Petri Nets Repository: a tool to benchmark and debug Petri Net tools

Lom Messan Hillah, Fabrice Kordon

Petri Nets Repository, a tool to benchmark and debug Petri Net tools

Lom Messan Hillah[1] and Fabrice Kordon[2]

[1] Univ. Paris Nanterre, LIP6 CNRS UMR 7606, F-75005 Paris, France
(lom-messan.hillah@lip6.fr)
[2] Sorbonne Universités, UPMC Univ. Paris 06, LIP6 CNRS UMR 7606, F-75005 Paris, France
(Fabrice.Kordon@lip6.fr)

Abstract. For a given scientific community, being able to use a common and rich accepted benchmark for the evaluation of algorithms and prototypes is an added value. The goal of this paper is to present Petri Nets Repository, an open Petri nets models database. It offers two main ways to navigate through the benchmark using criteria related to Petri net properties: a Web interface, and a Web service API (REST). So far, this database embeds the models from the Model Checking Contest, as well as those of the discontinued Petriweb.
A placeholder is available to store, when possible, the outputs of the Model Checking Contest; then for the corresponding models there will be formulas and their accepted results available too. We believe this would help the community to easily create oracles to debug new algorithms and tools.

1 Introduction

Motivation. Today, it is much easier, and more requested, to provide reproducible and comparable data when evaluating one's contribution. Thus, there is a strong need for common benchmarks that are shared by a given community. Moreover, when prototype tools are not only proofs-of-concepts, but also a key entry to process industrial problems, there is a need to ensure quality of developed software, which can be partially achieved by using benchmarks whose results are already known.

Petri Nets Repository (PNR) Objectives. To achieve the goal stemming from the first motivation, communities have elaborated competitions where programs of a given area are tested against the same benchmarks and rated. There are established major events concerning different areas: the SAT competition[1] (9 editions since 2002), the Satisfiability Modulo Theories Competition[2] (11 editions since 2005), the Hardware Model Checking Contest[3] (8 editions since 2007), the Rigorous Examination of Reactive Systems Challenge[4] (6 editions since 2010), the Timing Analysis Contest[5] (4 editions since

[1] http://www.satcompetition.org
[2] http://smtcomp.sourceforge.net/
[3] http://fmv.jku.at/hwmcc15/index.html
[4] http://rers-challenge.org
[5] https://sites.google.com/site/taucontest2015/

2011), and the Competition on Software Verification[6] (5 editions since 2012). The existence of these long-lasting events is a clear indication of interest and usefulness. In most cases, benchmarks are freely available to the involved communities.

In the Petri net Community, the Model Checking Contest [10,9] was created in 2011 and reaches its 7^{th} edition in 2017. As for other competitions, it provides an evolving benchmark composed, in 2016, of 664 models dispatched among 67 Petri nets.

However, achieving the goal stemming from the second motivation requires more: we need a way to query models according to identified characteristics that could allow us to check performances of an algorithm for known conditions. This requires a sophisticated environment allowing to extract sub-benchmarks with dedicated specificities.

This is the main objective of Petri Nets Repository (PNR). It is built on top of a Petri net model database that anybody can query. PNR can be accessed either manually via a Web browser interface or programmatically via a REST interface [5]. It contains models from the Model Checking Contest, but we also imported Petri nets from a previously existing database: Petriweb [7]. PNR is not only a database than can be queried via a web interface as Petriweb was. It also offers an API to automate queries, thus enabling its use in the context of tool testing and benchmarking (as detailed in section 4).

Contents. Section 2 details the architecture of PNR, and Sect. 3 presents some technical characteristics of the tool. Section 4 illustrates the typical use of PNR, and Sect. 5 sketches our perspectives in extending PNR before a conclusion in Sect. 6.

2 Architecture

Petri Nets Repository (PNR) is a Web application that currently offers a large collection of Petri net models provided by the community.

