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 is presented to integrate PVS executable specifications and Stateflow models. It uses web services to enable a seamless exchange of simulation events and data between PVSio and Stateflow. The approach's effectiveness is demonstrated on a medical device prototype. The prototype's user interface is a PVS specification with its software controller implemented in Stateflow. Using the web services approach, a simulation is run over the prototype, during which simulation data produced in PVSio and Stateflow are exchanged properly and smoothly. Such integration allows the wide range of applications developed in Stateflow to be complemented with the rigor of PVS verification.

Keywords: Simulation, PVSio, Stateflow.

1 Introduction

Model based engineering has been increasingly adopted to develop complex control systems that demand high assurance of safety and quality. PVS [9] and MathWorks Simulink [2] are modeling frameworks widely used in both industry and academia. Each has a native simulation 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.

Designing a complex system often requires a combination of modeling and verification tools, such as PVS and Simulink. Reasons include: 1) different modeling tools have their own strengths and limitations, making them suitable for

⋆ Corresponding author.

different tasks; 2) one modeling tool might have been used to develop legacy models that are reused in a new project that depends on another tool; and 3) different development teams may prefer different tools, based on their expertise.

The integration of PVS and Simulink environments can therefore benefit system designers, allowing them to model part of the system in PVS and the rest in Simulink. However, in reality, PVSio and Simulink (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.

Contribution. The present work illustrates a new approach for integrating PVSio with Stateflow. Specifically it establishes web services to create a communication infrastructure between the two frameworks. An illustrative example is presented that applies the approach to a medical device prototype. It's user interface is specified in PVS and its software controller in Stateflow. A simulation of the prototype demonstrates that its PVSio and Stateflow parts can interoperate.

Related work. Research on integration of Stateflow and other modeling measures is generally based on the idea of performing a translation between Stateflow models and other formal specifications. For example, in [5], a formal semantics of Stateflow was developed, based on which Stateflow models were converted into SAL (Symbolic Analysis Laboratory) specifications. Similarly, in [12], a tool is presented to translate Stateflow models into Lustre specifications. In [11], Stateflow models are generated from formal specifications based on Event-B semantics. A fairly comprehensive overview of other similar approaches can be found in [4, 10]. These approaches have the advantage of allowing formal verification of the whole system. A major difficulty though is the lack of (publicly available) formal semantics for Stateflow. As pointed out in [5], a formal operational semantics can be defined only for a subset of the full Stateflow semantics. It is 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 a subset of its semantics. Our approach alleviates this model-integration issue, at least where simulation of the whole model is the major goal, whilst opening up new verification possibilities. It does this by creating a communication infrastructure between PVSio and Stateflow models, rather than performing model translation.

2 The approach for integrating PVSio with Stateflow

Our approach establishes two web services, PVSio-web [8] and Stateflow-web, to create a communication protocol between PVSio and Stateflow (see Figure 1)[5]. Each model runs in parallel sending data the other needs to continue the simulation via the connection as and when relevent events occur in each. The protocol

[5] The tools and example are available from http://www.XXXXXXXXXXXXXXX

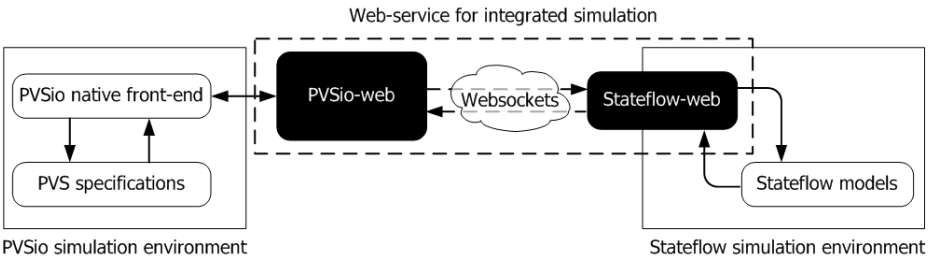


Fig. 1. The developed approach for integrated simulation.

is 'tool-neutral' in the sense that it enables seamless exchange of events and data between PVSio and Stateflow during model 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 the Stateflow environment. The former is tailored to the PVSio environment. 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 the interaction with the Stateflow environment.

More specifically, 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. These handlers also convert PVS expressions into tool-neutral 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. C++ handlers are responsible for exchanging tool-neutral simulation events with PVSio-web based on a Websockets communication library. 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 for 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.

