# INFO-H420
# Management of Data Science and Business Workflows
## *Part I*
## *2. Process Modeling*

Dimitris SACHARIDIS

2023-2024

# BPM structure

# BPMN Main Elements - Recap

# BPMN Gateways

| Exclusive (XOR) | Parallel (AND) | Inclusive (OR) |
| --- | --- | --- |
| • Exclusive decision take one branch<br>• Exclusive merge Proceed when one branch has completed | • Parallel split take all branches<br>• Parallel join proceed when all incoming branches have completed | • Inclusive decision take one or several branches depending on conditions<br>• Inclusive merge proceed when all active incoming branches have completed |

# Critique this model

# Is this better?

# Identifying sub-processes

# Using the Expanded Sub-Process Notation

# Sub-processes

- An activity in a process can invoke a separate sub-process
- Use this feature to:
  1. Decompose large models into smaller ones, making them easier to understand and maintain

# Guideline: Multi-level modeling

- Level 1: **value chain**
  - Simple linear description of the phases of the process
  - No gateways
  - Each activity chain is a sub-process
- Level 2+: expand each activity in the value chain, add incrementally the following:
  - Decisions, handoffs (lanes, pools)
  - Parallel gateways, different types of events
  - Data objects & data stores
  - And as much detail as you need, and no more

# Guideline: Multi-level modeling (cont.)

- At each level, decompose according to:
  - Logical milestones towards achieving the outcome of the process
  - Major objects used in the process
- Decompose until processes are of “reasonable” size
  - e.g., up to 20 nodes (tasks+events+gateways) per model

# Non-BPMN value chain “chevron” notation

# Sub-processes

- An activity in a process can invoke a separate sub-process
- Use this feature to:
  1. Decompose large models into smaller ones, making them easier to understand and maintain
  2. Share common fragments across multiple processes

# Shared sub-process

# Sub-processes

- An activity in a process can invoke a separate sub-process
- Use this feature to:
  1. Decompose large models into smaller ones, making them easier to understand and maintain
  2. Share common fragments across multiple processes
  3. Delimit parts of a process that can be:
     - Repeated
     - Interrupted

# STRUCTURED REPETITION

# Block-structured repetition: Activity loop

*Activity loop markers* allow us to state that a task or a sub-process may be repeated multiple times

# More on rework and repetition

# Example: block-structured repetition

# Parallel repetition: multi-instance activity

The multi-instance activity provides a mechanism to indicate that an activity is executed ***multiple times concurrently***

Useful when the same activity needs to be executed for multiple entities or data items, such as:

- Request quotes from multiple suppliers
- Check the availability for each line item in an order separately
- Send and gather questionnaires from multiple witnesses in the context of an insurance claim

# Example: multi-instance activity

**Procurement**

In procurement, typically a quote is to be obtained from all preferred suppliers (assumption: five preferred suppliers exist). After all quotes are received, they are evaluated and the best quote is selected. A corresponding purchase order is then placed.

# Is this elegant?

## Procurement

# Solution: with multi-instance activity

## Procurement

# EVENTS

# Events

In BPMN, events model something instantaneous happening during the execution of a process

Types of event:

- Start
- Intermediate
- End

# BPMN event types

**Untyped Event** – Indicates that an instance of the process is created (start) or completed (end), without specifying the cause for creation/completion

**Start Message Event** – Indicates that an instance of the process is created when a message is **received**

**End Message Event** – Indicates that an instance of the process is completed when a message is **sent**

**Intermediate Message Event** – Indicates that an event is expected to occur during the process. The event is triggered when a message is **received** or **sent**

# Comparison with sending/receiving tasks

# So, when to use what?

Use message events only when the corresponding activity would simply send or receive a message and do nothing else

# Temporal events

**Start Timer Event** – Indicates that an instance of the process is created at certain date(s)/time(s), e.g. start process at 6pm every Friday

**Intermediate Timer Event** – Triggered at certain date(s)/ time(s), or after a time interval has elapsed since the moment the event is “enabled” (delay)

# Example

## PO handling

A Purchase Order (PO) handling process starts when a PO is received. The PO is first registered. If the current date is not a working day, the process waits until the following working day before proceeding. Otherwise, an availability check is performed and a PO response is sent back to the customer.

