

Proposal for: Requirements and Evaluation of tool papers for PETRI NETS

Giuliana Franceschinis, Kees van Hee, Ekkart Kindler, Fabrice Kordon,
Lars M. Kristensen, and Karsten Wolf

Abstract. This paper gives guidelines for the publication of papers on software tools, in particular for Petri net tools. The guidelines are illustrated by an example.

1 Introduction

In computer science in general and in the field of Petri nets in particular, software tools are becoming more and more important. Often, the scientific work in our field results in algorithms and techniques that can be supported by software tools, which can be applied either by researchers or by practitioners to analyze or design complex systems. Society is more and more expecting practical results from scientists. Hence, in many cases, stand-alone scientific publications will not be enough in the future.

For these reasons the PETRI NETS community (officially: International Conference on Application and Theory of Petri Nets) stimulates the development of software tools by giving the opportunity to researchers to publish tool papers in the proceedings of the PETRI NETS conferences. Papers can be submitted in a special category *tool paper* to PETRI NETS.

In the last years, there have been several problems with submitted tool papers, and there has been a recurrent discussion on how a good tool paper should look like. Some tool papers seemed to present a nice tool, but the presentation was not really helpful to the audience of PETRI NETS. Other tool papers looked very much like regular papers presenting some algorithm or method; but the authors, for whichever reason, submitted them as tool papers. Part of the problem was, that there are no clear guidelines what a tool paper should cover and how a good tool paper should look like. This paper should give some guidelines for writing as well as for evaluating tool papers at PETRI NETS.

2 Requirements

This section states some basic requirement for tool papers and the presented tools. Clearly, the focus of a tool paper should be on the tool and present the tool and its features, and the features should be presented from a user's point of view. It should not focus on the inner workings of the tool, or on some algorithms and methods that could be published as a regular paper.

Therefore, the most basic acceptance criterion is that the presented tool is available for evaluation for the reviewers during the reviewing process and, if the paper is accepted, for the readers. This does not require that the tool is open or free software; but it requires that at least a demo version with some documentation is freely available, preferably via a web-page. The demo version must make it possible to evaluate the features of the tool that are presented in the paper.

Moreover, a tool paper must present a tool that has not been presented before as a tool paper in the proceedings of PETRI NETS. And the tool must be of sufficient interest for the community; this could be either for the methods or algorithms it features, for its advanced usability or smoother integration of different methods, for a new open architecture, or for the general potential of being a widely used tool. In exceptional cases, a tool that was presented at PETRI NETS already can be presented another time. In such cases, the tool must have significant new features, and the paper must clearly point out the new features that justify another presentation of that tool. Note that a tool can be a monolith, which means that it is one piece of software with a clear functionality or it can be a set of related tools, under one name. In the latter case we consider each member of the set as a tool itself.

3 Format

A tool paper should clearly address the following issues:

1. The *objectives* of the software tool: A description of the purpose and application domain of the tool, its intended use and application scenarios, as well as the typical types of users.
2. The *functionality* of the tool: What can you do with the tool and what is the input and resulting output (at an abstract level). This can be done by means of use cases, preferably using a (small) running example. Here also the essential methods and algorithms that are used to transform input into output could be explained, again, at a high level. Note that the paper should not present the theory behind the tool, but instead should provide references to the literature.
3. The *architecture* of the software tool: How is the tool composed from existing or new components and what are the main design decisions and rationale underlying the tool? For each relevant new component the functionality should be described and the interfaces with other components. Further, if relevant, the information architecture (logical structure of the database of the tool) should be described. Finally, the runtime environment of the system should be specified. Moreover, the interfaces to other tools and possible interchange of data with other tools should be discussed.
4. Some interesting *use cases* that illustrate the working and use of the tool and, possibly, a discussion on the experiences obtained while using the tool. A detailed evaluation of a tool, however, is beyond the scope of a tool paper; this could be a regular paper. If such studies exist already, a tool paper is free to refer to them.

5. A *comparison* with other tools or a former version of the same tool. A tool paper should clearly point out what its main characteristics are and relate them to the features and characteristics of existing tools. Also, a discussion of the limitations of the tool is welcome.
6. The paper should clearly indicate where the information can be found to *obtain*, *install*, and *start* the tool, and what the *license* conditions are. This information, possibly available on the web, must provide enough details so that the intended type of user is able to get the tool started.

