A Prototyping Process for Medical Devices and Systems

Cinzia Bernardeschi, Andrea Domenici and Maurizio Palmieri

Department of Information Engineering, University of Pisa, L. Lucio Lazzarino, 1, 56126 Pisa, Italy

Abstract

This paper proposes a model-based and formal approach to the development of medical systems: formal modeling is used for system specification, design, and analysis; and interactive simulation, involving end users, is used for system validation. In this approach, the behavior of a medical device is modeled as a hybrid automaton, and concurrently, a realistic interactive graphic interface is developed. The behavioral model is translated into a logic theory, used to formally verify its correctness. The theory is also an executable model that can be integrated with the graphic interface. The resulting virtual device prototype is then simulated within a model of a clinical environment including the model of the patient and other devices. This approach results in higher dependability in patient care: formal methods lead to verifiability of the development process, and simulation allows validation of specifications and designs.

Keywords

Formal specification, prototyping, interactive simulation, co-simulation

1. Introduction

Medical systems, ranging in complexity from simple sensors to integrated clinical environments, are safety-critical cyber-physical systems that clearly must satisfy the most stringent safety and dependability requirements. In spite of that, rigorous development methods and formal verification are not commonly applied in the specification and design of medical systems. The standard approach to ensure safety relies on testing and simulation, which are not sufficient, especially when human interaction is involved. In particular, overlooking subtle issues in designing user interfaces may lead to fatal accidents [1].

This paper proposes an approach to developing medical systems, based on the following concepts: (i) use of *formal modeling* for system specification, design, and analysis, and (ii) use of *interactive simulation* involving end users for system validation.

Formal modeling means using a formal language, such as state machines or logic, to create system models at all levels of abstraction, from specification to implementation. Interactive simulation means allowing developers, maintainers, and operators to interact with a simulated system offering a realistic interface. Developers take advantage of formal modeling as it lends precision and verifiability to the development process, and use simulation to validate

MMHS 2021: 4th International Workshop on (Meta)Modeling for Healthcare Systems, June 21, 2021, Bergen, Norway

cinzia.bernardeschi@unipi.it (C. Bernardeschi); andrea.domenici@unipi.it (A. Domenici); maurizio.palmieri@ing.unipi.it (M. Palmieri)
0000-0003-1604-4465 (C. Bernardeschi); 0000-0003-0685-2864 (A. Domenici); 0000-0002-6177-0928 (M. Palmieri)

specification and design. Maintainers and operators can rely on clear documentation derived from formal models, and use simulation for training and exploration both of system features and system flaws. All this leads to higher levels of dependability for patient care, and its applicability ranges from single devices to complex care environments such as Integrated Clinical Environments (ICE) [2].

In particular, this paper shows the application of the proposed process to modeling and simulation of the SC7000 patient monitor, focusing on the user interface and to the alarm system, and the integration of the developed prototype in the simulation of a simple ICE.

This paper is organized as follows: Sec. 2 provides background information on the tools used in this work; Sec. 3 introduces the proposed process from the point of view of formal modeling, and discusses the role of user interface modeling and prototype simulation; Sec. 4 describes the development of the graphical and behavioral model of a real patient monitor; Sec. 5 describes the co-simulation of an ICE including the monitor; conclusions are drawn in Sec. 6.

2. Background and Related Work

This section introduces background information on the modeling and simulation tools and techniques underlying the proposed development process: The Prototype Verification System (PVS) [3] and the INTO-CPS co-simulation framework [4].

From the literature on the application of formal methods to the development and verification of medical devices, we may cite Masci et al. [5], who advocate the formalization of design guidelines. Safety assurance for medical systems is discussed by Leite et al. [6]. A conceptual model and safety requirements for integrated clinical environments is found in [2], which lays out a conceptual model and safety requirements. High-order logic has been used also as a specification language for ICEs [7] and in the development of an interactive prototype of an infusion pump [8]. The work on simulation of infusion pumps is in part motivated by the danger of adverse events caused by poorly designed user interfaces. Vicente et al. [1] discuss evidence for such danger, and the use of interactive simulation as a training tool is reported in [9].

