

Improving the significance of benchmarks for Petri nets model checkers

Steve Hostettler, Alban Linard, Alexis Marechal, and Matteo Risoldi

Université de Genève, Route de Drize 7, 1227 Carouge - Switzerland
{FirstName.LastName}@unige.ch

Abstract. Benchmarking is a fundamental activity to rigorously quantify the improvements of a new approach or tool with respect to the state of the art. Generally, it consists in comparing results of a given technique with more or less similar approaches. In the Petri nets community, the comparison is often centered on model checking and/or state space calculation performance. However, there is sometimes little justification for the choice of the techniques to compare to. Also, benchmarks often lack context information, such as the exact model used, or how to reproduce the results. This makes it difficult to draw conclusions from the comparisons.

We conducted a survey among the Petri nets community in which we gathered information about the used formalisms and techniques. This revealed an unanimous interest for a common repository of benchmarks. The survey shows that existing efforts in this direction suffer from limitations that prevent their effectiveness.

In this article we report the results of the survey and we outline perspectives for improving Petri nets benchmark repositories.

1 Introduction

One of the goals of developing modern model checkers is often improving the performance of existing tools, whether in terms of time, memory consumption or scalability. A rigorous scientific method requires these improvements to be quantified in order to communicate them to the community. Usually this is done by running the model checker against a set of known benchmarks. A benchmark is

[...] a standard or point of reference against which things may be compared or assessed.[1]

Benchmarks can range from academic examples which focus on a very specific aspect (*e.g.*, the dining philosophers for testing scalability) to more general, real-life examples (*e.g.*, an avionics system with real-time, concurrency and dependability requirements). Benchmark results should be useful when comparing to the state of the art. However, a meaningful comparison of benchmarks is often difficult for a number of reasons. One of them is that the typical article has no space to give enough details about features of the used model that could have greatly influenced the benchmark performance. For example, using a low-level modeling formalism with ad-hoc optimizations for the tool under test can bias the results. Also, even when one tries to use the same model for benchmarking as has been used for another tool, the translation from one

formalism to another can be less than trivial, if not a daunting task. Formalisms differ in semantics, in the way of expressing information, and in expressive capabilities. Some are more abstract and can tackle several semantic aspects (*e.g.*, time, concurrency, algebras...) while others are more restricted. Translating, for example, a Coloured Petri Net (CPN) to a P/T net (PTN) requires some simplifications that have an impact on the interpretation of performance results.

We ran a survey in the Petri nets community, asking what formalisms are being used, what type (if any) of model checking is performed and what are the opinions on the hypothesis of standardizing benchmarks. This article is based on the answers of the community. In it we propose a way to improve the current situation by helping standardizing and classifying models used for benchmarks. We review previous efforts in creating repositories, analyze their limitations and how they can evolve to offer better functionality. The goal of the article is fostering a discussion towards a better, more usable repository of models that the community can benefit from.

The structure of the article is as follows. Sec. 2 reports the answers to the survey and the requirements that have been identified as a consequence. Sec. 3 analyzes existing related work. Sec. 4 illustrates our proposal. Sec. 5 discusses some critical points and open questions of the proposal. Sec. 6 draws conclusions, followed by references and by an appendix with complete diagrams and survey graphs.

2 Requirements

To identify desirable requirements for a repository which improves the current situation, we performed a survey among the Petri net community members. We asked 40 groups and individual researchers to participate. We received feedback from 18 (45%) of them. Here we are only reporting pertinent questions and answers; the complete list is freely available on request to smv@unige.ch. Graphs with answer percentages for questions reported here are in appendix A.

A fundamental couple of questions we asked was:

- Would you find useful to have a central repository for models?
- Would you find useful to have test beds¹ for each axis of research in verification?

The answer has been a resounding 100% yes for both questions. This confirms the suggested lack of a unified resource for benchmarking models that can adapt to different types of research. There have been some however who expressed doubts about the fact that previous approaches have been tried and did not completely succeed. Also, a worrying factor among respondents was the burden of maintenance of a model repository.

Following this, the next useful step in understanding the requirements of the community is identifying what the mentioned axes of research are. This means differentiating and quantifying the following aspects:

- What modeling formalisms are used (Fig. 9). The vast majority of groups focus on PTNs or CPNs. Other types of Petri nets are however well represented. In most cases, the same researchers use multiple formalisms.

¹ Test beds can be a range of models with specific characteristics on which different research groups can perform benchmarks of their approach.

- What is the main purpose of the respondents’ research (Fig. 10). 95% is interested in property validation and verification (the only respondent who isn’t is rather interested in modeling and simulation). Those who use higher-level Petri nets formalisms all put an accent on Modeling as a central purpose.
- Among those who are interested in property verification, we asked to specify the kind of properties (Fig. 11). They all verify invariants, a majority (70%) verifies CTL, and about 50% verify LTL.

These points underline a need for a repository to manage models using different formalisms, and centered on different types of semantics. Also, models should be coupled with different types of property definitions.

