chi+med
making medical devices safer

EPSRC Programme Grant EP/G059063/1

Public Paper no. 204

Combining PVSio with Stateflow

Paolo Masci, Yi Zhang, Paul Jones, Patrick Oladimeji, Enrico D'Urso, Cinzia Bernardeschi,
Paul Curzon & Harold Thimbleby

Masci, P., Zhang, Y., Jones, P., Oladimeji, P., D'Urso, E., Bernardeschi, C., Curzon, P., & Thimbleby, H. (2014). Combining PVSio with Stateflow. *Proceedings of the 6th NASA Formal Methods Symposium (NFM-2014)*, 209–214. Lecture Notes in Computer Science, vol. 8430. Springer.

PP release date: 11 February 2014

file: WP204.pdf

UCL UCLIC Swansea University Prifysgol Abertawe FITLAB Queen Mary University of London CITY UNIVERSITY LONDON EPSRC Engineering and Physical Sciences Research Council

Combining PVSio with Stateflow

Paolo Masci[1⋆], Yi Zhang[2], Paul Jones[2], Patrick Oladimeji[3], Enrico D'Urso[4], Cinzia Bernardeschi[4], Paul Curzon[1], and Harold Thimbleby[3]

[1] School of Electronic Engineering and Computer Science
Queen Mary University of London, United Kingdom
{p.m.masci,p.curzon}@qmul.ac.uk

[2] Center for Devices and Radiological Health
U.S. Food and Drug Administration, Silver Spring, Maryland, USA
{yi.zhang2,paul.jones}@fda.hhs.gov

[3] Future Interaction Technology Lab (FITLab)
Swansea University, United Kingdom
{p.oladimeji,h.thimbleby}@swansea.ac.uk

[4] Dipartimento di Ingegneria dell'Informazione
Universitá di Pisa, Italy
e.durso@studenti.unipi.it,c.bernardeschi@unipi.it

Abstract. An approach to integrating PVS executable specifications and Stateflow models is presented that uses web services to enable a seamless exchange of simulation events and data between PVS and Stateflow. Thus, it allows the wide range of applications developed in Stateflow to benefit from the rigor of PVS verification. The effectiveness of the approach is demonstrated on a medical device prototype, which consists of a user interface developed in PVS and a software controller implemented in Stateflow. Simulation on the prototype shows that simulation data produced is exchanged smoothly between in PVSio and Stateflow.

Keywords: Simulation, PVSio, Stateflow.

1 Introduction

Model based engineering is being increasingly adopted to develop complex control systems that demand high assurance of safety and quality. Designing a complex system often requires a combination of modeling and verification tools, such as PVS and Simulink. Reasons include: (i) different modeling tools have their own strengths and limitations, making them suitable for different tasks; (ii) one modeling tool might have been used to develop legacy models that are reused in a new project that depends on another tool; (iii) different development teams may prefer different tools, based on their expertise.

PVS [9] and MathWorks Simulink [2] are two modeling frameworks widely used in both industry and academia, each of which has a native simulation

⋆ Corresponding author.

environment for model animation. PVSio [7] is the simulation environment of PVS. Simulink enables the simulation of system models with mixed discrete and continuous control logic; its Stateflow component [3] models the discrete control of these systems.

The integration of PVS and Simulink environments can benefit system designers, allowing them to model part of the system in PVS and the rest in Simulink. However, in reality, PVSio and Simulink (and Stateflow in particular) are not interoperable. That is, PVS specifications and Stateflow models that correspond to different parts of a system cannot be simulated together. As a result, designers have to sacrifice freedom and flexibility, and model the discrete control of the entire system in either PVS higher-order logic or Stateflow.

Contributions. We present a new, flexible approach for integrating PVSio with Stateflow. Specifically, our approach establishes web services to create a communication infrastructure between these two frameworks. An illustrative example is presented that applies the approach to a non-trivial medical device prototype, with a user interface specified in PVS and a software controller developed in Stateflow. Simulation of the prototype demonstrates that the PVSio and Stateflow components can interoperate effectively. The tools and example models are available at http://www.pvsioweb.org .

