Visualizing the Soundness of Workflow Nets

Christian Flender[1], Thomas Freytag[2]

[1] Queensland University of Technology
126 Margaret Street, QLD 4000, Australia
[2] University of Cooperative Education (Berufsakademie)
Erzbergerstrasse 121, 76133 Karlsruhe, Germany

[flender | freytag]@woped.org

Petri nets and in particular workflow nets as a process description language allow the application of formal verification methods, e.g., to prove soundness. Although tools for soundness checks do already exist, it is still difficult for the workflow designer to correct the process model according to the output of these tests – mostly because diagnostic messages are not directly linked to the graphical model and intuition of the error source is missing. In this paper we propose an idea to overcome this problem. We present concepts to visualize verification output, in particular soundness violation messages in a more user-friendly way.

1 Introduction

Modern information system architectures allow the direct enactment of business processes designed on a conceptual level [DuAa+05]. In order to prevent execution errors at runtime, build-time verification mechanisms as part of process definition tools have become more and more essential. Petri nets [Petr62] and in particular workflow nets [Aals98] [AaHe02] provide a strong mathematical fundament for modeling and analyzing business processes in this sense. Their formal nature allows tool-supported checks, *e.g.*, for soundness, an important property of workflow processes. Sound business processes can be considered as correct in terms of structure and behavior. Although software tools for verification and diagnosis already exist, non-soundness-versed designers currently are not well-supported in correcting process models violating soundness. Woflan [Verb04] is a soundness verification tool able to interact with process modeling products, *e.g.*, COSA [Cosa06], YAWL[1] [Yawl06] or WoPeD [Wope06]. Currently Woflan's diagnosis results appear in textual format as lists of violated properties, providing no direct references to the underlying graphical model and making it difficult to locate the error source causing unsoundness. This paper tries to overcome these problems by providing a user-friendly visualization concept for soundness violation messages, giving the designer a deeper insight in the nature of soundness-violating modeling errors. The paper is structured as follows: In section 2, the soundness algorithm and its verification output are roughly sketched. Section 3 presents a concept how to visualize soundness violation messages. In section 4, implementation issues are discussed and finally an outlook to the future is given.

2 The soundness algorithm

A sound WF-net has to satisfy three requirements [Aals97]: Firstly, it must have the *option to complete*, *i.e.*, each execution starting in *M[i]*[2] finally leads to *M[o]* and thus the short-circuited[3] WF-net is live. Secondly, *proper completion* is required, *i.e.*, the only terminating state is *M[o]* and thus the short-circuited WF-net is bounded. Thirdly, *no dead transitions* may be present, *i.e.*, each task has to contribute to the process purpose. An algorithm for checking soundness was published most comprehensively in [Verb04] and successfully implemented by Woflan [Wofl06]. Woflan performs an elaborated sequence of analytic steps within a complex algorithm. As soon as a net is inevitably unsound, verification output is generated and the underlying error causes are listed in an error report for correction hints. These reports are usually difficult to read and fully understandable only by process designers with a deeper insight into the algorithmic details.

[1] YAWL interacts with WofYAWL, a verification module based on Woflan
[2] *M[i]* is the marking putting one token on the starting place and no tokens elsewhere, *M[o]* is the marking putting one token on the end place and no tokens elsewhere
[3] This means that the end place is connected to the starting place by an additional "virtual" transition (*cf.* [Aals97]).

In order to improve the interpretation of these error reports and to simplify (un)soundness diagnostics, we first of all propose to divide the verification process into five distinct *soundness violation classes*, each of them defining one major stage of Woflan's soundness algorithm.

Each class represents a Boolean answer to one of the following questions:

- Class 1: "*Is my process definition a WF-net?*"
- Class 2: "*Is my WF-net interim sound?*"
- Class 3: "*Is my WF-net bounded?*"
- Class 4: "*Is my non-s-coverable WF-net live?*"
- Class 5: "*Is my s-coverable WF-net live?"*

Using this rough classification, the modeler becomes aware of the verification progress and thus should always be able to keep track of those properties already satisfied at the point of unsoundness notifications. Additionally, the violated soundness criteria can be derived easily from the reported class number. For instance, in class 5, a WF-net is bounded (here proven by S-coverability) and not live and thus unsound.