Note that a tool paper is limited to 10 pages in LNCS format. Therefore, each of these issues must be described in a compact way. For more detailed information, the paper can refer to other publications and, preferably, some web pages. Still a tool paper must be readable in isolation, i. e. it should be self-contained.

4 Example

There are many good examples of tool papers of PETRI NETS (see e.g. the first two references). Here, we use the first one as an illustration of these guideline: “Petriweb: a Repository for Petri Nets [1]”. We made some changes to the original paper in order to better illustrate these guidelines.

4.1 Objectives

Many tools exist to support the modeling process. One tool may be better suited for designing models, another for analysis. Therefore, in a typical development or research environment, multiple tools are used in combination. This creates the need to work on the same models with different tools. This can be addressed by defining a standard file format that all tools can use.

Models need not only be shared by tools, but also by different users. For instance, a model may need to be reviewed by a colleague of the designer. We can send the model, and the colleague can open and use the model, but as soon as new versions appear, it becomes hard to make sure the right version is always used. Here, a shared location for the models is needed.

A shared collection of models also encourages users to reuse existing models or parts of them. This can be useful for different kinds of users. Designers, who employ modeling to describe and design systems, can use this to streamline the modeling process. Researchers and educators can build up collections of models used as illustrations, e.g. examples or counterexamples in proofs.

Most collections of models will be assembled in the context of a specific project, with a small group of participants. But collections can also be turned into company-wide or world-wide resources. In such cases, users will rarely be familiar with all the models in the collection, and collections can grow quite large.

The tool presented here is meant for the management of large collections of models, in particular for storage and retrieval of models and for interfacing with tools to manipulate the models.

A specific domain of application is process modeling with Petri nets at our university. Many examples in course material and exercises are reused over the years; they are often recreated from memory or from paper. It is attractive to make such examples available in a shared repository, accessible by both students and teachers. Here, the need for both browsing and searching facilities is evident. Since users recognize Petri nets by their graphical representation, browsing the collection can only be supported with a graphical browser. In larger collections, users also need to filter the collection based on properties of the content. For instance, users may want to find examples of Petri nets that are bounded and contain a deadlock.

4.2 Functionality

Petriweb is a web application for managing repositories of Petri nets. A Petriweb installation can host many different collections, each with their own administrators and users. Petri web has two sets of functions:

- Retrieval of Petri net models.
- Model management.

An example query Petriweb needs to support: “Find the smallest net present that is live, unbounded and free-choice”. Another example, from a developer’s perspective: “Find a component with this interface and performing task P ”. Queries must also include metadata, e.g. the author or original publication of the net. We see that Petriweb must support different kinds of properties: structural properties, e.g. the number of places; behavioral properties, such as boundedness or liveness; and metadata.

Retrieval is performed by means of the following *search criteria*.

- *Metadata*

By allowing metadata as properties in Petriweb, the user uploading the net can specify related information about it, such as when it was created, by whom, etc.

- *Application characteristics*

They describe what the model is about: in what domain, for what purposes, etc.

- *Structural and behavioral properties*

Simple structural properties can be determined by simple programs, while more complex structural and behavioral properties such as free choice, liveness, boundedness, can be determined with existing Petri net analysis tools. Many such tools can be called as filters that take a net as input and produce results as output. Petriweb can incorporate this through *automated* properties. These are not specified by the user while uploading the net, but instead, computed automatically by invoking an XSLT stylesheet or an external command. This allows non-trivial structural and behavioral properties to be determined automatically; they can even be used in search criteria.

