Towards Domain-Specific CPN Modelling Languages

Alejandro Rodríguez Tena[1], Fernando Macías[1],
Lars Michael Kristensen[1], and Adrian Rutle[1]

Department of Computing, Mathematics, and Physics
Western Norway University of Applied Sciences, Bergen
{arte,fmac,lmkr,aru}@hvl.no

Software systems engineering is a comprehensive discipline involving a multitude of activities such as requirements engineering, design and specification, implementation, testing, and deployment. Model-driven software engineering (MDSE) [4] is one of the emergent responses from the scientific and industrial communities to tackle the increasing complexity of software systems. MDSE utilises abstractions for modelling different aspects – behaviour and structure – of software systems, and treats models as first-class entities in all phases of software development. One way to increase the adoption of MDSE is to develop modelling approaches which reflect the way software architects, developers and designers, as well as organisations, domain experts and stakeholders handle abstraction and problem-solving. In this context, domain-specific (meta)modelling (DSMM) [5] has been proposed as an approach to unite software modelling and abstraction, software design and architecture, and organisational studies. The main aim is to fill the gap between these fields and make modelling more widely applicable than it currently is [15].

The development of distributed software systems is particularly challenging. A major reason is that these systems possess concurrency and non-determinism which means that the execution of such systems may proceed in many different ways. To cope with the complexity of modern concurrent systems, it is therefore crucial to provide methods that enable debugging and testing of central parts of the system design prior to implementation and deployment [7]. Since concurrency, communication and synchronisation are increasingly present in our lives, it is priority to put efforts in improving the current techniques to deal with them. One way to approach the challenge of developing concurrent systems is to build an executable model of the system. Constructing a model and simulating it usually leads to significant insights into the design and operation of the system considered and often results in a simpler and more streamlined design.

Modelling of distributed systems. Coloured Petri Nets (CPNs) form a graphical language designed to construct models of distributed systems i.e. communication protocols [6], data networks [3], distributed algorithms [11] and embedded systems [2]. CPNs combine classical Petri nets [10] with the functional programming language Standard ML [14]. The modelling language is suited for discrete-event processes that include choice, iteration, and concurrent execution. A CPN model of a system is an executable model representing the states of the system and the events (transitions) that can cause the system to change state. The CPN modelling language also makes it possible to organise a model into a hierarchically related set of modules, and it has a time concept to represent the time taken to execute events.

One advantage of CPNs is that they contain few but powerful modelling constructs. This means that the modeller has few constructs that need to be mastered in order to apply the language. However, several recent applications of CPNs [13] have shown that it would be beneficial to be able to develop domain-specific variants that would make it possible to support:

Modelling patterns representing commonly used approaches to capture concepts from the problem domain.

Modelling restrictions forcing the modeller to use only certain constructs in the language when modelling concepts from the problem domain.

Subtyping of elements allowing specific interpretation of certain model elements such as places, transitions, and arcs.

Figure 1 (left) shows an example from the embedded software domain [8] in which substitution transitions and certain places have been subtyped as representing interfaces and events for code generation purpose. Figure 1 (right) shows an example from the control system domain in which modelling patterns are used to consume events (Fig 1(right,top)) and update a process variable (Fig 1(right,bottom)) based on input received from the environment. The patterns in turn put restrictions on the arcs and arc expressions.

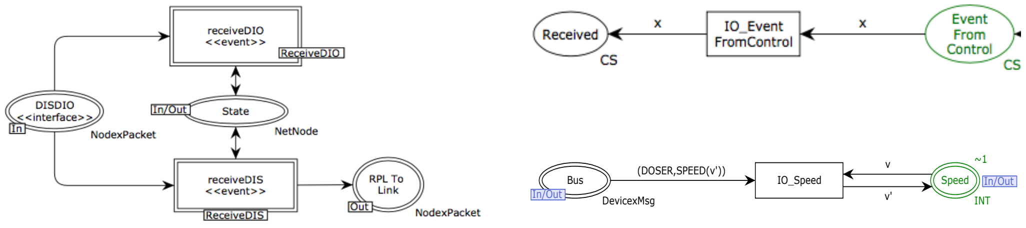

Figure 1: Examples of domain-specific concepts in CPN models.

A Metamodel for Coloured Petri Nets. The lack of extensibility of the CPN language and lack of adaptability provided by CPN Tools have motivated us to develop a model-driven infrastructure for CPN. The first step towards this has been to develop a metamodel for CPN. The definition of this metamodel will support application of model transformations (for definition of model semantics), domain-specific metamodelling (for creation of domain-specific versions of CPN), and abstraction (for definition of modelling patterns and restrictions).