Related work. Research on integration of Stateflow with other modeling tools is generally based on the idea of performing a translation between Stateflow models and another formal specification. For example, in [5], a formal semantics of Stateflow is developed to enable the translation of Stateflow models into SAL (Symbolic Analysis Laboratory) specifications. Similarly, in [12], a tool is presented that translates Stateflow models into Lustre specifications. In [11], Stateflow models are generated from formal specifications based on Event-B semantics. A good overview of similar approaches can be found in [4, 10]. Such approaches have the advantage of allowing formal verification of whole systems.

2 The approach for integrating PVSio with Stateflow

The most significant challenge in integrating PVS with Stateflow is the lack of a publicly available formal semantics for Stateflow. As argued in [5], a formal operational semantics can be defined only for a subset of Stateflow. It is therefore not possible to faithfully translate Stateflow models that use constructs outside of the formalized subset. Similarly, Stateflow models translated from other models can use only the formalized subset of its semantics. In contrast, our approach alleviates this issue by enabling communication between PVSio and Stateflow models, rather than performing model translation. This offers designers more freedom and reliability, since no restricted translation is involved.

Our approach establishes two web services, PVSio-web [8] and Stateflow-web, to create a communication protocol between PVSio and Stateflow (see figure 1). Each model runs in parallel, sending data and events (when they occur) the other needs to continue the simulation. The protocol is “tool-neutral” in the sense that it enables seamless exchange of events and data between PVSio and Stateflow

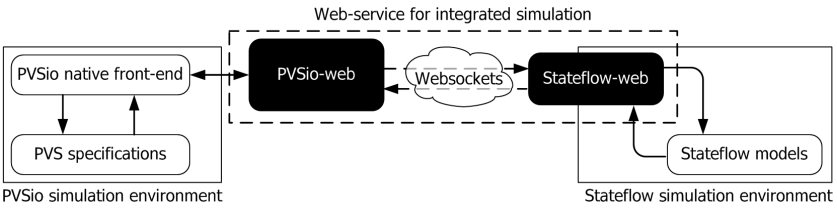


Fig. 1. The developed approach for integrated simulation.

during simulation, without changing either of these environments. Thus, it preserves the underlying semantics of PVSio and Stateflow environments.

PVSio-web, our web-server for PVSio, comprises a *tool-specific* communication interface to connect to the PVS environment, and a *tool-neutral* communication interface to exchange simulation events with Stateflow-web. The former is tailored to the PVSio environment, while the latter utilizes the Websocket standard (a low-latency communication protocol) and encodes simulation events in the widely-used open-standard format JSON (JavaScript Object Notation). Handlers defined within PVSio-web programmatically intercept and inject simulation events, thus enabling interaction with the Stateflow environment.

Handlers in PVSio-web are implemented as JavaScript functions, which interact with PVSio by submitting PVS higher-order logic expressions to the PVSio command prompt and then reading PVSio responses. The handlers also convert PVS expressions into simulation events that can be exchanged with and understood by Stateflow-web. To ease the conversion, PVS expressions are specified as transition functions over a PVS record type `state`. Each field of `state` specifies data or commands that need to be exchanged with the Stateflow model. The original PVS theory is kept unchanged.

Stateflow-web has a similar design to PVSio-web. Its handlers are specified as either Statechart diagrams (i.e., state machines) or C++ classes. Statechart diagram handlers are used to trigger transitions in the Stateflow model based on the commands received from PVSio-web, and to update simulation data in the Stateflow model accordingly. These handlers also intercept simulation events and data produced by Stateflow and translate them into the format that PVSio-web understands. C++ handlers are responsible for exchanging simulation events with PVSio-web based on a Websocket communication library.

3 Example: A Patient Controlled Analgesia (PCA) device

The effectiveness of the approach is illustrated using a medical device prototype: the Generic Patient Controlled Analgesia (PCA) pump [1]. PCA infusion pumps are widely used for delivering pain-relief drugs to patients. PCA pumps offer a patient-controlled feature ("bolus") to briefly boost drug delivery on demand. Bolus features are controlled, so a patient cannot voluntarily give themselves too high a dose.