Of the total number of respondents, 95% perform state space exploration, and almost half employ structural verification methods (Fig. 12). Among those exploring the state space, the majority use explicit state space representation, associated in most cases to some kind of symbolic encoding (Fig. 14). The quasi-totality of respondents is developing and maintaining a tool (Fig. 13). Also, a relevant 30% is using parallel algorithms to boost performance. According to these data, it is relevant to have a way to compare these tools on the basis of their efficiency when managing large state spaces (in terms of memory, time or other metrics).

Respondents to the survey also had the opportunity of adding free-text remarks to their answers. One interesting fact that emerged from this is that most of them have some sort of collection that they routinely use for benchmarks. These models are either from the literature or, in some cases, by some model collection which may or may not be publicly available. Apart from the most typical toy use cases (*e.g.*, dining philosophers, communication protocols...) no overlap was highlighted in the set of models used. This confirms the difficulty in comparing the benchmarks.

On the basis of these survey results, we think that a suitable repository should fulfill the following list of requirements:

- i. It should allow for formalism-independent descriptions of models.
- ii. It should be possible to include one or more formalism-specific implementations (or *instances*) of the models, possibly in different formalisms.
- iii. It should be possible to express properties to check, using different types of property formalisms that focus on different semantic aspects
- iv. Models, instances and properties should be classified and searchable by *characteristics*. These can be seen as a sort of “tags” with which models can be selected that are suitable for a certain type of verification activity. Characteristics should be suggested by users when creating models.
- v. The repository should store “benchmarks”. A benchmark should be seen here as a collection of runs, done on a certain platform, and performed in order to verify some property on a given model. Each run should be characterized by a tool and tool version, as well as by used source files and possible parameters.
- vi. It should be possible for registered users to add or update new models, instances, properties or benchmarks in a collaborative way, classifying them accordingly.
- vii. Registered users should be able to comment and rate existing models, instances and benchmarks.

- viii. The administration workload should be kept to a minimum. Two fundamental tasks should be managing users (to remove unused profiles or fix access problems) and characteristics (to solve eventual duplications). Deleting models/instances/benchmarks should also be an admin's on-request task, while registered users should rather update them.

We defined use cases for these requirements. Their visual representation is found in Fig. 7 (we will not give here the full textual representation of each use case). They will be discussed in more detail in Sec. 4.

3 State of the Art

As we already said, there is a clear need among the Petri nets community for a central repository for models and test beds. There have already been proposals in this direction: we can cite for example Avrunin et al. from the University of Massachusetts (UMASS) [2] who studied the question and even started an implementation, and [3] who proposed a set of interesting benchmarks, without adding source codes for them. We will present here some of the existing repositories and how they fare with the requirements from the previous section.

BEEM Benchmarks for Explicit Model Checkers (BEEM) [4] is a repository of benchmarks. It contains a predefined set of parametric models expressed in a dedicated low level language called DVE. This language can be automatically translated to other formalisms, the repository provides automatic translation to the Promela language. Models are organized by categories of problems (e.g. mutual exclusion, leader election, etc.). Properties are expressed in LTL. Benchmarks are performed using a model checking tool called DiVinE. BEEM contains 57 benchmarks.

PetriWeb PetriWeb [5] is a collaborative repository of Petri nets, i.e. users can submit and store their models. It supports a particular variant of Petri nets with special features like XOR transitions. Models are expressed with a PNML based definition of these Petri Nets, called EPNML. They are organized by properties that are also managed by the community. PetriWeb contains 79 models, by 5 different authors.

pnmlWEB pnmlWEB [6] is another collaborative repository of PetriNets. Users can submit models expressed in PNML, without restriction, along with a description in natural language. This repository doesn't have actual models yet.

VLTS The Very Large Transition Systems benchmark suite (VLTS) [7] contains a set of benchmarks for concurrent systems. These benchmarks are sometimes taken from industrial examples. They are expressed as Labelled Transition Systems, which makes it difficult to reuse the same benchmarks on any given tool. VLTS contains 40 different benchmarks.

UMASS repository Avrunin et al. proposed a repository [8] where models would be expressed in natural language, with implementations written in Ada. Models are submitted along with examples of properties to check, in different languages (CTL, QREs, etc.). No benchmarks are stored in the repository. This project has not been updated since 2003, and it seems that the development is suspended.

The summary of the position of these examples with regards to the requirements expressed in Sec. 2 can be seen in Table 1. We can see that:

- there is no repository that allows the storage of benchmarks for models without relying on a precise formalism.
- there is no collaborative repository that allows the storage of benchmarks.
- there are some collaborative repositories, but none of them allows evaluation by other users using comments or ratings.

We should also mention that the collaborative experiments were not completely successful: PetriWEB has a set of models but they were submitted by a reduced number of authors, and pnmlWEB is not actually used. The UMASS repository also aimed to be collaborative, but this does not seem to be implemented and the project is now abandoned.

| Repository | BEEM | PetriWEB | pnmlWEB | VLTS | UMASS |
|--|------|----------|---------|------|-------|
| Formalism independent models i | | | ✓ | | ✓ |
| Implementations ii | ✓ | ✓ | | ✓ | ✓ |
| Properties iii | ✓ | ✓ | | | ✓ |
| Classification, search iv | ✓ | ✓ | | ✓ | |
| Benchmarks v | ✓ | | | ✓ | |
| Collaborative vi | | ✓ | ✓ | | |
| Comments, rates vii | | | | | |
| Low admin workload viii | | ✓ | ✓ | | |

Table 1. Repositories classification.

