Using PVSio-web and SAPERE for rapid prototyping of user interfaces in Integrated Clinical Environments

Paolo Masci*
Queen Mary Univ. of London,
United Kingdom
p.m.masci@qmul.ac.uk

Piergiuseppe Mallozzi
University of Pisa, Italy
piergiuseppe.mallozzi
@gmail.com

Francesco Luca
De Angelis
University of Geneva, ISS
Carouge, Switzerland
francesco.deangelis@unige.ch

Giovanna Di Marzo
Serugendo
University of Geneva, ISS
Carouge, Switzerland
giovanna.dimarzo@unige.ch

Paul Curzon
Queen Mary Univ. of London
United Kingdom
p.curzon@qmul.ac.uk

ABSTRACT

Integrated clinical environments (ICEs) consist of interoperable medical devices that seamlessly exchange data and commands to create safety interlocks and closed loop controls to improve the quality of care delivered to the patient. Currently at the prototype stage, they promise to form the basis of a new generation of healthcare systems for high acuity patients. Regulators such as the US Food and Drug Administration are promoting the development of tools and techniques for verification and validation of essential safety requirements for ICEs. To date, little research has focused on the certification and assurance of their user interfaces with respect to use errors. In this work, we demonstrate how the PVSio-web prototyping tool can be conveniently used in combination with the communication framework SAPERE to generate realistic ICE systems prototypes from formal models. This approach is particularly suitable for exploring requirements, design, and regulatory issues of usability and safety of the user interfaces of ICE systems. An example ICE system prototype is presented, along with an example analysis demonstrating how the prototype can be used to explore subtle user interface design issues that could lead to usability and safety hazards in clinical scenarios.

Keywords

Interoperable medical devices, Prototyping toolchain, Formal methods technologies

*Corresponding author.

1. INTRODUCTION

Modern medical devices have communication capabilities that can be exploited to improve the safety and effectiveness of the overall medical system. For example, consider a clinical situation where an infusion pump is infusing a painkiller, such as morphine, into the bloodstream of a critically ill patient. If a monitoring device is connected to the patient, then the monitor could interoperate with the pump to stop the infusion when the onset of a respiratory depression is detected, thus saving the patient's life.

The benefits of interoperable medical devices are clear. However, careful design decisions need to be taken to ensure safety of operation. For example, in the previous example, what happens if the patient monitor is accidentally misconfigured, operated incorrectly, or malfunctioning? Is it safe to substitute one interoperable patient monitor with another interoperable monitor in the middle of a clinical situation? It is certainly desirable to have assurance that the infusion pump would operate as safely as in a case where the pump is not exchanging data and commands with the patient monitor.

Regulators such as the US Food and Drug Administration (FDA) are promoting the development of experimental platforms to facilitate the analysis of requirements and architectures for interoperable medical devices. A platform that is currently gaining traction is the Integrated Clinical Environment (ICE) [4]. It is a prototypical model of the next generation medical system for high acuity patients. The conceptual architecture of the ICE prototype (adapted from [4]) is shown in Figure 1. It includes the following main types of technological elements.

► *ICE-compatible medical devices*: medical devices capable of sending and receiving application data and commands using an ICE Network Interface. The ICE Network Interface can be either incorporated in the medical device, or be external to it.

► *An ICE Network Controller*, which creates the necessary communication infrastructure to enable application-level in-

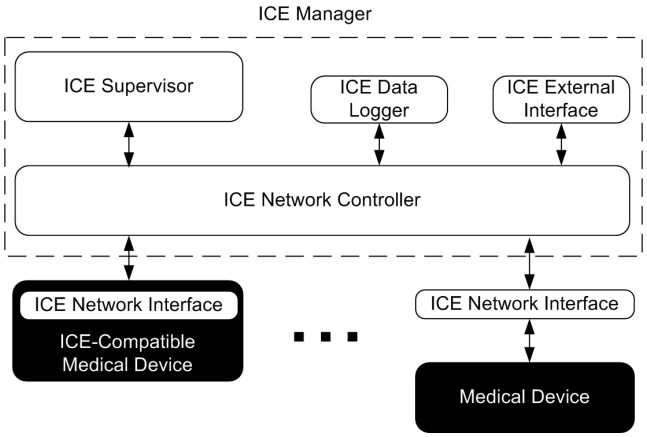


Figure 1: ICE architecture (adapted from [4]).

teroperability among ICE-compatible equipment.

► *An ICE Supervisor*, which defines the application logic to orchestrate and combine ICE-compatible medical devices in a safe manner. Example application logic elements include: safety interlocks, closed-loop controls, and integration of alarm conditions.

► *An ICE Data Logger*, which timestamps and logs data for retrospective analysis of incidents, near-misses, and performance figures.

► *An ICE External Interface*, which connects ICE with other networks, e.g., the hospital's healthcare facility networks, or the Internet.

► *An ICE Manager*, which groups together the four core elements of the ICE architecture: ICE Supervisor, ICE Network Controller, ICE Data Logger and ICE External Interface.

To date, little research has focused on the certification and assurance of the user interfaces of ICE systems with respect to use errors. In the present work, we start to address this gap.

