The Benefits of Using Interactive Device Simulations as Training Material for Clinicians: an Experience Report with a Contrast Media Injector Used in CT

Cinzia Bernardeschi
cinzia.bernardeschi@unipi.it
Department of Information Engineering
University of Pisa, Italy

Paolo Masci
paolo.masci@inesctec.pt
INESC TEC and Universidade do Minho
Braga, Portugal

Davide Caramella
davide.caramella@med.unipi.it
Dipartimento di Ricerca Traslazionale e delle Nuove Tecnologie in Medicina e Chirurgia
University of Pisa, Italy

Ruggero Dell'Osso
ruggero.dellosso@med.unipi.it
Dipartimento di Ricerca Traslazionale e delle Nuove Tecnologie in Medicina e Chirurgia
University of Pisa, Italy

ABSTRACT

This paper reports on our experience in developing training material for a hospital, in the form of an interactive simulation of a medical device routinely used in the hospital. The subject device is a commercial contrast media injector used in Computed Tomography (CT) scans. The specification of the device was reverse engineered using in combination the user manual, direct interaction with the real device, and the results of a field study we conducted that focused on how expert users routinely operate the device. The interactive simulation greatly helped to identify critical workflows that could induce accidental use errors that lead to dangerous situations such as failure to correctly detect air-in-line before starting the injection. The interactive simulation proved also useful to stimulate a constructive discussion within a multidisciplinary team of engineers and clinicians, about possible design improvements to the device that could prevent the identified critical workflows.

KEYWORDS

Contrast media injectors; Interactive simulation; Training material for clinicians.

1 INTRODUCTION

Contrast media injectors are medical devices routinely used in diagnostic imaging exams such as computed tomography (CT), magnetic resonance imaging (MRI) and angiography (XA), to inject intravenously a fluid, called contrast media, necessary to enhance the visibility of normal body structures as well as lesions.

Iodinated media can be nephrotoxic, being a known cause of possible acute renal failure in hospitalised patients [12]. To minimize the risk of adverse health problems, it is therefore important to set up the injector so that it delivers the minimal amount of contrast media necessary for the diagnostic task. This is especially important for elderly patients (they are more vulnerable to contrast media), and for patients undergoing multiple diagnostic imaging scans in a short period of time. Modern injectors typically provide a watch-dose function that helps clinicians validate the volume of contrast media entered in the device, based on the patient's physical parameters, pathology, and organ or tissue to be analyzed with the diagnostic imaging exam.

Use error with infusion devices, as well as with other medical devices, is a known source of incidents in healthcare [1, 9]. Whilst use error is defined as an act of omission or commission performed by the user that causes a device to respond unexpectedly [4], it is important to highlight that incident investigations usually reveal that use errors are due to latent design flaws in the device, rather than careless behavior or lack of training of clinicians (see [14] for a more in-depth discussion of use errors with medical devices). An example design flaw that can induce use errors is the data entry system of a device silently discarding decimal point key presses for certain range of values [5]. These types of flaws create traps for the users, that can be hard to avoid even for experienced users.

Advanced analysis tools can be used to facilitate early detection of latent software issues. Verification tools based on formal methods technologies, for example, can be used by manufacturers within the development process to analyze the source code of their products (e.g., see the analysis presented in [7]). Similar tools can also be used by healthcare providers, to improve procurement decisions and to develop training material (e.g., interactive simulations of a device) suitable to raise awareness about latent design flaws in user interface software. In this paper, we focus on the healthcare provider perspective.

Contribution. We describe how we used a prototyping and analysis toolkit to support the development of an interactive simulation of a contrast media injector used in a large academic hospital. The simulation was created to support training of clinicians, and helped to identify and raise awareness among clinicians of critical workflows that could induce accidental use errors that have safety implications.