4 Proposal

In this section we propose a high level design that partially fulfills the requirements that have been gathered in Sec. 2. We will go through the requirements and see how they are satisfied. The proposal’s design is described using standard Unified Modeling Language (UML) artifacts that are presented in appendix A. Fig. 7 shows the use case diagram that describes the main functionalities. Fig. 8 presents a class diagram of the repository’s Domain Object Model (DOM). For the sake of explanation we will extract and present selected chunks of the main class diagram. We reference classes of the DOM using the following notation: [*DOMClass*].

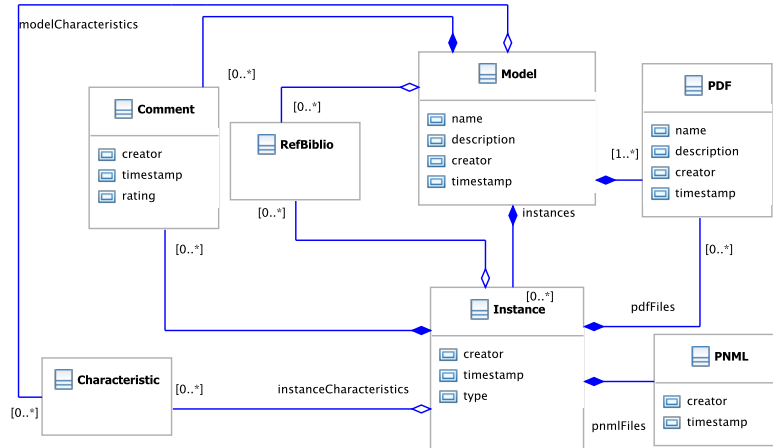


Fig. 1. Detailed view of the Model-Instance semantics group

Requirements **i** & **ii** are handled by a clear separation between the problem to represent and its implementation. This is presented in Fig. 1 by the *[Model]* class that represents the problem to model (*e.g.*, the dining philosophers with deadlock) along with its instances (*[Instance]*) (a model expressed in a specific formalism, *e.g.*, CPN or Timed Petri Net (TPN)). The model can be accompanied by *[PDF]* files, containing a description that should be agnostic of a specific formalism or Petri net flavor.

Model creators can also add bibliographies (*[RefBiblio]*), comments (*[Comment]*) and characteristics (*[Characteristic]*) to the problem description. Characteristics are like tags, defining what interesting aspects the model is concerned with (*e.g.*, concurrency, state space explosion, real time...). These artifacts should give a clear and detailed representation and enable subsequent uses of the model by other scientists, like inferring new instances of the model using different formalisms in order to model different aspects of the problem. The problem description can be instantiated (*[Instance]*) several times, for example using different class of Petri nets. Ideally the instance itself should be described using the Petri Net Markup Language (PNML) (*[PNML]*) as it provides a portable and standard way of describing Petri net models, or using PDF documents (*[PDF]*) if PNML is not possible. As for the model (*[Model]*), its *[Instance]* description can be specified using bibliographies (*[RefBiblio]*), comments (*[Comment]*) and characteristics (*[Characteristic]*). Making these fields searchable would enact requirement **iv**. Because we believe that the only applicable way to maintain such a repository in the long term is a community driven approach (requirement **vi**), any logged-in user can modify models or instances. For the sake of simplicity we did not represent the versioning mechanism in the DOM, as we consider it a detail concerning the implementation platform.

To fulfill requirement **iii**, instances (*[Instance]*) and goals (*[Goal]*) are linked together (see Fig. 2). The *[Goal]* is what the benchmark is focusing on. It can be some-