Figure 1 depicts the architecture of PNR. It is composed of a front end that exposes two interfaces to end users (through a browser and a REST API), a middle end and a back end. Figure 1 also shows typical clients of PNR, indicating that they can interact with it in two ways, over HTTP. The client with a Web browser interacts with PNR

[6] https://sv-comp.sosy-lab.org

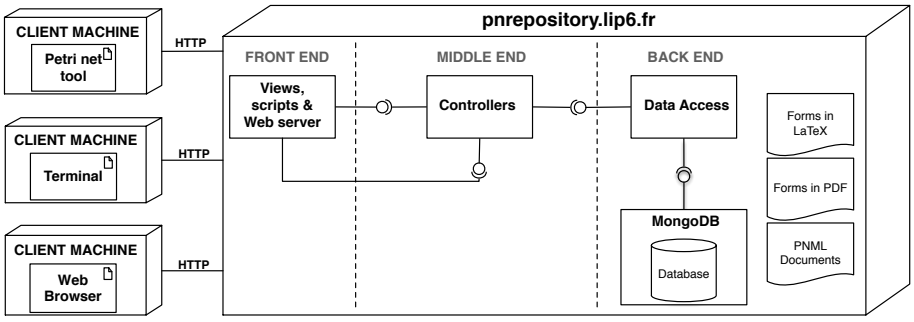

Fig. 1. Overview of the architecture of Petri Nets Repository

through the graphical interface and those with other specific tools (*e.g.* a terminal or a Petri net tool) can communicate with PNR using the exposed REST API. In any case, all clients including browsers use the REST API.

The front end Web interface. The Web front end is composed of a set of views encoded in templatized Scala/HTML files and a Web server that serves these compiled views to any browser. We additionally developed some client-side processing scripts to complement the dynamic rendering of the views in the browser. These scripts are written in CoffeeScript [2] and then compiled into Javascript.

The browser shows the content of the Petri net metadata in a table, fetched from PNR. Metadata typically is composed of a model name, type (*i.e.* P/T or Colored net), size, whether it has parameters, and the values of the statically defined properties (*e.g.* deadlock: false) in the LaTeX file used as a model submission template in the model submission kit[7] of the Model Checking Contest. All metadata from PNR are structured in JavaScript Object Notation (JSON) [4].

End users can filter the content of the table over the defined properties, or even provide an ad hoc query formulated as a logical expression. The search function this type of query triggers is performed locally, on the client side. Finally, end users can also open the PDF forms of the models from the table and download the archive containing their PDF forms or PNML [8] documents, upon selection.

The front end REST API. The concept of route encodes the URLs of the REST API in a text file. A route is composed of three parts: the HTTP method (*e.g.* GET, POST), the path of the queried resource, and the fully-qualified name of the program handling the request. For example, `GET /mcc/models/all/metadata.json controllers.MCCBrowser.browse()` specifies the Java method that handles the querying metadata about all the Petri net models in the MCC collection. All paths are relative to the base URL of PNR (*i.e.* `http://pnrepository.lip6.fr`).

Some routes return the views along with data used in the client-side scripts to render them in the browser. For example, `/mcc/models/all/browser.html` returns the HTML used to render the collection of all the models in the MCC, along with the list of statically defined properties found in their LaTeX description files. Some routes only return data in JSON, for example, the public URL to the PNML file of a given model identifier in a GET request: `/mcc/models/:id/pnml.json`. Other routes return raw data, for example, an archive containing the PNML files of a set of models: `/mcc/pnml/*file`. The client-side scripts use these routes.

The API, whose reference documentation is published on `/api/apiref.html`[8], is useful for any user or client tool to query the collections: fetch metadata, download PDF or PNML files of models according to some search criterion. For instance, using a command line tool like cURL [1], one can fetch the metadata of all the models in the MCC collection in a JSON array with:

Listing 1.1. Command line to fetch the metadata of all the models in the MCC collection

```
curl -o pnr-mcc-metadata.json
  http://pnrepository.lip6.fr/mcc/models/all/metadata.json
```

[7] `http://mcc.lip6.fr/archives/ModelSubmissionKit.tar.gz`

[8] The JSON version of the API reference documentation is at `/api/apiref.json`

The resulting JSON array contains records encoded as JSON objects. Each record describes a Petri net in the queried collection, like the DES model shown in Listing 1.2.

