*ARCS 2012 Workshops (G. Mühl, J. Richling, A. Herkersdorf eds.)*
*8th Workshop on Dependability and Fault Tolerance*
*Lecture Notes in Informatics (LNI), Volume P-200, © Gesellschaft für Informatik, Bonn 2012*

# Model-based Testing of Autonomous Systems based on Coloured Petri Nets

Raimar Lill, Francesca Saglietti

Chair of Software Engineering
University of Erlangen-Nuremberg
Martensstr. 3
91058 Erlangen, Germany
raimar.lill@informatik.uni-erlangen.de
saglietti@informatik.uni-erlangen.de

**Abstract:** The use of autonomous systems, including cooperating agents, is indispensable in certain fields of application. Nevertheless, the verification of autonomous systems still represents a challenge due to lack of suitable modelling languages and verification techniques. To address these difficulties, different modelling languages allowing concurrency are compared. Coloured Petri Nets (CPNs) are further analysed and illustrated by means of an example modelling autonomous systems. Finally, some existing structural coverage concepts for Petri Nets are presented and extended by further criteria tailored to the characteristics of CPNs.

## 1 Introduction

For reasons of flexibility and time efficiency modern software-based applications tend to decentralize the target functionality among a number of cooperating autonomous subsystems. This results in highly decoupled system units and an increasing multiplicity of interplay. Examples include mobile agents, as typical for robot applications.

In general, the term autonomy is taken to indicate entities (humans, populations or technical systems) capable of providing themselves with their own laws. This includes the moral responsibility of individuals for their actions, the self-government of human populations, as well as the capacity of technical systems to make rational and informed decisions.

The optimal degree of autonomy lies between the two extremes represented by fully decoupled agents and fully central controllers. It relies on essential rules of co-existence involving different communication patterns like the following ones:

- **Separation of concerns:** different agents may autonomously cooperate by carrying out parallel individual and independent sub-tasks.
- **Synchronization:** as soon as each independent sub-task has been separately and autonomously concluded, a synchronization mechanism establishes appropriate communication between the cooperating parts for the purpose of organizing the next common activity.
- **Coordination:** the communication of synchronized agents must be supported by a coordination mechanism determining a consensus on how to proceed. Typically, coordination tasks may be required to avoid or resolve conflicts due to concurrency, e.g. by mutual exclusion in case of shared resources.
- **Delegation:** for the purpose of carrying out a complex activity, a process may delegate a sub-task to another process by invoking its support through synchronous or asynchronous message passing.
- **Feedback:** agents may mutually influence their local or global behaviour by providing information on particular operating scenarios recently experienced, e.g. information on road traffic exchanged among communicating vehicles. The degree of influence exercised may depend on the amount and consistency of information broadcasted or by the number of agents broadcasting it.

From the perspective of verification, one of the major challenges posed by autonomous systems refers to their inherent lack of compositionality. In fact, understanding the local behaviour of each individual system part does not suffice to comprehend all potential implications on global behaviour. In other words, the classical verification strategy based on separation of concerns provides only limited support here, as it does not allow for the deduction of global properties by derivation or composition of local properties.

The observable *emergent behaviour* may reflect the intended target behaviour to be achieved by cooperation, but sometimes it may also reveal as surprising, undesired, or even unsafe by cascading of unpredicted side effects.

In order to model concurrent behaviour of cooperating autonomous systems capturing the interaction patterns mentioned above, an appropriate modelling notation is to be selected, capable of supporting both the representation and the analysis of the system considered. As the observation and measurement of system behaviour in real situations is crucial for testing autonomous systems, representative test scenarios have to be derived from the modelling notation selected.

The rest of the article is structured as follows: in section 2 different modelling notations are compared w.r.t. the considerations mentioned above. In section 3 CPNs are introduced by illustrating their syntactical and semantical highlights. In section 4 a CPN model of a system consisting of autonomous robots is introduced and illustrated in detail. This example confirms the decision taken in favour of CPNs and emphasizes the value of the corresponding modelling tool CPN Tools [JKW07]. In chapter 5 existing coverage criteria for Petri Nets are presented before introducing novel coverage criteria explicitly tailored to CPNs by focusing on token colours and on the occurrence of interleaving events in the net.

