New features in CPN-AMI 3 : focusing on the analysis of complex distributed systems

A. Hamez, L. Hillah, F. Kordon, A. Linard, E. Paviot-Adet, X. Renault and Y. Thierry-Mieg
LIP6, Université Pierre & Marie Curie, 4 place Jussieu, 75252 Paris cedex 05, France
cpn-ami@lip6.fr, http://www.lip6.fr/cpn-ami

Due to the state-space size explosion problem, behavioral analysis techniques are difficult to scale up to industrial size problems. Our group couples research on analysis tools with an introspection on modeling and software engineering techniques.

CPN-AMI is an integrated development and analysis environment dedicated to Petri nets. The numerous services it offers are built by a homogeneous integration of tools developed internally, and third-party tools from partner universities. These tools include state of the art algorithms and data-structures. This third major release offers better support for modeling and analysis of very large systems.

New features in CPN-AMI 3

This paper briefly presents the new features in CPN-AMI 3. *PetriScript* responds to a need to program flexible and compositional Petri nets. *A symbolic model checker* allows to transparently use symmetries, to tackle larger systems. *A symbolic unfolder* allows to compute structural properties (such as boundedness), without state-space exploration. *A prototype support for PNML* enables connexion with other Petri net tools.

The PetriScript language PetriScript [5] has been designed to ease assembling and parametrization of Petri net modules. PetriScript allows to construct large models and test them in various configurations, by using a compositional bottom-up approach : small component's behaviors may be modeled separately, then assembled according to a certain configuration, prior to running analysis tools. We successfully applied this approach to model and analyse PolyORB [7], and this tool captures usage patterns identified in that case-study.

PetriScript's main purpose is to automate modeling operations such as merging or connecting places and transitions. To do so, it provides classical control instructions such as tests, loops,..., as well as a macro system to parameterize Petri nets.

Supported operations on Petri net objects are: creation, modification, connection, deletion and fusion of nodes. Creation and connection operations, combined with control instructions, are of particular interest when creating repeated patterns ; fusion operators are useful when assembling patterns and/or modules.

The fusion operation can merge either single nodes or lists of nodes. Lists are built by adding nodes one by one, or by using regular expressions on Petri net objects attributes. For example, you can insert into a list all places having a color domain equal to *color**.

So, by using an appropriate naming scheme and PetriScript, it is very easy to automate construction and assembly of Petri nets.

Model checking on the symbolic reachability graph This model-checking service exploits symmetries to offer better scaling up of verification. The principle is to construct an aggregated state-space graph (the "symbolic reachability graph" SRG), where nodes represent equivalence classes of states. Depending on the permittivity of the equivalence relation used, SRG nodes may represent an exponential number of "concrete" states, thus allow to scale up verification to industrial size examples.

Distributed systems frequently contain repeated component instances, that only differ by their identity. Typical examples are processes executing the same code, but having different *pid*, or memory addresses. Another example is large value domains, obtained by discretization of continuous system variables (e.g. altitude), of which only few values are critical for control ("never open landing gear above *Max* feet").

Analysis techniques to exploit such symmetries have been implemented, like in Murphi [8] or GreatSPN [4]. However these tools require the designer to identify admissible symmetries, leading to a cumbersome formalism, and making modifications of a model costly. We have developed a tool using algorithms described in [11] that allows to automatically extract all significant symmetry information, allowing the user to transparently use these techniques. This is particularly important in the context of reconfigurable models and models generated from higher level specifications, where such symmetry information is not present.

Thanks to strong collaborations with Torino and Alessandria, we have based our CPN-AMI tool on GreatSPN's implementation of SRG construction (development of the corresponding component is now shared between Paris and Torino). For full state-space generation, the user has thus simply to launch the appropriate service, that will detect any and all exploitable symmetries, before running GreatSPN and re-importing the results.

LTL model-checking is performed thanks to Spot [1] [3], a model-checking library with support for plugging in third party state-space generators, now coupled with GreatSPN (which doesn't natively support LTL).

