PVSio-web 2.0: Joining PVS to HCI

Paolo Masci[1]*, Patrick Oladimeji[3], Yi Zhang[2],
Paul Jones[2], Paul Curzon[1], and Harold Thimbleby[3]

[1] Queen Mary University of London, United Kingdom
{p.m.masci,p.curzon}@qmul.ac.uk

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

[3] Swansea University, United Kingdom
{p.oladimeji,h.thimbleby}@swansea.ac.uk

Abstract. PVSio-web is a graphical environment for facilitating interactive (human-computer) systems modelling and prototyping. Using PVSio-web, one can generate and evaluate realistic interactive prototypes from formal models. PVSio-web has been successfully used over the last two years for analysing commercial, safety-critical medical devices, and has been used to create training material for device developers and device users. It has also been used for medical device design, by both formal methods experts and non-technical end users (including doctors and nurses).

This paper presents the latest release of the tool, PVSio-web 2.0, which will be part of the next PVS distribution. The new tool architecture is discussed, and the rationale behind its design choices are presented.

PVSio-web tool: http://www.pvsioweb.org

Keywords: Prototyping; User Interface Analysis; Practical Formal Methods.

1 Introduction

Inadequate user interface design is repeatedly reported as a root cause of many incidents in healthcare [1, 2], avionics [3], and other safety-critical domains [4]. Design and analysis of user interfaces often requires a multidisciplinary team of human factors specialists, engineers, and end users to validate requirements, specifications, and implementation details. Rigorous formal methods tools can enable early identification of potential design issues. State-of-the-art verification tools like PVS [5], however, generally have minimalist front-ends that create barriers when formal methods experts need to work in a multidisciplinary team and engage with non-experts of formal methods technologies — e.g., to validate hypotheses included in the formal models, or to discuss formal analysis results.

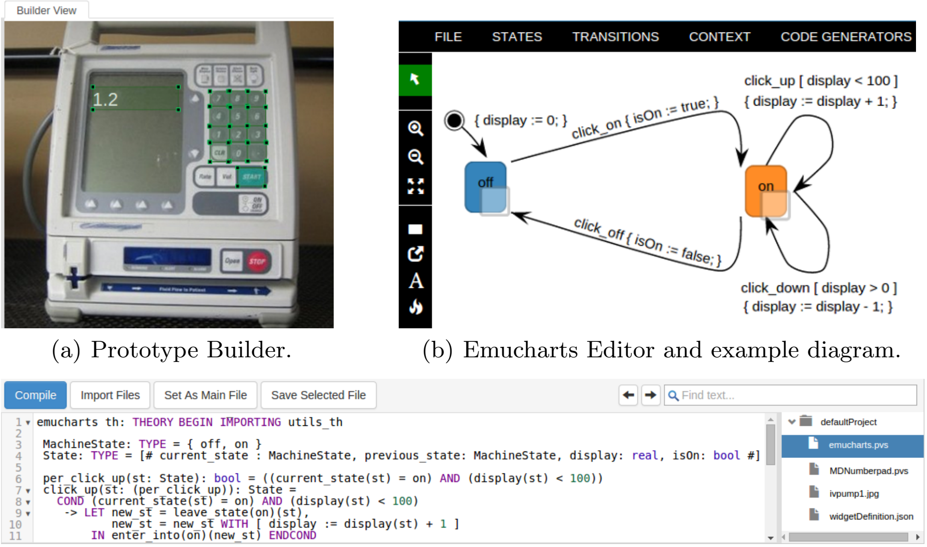
* Corresponding author.

The tool presented in this paper, PVSio-web, significantly reduces these barriers. PVSio-web is a web-based environment for modelling and prototyping interactive (human-computer) systems in PVS, and is particularly suitable for: *validating hypotheses* included in formal models and formal properties before starting the verification process; *demonstrating formal analysis results* to engineers and domain specialists in a way that is easy to comprehend; and *enabling lightweight formal analysis* of user interfaces based on user-centred design methods, such as user testing and expert walkthroughs of prototypes. PVSio-web can be freely downloaded with the latest version of PVS [6] or from our repository [7].

