Integration of roll-back algorithms into process architecture

Deliverable D5.3

FFG – IKT der Zukunft
SHAPE Project
2014 – 845638

Table 1 Document Information

Project acronym:	SHAPE
Project full title:	Safety-critical Human- & dAta-centric Process management in Engineering projects
Work package:	5
Document number:	5.3
Document title:	Integration of Re-configuration/Re-scheduling Algorithms into Process Architecture
Version:	0.1
Delivery date:	E03/2016 (M3)
Actual publication date:	01.04.2016
Dissemination level:	Public
Nature:	Technical Report
Editor(s) / lead beneficiary:	Siemens
Author(s):	S. Sperl
Reviewer(s):	G. Havur, A. Polleres

Table 2 History

Version	Changes	Authors
0.1	Created Structure	S. Sperl

Contents

1 Introduction

A rollback returns the process to some previous state, which is important for the integrity of the system in case of errors. There are various techniques and to achieve this effect with different technical and readability limitations for their use. This document intends to give an overview over the available techniques for "returning to a previous state" available in Camunda/BPMN.

2 Related Work

Avizienis et al. [1] present a taxonomy of error handling techniques: Rollback, rollforward, and compensation. This categorization of exception handling strategies has been applied to process modeling by Russell et al. [7]. They identify three strategies in regard to process exception handling: *No action*, *rollback*, and *compensation*: A rollback is an operation which returns the process execution to some previous state, and a compensation is the action taken to recover from error or cope with a change of plan.

In long running business process instances, rollback operation requires a compensation process that *undo*s the effects of executed actions up to the rollback state, because parts of a transaction (e.g., communications with external agents) are inherently impossible to undo. Butler et al. [2] describe the StAC language which can be used to specify the orchestration of activities in long running business transactions and therefore, they show that compensation is more general than traditional rollback in database transactions. In a BPMN compliant representation, Ritter et al. [6] make use of rollback as an exception strategy resulting in general patterns for exception handling and compensation. Urban et al. [8] introduce assurance points enhance the use of integration rules, providing checkpoints that are placed at critical locations in the flow of a process, which are also used as intermediate rollback points to support compensation, retry, and contingent procedures in an attempt to maximize forward recovery. van Beest et al. [9] focus on run-time handling of interference by identifing and resolving potentially erroneous situations. They define dependency scopes to represent the dependencies between processes and data sources, which allows a rollback to be executed.

Golani and Gal [3] describe rollback and stepping forward in process modeling. In their model, an exception handler is expected to perform first its set of rollback tasks and then its set of stepping forward tasks, although they do note that either or both sets of tasks may be empty. In a similar fashion, Zhao et al. [10] propose a self-healing framework with rollback operation in order to provide reliable business process execution. In their work, not only the defined set of operations are rolled back but also related data is revoked for maintaining transactional consistency.

Implementing rollback mechanisms for already existing modeling languages might have some shortcomings and limitations. For addressing this issue, Rabbi et al. [5] describe a new modelling language Compensable WorkFlow nets (CWF-nets) for allowing compensable transactions (A type of transaction whose effect can be semantically undone even after it has committed) with the help of *t-calculus* [4] which was developed to aid in the creation and verification of compensable systems.

3 Rollback in Camunda

In general there are two distinct ways of managing rollbacks in Camunda.

- On the database level.
- On a process (BPMN supported) level

The main difference is transactions on a process level might run over days and months, which is not advised or even possible for database transactions.

4 Database Level Rollback

Key point is that these transactions are not modelled by BPMN elements, these transactions are typically cancelled by exceptions thrown in the process specific Java event handlers.

The main concept for database level transactions in Camunda is the wait state. Whenever a wait state is reached the process state is persisted in the database.

In Camunda the user task, receive task, message event, timer event and signal event are all wait states by default.

Other activities can be turned into wait states via the XML attributes camunda:asyncBefore and camunda:asyncAfter

```
<serviceTask id="useService" name="Use Service"
    camunda:asyncBefore="true" camunda:class="at.shape.DelegateClass" />
```

```
<serviceTask id="useService" name="Use Service"
    camunda:asyncAfter="true" camunda:class="at.shape.DelegateClass" />
```

The effect of these annotations are; that whenever a user enters data which is followed by processing steps and that processing fails, the process returns to the user task, as seen in Figure 1.

