Modeling Communication Network Requirements for an Integrated Clinical Environment in the Prototype Verification System[*]

Cinzia Bernardeschi   Andrea Domenici[†]   Paolo Masci[‡]

Abstract

Health care practices increasingly rely on complex technological infrastructure, and new approaches to the integration of information and communication technology in those practices lead to the development of such concepts as *integrated clinical environments* and *smart intensive care units*. These concepts refer to hospital settings where therapy relies heavily on inter-operating medical devices, supervised by clinicians assisted by advanced monitoring and co-ordinating software. In order to ensure safety and effectiveness of patient care, it is necessary to specify the requirements of such socio-technical systems in the most rigorous and precise way. This paper presents an approach to the formalization of system requirements for communication networks deployed in integrated clinical environment, based on the higher-order logic language of a theorem-proving environment, the Prototype Verification System.

1 Introduction

Modern clinical practices involve a large number of medical devices for disparate functions, such as ventilators, infusion pumps, laser scalpels, or surgical robots, and a vast array of monitoring and diagnosis devices, from pulse oxymeters to imaging equipment. Until now, each device usually operates independently of other devices and is supervised by clinicians, but technological innovations foster new ways of using medical equipment, which rely on the interconnection of different devices under computer-assisted human supervision. Computer-assisted

[*]Postprint. Published in: 2016 IEEE Symposium on Computers and Communication (ISCC). DOI: 10.1109/ISCC.2016.7543728. **©2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.**

[†]C. Bernardeschi and A. Domenici are with the Department of Information Engineering, University of Pisa, Italy. E-mail: {cinzia.bernardeschi,andrea.domenici}@ing.unipi.it

[‡]P. Masci is with HASLab/INESC TEC and Universidade do Minho, Campus de Gualtar, Portugal. E-mail: paolo.masci@inesctec.pt

supervision affords many benefits, including the automatization of routine procedures and, above all, the implementation of safety mechanisms. Further, computerized management of inter-operating devices can be integrated with information systems, both local and wide-area, to access and maintain information on patients and therapies, both at individual and statistical level.

This trend in clinical practices has led clinicians and medical equipment producers to formulate such concepts as *Integrated Clinical Environment* (ICE) [5] and *smart ICU* (Intensive Care Unit) [6]. These clinical settings are safety-critical socio-technical systems whose behavior is determined by complex interactions between people and machines, needing precise and rigorous requirements.

In these types of environments, interoperability [13] is a crucial concern. Interoperability allows a seamless flow of information between many disparate devices, so that different equipment from different vendors can communicate over different networks. Existing interoperability standards written for generic applications must be constrained by imposing the additional requirements of clinical applications. Such clinical-oriented standards will make it easier to connect future biomedical devices and clinical information system by formulating a set of interoperability requirements [8, 15].

A central component of an ICE (Fig. 1) is the *communication network*. In a typical setting, the network enables communication between many different devices in the hospital area and the ICE supervisor, and possibly among devices. Sensors for physiological parameters and therapy-delivering devices may be carried on a patient's body, and signals to and from such equipment are exchanged with the supervisor and maybe displayed on smartphones. Other equipment may be operated by clinicians, such as ultrasound scanners, or bar-code readers used to identify patients and drugs. Further, the local network is connected to a wider-area network in order to access various information systems, such as patient databases.

The main contribution of this paper is a theory in a higher-order logic language defining requirements for the communication network of an ICE. This logic specification is a formal reference model for the ICE realization, and for verification of its properties, in particular of its safety requirements.

An extensive requirements specification for an ICE communication network is out of the scope of this paper, and only the basic ideas illustrated by short excerpts are introduced. The specification refers to a network whose nodes are medical devices and computers, all equipped with wireless network interfaces. In general, all nodes are mobile and subject to the well known issues of wireless networks, such as sporadic loss of connectivity. We may observe that wired devices, if they can be unplugged and moved, are logically equivalent to wireless ones, otherwise they can be seen as a degenerate case of wireless nodes.

To the authors' knowledge, no other work on the application of formal logic to medical systems has appeared in the literature so far. Among works related to the present paper, the use of the Prolog language to formalize a portion of the U.S. Health Insurance Portability and Accountability Act (HIPAA) [9] can be cited. Medical processes have been modeled with Guarded High-level

Message Sequence Charts (g-HMCS) [10], and a knowledge-based distributed system, K4CARE, is used to support the needs of senior individuals requiring a personalized home care assistance [3].