2.1. PVS and PVSio-web

The Prototype Verification System (PVS) [3] is an interactive theorem proving environment based on a higher-order logic specification language and on a sequent calculus deduction system. With PVS, a system can be modeled as one or more theories, and system properties can be proved by issuing commands that apply inference rules. A built-in type-checking algorithm verifies the consistence of formulas and declaration and, when necessary, it generates *type correctness conditions* (TCC), i.e., auxiliary theorems that the developer must discharge. The PVS environment includes an interpreter, or *ground evaluator*, called PVSio [10], that evaluates fully instantiated function applications. The PVSio package includes functions that produce side effects (e.g., input and output) without affecting the purely declarative nature of a specification.

Another extension to PVS is the PVSio-web [11] toolkit, an environment for modeling, analyzing, and prototyping of human-machine interactive systems. With this toolkit, developers can create a realistic interactive image of the physical interface of a device, and associate areas of the image representing interface components (i.e., displays, buttons, etc.) with operations modeling device behaviors, e.g., the device response to pushing a button. Behaviors can also be specified with an automata-based graphic language, the Emucharts diagram [12]. The toolkit includes code generators for the automatic implementation of Emucharts automata into various languages, including C [13] and the language of the PVS prover. The latter is produced with the translation patterns introduced in [14]. The toolkit has a web interface to build the device prototype and run interactive simulations where a user operates the simulated device through its image.

2.2. INTO-CPS

The Integrated Tool Chain for Model-based Design of Cyber Physical Systems (INTO-CPS) [4] is a co-simulation environment complying with the Functional Mock-up Interface standard (FMI) [15]. The FMI standard has been developed to enable (sub)system models created by different tools to be integrated and simulated together, under the orchestration of a *master* co-simulation algorithm. The INTO-CPS tool-chain provides a *co-simulation orchestration engine* (COE) implementing a master algorithm, and a user interface to set up a co-simulation by choosing and connecting the subsystem models. The COE runs the co-simulation by activating specific simulators for the different submodels and dispatching data among them. In the FMI standard, each submodel is contained in a *functional mock-up unit* (FMU). Besides the model, the FMU may contain a simulation environment and any other resource needed for simulation. The FMI does not mandate any type of implementation for FMUs, as it only defines a set of C APIs that they must support.

3. Formal Modeling

The main contribution of the process proposed in this paper is probably the adoption of formalisms and tools that together provide expressiveness in modeling, rigor and flexibility in verification, and interactivity in simulation.

3.1. System Behavior Model

Among the numerous formalisms available, our approach relies on *hybrid automata* [16, 17] and *higher-order logic* [18]. Automata are a familiar notion to system developers, and hybrid automata, in particular, can describe precisely such systems as medical devices, which often exhibit different time-continuous behaviors in different modes of operation. Further, automata have graphic notations that add to their expressiveness. A higher-order logic can be used to specify both a system's structure and its requirements, and computer-assisted theorem-proving is then used to check if the system's structure satisfies the requirements. However, higher-order logic is probably less familiar than automata to most developers. For this reason, we use a graphical language to build automaton-based models and a tool to generate higher-order logic theories from those models. The role and usage of these languages and tools can be explained without delving into details that will be exposed in later sections. Briefly, an Emucharts model describes a system whose instantaneous state is defined by the values of a finite set of state variables and by one mode of operation (the *current* mode) out of a finite set of

Table 1
Emucharts model elements.

Element	Corresponding to
state variables	patient or device parameters, time
trigger events	user actions, alarm limit crossings, timeouts
modes of operation	states where the device is performing given tasks or waits for user actions
transitions	switching between modes in response to triggers
actions	updating state variables when transitions occur
guards	enabling conditions for transitions, on state variables or time intervals.

modes. Transitions between modes are triggered by events but can be disabled if certain *guard* conditions do not hold. Transitions may also execute actions that change the values of state variables. Table 1 shows how the Emucharts model relates to a device.

A logic theory can be mechanically generated from an Emucharts model according to a procedure first presented in [14] and implemented in the PVSio-web PVS code generator. The core of the theory is a deterministic function ranging over the set of pairs (*state*, *event*) and returning a new state. System properties can then be expressed as theorems to be proved within the theory: For example, one may formally prove that a vital parameter will never exceed certain limits, or that the system will always react correctly to changes of parameters.