## 2 Comparison of Modelling Notations for Autonomous Systems

For the purpose of comparing different notations modelling autonomous systems, the following evaluation criteria were considered:

- **Understandability** concerns the clarity of a modelling representation, including the availability of graphical visualization techniques.
- **Well-definedness** ensures a unique interpretation of the underlying operational semantics as offered by formal languages and required for the analyzability of central properties like state reachability. A well-defined semantics is the basis for tool support concerning the analysis of model data.
- **Scalability** concerns the ease of widening the problem dimensions without prohibitively increasing problem complexity and representational size.
- **Testability** concerns the ease of capturing and visualizing the multiplicity of behaviour by adequate coverage concepts and metrics; central to these activities are tools supporting the editing, import and export of data.

In the light of the criteria mentioned above a number of well-known and widely used modelling languages allowing for concurrent behaviour are compared in the following.

- **Process algebras** like CCS [Mi80] or CSP [Ho78] provide a formal algebraic description of model concurrency aspects capturing communication by explicit algebraic *send* and *receive* operators.
- **UML activity diagrams** provide a semi-formal, graphical representation of activity flows which can be extended by dedicated profiles to include real-time concurrent behaviour; among them are the profile "Schedulability, Performance and Time" (SPT) [Om05], which allows for quantitative performance predictions, and its UML 2 successor "Modelling and Analysis of Real-Time and Embedded Systems" (MARTE) [Om11].
- **Petri Nets** [Mu89] formalize concurrent behaviour by graphical entities representing actions (so-called *transitions*), as well as pre- and post-conditions (so-called *places*). The fulfilment of conditions is represented by tokens marking corresponding places. In this way, Petri Nets succeed in capturing the interactions of autonomous systems by decentralizing the information referring to their states.
- **Coloured Petri Nets** [JKW07] are an extension of Petri Nets allowing for the refinement of pre- and post-conditions while supporting scalability by use of different token types (so-called *token colours*). Details are presented in section 3.

Process algebras possess a well-defined algebraic theory making them adequate for static analysis purposes. The lack of visual representation, however, does not offer sufficient support to intuitive comprehension and graphical coverage concepts.

UML activity diagrams are widely used and provide an intuitive visualization of concurrency aspects. They lack, however, a formal operational semantics [SH05], as required by rigorous analysis techniques.

Petri Nets are well suited for visualization and analysis purposes [Mu89]; unfortunately, the decentralized representation of states by generic tokens may lead to an exponential growth of places and transitions. In other words, their scalability may be severely limited.

Compared to the other approaches, Coloured Petri Nets reveal as a promising candidate for modelling autonomous systems: CPNs are based on a sound mathematical basis resulting in unambiguous models that can be analyzed by simulation or formal techniques. Furthermore, they benefit from the visual clarity of Petri Nets by sharing their graphical elements, while overcoming the Petri Nets limitations concerning scalability: in fact, the use of specific tokens supports a compact state representation. In other words, when compared with Petri Nets CPNs offer higher scalability thanks to the higher level of the language.

Consequently, CPNs are selected as a modelling notation to be further investigated.

## 3 Coloured Petri Nets

Coloured Petri Nets [Je94] differ from the original Petri Net notation by type-specific tokens. Depending on the colour set associated with a particular token, this token may assume different values denoted as the *token colours*. Additionally, transitions and arcs can be annotated by conditional expressions controlling the transition firing. In more detail, a CPN is a 9-tuple (P, T, A, Σ, V, C, G, E, I), where

- P denotes a finite set of nodes $p \in P$ denoted as *places*;
- T denotes a disjoint finite set of nodes $t \in T$ denoted as *transitions*, i.e. $P \cap T = \varnothing$;
- $A \subseteq P \times T \cup T \times P$ denotes a set of directed *arcs* connecting either places with transitions or transitions with places;
- Σ denotes a finite set of non-empty sets denoted as *colour sets*; the elements of each colour set are denoted as *colours*;
- V denotes a finite set of *variables*, each varying over a colour set, i.e. $type[v] \in \Sigma \; \forall \; v \in V$;
- C: $P \rightarrow \Sigma$ denotes a function (the so-called *colour function*) attaching to each place $p \in P$ a colour set $C(p) \in \Sigma$; $C(p)_{MS}$ denotes the multi-set over C(p) where each colour of the colour set C(p) may occur more than once;
- G: $T \rightarrow EXP_V$ is a function attaching to each transition $t \in T$ a *guard,* i.e. a conditional expression G(t) over variables $v \in V$ with type[G(t)] = Bool;
- E: $A \rightarrow EXP_V$ is a function attaching to each arc $a \in A$ an *expression* E(a) over variables $v \in V$, with $type[E(a)] \in C(p)_{MS}$, where p is a place connected with arc a;
- I: $P \rightarrow C(p)_{MS}$ denotes a function attaching to each place $p \in P$ a so-called *initial marking* M(p) of type $C(p)_{MS}$.