There exists work on metamodels for Petri Nets [1]. These metamodels have been developed for the purpose of tool interoperability for general purpose Petri nets, and not the domain-specific aspects that we aim to address. Figure 2 shows a first attempt to develop a metamodel with the Eclipse Modeling Framework (EMF). The next step is to put this metamodel in a multilevel context using MultEcore [9] to facilitate refinement of concepts from CPN to reflect domain concepts. This metamodel captures all models that can be built using CPN [7]. In addition to the concepts represented by the class model in the figure, we have the following constrains in the metamodel:

- A module cannot be a submodule of itself, i.e., if we follow the associations through substitution transitions and modules then we cannot encounter the same module twice.
- If a port is associated with a socket place, then the socket place must be connected to a substitution transition that has the module to which the port belongs as its submodule.
- Associated port and (socket) places must have identical colour sets, and if the port place has an initial marking it must be equal to the associated socket place.

These constrains can be expressed using the Object Constraints Language (OCL). However, in order to obtain a more uniform metamodel, we are currently investigating how they can be expressed directly using formal diagrammatic notations in DPF [12].

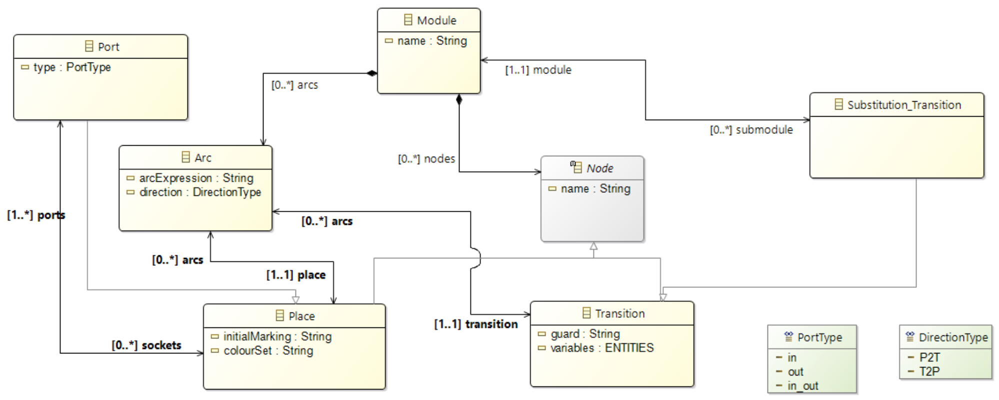

Figure 2: Metamodel for Colored Petri Nets

References

[1] Petri nets markup language. http://www.pnml.org/papers.php.

[2] M. A. Adamski, A. Karatkevich, and M. Wegrzyn. *Design of embedded control systems*, volume 267. Springer, 2005.

[3] J. Billington and M. Diaz. *Application of Petri nets to communication networks: Advances in Petri nets*. Number 1605. Springer Science & Business Media, 1999.

[4] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[5] J. de Lara, E. Guerra, and J. Sánchez Cuadrado. Model-driven engineering with domain-specific meta-modelling languages. *Software & Systems Modeling*, 14(1):429–459, 2015.

[6] J. Desel, W. Reisig, and G. Rozenberg, editors. *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3018 of *Lecture Notes in Computer Science*. Springer, 2004.

[7] K. Jensen, L. M. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3):213–254, Jun 2007.

[8] L. Kristensen and V. Veiset. Transforming cpn models into code for tinyos: A case study of the rpl protocol. In *Proc. of ICATPN'16*, volume 9698 of *LNCS*, pages 135–154. Springer, 2016.

[9] F. Macías, A. Rutle, and V. Stolz. MultEcore: Combining the best of fixed-level and multilevel metamodelling. In *MULTI*, volume 1722 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[10] W. Reisig. *Petri nets: an introduction*, volume 4. Springer Science & Business Media, 2012.

[11] W. Reisig. *Elements of distributed algorithms: modeling and analysis with Petri nets*. Springer Science & Business Media, 2013.

[12] A. Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, Department of Informatics, University of Bergen, Norway, 2010.

[13] K. I. F. Simonsen, L. M. Kristensen, and E. Kindler. Pragmatics annotated coloured petri nets for protocol software generation and verification. *TopNoC*, 11:1–27, 2016.

[14] J. D. Ullman. *Elements of ML programming*. Prentice-Hall, Inc., 1994.

[15] J. Whittle, J. E. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, 2014.