Towards a Formalization of System Requirements for an Integrated Clinical Environment

[Invited Paper]

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

Andrea Domenici
Department of Information Engineering
University of Pisa, Italy
andrea.domenici@iet.unipi.it

Paolo Masci
School of Electronic Engineering and Computer
Science, Queen Mary University of London, UK
p.m.masci@qmul.ac.uk

ABSTRACT

Interoperability of medical devices, and their interface to clinicians and patients, are critical issues for the safety and effectiveness of patient care. Ongoing efforts strive at establishing standards for integrated clinical environments, which may connect and co-ordinate several medical devices and interface them to patients, clinicians, and hospital information systems. In this paper, an approach to the formalization of system requirements for an integrated clinical environment is presented. The formalization relies on the higher-order logic language of the Prototype Verification System.

Categories and Subject Descriptors

D.2.1 [**SOFTWARE ENGINEERING**]: Requirements/ Specifications; D.2.4 [**SOFTWARE ENGINEERING**]: Software/Program Verification—*Formal methods*

General Terms

Reliability, Standardization

Keywords

Integrated clinical environment, PVS, Formalization

1. INTRODUCTION

Hospital care relies on a large number of medical devices, used both for monitoring patient conditions and for administering therapy. Typically, each device is self-standing, i.e., it operates independently of other devices and offers its own interface to clinicians, and it is up to the clinicians to supervise the set of available devices. For example, a patient may be administered a sedative with an infusion pump, while a clinician observes the patient's conditions through the data displayed by a pulse oxymeter. The clinician then changes the pump settings, or maybe stops or restarts delivery, according to his or her perception of patient conditions, based mainly on the oxymeter's readings. The safety of this humans-in-the-loop control process is enhanced by the availability of automatic alarms triggered by patient's parameters crossing certain thresholds.

In the above scenario, hazards arise from device malfunctions and human errors. The latter have many causes, including ill-designed interfaces [6] and incorrect information about the patient. The issue with interfaces is aggravated by the fact that devices with the same purpose may have different interfaces, depending on vendor or model.

Risks from such hazards may be reduced by *Integrated Clinical Environments* (ICE). An ICE is a computer system that mediates interactions between clinicians, medical devices, and information systems, providing services to enhance safety and effectiveness of patient care and possibly enforcing safety rules and standards.

Ongoing efforts strive at establishing standards for integrated clinical environments. Since ICE systems are inherently safety-critical, their requirements must be specified in the most rigorous and formal way. Formalization of such requirements, however, is difficult because the safety of an ICE is affected by a wide spectrum of factors, ranging from the operation of hardware and software to hospital procedures related, e.g., to patient identification or transmission of medical orders. An ICE is an example of a socio-technical system, i.e., one whose behavior is determined by complex interactions between people and machines.

In this paper, an approach to the formalization of system requirements for an integrated clinical environment is presented. This approach is illustrated by examples showing how requirements, expressed in natural language, can be translated into a formal logic language. This language is the one used in the Prototype Verification System, a theorem-

MOBIHEALTH 2015, October 14-16, London, Great Britain

DOI 10.4108/eai.14-10-2015.2261701

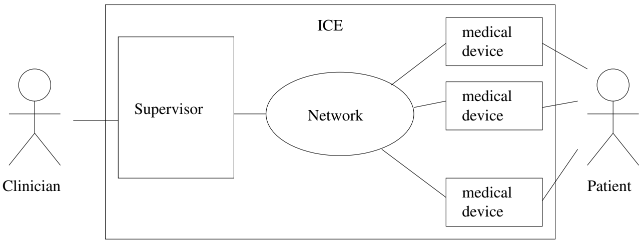

Figure 1: An Integrated Clinical Environment (adapted from [1]).

proving environment aimed at the formal specification and verification of complex systems. This paper is focused on requirements specification, leaving verification of requirements consistency and achievement of safety goals to further work.