The dynamic behaviour of a CPN is given by state changes due to successive firings of transitions.

For a given transition $t \in T$ a *variable binding* associates each variable occurring in at least one input arc expression of t with one of the colours belonging to the colour set of the corresponding input place(s).

A transition $t \in T$ with input places $p_i$ and input arcs $a_i$: $p_i \rightarrow t$, $i \in \{1,\ldots,k(t)\}$ is *enabled w.r.t. a particular variable binding* if and only if

- the value of G(t) w.r.t. the given variable binding is true and
- for each colour of an input place the colour multiplicity in the multi-set obtained by evaluating the input arc expression $E(a_i)$ w.r.t. the given variable binding is not higher than the number of tokens of the same colour in the corresponding input place.

After *firing* a transition t w.r.t. an *enabling variable binding*, a new marking is obtained from the previous marking by

- removing from each input place as many tokens for each of its colours as indicated by the colour multiplicity in the multi-set resulting by evaluating the corresponding input arc expression w.r.t. the enabling variable binding;
- adding to each output place as many tokens for each of its colours as indicated by the colour multiplicity in the multi-set resulting by evaluating the corresponding output arc expression w.r.t. the enabling variable binding.

## 4 The Robot Factory

In the following, a CPN model of an autonomous robot system is presented. The model was built using CPN Tools [JKW07]. Net annotations are expressed in the language CPN ML which extends the functional programming language SML [Mi97] by additional constructs for defining the CPN elements presented above. As commonly used in Petri Net visualizations, places and transitions are represented by circles and rectangles respectively. Guards are encapsulated in square brackets, arc inscriptions are annotated along the corresponding arcs and places are assigned corresponding colour sets by cross products of pre-defined basic colour sets.

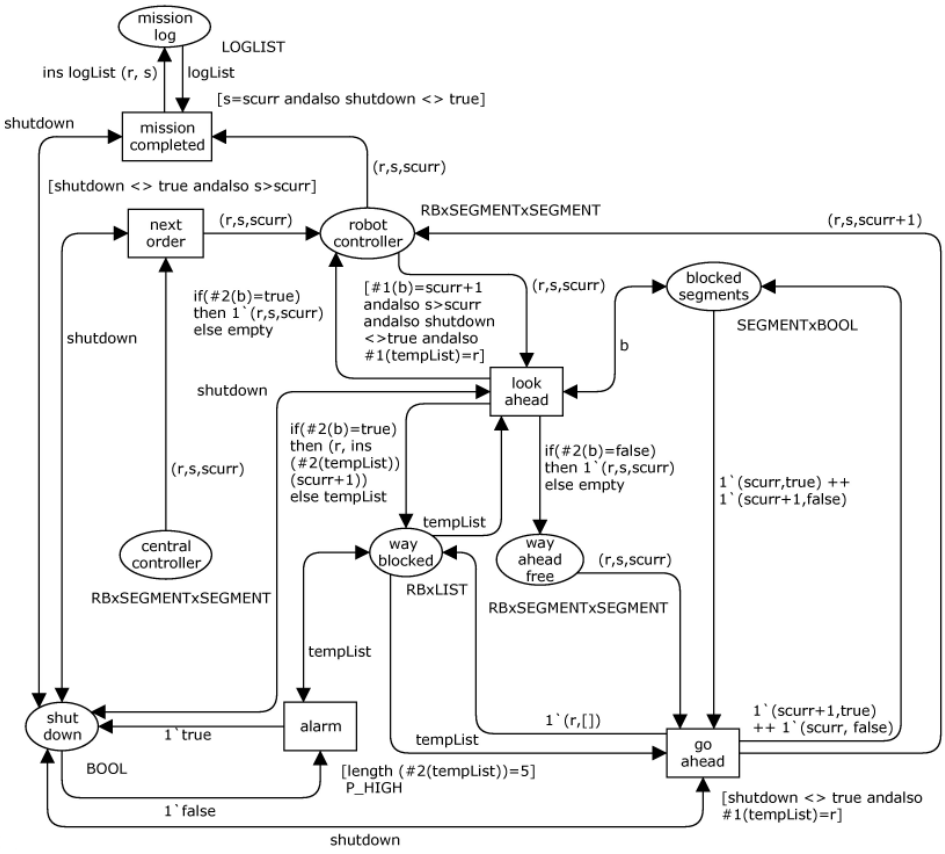


Figure 1: CPN Model of the Robot Factory

The example depicts a factory where robots of type *RB* move from one area – a so-called *SEGMENT* – to another one. A central controller sends orders *(r, s, scurr)* to the local robot controllers where

- *r* represents the *robot* addressed,
- *s* represents the *target segment* of robot *r*, and
- *scurr* represents the *current location* of robot *r*.