3.2. User Interface Model

The user interfaces of medical devices are typically made of electromechanical components (such as toggle switches, push buttons, selectors, lamps…) and electronic ones (such as alphanumeric displays, LED or cathode-ray screens, touch screens…). Devices may also be remote-controlled from a computer console. Designing a human-machine interface (HMI) is difficult, as it must be accessible and readable, and it should invite operators to perform the correct actions but without forcing them to change their working habits, all within hard physical constraints. A poorly designed HMI can at best hinder everyday operations, and at worst cause errors with adverse, even fatal consequences.

With the process proposed in this paper, a developer, or preferably a potential device operator, interacts with an accurate interactive image of the device, which may be a photograph of a physical prototype, if available, or a realistic rendering, showing all the displays and controls, where the areas corresponding to controls are sensitive to operator's actions with a mouse or a touch screen, and the displays show the device outputs.

Graphical displays as described above are made with the PVSio-web toolkit, which provides an editor coded in JavaScript, called the *prototype builder*, to select areas of an image (see Fig. 1) and link them to the input arguments (if the selected area is a control) or the results (if the selected area is a display) of specific PVS functions. These functions are defined in the logic model of the device and model the transitions of the automaton model.

3.3. Simulation

A single device can be simulated as a standalone system. For example, an operator can set the parameters of a simulated infusion pump and observe the progress of the infusion by reading the infused volume on the display, then he or she may perform actions such as suspending the infusion or delivering a bolus. A patient monitor can be simulated by reading recorded values of a patient's parameters from a file, and the display may be switched between blood pressure and heart rate, and so on.

More interesting scenarios can be explored by *co-simulating* more devices, together with a model of a patient's response to drugs or stimuli, such as ventilation. Co-simulation is based on packing different simulation programs into software containers called *functional mock-up units* (FMU) that offer a standard interface to a program that co-ordinates their concurrent execution. In this way, it is possible to simulate, e.g., the interaction of an infusion pump with a patient controlled by a monitor, where the monitor model is written in PVS as described above, the patient's pharmacokinetic model has been developed in Modelica by another clinic, and the Simulink model of the pump has been acquired from the manufacturer.

4. Prototyping a Patient Monitor

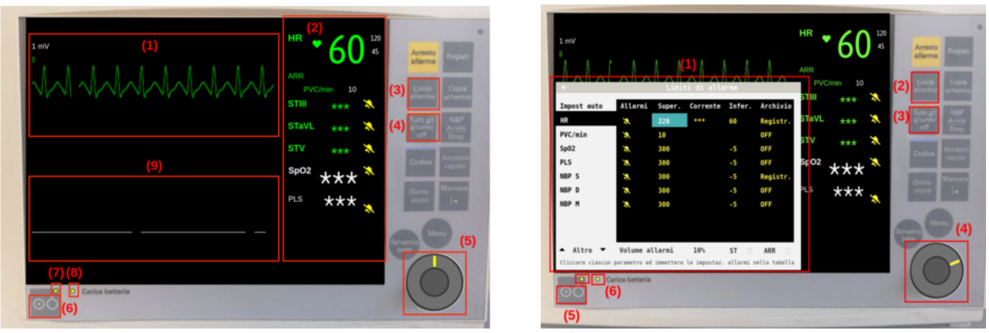
As an example application of the proposed process, this section presents the development of the graphical user interface and the behavioral model of the SC 7000 patient monitor [19]. In this particular case, the purpose is making a virtual prototype that can be used both to validate an *existing* product and to train operators [20]. As usual in real practice, detailed product specifications are not available, so the model has been built on information garnered from the operating manual and experimentation on the real device. In the development of a *new* device, the model would be based on written specifications and interaction with prospective users.

4.1. The Siemens SC 7000 Patient Monitor

The Siemens SC 7000 Patient monitor displays and stores patient parameters and enables operators to set alarms triggered by the crossing of threshold values. The monitored parameters include ECG, respiratory rate, pulse oxymetry, and more. In this work, only the functions related to heart rate and oxygen saturation (SpO2) have been simulated.

