WofBPEL: A Tool for Automated Analysis of BPEL Processes*

Chun Ouyang[1], Eric Verbeek[2], Wil M.P. van der Aalst[1,2], Stephan Breutel[1], Marlon Dumas[1], and Arthur H.M. ter Hofstede[1]

[1] Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
{c.ouyang, sw.breutel, m.dumas, a.terhofstede}@qut.edu.au
[2] Department of Technology Management, Eindhoven University of Technology, GPO Box 513, NL-5600 MB, The Netherlands
{h.m.w.verbeek, w.m.p.v.d.aalst}@tm.tue.nl

1 Introduction

The Business Process Execution Language for Web Service, known as BPEL4WS, more recently as WS-BPEL (or BPEL for short) [1], is a process definition language geared towards Service-Oriented Computing (SOC) and layered on top of the Web services technology stack. In BPEL, the logic of the interactions between a given service and its environment is described as a composition of communication actions. These communication actions are interrelated by control-flow dependencies expressed through constructs close to those found in workflow definition languages. In particular, BPEL incorporates two sophisticated branching and synchronisation constructs, namely "control links" and "join conditions", which can be found in a class of workflow models known as *synchronising workflows* formalised in terms of Petri nets in [3].

In the field of workflow, it has been shown that Petri nets provide a suitable foundation for performing static verification. Workflow verification engines such as Woflan [7] are able to analyse Petri net-based workflow models for various purposes such as soundness verification. Therefore, by translating BPEL processes to Petri nets and applying existing Petri net analysis techniques, we can perform static analysis on BPEL processes.

To provide tool support for the analysis of BPEL processes, we developed WofBPEL, and a companion tool BPEL2PNML. BPEL2PNML translates BPEL process definitions into Petri nets represented in the Petri Nets Markup Language (PNML). WofBPEL, which is built using Woflan, performs static analysis on the output produced by BPEL2PNML. Currently it supports three types of analysis: detection of unreachable actions, detection of conflicting message-consuming activities, and metadata generation for garbage collection of unconsumable messages, as detailed in Sect. 2.2.

As part of the design of BPEL2PNML, we formally defined a mapping from BPEL to Petri nets. This mapping is described in [5] and compared with other

* Supported by an Australia Research Council (ARC) Discovery Grant (DP0451092).

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 484–489, 2005.

formalisations of BPEL (see [5]). When surveying previous formalisations of BPEL, we found that none of them had led to a publicly available tool that could be used to perform the types of analysis targeted by WofBPEL over the full set of BPEL control-flow constructs. We also found that previous formalisations of BPEL in terms of Petri nets [4, 6] map control links and join conditions to high-level Petri nets, which are usually less suitable for static analysis of control flow properties than plain Petri nets due to complexity issues. Accordingly, we extended the approach previously sketched in [3] to fully capture control links and join conditions in terms of plain Petri nets. This resulted in a detailed and comprehensive mapping that is more detailed and suitable for the types of analysis targeted by WofBPEL than previous proposals.

WofBPEL and BPEL2PNML are available under an open-source license at `http://www.bpm.fit.qut.edu.au/projects/babel/tools`.

2 Tool Description

2.1 Architecture

Fig. 1 depicts the role of WofBPEL and BPEL2PNML in the analysis of BPEL processes. The BPEL process code may be manually written or generated from a BPEL design tool, e.g. Oracle BPEL Designer. BPEL2PNML takes as input the BPEL code and produces a file conforming to the Petri Net Markup Language (PNML) syntax. This file can be given as input to WofBPEL which, depending on the selected options, applies a number of analysis methods and produces an XML file describing the analysis results. It may also be used as input to general-purpose Petri net analysis tool, e.g. PIPE.[1] In addition, the PNML file obtained as the output from BPEL2PNML also includes layout information, and can thus be used to generate a graphical view of the corresponding Petri nets.

2.2 Automated Analysis

Below we describe the three types of analysis that are currently supported by WofBPEL.

Reachability analysis. Consider the BPEL process definition in Fig. 2 where both the XML code and a graphical representation are provided. During the execution of this process, either A1 or A2 will be skipped because these two activities are placed in different branches of a *switch* activity and in any execution of a *switch* only one branch is taken. Thus, one of the two control links x1 or x2 will carry a negative token. On the other hand, we assume that the join condition attached to activity A3 (denoted by keyword "AND") evaluates to true iff both links x1 and x2 carry positive values. Hence, this join condition will always evaluate to false and activity A3 is always skipped (i.e. it is unreachable).