Related work. SCR [8] is a toolset for the analysis of system requirements and specifications. Using SCR, one can formally specify the behaviour of a system, use visual front-ends for demonstrating the system behaviour based on the specifications, and use a group of formal methods tools (including PVS) for the analysis of system properties. In contrast to our tool, SCR lacks specialised functionalities needed for the analysis of user interfaces, such as rapid generation of user interface prototypes, deployment of prototypes on mobile devices, and logging of user interactions. Simulink [9] is a de-facto standard environment for model-based design and analysis of dynamic systems. It provides a graphical model editor based on Statecharts, and functions for rapid generation of realistic prototypes. Unlike our tool, Simulink offers very limited functions for prototyping user interface designs. More importantly, its architecture is not open, preventing it from being used with PVS or other formal analysis tools. In [10], an approach to develop realistic device prototypes using graph models and interactive pictures is presented, but the approach is not supported by a development environment, and the prototypes are manually crafted. PetShop [11], IVY [12] and similar verification tools for formal analysis of user interfaces lack functions for rapid generation of realistic prototypes. Other verification tools like Bandera [13] are not specialised for user interface analysis, and features such as rapid prototyping are out of their intended functionalities.

2 PVSio-web: System Overview and Applications

PVSio-web provides a formal methods based, sophisticated graphical front-end for modelling and prototyping interactive (human-computer) systems. It transforms the animation capabilities of PVS, and enables the user to rapidly generate realistic prototypes in two steps: first, a picture of the user interface is loaded into the tool; second, programmable areas are created over the picture and linked to the formal model specifying the human-system interaction. Programmable areas over input widgets (e.g., buttons) define how user actions are translated into PVS expressions that can be animated within PVSio [14], the native animation environment of PVS. Programmable areas over output widgets (e.g., displays) define how results returned by PVSio are rendered into visual elements of the prototype, so the visual appearance of the prototype can closely resemble the appearance of the real system in the corresponding state.



(a) Prototype Builder.

(b) Emucharts Editor and example diagram.

```
emucharts_th: THEORY BEGIN IMPORTING utils_th

 MachineState: TYPE = { off, on }
 State: TYPE = [# current_state : MachineState, previous_state: MachineState, display: real, isOn: bool #]

 per_click_up(st: State): bool = ((current_state(st) = on) AND (display(st) < 100))
 click_up(st: (per_click_up)): State =
   COND (current_state(st) = on) AND (display(st) < 100)
    -> LET new_st = leave_state(on)(st),
           new_st = new_st WITH [ display := display(st) + 1 ]
        IN enter_into(on)(new_st) ENDCOND
```

(c) Model Editor and a model snippet generated from the diagram in Figure 1(b).

Fig. 1. Screenshots of the main tools provided by the PVSio-web environment.

Figure 1(a) is a screenshot of PVSio-web generating a prototype, where framed boxes are programmable overlay areas: thus, our tool embeds a script in the area over button "0" that translates click actions over the button into a PVS expression `click_0(st)`, and evaluates this expression in PVSio. The function `click_0` is defined in the PVS model of the system, `st` being the current model state. Our tool automatically keeps track of model states during interaction with the prototype, and seamlessly replaces `st` with the current model state. The overlay over the display region renders the value of PVS expressions returned by PVSio (here, as numbers).

Applications. PVSio-web has been applied successfully in the analysis of commercial medical devices. Using PVSio-web, we have:

– Demonstrated previously undetected design issues in medical devices [15,16]
– Validated requirements for medical devices [17–20]
– Created training material [21] for device developers and device users

For example, the prototype shown in Figure 1(a) is one of many that has been used to analyse real medical devices, here a drug infusion pump. Our analysis focused on the data entry defining how the device responds when the user enters configuration parameters into the device, such as therapy data or patient data. The PVS model of the software was obtained by translating the source-code implementation of the user interface software into a PVS theory. Using PVSio-web, we generated a realistic prototype based on the formal model, and used it to perform quick exploratory analyses of the model to understand how to best formalise human factors principles as PVS theorems. An example human factors principle is **consistency**, asserting that the same user action should produce the same results in logically equivalent situations. We formalised this principle

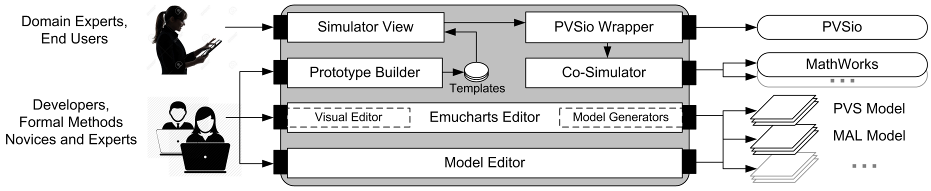


Fig. 2. PVSio-web architecture. Rectangular boxes represent the main modules of the tool; arrows between boxes represent use relationships between modules.

as a PVS theorem that checks whether the data entry system of the device consistently registered button clicks in all model states. This theorem allowed us to discover previously undetected issues with the decimal point (full details of the analysis are in [15]). The same prototype was also used to demonstrate the identified design issue to regulators and real device users (nurses, doctors, medical device trainers), resulting in the recognition of the safety implications of the identified issues. This and other prototypes generated using PVSio-web are currently used in training material for hospitals, device manufacturers, and regulators to raise awareness about general user interface software issues [21].