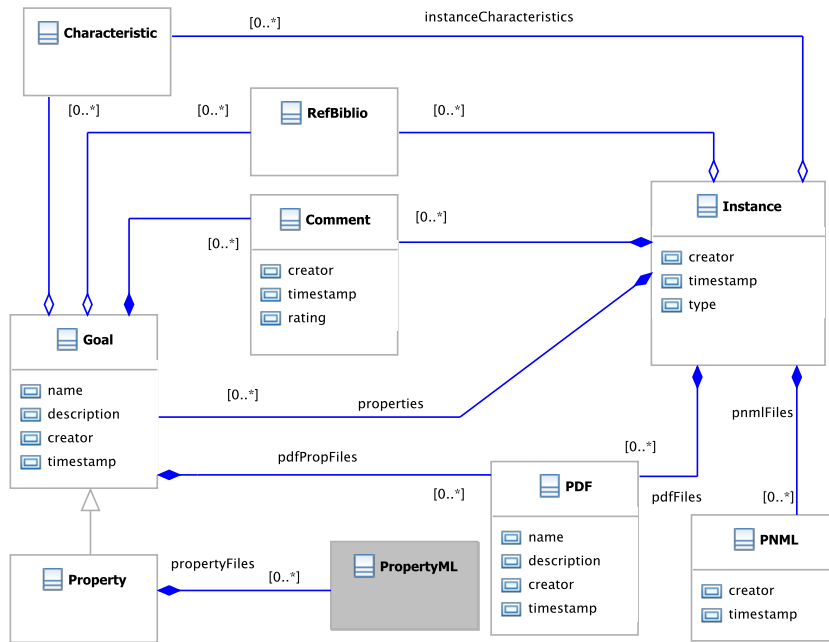


Fig. 2. Detailed view of the Instance-Property semantics group

thing general, like evaluating state space generation performance, or it can also be a *[Property]* to check. Each instance may have several associated goals. Goal descriptions can be enriched using bibliographies (*[RefBiblio]*), comments (*[Comment]*) and characteristics (*[Characteristic]*) (requirement iv). In case the goal is checking a property, a property specification *[Property]* should be included. Unfortunately, there is no standard language for properties equivalent to what PNML is for models. The greyed class *[PropertyML]* represents a future extension to the repository that would allow uploading property files expressed in such a standard language, if and when the community agrees on one.

Fig. 3 proposes a solution to satisfy requirement v. Benchmarks (*[Benchmark]*) are contained by a goal (*[Goal]*). A benchmark is a set of runs (*[Run]*) that have been done on the same platform (to be comparable). Each run contains a result, that is expressed as free text describing, for example, performance information. A run has one and only one context (*[Context]*) that contains all the required information and steps to reproduce the run. These are namely the source files (*[SourceFile]*) and the arguments (*[Argument]*) which the tool (*[Tool]*) should be launched with.

As for the rest of the DOM, *[Benchmark]* and *[Argument]* can be characterized and commented by any contributor.

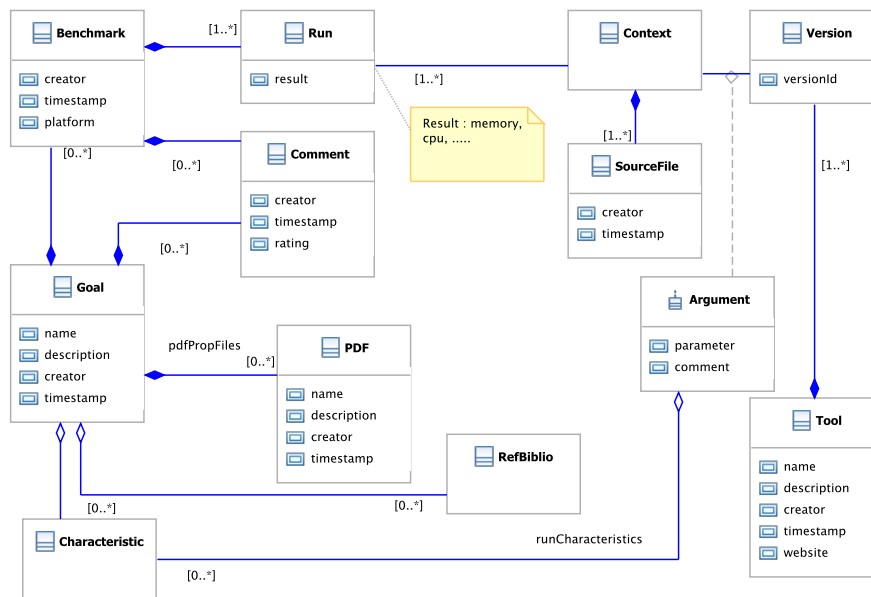


Fig. 3. Detailed view of the Benchmark semantics group

From a functional point of view, we identify three user roles (see Fig. 7):

- A guest that can search and browse the repository (requirement iv).
- A registered user, that inherits from the guest and can also create/update objects, and add comments and evaluations (requirements vi & vii).
- An administrator that can handle basic housekeeping tasks (requirement viii).

Fig. 4 shows the use cases that are executable by the guest user. Guests can apply for registration to gain access to the repository as a registered user. It is worth noting that the registration does not have to be validated by the repository administrator. Indeed, to reduce maintenance cost any human being can register her/himself as a repository user. The fact that it is a human should be enforced by a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) mechanism. Guests are also able to search the repository, however Denial of Service (DOS) attack should be prevented with some limiting mechanism. Once the results of the search have been returned, users can browse them and eventually they can see the details of a given record. These details present the complete object graph ($[Model]$, $[Instance]$, $[Goal]$ or $[Benchmark]$) related to a given result, as well as their dependencies ($[Comment]$, $[PDF]$, $[Characteristic]$...).

