5th Workshop on Medical Cyber-Physical Systems

MCPS'14, April 14th, 2014, Berlin, Germany

Edited by

Volker Turau
Marta Kwiatkowska
Rahul Mangharam
Christoph Weyer

OASICS

Editors

Volker Turau
Institut of Telematics
Hamburg University of Technology
`turau@tuhh.de`

Marta Kwiatkowska
Department of Computer Science
University of Oxford
`marta.kwiatkowska@cs.ox.ac.uk`

Rahul Mangharam
Department of Electrical Systems Engineering
University of Pennsylvania
`rahulm@seas.upenn.edu`

Christoph Weyer
Institut of Telematics
Hamburg University of Technology
`c.weyer@tuhh.de`

ACM Classification 1998
A.0 Conference proceedings

ISBN 978-3-939897-66-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-66-8.

Publication date
April, 2014

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.MCPS.2014.i

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 2190-6807

www.dagstuhl.de/oasics

Contents

Preface
Marta Kwiatkowska, Volker Turau, and Rahul Mangharam i

First Session

A Generic User Interface Architecture for Analyzing Use Hazards in Infusion Pump Software
Paolo Masci, Yi Zhang, Paul Jones, Harold Thimbleby, and Paul Curzon 1

An Approach to Integrate Distributed Systems of Medical Devices in High Acuity Environments
David Gregorczyk, Stefan Fischer, Timm Busshaus, Stefan Schlichting, and Stephan Pöhlsen 15

Simulations of the Cardiovascular System Using the Cardiovascular Simulation Toolbox
Gabriela Ortiz-León, Marta Vílchez-Monge, and Juan J. Montero-Rodríguez 28

Adaptive Failure Detection and Correction in Dynamic Patient-Networks
Martin Ringwelski, Andreas Timm-Giel, and Volker Turau 38

Second Session

Challenges and Opportunities in Design of Control Algorithm for Artificial Pancreas
Mahboobeh Ghorbani and Paul Bogdan 49

Automatic Resource Scaling for Medical Cyber-Physical Systems Running in Private Cloud Computing Architecture
Yong woon Ahn and Albert Mo Kim Cheng 58

Third Session

Modeling of Reconfigurable Medical Ultrasonic Applications in BIP
Stefanos Skalistis and Alena Simalatsar 66

A Domain Specific Language for Performance Evaluation of Medical Imaging Systems
Freek van den Berg, Anne Remke, and Boudewijn R. Haverkort 80

A Safety Argument Strategy for PCA Closed-Loop Systems: A Preliminary Proposal
Lu Feng, Andrew L. King, Sanjian Chen, Anaheed Ayoub, Junkil Park, Nicola Bezzo, Oleg Sokolsky, and Insup Lee 94

5th Workshop on Medical Cyber-Physical Systems (MCPS'14).
Editors: Volker Turau, Marta Kwiatkowska, Rahul Mangharam, and Christoph Weyer
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Fourth Session

Evaluating On-line Model Checking in UPPAAL-SMC using a Laser Tracheotomy Case Study
Xintao Ma, Jonas Rinast, Sibylle Schupp, and Dieter Gollmann 100

Integrating Safety Assessment into the Design of Healthcare Service-Oriented Architectures
Ibrahim Habli, Abdulaziz Al-Humam, Tim Kelly, and Leila Fahel 113

Design Pillars for Medical Cyber-Physical System Middleware
David Arney, Jeff Plourde, Rick Schrenker, Pratyusha Mattegunta, Susan F. Whitehead, and Julian M. Goldman 124

Poster Session

OR.NET – Approaches for Risk Analysis and Measures of Dynamically Interconnected Medical Devices
Franziska Kühn, Martin Leucker, and Alexander Mildner 133

Automated Verification of Quantitative Properties of Cardiac Pacemaker Software
Marta Kwiatkowska and Alexandru Mereacre 137

Potential Advantages of Applying Assurance Case Modeling to Requirements Engineering for Interoperable Medical Device Systems
Rick Schrenker, Jeff Plourde, Diego Alonso, David Arney, and Julian M. Goldman 141

Process-Oriented Analysis for Medical Devices
Vasiliki Sfyrla, Josep Carmona, and Pascal Henck 143

Preface

This volume contains the proceedings of the *5th Medical Cyber-Physical Systems Workshop: Medical Device Interoperability, Safety, and Security Assurance* held as part of CPSWeek'14 in Berlin, Germany, on April 14, 2014.