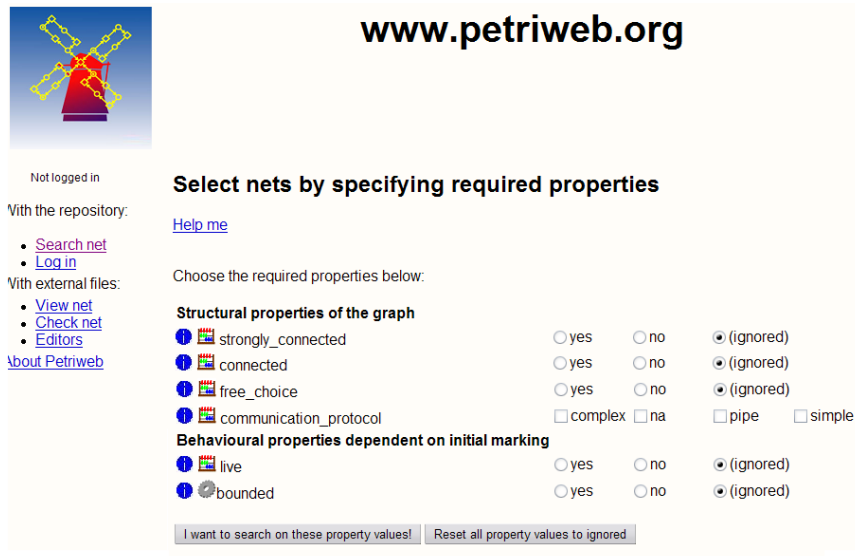


Fig. 1. Property based searching

– *Transformations*

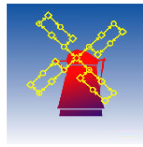
This mechanism can also be used to automate conversions from PNML to other file formats, e.g. the TPN format of Woflan [2]. Calling an analysis tool is often preceded by a transformation, but the results can also be presented to the user, e.g., as the input for a client-side tool. In supporting transformations, Petriweb becomes more than a repository: it functions as a mediator between different tools and formats.

Fig. 1 shows the user interface of Petriweb to enter the search criteria. Not only searching on properties is important, the graphical representation of a Petri net is also a great help in finding a specific Petri net. It is hard to describe a Petri net in words, such that others can find it; the graphical display of a net is much easier to recognize. Therefore, search results do not only list the Petri nets' names and properties, but also display them as diagrams. This allows a combination of searching and browsing. Fig 2 shows the user interface of Petriweb for browsing the search results.

The second group of functions concerns *model management*.

– *Addition*

Petriweb has been designed for public and private use. Petri nets can be shared within a community. Anyone can register at Petriweb and upload their own Petri nets. To allow this, a standard file format is used for uploading Petri nets: the Petri Net Markup Language (PNML). Before adding the Petri net, Petriweb first checks its syntax against a PNML syntax definition [3]. The net is then parsed and stored.



www.petriweb.org

Not logged in

With the repository:

- [Search net](#)
- [Log in](#)

With external files:

- [View net](#)
- [Check net](#)
- [Editors](#)

[About Petriweb](#)

Name: A or B

Structural properties of the graph

strongly_connected:	Unknown
connected:	Unknown
places:	Unknown
free_choice:	Unknown
communication_protocol:	Unknown

Behavioural properties dependent on initial marking

live:	Unknown
bounded:	Unknown

Exported File formats

woflan:	click to view
PNML (regenerated):	click to view
PNML (original):	click to view

Name: basictransformations

Structural properties of the graph

strongly_connected:	Unknown
connected:	Unknown
places:	Unknown
free_choice:	Unknown
communication_protocol:	Unknown




Fig. 2. Search results

– *Approval*

While anyone can be registered at Petriweb and upload Petri nets to it, nets go through a built-in approval mechanism. A net can be in three possible states: uploaded, approved, or deleted. After a user uploads a Petri net for a community, the community moderator receives a notification and can decide whether the Petri net is approved or denied. In this way, collections remain manageable.

– *Retrieval*

Sharing the Petri nets also means that it is possible to download and use the Petri nets from Petriweb. Petriweb supports this in two ways, by allowing downloading the original uploaded file, or a file generated from the parsed information. In this way it is possible to serve as many tools as possible.

Petriweb is primarily intended to serve the public community by sharing Petri nets. Users may not want to share their Petri nets with the total public community, but only with a small group of users. Therefore, Petriweb features a built-in mechanism to support communities. Each registered user can request a community. After the Petriweb administrator has approved the request, the user becomes moderator of the private community and can invite users to join. In this way restricted project areas can be defined and used, with the full search and properties functionality of Petriweb.