The middle end. The middle end is composed of the business logic programs (in Java) that handle the requests to the exposed REST API, configuration, logging, and access to the database, the PDF, and the PNML files of the models. A key feature of our REST API is the search function over a particular collection of models (*e.g.* `/pweb/search.json` for the Petriweb collection), or over all of them (*i.e.* `/all/search.json`). This search API can potentially inflict a heavy workload upon the back end and the database depending on the frequency and complexity of the requests. Consequently, we have conditioned its successful invocation to providing valid Java Web Tokens (JWT) [12] that we emit upon request, to the interested end users for their own usage.

Listing 1.2. The metadata of the DES model in JSON


```
{"modelName":"DES (Data Encryption Standard)","modelType":"PT",
"authorName":"Wendelin Serwe and Hubert
  Garavel","authorContact":"wendelin.serwe@inria.fr",
"modelOrigin":"Company{NIST (USA)}","modelShortDescription":"These nets are
  derived from the LNT specification of the DES (Data Encryption Standard)
  symmetric-key encryption algorithm.",
"scalingParamName":"(N, V)","parameterised": "TRUE","modelFixedSize":"null",
"ordinary":"True","simpleFreeChoice":"False","extendedFreeChoice":"False",
"stateMachine":"False","markedGraph":"False","connected":"True",
"stronglyConnected":"False","sourcePlace":"True","sinkPlace":"Unknown",
"sourceTransition":"False","sinkTransition":"Unknown","loopFree":"Unknown",
"subConservative":"False","conservative":"False","nestedUnits":"True",
"safe":"True","deadlock":"Unknown","reversible":"Unknown","quasiLive":"Unknown",
"live":"Unknown","year":"2016","lastUpdated":"2016-06-07T20:53:38",
"modelID":"DES","modelBase":"MCC","links": [
{"rel":"self","href":"http://pnrepository.lp6.fr/mcc/models/DES/metadata.json"},
{"rel":"pnml","href":"http://pnrepository.lp6.fr/mcc/models/DES/pnml.json"},
{"rel":"pdf","href":"http://pnrepository.lp6.fr/mcc/models/DES/pdf.json"}]}
```


The back end. The back end consists of both the database containing the metadata about the Petri nets in the collections mentioned above, and the component handling the corresponding data and interfacing the middle end with the database. This back end component also builds the metadata from parsing the LaTeX files describing the Petri nets. These LaTeX files provided by model submitters are thus the primary source of the data model of the Petri nets. Their design has been refined over the years by the MCC model Committee in agreement with the community. We also used it to integrate the imported Petri nets from Petriweb. PNR is powered by MongoDB [13].

We have organized PNR in separate collections: MCC [10,9], Petriweb, and soon, the "Very Large Petri Nets" Benchmark Suite [6]. Firstly, we then show their different backgrounds and purposes, thus maintaining their history. Secondly, we distinguish the specific services offered on top of each collection according to its background. For instance, the MCC collection outputs yearly results that will soon be linked to the models. However, queries can be dispatched over all the collections, and all the models share the same statically defined properties. The authors of these collections have provided their formal agreement to publishing their models in PNR.

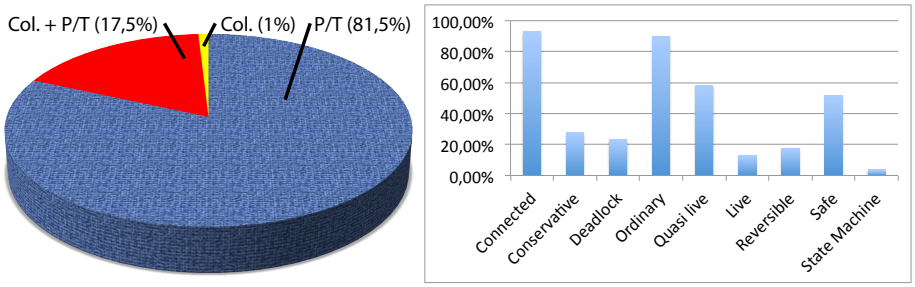

Fig. 2. Distribution of P/T versus colored (left) and according to some typical properties (right).