2 RELATED WORK

Work on formal modelling and simulation of human-machine interaction with medical systems has gained momentum in these last few years, with various research groups focusing on the topic. For example, in [11], a library of patient and medical device models are developed to support the validation of medical cyber-physical

MCPS'18, April 2018, Porto, Portugal.

Figure 1: The injector system: workstation (on the left) and injector device (on the right).

systems. In [10], control theory is used as a basis to develop a simulation of a human operator interacting with a device providing both continuous control (e.g., joysticks) and discrete controls (e.g., buttons). Their simulation builds on the infrastructure of the PVSio-web toolkit we have used in the our work for creating the interactive prototype. However, these and other similar tools are designed to support the work of engineers. Their target is therefore different from our work, which aims to support multi-disciplinary analysis of human-centred medical systems, as well as create training simulations for clinicians.

3 THE CONTRAST MEDIA INJECTOR

The subject device of this work is a dual-syringe injector that performs injection of contrast and saline into the bloodstream of patients, commonly used in CT. Figure 1 shows the complete injection system that includes a workstation and an injector. Clinicians need to use both devices to carry out an injection.

The workstation allows clinicians to set up and manage personalized injection protocols for different patients, based on parameters such as patient weight, past scan procedures, current scan settings, diagnostic tasks, etc. The workstation seamlessly communicates with the injector, allowing clinicians to monitor progress of the injection directly from the workstation screen. The two syringes of the injector are displayed using color-coded pictures on the workstation screen (green for contrast, and blue for saline). Interaction with the workstation is carried out through a touchscreen display.

The injector has a front panel with a number of buttons and displays that allows clinicians to set up specific amounts of contrast media and saline. The body of the device includes an injector head (where syringes are inserted), plungers for controlling the volume of liquid in the syringes (plungers are automatically advanced and retracted when syringes are inserted and removed), and tubes/needles (used to connect syringes and the patient).

The front panel of the injector uses a number of displays and LEDs to provide feedback to the clinician about the state of the device. Two seven-segments displays labelled *VolumeA* and *VolumeB* report, depending on the device mode, either the volume of liquid in the syringe, or the position of the plunger. Each display has three significant digits, and can render only integer numbers. One LED light placed next to a lock symbol indicates whether the injection protocol has been set and locked from the workstation. Two large LED lights indicate whether an injection is running.

Various buttons on the front panel of the injector allow clinicians to operate the device. An *autoload* button, can be used to move the plungers and load the volume of saline and contrast configured by the clinician on the workstation. Two buttons, *plus (+)* and *minus(-)*, allow clinicians to increase/decrease the target volume of saline and contrast to be loaded in the syringes. Two buttons *FillA* and *FillB* activate the autoload sequence for the syringes. A *Manual load* button enables manual adjustment of the plunger position using the chevron keys available on the front panel of the injector. The speed at which the volume is increased/decreased depends on where the chevron keys are pressed – pressing next to the tip of the chevron key leads to quicker changes. A *Prime* button can be used to remove air-in-line. A *Check-Air* button is for checking air-in-line. An *Arm* button is used to make the system ready for an injection. An *Abort* button terminates an injection procedure and disarms the injection. A *Start/Hold* button allows to start the injection (when the injection is not started), and to pause the injection (this function is active when an injection is running).

4 THE INTERACTIVE SIMULATION

We developed an interactive simulation of the complete injection system using the PVSio-web [6] prototyping. A screenshot of the simulation is in Figure 2. A live version of the simulation is available online at http://www.pvsioweb.org/demos/stellantV2[1].

Developing the PVSio-web simulation involved three main steps:

(Step 1.) Build an executable specification of the behavior of the system in the PVS language. The specification of the injector was obtained by reverse engineering the real system. We used a combination of information from the user manual, observation of the device behavior through direct interaction with the real device, and the results of a field study we conducted to understand how expert users operate the device.

(Step 2.) Take a picture of the user interface of the real system that could be used as a basis to create the visual appearance of the interactive simulation.