Fig. 5 presents the use cases executable by the registered user. This is the main user role, it inherits from the guest role as described in the main use case diagram. This role has all the capabilities of the guest, plus the ability to create and modify content in the repository. There is no authorization mechanism, that is once users have been logged

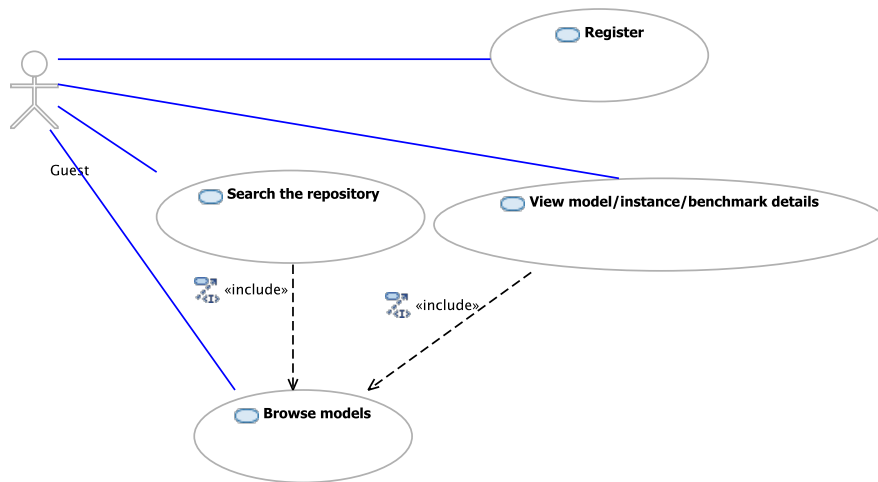


Fig. 4. Detailed view of the guest use cases

they can update any data (in a wikified way: requirement vi). Although ownership is not used for authorization purposes, the repository is protected against malicious users as well as against human mistakes by a versioning mechanism.

Finally, Fig. 6 shows the administrator use cases. This is mainly housekeeping, e.g., removing malicious users or cleaning up characteristics. Since the repository should be driven by the community in a wikified way, the amount of administration should be kept to a minimum (requirement viii).

We will now give two example workflows. The first workflow describes a scientist that is looking for a new model in order to evaluate her own research. The second presents a typical review process involving benchmarks submitted to a computer science conference.

Scientist Sue connects to the repository and is looking for interesting models and properties to check. Since Sue is working on high-level models such as CPN that are also highly-concurrent, she constraints the search by setting criteria such as the instance formalism and by setting some characteristics such as a high concurrency level. She then browses the returned results and finds an interesting model: the dining philosophers. Although the instance is exactly what she is looking for, there is no associated PNML file. She therefore creates a PNML version of the CPN instance of the problem. In order to upload the PNML version she registers and logs in as a registered user. She also leaves some comments about the PNML version of the model and adds some information such as the theoretical bounds of the domains. She comes back a week later, logs in and add a bunch of properties to check. After that, she registers her tool and adds benchmarks. Because she wants to submit a paper to Petri Nets 2012, she gives the reviewers the address of her benchmarks in the repository.

Here comes the second workflow, reviewer Rob has to evaluate Sue’s paper and thus has to validate the benchmarks. Rob simply downloads the context [Context] of

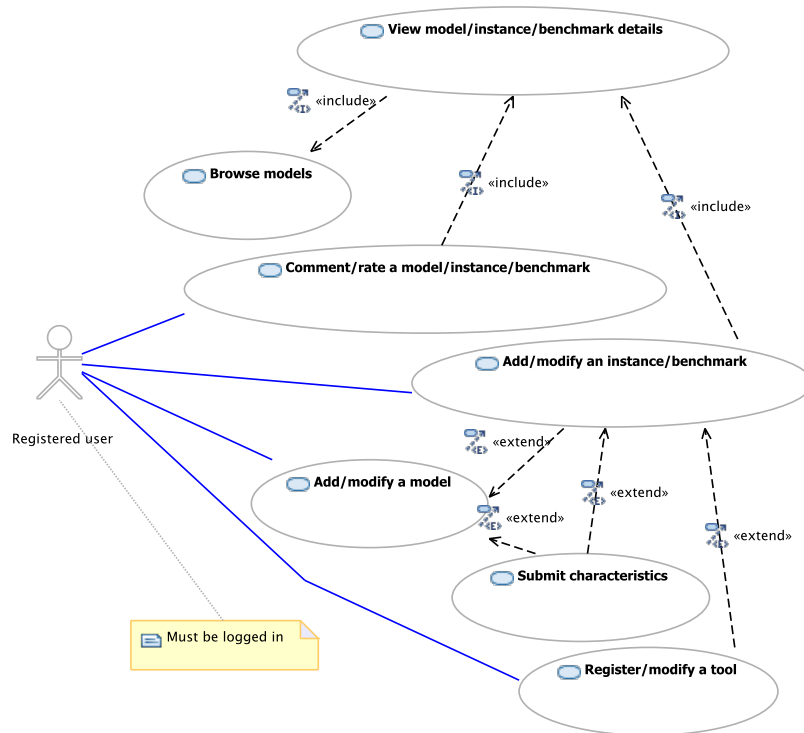


Fig. 5. Detailed view of the registered user use cases

Sue's benchmarks and reproduces them on his multicore machine. He then comments and rates the benchmarks using a special anonymous referee account. He also adds his own benchmarks, that quantify how much this highly-concurrent model benefits from parallel execution on his multiple processors.

