The ASCoVeCo State Space Analysis Platform: Next Generation Tool Support for State Space Analysis[⋆]

Invited Tutorial

Lars M. Kristensen and Michael Westergaard

Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark,
Email: {kris,mw}@daimi.au.dk

Extended Abstract

State space analysis is one of the main approaches to model-based verification of concurrent systems and is one of the most successfully applied analysis methods for Coloured Petri Nets (CP-nets or CPNs) [13, 16, 17]. The basic idea of state space exploration and analysis is to compute all reachable states and state changes of the concurrent system under consideration and represent these as a directed graph. From a constructed state space it is possible to verify and analyse a large class of behavioural properties by considering, e.g., the standard behavioural properties of CP-nets, traverse a constructed state space by means of user-defined queries, or conduct LTL and CTL model checking [4, 10]. Another main advantage of state space analysis is that it can be supported by computer tools in a highly automatic way, and it can provide counter examples demonstrating why the system does not have a certain property.

The main limitation of using state spaces to verify behavioural properties of systems is the *state explosion problem* [24], i.e., that state spaces of systems may have an astronomical number of reachable states which means that they are too large to be handled with the available computing power (CPU speed and memory). Methods for alleviating this inherent complexity problem is an active area of research and has led to the development of a large collection of *state space reduction methods*. These methods have significantly broadened the class of systems that can be verified and state spaces can now be used to verify systems of industrial size. Some of these methods [2, 15, 14, 25] have been developed in the context of the CPN modelling language. Other methods (e.g., [23, 22, 26, 11]) have been developed outside the context of the CPN modelling language, but state space reduction methods are generally independent of any concrete modelling language and hence also applicable for CP-nets.

A computer tool supporting state spaces must implement a wide range of state space reduction methods since no single reduction method works well on all systems. Both CPN Tools [5] and its predecessor Design/CPN [6] have supported state space analysis in it most basic form and experimental prototype libraries have been developed supporting state space reduction methods such as the symmetry method [15, 18], the equivalence method [14, 19], time condensed state spaces [3], and the sweep-line method [2, 21]. The software architectures of CPN Tools and Design/CPN have, however, made it difficult to support a collection of state space reduction methods in a coherent manner in these tools.

This tutorial presents the ASCoVeCo State Space Analysis Platform (ASAP) which is currently being developed in the context of the ASCoVeCo research project [1]. ASAP represents the next generation of tool support for state space exploration and analysis of CPN

⋆ Supported by the Danish Research Council for Technology and Production.

models. The aim and vision of ASAP is to provide an open platform suited for research, educational, and industrial use of state space exploration. This means that the ASAP will support a wide collection of state space exploration methods and have an architecture that allows the research community to extend the set of supported methods. Furthermore, ASAP will be sufficiently mature to be used for educational purposes, including teaching of advanced state space methods, as well as sufficiently mature to be used in industrial projects as has been the case with CPN Tools and Design/CPN. ASAP is a stand-alone tool and is able to load models created with CPN Tools. ASAP will be available for Windows XP/Vista, Linux, and Mac OS X.

The ASAP platform consists of a graphical user interface (GUI) and a state space exploration engine (SSE engine). Figure 1(left) shows the software architecture of the graphical user interface which is implemented in Java based on the Eclipse Rich Client Platform (RCP) [9]. The software architecture of the SSE engine is shown in Fig. 1(right). It is based on Standard ML (SML) and relies on the CPN simulator used in CPN Tools. The SSE engine implements the state space Exploration and model checking algorithms supported by ASAP. The state space exploration and model checking algorithms implemented rely on a set of Storage and Waiting Set components for efficient storage and exploration of state spaces. Furthermore, the SSE engine implements the Query Languages(s) used for writing state space queries and to verify properties.

