- **Start Event**
- **Intermediate Event**
- **End Event**

Gateways
- **Exclusive Gateway**
- **Event-based Gateway**
- **Parallel Gateway**
- **Inclusive Gateway**
- **Complex Gateway**

Swimlanes
- **Task**
- **Message Flow**
- **Event**
- **Decision**
- **Message Flow**

Data
- **Input**
- **Task**
- **Output**
- **Data Store**

Choreographies
- A **Choreography Task** represents an interaction (Message Exchange) between two Participants.
- Multiple **Participants Marker** denotes a set of Participants of the same kind.
- A **Choreography Sub-Process** contains a refined choreography with several interactions.

Collaboration Diagram
- **Pool (Collapsed)**
- **Participant A**
- **Participant B**
- **Participant C**

Messages
- **Message Flow**
- **Event**
- **Decision**
- **Message Flow**

Data
- **Input**
- **Task**
- **Output**
- **Data Store**

Events
- None: Untyped events include start point, state changes or final states.
- **Message**: Receiving and sending messages.
- **Timer**: Cyclic timer events, points in time, time spans or timeouts.
- **Escalation**: Escalating to an higher level of responsibility.
- **Conditional**: Reacting to changed business conditions or integrating business rules.
- **Link**: Off-page connections. Two corresponding link events equal a sequence flow.
- **Error**: Catching or throwing named errors.
- **Cancel**: Reacting to cancelled transactions or triggering cancellation.
- **Compensation**: Handling or triggering compensation.
- **Signal**: Raising across different processes. A signal thrown can be caught multiple times.
- **Multi**: Catching one out of a set of events. Throwing all events defined in a set.
- **Parallel Multi**: Catching all out of a set of parallel events.
- **Terminate**: Triggering the immediate termination of a process.

BPMN 2.0 - Business Process Model and Notation http://bpmb.de/poster
Resources as lanes: order fulfillment
From processes to collaborations

a more detailed collaboration
A negotiation without choice
BPMN Semantics
BPMN formal semantics?

Many attempts:
Abstract State Machines (ASM)
Term Rewriting Systems
Graph Rewrite Systems
Process Algebras
Temporal Logic

... Petri nets

(Usual difficulties with OR-join semantics)
Sound BPMN diagrams

We can exploit the formal semantics of nets to give unambiguous semantics to BPMN process diagrams BPMN collaboration diagrams

We transform BPMN process diagrams to wf nets BPMN collaboration diagrams to wf systems

A BPMN diagram is sound if its net is so We can reuse the verification tools to check if the net is sound
Translation of BPMN to Petri nets
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
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 events if needed
Pay attention to gateways

stands for

stands for

stands for
My suggestions

stands for

stands for

stands for
Simplified BPMN

Avoid OR-gateways
(all problems seen with EPC apply to BPMN as well)

Limited form of sub-processing

No transactions and compensations
The twist!

BPMN object

- event
- activity
- sequence flow
- message flow

net fragment

- transition
- transition
- place
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, ...)

no dummy objects!
The strategy

From BPMN process diagrams to WF nets in three steps

Travel

Start

Travel request

Book flight

Book hotel

Travel planned

Step 1
convert sequence flow
message flow

Step 2
convert flow objects

Step 3
enforce initial place
final place
Step 1: convert flows

We insert a place for each sequence flow and message flow.
Step 2: convert flow objects