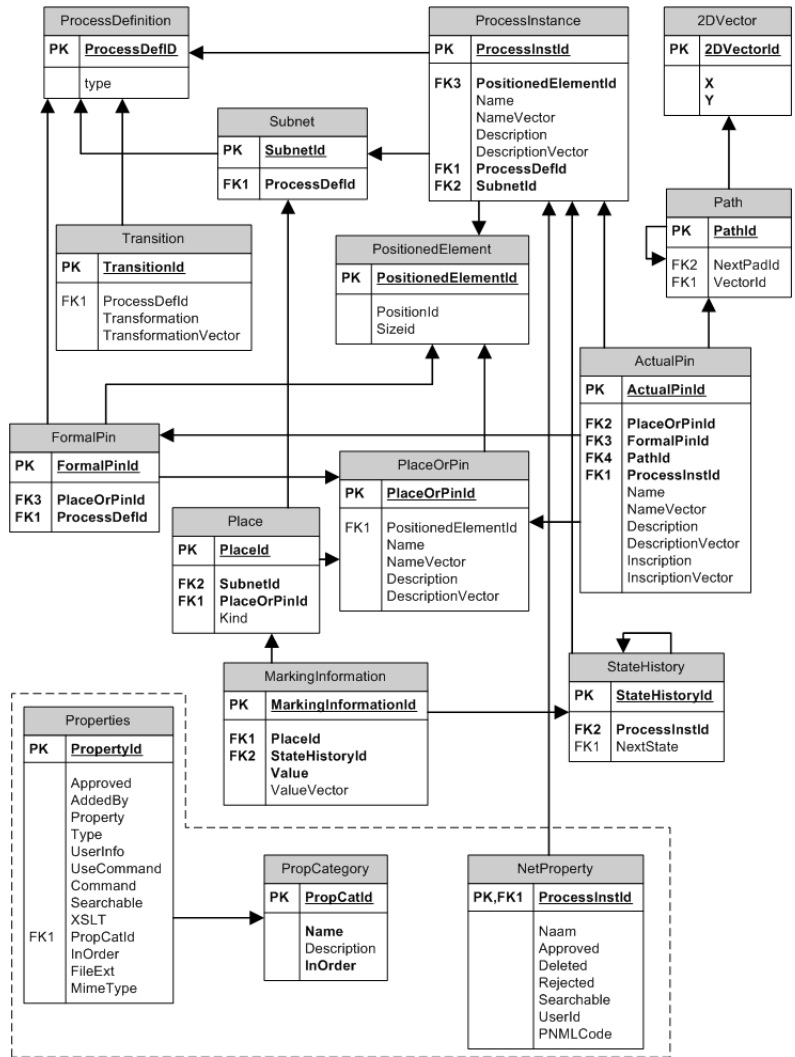


Fig. 3. Data schema of Petriweb

4.3 Architecture

Petriweb is based on the data schema given in Fig. 3. It consists of three parts:

- the structure and marking of the Petri net,
- the graphical information to display the Petri net, and
- the properties associated with the Petri net.

Petriweb supports component based models: definitions define the behavior and structure, and can be instantiated in other definitions. Each Petri net definition is either a transition definition or a (sub)net definition. For every transition and subnet, its definition and its instance are stored. Every instance is a part of a (sub)net. The relation between *ProcessInstance* and *Subnet* depicts the hierarchy of the Petri net. Places are also part of a (sub)net. A marking is stored in the *MarkingInformation* table. A Petri net can have a trace of markings (a firing sequence); this information is stored in the *StateHistory* table. Multiple markings and traces per net are supported. Arcs are not considered as separate objects in the Petri net, but instead, both transitions and subnets have connectable pins. These pins connect instances with places or other pins. Due to the separation of definition and instance, a distinction is made between formal and actual pins. A formal pin is part of the definition, an actual pin is an instance of the formal pin. Graphical information is part of the data schema, although not all relations are drawn in the figure.