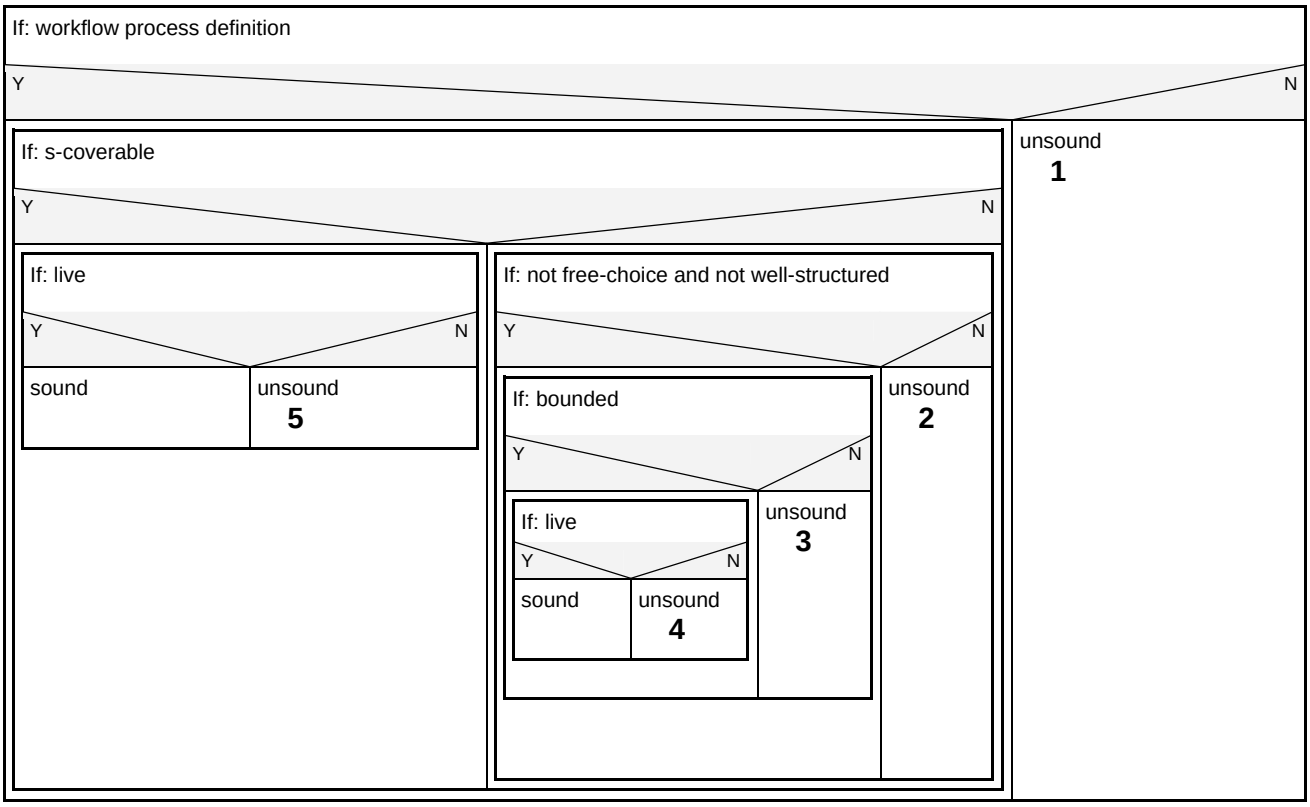

Figure 1. Structogram of the soundness algorithm

Soundness violations often are directly caused by incorrect usage or linkage of single nodes or sets of nodes inside the process model. For example, a node is connected to the rest of the net in a way not fulfilling the WF-net property or a dead transition is causing a livelock.

Violation class	Possible violation sets
1 (WF-net)	Inappropriate start or end places Nodes that are not strongly connected
2 (Interim Soundness)	Places not covered by S-components Mismatch or confusion sets
3 (Boundedness)	Unbounded places Unbounded sequences
4 and 5 (Liveness)	Dead transitions Non-live transitions Non-live sequences

Table 1. Violation classes mapped to violation sets

In the context of our visualization concept, the notion of a *violation set* is defined as one or more nodes in the graphical model which can be considered as (potential) "reason(s)" for a specific soundness violation. Table 1 shows how the five soundness violation classes mentioned above can be associated to potential violation sets.