3 Technical characteristics of the tool

We rely on Play [11], an open source Web application framework, to design the architecture of Petri Nets Repository. Play provides the standard model-view-controller architectural pattern upon which we built our three-tier Web-based enterprise application. Being stateless, RESTful, asynchronous, modular, and full of other carefully designed features, Play enables rapid Web application prototyping and simplifies its deployment.

In the following sections, we provide information about the content of the database and some scalability metrics.

Content of the database. The database was elaborated from the outputs of the community. First, the discontinued Petriweb was gathered by several colleagues and published via a Web site. Second, the collection coming from the Model Checking Contest has been progressively elaborated, year after year since 2011. The result is a heterogeneous collection of Petri nets coming from various areas: distributed algorithms, protocols, hardware systems, biological systems, etc. Moreover, they are issued from both academic and industrial case studies. This variety ensures a sound fairness for evaluation purposes, due to the variants in the application and modeling choices.

Figure 2 (left) shows that we have a significant number of colored Petri nets associated with their equivalent P/T nets, which is useful when comparing the efficiency of algorithms on colored nets, versus others on P/T nets. Figure 2 (right) also shows that, if some optimizations are considered for models having some identified structural characteristics (*e.g.* safe, or reversible), it is easy to select the corresponding subsets of models. This is also useful when evaluating the impact of an algorithm in such situations. Moreover, numerous models from the Model Checking Contest have scaling parameters, allowing to generate instances with an increasing complexity. This provides a way to experimentally evaluate the complexity of an algorithm.

We also aim at integrating the models from the "Very Large Petri Nets" Benchmark Suite (this is detailed in Sect. 5).

Scalability of the server. To observe how PNR[9] would perform when a large number of clients connect, we first tested the MCC collection Web page

[9] The PNR HTTP server is based on Netty (`http://netty.io/`)

(`/mcc/models/all/browser.html`) with 5000 clients over 1 minute. The average response time was 87 ms, with min/max at 80/1141 ms, and 0 error rate. We then tested the API endpoint `/mcc/models/all/metadata.json` with 1000 clients over 1 minute. The average response time was 205 ms, with min/max at 160/890 ms, and 0 error rate.

We then wanted to observe the behavior under a number N of requests every second and under a constant load of clients over a time duration T. We tested the endpoint `/mcc/models/all/metadata.json` with 100 clients per second over 1 minute. The average response time was 95 ms, with min/max at 81/964 ms, and 0 error rate. We then tested the same endpoint with a load from 250 to 1000 clients over 1 minute. The average response time was 319 ms, with min/max at 81/3114 ms, and 0 error rate.

These test cases showed that, under normal circumstances, PNR is fairly scalable. Its native HTTP server is robust enough to support reasonable workloads (*e.g.* the one we can expect from the community).

4 Typical use of Petri Nets Repository

Using a browser. The Web page displaying the collection of all the models in the MCC has tabbed content. The first tab, "MCC Collection Content", displays the list of Petri nets in a table, along with their properties. When hovering the mouse over a model name, its short description is shown in a small pop-up window. Clicking on the model name brings up a modal window that displays the "model form" (in PDF).