Contribution. We introduce a modelling and analysis toolchain for exploring requirements, design, and regulatory issues related to the usability and safety of ICE systems user interfaces. The toolchain is based on the PVSio-web [10, 15] prototyping tool, and the communication framework SAPERE [1]. It allows developers to generate realistic interactive prototypes of both ICE-compatible medical devices and of the ICE supervisor, and link them together using real communication networks. It uses a model-based approach centred on formal methods technologies. An illustrative example is presented where the toolchain is used to develop an ICE system prototype based on commercial medical devices. An example analysis is also presented to demonstrate how prototypes generated with the toolchain can be used to explore subtle issues in the design of ICE user interfaces.

Organisation. The rest of the paper is organised as follows. In Section 2, related work on tools for modelling and analysis of ICE systems is compared and contrasted to our work. In Section 3, PVSio-web and SAPERE are introduced, as they are the pillars of our toolchain. Then, the main contributions of our work are presented: our toolchain for prototyping ICE systems (in Section 4); an example use of our toolchain for rapid prototyping a realistic ICE system based on commercial medical devices (in Section 5); an example analysis based on the developed prototype is then presented (in Section 6). Finally, Section 7 presents concluding remarks and future work.

2. RELATED WORK

Several research groups have focused on developing testbeds and physical prototypes of ICE systems. The Massachusetts General Hospital MD PnP Lab (www.mdpnp.org), for example, has developed an ICE testbed to facilitate multidisciplinary discussion about safety and performance of ICE systems. In their testbed, they use modified versions of commercial medical devices. Similarly, other two research labs — the Precise Research Center (precise.seas.upenn.edu) and Santos Lab (santoslab.org — developed a Java-based programming environment [7, 17] and a testbed [6] for exploring performance requirements and interoperability of data formats in ICE systems. These approaches are excellent for demonstrating the technical feasibility of ICE systems, but lack the flexibility needed for systematic exploration of different configurations and clinical scenarios. From this perspective, our toolchain complements their effort well: it focuses on user interfaces, rather than performance and data formats; it provides a faster and cheaper approach for rapid prototyping with respect to developing physical prototypes; and it enables the use of advanced formal methods technologies (e.g., theorem proving) for the analysis of the business logic of the system in addition to testing-based approaches. Also, as will be explained, prototypes developed with our toolchain can be used within their testbeds, as our prototypes can use real communication networks to exchange data and commands with other prototypes and devices connected to that network.

Other research has focused on requirements for ICE systems. For example, in [19] and [8], high level requirements are discussed that can be used to assess the safety and effectiveness of the ICE system. In [18], a systematic approach is also presented identifying system requirements that can mitigate identified hazards. Our work also complements this strand of research well, as prototypes developed with our toolchain facilitate multidisciplinary discussion between engineers and domain experts for identifying and validating system requirements. Furthermore, our toolchain gives a testbed with which to demonstrate how a variety of formal tools can be usefully used to formalise and verify ICE requirements. It can also be used to explore the different kinds of assurance arguments that can be supported by a verification effort. This can be based on concrete examples developed within the toolchain.

3. BACKGROUND

Our toolchain for model-based generation of ICE systems prototypes is based on the prototyping tool PVSio-web [10, 15] and the communication framework SAPERE [1]. In this