Atomic properties of an LTL formula allow to characterize relevant system configurations. In our framework, atomic properties may be either event based, i.e. "transition T fires", or state-based, i.e. "place P contains N tokens".

Event-based properties are specified by referring to an existing transition of the Petri net, and can be refined to observe specific firings of the transition by providing a standard guard expression.

State-based properties are specified by adding *observation transitions* to a net : the atomic state property is considered true in any state such that the observation *would be* fireable. These added transitions are *never fired*, they do not impact on the net's behavior, thus they may not have output arcs. However, they may have as many input or inhibitor arcs as necessary, and a guard, offering great flexibility in the definition of the target states.

LTL model-checking is run using on-the-fly algorithms, and yields an error trace if the property is not preserved. The formula to be checked may have an impact on the symmetries that can be used to construct the symbolic reachability graph, thus it is analyzed before computation to find the maximal symmetries such that the truth of the formula can be checked using the symbolic reachability graph. The whole procedure is entirely automatic and transparent for end-users of the platform.

This tool has been successfully used to verify behavioral properties of an intermediate size system, the resource broker of the PolyOrb middleware [7]. Analysis of realistic configurations of this middleware could not have been performed using classical techniques, as shown in Figure 1 that exhibits the exponential reduction in state-space size due to symmetry exploitation as the number of threads allocated to PolyORB varies.

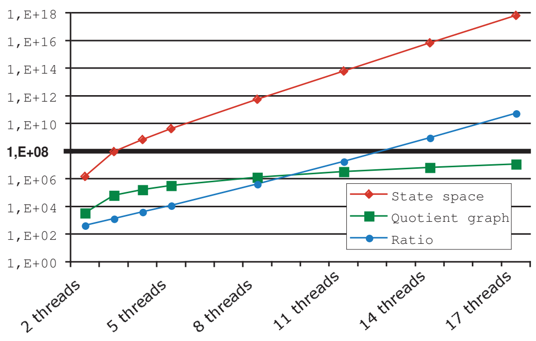

Figure 1. Evolution of concrete and symbolic state space sizes when varying parameters of the PolyOrb case study. 100 million states is a higher limit for most explicit techniques.

Optimized Colored nets unfolding As some structural properties cannot be verified on colored Petri nets, an operation, called *unfolding* is used to tranform these nets to P/T ones. Depending on the colored model, its unfolded net can contain dead parts whose size may vary from nothing to almost all nodes.

For example, in Well Formed Nets, integer expression, like $i \times j$ on arc valuations are not available, leading the modeler to use a pattern to emulate it. Unfolding of these patterns generates a huge number of transitions, but most of them are dead.

Several optimizations can be applied to remove these dead parts : removal of transitions guarded by `false`, removal of the maximal unmarked syphon and removal of orphaned marked places. All these are already implemented in our previous unfolder. However, optimization fails on huge unfolded nets since it needs to store the full unfolded net in memory.

So, our new unfolder uses a symbolic representation for places and transitions of the unfolded net, using Data Decision Diagrams [1], and an implicit one for arcs. The use of decision diagrams gives usually good performances both for execution time and memory usage, as operations on the symbolic representation make a heavy use of caching.

The `UniformizedTrain` [2] model makes a strong case for this method. Without optimizations, its unfolded net contains more than 10^5 places and 10^9 transitions. After optimization, it is reduced to 343 places

[1]`http://spot.lip6.fr`

and 202 transitions. Execution ends after only 7 seconds and uses less than 72 MB of memory.