[1]Google Chrome ver 62.x or newer version of the web browser is required to correctly execute the live demo.

Figure 2: Interactive simulation of the injector system.

(Step 3.) Use the PVSio-web library to create *interactive widgets* over the picture of the system. The library is implemented in JavaScript. Input widgets translate user actions over buttons into expressions of the executable PVS model to be evaluated to compute the system response. Output widgets mirror state attributes of the PVS model and resemble the look & feel of the real system in the corresponding state. For example, in the screenshot shown in Figure 2, output widgets are used to represent a system state where the syringes are plugged into the injector and spiked to a bag with saline and contrast liquids, and the injector has completed the process of loading the saline and contrast liquids in the syringes (the two seven-segments displays on the front panel of the injector indicate the volume of liquid loaded in the two syringes).

4.1 Executable specification

The executable PVS specification of the injection system was developed using the modelling approach described in [2]. It involves two main steps:

- Specify the system state as a PVS record type with relevant state attributes;
- Specify the behavior of the system as a set of transition functions that range over system states.

System state. The PVS record type for the injector system includes 57 state attributes: 38 attributes for the injector state; 14 for the workstation state; and 5 for the state of the syringes.

In the following, some aspects of the developed PVS model are illustrated. This illustration is not meant to be detailed or pedantic, as our aim is only to give the reader an understanding of what the PVS specification looks like. A detailed description of executable PVS specifications of medical devices can be found, e.g., in [3, 7].

A snapshot of the system state is in Listing 1. The syntax to specify a PVS record type is `[# a1: t1, ..., aN: tN #]`, where `a1 ... aN` are attribute identifiers, and `t1 ... tN` are attribute types. PVS provides an expressive specification language, which allowed us to select the most suitable attribute type from a wide range of pre-defined types, including basic types (bool, integer, reals, etc.), enumerations, arrays, sets, lists. Sub-types and user defined data-types can also be created.

```
state: TYPE = [# mode: Mode,
                 vol_saline: Volume,
                 vol_contrast: Volume,
                 lock_LED: LED,
                 ... #]
```

Listing 1: System state represented as a PVS record type

In the developed specification, we extensively used PVS subtypes as the PVS system automatically generates proof obligations for them — this is useful for checking well-formedness of the specification, including aspects such as correct use of types, coverage of conditions, and disjointness of conditions. An example sub-type is `Rate`, which is defined as a non-negative real number smaller than 200 (see Listing 2).

```
Rate: TYPE = { x: nonneg_real | x < 200 }
```

Listing 2: PVS sub-type for modelling infusion rate values

Transition functions. The developed PVS specification includes 33 transition functions: 26 for modelling the behavior of the injector, and 7 for modelling the workstation. The main focus of the simulation was the injector: this is the reason behind the small number of transition functions used for modelling the workstation. The size of the PVS specification is approx. 800 lines.

An example transition function created in the developed specification is `click_btn_manual` (see Listing 3), which models the effect of pressing the *Manual load* button available on the front panel of the injector to enable manual adjustment of the plungers position using the chevron keys of the injector. Other transition functions in the developed specification have a similar structure.

```
1 click_btn_manual(st: state): state =
2  st WITH [
3   mode := MANUAL,
4   vol_saline := plunger_saline(st),
5   vol_contrast := plunger_contrast(st),
6   vol_saline_confirmed := FALSE,
7   vol_contrast_confirmed := FALSE,
8   btn_manual_timeout := BTN_MANUAL_TIMEOUT
9  ]
```

Listing 3: Example transition function in PVS

The function has one argument `st` of type `state`, which represents the current system state. The function returns the next system state, which is constructed by modifying the value of relevant state attributes in the current state. The PVS language provides a syntax to write an override expression that can be used for this purpose (`WITH [ a1 := new_a1, ... aN := new_aN ]`). For this specific function, the override expression indicates that:

- The new device mode is `MANUAL` (see line 3 in Listing 3);
- Volumes of saline and contrast shown in the displays of the injector is calculated based on the position of the current position of the plungers (see lines 4-5 in Listing 3);
- Flags indicating whether the volumes have been confirmed are reset – these flags are used by a safety mechanism of the injector, which checks that clinicians have explicitly acknowledged the entered volumes before starting the injection (see lines 6-7 in Listing 3);
- A countdown timer (`btn_manual_timeout`) is set to a value given by the constant `BTN_MANUAL_TIMEOUT` – this timer is used to check inactivity of the user during manual

mode; it is part of a safety mechanism of the injector that guards against mis-configuration of the injector due to accidental button presses (see line 8 in Listing 3)

4.2 Interactive widgets

The interactive simulation includes the following PVSio-web widgets: 33 buttons, 9 displays, 5 LEDs, and 2 syringes.

Each button widget is seamlessly linked to a transition function in the PVS model: this is done through the APIs of the PVSio-web widget, which include a parameter for specifying the name of the transition function to be evaluated when a given user action is performed on the widget. Listing 4 shows an example use of the PVSio-web APIs for creating a widget for the *Manual load* button:

- `Button` is the widget constructor. The `new` operator is used to create a new object of type Button. The created widget is stored in a field `btn_manual` of a variable `sys`.
- The first argument of the constructor is a string defining the widget identifier. The PVSio-web toolkit uses this string as a basis to derive the name of the transition function in the PVS model to be linked to the widget – the full name of the transition function is constructed by concatenating the user action that activates the widget with the widget identifier. For example, when the user clicks on the button, the transition function that will be evaluated is `click_btn_manual`.
- The second argument is a structure defining the coordinates and size of the widget. This is necessary to create an interactive overlay area of the correct size for the image used as a basis for the visual appearance of the prototype, and to position the interactive area in the correct place (i.e., over the *Manual load* button in this case).
- The third argument provides information about the callback function to be invoked for refreshing the visual appearance of the prototype when the evaluation of the transition function associated with the button generates a new system state.

```
var sys = {};
sys.btn_manual = new Button("btn_manual", {
   top:792, left:210, width:38, height:38
 }, {
   callback: render
});
```

Listing 4: Example button widget

Each display and LED widget is seamlessly associated with a state attribute defined in the PVS specification. The creation of these widgets follows a pattern that is similar to that we have illustrated for button widgets. For example, Listing 5 shows how to create an LED widget:

- `LED` is the widget constructor;
- The first argument is the widget identifier;
- The second argument defines position and size of the widget;
- The third argument specifies the LED color.

```
sys.lock_LED = new LED("lock_LED", {
   top:916, left:221, width:13, height:13
 }, {
   color: "green"
});
```

Listing 5: Example display widget

The visual aspect of all widgets is periodically refreshed every time the PVS specification is evaluated. The evaluation of the specification occurs either when the user interacts with an input widget (e.g., presses a button), or periodically (if the device has internal timers that are ticking). A JavaScript function `render` contains the code for refreshing the widgets.

In its basic form, the `render` function simply parses the PVS state and invokes the `render` method of the widgets (see Listing 6). This function can be extended with custom JavaScript code necessary for mimicking specific aspects of the system that cannot be reproduced using just the widget. For example, in the developed simulation, we used custom code for rotating the injector upside down (this reflects the actual use of the injector, which clinicians need to rotate upside-down before starting the injection), and to create a simulation control panel that could be used to perform actions such as plugging the syringes into the injector head, spiking the syringes with the saline and contrast bags, and connecting the tubes to the syringes.

```
function render(err, event) {
  var res = stateParser.parse(event.data);
  if (res) {
    sys.btn_manual.render(res);
    sys.lock_LED.render(res);
    ...
  }
}
```

Listing 6: Render function

5 USING INTERACTIVE SIMULATION AS TRAINING MATERIAL

The development of the interactive simulation of the injection system was key to enable active engagement and constructive discussion in our multidisciplinary team of engineers and clinicians. This allowed the entire team to look closely and in a systematic manner into various design aspects of the system. This greatly helped engineers understand how clinicians use the system, and greatly helped the entire team to discuss and demonstrate various corner cases that could potentially have safety consequences in specific contexts. Under this perspective, the interactive simulator proved useful as training material for the system. Some of the identified corner cases are now discussed.