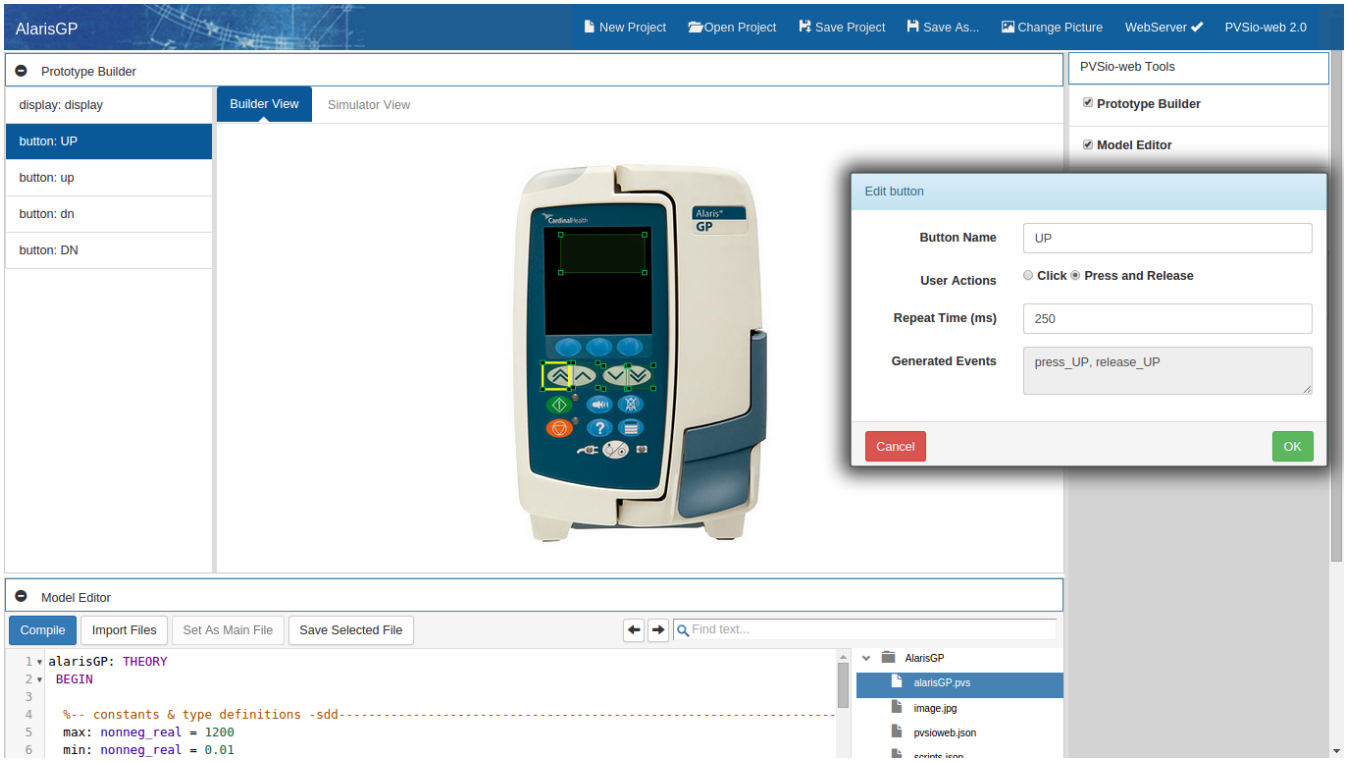


Figure 2: Snapshot of the PVSio-web environment while creating an interactive device prototype, in this case an infusion pump used in healthcare to infuse medications or nutrients to patients. Framed boxes of the picture highlight the interactive parts of the prototype.

section, we therefore introduce these two technologies, and illustrate the main functionalities that we have used in our toolchain.

3.1 PVSio-web

PVSio-web is a new research tool for modelling, prototyping, and analysis of interactive systems. It is based on the industrial strength formal verification system, PVS [16], which provides sophisticated proof, animation, and modelling environment. PVSio-web provides a graphical environment for creating realistic stand-alone prototypes based on underlying executable formal models (see Figure 2). To interact with a PVSio-web prototype, developers click buttons on the prototype's user interface, and watch the results of the interactions in real time on its displays. Underneath the graphical environment, PVSio-web uses the PVSio [14] component of PVS for model animation. The tool is particularly suitable for: validating system specifications and requirements before starting the verification process; demonstrating formal analysis results to engineers and domain experts in a way that is easy to understand; and enabling lightweight formal analysis based on user centered design methods, such as user testing and expert walkthroughs of prototypes. PVSio-web has been successfully used to demonstrate previously undetected design flaws in medical devices [12], and to explore the causal relationships between user interface design issues and software defects [9]. Additional information about how prototypes are generated using PVSio-web are presented as needed in Section 5, while illustrating the development of the ICE system prototype.

3.2 SAPERE

SAPERE [1] is a communication and coordination framework for heterogeneous networked systems. It provides a common platform to build distributed multi-agent pervasive systems, and offers automated functionalities for data exchange and aggregation of heterogeneous services, applications and sensors. The exposed interface infrastructure supports dynamic discovery of devices and services, as well as sophisticated nature-inspired aggregation algorithms for distributed processing of data. The framework has been successfully used to engineer multi-agent systems in the context of pervasive scenarios [20], and supports a development methodology that facilitates reusability and composability of services in pervasive systems [3].

The SAPERE coordination model is composed of the following components.

► **Live Semantic Annotations (LSAs)**: active tuples composed of one or several properties of type $(name, value)$ encapsulating information about entities taking part in coordination processes. LSAs can interact with each other; such process is guided by predefined laws named Eco-Laws.

► **LSA Space**: container of shared LSAs hosted in a single network node. Inside an LSA Space, reactions among LSAs are spontaneously fired by Eco-laws.

► **Agents**: active entities that coordinate the communication processes between external components (services, sen-

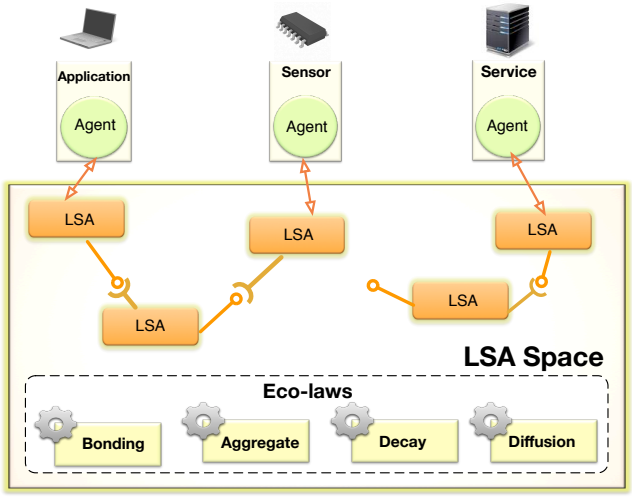


Figure 3: SAPERE coordination model

sors, applications, etc.) and the LSA Space. Each agent is bound to one LSA, and implements the logic for managing notification of events triggered when the LSA interacts with other LSAs of the same LSA Space.

► **Eco-laws**: active entities implementing four basic bio-inspired mechanisms that enforce interactions among LSAs. Eco-laws are executed periodically at a specific rate, tunable by the system administrator. In the developed toolchain, we use the following Eco-laws:

(i) *Bonding*: instant relationships among LSAs. A bond is a unidirectional link created between an LSA A containing a property of type $(name\_x, *)$ or $(name\_x, ?)$ and a second LSA B containing a property $(name\_x, value\_x)$. Once the bond is created, the agent associated with A can read and monitor changes to the value $value\_x$ of the property in B. Such mechanism represents the main modality to share information among agents. Values $*$ and ? are named operators and they allow for the creation respectively of one or several bounds at the same time.
Example: Given LSAs $A = [(temperature, ?)]$ and $B = [(temperature, 15)]$, a bond will be created between A and B because of the property $(temperature, ?)$. Through this bond, the agent associated with A will be able to read the property $(temperature, 15)$.