5 Discussion

This proposal raises a number of question marks, which should be object for discussion. We identified a list of open issues and potentially critical aspects.

What is the best type of collaborative form? A shared, multiuser repository where each user can contribute with custom material demands for a structure that can manage multiple contributions and concurrent edit. Given the current state of the art in web portals, a way to achieve this could be basing the repository on a wiki. However this raises a number of concerns and points to discuss, *e.g.*, what is the policy *w.r.t.* permissions, updating and creating new models, and approving changes.

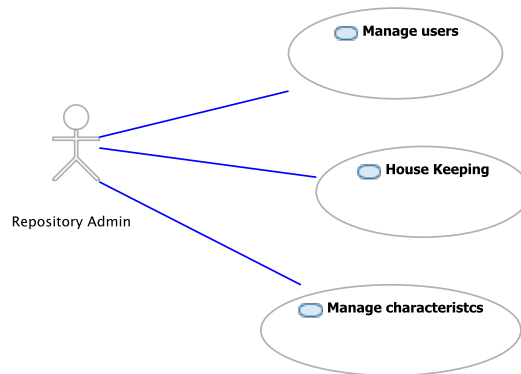


Fig. 6. Detailed view of the admin use cases

Building on existing work. The state of the art reveals that repositories exist that fulfill part of the requirements we are aiming for. Would it be reasonable to extend/revisit existing approaches rather than building new repositories?

Managing characteristics. As it was proposed, characteristics are like a tag cloud. As with tag clouds, one potential issue is duplication and chaos. Careful consideration should be given to the mechanism for submitting new features. It could be possible to rely on users, but this could be weak. Or the admin could approve characteristics for insertion, but this could slow down insertion and create contrasts between the admin and the users. Then again, a semi-automated, knowledge-based mechanism could be devised that tries to detect duplication and suggests to the users to re-use an existing characteristics instead of creating a new one.

Achieving critical mass. The rate of adoption is a key factor in the success of such a repository. The Petri nets community seems to express a strong interest in the repository concept, but probably something should be done to push the repository forward. For example, it could be used as a required standard form to upload benchmarks for articles submitted to the Petri Nets conference.

Workforce. We think that, although the repository can count on the community for building content, there has to be at least a kernel of people who are constantly implied in managing it and keeping it clean and functional. Given the nature of many academic positions, this could be a non-trivial requirement, that demands careful consideration.

6 Conclusion

We presented a proposal for a common repository of benchmark models for the Petri nets community. The proposal is based on a survey led in the community at the beginning of 2010. The purpose of this article is fostering a discussion on how to push forward the state of benchmarking for Petri nets.

While building a repository is a central aspect towards this goal, this is certainly not the only one. Community members should consider how to adopt practices that improve the quality of their results publications. We feel we can share the advice given in [2] when it comes to what the community should do to help itself. Tools should be freely available and accessible, in order to facilitate tuning results. Examples should be published in full, and not only as partial descriptions in already short articles. Notations and languages should be standardized to facilitate interchange. Common benchmarks should be identified, and research articles should really stress the empirical aspects. Furthermore, the scientific community should enforce a policy where empirical benchmarking is a required component for sound publication of tool results.

Initiatives in this direction, together with a common repository, can definitely bring some improvement to the quality of published results.

References

1. Frank R. Abate and Elizabeth Jewell. *The new Oxford American dictionary, second edition*. Oxford University Press, New York :, 2005.
2. George S. Avrunin, James C. Corbett, and Matthew B. Dwyer. Benchmarking finite-state verifiers. *STTT*, 2(4):317–320, 2000.
3. Diyaa addein Atiya, Néstor Cataño, and Geral Lüttgen. Towards a benchmark for model checkers of asynchronous concurrent systems. In *University of Warwick, United Kingdom*, 2005.
4. Radek Pelánek. BEEM: Benchmarks for explicit model checkers. In *in SPIN 07*, pages 263–267. Springer, 2007.
5. 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 *ICATPN*, pages 411–420, 2006.
6. Lom Hillah, Rachid Alahyane, Cuauhtemoc Castellanos, Monir Chaouki, and Issam Saïd. pnmlWEB, 2008. <http://pnmlweb.lip6.fr/>.
7. Stefan Blom and Hubert Garavel. Very Large Transition Systems, VLTS, 2009. http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html.
8. LASER laboratory, University of Massachusetts. Example repository for finite state verification tools, 2003. <http://laser.cs.umass.edu/verification-examples/>.

