PVSio-web 2.0

Quick Reference Manual

Contents

This quick reference manual presents the main components of PVSio-web [1, 2], and step-by-step examples illustrating basic use scenarios.

1 Simulator View

The **Simulator View** handles the execution of prototypes developed within PVSio-web, and logs user interactions with the prototype. This module renders the visual elements of the prototype, and implements functions for detecting and logging user actions over input widgets. It also translates user actions performed on the input widgets into PVS expressions; triggers the evaluation of PVS expressions in PVSio; and renders PVS expressions returned by PVSio into visual elements of the prototype. Translation of user actions into PVS expressions, and rendering of PVS expressions into visual elements are performed in real time using template scripts created with Prototype Builder.

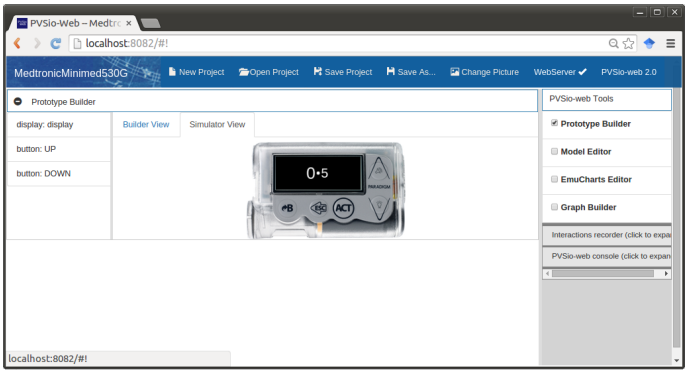

Figure 1: Simulator View while executing an example prototype.

To execute a device prototype (MedtronicMinimed530)

1. **Click 'Open Project'** in the tool bar.
2. **Select the MedtronicMinimed530 prototype** by clicking on the picture of the device.
3. **Activate the simulator** by clicking on the 'Simulator View' tab.
4. **Interact with the prototype** by clicking the up and down keys of the device user interface.

2 Prototype Builder

The **Prototype Builder** automates the generation of a prototype, providing a graphical environment with functions for defining the visual aspect of the prototype (typically, a picture) and for creating programmable overlay areas that enable interaction with the prototype. Overlay areas corresponding to input widgets define which user actions are recognised (e.g., press, release, click) and how these actions are translated into PVS expressions. The translation is performed using templates that map user actions to PVS functions on the basis of naming conventions. An example template is `click_<btn>(<st>)`, which translates clicks performed by the user over a button `<btn>` of the prototype into a PVS function that takes one parameter `<st>`, representing the current model state. Areas corresponding to output widgets use string filters to extract the actual value of the widget from PVS expressions returned by PVSio.

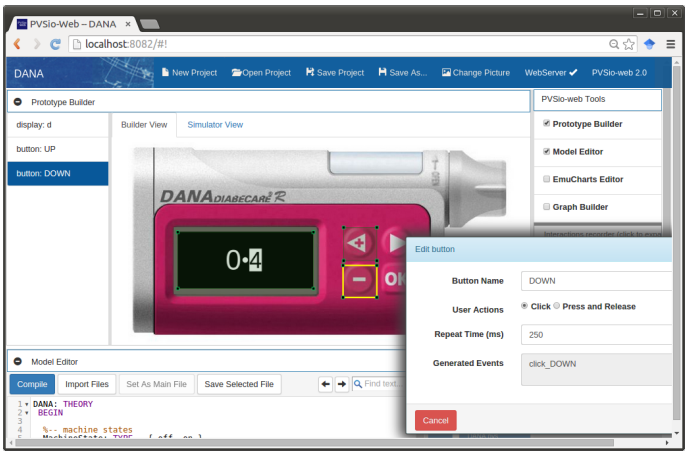

Figure 2: Prototype Builder while creating an interactive prototype based on a formal model.

To create a new device prototype (e.g., DANA)

1. **Click 'New Project'** in the tool bar.
2. **Select a User Interface Picture** of the device (example pictures are in /Downloads)
3. **Select PVS files** modelling the device behaviour (example pvs files are in /Downloads)
4. **Confirm the selections** by clicking the Ok button.

5. **Create an interactive display** by dragging over the display area of the device picture. A dialog window will open.
   (a) Enter d as value for the field Display name.
   (b) Enter c as value for the field Cursor name.
   (c) Click Ok to confirm the creation of the display.
6. **Create an interactive UP button** by dragging the mouse over button + of the device. A dialog window will open.
   (a) Select the tab Button.
   (b) Enter UP as value for field Button name.
   (c) Click Ok to confirm the creation of the button.
7. **Create an interactive DOWN button** by dragging the mouse over button - of the device. A dialog window will open.
   (a) Select the tab Button.
   (b) Enter DOWN as value for field Button name.
   (c) Click Ok to confirm the creation of the button.
8. **The prototype can now be executed**
   (a) Activate the simulator by clicking on the 'Simulator View' tab.
   (b) Interact with the prototype by clicking the + and - keys of the device user interface.