As co-chairs of this year's workshop, we are delighted to introduce the exciting programme of scientific papers the interested reader. Programmable medical devices, for example infusion pumps, glucose monitors, cardiac pacemakers and defibrillators, must be safe, secure and dependable, as otherwise human lives are put at risk. There is a growing need for methodologies, standards and regulatory procedures that necessarily have to involve all the stakeholders across academia and industry.

The workshop series was motivated by the idea to bring together the HCMDSS (High Confidence Medical Devices, Software, and Systems) and MD PnP (Medical Device Plug-and-Play Interoperability) communities of medical device specialists (researchers, developers, clinicians, regulators and policy makers) from clinical environments, industry, research laboratories, academia and government. The hope was that joint meetings would accelerate the development of science, technology and practice to overcome crucial challenges facing the design, manufacture, certification and use of medical devices. The workshop series provides a regular forum for the presentation of research and development covering all aspects of high integrity medical devices, software and systems, which is essential to support innovative, networked medical device systems to improve safety and efficiency in health care.

The first three workshops in the series were called the Joint Workshop on HCMDSS/M-DPnP. Since 2013, the workshop title was changed to Medical CPS to broaden its scope. The previous four workshops were as follows:

- The first Joint Workshop on HCMDSS/MDPnP: Improving Patient Safety through Medical Device Interoperability and High Confidence Software, Cambridge, MA, June 25–27, 2007.
- The second Joint Workshop on HCMDSS/MDPnP, CPSWeek 2009, San Francisco, April 16, 2009.
- The third Joint Workshop on HCMDSS/MDPnP, CPSWeek 2011, Chicago, April 11, 2011.
- The fourth Medical Cyber-Physical Systems Workshop: Medical Device Interoperability, Safety, and Security Assurance, CPSWeek 2013, Philadelphia, April 8, 2013.

The programme of this year's workshop is covers a broad range of themes relevant for the development, verification and practical application of medical devices, and includes a topical keynote lecture by Prof. Dr. Thomas Stieglitz, Laboratory for Biomedical Microtechnology, University of Freiburg, Germany, entitled "Neural Implants – about science and fiction". The remainder of the programme is composed of 8 full and 5 short papers selected from 20 submissions, as well as 5 poster presentations. Among the themes covered, we would like to highlight aspects of novel methodology that is being put forward, and specifically user interface design for hazard analysis, application of runtime verification to ensure safety and modelling of reconfigurability of ultrasonic medical devices. The medical applications featuring in the programme include the cardiovascular system, laser tracheotomy and artificial pancreas.

This workshop would not have been possible without the support of the Steering Committee, and particularly Prof. Insup Lee who offered valuable advice. We are very grateful to the Programme Committee members and additional reviewers for their timely and rigorous reviewing.

As a final note, we hope that the participants of the workshop and readers of the proceedings volume will find the programme exciting, and that the meeting stimulates closer collaboration between the stakeholders in this important and fast moving field, and eventual progress towards safe, secure and dependable medical devices.

March 2014

Marta Kwiatkowska
Volker Turau
Rahul Mangharam

Workshop Organization

Programme Committee

- Arvind Easwaran (Nanyang Technological University, Singapore)
- José Maria Fernandes (IEETA, University of Aveiro, Portugal)
- Stefan Fischer (University of Lübeck, Germany)
- Radu Grosu (Vienna University of Technology, Austria)
- Martin Leucker (University of Lübeck, Germany)
- Sören Lewis (Otto Bock Healthcare, Germany)
- Maria Lindén (Maladalen University, Sweden)
- Jane W. S. Liu (Institute of Information Science, Academia Sinica, Taiwan)
- Wendy MacCaull (St. Francis Xavier University, Canada)
- Dominique Méry (LORIA, France)
- Christain Renner (University of Lübeck, Germany)
- Sibylle Schupp (Hamburg University of Technology, Germany)
- Vasiliki Sfyria (Viseo Research and Development, France)
- Alena Simalatsar (EPFL, Switzerland)
- Scott A. Smolka (Stony Brook University, USA)
- Andreas Timm-Giel (Hamburg University of Technology, Germany)
- Hoc Khiem Trieu (Hamburg University of Technology, Germany)
- Alan Wassyng (McMaster University, Canada)