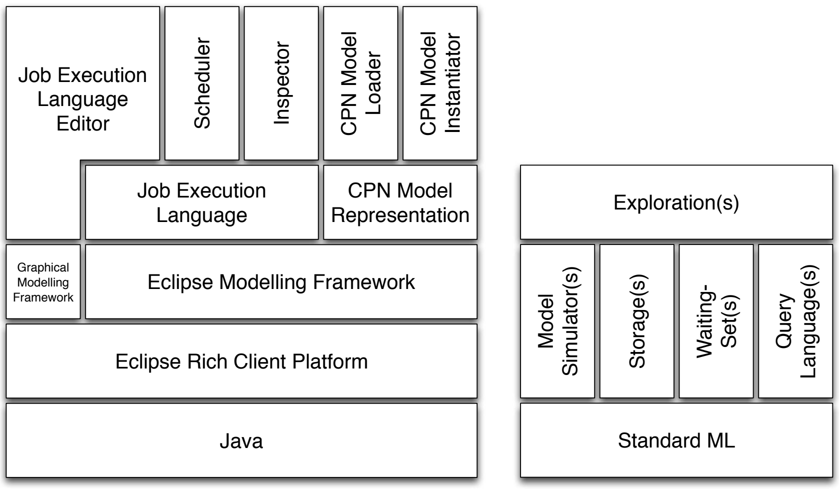


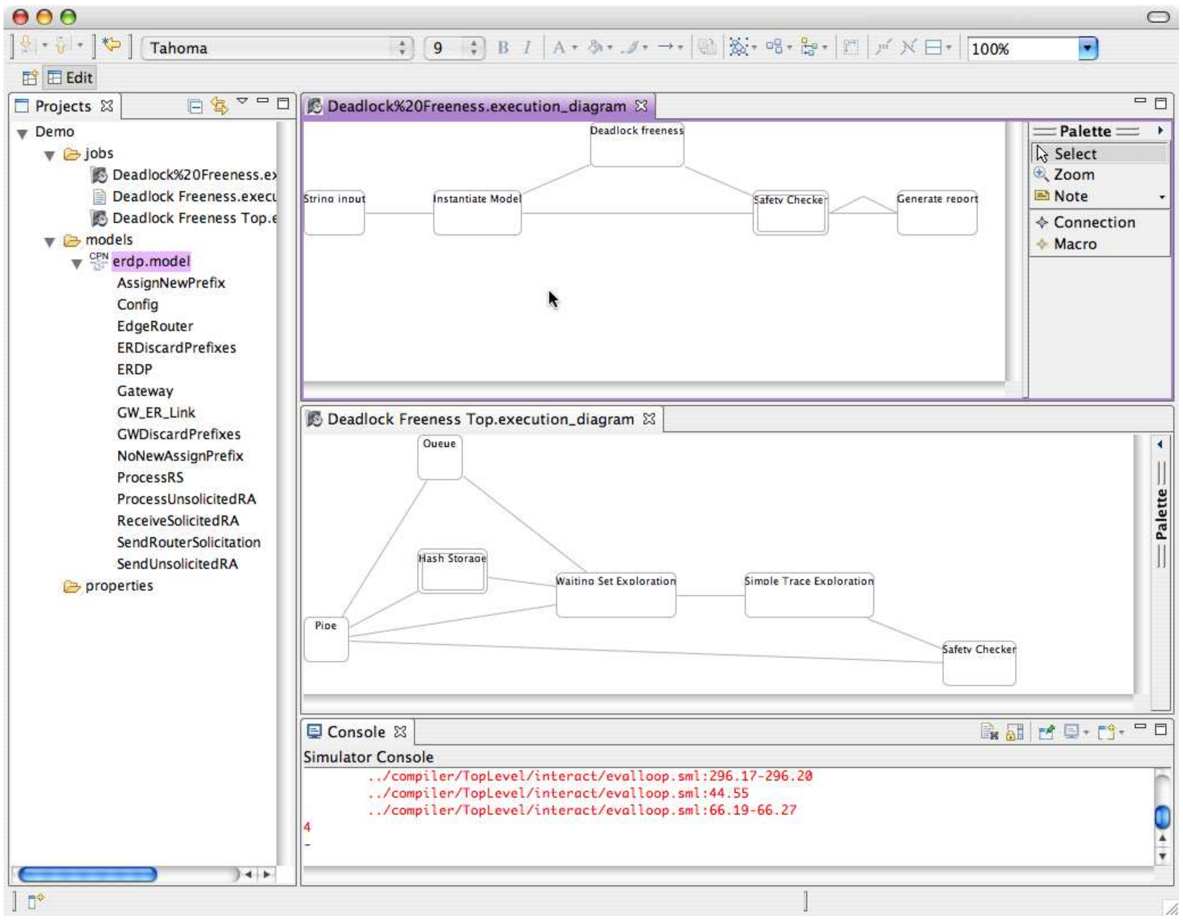
Fig. 1. ASAP platform architecture: GUI (left) and SSE engine (right).