[1] https://sourceforge.net/projects/petri-net

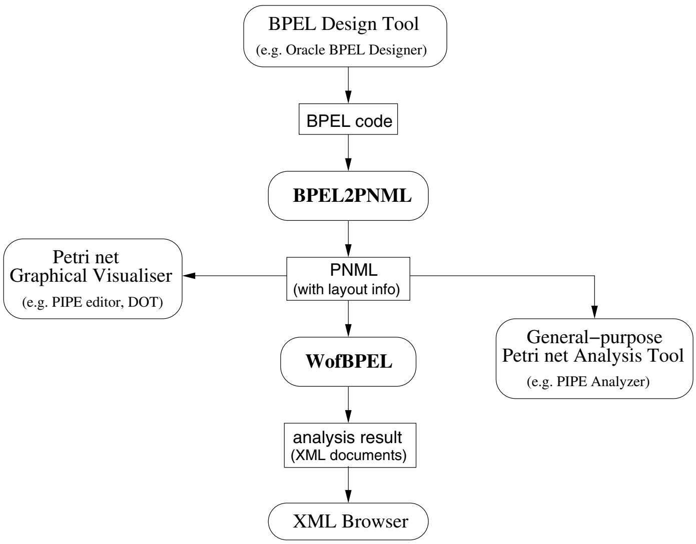

Fig. 1. Analysing BPEL processes using WofBPEL/BPEL2PNML

WofBPEL can detect unreachable activities in a BPEL process such as the one in the previous example. To perform this "unreachability" analysis, WofBPEL relies on two different methods, namely *relaxed soundness* and *transition invariants*. The former is complete but more computationally expensive than the latter. Relaxed soundness [2] takes into account all possible runs to get from an initial state (represented by the marking with one token in the designated input place) to the desired final state (represented by the marking with one token in the designated output place). Every transition which is covered by any of these runs is said to be relaxed sound. On the other hand, transitions that are not covered by these runs are called not relaxed sound. If we assume that the goal of the Petri net is to move from the initial state to the desired final state, then transitions that are not relaxed sound clearly indicate an error, because they cannot contribute in any way to achieving this goal.

However, to check for relaxed soundness we need to compute the full state space of the Petri net, which might take considerable time, especially given the fact that our mapping will generate a lot of parallel behaviour (note that even switch and pick activities are mapped onto parallel behaviour, as the unchosen branches need to be skipped). Therefore, computing relaxed soundness might be a problem.

To alleviate this state space problem, we can replace the relaxed soundness by another property known as transition invariants. Basically, a transition invariant is a multiset of transitions that cancel out, that is, when all transitions from the multiset would be executed simultaneously, then the state would not change. It is straightforward to see that any cycle in the state space has to correspond to some transition invariant. However, not all transitions in the state space will be covered by cycles. For this reason, we add an extra transition that removes a

```
<process name="unreachableTask"
   targetNamespace="http://samples.otn.com"
   suppressJoinFailure="yes"
   xmlns:tns="http://samples.otn.com"
   xmlns:services="http://services.otn.com"
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
 <flow name="FL" suppressJoinFailure="yes">
 <links>
   <link name="x1"/>
   <link name="x2"/>
 </links>
 <switch name="SW">
   <case>
    <invoke name="A1">
     <sources>  <source linkName="x1"/>   </sources>
    </invoke>
   </case>
   <otherwise>
    <invoke name="A2">
     <sources>  <source linkName="x2"/>   </sources>
    </invoke>
   </otherwise>
 </switch>
 <invoke name = "A3">
  <targets>
   <joinCondition>
    bpws:getLinkStatus('x1') and bpws:getLinkStatus('x2')
   </joinCondition>
   <target linkName="x1"/>
   <target linkName="x2"/>
  </targets>
 </invoke>
 </flow>
</process>
```

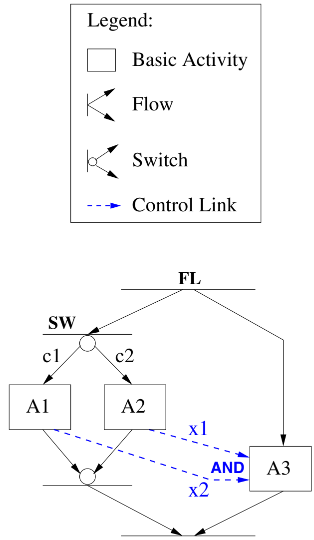

Fig. 2. Example of a BPEL process with an unreachable activity

token from the designated output place and puts a token into the designated input place. As a result, every run from the initial state to the final state will correspond to a transition invariant, and we can use transition invariants instead of relaxed soundness to get correct results. However, the results using transition invariants are not necessarily complete, because transition invariants might exist that do not correspond to runs in the Petri net. This discrepancy is due to the fact that transition invariants totally abstract from states, they more or less assume that sufficient tokens exist to have every transition executed the appropriate number of times.

A summary of the output of WofBPEL for the above example follows:

```
<net file="controlLink03PNML.xml">
  <structure ...
    <Node ... label="inv25 {name=A3}"/>
  </structure>
  <behavior ...
    <Node ... label="inv25 {name=A3}"/>
  </behavior>
  ...
</net>
```

The "structure" element contains the output of the unreachability analysis using the relaxed soundness technique, while the "behavior" element contains the output using the transition invariant technique. In this example, both techniques detect the same set of unreachable nodes in the net, one of which is labelled "inv25 name=A3" (indicating that this node is an "invoke" activity named A3 in the original BPEL process). In fact, we are not aware of any BPEL process definition where the relaxed soundness technique detects unreachable activities that are not detected by the transition invariant technique.