# Recap: Message and Timer events

| Start | Intermediate | | End |
|---|---|---|---|
| **Catching** | **Catching** | **Throwing** | **Throwing** |
| process starts upon message received | message received during the process | message sent during the process | process ends upon message sent |
| process starts when time event occurs | time event occurred (to model delay) | | |

# Data-based vs. Event-based decision

With the XOR-split gateway, a branch is chosen based on conditions that evaluate over available **data**

→ The choice can be made immediately after the token arrives from the incoming flow

Sometimes, the choice must be delayed until an **event** happens

→ The choice is based on a “race” among events

Two types of XOR split:

data-driven XOR-split

event-driven XOR split

# Choices outside our control…

**Stock replenishment**

A restaurant chain submits a purchase order (PO) to replenish its warehouses every Thursday. The restaurant chain's procurement system expects to receive either a "PO Response" or an error message. However, it may also happen that no response is received at all due to system errors or due to delays in handling the PO on the supplier's side. If no response is received by Friday afternoon or if an error message is received, a purchasing officer at the restaurant chain's headquarters should be notified. Otherwise, the PO Response is processed normally.

# Solution: event-driven XOR split

## Stock replenishment

# EXCEPTION HANDLING

# Abortion (terminate event)

Exceptions are events that deviate a process from its “normal” course

The simplest form of exception is to notify that there is an exception (negative outcome)

This can be done via the Terminate end event: it forces the whole process to *abort* (“wipes off” all tokens left behind, if any)

# Example: terminate event

Abort the process by removing all tokens…

# Exception handling

Handling exceptions often involves stopping a sub-process and performing a special activity

Types of exceptions for an activity (task/sub-process) in BPMN:

**External**: something goes wrong outside the process, and the execution of the current activity must be interrupted. Handled with the Message event

**Internal**: something goes wrong inside an activity, whose execution must thus be interrupted. Handled with the Error event

**Timeout**: an activity takes too long and must be interrupted. Handled with the Timer event

All these events are catching intermediate events. They stop the enclosing activity and start an exception handling routine.

# Let’s extend our PO handling process

**PO handling**

A PO handling process starts when a PO is received. The PO is first registered. If the current date is not a working day, the process waits until the following working day before proceeding. Otherwise, an availability check is performed and a PO response is sent back to the customer.

# Let’s extend our PO handling process

**PO handling**

A PO change request may be received anytime after the PO is registered. This request includes a change in quantity or line items. When such a request is received, any processing related to the PO must be stopped. The PO change request is then registered. Thereafter, the process proceeds as it would do after a normal PO is registered. Further, if the customer sends a PO cancelation request after the PO registration, the PO processing must be stopped and the cancelation request must be handled.

# Solution: exception handling

## PO handling

# Internal exception: error event

**Error Event** – Indicates an error: the "end" version generates an error event while the "catching intermediate" version consumes it when <u>attached</u> to the boundary of an activity

# Example: internal exception

**PO handling**

Consider again our “PO Handling process” example with the following extension: if an item is not available, any processing related to the PO must be stopped. Thereafter, the client needs to be notified that the PO cannot be further processed.

# Solution: internal exception

## PO handling

# Example: activity timeout

**Order-to-transportation quote**

Once a wholesale order has been confirmed, the supplier transmits this order to the carrier for the preparation of the transportation quote. In order to prepare the quote, the carrier needs to compute the route plan (including all track points that need to be traversed during the travel) and estimate the trailer usage.

By contract, wholesale orders have to be dispatched within four days from the receipt of the order. This implies that transportation quotes have to be prepared within 48 hours from the receipt of the order to remain within the terms of the contract.

# Solution: activity timeout

## Order-to-transportation quote

# More on the PO handling example…

## PO handling

The customer may send a request for address change after the PO registration. When such a request is received, it is just registered, without further action.

# Non-interrupting boundary events

Sometimes we may need to trigger an activity **in parallel** to the normal flow, i.e. without interrupting the normal flow.

This can be achieved by using *non-interrupting* boundary events

# Solution: non-interrupting boundary events

## PO handling

# Summary

- In this lecture we have learned about:
  - BPMN sub-processes
  - Repetition markers: loop marker and parallel multi-instance marker
  - Events: timer, message and error events
  - Event-based choice gateway
  - Boundary events: interrupting and non-interrupting
  - Error events (throw and catch)

# Acknowledgements


- All material comes from Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A. Reijers, authors of the “Fundamentals of Business Process Management” book.