2. ICE REQUIREMENTS

An ICE (Fig. 1) can be defined as an "*environment that combines interoperable heterogeneous medical devices and other equipment integrated to create a medical system for the care of a single high acuity patient*" [1].

An ICE is part of a wider environment, the hospital, and therefore its requirements span a wide range of issues, from administrative procedures (e.g., related to patient identification) to device operations. For example, a requirement related to patient data and identification might capture concerns about what information shall be available to clinicians to check that the right treatment is delivered to right patient. This information might include, for example, patient ID, demographical data (name, age, etc.), location, etc.. A requirement related to the operation of a specific device, namely an infusion pump for *patient-controlled analgesia* (PCA), might capture information about safety-interlock mechanisms that can be used to ensure the safe delivery of a treatment in the case of device malfunction. For example, to prevent overdose with a PCA pump, a safety requirement could be defined to ensure that additional doses of anesthetic (*bolus*) shall be inhibited in the presence of error conditions.

Other issues that ICE requirements must take into account include clinician authorizations and authentication, alarms and warnings, interconnection between ICE and devices, human-machine interaction, and more. Expressing the requirements in a formal language will result in a large and complex conceptual model that can be checked for *consistency* (no requirement contradicts another one).

Any effort towards the formalization of ICE requirements will help developers to discover and address the typical problems of informal systems specification, such as ambiguities inherent in the natural language in which the requirements are originally expressed, and implicit assumption. In this work, we cover this aspect of the formalization of safety requirements, and present an example in the next sections.

The formalization of the safety requirements leads to generating a *safety reference model* of the system that captures information about what needs to be verified in a system design to ensure safety of operation. To carry out this verification, the reference model is populated with details about the behavior of the system under analysis, e.g., following the approach illustrated in [4, 3]. The instantiated model defines specific constraints that shall hold in the system to prove that the real system meets the requirements formalized in the safety reference model.

3. FORMALIZATION

The typed higher-order logic of the *Prototype Verification System* (PVS) [8] has been used for the formal specification of many kinds of systems, including medical devices [3, 7, 6]. In the PVS, a system is modeled by a *theory*, i.e., a set of statements in the PVS logic language describing the system by means of variable, constant, and function definitions, and of axioms and theorems about them. Properties of the systems can then be proved with respect to the theory, using the interactive PVS theorem prover.

A PVS theory can refer to other theories, thus enabling a modular, hierarchical composition of complex systems from subsystems. Another feature that makes the PVS language attractive for systems specification is its flexible and expressive type system. With this type system it is possible to specify all the datatypes available to programming languages, but also to define types that abstract from any unnecessary details: It is then possible to state that the members of a given type satisfy some properties, without any reference to the implementation of the members. Further, subtypes can be specified by stating the properties which characterize the subtype members.

Thanks to these features, the PVS language is well-suited to specifying such a complex system as an ICE. As previously discussed, ICE requirements must span different domains and different levels of abstraction and generality: For example, there are requirements affecting all devices irrespective of their kind, and requirements affecting specific kinds, such as infusion pumps, X-ray machines, or respiration monitors. The modular composability of PVS theories and the flexibility of the type system make it possible to structure the overall specification in a set of interrelated theories, each devoted to a specific (sub)domain or level of abstraction. Such a specification would be easily maintainable, in case of changes of regulations or introduction of new equipment or therapies.

3.1 Domain identification

A fundamental step in drafting a specification is *domain identification*, i.e., recognizing and representing the fundamental concepts in the application domain. For example, consider the requirement on patient identification mentioned in Section 2 above. Some significant words in that text are *patient* and *information.* Such words as *patient ID*, *location*, or *age*, denote items of information pertaining to a patient. Many other requirements will mention patients and patient-related concepts, such as blood pressure or pulse rate. The concepts related to patient identification can be grouped in a theory as follows:

```
patients_th: THEORY
begin
   patient: TYPE+
    patient_ID: TYPE+
    patient_location: TYPE+
    id(p: patient): patient_ID
    location(p: patient): patient_location
    ...
end patients_th
```

The above fragment says that `patient`, `patient_id`, and `patient_location` are nonempty types, and that functions `id` and `location` return patient `p`'s identifier and location, respectively. Similar type and function definitions for other attributes are not shown.

