Transforming BPEL to Petri Nets

Sebastian Hinz, Karsten Schmidt, and Christian Stahl

Humboldt–Universität zu Berlin,
Institut für Informatik, D–10099 Berlin
{hinz, kschmidt, stahl}@informatik.hu-berlin.de

Abstract. We present a Petri net semantics for the Business Process Execution Language for Web Services (BPEL). Our semantics covers the standard behaviour of BPEL as well as the exceptional behaviour (e.g. faults, events, compensation). The semantics is implemented as a parser that translates BPEL specifications into the input language of the Petri net model checking tool LoLA. We demonstrate that the semantics is well suited for computer aided verification purposes.

keywords: Business process modeling and analysis, Formal models in business process management, Process verification and validation, BPEL, Petri nets.

1 Introduction

The Business Process Execution Language for Web Services (BPEL) is part of ongoing activities to standardize a family of technologies for web services. A textual specification [1] appeared in 2003 and is subject to further revisions. The language contains features from previous languages, for instance IBM’s WSFL [2] and Microsoft’s XLANG [3]. The textual specification is, of course, not suitable for formal methods such as computer aided verification. With computer aided verification, in particular model checking, it would be possible to decide crucial properties such as composability of processes, soundness, and controllability (the possibility to communicate with the process such that the process terminates in a desired end state). For a formal treatment, it is necessary to resolve the ambiguities and inconsistencies of the language which occurred particularly due to the unification of rather different concepts in WSFL and XLANG.

Several groups have proposed formal semantics for BPEL. Among the existing attempts, there are some based on finite state machines [4,5], process algebras [6], and abstract state machines [7,8]. Though all of them are successful in unravelling weaknesses in the informal specification, they are of different significance for formal verification. The semantics based on abstract state machines are feature-complete. However, Petri nets provide a much broader basis for computer aided verification than abstract state machines. Most of the other approaches typically do not support some of BPEL’s most interesting features such as fault, compensation, and event handling.

In this paper, we consider a Petri net semantics for BPEL. The semantics is complete (i.e., covers all the standard and exceptional behaviour of BPEL),
and *formal* (i.e., feasible for model checking). With Petri nets, several elegant technologies such as the theory of workflow nets [9], a theory of controllability [10,11], a long list of verification techniques [12] and tools [13,14,12] become directly applicable. The Petri net semantics provides patterns for each BPEL activity. Compound activities contain slots for the patterns of their subactivities. This way, it is possible to translate BPEL processes automatically into Petri nets. Using high-level Petri nets, data aspects can be fully incorporated while these aspects can as well be ignored by switching to low-level Petri nets.

We first explain the general concepts of BPEL. Afterwards we introduce the principles of our Petri net semantics and explain the Petri net patterns for a few typical BPEL activities. Then we report first experiences with an automated translation of BPEL into Petri nets, and subsequent model checking. Finally, we discuss some ideas for an extension of our technology that aims at models which are better suitable for model checking.

## 2 Introduction to BPEL

BPEL is a language for describing the behaviour of business processes based on web services. Such a business process can be described in two different ways: either as *executable business process* or as *business protocol*. An executable business process which is the focus of this paper models the behaviour and the interface of a *partner* (a participant), in a business interaction. A business protocol, in contrast, only models the interface and the message exchange of a partner. The rest of its internal behaviour is hidden. Throughout this paper, we will use the term *BPEL process* instead of “executable business process specified in BPEL”. Executing a BPEL process means to create an *instance* of this process which is executed.

For the specification of the internal behaviour of a business process, BPEL provides two kinds of *activities*. An activity is either an *elementary activity* or a *structured activity*. The set of elementary activities includes: *empty* (do nothing), *wait* (wait for some time), *assign* (copy a value from one place to another), *receive* (wait for a message from a partner), *invoke* (invoke a partner), *reply* (reply a message to a partner), *throw* (signal a fault) and *terminate* (terminate the entire process instance).

A structured activity defines a causal order on the elementary activities. It can be nested with other structured activities. The set of structured activities includes: *sequence* (nested activities are ordered sequentially), *flow* (nested activities occur concurrently to each other), *while* (while loop), *switch* (selects one control path depending on data) and *pick* (selects one control path depending either on timeouts or external messages). The most important structured activity is a *scope*. It links an activity to a transaction management. It provides a *fault handler*, a *compensation handler*, an *event handler*, correlation *sets* and data *variables*. A *process* is a special *scope*. More precisely, it is the outmost *scope* of the business process.

---

1 We use this type-writer font for BPEL constructs.