

Combining Petri Nets and UML for Model-based Software Engineering

João M. Fernandes

Dep. Informática / Centro Algoritmi
Universidade do Minho
4710-057 Braga
Portugal

Abstract

UML is by far the most widely used modelling language used nowadays in software engineering, due to its large scope and its wide tool support. This software standard offers many diagrams that cover all typical perspectives for describing and modelling the software systems under consideration. Among those diagrams, UML includes diagrams (activity diagram, state machine diagram, use case diagrams, and the interaction diagrams) for describing the behaviour (or functionality) of a software system. Petri nets constitute a well-proven formal modelling language, suitable for describing the behaviour of systems with characteristics like concurrency, distribution, resource sharing, and synchronisation. Thus, one may question why not combining some UML diagrams with Petri nets for effectively supporting the activities of the software engineer. The usage of Petri nets for/in Software Engineering was addressed by several well-known researchers, like, for example, Reisig [6], Pezzè [1], Machado [5], and Kindler [4].

In this invited paper, we discuss some alternatives to introduce Petri nets into a UML-based software development process. In particular, we describe how Coloured Petri Net (CPN) models can be used to describe the set of scenarios associated with a given use case. We describe three different alternatives that can be adopted to achieve that purpose.

The first approach, initially presented in [7], suggests a set of rules that allow software engineers to transform the behaviour described by a UML 2.0 sequence diagram into a CPN model. Sequence diagrams in UML 2.0 are much richer than those in UML 1.x, namely by allowing several traces to be combined in a unique diagram, using high-level operators over interactions. The main purpose of the transformation is to allow the development team to construct animations based on the CPN model that can be shown to the users or the clients in order to reproduce the expected scenarios and thus validate them. Thus, non-technical stakeholders are able to discuss and validate the captured requirements. The usage of animation is an important topic in this context, since it permits the user to discuss the system behaviour using the problem domain language.

In the second approach, discussed in [3], we assume that developers specify the functionality of the system under consideration with use cases, each of which is described by a set of UML 2.0 sequence diagrams. For each use case, there should exist at least one sequence diagram that represents and describes its main scenario. Other sequence diagrams for the same use case are considered to be variations of the main sce-

nario. The transformation approach allows the development team to interactively play or reproduce any possible run of the given scenarios. In particular, the natural characteristics of the CPN modelling language facilitate the representation of the hierarchy and concurrency constructs of sequence diagrams.

The third alternative, considered in [2], is an improvement with respect to the previous approach and is targeted to reactive systems. We identify and justify two key properties that the CPN model must have, namely: (1) controller-and-environment-partitioned, which means constituting a description of both the controller and the environment, and distinguishing between these two domains and between desired and assumed behaviour; (2) use case-based, which means constructed on the basis of a given use case diagram and reproducing the behaviour described in accompanying scenario descriptions. We have demonstrated how this CPN model is useful for requirements engineering, since it provides a solid basis for addressing behavioural issues early in the development process, for example regarding concurrent execution of use cases and handling of failures.

References

1. G. Denaro and M. Pezzè. Petri Nets and Software Engineering. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 439–466. Springer, 2004. DOI 10.1007/b98282.
2. J.M. Fernandes, J.B. Jørgensen, and S. Tjell. Requirements engineering for reactive systems: Coloured petri nets for an elevator controller. In *14th Asia-Pacific Software Engineering Conference (APSEC 2007)*, pages 294–301. IEEE CS Press, December 2007. DOI 10.1109/APSEC.2007.81.
3. J.M. Fernandes, S. Tjell, J.B. Jørgensen, and O. Ribeiro. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net. In *6th Int. Workshop on Scenarios and State Machines (SCESM 2007), within ICSE 2007*. IEEE CS Press, May 2007. DOI 10.1109/SCESM.2007.1.
4. Ekkart Kindler. Model-Based Software Engineering and Process-Aware Information Systems. *Transactions on Petri Nets and Other Models of Concurrency*, 5460:27–45, 2009. DOI 10.1007/978-3-642-00899-3_2.
5. R. J. Machado, K. B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Requirements Validation: Execution of UML Models with CPN Tools. *International Journal on Software Tools for Technology Transfer*, 9(3–4):353–369, 2007. DOI 10.1007/s10009-007-0035-0.
6. W. Reisig. Petri Nets in Software Engineering. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 63–96. Springer, 1987. DOI 10.1007/3-540-17906-2_22.
7. O. Ribeiro and J. M. Fernandes. Some Rules to Transform Sequence Diagrams into Coloured Petri Nets. In K. Jensen, editor, *7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006)*, pages 237–256, October 2006.