A Appendix

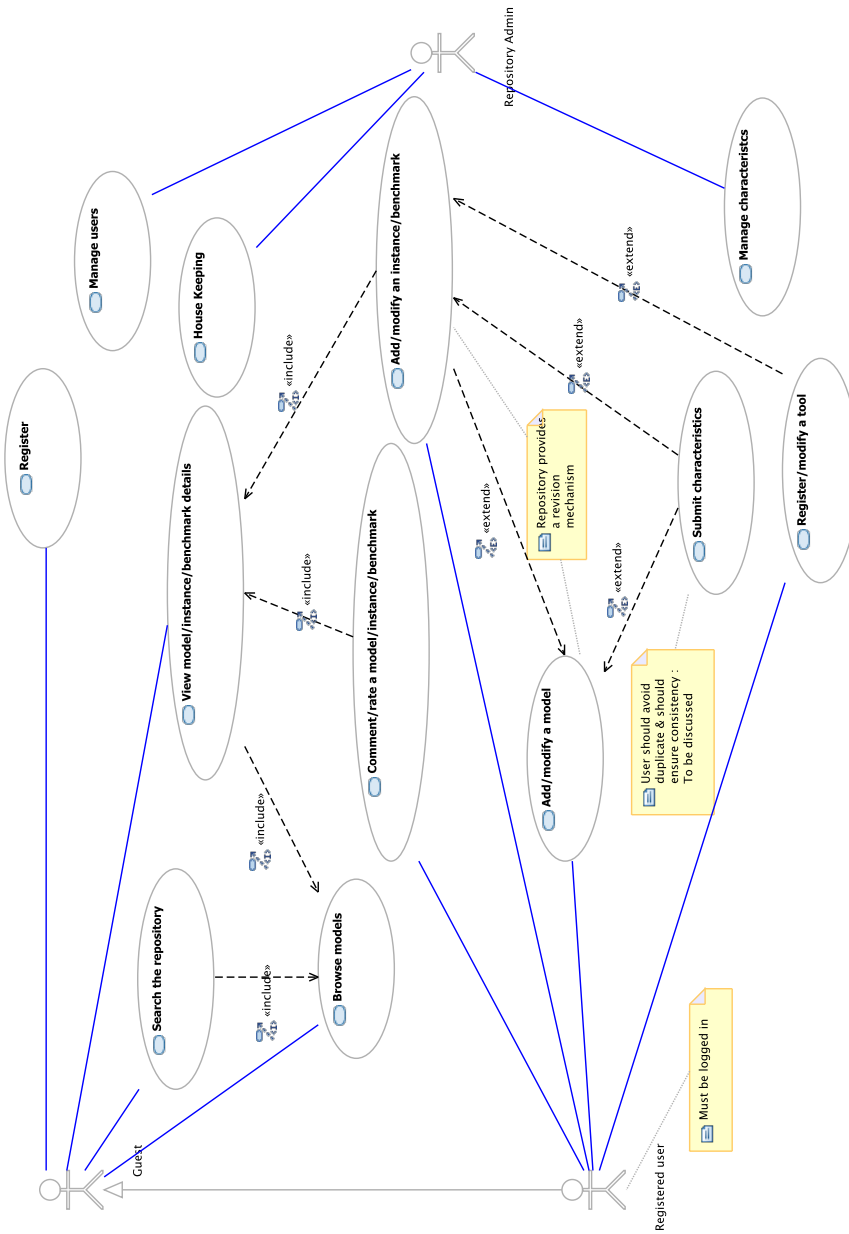


Fig. 7. Requirements for a model repository

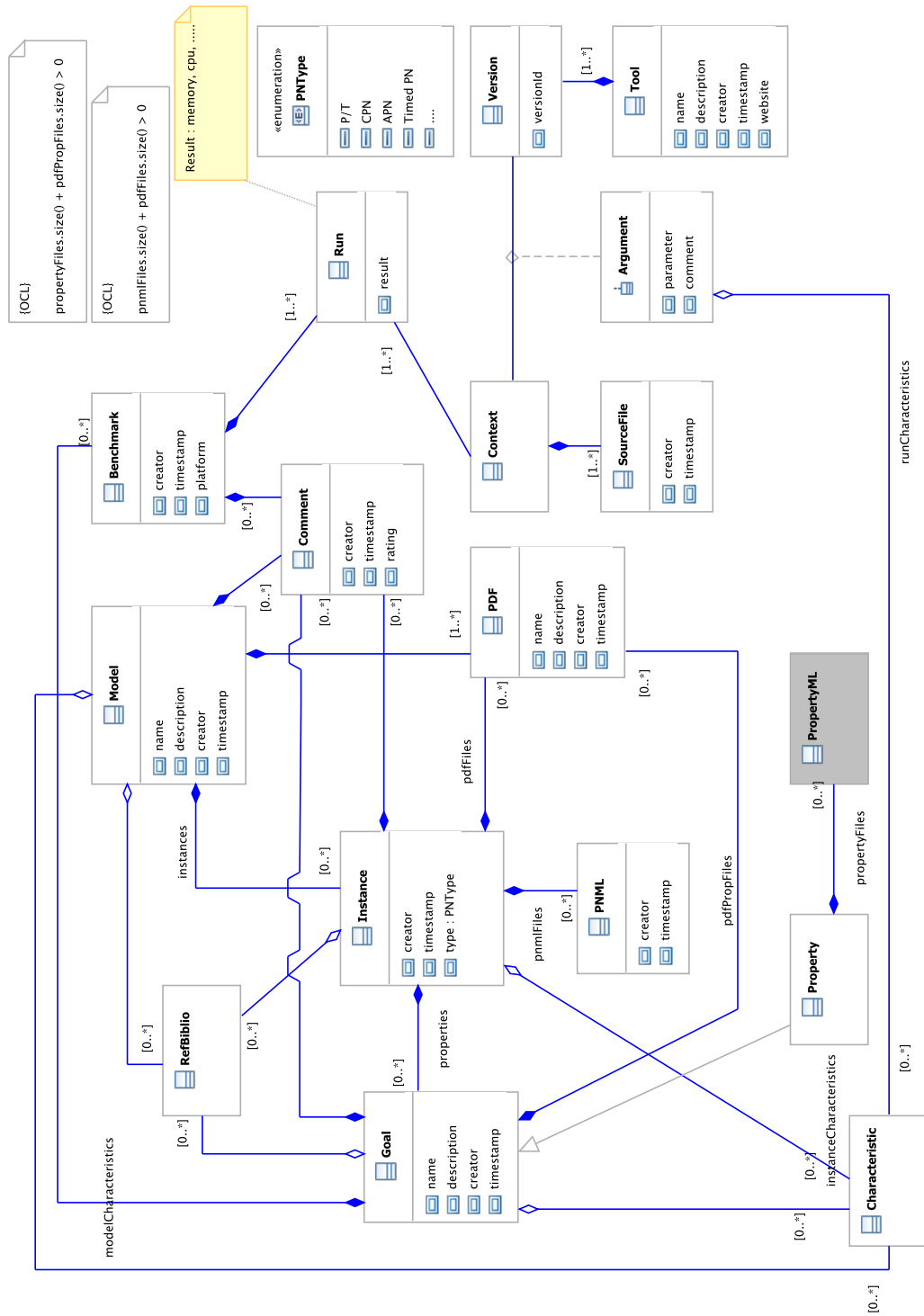


Fig. 8. Proposal for a model repository

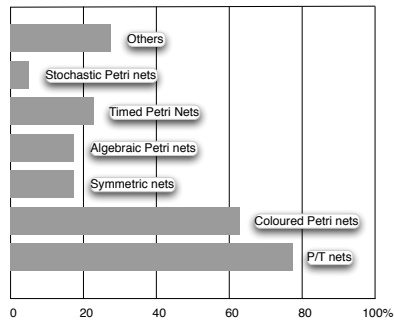


Fig. 9. Which formalism are you using?

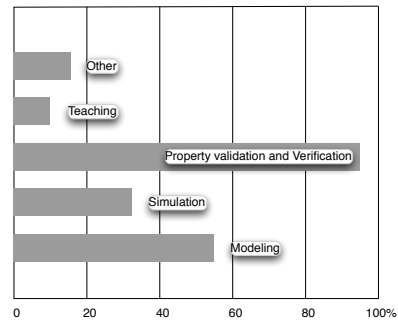


Fig. 10. What is your main purpose?