Then insert transitions
Step 2: gateways

BPMN object

- AND split / join
- XOR split
- XOR join

net fragment

- transition
Step 2: event-based

BPMN object

net fragment

Step 1
sequence flow
message flow

Step 2
place fusion
Step 3: add unique start

XOR start
Step 3: add unique end

Steps 1+2

Step 3 unique start

XOR end

(sometimes AND can be preferred)
Example:
Order process
Order process

Sound?
Order process: step 1

Order process

Step 1
sequence flow
message flow

Check credit card

Prepare products

Ship products
Order process: step 2

Step 2 flow objects
Order process: (desugar)
Order process: step 3

Step 3:
- Enforce initial place
- Enforce final place
Soundness analysis

Not sound!
Soundness analysis
Soundness analysis
Soundness analysis

Not sound!
Example:
Travel itinerary
Travel itinerary

Sound?
Travel itinerary: step 1

Step 1 sequence flow
message flow

Draft itinerary

Confirm itinerary

Discuss itinerary

Change itinerary

ok? yes

no
Travel itinerary: step 2
Travel itinerary: (desugar)
Travel itinerary: step 3

Step 3: enforce initial place final place
Soundness analysis

Not sound!
Soundness analysis
Example: Always sushi
Always sushi

Sound?
Sushi lover

Sound?
Sushi lover: step 1
Sushi lover: step 2

Step 2
flow objects
Sushi lover: (desugar)
Soundness analysis

safe & sound
(s-net)
Sushi doomed

Sound?
Sushi doomed: step 1
Sushi doomed: step 2

Step 2
flow objects
Sushi doomed: (desugar)
Sushi doomed: step 3

Step 3
enforce
initial place
final place
Soundness analysis

safe & sound
(s-net)
Sushi system

Sound?

Sushi doomed
Sushi system: step 1
Sushi system: step 1+2+3
Soundness analysis

Sound!
Soundness analysis
Example: Buyer - Reseller
Buyer 3 and Reseller 2

Sound?
Buyer 3 sound?

Step 1 + 2 + 3
Buyer 3 soundness analysis

Safe and sound!
Reseller 2 sound?

Step 1 + 2 + 3
Reseller 2 soundness analysis

Safe and sound!
Buyer 3 + Reseller 2: sound?

Step 1 + 2 + 3
Buyer 3 + Reseller 2: analysis

Not sound!
Buyer 3 + Reseller 2: analysis

Not sound!
Step 0: preprocessing BPMN diagrams
Overview

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

ACTIVITY
- Task
- Sub-process Invocation Activity
- Activity Loopying
- 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.
Activity looping

(a) “while-do” loop

(b) “do-until” loop

Fig. 4. Macro expansions for repeated activities.

Fig. 5. Macro expansion for a multi-instance activity where it is a subprocess.

Fig. 6. Mapping of a subprocess without exception handling.

Fig. 7. Calling a subprocess via a subprocess invocation activity.

Fig. 8. Mapping of a task with an exception flow.
Multiple instances (design-time bounded)

Fig. 4. Macro expansions for repeated activities.

Fig. 5. Macro expansion for a multi-instance activity where \( n \) is known at design time.

Fig. 6. Mapping of a subprocess without exception handling.

Fig. 7. Calling a subprocess via a subprocess invocation activity.

Fig. 8. Mapping of a task with an exception flow.
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.

Fig. 4. Macro expansions for repeated activities.

Fig. 5. Macro expansion for a multi-instance activity where $n$ is known at design time.

Fig. 6. Mapping of a subprocess without exception handling.

Fig. 7. Calling a subprocess via a subprocess invocation activity.

Fig. 8. Mapping of a task with an exception flow.
Exception handling: single task

Exception handling in BPMN 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].

Fig. 4. Macro expansions for repeated activities.

Fig. 5. Macro expansion for a multi-instance activity where $n$ is known at design time.

Fig. 6. Mapping of a subprocess without exception handling.

Fig. 7. Calling a subprocess via a subprocess invocation activity.

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

accounts for separate execution of multiple instances

Call subprocess $P$ to ensure that once $P_0$ needs to be cancelled, regardless of the reason why $P_0$ is cancelled the execution of $P$, the execution of $P_0$, the execution of $P$, or receive messages but not both (such as a 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 task followed by a receive task. The BPMN specification distinguishes between sending an unsolicited message and sending a message in response to an event generated by a task. In the former case, the receiving arc to the transition modelling a receive action. A special case is the mapping of a message flow to a start event by the place that signals triggering the start event to an arc linking the transition that models sending the received. In this case, the message flow is directly mapped where the process is instantiated each time a message is received by a task or an end event and event to abort the process, or wire this flow back to the normal sequence flow if the exception has been properly handled. Finally, we note that if a subprocess is called, regardless of the reason why it may be cancelled, the execution of $P$ stops as well.

For example, while checking the stock availability for the product in a purchase request, the Seller should interrupt the stock availability check and handle the order cancellation from the customer. Upon this event, the Seller may receive an order cancellation from the customer. In terms of BPMN conventions for throwing and catching events, the lightning is empty for the intermediate message event to an activity's boundary, as shown in Fig. 9. Exception handling: sub-processes

Exception handling: sub-processes

3.5. Message flow

3.6. Initial marking configuration

Initial state of a BPMN model can be specified by accounts for separate execution of multiple instances.