4.2. Graphical user interface

The device front panel has a power switch, two LEDs for the power supply, the screen, twelve fixed keys, and a rotary knob incorporating a push button. Some keys open a menu that is browsed by turning the rotary knob and clicking it to select an option. In Fig. 1 (left), the red outlines mark the active areas for the interface widgets of the prototype. Area (1) displays a real-time simulated ECG waveform, whose instantaneous value is displayed in area (2) along with other parameters. Areas (3) and (4) are the fixed keys *alarm limits* and *all alarms off*, and area (5) is the rotary knob. Areas (6), (7), and (8) are the power button and LEDs. Area (9) displays the SpO2 waveform.



	Allarmi	Super.	Corrente	Infer.	Archivio
HR	🔕	220	***	60	Registr.
PVC/min	🔕	10			OFF
SpO2	🔕	300		-5	OFF
PLS	🔕	300		-5	OFF
NBP S	🔕	300		-5	Registr.
NBP D	🔕	300		-5	OFF
NBP M	🔕	300		-5	OFF

▲ Altro ▼ Volume allarmi 10% ST ARR

Cliccare ciascun parametro ed immettere le impostaz. allarmi nella tabella

Figure 1: Prototype graphical interface widgets (left), Alarm management widgets (right).

Figure 1 (right) shows the widgets involved in alarm management. The *alarm limits* key, simulated by area (2), opens the menu shown in area (1). Rotation of the knob in area (4) is done with mouse scrolling, and pushing is done by clicking.

4.3. Modeling device behavior

Using the operating manual and experiment on the real device, the elements of the automaton have been identified as explained in Sec. 3. Figure 2 shows a fragment of the Emucharts model concerning heart monitoring and oxygen saturation. From the initial menu (INIT), pressing the *alarm limits* key (event *alarm_lim_key*) triggers a transition to mode *alarm limits* (ALM_LIMITS). The transition resets variable *et*, modeling elapsed time since the activation of a mode. From the *alarm limits* mode, the user turns the knob right (event *knob_right*) to step through menu options, each corresponding to a mode (AUTO_SET, HBR_ALM, PVC/min_ALM, SpO2_ALM). Turning the knob left (event *knob_left*) steps the cursor back. The *tick* event models the passing of time in discrete steps (simulating 1 s intervals). Transitions triggered by *tick* leave the state unchanged up to sixty time units, and switch to the initial mode above that limit. This models the timeout mechanism that returns control to the initial menu if no user actions occur within one minute.

Among the various interfaces, special focus has been given to alarms, particularly to keys *alarm limits* for alarm management and *all alarms off* for deactivating all active alarms.

Animating a prototype makes it possible to perform what-if analyses under different scenarios, possibly following use cases and system-test plans from requirements specification documents. In the specific case, where the prototype is actually a virtual replica of a marketed product, interactive animation was used to reveal behaviors that would be hard to infer from the user manual or the specifications.

For example, it was found that in the alarm setting modes it is not possible to move backward directly from a submenu to the mode's initial menu (INIT in the Emucharts diagram), which is somewhat counter-intuitive and inconvenient. Finally, animation turns a device prototype into a training tool for medical personnel, providing immediate feedback and increasing confidence in the device.

A PVS theory was generated from the Emucharts model, and the PVS type checker found