3 The PVSio-web Architecture

The architecture of the latest release of PVSio-web is shown in Figure 2. The architecture follows the Model-View-Controller [22] design pattern, creating a clear separation between modules responsible for the behaviour of the prototype and those responsible for its visual appearance. In particular, the behaviour of a prototype is entirely specified using PVS executable models animated within PVSio [14]. Note that PVSio is used *as is*, without any modification that might compromise its correctness or sound integration with PVS.

PVSio-web provides multiple facets or perspectives; that is, it combines different user interfaces specifically designed for different target users. One facet, Simulator View, is designed for domain specialists and end users: it includes only elements and functionalities for exploring the behaviour of prototypes generated with PVSio-web. The other facets are designed for developers and formal methods specialists. They provide tools for creating the visual appearance of the prototype (Prototype Builder), and for editing the PVS model. Advanced PVS users can use the Model Editor for editing and type-checking PVS models; developers who are novice PVS users can use a visual model editor (Emucharts Editor) for developing formal models using a graphical notation based on Statecharts [23]. The facets work well together, allowing people with different background and expertise to work together with the same underlying formal models.

► 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. Example prototypes executed within Simulator View are illustrated in [16, 21].

▶ 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. A more detailed description of this module is in [24].

▶ 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. Language constructs for checking well-formedness of the model are automatically embedded in the generated models. For example, the PVS model generator automatically introduces subtyping [25] relations so that consistency and coverage of conditions can be checked with the PVS type-checker (e.g., see the PVS model snippet in Figure 1(c)).

▶ 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.

▶ The **PVSio Wrapper** spawns PVSio processes needed for model animation, and hides the native read-eval-print loop of PVSio behind an API implementing a standard observer pattern [26], with functions for sending commands to PVSio, and for receiving call-backs when PVSio returns a result. This module implements mechanisms to disable inappropriate configurations of our tool, e.g., it disallows spawning multiple concurrent instances of PVSio for demonstrating concurrent systems (the demonstration of such systems must be based on a PVS model that explicitly defines the concurrent behaviour).

▶ The **Co-Simulator** creates a communication infrastructure that enables exchange of simulation events and data between PVS models animated within PVSio and models animated within other simulation environments. This allows one to reuse legacy models already developed using other analysis tools, as well as to employ different modelling/analysis tools in combination for verifying properties of the wider system. Example prototypes using this module to perform co-simulation of PVS models and Simulink models are described in [27,28].

4 Implementation

The core modules of PVSio-web are entirely implemented in JavaScript, which eases the deployment of PVSio-web to mobile devices (tablets, smartphones, etc) allowing demonstrations to be given very conveniently to domain experts. A client-server architecture is used. The server builds on NodeJS [29], and the client relies on the JavaScript engine of web browsers. Jison [30] is employed to automatically generate language parsers from production rules. Model generators use Handlebars [31] for generating formal specifications from model templates. PVSio-web 2.0 includes model generators for PVS, MAL [12], PIM [32], and VDM-SL [33]. PVSio-web text editors build on CodeMirror [34]. PVSio-web visual editors build on D3.js [35].

PVSio-web was first released early 2013 [24]. The tool has been continuously extended with new features, and re-engineered to improve modularity and the overall quality of code. JSLint [36] and Jasmine [37] are routinely used to ensure that our implementation is compliant with established coding standards and that the code is well-formed. The latest version of PVSio-web consists of 18,000 lines of JavaScript code.

