A Tool Chain for Lazy Soundness

Frank Puhlmann

Business Process Technology Group
Hasso-Plattner-Institute for IT Systems Engineering
at the University of Potsdam
D-14482 Potsdam, Germany
puhlmann@hpi.uni-potsdam.de

Abstract. This paper introduces a prototypic tool chain to investigate the feasibility of deciding lazy soundness for Business Process Diagrams (BPD). We utilize a graphical editor to create BPDs, export them to XML, convert them to formal π-calculus expressions, and finally use existing π-calculus reasoners to decide lazy soundness.

1 Introduction

Business Process Management (BPM) aims at designing, enacting, managing, analyzing, and adapting business processes [1]. This paper focuses on a special kind of analysis, called *verification*. Verification proves correctness of business processes regarding structural constrains like *deadlocks* or *livelocks* that require a formal semantics of the routing constructs contained in business processes. The correctness criterion investigated is called *Lazy Soundness* [2].

In order to show the feasibility of deciding lazy soundness for business processes we developed a prototypic tool chain. A business process diagrams (BPD) is created graphically using the Business Process Modeling Notation (BPMN) [3]. The BPDs are then exported to an intermediate XML file that provides a generic abstraction from concrete modeling notations. The business processes contained in the XML file can already be checked for structural constraints like connectedness of the nodes. To prove lazy soundness, the XML file is converted to π-calculus expressions. The π-calculus is a generic process algebra that is used to give formal semantics to common patterns of behavior found in business processes [4]. The formalizations are then applied to prove lazy soundness using existing tools.

The remainder of this paper is structured as follows. It starts by introducing the context, i.e. give a definition of lazy soundness and introduce the π-calculus representation of business processes. Thereafter we discuss the architecture of the tool chain and illustrate it using an example. Finally the paper is concluded by discussing related work and further developments.

2 Context

The tool chain is based on the concepts and algorithms for lazy soundness introduced in [2]. The theoretical foundations are given by the π-calculus [5]. While

lazy soundness is a new correctness criterion for the BPM domain, the π-calculus is already in discussion as a formal foundation for BPM [6,7]. Lazy soundness is based on *structural soundness*, informally given by:

> A business process is *structural sound* if and only if there is (a) exactly one initial activity, (b) exactly one final activity, and (c) all activities are on a path from the initial to the final activity.

Furthermore, lazy soundness requires *semantic reachability*, meaning that an activity B is reachable from another activity A (i.e. their exists a path between them) according to the semantics of all other activities such as splits and joins. Lazy soundness is then given by:

> A business process is *lazy sound* if and only if (a) the final activity is semantically reachable from every other activity semantically reachable from the initial activity until the final activity has been executed, and (b) the final activity is executed exactly once.

The definition states that a lazy sound business process is deadlock and livelock free as long as the final activity has not been executed. So called *lazy* activities might still be or become executed. Those are usually required for clean–up or subsequent activities. Examples are activities before a *discriminator* or *n–out–of–m–join* that has already been executed (i.e. receive remaining messages in interacting business processes) or activities triggered by *multiple–instances–without synchronization* patterns [8]. In terms of Petri nets, lazy soundness supports processes where tokens can remain in the net. Again, a detailed discussion can be found in [2].

A formal semantics for business processes is given by the π-calculus. In [4,9] we have shown how different routing patterns are mapped to π-calculus expressions. Basically, each activity of a business process is mapped to a corresponding π-calculus process. The processes then trigger themselves using a pre– and post-condition approach. Reasoning about lazy soundness is done using *weak open bisimulation*. Informally, two π-calculus processes are weak open bisimulation equivalent if they have the same observable behavior regarding certain observability predicates. Weak open bisimulation can be evaluated using existing tools.

3 Architecture

Figure 1 depicts the tool dependencies and document flows in the tool chain. Tools or scripts are shown as rectangles, whereas documents are denoted as notes. The components developed by our group are shown inside the dotted area.