We may note that each attribute could be implemented in different ways, such as integer numbers, strings, or records, but such details are inessential since they are never addressed in the ICE requirements, and have been consequently abstracted out in the terse type declarations.

Another conceptual domain concerns medical devices. From the documents on the ICE requirements, and from general knowledge, a medical device can be seen abstractly as a system that can "read" the values of physical quantities, receive commands, and execute actions. A device has a state defined by the values of a set of parameters. How the values of parameters and of physical quantities are represented in a given device is seldom, if ever, addressed in the ICE documentation. Actually, value representation is often an irrelevant detail, unless human interaction or control algorithms are involved. So, a theory of medical devices should depend on a theory of parameters embodying implicit knowledge on physical values:

```
parameters_th: THEORY
    parameter: TYPE+
    dimension: TYPE+
    unit: TYPE+
    pulse_rate: parameter
    blood_pressure: parameter
    duration:      dimension
    mass:          dimension
    hr:        unit  % hours
    mg:        unit  % milligrams
    g_per_l:   unit  % grams per liter
    parm_dimension(p: parameter): dimension
    unit_dimension(u: unit): dimension
    value: TYPE = [# magn: number, units: unit #]
    parm_value(p: parameter): value
    ...
end parameters_th
```

In the above theory, the abstract types for parameters, dimensions, and units are defined, then a few sample parameters (pulse rate and blood pressure) are introduced, along with physical dimensions and units. A value can then be modeled as a record with a `magn` (magnitude) and a `units` field. Functions return a parameter's value and dimensions.

A theory for devices could be written in the following fashion:

```
devices_th: THEORY
    IMPORTING parameters_th
    device: TYPE+
    state:  TYPE = setof[parameter]
    command: TYPE+
    display:  TYPE = setof[parameter]
    commands: TYPE = setof[command]
    panel: TYPE = [# displ: display,
                       cmds: commands #]
    st(d: device): state
    pnl(d: device): panel
    ...
end devices_th
```

This theory says that a device state is a set of parameters, a device front panel has a set of displayed parameters and a set of commands, and functions (`st` and `pnl`) access a device's state and panel. Different types of devices, such as infusion pumps, are modeled as subtypes of `device` in the respective theories, which introduce device-specific commands, parameter, and functions describing state transitions:

```
infusion_pumps_th: THEORY
    IMPORTING devices_th
    infusion_pump: TYPE+ FROM device
    pause_cmd: command
    incr:      command % increment value
                       % of parm currently edited
    decr:      command % decrement value
                       % of parm currently edited
    bolus:     command % deliver a bolus
    pwr:       command % power on/off
    ...
end infusion_pumps_th
```

3.2 Requirements formalization

Let us now consider the following requirements[1] for the ICE system:

- *Remote control by the ICE system shall not be locally overridden. Commands entered from the front panel that change pump operation shall be disabled, except for the* pause *command.*
- *The ICE system shall require confirmation when clearing the pump settings and resetting the pump.*

Even if the above requirements are specific to a particular kind of device, they concern the interaction between a device and the ICE system, and are likely to apply to other device kinds. Therefore they may be dealt with in a theory of interactions, where such concepts as *remote* or *local* control, *enabling* or *disabling*, and *issuing* or *confirming* commands are modeled:

```
interactions_th: THEORY
```