5 Conclusions and future directions

PVSio-web shows it is possible and productive to make realistic user interfaces, with all the benefits of web access (mobility, platform independence), connected closely to formal methods tools. PVSio-web makes professional formal methods accessible to end users and others, as is required in best practice for user interface design. We believe that our tool has the potential to improve the development of safe and dependable device user interfaces, as it facilitates using formal methods practices in an area of product design that has typically not made use of this technology. The tool is gaining popularity: it was downloaded over 1,600 times in 2014 [38]; research groups are exploring applications of the tool to the analysis of commercial products in other application domains (e.g., Honeywell is using it to analyse flight decks [39]). PVSio-web has been successfully used in tutorials [40,41] to explain the structure of PVS models, and the meaning of PVS theorems to researchers and students that were not familiar with formal methods. Other universities [42–45] are also using our tool as a basis for projects and thesis for their students. Current and future development directions include improved support for advanced formal verification techniques. For example, we are developing a new visual front-end, Proof Explorer [46], to ease the demonstration of formal proofs, the generation of test cases from verification results, and the development of proof strategies specialised for the analysis of user interface software (example strategies are informally described in [20]). We are additionally developing model generators and co-simulators for linking our tool with other popular formal methods tools, including SAL [47], KeYmaera [48], and Uppaal [49]. We are also regularly adding new case studies in medical and other domains, e.g., for avionics and aerospace.

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.

Disclaimer. The mention of commercial products, their sources, or their use in connection with material reported herein is not to be construed as either an actual or implied endorsement of such products by the U.S. Department of Health and Human Services.

References

1. L. Simone, "Software-related recalls: An analysis of records," *Biomedical Instrumentation & Technology*, vol. 47, no. 6, pp. 514–522, 2013.
2. US Food and Drug Administration (FDA), "Manufacturer and User Facility Device Experience Database (MAUDE)." http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/PostmarketRequirements/ReportingAdverseEvents/ucm127891.htm.
3. G. Gelman, K. Feigh, and J. Rushby, "Example of a complementary use of model checking and human-performance simulation," *IEEE Transactions on Human-Machine Systems*, vol. 44, no. 5, pp. 576–590, 2014.
4. L. Millett, M. Thomas, D. Jackson, *et al.*, *Software for Dependable Systems:: Sufficient Evidence?* National Academies Press, 2007.
5. S. Owre, J. Rushby, and N. Shankar, "PVS: A Prototype Verification System," in *11th International Conference on Automated Deduction (CADE)*, vol. 607 of *Lecture Notes in Artificial Intelligence*, pp. 748–752, 1992.
6. "PVS Specification and Verification System — GitHub repository." https://github.com/samowre/PVS.
7. "PVSio-web tool — GitHub repository." https://github.com/thehogfather/pvsio-web.
8. C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj, "Scr: A toolset for specifying and analyzing software requirements," in *Computer Aided Verification*, pp. 526–531, Springer, 1998.
9. "Mathworks Simulink." http://www.mathworks.com/products/simulink.
10. H. Thimbleby and J. Gow, "Applying graph theory to interaction design," in *Engineering Interactive Systems*, pp. 501–519, Springer, 2008.
11. P. Palanque, J. Ladry, D. Navarre, and E. Barboni, "High-fidelity prototyping of interactive systems can be formal too," in *Human-Computer Interaction. New Trends*, pp. 667–676, Springer, 2009.
12. J. Campos and M. Harrison, "Interaction engineering using the ivy tool," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS09)*, pp. 35–44, ACM, 2009.
13. J. Hatcliff, M. B. Dwyer, C. Păsăreanu, *et al.*, "Foundations of the bandera abstraction tools," in *The Essence of Computation*, pp. 172–203, Springer, 2002.