The user can filter the content of the table using the properties displayed in the quick selection panel "Filter the models using the following properties" (shown in Fig. 3). While this search method only allows the conjunction of properties, the "Advanced Search" feature (search field in the top-right corner above the table) allows any logical expression combining the properties using the connectors && (AND), || (OR), ! (NOT), and ? (UNKNOWN). For example, one can search for sub-conservative or reversible models with `subconservative || reversible`. Any other arbitrary string that is not a logical expression can also be used to search the table, *e.g.* the name of an author to look for models s/he has submitted. The content of the table is updated as you type. The Reset feature resets the search, displaying the table back in its original state. The CSV and PDF feature exports the content into CSV and PDF files, respectively. The

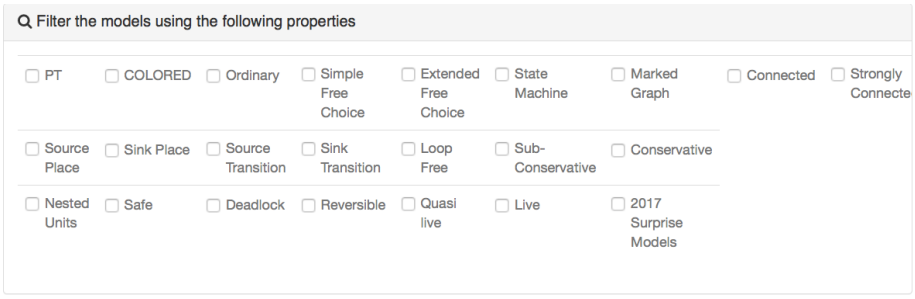

Fig. 3. The quick selection panel of the MCC Collection

content of the table is paginated so that each page initially displays ten rows to keep a reasonable overall length of the visible page in the browser. However, the user can bypass this setting by choosing how many rows to display. Therefore, it is possible to display all rows in a single page. It is also possible to arrange the columns visibility by selecting which ones to hide or show. Finally, once one or more models have been selected, the user can request to download their forms (in PDF) or PNML documents in an archive. The button to ask the download is located in the bottom-left corner below the table.

The "MCC Collection Cover Flow" tab displays the thumbnails of the models PDF forms, allowing another kind of navigation that is more visualization oriented. The "MCC Contributors" tab displays the list of model submitters, and relates them to the models they have submitted. Finally, the "Metrics" tab shows charts on the contents of the MCC Collection.

The page displaying the collection of Petri nets from Petriweb has the same layout as the one for the MCC, except it has not a list of individual contributors since this information was not provided in the data we had recovered.

Using a proprietary tool. We refer to a "proprietary" tool in this context as, for instance, a command line tool like cURL, or a specific Petri net tool. Using such a tool, the end user can invoke the REST API and use the result to build whatever application suits his purpose by using the data from Petri Nets Repository.

The same set of actions that the user can perform through a browser can also be achieved directly using the REST API of PNR. In Section 2, we showed an example in which we fetch the metadata of all models in the MCC collection, using the command line tool cURL. Next, we detail two more examples using cURL again, combined with other command line tools, in particular jq [3] which allows the processing of the resulting JSON data.

For example, to download the PNML documents of Diffusion2D, the command line is the one in Listing 1.3 (using a GET):

Listing 1.3. Command line to download the PNML documents of Diffusion2D in an archive

```
curl http://pnrepository.lip6.fr/mcc/models/Diffusion2D/pnml.json | jq '.href' |
  xargs -I % curl -o Diffusion2D-pnml.tar.gz http://pnrepository.lip6.fr%
```

In the first part of the command, the server responded to the initial request with JSON data specifying the URL to the archive containing the PNML documents. Using jq, we extracted the value of the path field in the second part of the command and fed it to cURL in the last part of the command to download the archive.

We can also perform a search on all the collections in PNR using a query formulated in a JSON payload structured like in Listing 1.4. The `expression` field contains a logical expression that is formulated in the same way as the "Advanced Search" feature allows in the browser. Alternatively, the `fields` contains a mapping to the exact values that the specified properties must have on the matched models (*i.e.* ordinary = True AND live = False AND safe = Unknown). The query in the `expression` field has precedence over the one in `fields`. To have the search function consider the mappings in `fields` instead, it suffices to set the `expression` field to null.

