MODELING BPEL WORKFLOWS WITH FAULT HANDLING

Máté KOVÁCS
Advisors: Dániel VARRÓ, László GÖNCZY

I. Introduction

The classical way of administering the activities of an organization involves data that are kept on paper and decisions that are made according to that data. There are also atomic activities that carry out the modifications on work items, thus modifying the information on the paper.

Today these concepts serve as a model of electronically executed business processes. The information on products is stored in databases, and the workflow is executed by a workflow engine. Decisions concerning the activities to be carried out on a work item are made autonomously in many cases in a programmatic way. Business processes are used to coordinate the activities of the organizations, and maintain information on the products.

The correct functionality is crucial to the successful operation of the company, thus business process designers and IT professionals need modeling and verification methodologies to reduce the risk of malfunctioning to a minimum. There are several initiatives that aim to provide a formal modeling technique to workflows, which could contribute to the quality of business processes being enacted. These techniques are usually based on already existing formalisms, such as finite state automata [1], Petri-nets [2] and dataflow networks [3]. However, the fault handling mechanisms are out of the scope of these existing methodologies.

In this paper a business process modeling technique is presented that also deals with the fault and compensation handling features of the Business Process Execution Language (BPEL) [4].

II. The Model of Business Processes

A. The BPEL Language

The Business Process Execution Language (BPEL) is designed to implement business processes. As such, the most important building block of a workflow is the *activity*. *Structured activities* are used to define the control flow of the *basic activities* that carry out modifications on data. Structured activities include *sequence*, *selection*, *iteration* and *parallel execution*.

A workflow implemented in BPEL may be organized into a hierarchy using the concept of *scopes*. Scopes provide a special sort of fault handling for the workflow fragment wrapped by it. According to the terminology used by the standard [4], basic activities may throw *faults* in case of malfunction that are caught by one of the *fault handlers* of the containing scope. The process fragment contained by the fault handler is dedicated to the reversal of the effect of the partially executed workflow fragment of the scope. The effect of a successfully terminated scope may be compensated by its *compensation handler*. The compensation is initialized from outside of the scope.

However, the precise modeling of the semantics defined by the standard is not trivial for several reasons. BPEL provides implicit functionality when either a fault handler or the compensation handler is missing: a snapshot of the state space of the workflow is taken when a scope successfully finishes, and later the compensation handler is run in the snapshot world, not affecting the state space.

B. The Formal Model of the BPEL Language

A BPEL process is mapped into a transition system to represent its states during execution. The initial state of the transition system corresponds to the initialization of the process. Each activity is

mapped into a small automaton depending on its kind. The state space of the workflow model is the composite of that of the constituent automata.

The state spaces of the automata modeling activities have three states in common: *not yet startable, activated*, and *terminated*. The states and state transitions are specific to the actual activity between the states activated and terminated.

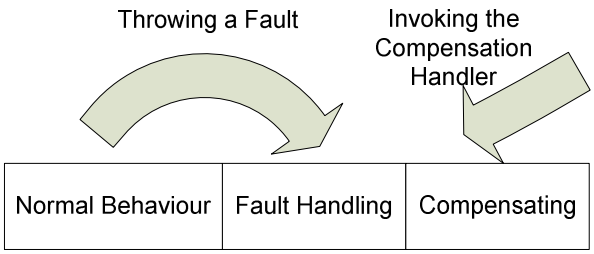

Figure 1: Structure of scopes

Fault handling techniques are closely related to the concept of scopes that have three different running modes with different purpose, namely *normal behavior*, *fault handling* and *compensating*. The finite state automaton modeling a scope maintains this information in its state space as it is shown in Figure 1. Each activity residing in any of the running modes of the scope may be of arbitrary complexity in hierarchy and functionality as well.

First, when the scope is triggered, the activity residing in the region of normal behavior is executed. If an activity in the mode of the normal behavior throws a fault, the running mode is switched into fault handling, and the activity inside the fault handler is triggered. The compensation of a scope may be triggered from the container scope.

III. Conclusion

I have implemented a model transformation to facilitate the practical usage of the business process modeling technique introduced above. The model transformation is implemented as a graph transformation running in the VIATRA2 framework. I have also created the importer of the BPEL language and the exporter of the SAL, which were possible due to the modular architecture of the framework. Furthermore, the metamodels of both of the languages were also created.

Model checking can be carried out on the business process model using the SAL. Requirements against the workflow have to be composed as temporal logical expressions that may also take into consideration the behavior of the process in case of an external fault. SAL is capable of the evaluation of those expressions based on the workflow model.

Our future plans include the extension of the methodology, so that it supports the automatic generation of the temporal expressions representing certain general requirements thus enhancing the usability of the method.

References

[1] J. Koehler, G. Tirenni, S. Kumaran, *From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods* In Proc. *of the 6th International Enterprise Distributed Object Computing Conference*, pp. 96-106, 2002.

[2] Wil van der Aalst and Kees van Hee: *Workflow Management*, The MIT Press 2002.

[3] Máté Kovács and László Gönczy: "Simulation and formal analysis of workflow models", In *Proc. of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques,* pp 215-224, Vienna, AUSTRIA, Elsevier 2006.

[4] *T.* Andrews, F. Curbera et al., *Business Process Execution Language* Version 1.1. IBM, Microsoft, BEA et al. 2003. URL: ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf