An Approach for the Engineering of Protocol Software from Coloured Petri Net Models: A Case Study of the IETF WebSocket Protocol

Lars Michael Kristensen

Department of Computing, Bergen University College, Norway
Email: lmkr@hib.no

Invited Talk

The vast majority of software systems today can be characterised as concurrent and distributed systems as their operation inherently relies on protocols executed between independently scheduled software components. The engineering of correct protocols can be a challenging task due to their complex behaviour which may result in subtle errors if not carefully designed. Ensuring interoperability between independently made implementations is also challenging due to ambiguous protocol specifications. Model-based software engineering offers several attractive benefits for the implementation of protocols, including automated code generation for different platforms from design-level models. Furthermore, the use of formal modelling in combination with model checking provides techniques to support the development of reliable protocol implementations.

Coloured Petri Nets (CPNs) [3] is formal language combining Petri Nets with a programming language to obtain a modelling language that scales to large systems. In CPNs, Petri Nets provide the primitives for modelling concurrency and synchronisation while the Standard ML programming language provides the primitives for modelling data and data manipulation. CPNs have been successfully applied for the modelling and verification of many protocols, including Internet protocols such as the TCP, DCCP, and DYMO protocols [1, 4]. Formal modelling and verification have been useful in gaining insight into the operation of the protocols considered and have resulted in improved protocol specifications. However, earlier work has not fully leveraged the investment in modelling by also taking the step to automated code generation as a way to obtain an implementation of the protocol under consideration.

In earlier work [5], we have proposed the PetriCode approach and a supporting software tool [7] has been developed for automatically generating protocol implementations based on CPN models. The basic idea of the approach is to enforce particular modelling patterns and annotate the CPN models with code generation *pragmatics*. The pragmatics are bound to code generation templates and used to direct a template-based model-to-text transformation that generates the protocol implementation. As part of earlier work, we have demonstrated the use of the PetriCode approach on small protocols. In addition, it has been shown that our approach supports code generation for multiple platforms, and that it leads to code that is readable and also compatible with other software [6].

In the present work we consider the application of our code generation approach as implemented in the PetriCode tool to obtain protocol software implementing the IETF WebSocket protocol [2] protocol for the Groovy language and platform. This demonstrates that our approach and tool scales to industrial-sized protocols. The WebSocket protocol is a relatively new protocol and makes it possible to upgrade an HTTP connection to an efficient message-based full-duplex connection. WebSocket has already become a popular protocol for several web-based applications such as games and media streaming services where bi-directional communication with low latency is needed.

The contributions of our work include showing how we have been able to model the WebSocket protocol following the PetriCode modelling conventions. Furthermore, we perform formal verification of the CPN model prior to code generation, and test the implementation for interoperability against the Autobahn WebSocket test-suite [8] resulting in 97% and 99% success rate for the client and server implementation, respectively. The tests show that the cause of test failures were mostly due to local and trivial errors in newly written code-generation templates, and not related to the overall logical operation of the protocol as specified by the CPN model. Finally, we demonstrate in this paper that the generated code is interoperable with other WebSocket implementations.

Acknowledgement. The results presented in this invited talk is based on joint work with Kent I.F. Simonsen, Bergen University College and the Technical University of Denmark, and Ekkart Kindler, the Technical University of Denmark.

References

1. J. Billington, G.E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 210–290. Springer, 2004.
2. I. Fette and A. Melnikov. The websocket protocol, 2011. `http://tools.ietf.org/html/rfc6455`.
3. 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.
4. L.M. Kristensen and K.I.F. Simonsen. Applications of Coloured Petri Nets for Functional Validation of Protocol Designs. In *Transactions on Petri Nets and Other Models of Concurrency VII*, volume 7480 of *LNCS*, pages 56–115. Springer, 2013.
5. K. I. F. Simonsen, L. M. Kristensen, and E. Kindler. Generating Protocol Software from CPN Models Annotated with Pragmatics. In *Formal Methods: Foundations and Applications*, volume 8195 of *LNCS*, pages 227–242. Springer, 2013.
6. K.I.F. Simonsen. An Evaluation of Automated Code Generation with the PetriCode Approach. In *To appear in Proc. of PNSE'14*, 2014.
7. K.I.F. Simonsen. PetriCode: A Tool for Template-based Code Generation from CPN Models. In *SEFM 2013 Collocated Workshops: BEAT2, WS-FMDS, FM-RAIL-Bok, MoKMaSD, and OpenCert*, volume 8368 of *LNCS*, pages 151–166. Springer, 2014.
8. Tavendo GmbH. *Autobahn|Testsuite*. `http://autobahn.ws/testsuite/`.