Additional Reviewers

- Ezio Bartocci
- Timm B. Busshaus
- Milan Ceska
- David Gregorczyk
- Alexandru Mereacre
- Nicola Paoletti
- Neeraj Singh
- Annette Stümpel
- Anton Tarasyuk
- Daniel Thoma

A Generic User Interface Architecture for Analyzing Use Hazards in Infusion Pump Software

Paolo Masci[1], Yi Zhang[2], Paul Jones[2], Harold Thimbleby[3], and Paul Curzon[1]

1 **Queen Mary University of London**
London, United Kingdom
`{p.m.masci,p.curzon}@qmul.ac.uk`

2 **Center for Devices and Radiological Health**
U.S. Food and Drug Administration, Silver Spring, MD, USA
`{yi.zhang2,paul.jones}@fda.hhs.gov`

3 **College of Science, Swansea University**
Swansea, Wales, United Kingdom
`h.thimbleby@swansea.ac.uk`

Abstract

This paper presents a generic infusion pump user interface (GIP-UI) architecture that intends to capture the common characteristics and functionalities of interactive software incorporated in broad classes of infusion pumps. It is designed to facilitate the identification of use hazards and their causes in infusion pump designs. This architecture constitutes our first effort at establishing a model-based risk analysis methodology that helps manufacturers identify and mitigate use hazards in their products at early stages of the development life-cycle.

The applicability of the GIP-UI architecture has been confirmed in a hazard analysis focusing on the number entry software of existing infusion pumps, in which the GIP-UI architecture is used to identify a substantial set of user interface design errors that may contribute to use hazards found in infusion pump incidents.

1998 ACM Subject Classification K.6.1 Systems Analysis and Design, H.5.2 User Interfaces, D.2.11 Software Architectures

Keywords and phrases Infusion Pump, Hazard Analysis, Use Hazards, User Interface, Interactive Software, Design Errors

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.1

1 Introduction

Infusion pumps are a class of medical devices widely used in various clinical settings to deliver fluids (including medication and nutrients) into a patient's body in a controlled (prescribed) manner. The safety of infusion pumps, however, has been one of the top concerns in health care for a number of years [7]. For example, during the period from 2005 to 2009, 87 recalls associated with infusion pumps were reported in US due to defective design or manufacturing problems [8]. Through the analysis of incidents involving infusion pumps, medical device regulators such as the US Food and Drug Administration (FDA) concluded that two of the major factors contributing to infusion pump failures were *software defects* and *user interface issues* [33].

Many user interface issues can be associated with software problems. For instance, a key bounce may be caused by a defect in the interrupt-handling code, and problems related to configuring the pump may be caused by inappropriate user-device interaction logic. However, there are also user interface issues that have roots in the software engineering process. For example, user interface problems may arise if the device's interaction behavior is inconsistent with its user manual or labeling. Either type of user interface issues can affect the device operation and hence affect patient safety. To help reduce or eliminate these issues, a systematic risk management process should be carried out.

ISO14971:2007 is a standard that provides a systematic risk management process for medical devices. In brief, it consists of five distinct activities. First, a hazard analysis is performed to identify all known and foreseeable hazards and their causes, where a *hazard* is defined as a potential source of physical injury or damage to people or environment. Second, risk estimation is performed to assess the probability of occurrence and severity of harm of each hazard, the combination of which is defined as *risk*. Third, risk evaluation is conducted to decide if every identified risk is acceptable based on pre-defined acceptability criteria. Fourth, if a risk is decided as unacceptable, control measures are designed and implemented to eliminate it or to mitigate it to an acceptable level. Finally, verification and validation activities are conducted to ensure that the designed control measures are effective. These five activities iterate and interleave until the device's overall residual risk after mitigation is acceptable.

