The Petri Net API

A collection of Petri net-related functions

Niels Lohmann, Stephan Mennicke, and Christian Sura

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
pnapi@service-technology.org

Abstract. This paper introduces the Petri Net API, a C++ library of Petri net-related functions. The Petri Net API is currently used in more than a dozen Petri net tools, ranging from compilers to verification tools.

1 Introduction

Algorithms to reason about the correctness of distributed systems usually have devastating worst-case complexities. Nevertheless, experiences in the field of model checking show that the feared state space explosion can be alleviated by state space reduction techniques or symbolic representations [5]. Therefore, novel techniques to verify correctness often require a proof-of-concept implementation to conduct experiments or to demonstrate feasibility on realistic input data. We recently investigated the academic software development process [14] and claimed that single purpose tools have the right granularity to be implemented using rapid prototyping techniques. A prerequisite for this is the encapsulation of frequently recurring functionality into reusable libraries to avoid an unnecessary "reinvention of the wheel" and to minimize the amount of untested ad-hoc code.

This paper introduces with the Petri Net API such a reusable library. It was originally derived from the back-end of the compiler BPEL2oWFN [12] and offered simple net management functionality and the file output in several formats. In the last years, the functions of the Petri Net API have been revised and collected into a consistent C++ library. The main focus of the API is to *organize* Petri nets, rather than to implement verification algorithms (i.e., to build and analyze state spaces) or to provide a graphical front-end. We claim that these tasks should be part of a dedicated tool rather than a library that is designed to be shared by several tools. As of September 2010, the Petri Net API is used by 14 tools, see `http://service-technology.org/tools` for an overview.

The rest of this paper is organized as follows. The next section presents the features that are implemented by the Petri Net API. Then Sect. 3 shows how the Petri Net API can be used in novel tools, discusses its availability, and the integration in third-party tools. Section 4 presents a small case study in which the Petri Net API is used to perform some structural modifications to check a correctness notion for workflow nets. Section 5 briefly compares the Petri Net API to existing frameworks, before Sect. 6 concludes the paper.

2 Features

The Petri Net API implements the following features to organize Petri nets.

- Petri net creation and manipulation (creation, modification, deletion, copying, and search of nodes and arcs),
- file import and export of Petri nets in various file formats (PNML [9], LoLA [22] file format, Fiona [15] open net format, Woflan [21] workflow nets),
- generation of graphical representation using Graphviz dot,
- efficient application of structural reduction rules [17,19,18],
- structural checks (e.g., workflow net structure, free-choice property),
- import from automata (using the region theory tools Petrify [6] or Genet [3]),
- export to automata,
- support for open nets [23] (ports, net composition)
- organization of final markings, and
- support for role annotations.

The Petri Net API can be easily extended to new features. As of now, we only moved features from a tool into the Petri Net API when this feature is used by at least one other tool. This avoids a cluttered API full of too specific functions. At the same time, it ensures a high test case coverage of the features.

3 Usage

Integration as C++ library. The API itself is written in C++ and can be integrated into other tools with no more effort than including a header file. Listing 1.1 shows example code to read a file in PNML format, structurally reduce it using the Murata rules [17], and output a graphical representation.

Listing 1.1. Example using the Petri Net API as C++ library.

```
#include <pnapi/pnapi.h>
using namespace pnapi;

int main() {
  // read PNML net from file
  std::istream in("file.pnml", std::ios_base::in);
  in >> io::pnml >> net;

  // apply reduction rules
  net.reduce(PetriNet::SET_MURATA);

  // output the Petri net in Graphviz dot format
  std::cout << io::dot << net;

  return 0;
}
```

The Petri Net API complies with the 1998 ANSI/ISO C++ standard and can be compiled on many platforms, including Microsoft Windows, Sun Solaris, GNU/Linux, Mac OS X, and other UNIX-based operating systems.

Command-line tool. Beside the direct integration, we implemented a command-line tool `petri` (part of the Petri Net API distribution) for the most common operations. The call

```
petri *.pnml --input=pnml --reduce=murata --output=png
```

performs a similar transformation as the code in Listing 1.1, but is also able to read multiple files and to directly create graphics files. This front-end tool is very useful in shell scripts to process large libraries of nets.

Integration into third-party tools. The Petri Net API is currently indirectly (as front-end tool or library) integrated into the business process modeling tool Oryx [7], the process mining toolkit ProM [20], and the YAWL workflow editor [2]. In all tools, the Petri Net API organizes the exchange of Petri Net models in PNML format.