3 A soundness visualization concept

In this section we present ideas how to present soundness verification results in a user-friendly way to the modeler by visualizing the associated violation sets inside the graphical process model. Figure 2 illustrates a suggested graphical representation of violation sets. Errors caused by single-node violation sets are visualized in black. Unordered violation sets consisting of more than one node are colored in grey. Violation sets which are an (ordered) sequence of nodes appear in light grey. Additional verification output information appears as encoded node inscriptions on the associated violation set elements. The first letter of the inscription is the violation class number (1…5), the following letters stand for the abbreviation of the error detail, *e.g.*, **`dt`** for "dead transition" and so on. For instance, a node of a WF-net which is not bounded (class 3) is colored in black with the inscription **`3up`** (**`up`** = error reason "unbounded place").

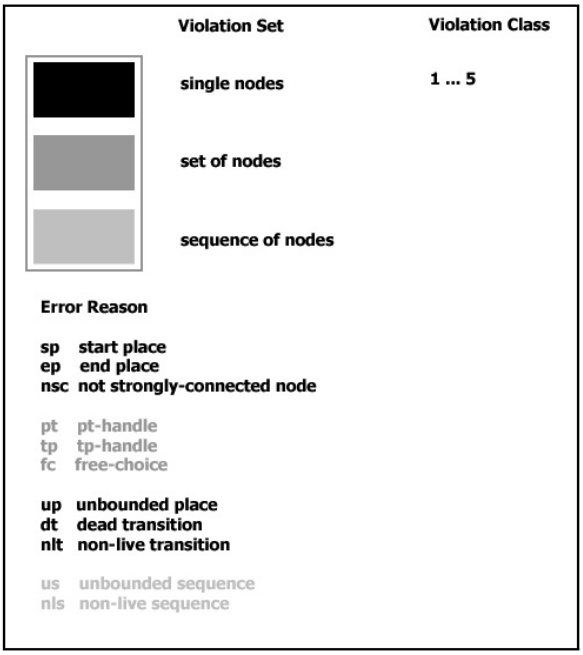

Figure 2. Soundness violation visualization

3.1 Visualization of class 1 violations

A WF-net must have exactly one input place and one output place and (if this is the case) all nodes have to be strongly connected. If this does not hold, the verification output consists of one or more single nodes not satisfying these requirements. In Figure 3 the process model is not a WF-net since it has more than one input (and also more than one output) place and the transition **`verify`** is not strongly connected to all other nodes. The visualization consists of black nodes with associated inscriptions for all improper nodes.

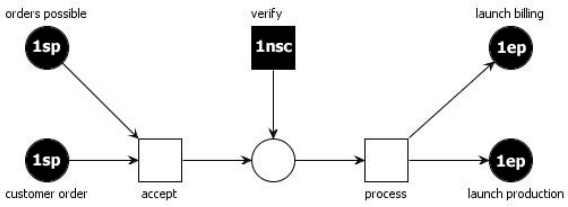

Figure 3. Example visualization of violation class 1 (WF-net)

3.2 Visualization of class 2 violations

Violation class 2 indicates that a process model is a WF-net but not *interim sound*. Interim soundness is a term for the property that a non-s-coverable net contains confusions (*i.e.*, is not free-choice) and mismatches (*i.e.*, has PT-handles or TP-handles). A non-interim sound net cannot be sound[4]. Figure 4 illustrates a set of nodes being part of a mismatch (in this case a PT-handle) causing the (non-s-coverable and confusion-free) WF-net not to be interim sound and thus unsound. The visualization displays all involved nodes in grey color with inscription **`2pt`** standing for violation class 2 and (the potential) error reason "PT-handle".

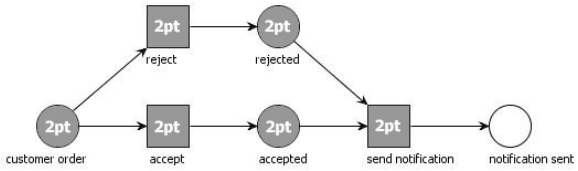

Figure 4. Example visualization of violation class 2 (PT-handle)

3.3 Visualization of class 3 violations

Violation class 3 stands for those WF-nets which are not sound because the short-circuited net is unbounded and thus the process does not terminate properly. Typically, this situation is caused by one or more unbounded places. Apart from this, the modeler should be informed under which circumstances the unbounded situation may occur. A second violation set (a sequence of nodes) is displayed, showing the associated firing sequence leading to the unwanted situation. Figure 5 shows the two violation sets in the color scheme defined above (black = unbounded place, grey = associated firing sequence). Note that in this example not all executions lead to unboundedness. Firing transition **`launch production`** instead of **`launch billing`** would make the net behave properly.