The paper is structured as follows: Section 2 introduces the specification language of PVS, Section 3 describes the general methodology to formalize the ICE requirements, Section 4 briefly discusses the high-level communication requirements for an ICE, Section 5 presents the PVS theories for the ICE communication network, in Section 6 the use of these theories for implementation and verification purposes is discussed, and Section 7 concludes the paper.

2 The PVS specification language

The typed higher-order logic of the *Prototype Verification System* (PVS) has been used for the formal specification of medical devices [7, 12, 11], among several other kinds of systems. In the PVS, a system is modeled by a *theory*, i.e., a set of statements describing the system by means of variable, constant, and function definitions, and of axioms and theorems about them. In particular, the language makes it possible for functions to return functions and pass functions as function arguments. Properties of the system, expressed as theorems, 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. With the PVS type system it is possible to use 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.

A PVS extension, the PVSio package [14], adds a prototyping capability to the PVS environment. This is possible because PVS functions are total and can be effectively computed when applied to *ground*, i.e., fully instantiated, arguments. The PVSio package provides a ground evaluator and a library of functions with side effects, e.g., reading inputs and producing outputs, thus allowing a PVS theory to be interpreted and executed, much in the style of logic programming languages.

The PVS syntax is rather complex, and some details will be given in the text. Only few rules need to be given in advance: (i) Function types are defined by signatures, i.e., "$[\mathit{domain} \to \mathit{range}]$", where the range may be another function type; (ii) function definitions and applications may be written in Curried form, i.e., $f(x)(y)$ is syntactically equivalent to $f(x, y)$; (iii) subtypes can be defined by a clause of the form *subtype: TYPE FROM type*, or by set comprehension, e.g., $\{n : \mathit{nat} \mid \mathit{odd}(n)\}$; (iv) formulas labeled as *AXIOM* are taken by the prover as proved, while formulas labeled as *THEOREM* or *LEMMA* must be proved.

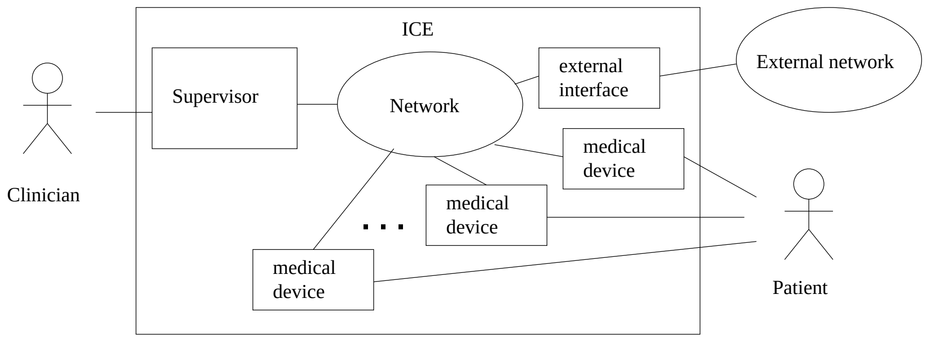

Figure 1: An Integrated Clinical Environment (adapted from [5]).

3 Formalizing ICE Requirements

The requirements for an ICE span a wide range of issues, from administrative procedures to device operations. For example, procedures to manage patient identity data may reduce the risk of delivering a treatment to the wrong patient, while compliance with safety standards may prevent device failures. The requirements must take into account such issues as clinician authorizations and authentication, alarms and warnings, interconnection between ICE and devices, human-machine interaction, and more. Expressing the requirements in a formal language results in a large and complex conceptual model that can be checked for *consistency*.

The specification process can be structured in two activities, *domain identification* and *requirements formalization*, discussed in the rest of this section.

3.1 Domain identification

Domain identification means recognizing and representing the fundamental concepts in the application domain. The application domain of an ICE is composed of several (sub)domains, each structured in levels of abstraction.

Using a higher-order logic as a specification language, a domain is modeled by a theory defining types representing domain concepts, and functions and axioms representing relationships among concepts. For example, a theory on patient identification may define the types *patient* and *patient_identifier*; the fact that a patient has an identifier may be expressed by a function *id* returning the identifier of a patient, or possibly a set of identifiers, if allowed.