(ii) *Decay*: reduction of information relevance. This mechanism automatically updates an LSA containing a property $(decay, n)$ by decreasing the numeric property value n by one unit at every execution of the Eco-law. Once $n = 0$, the whole LSA is removed from the space.
Example: the LSA $[(temperature, 15), (decay, 10)]$ will be deleted from the LSA Space after 10 executions of the *decay* Eco-law.

Eco-laws generate notifications to keep agents updated respect to the changes applied to their tuples. Several more complex bio-inspired mechanisms (e.g. *gradient*, *chemotaxis*, *quorum sensing*,...) can be obtained by opportunely combining four basic Eco-laws (bonding, decay, aggregate, and diffusion) provided by SAPERE.

4. OUR TOOLCHAIN

We now introduce our modelling and analysis toolchain for exploring requirements, design, and regulatory issues related to the usability and safety of ICE systems user interfaces. The PVSio-web prototyping tool is used to generate realistic interactive prototypes of both ICE-compatible medical devices and of the ICE supervisor. The communication services provided by SAPERE are used to enable exchange of data and commands between prototypes developed using PVSio-web. To enable the combined use of PVSio-web and SAPERE, we have developed a new PVSio-web extension, the *PVSio-web Network Controller*. This new extension installs virtual communication ports on the PVSio-web prototypes, and allows developers to "plug in" PVSio-web prototypes in to real communication networks to discover devices connected to the network, and exchange data and commands with them.

Our toolchain based on PVSio-web and SAPERE therefore leads to ICE system prototypes that include the following elements (see Figure 4):

► **A PVSio-web prototype for the ICE Supervisor.** A PVS model defines the internal logic of the supervisor and the behaviour of the supervisor's user interface. A picture of the supervisor's user interface defines the visual appearance of the prototype.

► **A PVSio-web prototype for each ICE-Compatible Medical Device in the system.** A PVS model defines the interactive behaviour of each device. A picture of the device user interface defines the visual appearance of each prototype.

► **A PVSio-web Network Controller** to enable real-time exchange of data and commands between the PVSio-

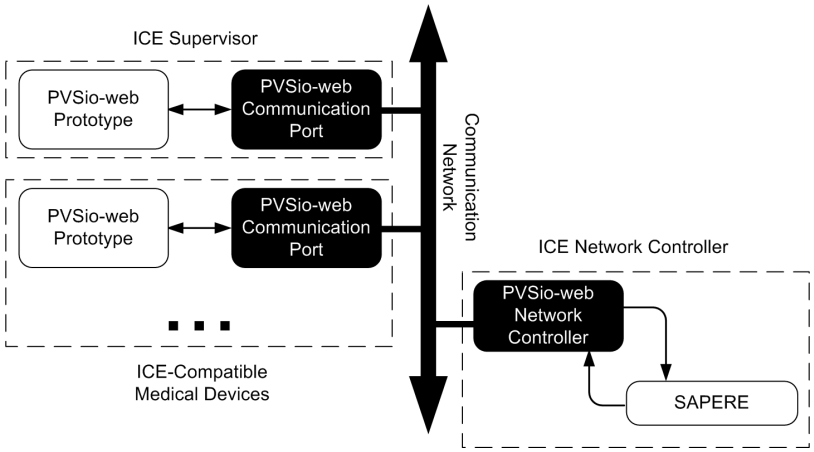


Figure 4: Logic architecture of the ICE system prototype generated using PVSio-web and SAPERE. Boxes in the picture represent functional modules (black boxes identify new PVSio-web extensions developed to support interoperability between prototypes). Arrows between modules represent flow of commands/data.

web prototypes during simulation. The PVSio-web Network Controller uses an instance of SAPERE to implement the communication service.

4.1 The PVSio-web Network Controller

The PVSio-web Network Controller uses SAPERE to mimic a publish-subscribe communication protocol. That is, each PVSio-web prototype includes a Network Controller Agent that can publish messages for the PVSio-web prototype, and receive messages published by other PVSio-web prototypes connected to the same network. To do this, each Network Controller Agent includes a cluster of SAPERE agents (see Figure 4). Agents in the cluster implement different functions: one agent (PublishAgent) is for publishing messages; and one or more agents (SubscribeAgents) are used for receiving messages published by other PVSio-web prototypes through their PublishAgent.

The developed PVSio-web Network Controller has a graphical user interface to monitor and control the status of the network. A snapshot of the Network Controller user interface is in Figure 6. It shows a scenario with three PVSio-web prototypes (represented as labelled boxes with control buttons disconnect/remove) are connected to the network and exchange messages. One prototype acts as (ICE) supervisor, whilst the other two act as (ICE-compatible) medical devices. In the snapshot, SAPERE agents are represented as circles. Lines connecting circles represent flow of messages and commands between prototypes. Note that, in Figure 6, the supervisor prototype has two SubscribeAgents because it is designed to receive data published by two device prototypes connected to the network. The device prototypes, on the other hand, have just one SubscribeAgent, as they are designed to receive data only from the supervisor in this case.