The ASAP GUI makes it possible to create and manage *verification projects* consisting of a collection of *verification jobs*. The GUI has three different *perspectives* for working with verification projects:

– An *editing perspective* for creating and editing verification jobs.
– An *execution perspective* for controlling the execution of verification jobs.
– An *inspection perspective* for inspecting and interpreting analysis results.

Verification jobs are constructed and specified using the verification Job Execution Language (JEL) and the JEL Editor. JEL is a graphical language inspired by data-flow diagrams that makes it possible to specify the CPN models, queries, state space explorations, and processing of analysis results that constitute a verification job. JEL and the JEL Editor are implemented using the Eclipse Modelling Framework (EMF) [8] and the Graphical Modelling Language [7] (GMF). The ASAP GUI additionally has a Model Loader component and a Model Instantiation component that can load and instantiate CPN models created with CPN Tools. Hence, models created using CPN Tools can be used directly with ASAP and since ASAP relies on the same underlying CPN simulator as CPN Tools, there is no CPN semantical gap between the two tools. It is worth noticing that it is only the CPN Model Loader, CPN Model Instantiator, and CPN Model Representation components that are specific to CPN models. The other components are independent of the CPN modelling formalism.

Figure 2 shows a snapshot of the graphical user interface in the editing perspective. The user interface consists of three main parts apart from the usual menus and tool-bars at the top.



```
        ../compiler/TopLevel/interact/evalloop.sml:296.17-296.20
        ../compiler/TopLevel/interact/evalloop.sml:44.55
        ../compiler/TopLevel/interact/evalloop.sml:66.19-66.27
4
-
```

Fig. 2. Snapshot of the graphical user interface – editing perspective.

To the left is an overview of the verification projects loaded, in this case just a single project named Demo is loaded. A verification project consists of a number of verification jobs, models, and queries. In this case there are three verification job all concerned with checking deadlock properties. A CPN model named erdp is loaded, which is a CPN model of an edge router discovery protocol [20]. At the bottom of the user interface is a console, which makes it possible to interact directly with the SSE engine using SML. This makes it easy to experiment with the tool and to issue queries that need not be stored as part of the verification project. The large area at the top-right is the editing area. The editing area can be used to edit queries and verification jobs. In this case two jobs are being edited and the two windows shows the graphical representation in JEL of the parts being edited.

ASAP has an interface to the CPN simulator providing a set of primitives that makes it easy to access the transition relation of the CPN model and thereby augment the platform with new state space exploration and model checking algorithms. The primitives available is specified by the `MODEL_SIMULATOR` SML signature (interface) listed in Fig. 3. The signature deals with `state`s and `event`s (ll. 3–4). It is possible to get the initial states of the model (l. 9). Each state is represented as the actual state and all enabled events in that state. In order to accommodate non-deterministic formalisms, we allow a number of initial states, so a list of states is returned instead of just a single state. From a state and an event, it is possible to get (all) successor states (l. 12), and from a state and a sequence (list) of events, it is possible to get (all) states that are the result of executing the sequence of events from the specified state (l. 16). If a user tries to execute an event that is not enabled, the `EventNotEnabled` exception (l. 6) is raised in both `nextStates` and `executeSequence`.

```
 1  signature MODEL_SIMULATOR =
 2  sig
 3      eqtype state
 4      eqtype event
 5
 6      exception EventNotEnabled
 7
 8      (* --- get the initial states and enabled events in each state --- *)
 9      val getInitialStates : unit -> (state * event list) list
10
11      (* --- get the successor states and enabled events in each successor state --- *)
12      val nextStates : state * event -> (state * event list) list
13
14      (* --- execute an event sequence and return the list of resulting states --- *)
15      (* --- and enabled events --- *)
16      val executeSequence : state * event list -> (state * event list) list
17  end
```

Fig. 3. SML signature of the model simulator interface.

The SSE engine currently implements full state space exploration, bit-state hashing [11], hash compaction [26], the comback method [25], state caching [12], and the equivalence method [14] based on canonicalisation of equivalent markings. All these methods have been implemented using the CPN simulator interface in Fig. 3 and the storage and waiting set components provided by the SSE engine. Currently only verification of safety properties is supported by ASAP. State space exploration can be done both breadth-first and depth-first. It is worth observing that the model simulator interface allows the state space exploration and model checking algorithms to be implemented such that they are independent from a concrete modelling language.