3.1 Availability

The Petri Net API is free open source software, licensed under the GNU LGPL 3.[1] The most recent version together with its manual can be downloaded at `http://service-technology.org/pnapi`.

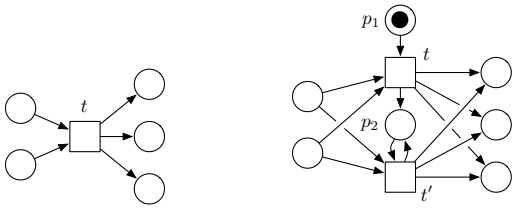
4 Case study: Checking relaxed soundness

To demonstrate the usage of the Petri net API in a realistic setting, we discuss a small case study in this section. We show how *relaxed soundness* [8], a correctness property of workflow nets [1], can be translated into a series of reachability problems that can be checked by LoLA [22].

Relaxed soundness requires for every transition of the workflow net to occur in at least one successful firing sequence from the initial marking $[i]$ to the final marking $[o]$. Dehnert and Aalst [8] provided a verification algorithm for this property in which builds the state space of the workflow net and then analyzes its runs. This algorithm is implemented in the tool Woflan [21]. It is, however, not clear whether state space reduction techniques are applicable.

Alternatively, we can reformulate the above requirement in a reachability problem as follows. Given a workflow net N and a transition t of N, we can construct a Petri net N_t which only reaches a final marking iff N reaches a final marking by a transition sequence which includes t. We create N_t by adding to N a transition t' and two places p_1 and p_2. Figure 1 illustrates the construction. Thereby, p_1 is marked as long t has not yet fired, and p_2 is marked after t has fired at least once. We then can check wether the marking $[o, p_2]$ is reachable from the initial marking $[i, p_1]$. If this check succeeds for all nets N_t, we can conclude relaxed soundness of N.

[1] GNU Lesser General Public License, `http://www.gnu.org/licenses/lgpl.html`



(a) subnet of N (b) subnet of N_t

Fig. 1. Transforming relaxed soundness into a reachability problem.

Listing 1.2. Construction of N_t using the Petri Net API.

```
#include <pnapi/pnapi.h>
using namespace pnapi;

void constructAnalysisNets(PetriNet &N) {
  // iterate transitions
  PNAPI_FOREACH(trans, N.getTransitions()) {
    // create copy of the net
    PetriNet Nt(N);

    // create analysis subnet for current transition (see Fig. 1)
    Place &p1 = Nt.createPlace();
    Place &p2 = Nt.createPlace();
    Transition *t = Nt.findTransition((*trans)->getName());
    Transition &tprime = Nt.createTransition();

    // connect analysis subnet
    PNAPI_FOREACH(p, t->getPreset()) {
      Nt.createArc(**p, tprime);
    }
    PNAPI_FOREACH(p, t->getPostset()) {
      Nt.createArc(tprime, **p);
    }
    p1.setTokenCount(1);
    Nt.createArc(p1, *t);
    Nt.createArc(*t, p2);
    Nt.createArc(p2, tprime);
    Nt.createArc(tprime, p2);

    // write nets into LoLA files
    std::ofstream o;
    o.open("N_" + (*trans)->getName() + ".lola", std::ios_base::trunc);
    o << pnapi::io::lola << Nt << std::endl;
    o << "FORMULA (" << pnapi::io::lola << Nt.getFinalCondition()
      << " AND " << p2.getName() << " = 1 )" << std::endl;
  }
}
```

Listing 1.2 shows a function implementing this construction. It assumes a Petri net N is given and writes, for each transition t of N, a Petri net N_t in LoLA file format together with a formula expressing the final marking to disk.

The implementation is straightforward and is — due the encapsulation of the Petri net functions — nearly on a pseudocode level. The Petri net model checker LoLA can read these generated files and check whether the included formula can

be satisfied by a reachable marking. The described transformation and check is implemented as service-oriented extension of the business process editor Oryx [7].

5 Related work

The idea to collect Petri net-related functions in a library or toolbox is not new (see [16,4] and the Petri Net World Website[2] listing hundreds of tools). We discuss two prominent examples.

The *Petri Net Kernel* [11] was designed as a modular kernel that is designed to integrate Petri net algorithms. Petri net types are not fixed, but can be freely defined and extended. Similarly, the *PNML framework* [10] is a reference implementation of the PNML standard. It is designed to facilitate import and export of PNML standard compliant files and provides a complex meta model to support different kinds of Petri nets. Again, its focus lies on flexibility.