The command line to send a search query on all the collections with the payload of Listing 1.4 is the one in Listing 1.5 (using a POST):

Listing 1.4. A search query payload

```
{ "expression" : "(!live && ordinary) || (safe && quasilive)",
  "fields": [{"ordinary": "True"}, {"live": "False"}, {"safe": "Unknown"}] }
```

Listing 1.5. Command line to request a search on all collections of PNR

```
curl -H "Content-Type: application/json" -H "uuid: ..." -H "Authorization: Bearer
  ..." -X POST --data @Search-Payload.json
  http://pnrepository.lip6.fr/all/search.json -o resultset.json
```

noindent The search query payload is stored in a file named `Search-Payload.json`. The required Authorization header carries the JSON Web Token that will be first checked before the search query is granted. The uuid header carries the unique identifier issued to the user who has requested the authorization token from us. That identifier must be associated with the token in the request to the API. The result set, saved in `resultset.json`, is structured like in Listing1.6

Listing 1.6. The result set yielded by the search query in Listing 1.5

```
{ "code": 200, "status": "success", "size": 49, "resultset": [{...}, {...}, ...] }
```

It shows that the result set contains 49 models that matched the logical expression in Listing 1.4. The `resultset` field is an array of JSON objects, each one describing a matching model with the same structure as in Listing 1.2. The interested user can then loop through this result set and, for example, download the corresponding PDF or PNML documents with subsequent invocations of the API.

Listing 1.7. An oracle for MyTool written in bash

```
#Query PNR to get the list of models with deadlocks
curl -s -H "Content-Type: application/json" -H "uuid: ..." -H "Authorization:
  Bearer ..." -X POST --data '{"expression": "deadlock"}'
  http://pnrepository.lip6.fr/all/search.json -o /tmp/fetch.json
#Get model ids and put them as arguments for later use in the loop
set $(cat /tmp/fetch.json | jq '.resultset [].modelID' | sed -e 's/"//g')
#List of json files each referring to the URL of the PNML archive for a model
cat /tmp/fetch.json | jq '.resultset [].links[] | select(.rel | contains("pnml"))
  | .href' | sed -e 's/"//g' > /tmp/pnml_lists
#Loop on selected models (get all their instances and process them)
rm -rf /tmp/checks ; mkdir -p /tmp/checks
cat /tmp/pnml_lists | while read jsonUrl ; do
        echo "--- retrieving PNML files for model $1"
        # Extract the URL for the PNML archive
        url=$(curl -s $jsonUrl | jq '.href'  | sed -e 's/"//g')
        curl -s -o /tmp/checks/archive.tar.gz $url
        (cd /tmp/checks # process all instances in a temp directory
        tar xzf archive.tar.gz
        for instance in $1/PT/*.pnml ; do # processing all P/T PNML files
                test=$($MyToolPath -deadlock $instance | grep "deadlock=True")
                if [ -z "$test" ] ; then
                        echo "   $instance ==> NOT PASSED"
                else
                        echo "   $instance ==> PASSED"
                fi
        done)
        shift
done
```

Benchmarking a tool. Dynamic access to a large repository of models is useful for benchmarking. Let us consider the case of MyTool that has a deadlock detection function

to be tested. The simple bash script presented in listing 1.7 sets up an oracle that can automatically test the outputs of MyTool against all the models having deadlocks in PNR (a similar script can be elaborated to test the tool when there are no deadlocks). This script will automatically consider new models inserted in PNR.

This benchmarking example shows that debugging can be easily performed on known computed properties. It will be easy to perform similar tests on the behavioral properties computed in the MCC collection (*e.g.* the size of the state space, or some formulas, etc.) as soon as they are inserted in PNR.

5 Perspectives

There are mainly two features we are planning to enrich PNR with, from a short term to a longer term. The first one deals with the integration of the useful "Very large Petri Nets" (VLPN) Benchmark Suite [6]. The second one is about integrating the yearly results of the Model Checking Contest (MCC).

The VLPN benchmark. This is a collection of several hundreds of Petri nets generated from high-level specifications of meaningful systems. They can contain up to 131 216 places and 16 967 720 transitions. So far, the connection to this collection is performed thanks to a simple link to the original Web site. As soon as the database of VLPN is imported, the services of PNR will be available on it too.