5. EXAMPLE

In this section, we use our toolchain based on PVSio-web and SAPERE to develop an ICE system prototype. The aim of this example is to demonstrate that our toolchain facilitates the rapid generation of realistic ICE system prototypes suitable to explore usability- and safety-related aspects of ICE user interfaces. This is especially useful at the early stages of the development lifecycle of these systems, when physical prototypes of ICE-compatible devices and the ICE infrastructure are not readily available, as well as when a full specification of the system is still under development.

In the following, we first present a description of the functionalities of our example ICE system. Then, we illustrate the steps followed to generate a realistic ICE system prototype using our toolchain. Then, in Section 6, we discuss the types of analysis that can be carried out with the prototype, and illustrate an example analysis that reveals subtle gaps in the interactive behaviour of the system that could lead to usability and safety issues.

5.1 Description of the ICE system

The considered ICE system is based on one of the representative ICE clinical scenarios illustrated in [4] and [19]. It includes two types of medical devices: a patient controlled analgesia (PCA) pump, which delivers a continuous infusion of a pain killer (morphine) into the bloodstream of a critically ill patient; and a patient monitor, which continuously measures the patient's condition by checking her/his respiratory rate and blood oxygen saturation level. The ICE supervisor implements a safety interlock application that automatically stops the infusion when the patient's monitored parameters indicate the onset of a respiratory depression.

5.2 Infusion pump prototype

The infusion pump prototype is based on a commercial medical product [2]. The device has a user interface with a display and buttons for setting up the infusion parameters and control the delivery of the therapy. A blueprint of the device user interface is shown in Figure 7(a). It includes:

► A display, capable of rendering the current pump mode, the infusion parameters programmed in the pump; and labels for three programmable function keys;

► Two pairs of chevron keys for editing infusion parameters

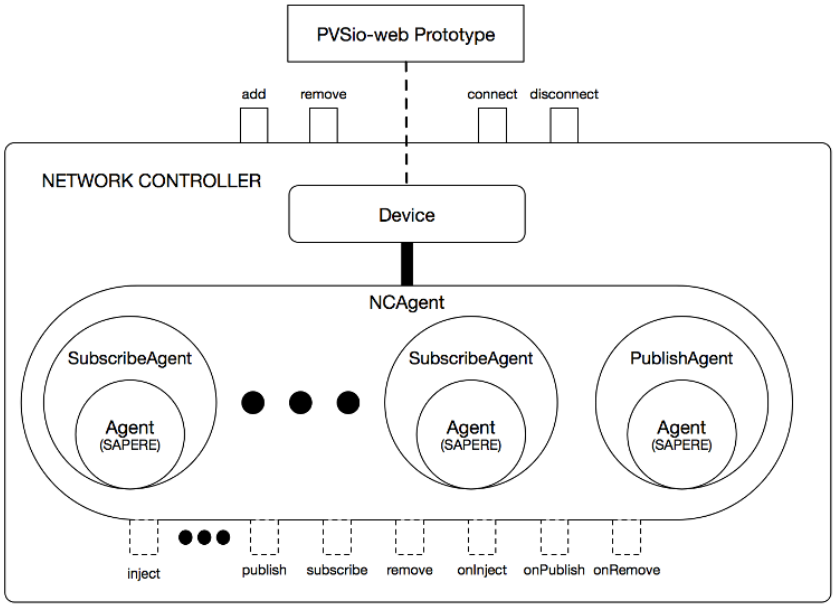


Figure 5: Architecture of the PVSio-web Network Controller.

and selecting menu items;

► A bolus button, to trigger the delivery of bolus doses;

► A pause button, to pause the infusion;

► A power button, to power the pump on and off

The device has three main modes of operation: *infusing*, where the pump is infusing; *on hold*, where the infusion is paused; and *alarm*, where the pump signals situations that need to be solved by a human operator, e.g., occlusion of the infusion line. Setting up the infusion involves interacting with the pump user interface to configure two main parameters: infusion rate, and volume to be infused. Once these parameters are set, the infusion can be started. Infusion parameters can be changed while the infusion is running, e.g., to adjust the therapy to changed patient conditions. A detailed description and analysis of the user interface design of this device is in [11, 15, 5].

Developing the prototype. Using PVSio-web, we generated an interactive prototype of the device using an executable PVS model of the device and a picture of the real device (see Figure 7(b)). That is, we loaded a picture of the real device in our PVSio-web prototyping environment, created interactive areas over buttons and displays of the picture, and linked them with the PVS model of the device. Namely, user actions over buttons in the picture are associated with state transition functions defined in the PVS model. These functions take the current state of the model as argument, and return the next model state as result of the function evaluation. For example, click actions over the *run* button are associated with a function `click_run` defined in the PVS model. Therefore, every time the *run* button is clicked by the user, PVSio-web automatically triggers the evaluation of function `click_run` over the current model state. The next model state is thus obtained, and rendered using the displays of the prototype. For display elements, interactive areas are created over the device displays and LED lights, and associated with corresponding fields in the PVS model state. Complex display elements that render multiple information are segmented into simpler sub-displays. For example, the device display includes a topline, which is associated with a field `topline` defined in the state of the PVS model, and 4 additional sub-displays, each rendering a different infusion parameter (in this case, infusion rate, volume to be infused, volume infused, and remaining time). The PVS model used for the prototype includes information about which sub-display is visible in which device mode. This information is used by PVSio-web to dynamically reveal and conceal sub-displays of the prototype, and thus closely resemble the visual look and feel of the real device in different screens.