In contrast, the Petri net API has a fixed feature set and new features are only added when they are also used by other tools. Furthermore, it was not originally designed as a generic Petri net framework, but was created by "outsourcing" Petri net-related code from actual tools. Finally, it is implemented in C++ due to performance considerations.

6 Conclusion

This paper introduced the Petri Net API, a library of Petri net-related functions. We observed that the Petri Net API greatly simplified rapid prototyping. The encapsulation of Petri net-related functions lead to smaller tools which could focus on their main functionality; see [13] for a discussion. After four years of development and a consolidated feature set, we claim that this API is useful to other researchers in the Petri net community. The main advantage of the Petri Net API is that its implemented functions are heavily used for several years and thus has a well-tested and justified feature set.

Acknowledgments. The authors thank Christian Gierds, Dennis Reinert, Georg Straube, Robert Waltemath, and Martin Znamirowski for their work on earlier versions of the Petri Net API.

References

1. Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
2. Aalst, W.M.P.v.d., Hofstede, A.H.M.t.: YAWL: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)

[2] `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/`

3. Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: a tool for the synthesis and mining of Petri nets. In: ACSD 2009. pp. 181–185. IEEE (2009)
4. Chouikha, A., Fay, A., Schnieder, E.: Konzept eines Frame- works für Petrinetz-Editoren. In: AWPN 1998. pp. 32–37. No. 694 in Research Reports, Universität Dortmund, Fachbereich Informatik (1998)
5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
6. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Trans. Inf. and Syst. E80-D(3), 315–325 (1997)
7. Decker, G., Overdick, H., Weske, M.: Oryx - an open modeling platform for the bpm community. In: BPM 2008. pp. 382–385. LNCS 5240, Springer (2008)
8. Dehnert, J., Aalst, W.M.P.v.d.: Bridging the gap between business models and workflow specifications. Int. J. Cooperative Inf. Syst. 13(3), 289–332 (2004)
9. Hillah, L.M., Kindler, E., Kordon, F., Petrucci, L., Trèves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. Petri Net Newsletter 76, 9–28 (2009)
10. Hillah, L.M., Kordon, F., Petrucci, L., Trèves, N.: PNML framework: An extendable reference implementation of the Petri Net Markup Language. In: PETRI NETS 2010. pp. 318–327. LNCS 6128, Springer (2010)
11. Kindler, E., Weber, M.: The Petri Net Kernel - an infrastructure for building Petri net tools. STTT 3(4), 486–497 (2001)
12. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin, Berlin, Germany (2007)
13. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. pp. 297–307. LNCS 6128, Springer (2010)
14. Lohmann, N., Wolf, K.: How to implement a theory of correctness in the area of business processes and services. In: BPM 2010. pp. 61–77. LNCS 6336, Springer (2010)
15. Massuthe, P., Weinberg, D.: Fiona: A tool to analyze interacting open nets. In: AWPN 2008. pp. 99–104. CEUR Workshop Proceedings Vol. 380, CEUR-WS.org (2008)
16. Menzel, T.: Entwurf und Implementierung eines objektorientierten Frameworks zur Petrinetzbasierten Modellierung. In: AWPN 1997. pp. 25–30. No. 85 in Informatik-Berichte, Humboldt-Universität zu Berlin, Institut für Informatik (1997)
17. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
18. Pillat, T.: Gegenüberstellung struktureller Reduktionstechniken für Petrinetze. Diplomarbeit, Humboldt-Universität zu Berlin (2008), (in German)
19. Starke, P.H.: Analyse von Petri-Netz-Modellen. Teubner Verlag (1990)
20. Verbeek, E., Buijs, J., Dongen, B.v., Aalst, W.M.P.v.d.: Prom 6: The process mining toolkit. In: BPM 2010 Demos. CEUR Workshop Proceedings Vol. 615, CEUR-WS.org (2010)
21. Verbeek, H.M.W., Basten, T., Aalst, W.M.P.v.d.: Diagnosing workflow processes using Woflan. Comput. J. 44(4), 246–279 (2001)
22. Wolf, K.: Generating Petri net state spaces. In: ICATPN 2007. pp. 29–42. LNCS 4546, Springer (2007)
23. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency 5460(2), 152–171 (2009)