In the ISO14971 risk management process, the identification of hazards constitutes the first step and provides the basis for subsequent activities. In our previous work [10, 36], we illustrated the benefits of using a Generic Infusion Pump (GIP) architecture to support a systematic hazard analysis early in the design process. The GIP architecture, which captures common characteristics and functionalities of broad classes of infusion pumps, serves as a reference 'standard' for reasoning about hazards and potential causal factors in infusion pump designs.

In this paper, we present an effort that extends our previous work focusing on the user interface design and its associated use hazards. In particular, we extend the GIP architecture with more details that reflect common user-device interaction designs in existing infusion pumps. This serves to establish a foundation for reasoning about use hazards in infusion pumps and their contributing factors rooted in defective interaction (software) design.

Contributions. The contributions of this paper are as follows: (*i*) a Generic Infusion Pump User Interface (GIP-UI) architecture is presented that can be used to reason about design defects in infusion pump user interface software that may potentially cause use hazards; (*ii*) an analysis of infusion pump incident reports and other information sources is presented that summarizes the common use hazards in infusion pumps on the market related to number entry tasks; and (*iii*) a preliminary hazard analysis that uses the GIP-UI to identify and reason about design errors commonly present in infusion pump number entry software and their relation to use hazards.

It is worth noting that the use hazards considered in this work, as well as software design defects that cause these hazards, are common in other interactive medical devices (e.g., ventilators) that have similar number entry systems. Therefore, we believe that the GIP-UI architecture may also be useful to the hazard analysis on these devices.

Organization. Section 2 presents background information about the GIP reference model. Section 3 elaborates the details of the GIP-UI architecture. Section 4 demonstrates how the GIP-UI architecture can be used to facilitate the analysis of use hazards related to user interface (in particular, the number entry software of a user interface). Section 5 compares our work with other related work. Finally, Section 6 concludes the paper.

2 Background: the Generic Infusion Pump model

The GIP model is a safety reference model for infusion pumps that captures the common functionalities of modern infusion pumps. Figure 2 illustrates an abstract architecture of the GIP model, and the functionalities of each of its components are follows:

The *GIP controller* represents an abstraction of the pump software that manages the overall infusion process and supervises interaction among all GIP components. It is responsible for instructing the pumping mechanism to deliver fluid as prescribed, as well as detecting and reporting alarm and warning conditions.

The *GIP user interface (GIP-UI)* represents an abstraction of the device elements (hardware and software) that enable interaction with the user. That is, the user can enter data (e.g., infusion parameters and pump settings) and control pump operation (e.g., start/stop infusion). The user can also receive from the user interface feedback on the pump and infusion status.

The *device data recorder* represents an abstraction of the logging mechanism used to record the pump's operational history (such as critical data and important events) to facilitate problem diagnosis.

The *fluid pathway* represents an abstraction of the following elements: fluid reservoir, which stores the fluid to be delivered; infusion set, which is usually an IV needle; and delivery interface, which is a tube that connects the fluid reservoir to the infusion set.

A more detailed description of the GIP model can be found in [10, 36].

3 GIP User Interface (GIP-UI)

The GIP model was originally designed to provide a basis for analyzing the safety and correctness of the software control logic in infusion pumps. It had a simple means for input/output, and therefore lacks any design or engineering details on user interface design. The GIP-UI architecture, depicted in Figure 1, replaces this user interface part with new architectural details, in order to establish a basis for reasoning about use hazards and interface design errors in infusion pumps.

One important principle in the GIP-UI architecture is to enforce a clear separation between high-level functionalities of user interactions and low-level functionalities that enable communication among its components (e.g., user interface elements that translate electrical signals into logical events). Moreover, issues associated with user manuals and use environment are also considered to enable a more comprehensive (hazard) analysis of user interface design.

The role of each component in the GIP-UI architecture is defined as follows.

GIP-UI widgets. This component represents an abstraction of the mechanical and hardware elements of user interface that enable pump-user interaction. Widgets can be either input or output widgets. Examples of input widgets include buttons, switches, and knobs, while examples of output widgets are displays, alarms, LED lights.

GIP-UI Software. This component represents an abstraction of the software regulating all functionalities of the user interface. It translates the user input received from the input widgets to commands understandable by the pump control software, manages interactive tasks such as number entry, and manages the output widgets to provide feedback (such as alarms and infusion status) to the user.