5.3 Patient monitor prototype

The patient monitor prototype is based on a commercial medical product [13]. The device has a touchscreen user interface, and a mechanical button to power on and off the monitor. A blueprint of the user interface of the device is shown in Figure 8(a). It includes:

► A touchscreen display for rendering the patient's vital signs, and to setup alarm thresholds. Here, we consider two vital signs, as indicated in the clinical scenarios described in [4] and [19]: the patient's blood oxygen saturation level (SpO2), and the respiratory rate (RRa).

► A power button, to power on and off the patient monitor.

The device has two main modes of operation: monitoring, and alarming. When in monitoring mode, the device renders

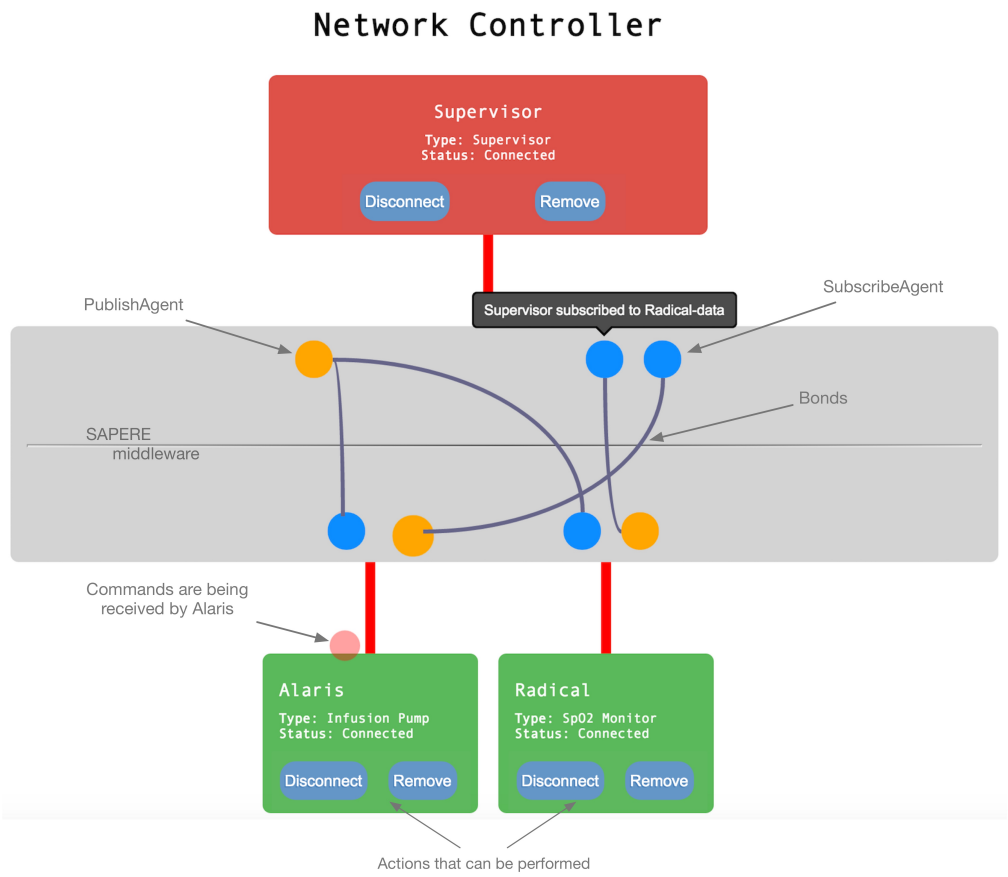


Figure 6: User interface for accessing the functionalities of the PVSio-web Network Controller. Snapshot of an ongoing simulation. Circles represent SubscribeAgents or PublishAgent. Lines between circles represent flow of information between agents.

the current values of SpO2 and RRa, the recent history of measured values, and the alarm levels. When alarming, a blinking alarm icon is rendered next to the current value of the patient's vital sign that is generating the alarm. By default, the device is in monitoring mode. The device mode changes into alarming when the current value of at least one of the monitored vital signs exceeds given alarm thresholds pre-set by the operator (typically, a nurse).

Developing the prototype. Using PVSio-web, we generated an interactive prototype of the device (see Figure 8(b)). As for the infusion pump prototype, we then loaded a picture of the real device in PVSio-web, and created interactive areas over buttons and displays to link them to the PVS model of the device. Button clicks on the power button are thus associated with a function `click_onOff` defined in the PVS model. Two display elements were created, one for each monitored vital sign. Each display element is further segmented into five sub-displays: three sub-displays render numerical data (current value, and maximum and minimum alarm levels); another sub-display renders alarms (in this case, an alarm icon); and one additional sub-display renders historical values (tracings displays).

5.4 ICE supervisor

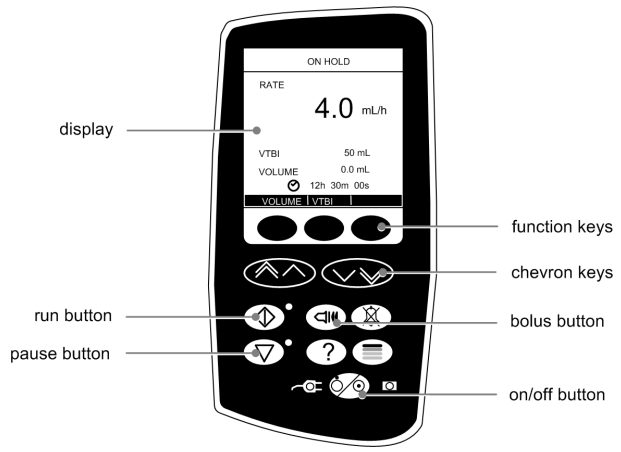
The blueprint of an ICE supervisor user interface is shown in Figure 9(a). It includes:

► A touchscreen display for rendering the infusion pump status.