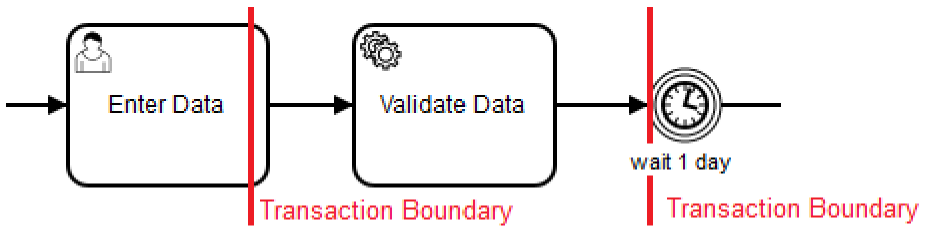

Figure 1 Transaction Boundaries in Camunda

Camunda supports the Java Transaction API (JTA) [1], therefore other JTA compatible (database) systems can be incorporated in a Camunda transaction.

5 Business Level Rollback

Dealing with rollbacks can also be done explicitly via business process model elements, but note that in Camunda none of these by default interact with the database level transaction system or the Java exceptions. Camunda supports various BPMN elements which is a more efficient notation in comparison than manual error handling, as seen in Figure 2.

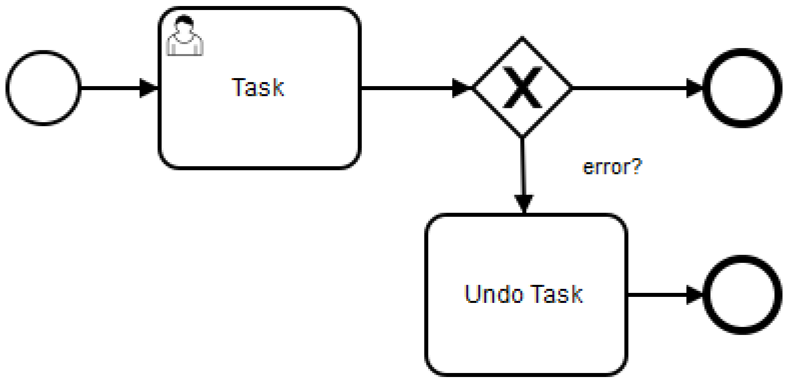

Figure 2 Basic Error Handling

5.1 Error Handling/Event

To start with a more sophisticated way to do simple error handling is with the error event, as shown in Figure 3.

[1] http://www.oracle.com/technetwork/java/javaee/jta/index.html

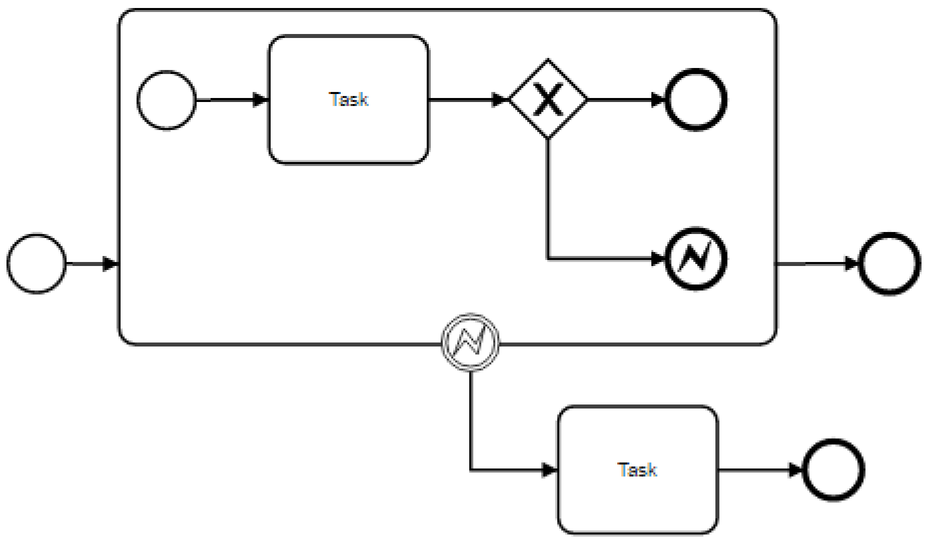

Figure 3 External Error Handling with Error Boundary Event

The external error handling (cf. Figure 3) has the disadvantage that it does not have access to the process variables in contrast to the internal error handling 4 (cf. Figure 4).