As part of the development of ASAP, we have also developed a test and benchmarking tool containing a collection of small, medium, and large CPN models. This benchmarking suite makes it simple to profile the performance (e.g., time and space usage) of state space methods and their implementation. The benchmarking tool includes an SQL database where profiling data can be stored and a web-based GUI that makes it possible to query the database and display the results graphically.

During the CPN workshop we plan to make a first technology preview release of ASAP. For version 1.0 we plan to complete the implementation of the simulator interface in the SSE engine, and additionally implement the sweep-line method [2, 21], time condensed state spaces [3], and CTL and LTL model checking [10, 4]. Also, we plan to implement and provide the same set of state space query functions available in the state space tool of CPN Tools. In the user interface we will complete the implementation of the JEL Editor, implement an SML Query Editor, and implement the inspection perspective. Also, we will implement a functionality similar to the *state space report* and a simple interactive and automatic visualisation of state spaces as known from CPN Tools.

Acknowledgements. The authors wish to acknowledge the work of Surayya Urazimbetova and Mads K. Kjeldsen who are significantly contributing to the development of ASAP via their employment as student programmers in the ASCoVeCo project.

References

1. The ASCoVeCo Project. `www.daimi.au.dk/ ascoveco`.
2. S. Christensen, L.M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In *Proc. of TACAS'01*, volume 2031 of *LNCS*, pages 450–464. Springer-Verlag, 2001.
3. S. Christensen, L.M. Kristensen, and T. Mailund. Condensed State Spaces for Timed Petri Nets. In *Proc. of 22nd International Conference on Application and Th eory of Petri Nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 101–120. Springer-Verlag, 2001.
4. E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, 1999.
5. CPN Tools. `www.daimi.au.dk/CPNTools`.
6. Design/CPN. `www.daimi.au.dk/designCPN`.
7. Eclipse Graphical Modelling Framework (GMF). `www.eclipse.org/modeling/gmf/`.
8. Eclipse Modelling Framework (EMF). `www.eclipse.org/modeling/emf/`.
9. Eclipse Rich Client Platform (RCP). `www.eclipse.org/home/categories/rcp.php`.
10. E. A. Emerson. *Temporal and Modal Logic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 995–1072. Elsevier, 1990.
11. G.J. Holzmann. An Analysis of Bitstate Hashing. *Formal Methods in System Design*, 13:289–307, 1998.
12. C. Jard and T. Jeron. Bounded-memory Algorithms for Verification On-the-fly. In *Proc. of CAV'91*, volume 575 of *LNCS*, pages 192–202. Springer-Verlag, 1991.
13. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1:Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
14. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2: Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
15. K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. *Formal Methods in System Design*, 9, 1996.
16. K. Jensen and L.M. Kristensen. *Coloured Petri Nets – Modelling and Validation of Concurrent Systems.* Springer-Verlag, In preparation.
17. K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *In International Journal on Software Tools for Technology Transfer (STTT)*, 9(3-4):213–254, 2007.

18. J.B. Jørgensen and L.M. Kristensen. Computer Aided Verification of Lamports Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):714–732, 1999.
19. J.B. Jørgensen and L.M. Kristensen. Verification of coloured petri nets using state spaces with equivalence classes. In *Petri Net Approaches for Modelling and Validation*, volume 1 of *LINCOM Studies in Computer Science*, chapter 2, pages 17–34. Lincoln Europa, 2003.
20. L.M. Kristensen and K. Jensen. Specification and validation of an edge router discovery protocol for mobile ad-hoc networks. In *Proc. of Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 248–269. Springer-Verlag, 2004.
21. L.M. Kristensen and T. Mailund. A Generalised Sweep-Line Method for Safety Properties. In *Proc. of FME'02*, volume 2391 of *LNCS*, pages 549–567. Springer-Verlag, 2002.
22. D. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1993.
23. A. Valmari. Error Detection by Reduced Reachability Graph Generation. In *Proceedings of the 9th European Workshop on Application and Theory of Petri Nets*, pages 95–112, 1988.
24. A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.
25. M. Westergaard, L.M. Kristensen, G.S. Brodal, and L.A. Arge. The ComBack Method – Extending Hash Compaction with Backtracking. In *Proc. of ICATPN'07*, volume 4546 of *Lecture Notes in Computer Science*, pages 445–464. Springer-Verlag, 2007.
26. P. Wolper and D. Leroy. Reliable Hashing without Collision Detection. In *Proc. of CAV'93*, volume 697 of *LNCS*, pages 59–70. Springer-Verlag, 1993.