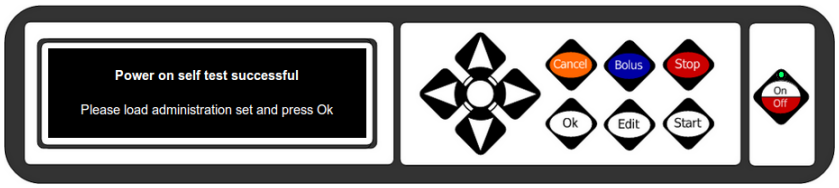


Fig. 2. The visual appearance of the GPCA user interface.

The aim of the Generic PCA (GPCA) pump is to capture functionalities shared by existing commercial PCA pumps and provide a common basis for healthcare stakeholders to discuss and assess their safety.

3.1 The Generic Patient Controlled Analgesia (GPCA) model

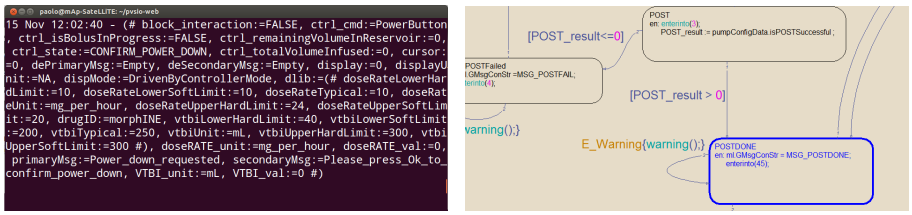
The two primary software components of the GPCA pump are the *user interface* and *software controller*. While the user interface manages the interaction with the users (nurse or patient), the software controller regulates the drug infusion process and handles alarms and warnings. These two components exchange information (events and data) during model execution to simulate typical infusion scenarios. The information exchanged can be divided into four categories: *infusion parameters*, including the infusion volume and rate programmed by the user through the user interface; *user actions*, which are commands (such as start or stop infusion) that the user issues through the user interface; *current state*, the current operational status of the software controller; and *infusion status*, the status of currently active infusion, including bolus dosage, infusion rate, and the volumes of drug delivered and to be infused.

A model of the GPCA was previously developed in Stateflow, in which a naïve user interface was implemented for demonstration purposes. For this paper, we replaced this naïve user interface with a more sophisticated one [6], which was implemented as a PVS executable specification. This sophisticated user interface has been verified in PVS for basic safety properties (see [6] for details).

The objective of this study, then, is to use the presented approach to connect the PVS-based user interface with the Stateflow-based software controller, and perform a simulation over the entire GPCA pump model.

3.2 Simulation of GPCA model

We were able to run simulations over the integrated GPCA model using our approach. During the simulation, users interact with the PVS-based user interface by pressing buttons and reading display elements of the graphical front-end shown in figure 2. Each user interaction is captured by PVSio-web handlers, which in turn send PVS expressions to PVSio for model animation. PVSio-web links with Stateflow-web to exchange simulation events generated by the software controller simulated in parallel within Stateflow. For example, figures 3(a)



(a) PVSio simulation of the user interface. (b) Stateflow simulation of the controller.

Fig. 3. Close-up view of the simulator's output during an execution of the GPCA.

and 3(b) respectively demonstrate the simulation state in PVSio and Stateflow for the scenario where a pump successfully passes the power-on self test.

To allow the GPCA to be simulated, dedicated handlers were implemented in PVSio and in Stateflow to enable communication of events and data. On the PVSio-web side, three Javascript functions were defined:

- **Create a connection:** *gipConnect* establishes a Websocket connection with Stateflow-web on a given port. It calls functions provided by Node.js[5].
- **Messages from Stateflow-web:** *gipReceive* is invoked every time a message is received over the Websocket connection. It receives tool-neutral simulation events and data from Stateflow-web. These events and data specify the current state of the software controller and the infusion status. They are converted into PVS expressions that can be evaluated in PVSio.
- **Messages to Stateflow-web:** *gipSend* parses predefined fields of the state returned by PVSio after it has evaluated a PVS expression. The values of these fields are used to generate tool-neutral messages containing simulation events and data to be sent to the software controller.