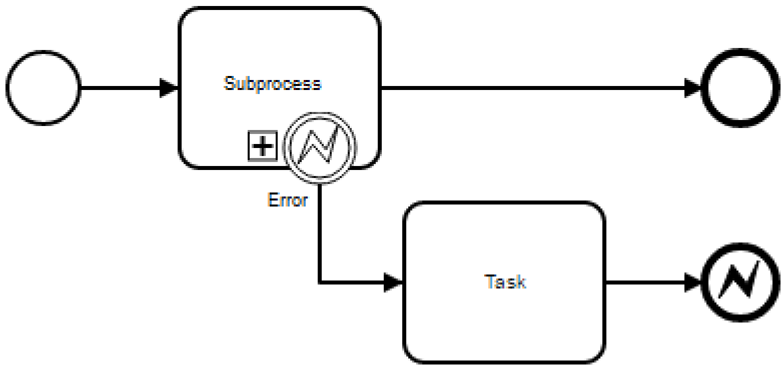

Figure 4 Internal Error Handling with Error Event

5.2 Compensation

While error events deal with irrecoverable situations, compensation is able to undo executed activities (cf. Figure 5). Camunda internally registers a compensation event subscription to the activity whenever the annotated activity (or subprocess) finishes. If the activity was executed successfully multiple times the compensation will be executed an equal amount of times in reversed order.

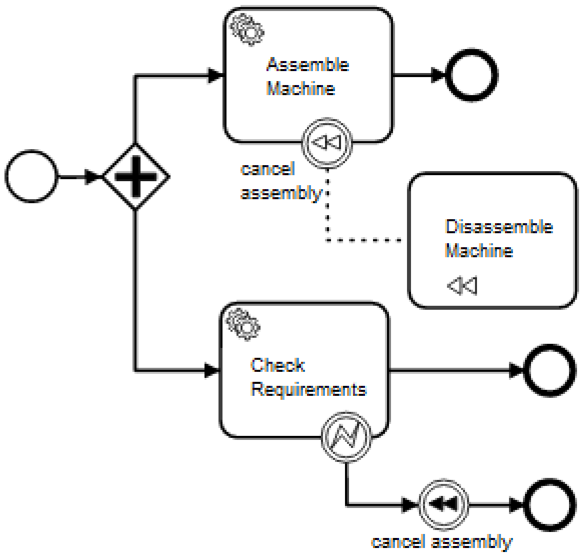

Figure 5 An example for compensation handling.

In addition to compensations handling on the boundaries of activities, compensation handling can also be defined for an entire subprocess via the compensation start event. In contrast to boundary compensation handling, the subprocess compensation start event will consume the compensation event, this means that other compensation events within the subprocess must be triggered manually, as seen in Figure 6.