► A touchscreen display for rendering the status of the patient monitor.

► A power button, to power the ICE supervisor on and off.

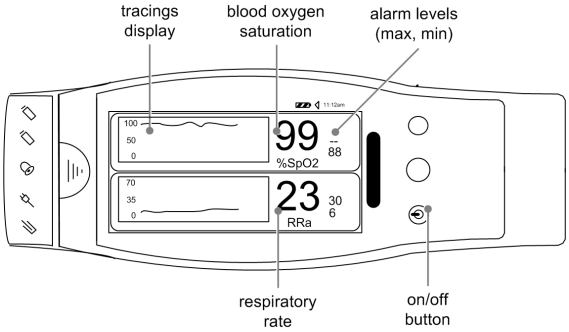
By default, the ICE supervisor user interface renders the same information shown on the main screens of the medical devices connected to the system. For the patient monitor, the ICE display therefore presents the current value of the measured SpO2 and respiratory rate, the alarm levels, and the recent history of measurements. For the infusion pump, the ICE display shows the infusion parameters (infusion rate, volume to be infused, volume infused, and remaining time), and the current pump status (on hold, infusing, etc.). Additionally, the ICE user interface allows operators to view historical data of infusion parameters. This function can be accessed with a touch gesture on the infusion parameters rendered on the prototype display.



(a) Blueprint of the AlarisGP pump, identifying the main elements of the user interface.

(b) PVSio-web prototype of the AlarisGP pump. The figure shows a snapshot of the prototype executed within PVSio-web.

Figure 7: ICE-compatible infusion pump prototype based on the Alaris GP pump.



(a) Blueprint of the Radical7 patient monitor, identifying the main elements of the user interface.

(b) PVSio-web prototype of the Radical7 monitor. The figure shows a snapshot of the prototype executed within PVSio-web.

Figure 8: ICE-compatible patient monitor prototype based on the Radical7 monitor.

The safety interlock mechanism implemented in the ICE supervisor has the following logic. If the measured levels of two monitored parameters (in this case, SpO2 and RRa) are outside given safety thresholds, then a command is automatically sent to the pump to stop the infusion, and a respiratory distress alarm is also generated to alert the nurse. The generated alarm is marked as high priority if the measurements are critically low and indicate distress. Otherwise, the alarm is marked as medium priority.

Developing the prototype. Unlike the other two prototypes, realistic physical prototypes of ICE supervisors are not yet publicly available. We therefore borrowed the skin of another commercial product, and used it to create the prototype (see Figure 9(b)). A PVS model was also developed to specify the internal logic of the supervisor for detecting respiratory distress and send commands to the pump. The internal logic is specified in a function *tick*, which models the periodic operations performed by the device to check whether the conditions are met that trigger a pause command for the pump. The state of the supervisor includes fields for storing data received from the pump and the monitor. For the developed system, the pause command sent by the ICE supervisor to the pump has the same effect as pressing the pause button on the infusion pump user interfaces.

6. EXAMPLE ANALYSIS

Using the developed prototype, we explored usability- and safety-related aspects of the ICE user interface. The performed analysis is based on simulations, and involved interacting with the developed prototypes in different clinical situations. In particular, we were interested in checking what happens when one of the ICE-compatible devices (either the infusion pump or the patient monitor) is re-programmed in the middle of an alarm situation.

Intersting cases were quickly identified where the system behaved unexpectedly because of subtle combinations of events. For example, we found that the safety interlock application can cause mode changes in the pump while entering infu-

(a) Blueprint of ICE supervisor.

(b) PVSio-web prototype of the ICE supervisor executed within PVSio-web.

Figure 9: ICE supervisor prototype.

sion parameters. For the specific device used in the ICE prototype, this happened when a pause command sent by the ICE supervisor is received by the pump while the pump is infusing and the operator is reprogramming the volume to be infused (VTBI). Upon receiving the pause command, the pump suddenly changes mode of operation from *Edit VTBI* to *Edit Rate*, without halting data entry. The combination of events could happen either coincidentally at the same time when the operator is reprogramming the pump, or as a result of the reprogramming of the pump. In either case, if the data entry mode changes and the operator does not notice the change, a programming error could occur, where volume to be infused is accidentally changed. It is worth noting that the identified combination of events are unlikely to happen when the pump is operated as a stand-alone device, disconnected from the ICE system – to reproduce the same situation, the operator would need to press simultaneously the data entry buttons and the pause button on the infusion pump user interface. When connected to ICE, on the other hand, the pause command is automatically triggered by the ICE supervisor, potentially at any time.