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

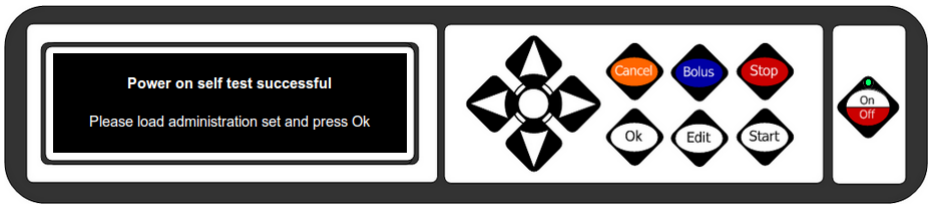


Fig. 2. The visual appearance of the GPCA user interface.

are widely used to automatically deliver drugs to patients for pain relief, while offering a patient-controlled feature ('bolus') to boost drug delivery on demand. As a conceptual framework, the aim of the generic PCA (GPCA) pump is to capture functionalities shared by existing PCA pumps and provide a common basis for healthcare stakeholders to discuss and assess their safety.

3.1 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*, i.e. 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; *infusion status*, the status of currently active infusion, including bolus dosage, infusion rate, and the volumes of drug delivered and to be infused.

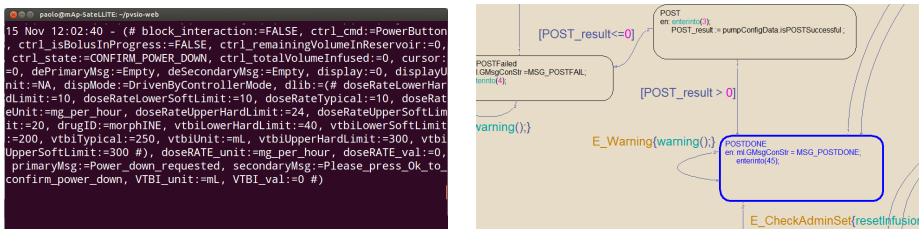
A model of the GPCA pump has been previously developed in Simulink/Stateflow, in which a naive user interface was implemented for demonstration purposes. For this study, we replace this naive interface with a more sophisticated one, as presented in [6], which is implemented as PVS executable specifications. This sophisticated user interface has been verified in PVS for basic safety properties (see [6] for details).

The objective of this study 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 this 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 3(a). Each user interaction is captured by PVSio-web handlers, which in turn send PVS expressions to PVSio-web for model animation. PVSio-web also links with Stateflow-web to exchange simulation events generated by the software controller simulated in parallel within Stateflow. For example, Figures 3(a) and 3(b) respectively demonstrate the current simulation state in PVSio and Stateflow for the scenario where the pump successfully passes power-on test and the user presses the power-off button.



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

Fig. 3. Close-up view of the simulators output during an execution of the GPCA.

To allow the GPCA to be simulated, dedicated handlers were implemented in PVSio and Stateflow, to allow communication between them. On the PVSio-web side, three Javascript functions were defined for the user interface:

- **Create a connection:** *gipConnect* establishes a Websocket connection with Stateflow-web on a given port. It calls functions provided by Node.js [**?**], the Javascript runtime environment used to implement PVSio-web.
- **Messages from Stateflow:** *gipReceive* is invoked every time a message is received over the Websocket connection. It receives tool-neutral simulation events and data, and converts them into PVS expressions that can be evaluated in PVSio. The generated PVS expressions are of a PVS record type containing two fields: the current state of the software controller, and the infusion status.
- **Messages to Stateflow:** *gipSend* parses predefined fields of the state returned by PVSio after it has evaluated a PVS expression. The value of these fields is used to generate tool-neutral simulation events and data to be sent to the software controller.

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

- **Communicating with PVSio:** *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 to the Stateflow model.

– **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 vary for different Stateflow models.

4 Conclusions

The approach presented here uses web services to connect PVSio (the simulator of the theorem proving system PVS) and Stateflow (the discrete modelling component of Simulink), and thus provides a seamless way to integrate these two main-stream modeling/verification tools. In this way, the troubles and errors of translating design models written in different tools are avoided, and fast design prototyping becomes possible for designs modeled with multiple tools.

In the case study, a design model written in Stateflow is connected to a formally verified user interface implemented in PVS. This suggests an alternative way to verify Stateflow models. That is, the correctness of Stateflow models can be evaluated through the PVS user interfaces 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*, 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 *Safecomp2013*, 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 proceedings of *FMIS2013*, 2013.
9. S. Owre, J.M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *CADE '92*, volume 607 of *Lecture Notes in Artificial Intelligence*, 1992.
10. Pritam Roy and Natarajan Shankar. Simcheck: An expressive type system for simulink. In *NASA Formal Methods*, pages 149–160, 2010.

11. M Satpathy, S Ramesh, Colin Snook, NK Singh, and Michael Butler. A mixed approach to rigorous development of control designs. 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 *Proceedings of the 4th ACM international conference on Embedded software*. ACM, 2004.