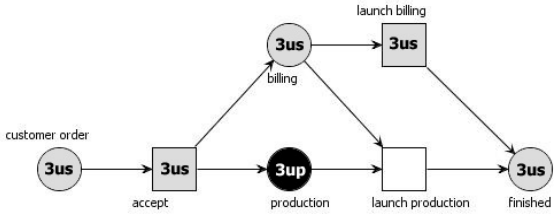

Figure 5. Example visualization of violation class 3 (unboundedness)

3.4 Visualization of class 4 violations

Violation class 4 handles the situation where a short-circuited WF-net is bounded but non-live, *i.e.*, it has no option to complete. Non-liveness of a WF-net is usually caused by the existence of either dead transitions or non-live transitions. Figure 6 shows an example WF-net with transition **`send notification`** being obviously dead, *i.e.*, being not able to fire in any execution. The associated visualization has to mark the transition symbol as a class 4 error with a single node violation set, *i.e.*, in black color.

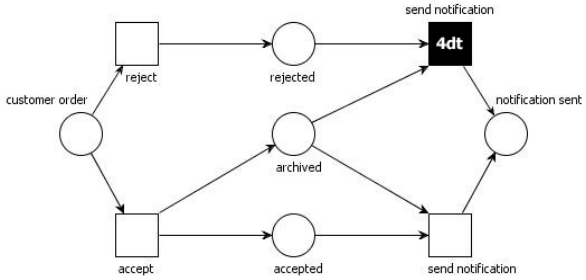

Figure 6. Example visualization of violation class 4 (deadlock caused by a dead transition)

[4] This can be derived from the fact that that a sound free-choice WF-net or a sound well-structured WF-net is always s-coverable [VeBa+01]

Whereas dead transitions are single node violation sets, soundness violations based on non-live transitions are more difficult to visualize. A sequence of nodes is needed to symbolize the detailed history of the non-live situation. Figure 7 visualizes the firing sequence **`accept - launch billing,`** causing the non-liveness of transition **`launch production`**. Note that this transition is not dead since there is another firing sequence which is enabling it. The visualization is done here by highlighting the associated firing sequence in light grey color.

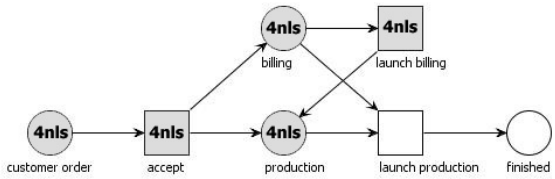

Figure 7. Example visualization of violation class 4 (deadlock caused by a non-live transition)

3.5 Visualization of class 5 violations

This last error class is very similar to class 4. The difference is that in class 5, boundedness has previously been verified by covering the net with S-components rather than by other techniques (*cf.* algorithm in Figure 1). Figure 8 shows an example of an s-coverable WF-net with a non-live sequence. The visualized violation set is a sequence of nodes representing the firing of transitions towards this situation, inscribed with the error reason **`nls`** (= "non-live sequence").

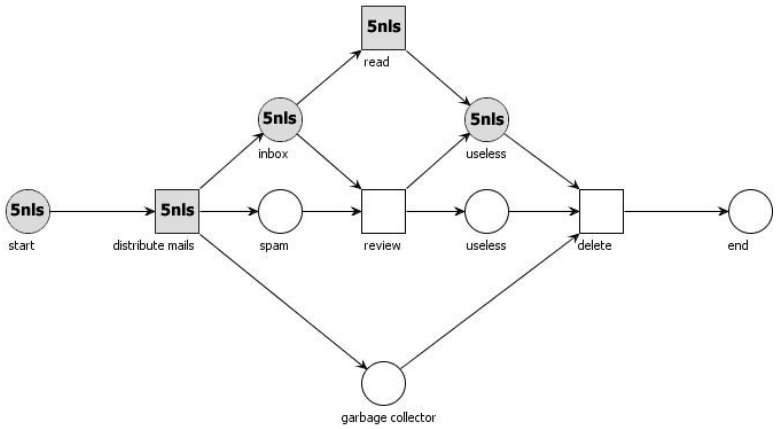

Figure 8. Example visualization of violation class 5 (non-liveness)