First of all, we utilize a *graphical editor* for designing business process diagrams. The editor is equipped with a set of *BPMN stencils* annotated with additional information. Based on this information, an *XML exporter* script is able to generate an XML description of the business process diagram by interacting with the editor. The *XML* representation of the business process can already

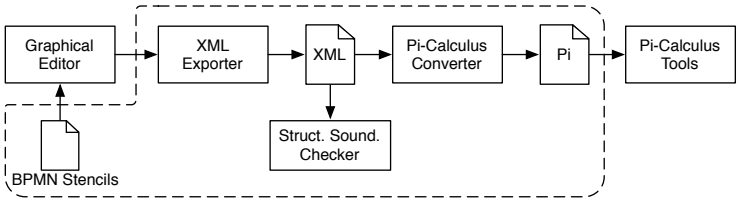


Fig. 1. Architecture of the tool chain.

be proved to be structural sound by a *structural soundness checker* script. Furthermore, it can be used as input for a *pi-calculus converter* script that maps the XML file to a proprietary ASCII notation representing π-calculus processes. The implemented algorithm is described in [2]. The file containing the π-calculus processes can then directly be used as an input for existing *π-calculus tools* for reasoning.

Technically, the feasibility study has been developed on Mac OS X. *OmniGraffle Professional* is utilized as a graphical editor[1], but other editors are also possible. OmniGraffle is fully programmable using *AppleScript* that was used for implementing the XML exporter. Both, OmniGraffle and AppleScript, provide an easy and convenient way of designing and exporting business process diagrams. The π-calculus converter and the structural soundness checker have been implemented as Ruby scripts, so they are OS–independent. The π-calculus tools compatible with our scripts are MWB and ABC, the two major reasoners for π-calculus [10,11]. Both are also available on various platforms.

4 Example

After introducing the theoretical foundations and architecture of the tool chain, we are now ready to give an illustrating example. Figure 2 shows a business process starting with a parallel split, leading to the parallel execution of activities A, B, and C. These activities can represent sub-processes for contacting three different experts for writing an expertise. A *2–out–of–3–join* continues the execution at activity D after two of them are ready. However, some cleanup work is left for the remaining activity, e.g. receiving the last expertise and paying the expert. Activity D spawns of three multiple instances of itself, sending the accepted expertises to three different involved persons. While the expertises are still in delivery, the business process is already finished.

The interesting point regarding lazy soundness are the lazy activities that are left behind. This might be one of A, B, or C, as well as the three instances of D. To prove the business process to be lazy sound, we need to export it from our graphical editor using the XML exporter tool. The tool creates an XML file representing a so called *process graph* of the BPD. A process graph is a

[1] `http://www.omnigroup.com/applications/omnigraffle`

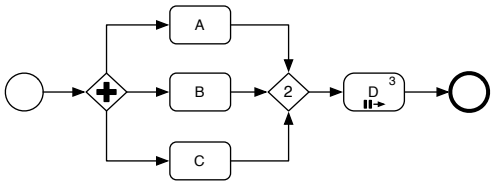


Fig. 2. Example business process diagram.

mathematical structure to describe the static aspects of a business process (see also [2]). The XML representation of the process graph looks as follows:

Example 1 (XML representation of the example).

```
<model>
   <process id="1" type="BPMN">
      <node id="1025" type="MI without Sync" name="D" count="3"/>
      <node id="538" type="End Event"/>
      <node id="748" type="N-out-of-M-Join" continue="2"/>
      <node id="790" type="Task" name="C"/>
      <node id="789" type="Task" name="B"/>
      <node id="717" type="AND Gateway"/>
      <node id="677" type="Task" name="A"/>
      <node id="534" type="Start Event"/>
      <flow id="799" type="Sequence Flow" from="1025" to="538"/>
      <flow id="798" type="Sequence Flow" from="748" to="1025"/>
      <flow id="797" type="Sequence Flow" from="790" to="748"/>
      <flow id="796" type="Sequence Flow" from="789" to="748"/>
      <flow id="795" type="Sequence Flow" from="677" to="748"/>
      <flow id="794" type="Sequence Flow" from="717" to="790"/>
      <flow id="792" type="Sequence Flow" from="717" to="789"/>
      <flow id="791" type="Sequence Flow" from="717" to="677"/>
      <flow id="671" type="Sequence Flow" from="534" to="717"/>
   </process>
</model>
```

Using the structural soundness checker script, the process graph contained in the XML file can be proved to be structural sound (omitted here). The dynamic aspects of the business process are generated out of the type descriptions for each node contained in the XML file by the π-calculus converter. The formal description is furthermore enhanced with lazy soundness annotations as well as a special process called S_{LAZY} used for reasoning later on:

Example 2 (π-calculus representation of the example).

```
agent N1025(e798,e799)=e798.(t.0 | t.0 | t.0 | 'e799.0 | N1025(e798,e799))
agent N717(e671,e794,e792,e791)=e671.t.('e794.0 | 'e792.0 | 'e791.0 |
      N717(e671,e794,e792,e791))
agent N677(e791,e795)=e791.t.('e795.0 | N677(e791,e795))
agent N534(e671,i)=i.t.'e671.0
agent N538(e799,o)=e799.t.'o.N538(e799,o)
agent N748(e797,e796,e795,e798)=(^h,run)(N748_1(e797,e796,e795,e798,h,run) |
      N748_2(e797,e796,e795,e798,h,run))
agent N748_1(e797,e796,e795,e798,h,run)=e797.'h.0 | e796.'h.0 | e795.'h.0
agent N748_2(e797,e796,e795,e798,h,run)=h.h.'run.h.N748(e797,e796,e795,e798) |
      run.t.'e798.0
agent N790(e794,e797)=e794.t.('e797.0 | N790(e794,e797))
agent N789(e792,e796)=e792.t.('e796.0 | N789(e792,e796))
agent N(i,o)=(^e799,e798,e797,e796,e795,e794,e792,e791,e671)(N1025(e798,e799) |
```

```
      N717(e671,e794,e792,e791) | N677(e791,e795) | N534(e671,i) | N538(e799,o) |
      N748(e797,e796,e795,e798) | N790(e794,e797) | N789(e792,e796))
agent S_LAZY(i,o)=i.t.'o.0
```

The input style generated corresponds to MWB as well as ABC. Each node of the XML file has been mapped to a π-calculus process (denoted as *agent* in the syntax). For instance, the initial node is given by $N534$, or the 2–out–of–3–join by $N748$. Helper agents are denoted with an index, like 748_1. BPMN sequence flows have been mapped to π-calculus names, representing dependencies between the agents. For instance, $N717$ can only start after $N534$ has emitted the name $e671$ (an agent emits a name using $'name$ and receives a name by simply stating it, i.e. $name$). To make reasoning possible, all agents representing nodes are placed in parallel in agent N. For accuracy, the identifiers provided by the graphical editor are used. The generated agents can now be imported into existing π-calculus reasoners such as MWB:

```
 The Mobility Workbench
 (MWB'99, version 4.136, built Fri Apr  7 16:02:07 2006)
1
MWB>input "agents.mwb"
MWB>weq N(i,o) S_LAZY(i,o)
The two agents are equal.
Bisimulation relation size = 317.
```

The first statement imports the π-calculus process definitions. Lazy soundness can now be decided using weak open bisimulation between process $N(i, o)$ and $S_{LAZY}(i, o)$. The parameters i and o can be observed for deciding whether the business process is started (by observing i) or the final activity is reached (by o). If o is not observed exactly once, the process is not lazy sound. S_{LAZY} is already proved to be lazy sound, since it simply receives i one time and emits o one time. The *weq* statement now checks if $N(i, o)$ equals S_{LAZY} regarding the observable behavior. As both are equal, also $N(i, o)$ is lazy sound. Interestingly, components of $N(i, o)$ representing lazy activities are still active. However, they do not trigger the final activity (the one that emits o) again.

A counterexample can be given by modifying the parallel split of figure 2 to an exclusive decision. This results in a change of agent $N717$ of the π-calculus representation:

```
agent N717(e671,e794,e792,e791)=e671.t.('e794.N717(e671,e794,e792,e791) +
         'e792.N717(e671,e794,e792,e791) + 'e791.N717(e671,e794,e792,e791))
```

Now, either activity A, B, or C are activated. As can easily be deduced, this leads to a deadlock since the 2–out–of–3–join expects at least two activities to be finished beforehand. By asking MWB using the changed agent $N717$ this can be proved:

```
MWB>weq N(i,o) S_LAZY(i,o)
The two agents are NOT equal.
```

Hence, the modified business process is not lazy sound.

Drawbacks. During early experiments using MWB and ABC for deciding lazy soundness of different business processes, we already discovered several issues. First of all, weak open bisimulation is undecidable in general. Thus, some inputs will not give a result. To make matters worse, current implementations of MWB and ABC rely on depth first search, wasting computing power where breadth

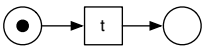
first search would already disprove lazy soundness (i.e. finding other paths that lead to deadlocks and livelocks). However, small to mid-size processes can be proved in reasonable time (see [2] for timing results). Furthermore, due to the non–local semantics of the synchronizing merge pattern (or–join), business processes containing this pattern are never lazy sound. As a concluding remark, also backtracking of errors found in the π-calculus representation to the graphical notation is currently quite difficult. Using optimized reasoners and enhancing the π-calculus representation with additional debugging information, most of the problems can be solved.