To limit the complexity of the example, the robots only move along narrow lanes and obstacle passing manoeuvres are not included, thus limiting the robot behaviour to forward movement. It should be noted that the central controller does not hinder system autonomy, as it only provides the robots with orders, without dictating their behaviour for fulfilling their missions. Robots autonomously check whether their way to the next segment is free or blocked (modelled by transition *look ahead*). A robot can access this information thanks to its sensors (e.g. by camera or laser techniques). The functioning of the sensors is not explicitly modelled here, but could be easily included, e.g. by a hierarchical design.

For the purpose of deriving valuable test scenarios from the modelled CPN, the necessary sensor data are simulated by the place *blocked segments* containing the local knowledge of each robot about the segment lying ahead.

If the sensors of a robot detect a blockade, the information about the target segment and the robot is logged in the place *way blocked* by maintaining a list of blockades for every robot. Five continuously detected blockades of a certain robot trying to access a specific segment raise an alarm (modelled by transition *alarm*). To ensure this, CPN Tools allows to assign different priorities to transitions. The alarm was prioritized to avoid the interference of other transitions resulting in a potential alarm delay. Raising an alarm prevents any further transition from firing, requiring the intervention of a human operator. If a blockade can be resolved before raising an alarm, the blockades are flushed from the corresponding list. This is realized by the use of list colour sets in analogy to abstract list data types in programming languages.

As soon as the way is not blocked by obstacles or other robots, a robot can move on (modelled by transition *go ahead*) resulting in an update of its current location *scurr*.

If a robot eventually detects that its current segment position *scurr* equals its target position *s*, the given order is completed (modelled by transition *mission completed*) and logged.

The strength of modelling the behaviour of autonomous systems by CPNs lies in their ability to capture a wide multiplicity of possible execution traces by a relatively compact representation. This results in a flexible format that can be easily adapted to application-specific scenarios by changing the configuration of the CPNs (e.g. by varying the number of segments or robots).

Possible execution traces regarding an initial net marking can be statically analyzed by exploring the reachability graph [Je94] or dynamically analyzed w.r.t. different test coverage criteria, as presented in the next chapter.

## 5 Testing Coverage Criteria for CPNs

Several CPN coverage criteria can be defined to determine appropriate scenarios during testing. For example, [ZH00] and [ZH02] introduce a number of coverage criteria originally defined for Predicate-Transition Petri Nets [GL81] which inspired the following classes of CPN testing criteria.

- **Transition-based coverage criteria** focus on the occurrences of transition firings, requiring individual transition firings as well as sequences of transition firings of given length.
- **State-based coverage criteria** focus on individual states or on state representatives of pre-defined state classes.

- **Flow-based coverage criteria** focus on the production or consumption of at least one token (regardless of its colour) by individual transition firings or by sequences of transition firings.

The above mentioned criteria can be extended to include the following coverage demands concerning CPN-specific entities, namely colour sets and variable bindings.

- **Colour-based coverage criteria** focus on the production or consumption of tokens belonging to pre-defined colour sets.
- **Event-based coverage criteria** focus on the individual or sequential occurrence of CPN events, where an *event* is defined as a transition together with an enabling variable binding.

The information needed to measure the coverage regarding the criteria presented above can be extracted from models created in CPN Tools by the framework Access/CPN [WK09] which permits to access the internal model data. It is planned to apply this framework in order to generate test cases fulfilling the above mentioned CPN coverage criteria by means of evolutionary techniques. Analogous approaches have already been successfully applied to structural code coverage [OS06], to integration testing [SP10], as well as to the filtering of operational experience for the purpose of reliability assessment [Sö10].

## 6 Conclusion

This article compares different modelling notations in terms of their expressive power and graphical support concerning structural testing of autonomous systems. The comparative evaluation of a number of alternative notations resulted in the selection of Coloured Petri Nets as the most promising option. As an example, a simple version of a robot factory was modelled by a CPN illustrating the benefits offered by its high scalability. Finally, a number of different CPN-based coverage criteria were introduced which will provide the basis for future research focused on the automatic generation of adequate testing scenarios.