Risk of incorrect injection settings. The Volume displays available on the front panel of the injector normally report a value corresponding to the volume of liquid loaded in the syringes. However, in certain operating modes for the injector, the display values have a different meaning:

- When the syringes are not plugged into the injector head, the display values indicate the maximum value of volume that can be loaded in the syringes;
- When the syringes are connected but empty, the display values indicate the current position of the syringe plungers;
- When button *AutoFill* available on the front panel of the injector is pressed, the display values indicate the *target volume* of liquid that will be loaded in the syringes, according to the injection protocol defined on the workstation. This value is reset to 0 as soon as buttons *FillA* or *FillB* available on the front panel of the injector are pressed.

Information on the front panel of the injector is not always sufficient to discriminate these different cases.

Risk of undetected air-in-line. For patient safety, it is important to check that air bubbles are purged from syringes and tubing before the injection. For this reason, the arming phase of the injector is disabled if the *CheckAir* button available on the front panel of the injector has not been pressed.

However, this button is only a placeholder, i.e., a functionality provided by the device to remind the clinician to verify the absence of air (the injector does not have any sensor for detecting air-in-line). In other words, pushing the *CheckAir* button does not trigger any actual check from the device – the clinician needs to look into syringes and tubes and make sure there are no air bubbles. If air bubbles are present, the clinician can use the *Prime* button to remove the air. This was not clear from the manual.

Risk of misprogramming the injector. The injector provides two main modalities for filling syringes, and two modalities for priming: automatic and manual. In manual mode, clinicians use the chevron keys on the front panel of the injector to load the volume prescribed by the protocol and prime the syringes. In automatic mode, a single button press on *FillA* and *FillB* buttons on the front panel of the injector allows clinicians to load liquid in each syringe, and then a single button press on the *Prime* button primes the syringes. These two modalities can be interleaved, e.g., one can perform automatic fill of a syringe, and then manually prime the syringe. It is important to note that the volume of liquid loaded using automatic mode is *larger* than the volume prescribed by the protocol: +1 mL for the contrast, and +5 mL for the saline. The reason for this is that the automatic prime function pushes exactly 1 mL of contrast and 5 mL of saline out of the syringes. If clinicians interleave the two modalities and accidentally omit to check the volume on the injector display, there is a risk of injecting a volume of saline and contrast that is slightly different than the intended values.

Risk of misreading values. Another issue concerns the phase in which the injector needs to be connected to the patient to start the injection. In this phase, the injector needs to be rotated of 180 degrees. The rotation moves air up in the syringes – this is a safety precaution for preventing air being injected in the veins of the patient. However, the rotation causes the displays provided on the front panel of the injector to be upside-down. This is particularly unfortunate, because the device uses seven-segments displays and certain numbers can be accidentally mis-read when the display is upside-down (e.g., 51 can be misread as 12, see also [13] for a more detailed discussion of problems with using seven-segments displays in medical devices).

6 CONCLUSION

We presented our work on the development of an interactive simulation of a medical device in a hospital. The simulation facilitated the multidisciplinary work necessary to obtain results that have strong *impact* and immediate utility to different stakeholders. Engineers have the knowledge on the technology, in our case on modelling and rapid prototyping of user interfaces. Clinicians played a fundamental role in the identification of critical scenarios, as well as in the description of how the medical device is routinely used in the real-world. A possible use of the simulation tool is to enhance the proficiency of the clinical users of the injector, helping them to avoid possible traps, thus increasing patient safety.