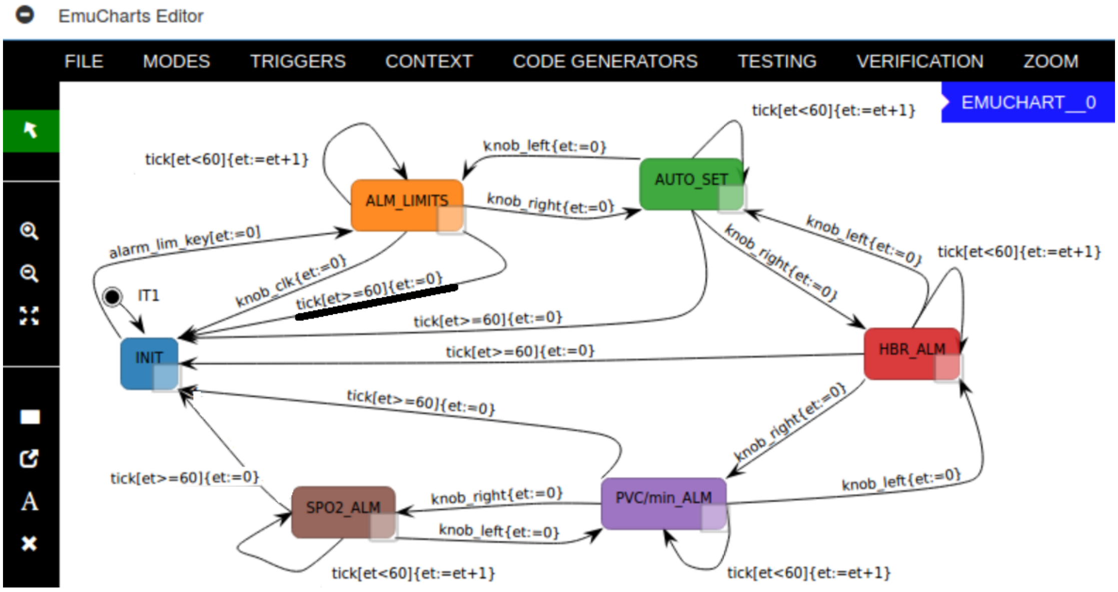


Figure 2: Partial Emucharts diagram for alarm setting.

a flaw in the first version of the model, which omitted the transition whose label has been underlined in Fig. 2. Such a transition, from *alarm limits* mode (ALM_LIMITS) to *init* mode (INIT), implements a timeout. The PVS theory translates each event into a multi-way conditional formula, a declarative counterpart of the *switch* statement in the C language. A missing transition causes the PVS conditional to violate the *coverage TCC* of a conditional, which means that some possible outcomes of the conditional are not taken into account. In this example, the user interface would remain in the ALM_LIMITS mode even after 60 s of inactivity. Such alterations in the normal behavior of a HMI have proved to be a common source of accidents. An attempt to prove the generated TCC guided the developers in finding and fixing the error.

5. Co-simulation in an Integrated Clinical Environment

After preliminary standalone simulations, the SC 7000 prototype has been added to a pre-existing co-simulation scenario, enclosed in a dashed line in Fig. 3. The scenario models a minimal ICE for intravenous infusion, consisting of an infusion pump and a remote-control device. The scenario was extended by plugging-in the FMUs for the monitor prototype and a patient model. The co-simulation begins with the patient presenting a heartbeat rate of 80 bpm. The infusion pump was programmed to infuse an analgesic at a rate of 8.0 ml/hr, causing the heartbeat rate to decrease until it fell below the alarm limit of 70 bpm, previously set on the monitor, and the alarm was activated.

In the co-simulation, an amount of analgesic was infused to the patient with the infusion pump available in the ICE. The values of the patient's parameters from the pharmacokinetic

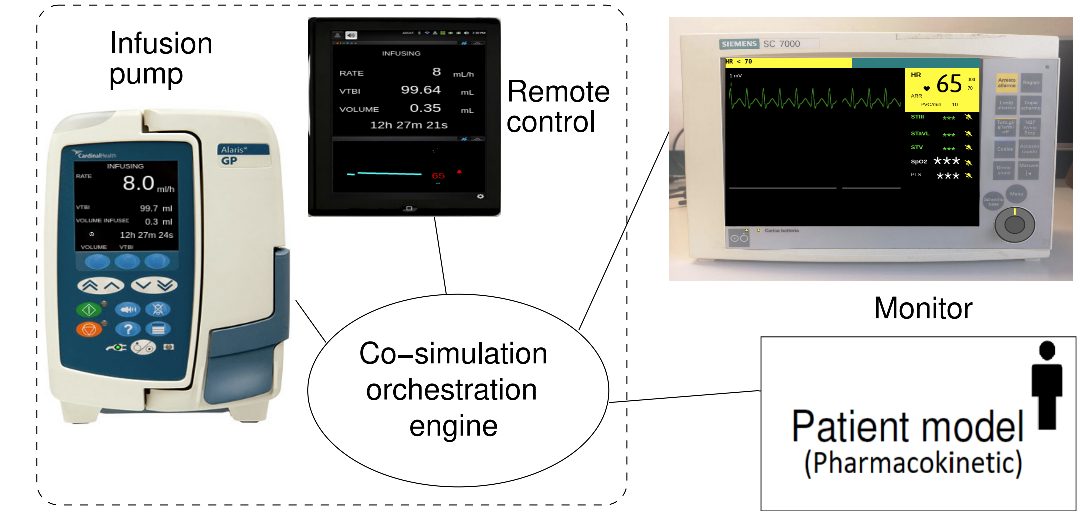


Figure 3: Co-simulation of a basic ICE.