5 Related Work

An important piece of related work is Woflan (`http://is.tm.tue.nl/research/woflan.htm`). Woflan is able to prove if two Petri nets are in a certain inheritance relation [12]. Most interesting is checking for *projection inheritance*, that has been derived from process algebra [13]. An informal description is as follows:

> "If it is not possible to distinguish the behaviors of x and y [x and y are Petri nets] when arbitrary tasks of x are executed, but only the effects of tasks that are also present in y are considered, then x is a subclass of y." [12].

Hence, y represents the S_{LAZY} process and x an arbitrary Petri net to check for conformance. S_{LAZY} is given as a Petri net consisting of two places and a transition t:



The transition t can be enhanced with arbitrary process structures. Since projection inheritance ignores remaining tokens in the Petri net, lazy soundness for Petri nets can be proved using Woflan. However, just as with ABC and MWB for π-calculus, the only feedback is a yes/no answer. Furthermore, using Petri nets for proving business processes to be lazy sound has two major drawbacks. First of all, not all workflow patterns can be represented in low-level Petri nets [14]. Thus, the number of possible business processes is restricted. Second, branching bisimilarity used for projection inheritance does not take into account link passing mobility. Link passing mobility is used inside service oriented architectures to represent dynamic binding of interaction partners [9]. Since weak open bisimulation supports link passing mobility, lazy soundness can be extended to interaction soundness. Interaction soundness proves an orchestration to be (lazy) sound regarding also its interactions inside a choreography. Since not all of the interaction partners are statically known (i.e. connected) to the orchestration at design–time, but instead are bound at run–time, a bisimulation technique based on link passing mobility is required.

6 Conclusion

In this paper we introduced a first prototypic tool chain to show the feasibility of deciding lazy soundness using π-calculus. In order to evaluate the tool chain, the scripts and examples are provided at `http://pi-workflow.org`. While the graphical editing and export is currently OS depended (Mac OSX 10.4 required), the conversion of the examples to π-calculus and reasoning runs on a variety of platforms. The very next step regarding the tool chain is to create a stable implementation. This implementation can then be used to analyze existing π-calculus tools as well as the proposed pattern formalizations for conformance regarding lazy soundness.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M.: Business Process Management: A Survey. In van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M., eds.: Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS, Berlin, Springer-Verlag (2003) 1–12
2. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS, Berlin, Springer Verlag (2006) 145–160
3. BPMI.org: Business Process Modeling Notation. 1.0 edn. (2004)
4. Puhlmann, F., Weske, M.: Using the Pi-Calculus for Formalizing Workflow Patterns. In van der Aalst, W., Benatallah, B., Casati, F., eds.: Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS, Berlin, Springer-Verlag (2005) 153–168
5. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I/II. Information and Computation **100** (1992) 1–77
6. Smith, H., Fingar, P.: Business Process Management – The Third Wave. Meghan-Kiffer Press, Tampa (2002)
7. Puhlmann, F.: Why do we actually need the Pi-Calculus for Business Process Management? In Abramowicz, W., Mayr, H., eds.: 9th International Conference on Business Information Systems (BIS 2006), volume P-85 of LNI, Bonn, Gesellschaft für Informatik (2006) 77–89
8. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.: Workflow Patterns. Technical Report BETA Working Paper Series, WP 47, Eindhoven University of Technology (2000)
9. Overdick, H., Puhlmann, F., Weske, M.: Towards a Formal Model for Agile Service Discovery and Integration. In Verma, K., Sheth, A., Zaremba, M., Bussler, C., eds.: Proceedings of the International Workshop on Dynamic Web Processes (DWP 2005). IBM technical report RC23822, Amsterdam (2005)
10. Briais, S.: ABC Bisimulation Checker. Available at: `http://lamp.epfl.ch/~sbriais/abc/abc.html` (2003)
11. Victor, B., Moller, F., Dam, M., Eriksson, L.H.: The Mobility Workbench. Available at: `http://www.it.uu.se/research/group/mobility/mwb` (2005)
12. van der Aalst, W., Basten, T.: Inheritance of Workflows: An approach to tackling problems related to change. Computing science reports 99/06, Eindhoven University of Technology, Eindhoven (1999)

13. Basten, T.: In Terms of Nets: System Design with Petri Nets and Process Algebra. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (1998)
14. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language (Revised version. Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane (2003)