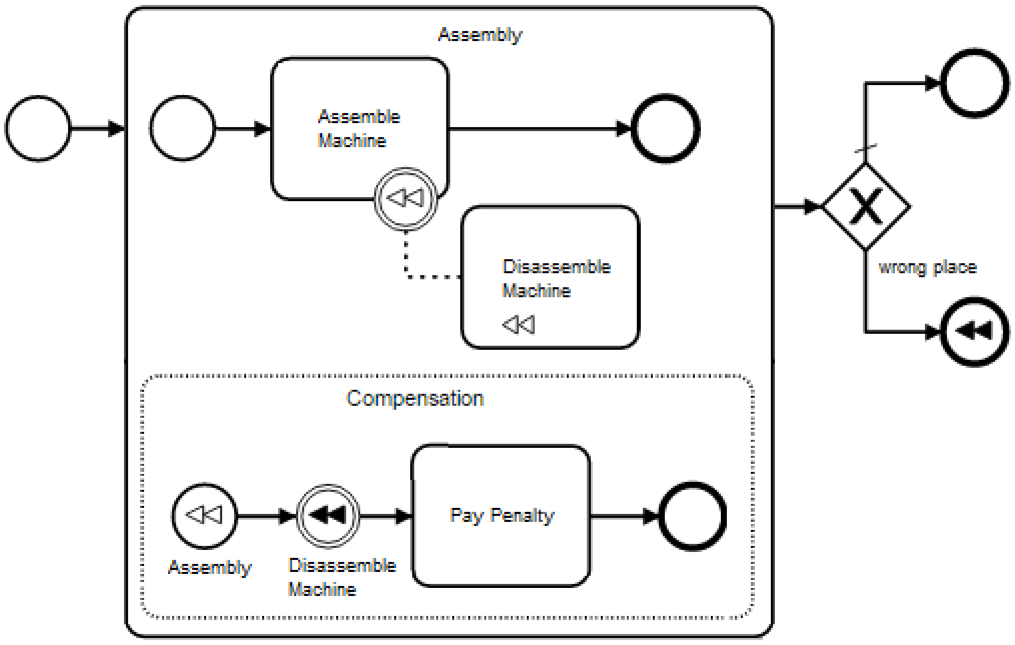

Figure 6 An example for compensation handling with a subprocess.

5.3 Transaction Subprocess

A transaction subprocess (e.g Figure 7) is used for making a group of activities fail or succeed as a whole. There can be three possible outcomes of a transaction;

- the transaction subprocess completes successfully
- the subprocess reaches a cancel end event
- any other event ends the subprocess

It is important to note that, compensation is performed by the transaction subprocess if and only if the cancel end event is reached.

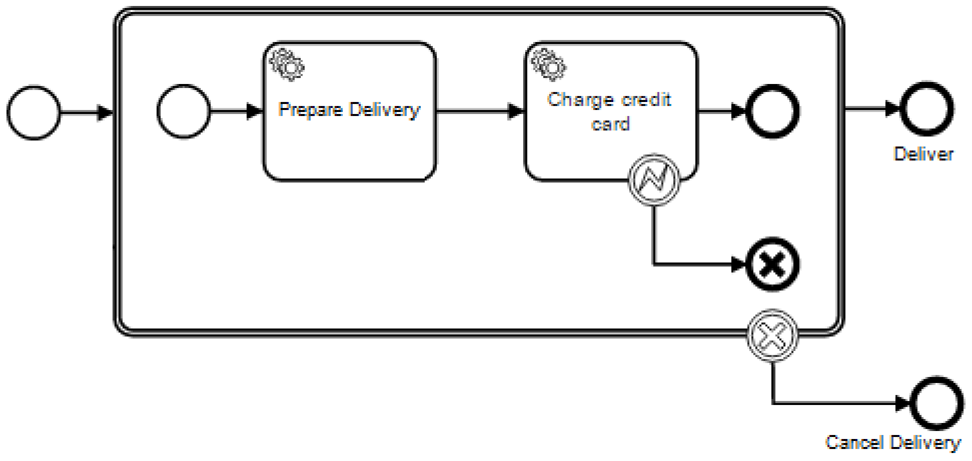

Figure 7 An example transaction subprocess

References

1 Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

2 Michael J Butler, Carla Ferreira, and Muan Yong Ng. Precise modelling of compensating business transactions and its application to bpel. *J. UCS*, 11(5):712–743, 2005.

3 Mati Golani and Avigdor Gal. Flexible business process management using forward stepping and alternative paths. In *International Conference on Business Process Management*, pages 48–63. Springer, 2005.

4 Jing Li, Huibiao Zhu, Geguang Pu, and Jifeng He. Looking into compensable transactions. In *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*, pages 154–166. IEEE, 2007.

5 Fazle Rabbi, Hao Wang, and Wendy MacCaull. Compensable workflow nets. In *International Conference on Formal Engineering Methods*, pages 122–137. Springer, 2010.

6 Daniel Ritter and Jan Sosulski. Exception handling in message-based integration systems and modeling using bpmn. *International Journal of Cooperative Information Systems*, page 1650004, 2016.

7 Nick Russell, Wil van der Aalst, and Arthur ter Hofstede. Workflow Exception Patterns. In *International Conference on Advanced Information Systems Engineering*, pages 288–302. Springer, 2006.

8 Susan D Urban, Le Gao, Rajiv Shrestha, and Andrew Courter. The dynamics of process modeling: New directions for the use of events and rules in service-oriented computing. In *The evolution of conceptual modeling*, pages 205–224. Springer, 2011.

9 Nick RTP van Beest, Pavel Bulanov, Hans Wortmann, and Alexander Lazovik. Resolving business process interference via dynamic reconfiguration. In *International Conference on Service-Oriented Computing*, pages 47–60. Springer, 2010.

10 Yuhai Zhao and Ying Yin. Dynamic self-healing mechanism for transactional business process. *Mathematical Problems in Engineering*, 2015, 2015.