**Acknowledgement:** It is gratefully acknowledged that part of the work reported was sponsored by the European Union Research Programme ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems), project R3-COP (Resilient Reasoning Robotic Co-operating Systems).

## References


[GL81] Genrich, H. J.; Lautenbauch, K.: System Modelling with High-Level Petri Nets. In: Theoretical Computer Science, Vol. 13, Issue 1. Elsevier, 1981; pp. 109-136.

[Ho78] Hoare, C. A. R.: Communicating Sequential Processes. In: Communications of the ACM, Vol. 21, No. 8. ACM Digital Library, 1978; pp. 666-677.

[Je94] Jensen, K.: An Introduction to the Theoretical Aspects of Coloured Petri Nets. In (de Bakker, J. W.; de Roever, W.-P.; Rozenberg, G. Eds.): A Decade of Concurrency - Reflections and Perspectives, Proc. REX School/Symposium, Noordwijkerhout, the Netherlands, 1993. Vol. LNCS 803, Springer-Verlag, 1994; pp. 230-272.

[JKW07] Jensen, K.; Kristensen, L. M.; Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In: International Journal on Software Tools for Technology Transfer (STTT), Vol. 9, No. 3-4. Springer-Verlag, 2007; pp. 213-254.

[Mi80] Milner, R.: A Calculus of Communicating Systems. Vol. LNCS 92, Springer-Verlag, 1980.

[Mi97] Milner, R. et al.: The Definition of Standard ML (Revised). MIT Press, 1997.

[Mu89] Murata, T.: Petri Nets: Properties, Analysis and Applications. In: Proc. IEEE, Vol. 77, No. 4. IEEE, 1989; pp. 541-580.

[Om05] Object Management Group: UML Profile for Schedulability, Performance and Time Specification. Version 1.1, formal/05-01-02, 2005.

[Om11] Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Version 1.1, formal/2011-06-02, 2011.

[OS06] Oster, N.; Saglietti, F.: Automatic Test Data Generation by Multi-Objective Optimisation. In (Górski, J. Ed.): Proc. 25th Int. Conf. on Computer Safety, Reliability, and Security, SAFECOMP 2006, Gdanks, Poland, 2006. Vol. LNCS 4166, Springer-Verlag, 2006; pp. 426-438.

[SH05] Störrle, H.; Hausmann, J. H.: Towards a Formal Semantics of UML 2.0 Activities. In (Liggesmeyer, P.; Pohl, K.; Goedicke, M. Eds.): Proc. Software Engineering 2005, Essen, Germany, 2005. Vol. LNI 64, Köllen Verlag, 2005; pp. 117-128.

[SP10] Saglietti, F.; Pinte, F.: Automated Unit and Integration Testing for Component-based Software Systems. In: Proc. Workshop on Dependability and Security for Resource Constrained Embedded Systems (D&S4RCES'10), Vienna, Austria, 2010. ACM Digital Library, 2010.

[Sö10] Söhnlein, S. et al.: Software Reliability Assessment based on the Evaluation of Operational Experience. In (Müller-Clostermann, B.; Echtle, K.; Rathgeb, E. P. Eds.): Proc. 15th International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability in Fault Tolerance, MMB&DFT 2010, Essen, Germany, 2010. Vol. LNCS 5987, Springer-Verlag, 2010; pp. 24-38.

[WK09] Westergaard, M.; Kristensen, L. M.: The Access/CPN Framework: A Tool for Interacting with the CPN Tools Simulator. In (Franceschinis, G.; Wolf, K. Eds.): Proc. 30th Int. Conf. on Applications and Theory of Petri Nets, PETRI NETS 2009, Paris, France, 2009. Vol. LNCS 5606, Springer-Verlag, 2009; pp. 313-322.

[ZH00] Zhu, H.; He, X.: A Theory of Testing High Level Petri Nets. In (Feng, Y.; Notkin, D.; Gaudel, M.-C. Eds.): Proc. 16th Int. Conf. on Software - Theory and Practice, IFIP World Computer Congress 2000, Beijing, China, 2000. Publishing House of Electronics Industry, 2000; pp. 443-450.

[ZH02] Zhu, H.; He, X.: A Methodology of Testing High-Level Petri Nets. In: Information and Software Technology, Vol. 44, No. 8. Elsevier, 2002; pp. 473-489.