4 Implementation issues

The next important step will be to implement the suggested visualization concepts into the open-source software WoPeD [Frey03] which provides a simple calling interface to Woflan[5] and is able to access the verification output on source-code level. This allows the graphical mapping of violation classes and violation sets to the graphical model inside the WoPeD editor window in the following way:

- Single node violations are displayed as highlighted elements with the inscription showing class number and error reason code
- Unordered violation sets are displayed as a set of highlighted elements, inscribed by the same error reason code
- Ordered violation sets (usually representing firing sequences) could either be visualized as numbered highlighted elements denoting their execution sequence or as a small animated "video sequence" showing the token game development of the associated error situation

[5] Because Woflan currently is only available for Windows platforms, this feature is only accessible when running WoPeD under Windows.

The prototypical implementation of these features is still work in progress. Important prerequisite steps like interfacing with Woflan have already been completed and the main focus can be put on realizing a graphical representation of soundness violations classified and visually encoded in the discussed way.

5 Conclusion and outlook

This paper has proposed and sketched a concept to support workflow process modelers in their task to create sound process definitions by allowing the presentation of diagnostic messages in a user-oriented way. This is done by using an intuitive visualization method for typical soundness violation situations reported by verification tools like Woflan in a textual, sometimes hard-to-read format. A classification of error situations has been given as well as some basic ideas how to map them to the graphical process model in terms of violation classes, violation sets and error reasons. The discussed features will soon be implemented in the workflow net tool WoPeD. Another interesting question for the future would be how to describe verification output in a more general, tool-independent way. An XML-based language could be thought of to be defined, possibly as an extension to PNML [WeKi03].

6 References

[AaHe02] VAN DER AALST, W.M.P; VAN HEE, K.: *Workflow Management - Models, Methods, Systems.* MIT Press, Cambridge, Massachusetts, 2002.

[Aals97] VAN DER AALST, W.M.P.: *Verification of Workflow Nets.* In: P. Azéma and G. Balbo (Eds.): Application and Theory of Petri Nets, LNCS 1248, pages 407-426. Springer-Verlag, Berlin, 1997.

[Aals98] VAN DER AALST, W.M.P.: *The Application of Petri Nets to Workflow Management.* The Journal of Circuits, Systems and Computers, 8(1):21-66, 1998.

[Cosa06] COSA: *http://www.cosa-bpm.com*, 08.2006.

[DuAa+05] DUMAS, M.; VAN DER AALST, W.M.P.; TER HOFSTEDE, A.: *Process-Aware Information Systems. Bridging People and Software Through Process Technology.* John Wiley & Sons, 2005.

[Frey03] FREYTAG, T.: *PWFtool - a Petri net workflow modelling environment.* Research Report, Catholic University of Eichstaett, 2003.

[HaVe+97] HAUSCHILD, D.; VERBEEK H.M.W.; VAN DER AALST, W.M.P.: *WOFLAN: a Petri-net-based Workflow Analyzer.* Computing Science Reports 97/12, Eindhoven University of Technology, Eindhoven, 1997.

[Petr62] PETRI, C.A.: *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962.

[VeBa+99] VERBEEK H.M.W.; BASTEN, T.; VAN DER AALST; W.M.P.: *Diagnosing Workflow Processes using Woflan.* Computing Science Report 99/02, Eindhoven University of Technology, Eindhoven, 1999.

[Verb04] VERBEEK, H.M.W.: *Verification of WF-nets.* Technische Universiteit Eindhoven, Dissertation, 2004.

[WeKi03] WEBER, M.; KINDLER, E.: *The Petri Net Markup Language.* In: H. Ehrig, W. Reisig, G. Rozenberg, H. Weger (Eds.): Petri Net Technology for Communication Based Systems, LNCS 2472, Springer Verlag, Berlin, 2003.

[Wofl06] WOFLAN: *Workflow Analyzer. http://is.tm.tue.nl/research/woflan*, 08.2006.

[Wope06] WOPED: *Workflow Petri Net Designer. http://www.woped.org*, 08.2006.

[Yawl06] YAWL: *Yet another workflow language. http://www.yawl.fit.qut.edu.au*, 08.2006.