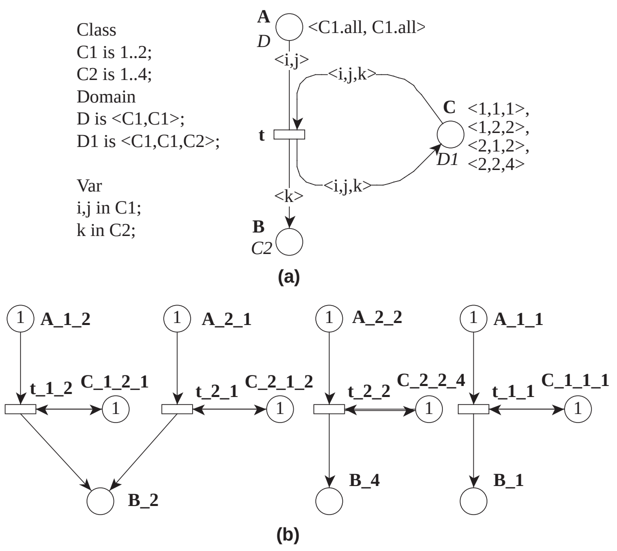

Figure 2. Multiplication table.

Place C in Figure 2 (a) models a multiplication table. Typically, unfolding this place leads to a great number of ordinary places (n^4 places if n is the multiplication table length) and only a few of them (n^2) are useful (see Figure 2 (b)).

Experimentation of the ISO/IEC-15909 exchange standard CPN-AMI serves as a testbench for the new PNML [9] (Petri Net Markup Language) standard (ISO/IEC-15909 part 2) since we are experimenting a prototype implementation. PNML is a XML-based universal transfer syntax for Petri nets to allow *interoperability* among Petri nets tools.

The interest of this prototype is to explore new ways to sustain PNML development and easy integration into various Petri net tools, backing the Standard. To achieve these objectives, OMG's Model-Driven Development principles [10] and techniques have been put into practice. [6] sets the rationales for such an approach.

References

[1] J-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P-A. Wacrenier. Data decision diagrams for petri net analysis. In J. Esparza and C. Lakos, editors, *23rd Intl. Conf. on Applications and Theory of Petri Nets*, number 2360 in LNCS, pages 101–120. Springer Verlag, 2002.

[2] A. de Groot, J. Hooman, F. Kordon, E. Paviot-Adet, I. Vernier-Mounier, M. Lemoine, G. Gaudiere, V. Winter, and D. Kapur. A survey: Applying formal methods to a software intensive system. In *The 6th IEEE Int. Symposium on High-Assurance Systems Engineering*, pages 55–64. IEEE Computer Society, 2001.

[3] A. Duret-Lutz and D. Poitrenaud. Spot: an extensible model checking library using transition-based generalized Büchi automata. In *Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, pages 76–83, 2004.

[4] GreatSPN: GRaphical Editor, Analyzer for Timed, and Stochastic Petri Nets. `http://www.di.unito.it/~greatspn/`.

[5] A. Hamez and X. Renault. *PetriScript Reference Manual.* LIP6, `http://www-src.lip6.fr/logiciels/mars/CPNAMI/MANUAL_SERV`.

[6] L. Hillah, F. Kordon, L. Petrucci, and N. Trèves. Building an api for iso/iec 15909, based on model engineering techniques. *Petri Net Newsletter*, 69:22–40, October 2005.

[7] J. Hugues, Y. Thierry-Mieg, F. Kordon, L. Pautet, S. Baarir, and T. Vergnaud. On the Formal Verification of Middleware Behavioral Properties. In *Proceedings of the 9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'04)*, volume ENTCS 133, pages 139 – 157. Elsevier, September 2004.

[8] N. Ip and D. Dill. Verifying systems with replicated components in murphi. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 147–158. Springer Verlag, 1996.

[9] E. Kindler. Software and Systems Engineering - High-level Petri Nets. Part2: Transfert Format. Working Draft for the Int. Standard ISO/IEC 15909 Part 2 - Version 0.6.3, June 2005.

[10] OMG. MDA Guide Version 1.0.1, document no: omg/2003-06-01, 2003.

[11] Y. Thierry-Mieg, C. Dutheillet, and I. Mounier. Automatic symmetry detection in well-formed nets. In *Proc. of ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 82–101. Springer Verlag, June 2003.