14. C. Muñoz, “Rapid prototyping in PVS,” Tech. Rep. NIA Report No. 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, 2003.
15. P. Masci, Y. Zhang, P. Jones, P. Curzon, and H. Thimbleby, “Formal verification of medical device user interfaces using pvs,” in *17th International Conference on Fundamental Approaches to Software Engineering (ETAPS/FASE2014)*, (Berlin, Heidelberg), Springer-Verlag, 2014.
16. P. Masci, P. Oladimeji, P. Curzon, and H. Thimbleby, “Tool demo: Using PVSio-web to demonstrate software issues in medical user interfaces,” in *4th Intl. Symposium on Foundations of Healthcare Information Engineering and Systems (FHIES2014)*, 2014.
17. P. Masci, A. Ayoub, P. Curzon, M. Harrison, I. Lee, and H. Thimbleby, “Verification of interactive software for medical devices: Pca infusion pumps and fda regulation as an example,” in *EICS2013, 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM Digital Library, 2013.
18. 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 (SafeComp2013)*, vol. 8153 of *Lecture Notes in Computer Science*, pp. 228–240, Springer Berlin Heidelberg, 2013.
19. P. Masci, R. Rukšėnas, P. Oladimeji, P. Cauchi, A. Gimblett, Y. Li, P. Curzon, and H. Thimbleby, “The benefits of formalising design guidelines: A case study on the predictability of drug infusion pumps,” *Innovations in Systems and Software Engineering, Springer-Verlag London*, 2013.
20. M. Harrison, P. Masci, J. Campos, and P. Curzon, “Demonstrating that medical devices satisfy user related safety requirements,” in *4th Intl. Symposium on Foundations of Healthcare Information Engineering and Systems (FHIES2014)*, 2014.
21. P. Masci, “Design issues in medical user interfaces.” https://www.youtube.com/watch?v=T0QmUe0bwL8.
22. G. Krasner and S. Pope, “A description of the Model-View-Controller user interface paradigm in the Smalltalk-80 system,” *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988.
23. D. Harel, “Statecharts: A visual formalism for complex systems,” *Science of Computer Programming*, vol. 8, pp. 231–274, June 1987.
24. 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.
25. N. Shankar and S. Owre, “Principles and pragmatics of subtyping in PVS,” in *Proc. of WADT ’99*, vol. 1827 of *Lecture Notes in Computer Science*, pp. 37–52, Springer-Verlag, 1999.
26. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software.* Pearson Education, 1994.
27. P. Masci, Y. Zhang, P. Jones, P. Oladimeji, E. D’Urso, C. Bernardeschi, P. Curzon, and H. Thimbleby, “Combining pvsio with stateflow,” in *Proceedings of the 6th NASA Formal Methods Symposium (NFM2014)*, (Berlin, Heidelberg), Springer-Verlag, April-May 2014.
28. C. Bernardeschi, A. Domenici, and P. Masci, “Integrated simulation of implantable cardiac pacemaker software and heart models,” in *2nd International Conference on Cardiovascular Technologies (CARDIOTECHNIX2014)*, ScitePress Digital Library (http://www.scitepress.org), 2014.
29. “Node.js.” http://nodejs.org.

30. “Jison JavaScript Parser Generator.” http://jison.org.
31. “Handlebars Semantic Templates.” http://handlebarsjs.com.
32. J. Bowen and S. Reeves, “Modelling safety properties of interactive medical systems,” in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS13, pp. 91–100, ACM, 2013.
33. P. Masci, P. Larsen, and P. Curzon, “Integrating the PVSio-web modelling and prototyping environment with Overture.” submitted to 13th Overture Workshop, satellite event of FM2015, 2014.
34. “CodeMirror text editor for web browsers.” http://codemirror.net.
35. “D3.js JavaScript library for dynamic creation and control of graphical elements.” http://d3js.org.
36. “JSLint JavaScript Code Quality Tool.” http://www.jslint.com.
37. “Jasmine JavaScript Testing Tool.” http://jasmine.github.io.
38. “Download statistics for package pvsio-web.” http://npm-stat.com/charts.html?package=pvsio-web.
39. B. Hall and D. Bhatt, “Formal Specification And Verification of Human Interactive Interfaces Incorporating Voice Control.” Project Proposal, Honeywell, 2013.
40. “Medical devices and HCI.” Full day tutorial at NordiCHI, 2014, http://cs.swan.ac.uk/~cspo/2014/nordichi/.
41. “Design and analysis of software for interactive medical devices.” PhD module at University of Pisa, 2014, http://phd.dii.unipi.it/formazione/item/85-dr-paolo-masci.
42. N. Robb, “Exploring Aspects of Automated Test Generation on Models.” Honour project, Waikato University, New Zealand, 2015.
43. I. Pascoe, “Usability study of a system that models interactive systems.” Honour project, Waikato University, New Zealand, 2015.
44. E. D’Urso, “Emulink: a graphical modelling environment for PVS,” Master’s thesis, University of Pisa, Italy, 2014.
45. C. Faria, “Web-base user interface prototyping and simulation.” Tese de Mestrado em Engenharia Informática, Universidade do Minho, Portugal, 2014.
46. “Proof Explorer.” https://github.com/thehogfather/ProofExplorer.
47. L. De Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari, “SAL 2,” in *Computer Aided Verification*, pp. 496–500, Springer, 2004.
48. A. Platzer and J. Quesel, “KeYmaera: A hybrid theorem prover for hybrid systems,” in *Automated Reasoning*, pp. 171–178, Springer, 2008.
49. G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, “Uppaal 4.0,” in *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, pp. 125–126, IEEE, 2006.