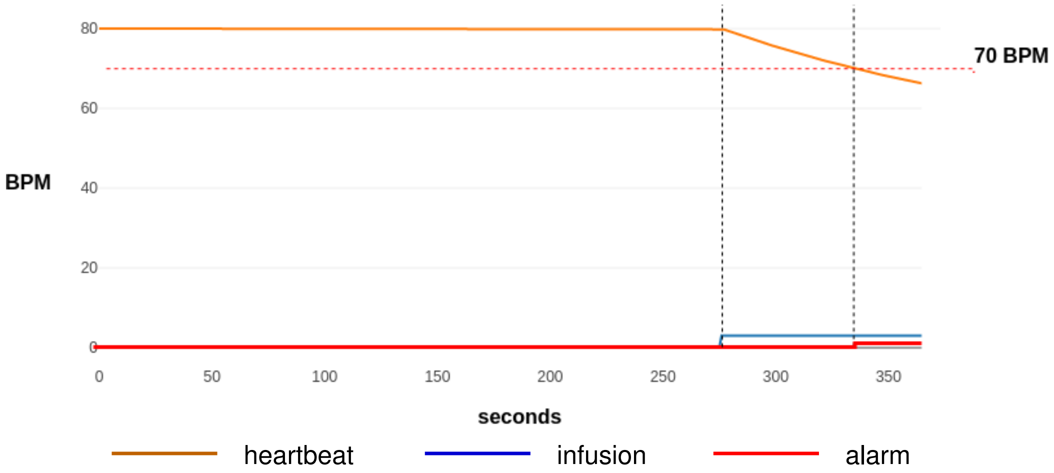


Figure 4: Plot of heartbeat rate.

model were displayed by the prototype of the medical monitor. The co-simulation showed how the patient’s heartbeat parameter changes during the infusion.

Figure 3 shows the prototype of the medical device when the alarm for the heart rate is on. The image shows the heart rate displayed in the notification bar at the top of the screen of the monitor (HR < 70), while the heart rate value of the patient is shown with a yellow background.

While the user interacts with the realistic interfaces of the devices, the system variables are recorded by the INTO-CPS application and plotted against time. Figure 4 shows the decreasing heartbeat rate during infusion, and the time when it crosses the 70 bpm limit (dashed line), triggering the pre-set alarm.

6. Conclusions

This work presented an approach to the development of medical systems, based on formal modeling and interactive simulation, together with the supporting tools. The development of a virtual prototype for a typical patient monitor has shown that: a formal model can be automatically checked for completeness and consistency; interactive theorem proving guide developers in finding and correcting causes of incompleteness and inconsistency; the verified formal model can be executed to simulate the device; and the prototype can be integrated in the co-simulation of a complex care environment where other subsystems are simulated with other tools. The limiting factor for the use of interactive theorem proving is the availability of developers familiar with that technique, but the device interfaces are modeled by theories with a uniform structure and the safety properties to be checked match a small number of patterns, so that the needed proof strategies could be learned quickly, and possibly automatized.

Further, the availability of a graphical prototyping environment makes it possible to build a prototype with a life-like interface that can be used to investigate dangerous design issues and to train device operators. In prospect, the availability of detailed models of device structure and behavior (not just of their interface) and of extended virtual reality support will make it possible to integrate the prototype into full digital twins of care environments.

Acknowledgments

Work partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence).

References

[1] K. Vicente, K. Kada-Bekhaled, G. Hillel, A. Cassano, B. Orser, Programming errors contribute to death from patient-controlled analgesia: Case report and estimate of probability, Can. J. Anaesth. 50 (2003) 328–32. doi:10.1007/BF03021027.

[2] F2761-2009, Medical Devices and Medical Systems — Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) — Part 1: General requirements and conceptual model, ASTM International, 2009.