Other similar exploratory analyses can be carried to articulate scenarios, requirements, and concerns related to the behaviour of the system. This is extremely valuable within the development lifecycle of ICE systems, as it helps engineers to understand *what* needs to be verified in the system to ensure safety of operation. Being based on formal models, the same prototypes can then be used for a full formal analysis of safety and usability properties, e.g., following the approach described in [12, 5]

7. CONCLUSIONS

Rigorous analysis tools based on formal methods technologies enable systematic and *full* exploration of system configurations and inputs. Tools, however, are needed that can reduce the perceived cost of using these powerful technologies. Tools are also needed that can be used by domain experts that are not familiar with formal methods to assess *what* is being verified. Without a clear understanding of what is being verified, the verification effort is potentially pointless, e.g., because the verified properties depart from the intended meaning of safety requirements used in assurance arguments for the system. Our prototyping toolchain moves in this direction, as it facilitates multi-disciplinary discussion of safety- and usability-related aspects of ICE user interfaces. Being based on formal models, the same models developed for the prototypes can be used for full formal analysis of the ICE system behaviour. An aspect that needs investigation is how to best include the ICE Network Controller in the formal analysis of the system. In the developed ICE system prototypes, in fact, the ICE Network Controller is software code, rather than formal models. A promising solution that we are exploring is the definition of abstract models of the ICE Network Controller that capture hypotheses about the quality of service offered by the controller, e.g., its ability to deliver messages to the intended destinations. This way of proceeding facilitates the verification of ICE systems for broad classes of ICE Network Controllers (as opposed to specific implementations of the ICE Network Controller). Further work is also needed to understand how nature-inspired algorithms, such as those implemented in SAPERE, can be used to improve the performance of the system. An interesting option is, under this perspective, to implement another variant of the PVSio-web Network Controller, that includes a different communication middleware. This would give us insights about the performance and reliability of different communication middleware.

Acknowledgments

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

References

[1] Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17, Part B:236 – 252, 2015. 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.

[2] Cardinal Health Inc. Alaris GP volumetric pump: directions for use, 2006.

[3] F. L. De Angelis, J. L. Fernandez-Marquez, and G. Di Marzo Serugendo. Self-composition of services in pervasive systems: A chemical-inspired approach. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 37–46. Springer International Publishing, 2014.

[4] J. M. Goldman. Medical Devices and Medical Systems-Essential safety requirements for 5 equipment comprising the patient-centric integrated clinical environment 6 (ICE)-Part 1: General requirements and conceptual model 7. *ASTM International*, 2008.

[5] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon. Demonstrating that medical devices satisfy user related safety requirements. In *4th International Symposium on Foundations of Healthcare Information Engineering and Systems (FHIES2014)*, 2014.

[6] A. King, D. Arney, I. Lee, O. Sokolsky, J. Hatcliff, and S. Procter. Prototyping closed loop physiologic control with the medical device coordination framework. In *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, pages 1–11. ACM, 2010.

[7] A. King, S. Procter, D. Andresen, J. Hatcliff, S. Warren, W. Spees, R. Jetley, P. Jones, and S. Weininger. An open test bed for medical device integration and coordination. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 141–151. IEEE, 2009.

[8] B. Larson, J. Hatcliff, S. Procter, and P. Chalin. Requirements specification for apps in medical application platforms. In *Proceedings of the 4th International Workshop on Software Engineering in Health Care*, pages 26–32. IEEE Press, 2012.

[9] P. Masci, P. Oladimeji, P. Curzon, and H. Thimbleby. Tool demo: Using PVSio-web to demonstrate software issues in medical user interfaces. In *4th International Symposium on Foundations of Healthcare Information Engineering and Systems (FHIES2014)*, 2014.

[10] P. Masci, P. Oladimeji, P. Curzon, and H. Thimbleby. PVSio-web 2.0: Joining PVS to Human-Computer Interaction. In *27th International Conference on Computer Aided Verification (CAV2015)*. Springer, 2015. Tool and application examples available at http://www.pvsioweb.org.

[11] P. Masci, R. Rukšėnas, P. Oladimeji, A. 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.

[12] P. Masci, Y. Zhang, P. Jones, P. Curzon, and H. Thimbleby. Formal Verification of Medical Device User Interfaces Using PVS. In *ETAPS/FASE2014, 17th International Conference on Fundamental Approaches to Software Engineering*, Berlin, Heidelberg, 2014. Springer-Verlag.

[13] Masimo. Radical7 patient monitor: system overview. http://www.masimo.com/rainbow/radical7.htm, 2015.

[14] C. Muñoz. Rapid prototyping in PVS. Technical Report NIA Report No. 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, 2003.

[15] 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.

[16] S. Owre, J. 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.

[17] S. Procter and J. Hatcliff. An architecturally-integrated, systems-based hazard analysis for medical applications. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on*, pages 124–133. IEEE, 2014.

[18] S. Proctor, J. Hatcliff, A. Fernando, and S. Weininger. Using stpa to support risk management for interoperable medical systems. 2015 STAMP Workshop Presentations, 2015.

[19] S. Weininger, Y. J. Kim, J. Hatcliff, V.-P. Ranganath, and Robby. Integrated clinical environment device model: Stakeholders and high level requirements. 2015.

[20] F. Zambonelli, A. Omicini, B. Anzengruber, G. Castelli, F. L. DeAngelis, G. Di Marzo Serugendo, S. Dobson, J. L. Fernandez-Marquez, A. Ferscha, M. Mamei, S. Mariani, A. Molesini, S. Montagna, J. Nieminen, D. Pianini, M. Risoldi, A. Rosi, G. Stevenson, M. Viroli, and J. Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17:236–252, 2015. Special Issue "10 years of Pervasive Computing" In Honor of Chatschik Bisdikian.