On the Stateflow-web side, two Stateflow blocks were defined:

- **Communicating with PVSio-web:** *Websocket communication bridge* is a System Function block implemented in C++. A standard communication library is used to send and receive messages over Websocket connections. Two input buses are used to intercept the state variables of the software controller and thus generate tool-neutral simulation events and data for the user interface. Three output buses are used to inject simulation events and data received from the user interface software.
- **Driving Stateflow model:** *UI Commands dispatcher* is a Statechart block that forwards simulation events and data to appropriate blocks in the Stateflow model. This Statechart has one input line that receives commands originated from the user interface; 21 output lines for redirecting received commands to the appropriate components in the GPCA Stateflow model. The number of output lines would of course vary for different Stateflow models.

[5] Node.js, a popular scalable network framework, is the Javascript runtime environment used to implement PVSio-web.

4 Conclusions

The approach presented in this paper for integrating PVS and Simulink uses standard web services to connect PVSio (the simulator of the theorem proving system PVS) and Stateflow (the discrete modeling component of Simulink). The approach thus provides a seamless and effective way to integrate these two mainstream modeling and verification tools. In this way, the hazards of translating design models composed in different tools are avoided, and fast and realistic prototyping becomes possible for designs modeled with multiple tools.

In the case study, a model written in Stateflow was connected to a formally verified user interface implemented in PVS. The success of this case study suggests an alternative way to verify Stateflow models: for example, the correctness of Stateflow models can be evaluated through PVS using methods like black-box testing (guided by PVSio) and assume-guarantee reasoning (supported by PVS).

Acknowledgments. This work is part of CHI+MED (EPSRC grant EP/G059063/1).

References

1. GPCA project. http://rtg.cis.upenn.edu/medical/gpca/gpca.html.
2. Mathworks Simulink. http://www.mathworks.com/products/simulink.
3. Mathworks Stateflow. http://www.mathworks.com/products/stateflow.
4. C. Chen, J. S. Dong, and J. Sun. A formal framework for modeling and validating Simulink diagrams. *Formal Aspects of Computing*, 21(5):451–483, 2009.
5. G. Hamon and J. Rushby. An operational semantics for Stateflow. In *Fundamental Approaches to Software Engineering (FASE)*, volume 2984 of *Lecture Notes in Computer Science*, pages 229–243. Springer Berlin Heidelberg, 2004.
6. P. Masci, A. Ayoub, P. Curzon, I. Lee, O. Sokolsky, and H. Thimbleby. Model-Based Development of the Generic PCA Infusion Pump User Interface Prototype in PVS. In *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 228–240. Springer Berlin Heidelberg, 2013.
7. C. Muñoz. Rapid prototyping in PVS. Technical Report NIA Report No. 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, 2003.
8. P. Oladimeji, P. Masci, P. Curzon, and H. Thimbleby. PVSio-web: A tool for rapid prototyping device user interfaces in PVS. In *5th International Workshop on Formal Methods for Interactive Systems (FMIS2013)*, 2013. Tool and application examples available at http://www.pvsioweb.org.
9. S. Owre, J.M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, 1992.
10. P. Roy and N. Shankar. SimCheck: a contract type system for Simulink. *Innovations in Systems and Software Engineering*, 7(2):73–83, 2011.
11. M. Satpathy, S. Ramesh, Colin Snook, N.K. Singh, and M. Butler. A mixed approach to rigorous development of control designs. In *IEEE Multi-Conference on Systems and Control (MSC 2013)*, August 2013.
12. N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, and F. Maraninchi. Defining and translating a safe subset of Simulink/Stateflow into Lustre. In *4th ACM International Conference on Embedded Software*. ACM, 2004.