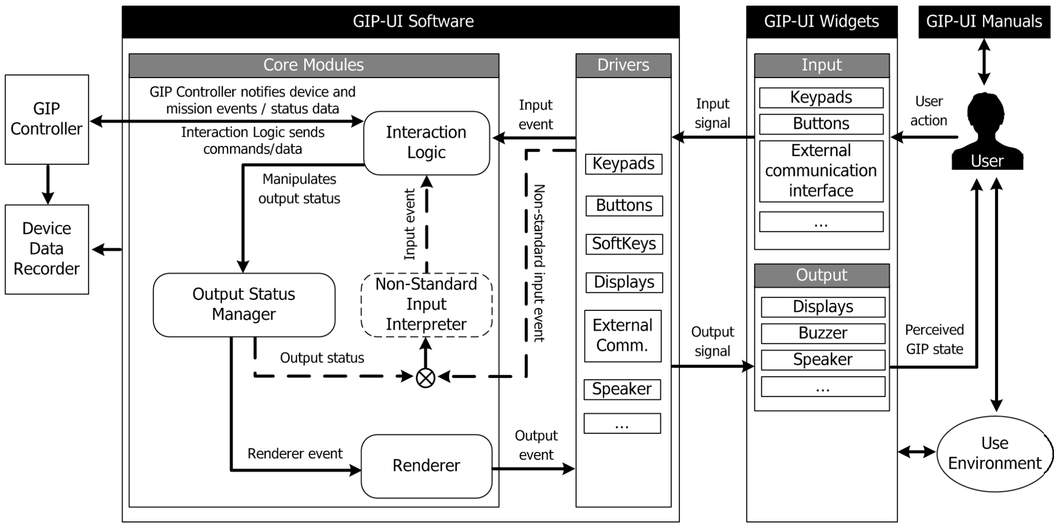


Figure 1 Logic architecture of the GIP user interface (GIP-UI). Labeled boxes represent an abstraction of functional components of the system. Arrows between components represent flows of information, user actions, or mechanical force between components. Dashed lines represent optional components and information flows.

The GIP-UI Software includes the following logic modules:

- *Core Modules.* These software modules define the interactive behavior of the device, i.e., how the pump responds to input events and how the device's operational state is updated as a consequence of input events. Input events are either from the user (labeled as 'user actions' in Figure 1) or from the pump control software (as abstracted as GIP Controller in Figure 2). The events from the GIP Controller are labeled as 'mission events' and 'status data' in Figure 1).
  The internal structure of the Core Modules is an instantiation of the well-known Model-View-Controller (MVC) [16] architectural pattern. The MVC pattern is chosen because it promotes a clear separation among different aspects of a user interface's core functionalities: defines the logic of interactions, decides what output information to be sent to the user, and decides how the output information is presented to the user. The Core Modules consist of:
  - The *Interaction Logic* , which represents an abstraction of the routines for handling user input events and forwarding them to the pump control software. These routines define: (i) input key sequences needed to interact with the device, and corresponding algorithms for processing input key sequences, and (ii) the communication protocol with the pump control software.
  - The *Output Status Manager* , an abstraction of the routines that specify feedback information, i.e., *what* feedback is presented to the user.
  - The *Renderer* , an abstraction of the routines for rendering feedback information on output widgets, i.e., *how* feedback is presented to the user.
  - The *Non-standard Input Interpreter* , which represents an abstraction of the routines for managing input widgets that are more complex than keypads or mechanical buttons. Examples of the non-standard input widgets include touchscreens or gesture recognition systems. For these input widgets, additional computation is needed to correctly interpret the user inputs. For example, in touchscreens, user gestures identify

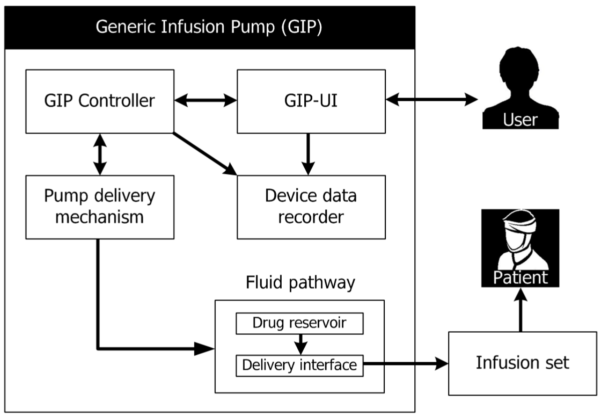


Figure 2 Logic architecture of the GIP. Labeled boxes represent an abstraction of functional components of the system, and arrows represent flows of information (e.g., software commands, hardware signals), user actions, or mechanical force between components. Note: Patient may overlap with User for some types of infusion pumps.

just coordinates on the screen, and further processing is needed to associate the screen coordinates to the user action (on the input widgets rendered on the screen)[1].

