Participant:17 - Interview Transcript

BPMN

Compensation mechanism & task resuming

> You suggested using the BPMN compensation mechanism for task resuming. We understood that compensation looks like a rollback. Why do you see it as a solution for resuming tasks?

Exactly, compensation is a rollback-like mechanism, but useful for adaptivity. Classic BPMN example: I book hotel, bus, flight; if the hotel becomes unavailable I cancel the other bookings and return to a prior state. In robotics: imagine a mission composed of tasks plus a parallel battery-monitoring task. If monitoring shows the robot won't complete the mission, compensation lets you undo actions and restore the starting state. Compensation is therefore useful to restore things cleanly rather than leaving partially completed side-effects (e.g., cards left on a table). It's not only deleting a token, it undoes actions.

> Good. We think compensation could be used for error handling/adaptation. Do you agree?

Absolutely. Exception handling via compensation brings the computation to a clearer state. It can revert state after a subprocess exception.

Loop / Exclusive Gateway

> What happens if at an exclusive gateway two outgoing conditions are both true? Does the XML/or tool order decide which path, or is there a more sophisticated mechanism?

If you use an exclusive gateway, BPMN requires outgoing conditions to be mutually exclusive, it does not allow both to be true. If both are false, you must provide a default flow. The case where multiple outgoing conditions may be true is the inclusive gateway, which behaves differently (it can allow multiple simultaneous branches). If your model only uses exclusive gateways, that problematic "both true" scenario should not occur.

> Do you agree with the representation of the loop?

The test semantics for looped tasks are separate: tasks with loop markers include a test attribute before or after execution, so you can model while/do-while behavior. The loop marker and MI (multiple-instance) markers plus completionCondition, let you express variants of for/while/do-while semantics.

> So, can we use the MI task marker sequential to express a for loop?

Yes. MI-sequential is basically a macro of loop semantics. You can use a collection or a cardinality to iterate (for), and there is a completionCondition attribute to stop earlier if needed. Also, looped tasks have test-before/test-after attributes so you can realize while/do-while patterns. MI-sequential behaves as a for-like construct when you iterate over a collection or use a cardinality.

> So MI-sequential can behave like for; loop marker + test attribute cover while/do-while.

Correct. The task is executed at least once if configured that way (do-while style) and you also have options to dynamically compute the number of iterations.

Time-based events, “wall-clock”, scheduling, resource contention

> We received this comment: “BPMN has time-based events but no notion of a "wall-clock" or schedule, which makes avoiding resource contention harder.” We know BPMN supports datetime-based time events, so we were wondering, what do you think about this comment

The participant has a point: timer semantics are under-specified in the standard. You can put a date/time (e.g., “May 1st”), durations, and timer events, but the standard does not mandate a precise format or a scheduling model. There’s no standard specification for how to represent workload or to reason about resource contention in the standard itself. People inject workload / scheduling parameters only at simulation or engine-configuration time. That makes scheduling/resource contention outside BPMN core semantics and tool-dependent.

You can specify durations on tasks and timer events, but not a standardized scheduling or wall-clock model. So yes, time events exist, but the handling and formats are under-specified; workload/resource modeling is left to simulation/engine extensions.