[3] S. Owre, J. M. Rushby, N. Shankar, PVS: A prototype verification system, in: CADE-11, Springer, London, UK, 1992, pp. 748–752. doi:10.1007/3-540-55602-8_217.

[4] P. G. Larsen, J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis, A. Sadovykh, Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project, in: 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data), 2016, pp. 1–6. doi:10.1109/CPSData.2016.7496424.

[5] P. Masci, R. Rukšėnas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, H. Thimbleby, The benefits of formalising design guidelines: A case study on the predictability of drug infusion pumps, Innov. Syst. Softw. Eng. 11 (2015) 73–93. doi:10.1007/s11334-013-0200-4.

[6] F. L. Leite, R. Adler, P. Feth, Safety Assurance for Autonomous and Collaborative Medical Cyber-Physical Systems, Springer, Cham, 2017, pp. 237–248. doi:0.1007/978-3-319-66284-8_20.

[7] C. Bernardeschi, A. Domenici, P. Masci, Towards a formalization of system requirements for an integrated clinical environment, EAI Endorsed Transactions on Self-Adaptive Systems 2 (2015). doi:10.4108/eai.14-10-2015.2261701.

[8] P. Masci, A. Ayoub, P. Curzon, I. Lee, O. Sokolsky, H. Thimbleby, Model-Based Development of the Generic PCA Infusion Pump User Interface Prototype in PVS, in: SAFECOMP, 2013, pp. 228–240. doi:10.1007/978-3-642-40793-2_21.

[9] C. Bernardeschi, P. Masci, D. Caramella, R. Dell'Osso, The benefits of using interactive device simulations as training material for clinicians: an experience report with a contrast media injector used in CT, ACM SIGBED Review 16 (2019) 41–45. doi:10.1145/3357495.3357500.

[10] C. Muñoz, Rapid prototyping in PVS, Technical Report NIA 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, Hampton, VA, USA, 2003.

[11] P. Oladimeji, P. Masci, P. Curzon, H. Thimbleby, PVSio-web: a tool for rapid prototyping device user interfaces in PVS, in: Proceedings of the 5th International Workshop on Formal Methods for Interactive Systems, 2013. doi:10.14279/tuj.eceasst.69.963.944.

[12] P. Masci, Y. Zhang, P. Jones, P. Oladimeji, E. D'Urso, C. Bernardeschi, P. Curzon, H. Thimbleby, Combining PVSio with Stateflow, in: Proceedings of the 6th NASA Formal Methods Symposium (NFM2014), Springer-Verlag, 2014, pp. 209–214. doi:10.1007/978-3-319-06200-6_16.

[13] G. Mauro, H. Thimbleby, A. Domenici, C. Bernardeschi, Extending a user interface prototyping tool with automatic MISRA C code generation, in: Proceedings of the Third Workshop on Formal Integrated DEvelopment Environment (F-IDE 2016), EPTCS, 2017, pp. 53–66. doi:10.4204/EPTCS.240.4.

[14] C. Bernardeschi, A. Domenici, P. Masci, A PVS-Simulink Integrated Environment for Model-Based Analysis of Cyber-Physical Systems, IEEE Transactions on Software Engineering 44 (2018) 512–533. doi:10.1109/TSE.2017.2694423.

[15] M. Palmieri, C. Bernardeschi, P. Masci, A framework for FMI-based co-simulation of human-machine interfaces, Software and Systems Modeling (2019) 601–623. doi:10.1007/s10270-019-00754-9.

[16] R. Alur, D. L. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235. doi:10.1016/0304-3975(94)90010-8.

[17] T. A. Henzinger, The theory of hybrid automata, in: LICS '96, IEEE Computer Society, 1996, pp. 278–292. doi:10.1109/LICS.1996.561342.

[18] D. Leivant, Higher order logic, in: D. M. Gabbay, C. J. Hogger, J. A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press, Inc., 1994, pp. 229–321.

[19] ASK-T898-03-7600, Siemens SC 7000/8000/9000XL Operating Manual, 1998. ASK-T898-03-7600, Siemens Medical Systems.

[20] J. E. L. Chavez La Valle, Design and Development of a User Interface Prototype for an Electromedical Device, Master's thesis, Dept. of Information Engineering, U. of Pisa, Italy, 2019.