3 Emucharts Editor

The **EmuCharts Editor** implements a visual editor and code generators for creating executable formal models. With this module, developers can: *define states* of the system model by drawing labelled boxes; *define state transitions* by drawing labelled arrows; *define state variables* representing relevant characteristics of the system state; and *generate executable models* from the visual diagram. Model generators employ modelling constructs from languages supported by popular analysis tools and programming languages: state labels are translated into enumerated type constants; state variables are translated into fields of a record type defining the system state; state transitions are translated into transition functions over system states.

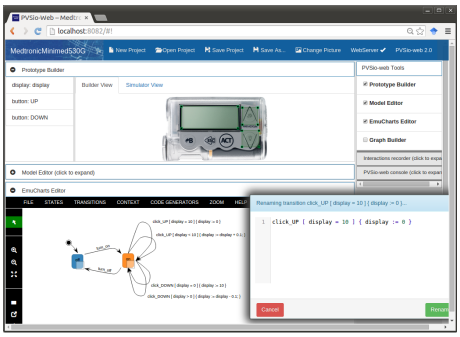

Figure 3: Emucharts Editor while creating an Emucharts diagram.

To create a new Emucharts diagram and generate formal models

1. **Load the Emucharts Editor** by selecting the Emucharts Editor tool in the PVSio-web Tools panel (depending on your screen resolution, you may need to scroll down the PVSio-web user interface to see the Emucharts Editor).

2. **Draw machine states** by clicking in the virtual canvas to create boxes using the box tool in the Emucharts palette

3. **Draw transitions** by dragging arrows between machine states using the arrow tool in the Emucharts palette

4. **Generate a formal model** (PVS, VDM-SL, etc.) based on the Emucharts diagram by selecting a model generator from the Code Generators menu of the Emucharts Editor

4 Model Editor

The **Model Editor** is a text editor for editing formal models, providing the typical functionalities of modern IDEs (syntax highlighting, autocomplete, search, etc) as well as a file browser to perform operations on the file system.

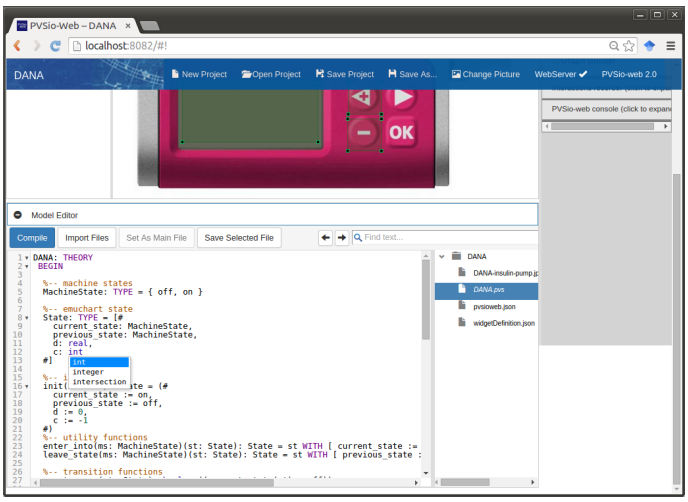

Figure 4: Model Editor while editing a PVS theory.

To edit a formal model in the Model Editor

1. **Load the Model Editor** by selecting the Model Editor tool in the PVSio-web Tools panel.
2. **Edit and compile (type-check) PVS files** using the tools provided by the Model Editor.

5 Testing the implementation of PVSio-web 2.0

JSLint and Jasmine are routinely used to ensure that our implementation is compliant with established coding standards and that the code is well-formed. To run example tests with Jasmine, type `localhost:8082/tests` in the web-browser.

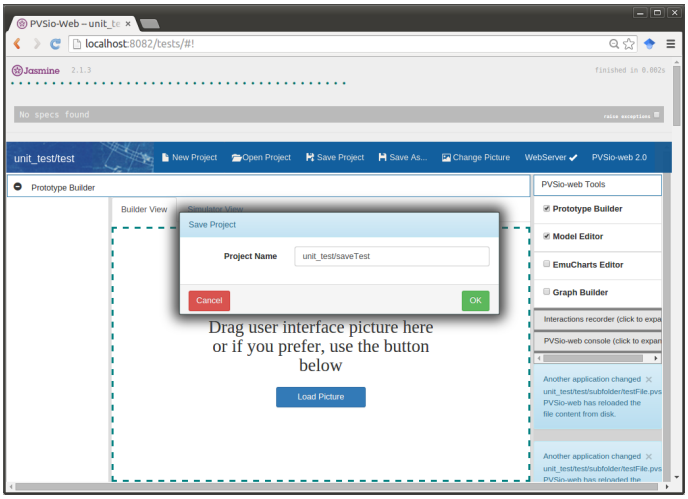

Figure 5: Testing the implementation of PVSio-web with Jasmine.

Acknowledgements. This work is part of CHI+MED (EPSRC grant [EP/G059063/1]). The authors would like to thank SRI International, in particular John Rushby, Sam Owre and Natarajan Shankar for supporting the development of our tool.

References

[1] P. Masci, P. Oladimeji, Y. Zhang, P. Jones, P. Curzon, and H. Thimbleby, "PVSio-web 2.0: Joining PVS to HCI," in *27th International Conference on Computer Aided Verification (CAV 2015)*, 2015. Tool and application examples available at http://www.pvsioweb.org.

[2] 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.