The MCC outputs. Since 2015, the Model Checking Contest has introduced a way to evaluate the quality of results based on the outputs of the participating tools. For a given formula, these outputs are summarized and, when a large majority of tools do agree, it is possible to state that the corresponding result is safe. This is the case of all the examinations proposed in the MCC: size of the state spaces, bounds of the nets, and numerous reachability, LTL and CTL formulae.

We would like to associate all these safe results to the corresponding models in order to provide an even more useful benchmark. If models are associated with a set of checked properties, then they can not only be used for benchmarking tools, but also serve as a basis to elaborate oracles for tools. In fact, such a function was manually set up by some 2016 MCC's tool developers (based on the outputs of the 2015 edition) to increase the reliability of their algorithm. Our objective is to make it possible to retrieve these results automatically from PNR, so that tool developers can use them even if they have not yet participated in the Model Checking Contest.

Towards Performance Benchmarking. Section 4 shows how some benchmarking of tools can be automated to check correctness of properties computation, that could even be enhanced with the exploitation of the outputs from the Model Checking Contest. This could lead to the elaboration of performance checking: tool developers might evaluate the impact of a given change in terms of performances.

Online model submission. Currently, the model submission procedure relies on a yearly call for models in the context of the MCC. Model submitters send their model forms in LaTeX to the model board before a given deadline. With PNR, we will eventually propose an online submission form. It would aim at becoming an easier alternative to submitting model information in a LaTeX file, while we will keep building the final

PDF version from LaTeX in the back end since this format provides us with numerous advantages in the management of the models.

6 Conclusion

This paper presents Petri Nets Repository (PNR), a large repository of 109 models, from which 706 instances (*i.e.* individual PNML files with different parameter values, when applicable) are derived. The collections of Petri nets are available to the community through a Web interface and a REST API.

Our objective is twofold. First, we are progressively building a reference benchmark that can be helpful to provide fair comparison between tools. Second, PNR is associated with programmatic ways to query it so that it is possible to elaborate oracles from the provided data. Petri Nets Repository is available at `http://pnrepository.lip6.fr`.

Acknowledgements. The authors thank Hubert Garavel for his helpful advice during the design of Petri Net Repository.

References

1. cURL, `https://curl.haxx.se`
2. Ashkenas, J.: CoffeeScript, `http://coffeescript.org`
3. Dolan, S.: jq, `https://stedolan.github.io/jq/`
4. ECMA: ECMA-404 The JSON Data Interchange Standard, `http://www.json.org`
5. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Techn. 2(2), 115–150 (2002)
6. Garavel, H.: The VLPN Benchmarck Suite, `http://cadp.inria.fr/resources/vlpn/`
7. Goud, R., van Hee, K.M., Post, R.D.J., van der Werf, J.M.E.M.: Petriweb: A Repository for Petri Nets. In: ICATPN. LNCS, vol. 4024, pp. 411–420. Springer (2006)
8. 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 (Oct 2009)
9. Kordon, F., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Chiardo, G., Hamez, A., Jezequel, L., Miner, A., Meijer, J., Paviot-Adet, E., Racordon, D., Rodriguez, C., Rohr, C., Srba, J., Thierry-Mieg, Y., Trịnh, G., Wolf, K.: Complete Results for the 2016 Edition of the Model Checking Contest. `http://mcc.lip6.fr/2016/results.php` (June 2016)
10. Kordon, F., Garavel, H., Hillah, L., Paviot-Adet, E., Jezequel, L., Rodríguez, C., Hulin-Hubard, F.: MCC'2015 - The Fifth Model Checking Contest. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) XI, 262–273 (2016)
11. Lightbend, Zengularity: Play Framework, `https://www.playframework.com`
12. M. Jones and J. Bradley and N. Sakimura: JSON Web Token (JWT), `https://tools.ietf.org/html/rfc7519`. Request for Comments 7519, Internet Engineering Task Force (2015)
13. MongoDB, Inc: MongoDB, `https://www.mongodb.com`