Properties Since different uses of the repository gives need of different kind of properties, Petriweb is designed to give each community or installation its own properties, i.e. the moderator of each community or installation needs to define the properties which have to be filled in when uploading a Petri net. In this way properties are as generic as possible. In Petriweb *properties* are stored in the table *Properties*. Each property belongs to a specific category. The table *PropCategory* contains information about the different categories. The table *NetProperty* contains the values of properties of (sub)nets in the repository. The table contains some standard properties that should always be available for every Petri net, such as its name, the location of its file and whether the (sub)net is approved. For each property defined in Petriweb (thus an entry in the table *Properties*) a column is present in this table. Automated properties can be derived either by applying a stylesheet (XSLT) or by calling a tool on the command line. The output is parsed into the database. In this way, any tool can be used to generate the value of a property, hence supporting as many disciplines as possible.

Communities To support *communities*, it is of importance to separate the different communities from each other, in such a way that there is no connection between them. An advantage of this approach is that only properties needed in the community are stored and shown. To support this, each community receives its own database to fill. In order to administer the different communities, the data schema of Fig. 4 is used.

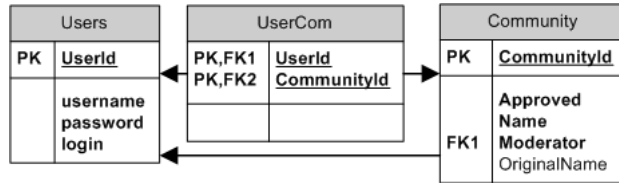


Fig. 4. Communities and their users

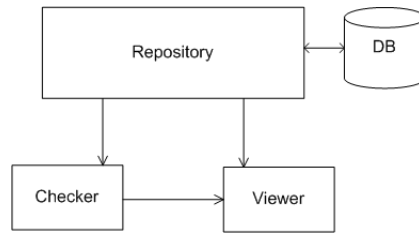


Fig. 5. Component model

Design The design of Petriweb is component based (Fig. 5). It is divided into three main components: repository, viewer and checker. The repository uses the viewer and checker. All components are designed and implemented as stand-alone applications. It was a goal to provide easily installation on different machines.¹ The implementation in PHP with MySQL meets this goal. Properties are designed in such a way that third party tools can be part of the repository. The implementation is such that it is also possible to integrate the upload and search of Petri nets into stand-alone tools, like an editor or analysis tool. The software is portable; different Petriweb installations can be easily installed. A publicly available installation of Petriweb can be found at <http://www.petriweb.org>. Also the source code is publicly available, and can be downloaded from the public website.

4.4 Use case

Typical uses of Petriweb is to search for Petri nets with certain properties, to verify a hypothesis, or disproof a property. See Fig. 2 and 1. The found Petri nets can be tested on the hypothesis. If all Petri nets satisfy it, one has to search for a counter example by hand. If such a counter example is found, the user can upload it, and specify meta data of the Petri net. Petriweb then calculates the automated properties. After the moderator approves the Petri net, it is added to Petriweb, and retrievable for other users.

¹ Petriweb has been tested on two platforms: Linux and Microsoft Windows.

4.5 Comparison with other tools

There are various solutions for sharing data. One of them is to use standard version control software, such as CVS or Subversion. However, these systems still require the users to be familiar with the organization of the material in terms of file names and directory structure. For larger collections, or an open-ended user base, this does not suffice. Additional facilities for searching and browsing will be required that employ knowledge of the model contents. Therefore, specialized repository software is needed that combines general file management with domain specific knowledge.

The only tool we know of that has comparable functionality is the workflow patterns website (cf. www.workflowpatterns.com). The difference is that this site is only meant for retrieval of patterns and not for mediator type functions of Petri web. Hence, in the workflow patterns website is it not possible to upload patterns, to share patterns in a closed community or to retrieve patterns based on some property.

4.6 Installation

Petriweb can be used by logging in as user on www.petriweb.org. There the user can create his own collection of models and he can use the existing set of public models. If one wants to obtain the sources please contact info@petriweb.org.

References

1. R. Goud, Kees M. van Hee, R. D. J. Post, and Jan Martijn E. M. van der Werf. Petriweb: A repository for petri nets. In Susanna Donatelli and P. S. Thiagarajan, editors, *ICATPN*, volume 4024 of *Lecture Notes in Computer Science*, pages 411–420. Springer, 2006.
2. H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
3. Michael Weber and Ekkart Kindler. The Petri Net Markup Language, April 2002.