Competing message-consuming activities. The BPEL specification [1] states that "a business process instance MUST NOT simultaneously enable two or more *receive* activities for the same partnerLink, portType, operations and correlation set(s)."[2] In other word, activities that can consume the same type of message may not be simultaneously enabled, where a message type is identified by a combination of a partner link, a port type, an operation, and an optional correlation set. Using the state space-based technique mentioned before, we can check this requirement in a straightforward way. Activities that handle events are receive activities, pick activities, and event handlers. Fig. 3 depicts an example of a process which involves two conflicting receive activities, namely rcv1 and rcv3.

```
<process name="competingMessages01"
       targetNamespace="http://samples.otn.com"
       suppressJoinFailure="yes"
       xmlns:tns="http://samples.otn.com"
       xmlns:services="http://services.otn.com"
       xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <flow suppressJoinFailure="yes">
  <sequence>
   <invoke name="A1" partnerLink="pl1" portType="pt1" operation="op2"/>
   <receive name="rcv1" partnerLink="pl2" portType="pt2" operation="op2"/>
  </sequence>
  <sequence>
    <receive name="rcv2" partnerLink="pl2" portType="pt2" operation="op3"/>
    <receive name="rcv3" partnerLink="pl2" portType="pt2" operation="op2"/>
  </sequence>
  </flow>
</process>
```

Fig. 3. An example of conflicting receive activities

This property can only be checked if the full state space has been generated. For this property, we could alleviate the possible state space problem by using well-known Petri net reduction rules. Except for the transitions that model the receipt of a message, we could try to reduce every place and every transition before generating the state space.

A summary of the output of WofBPEL for the above example follows:

```
<net file="competingMessages02PNML.xml">
  <structure noftransitions="0"></structure>
  <behavior noftransitions="0"></behavior>
  <error description="...">
    <events>
      <event name="rec34 {pL=pl2,pT=pt2,op=op2,name=rcv1}"/>
      <event name="rec37 {pL=pl2,pT=pt2,op=op2,name=rcv3}"/>
    </events>
    <state> ... </state>
    <path> ... </path>
  </error>
  ...
</net>
```

This XML document extract indicates that no unreachable tasks were found but two conflicting message-consuming activities were found. The document provides details of a state (i.e. a Petri net marking) in which a conflict occurs and of a path (i.e. a sequence of states) leading to the problematic state.

[2] For the purposes of this constraint, onMessage branches of a pick activity and event handlers are equivalent to a receive activity.

Garbage collection of queued messages. Again using the full state space, we can compute for each activity a in a BPEL process a set of message types MT_a such that a message type mt is in MT_a iff it is possible in the state space to consume mt after execution of a. In other words, each basic activity a is associated with a set of message types MT_a such that for each $mt \in MT_a$, there exists a run of the process where an activity that consumes a message of type mt is executed after a. Now, consider the situation where activity a has just been executed, a message m is present in the queue, and the type of m is *not* in MT_a. Then message m cannot be consumed anymore (by any activity). Thus, it can be removed from the queue (i.e. it can be garbage collected).

By computing this set for every activity in the BPEL process model, and piggy-backing it in the process definition that is handed over to a BPEL engine, the engine can use this information to remove redundant messages from its queue, thus optimising resource consumption. Specifically, the output of the analysis would be an annotated BPEL process where each basic activity is associated with a set of message types (identified by a partner link, a port type, an operation and optionally a correlation set). After executing an activity a, the BPEL engine could compare the set of message types (MT_a) associated to a with the current set of messages in the queue (M_q) and discard all messages in $M_q \backslash MT_a$.

About the demonstration. The demonstration will show how BPEL processes are mapped to Petri nets using a few representative examples, and will illustrate the above three types of analysis that WofBPEL can perform.

References

1. A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Liu, S. Thatte, P. Yendluri, and A. Yiu, editors. *Web Services Business Process Execution Language Version 2.0*. WS-BPEL TC OASIS, May 2005. Available via `http://www.oasis-open.org/committees/download.php/12791/`.
2. J. Dehnert. *A Methodology for Workflow Modelling: from Business Process Modelling towards Sound Workflow Specification*. PhD thesis, Technische Universität Berlin, Berlin, Germany, August 2003.
3. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, 2003.
4. A. Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services (In German)*. PhD thesis, Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany, 2003.
5. C. Ouyang, H.M.W. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. Technical Report BPM-05-15, BPMcenter.org, 2005. Available via `http://www.bpmcenter.org/reports/2005/BPM-05-15.pdf`.
6. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
7. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnozing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.