[1] The requirements were originally formulated for infusion pumps in [2], and here we adapted the requirement to the ICE system.

```
    IMPORTING devices_th
    control:  TYPE = remote, local
    controlled_under(d: device): control
           % is d controlled locally or remotely?
    issued(c: command):          bool
           % has command c been issued?
    issued_under(c: command):    control
           % has c been issued locally?
    enabled(c: command):         bool
    changer(c: command):         bool
           % does c change a parameter or mode?
    confirmation_requested(c: command): bool
    confirmed(c: command):              bool
    accepted(c: command):               bool
    ...
end interactions_th
```

With the theories hinted at above, it is possible to specify requirements in the form of axioms. For example, the requirement that inhibits local control (except for the *pause* command) when a pump is under ICE control would read as follows:

```
infusion_pump_reqmts_th: THEORY
    IMPORTING devices_th, interactions_th,
         infusion_pumps_th

remote_disable_local: AXIOM
  forall (p: infusion_pump):
      controlled_under(p) = remote
      => forall (c: command):
          cmds(pnl(p))(c) and changer(c)
              and c /= pause_cmd
          => not enabled(c) and enabled(pause_cmd)
    ...
end infusion_pump_reqmts_th
```

In the above snippet, the expression `cmds(pnl(p))(c)` means that `c` belongs to the set of commands (`cmds(...)`) accepted by the front panel (`pnl(...)`) of `p`. This (perhaps too) terse but precise expression stems from PVS's way of interpreting sets as logical predicates, i.e., defining a set implicitly defines a predicate characterizing its members. The above axiom then reads as "*for all pumps* `p`*, if* `p` *is remotely controlled, then all its commands which change parameter values or operation mode are disabled, except for the* pause *command*".

It may be observed that compliance to the above requirements would avoid adverse situations such as reported in [5], where simulation showed that an infusion pump could receive a pause command from the ICE while infusion parameters were being manually edited, resulting in over- or underdosing.

The examples in this section illustrate how a single rigorous formalism such as the PVS language is expressive enough to model system requirements from different perspectives, from administrative procedures to technical requirements on device operation and co-ordination.

4. CONCLUSIONS

This paper advocates the use of the PVS specification language to formalize system requirements for integrated clinical environments. A modular, hierarchical approach to the construction of a complex theory modeling ICEs and expressing their requirements has been illustrated by means of simple examples. Proof-of-concept PVS theories are currently being developed by the authors according to this approach.

Acknowledgements

Paolo Masci was funded by the CHI+MED project: Multidisciplinary Computer Human Interaction Research for the design and safe use of interactive medical devices project, UK EPSRC Grant Number EP/G059063/1.

5. REFERENCES

[1] 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.* IEC, International Electrotechnical Commission, 2008.

[2] The Generic Patient Controlled Analgesia Pump Hazard Analysis and Safety Requirements. `http://rtg.cis.upenn.edu/gip.php3`, retrieved 7/20/2015.

[3] 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.

[4] P. Masci, A. Ayoub, P. Curzon, M. D. Harrison, I. Lee, and H. Thimbleby. Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In *EICS2013, 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems.* ACM Digital Library", 2013.

[5] P. Masci, P. Mallozzi, F. L. De Angelis, G. Di Marzo Serugendo, and P. Curzon. Using PVSio-web and SAPERE for rapid prototyping of user interfaces in Integrated Clinical Environments. In *submitted to Verisure2015, Workshop on Verification and Assurance, co-located with CAV2015*, 2015.

[6] 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. *Innov. Syst. Softw. Eng.*, 11(2):73–93, June 2015.

[7] P. Masci, Y. Zhang, P. Jones, P. Curzon, and H. Thimbleby. Formal verification of medical device user interfaces using PVS. In S. Gnesi and A. Rensink, editors, *Fundamental Approaches to Software Engineering*, volume 8411 of *Lecture Notes in Computer Science*, pages 200–214. Springer Berlin Heidelberg, 2014.

[8] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Automated Deduction — CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer Berlin Heidelberg, 1992.