In an earlier work [1], examples of formal requirements were given for the ICE subdomains of *patient identification*, *physical parameters* (such as temperature or pressure), and *devices*.

For example, consider the *devices* subdomain. Its theory depends on the *physical parameters* theory, since the state of a device is defined by a set of controlled or observed parameters. A device has a *display*, which is also represented as a set of displayed parameters. The *front panel*, i.e., its external interface, is represented abstractly by its display and the set of commands it accepts. The generic theory for the domain of medical devices includes the definitions for the above concepts, and functions to access the parts or parameters of a device:

```
devices_th: THEORY BEGIN
    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
```

Note that the *TYPE+* keyword asserts that type *device* is nonempty.

At a lower abstraction level, 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. The following *infusion_pumps* theory is an example of a device-specific theory.

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

3.2 Requirements formalization

After domain identification, the requirements are specified as a set of axioms, grouped in theories according to their main subject. It should be noted that a requirement may involve concepts from different domains.

For example, consider an infusion pump that may be operated remotely (through the ICE supervisor) or locally (manually). One of its safety requirement forbids all local actions while the pump is under remote control, except for the *pause* action, so that a clinician may stop drug delivery to stall a perceived overdose situation. In order to express this requirement, some concepts from the *interaction* theory are needed, such as *locally* or *remotely controlled* devices, *locally* or *remotely issued* commands, command *instance*, and so on:

```
interactions_th: THEORY BEGIN
    IMPORTING devices_th
    control:  TYPE = {remote, local}
    cmd_id: TYPE = posnat
    cmd_instance: TYPE =
     [# cmd: command, inst: cmd_instance #]
    controlled_under(d: device):     control
    % has cmd_instance i been issued?
    issued(i: cmd_instance):            bool
    % issued locally or remotely?
    issued_under(i: cmd_instance):   control
    enabled(c: command):                bool
    % does c change a parameter or mode?
    changer(c: command):                bool
    ...
END interactions_th
```

The requirement can then be expressed as Axiom *remote_disables_local*:

```
infusion_pump_reqmts_th: THEORY BEGIN
  IMPORTING interactions_th,
            infusion_pumps_th
remote_disables_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 fragment, `pnl(p)` is the panel of device `p`, and `cmds(pnl(p))` is the set of commands accepted by `p`. In PVS, a set is interpreted as a predicate that is true only for each set member, so the expression `cmds(pnl(p))(c)` means that `c` belongs to the set of commands (`cmds(...)`) accepted by `p`. The axiom then means "*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*".

4 Communication-related ICE Requirements

Several system-level ICE requirements induce other requirements on the underlying network. Such requirements concern information integrity and availability, and system resilience against malfunctions or improper operation. The basic fact that an ICE is a set of interconnected devices implies that the network must be dependable. Also specific ICE requirements depend on the availability and correctness of the network. For example, data on patient conditions must be available also when the patient is moved to another room. Another important ICE requirement is that the supervisor must be notified of device disconnections.

A *communication* theory defines the high-level concepts of communications between devices and supervisor, such as destination device of a command instance, or issue and reception time of a command instance.

```
communication_th: THEORY BEGIN
IMPORTING ...
  connected(d: device): bool
  sent_to(i: cmd_instance,
          d: device, t: time): bool
  received_by(i: cmd_instance,
          d: device, t: time): bool
  ...
END communication_th
```

System-level requirements on communication can then be expressed in the following theory:

```
communication_reqmts_th: THEORY BEGIN
IMPORTING communication_th
  ...
  cmd_delivery: AXIOM
    forall (i: cmd_instance,
              d: device, t: time):
         connected(d)
           and sent_to(i, d, t)
      => exists (tr: time):
          received_by(i, d, tr)
            and t < tr
  once: AXIOM
    forall (i: cmd_instance,
              d: device, t, t1: time):
         received_by(i, d, t)
           and received_by(i, d, t1)
      => t1 = t
  disconnect_notification: AXIOM
    forall (d: device):
        not connected(d)
      => disconnect_alarm(d)
END communication_reqmts_th
```

The first two axioms above concern guarantee of delivery and integrity of communication: *cmd_delivery* states that every command instance i sent to a connected device d at time t will be received by d at a later time t_1, while *once* states that any command instance i received by a device d is received only once. Suppose, for example, that the ICE supervisor resets a life-supporting device so that it can be reprogrammed and then restarted. If the data packet carrying the reset command is duplicated and resent by a node, the spurious copy could reach the device after restart and reset it, blocking its life-supporting operation. The *once* axiom forbids this kind of hazard.

The third axiom requires that a disconnection notification related to device d be produced when d is disconnected.

5 The ICE Communication Network

The communication network must be highly reliable and available, but it must also be flexible and easy to use. In particular, it must enable device mobility, which allows moving patients and equipment.

The rest of this section sketches a network specification that is general enough to allow for many different choices of hardware and communication protocols.

Nodes are the communication interfaces of the medical devices and of the supervisor, or routing elements. Each device is mapped to one node.

```
nodes_th: THEORY BEGIN
  IMPORTING devices_th
  network_size: posnat
  node_id: TYPE = below(network_size)
  router_id: TYPE = finite_set[node_ids]
  device_id: TYPE = finite_set[node_ids]
  supervisor: node_id
  dev2node_f: TYPE = [device -> node_id]
  ...
END nodes_th
```

The network structure is represented as a directed graph, using the *digraphs* theory provided by the NASA PVS libraries [2], which is parametric with respect to the type of graph nodes. In the *network_graph* theory below, *topology* is the type of functions from node identifiers to finite sets of node identifiers, meant to represent physically connected nodes or the set of immediate neighbors of each node.

Type *network_graph* is the set of directed graphs representing the network connectivity. The graph has no self-edge.

```
network_graph_th: THEORY BEGIN
  IMPORTING nodes_th, digraphs[node_ids]
  topology: TYPE =
    [node_ids -> finite_set[node_ids]]
  network_graph: TYPE =
    {g: digraph[node_ids] |
      (FORALL (n: node_ids):
        vert(g)(n))
        and (forall (n, m: node_ids):
          edges(g)((n, m)) => (n /= m))}
  ...
 END network_graph_th
```

The *packet* theory defines packets as records with fields for timestamp, originating (*source*) node, sender and destination node, and payload:

```
packet_th: THEORY BEGIN
IMPORTING nodes_th, time_th
  packet: TYPE = [#
    timestamp: time,
    source_addr: node_id,
    sender_addr: node_id,
    destination_addr: finite_set[node_id],
    payload: finite_sequence[int] #]
END packet_th
```

The *network* theory defines the network state as a record with fields for a global clock, functions mapping each node to its receive buffer and to its physical location, and a log recording the sequence of packets processed by each node. Communication primitives, such as *forward*, handle packets and update the network state accordingly.

```
network_th: THEORY BEGIN
  IMPORTING time_th, receive_buffer_th,
    location_th
  network_state: TYPE = [#
    global_clock: time,
    net_rcv_buf: [node_id -> rcv_buf],
    net_location: [node_id -> location] #]
    log: [node_id ->
      finite_sequence[packet]] #]

  forward(p: packet)
    (forwarder: node_id)
    (net: network_state, g: network_graph):
        network_state = ...
  ...
END network_th
```

A network protocol is an algorithm executed by each node to propagate application-specific information. The algorithm updates the network state and depends on the network structure, as shown in the following theory:

```
protocol_th: THEORY BEGIN
  IMPORTING network_graph_th, network_th
  protocol: TYPE =
    [network_graph, node_id ->
      [network_state -> network_state]]
END network_th
```

5.1 Requirements

The requirements of the communication network derive from the higher-level ICE requirements, i.e., they express the properties that any network implementation must exhibit in order to be used in an ICE. Consider, for example, the *once* axiom in Theory *communication_reqmts* (Sec. 4). In terms of network-specific concepts, the absence of packet duplication can be expressed as "*in any network state, for all packets p and node n, the set of packets equal to p transmitted by n is either empty or a singleton*":

```
comm_netwk_reqmts_th: THEORY BEGIN
  IMPORTING ...
  no_duplication: AXIOM
    FORALL (net: network_state, p: packet):
      FORALL (n: node_id):
        empty?(transmitted(p, log(net), n) OR
        singleton?(transmitted(p, log(net), n)
  ...
END comm_netwk_reqmts_th
```

Other types of requirements concern the interaction between devices and supervisor at a higher abstraction level. For example, in order to express the *disconnect_notification* system requirement (Sec. 4) as a network requirement, the following declarations are included in the *network* theory:

```
  alarm_cause: TYPE = {disconnection, ...}
  severity_t: TYPE = {low, medium, high}
  disconnected(d: device): bool
  alarm(d: device,
    c: alarm_cause, s:severity): bool
  severity(d: device, c: alarm_cause):
    severity_t
```

The *alarm* function is true if device d is in the condition described by c, with severity level s. The latter is obtained by function *severity*, whose value depends both on the affected device and on the cause of the alarm.

The following function, from the *network* theory, checks if node n is disconnected, by analyzing the network graph in the current state. The actual definition of the function will be specified by axioms.

```
  node_disconn(s: network_state,
    g: network_graph, n: node_ids):
      bool
```

The following axioms from the *comm_netwk_reqmts* theory specify the above stated *disconnect_notification* requirement:

```
  dev_disconn: AXIOM
    FORALL (d: device):
      FORALL (s: network_state):
        FORALL (g: network_graph):
         node_disconn(s, g, dev2node(d))
         => disconnected(d)

  disconn_alarm: AXIOM
    FORALL (d: device):
      disconnected(d) =>
        alarm(d, disconnection,
          severity(d, disconnection))
```

The above discussion shows that PVS language is well suited to specifying such a complex system as an ICE. 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.

6 Verification

An advantage of the approach used in this work is its ability to describe a system at different levels of abstraction. A number of different versions of the theories can be developed for each component, each one at a different level of detail. The most abstract theories provide the declarations of the basic set of interface functions (i.e., functions meant to be used in other theories) and types. More detailed theories can be derived from the abstract definitions by specifying the definition of the functions and by extending types. If different versions of a theory provide the same declarations for interface functions and types, they are interchangeable, hence, when building the model, the minimal set of details needed for analysis can be used, by importing the appropriate version of the theory.

For example, consider the high-level definition of the *protocol* type in Section 5.1 above. An instance of that type is a function defining the sequence of actions performed by a generic node. Actions may depend on the content of received packets (e.g., the sender address of a received packet) and on the state of the node (e.g., the value of data gathered from the sensing equipment).

As an example of a concrete protocol specification, let us consider the *reverse path forwarding* protocol [4], a one-to-many algorithm designed to deliver packets to all nodes in the network. A simple version of this algorithm behaves as follows: A node n accepts a packet received from node p only if n believes that p is the best next hop on the path to the base station, as specified in the routing table. This protocol could be used by the supervisor to query all devices in order to check if they are all connected.

The following *rpf* theory contains the definition of the reverse path forwarding protocol.

```
rpf_th: THEORY BEGIN
IMPORTING ...
  ...
  rpf(g:network_graph, n: node_id)
      (net: network_state): network_state =
    IF empty?(net_rcv_buf(net)(n))
    THEN idle(n)(net, g, rt)
    ELSE
     LET rcvd_p =
           getpacket(net_rcv_buf(net)(n)),
       source_addr = source_addr(rcvd_p),
       sender_addr = sender_addr(rcvd_p),
       next_hop = next_hop(n, bstn)(g, rt)
     IN  IF sender_addr = next_hop
         THEN forward(rcvd_p)(n)(net,g,rt)
         ELSE drop(rcvd_p)(n)(net, g, rt)
         ENDIF
    ENDIF
END rpf_th
```

Functions *idle*, *forward*, and *drop* are low level single-hop communication primitives modeling the "no action" behavior, packet forwarding, and packet dropping, respectively, from the *network* theory. Function *next_hop* is declared in a *routing_table* theory (not shown).

Verification is accomplished by producing a formal model of the implementation to be verified. Then a verification theory can be built, where the *axioms* from the requirements theories are expressed as *theorems* on the implementation model, as shown in the following schema, where the *no_duplication* requirement is taken as an example:

```
implementation_th: THEORY BEGIN
IMPORTING protocol_th, rpf_th ...
  init_state: network_state
    = (# ... #)
  transmitted(p: packet,
      l: [node_id -> finite_sequence[packet]],
      n: node_id): bool =
    % a predicate depending on
    % the rpf protocol
END implementation_th

verification_th: THEORY BEGIN
IMPORTING implementation_th ...
  no_duplication_thm: THEOREM
   FORALL (net: network_state, p: packet):
    FORALL (n: node_id):
     empty?((transmitted(p, log(net)), n) OR
     singleton?((transmitted(p, log(net)), n)
END verification_th
```

The implementation theory contains assumptions on the implemented network, including structural and behavioral properties, the definition of, or assumption on, the initial state, and how a packet is transmitted through the *rpf* protocol. In the verification theory, it is then possible to prove that the chosen protocol satisfies the above requirement.

7 Conclusions

The development of ICEs poses many challenges, as they must face the complexity of socio-technical systems and satisfy strict safety requirements. In particular, rigorous requirements specification is an essential basis for development.

In this paper, an approach to the formalization of system requirements for a core subsystem of integrated clinical environments, the communication network, is proposed, elaborating on the guidelines presented in previous work. The fundamental feature of this approach is the use of a higher-order logic language, provided by the PVS theorem-proving environment.

The approach has been illustrated by providing and discussing short excerpts of logical theories describing concepts of, and requirements on, different aspects of communication networks for clinical environments, at different abstraction levels. The examples are meant to support the thesis that logic-based formal specification is a useful tool in the development of complex, safety-critical systems, including integrated clinical environments, as it enables developers to produce modular, detailed, and flexible specifications, which can then be used for formal verification.

Acknowledgments

This work was partially supported by Project “Analisi di dati sensoriali: dai sensori tradizionali ai sensori sociali (2015–16)” funded by the University of Pisa. Paolo Masci is financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF) within Project “NORTE-01-0145-FEDER-000016”.

References

[1] Cinzia Bernardeschi, Andrea Domenici, and Paolo Masci. Towards a formalization of system requirements for an integrated clinical environment. In *5th EAI International Conference on Wireless Mobile Communication and Healthcare (MOBIHEALTH 2015)*. Springer, 2015. In press.

[2] R. Butler and J. Sjogren. A PVS Graph Theory Library. Nasa technical memorandum 1998-206923, NASA Langley Research Center, Hampton, Virginia, 1998.

[3] F. Campana, A. Moreno, D. Riaño, and L. Z. Varga. K4Care: Knowledge-based homecare e-services for an ageing europe. In *Agent Technology and e-Health*, pages 95–115. 2008.

[4] Yogen K. Dalal and Robert M. Metcalfe. Reverse path forwarding of broadcast packets. *Commun. ACM*, 21(12):1040–1048, December 1978.

[5] 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.

[6] Neil A. Halpern. Advanced informatics in the intensive care unit: Possibilities and challenges., February 2014.

[7] 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.

[8] Jeff Kabachinski. What is Health Level 7? *Biomedical Instrumentation & Technology*, 40(5):375–379, 2006.

[9] P. E. Lam, J. C. Mitchell, and S. Sundaram. A formalization of HIPAA for a medical messaging system. In *Trust, Privacy and Security in Digital Business: 6th International Conference*, pages 73–85. September 2009.

[10] B. Lambeau, C. Damas, and A. van Lamsweerde. Process execution and enactment in medical environments. In *6th Workshop on Software Engineering in Health Care (SEHC'2014)*, 2014.

[11] Paolo Masci, Patrick Oladimeji, Piergiuseppe Mallozzi, Paul Curzon, and Harold Thimbleby. PVSio-web: Mathematically Based Tool Support for the Design of Interactive and Interoperable Medical Systems. In *Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare*, MOBIHEALTH'15, pages 42–45, ICST, Brussels, Belgium, Belgium, 2015. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[12] Paolo Masci, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. Formal verification of medical device user interfaces using PVS. In Stefania Gnesi and Arend Rensink, editors, *Fundamental Approaches to Software Engineering*, volume 8411 of *Lecture Notes in Computer Science*, pages 200–214. Springer Berlin Heidelberg, 2014.

[13] Bridget Moorman. Medical Device Interoperability: Standards Overview, April 2010.

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

[15] John G. Rhoads, Todd Cooper, Ken Fuchs, Paul Schluter, and Raymond P. Zambuto. Medical Device Interoperability and the Integrating the Healthcare Enterprise (IHE) Initiative. *Biomedical instrumentation & technology*, suppl.:21–27, 2010.