Transforming Platform Independent CPN Models into Code for the TinyOS Platform: A Case Study of the RPL Protocol

Vegard Veiset and Lars Michael Kristensen

Department of Computing, Bergen University College
Email: `vegard.veiset@stud.hib.no,lmkr@hib.no`

Abstract. TinyOS is a widely used platform for the development of networked embedded systems offering a programming model targeting resource constrained devices. We present a software engineering approach where Coloured Petri Net (CPNs) models are used as a starting point for developing protocol software for the TinyOS platform. The approach consists of five refinement steps where a platform-independent CPN model is gradually transformed into a platform-specific model that enables automatic code generation. To evaluate our approach, we use it to obtain an implementation of the IETF RPL routing protocol for sensor networks.

Introduction. Model-based software engineering and verification have several attractive properties in the development of flexible and reliable software systems. In order to fully leverage modelling investments, it is desireable to use the constructed models also for the implementation of the software on the platform under consideration. Coloured Petri Nets [2] (and Petri Nets in general) constitute a general purpose modelling language supporting platform-independent modelling of concurrent systems. Hence, in most cases, such models are too abstract to be used directly to implement software. In order to bridge the gap between abstract and platform independent CPN models and the implementation of software to be deployed, the concept of *pragmatics* was introduced in [4]. Pragmatics are syntactical annotations that can be added to a CPN model and used to direct code generation for a specific platform. The contribution of this paper is an approach [5] that exploits pragmatics in combination with a five step refinement methodology to enable code generation for the TinyOS platform. Applications for TinyOS [3] are implemented using the nesC programming language (a dialect of C) providing an event-based split-phase programming model. An application written in nesC is organised into a wired set of modules each providing an interface consisting of commands and events.

Refinement Steps. The model refinement starts from a platform independent CPN model constructed typically with the aim of specifying the protocol operation and performing model checking of the protocol design. Each step consists of a transformation that uses the constructs of the CPN modelling language to add details to the model. Furthermore, in each step pragmatics are added that direct the code generation performed after the fifth step:

Step 1: Component Architecture consists of annotating CPN submodules and substitution transitions corresponding to TinyOS components, and make explicit the interfaces used and provided by components.

Step 2: Resolving Interface Conflicts resolves interface conflicts allowing components to use multiple instances of an interface. This is done by annotating CPN arcs with information providing locally unique names.

Step 3: Component and Interface Signature adds type signatures to components and interfaces by creating explicit submodules for command and events, and by refining colour sets to reflect the interface signatures.

Step 4: Component Classification further refines the components by classifying them into four main types: timed, external, boot, and generic.

Step 5: Internal Component Behaviour consists of refining the modelling of the individual commands and events such that control flow and data manipulation become explicit and organised into atomic statement blocks.

After the fifth refinement step has been performed, the CPN model includes sufficient detail to be used as a basis for automated code generation.

The RPL Protocol and Code Generation. To evaluate our approach on an industrial-sized example, we have conducted a case study based on the RPL routing protocol [1] developed by the Internet Engineering Task Force. The RPL protocol allows a set of sensor nodes to construct a destination-oriented directed acyclic graph which can be used for multi-hop communication between sensor nodes. To support the automatic code generation for TinyOS, we have developed a software prototype in Java that performs a template-based model-to-text transformation on the models resulting from the fifth refinement step. The software prototype relies on the Access/CPN framework [6] to load CPN models created with CPN Tools. The code generator performs a top-down traversal of the CPN model where code templates are selected according to the pragmatic annotations on the CPN model elements encountered.

References

1. T. Winter et. al. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, 2012. Internet Engineering Task Force.
2. 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(3-4):213–254, 2007.
3. P. Levis. *TinyOS Programming.* Cambridge University Press, 2009.
4. K. Simonsen, L.M. Kristensen, and E. Kindler. Code Generation for Protocols from CPN Models Annotated with Pragmatics. In *Proc. of NWPT'12*, volume 403 of *Report in Informatics*, pages 46–48. University of Bergen, 2012.
5. V. Veiset. An Approach to Semi-Automatic Code Generation for the TinyOS Platform using Coloured Petri Nets. Master's thesis, Bergen University College, 2013.
6. M. Westergaard. Access/CPN 2.0: A High-Level Interface to CPN Models. In *Proc. of ICATPN'11*, volume 6709 of *LNCS*, pages 328–337. Springer, 2011.