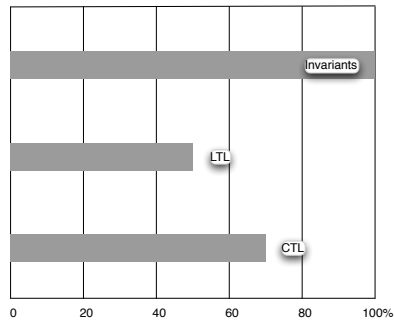


Fig. 11. Which kind of property do you check?

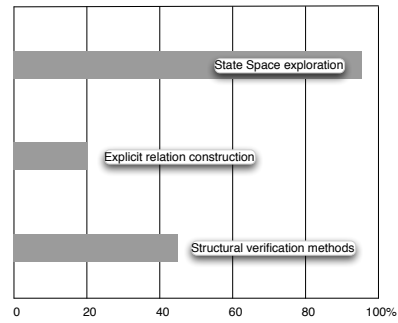


Fig. 12. Which kind of verification techniques?

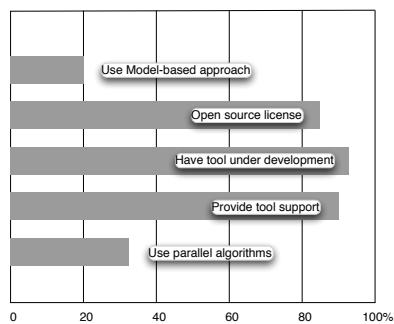


Fig. 13. Miscellanea

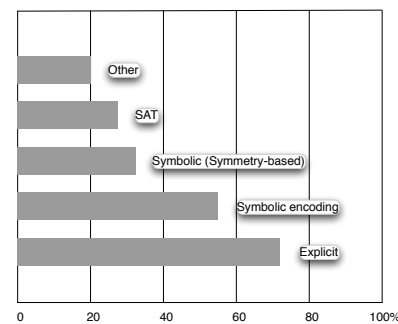


Fig. 14. What kind of state space techniques?