As further work, we intend to use formal methods technologies to verify the developed formal model against use-related safety requirements such as *the movement of plungers always disables the check-air flag*. This is made possible by the theorem prover available in the PVS framework. Finally, we intend to explore the possibility of automatic code generation from the formal model, using a source code generator we are developing for the PVSio-web environment [8].

7 ACKNOWLEDGMENTS

Paolo Masci is funded by the ERDF (European Regional Development Fund) through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme within the project POCI-01-0145-FEDER-006961, and by National Funds through the Portuguese funding agency FCT (Fundação para a Ciência e a Tecnologia) as part of project UID/EEA/50014/2013.

REFERENCES

[1] Association for the Advancement of Medical Instrumentation (AAMI). 2015. AAMI/FDA summit on ventilation technology.

[2] Michael D. Harrison, José Creissac Campos, and Paolo Masci. 2015. Reusing models and properties in the analysis of similar interactive devices. *Innovations in Systems and Software Engineering* 11, 2 (01 Jun 2015), 95–111. https://doi.org/10.1007/s11334-013-0201-3

[3] Michael D Harrison, Paolo Masci, José Creissac Campos, and Paul Curzon. 2017. Verification of User Interface Software: the Example of Use-Related Safety Requirements and Programmable Medical Devices. *IEEE Transactions on Human-Machine Systems* 47, 6 (2017), 834–846.

[4] International Organization for Standardization. 2000. *ISO 14971: medical devices-application of risk management to medical devices*.

[5] Paolo Masci, Patrick Oladimeji, Paul Curzon, and Harold Thimbleby. 2017. Using PVSio-web to demonstrate Software Issues in Medical User Interfaces. In *Software Engineering in Health Care*, Michaela Huhn and Laurie Williams (Eds.). Springer International Publishing, Cham, 214–221.

[6] Paolo Masci, Patrick Oladimeji, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. 2015. PVSio-web 2.0: Joining PVS to HCI. In *Computer Aided Verification*. Springer International Publishing, Cham, 470–478.

[7] Paolo Masci, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. 2014. Formal Verification of Medical Device User Interfaces Using PVS. In *Fundamental Approaches to Software Engineering*, Stefania Gnesi and Arend Rensink (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 200–214.

[8] Gioacchino Mauro, Harold Thimbleby, Andrea Domenici, and Cinzia Bernardeschi. 2017. Extending a user interface prototyping tool with automatic MISRA C code generation. *arXiv preprint arXiv:1701.08468* (2017).

[9] Blackford Middleton, Meryl Bloomrosen, Mark A Dente, Bill Hashmat, Ross Koppel, J Marc Overhage, Thomas H Payne, S Trent Rosenbloom, Charlotte Weaver, and Jiajie Zhang. 2013. Enhancing patient safety and quality of care by improving the usability of electronic health record systems: recommendations from AMIA. *Journal of the American Medical Informatics Association* 20 (2013).

[10] Gerrit Niezen and Parisa Eslambolchilar. 2016. A Human Operator Model for Medical Device Interaction Using Behavior-Based Hybrid Automata. *IEEE Transactions on Human-Machine Systems* 46, 2 (2016), 291–302.

[11] Lenardo C Silva, Hyggo O Almeida, Angelo Perkusich, and Mirko Perkusich. 2015. A model-based approach to support validation of medical cyber-physical systems. *Sensors* 15, 11 (2015), 27625–27670.

[12] MA Ten Dam, JF Wetzels, et al. 2008. Toxicity of contrast media: an update. *Neth J Med* 66, 10 (2008), 416–22.

[13] Harold Thimbleby. 2013. Reasons to question seven segment displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1431–1440.

[14] M Thomas and H Thimbleby. 2018. Computer Bugs in Hospitals: A New Killer. (2018).