- *Drivers.* These software modules translate digital signals received from Input Widgets into events that can be processed by the GIP-UI Core Modules, or translate output events instructed by the Core Modules into output signals that Output Widgets can understand and produce output accordingly. The input events from the Drivers can be divided into two types: standard or non-standard. Standard events are generated during interactions with classical mechanical input devices such as mechanical buttons. An example of a standard event is a button click. Non-standard events, on the other hand, are generated during interactions with input elements such as soft buttons or voice recognition systems.

GIP-UI Manuals. This component represents an abstraction of the reference material accompanying the pump, e.g., user manual and training material. Such material is part of the user knowledge and is therefore useful for identifying potential use hazards.

User. This models the characteristics of the intended user population. Examples of user characteristics include: user training level, cognitive and physical abilities, attitudes and behaviors. Note that the user can be the patient or her family members for certain devices, and hence may not be trained nor be an experienced operator.

Use Environment. This component abstracts the physical environment in which the pump is used. Environmental factors such as ambient light conditions can affect the device's interaction with the user and thus should be included into considerations during user interface design. In Figure 1, arrows between the use environment and other components indicate what part of the system can be affected by the use environment. For example, inappropriate temperature levels may affect the physical abilities of patients and users, or go beyond the operating temperature of the device itself.

[1] See [6] for a detailed illustration of the function of this module.

4 Validation of the GIP-UI architecture

The GIP-UI architecture explicitly defines how the device interacts with the user. Thus, it can be used as a basis to examine where and how the device incorrectly interacts with the user, which may affect the device's expected operation and ultimately create potential hazards. This makes it easier to identify potential use hazards, or to cross-check the identified use hazards to ensure that no important hazards be missed.

More importantly, we believe:

> Having a generic GIP-UI architecture facilitates the identification of common design defects in user interface designs and provides insight into the cause-effect relationship between these defects and infusion pump use hazards.

To prove the above hypothesis, we conducted a preliminary hazard analysis (PHA) using the GIP-UI architecture to 1) identify a set of common use hazards in infusion pumps related to number entry tasks and 2) to reason about the common design errors in user interface software that may contribute to these use hazards.

4.1 Hazard analysis based on the GIP-UI

Our PHA focuses on the number entry part of the infusion pump interface. The *number entry software* in an infusion pump is responsible for managing interaction with the user when infusion parameters or pump settings need to be configured. This is chosen as the focus of our PHA because it is critical to infusion pump safety, in the sense that design errors in it can cause use errors (e.g., mis-programming of the pump) with potentially severe consequences to the patient (typically, overinfusion or underinfusion).

Our PHA of the number entry part follows a top-down approach: it starts from postulated undesired system outcomes, and then works out the causes of the postulated outcomes at the system design level. In the analysis, undesired system outcomes are given by use hazards documented in infusion pump incident reports, including FDA Adverse Event reports.

4.1.1 Scope of the analysis

Number entry software is designed to support a *number entry task*, which identifies the sequence of actions carried out by the user when entering infusion parameters or pump settings. In the current generation of infusion pumps, the typical number entry task is carried out through the following three main actions:

1. An infusion parameter or pump setting is selected by the user
2. The selected item is edited by the user
3. The value is submitted by the user

The actions described above generally involve pressing buttons and keys on the pump user interface. Whenever the pump registers a button press or a key press, the number entry software performs a computation. The computation may modify the device state (e.g., a new infusion rate may be configured in the pump), and generate feedback on the user interface to present the new device state to the user.

Buttons and keys currently used for number entry can be described using two broad classes of widgets: *serial number entry widgets*, which allow the user to enter the digits of the values serially from the most significant to the least significant digit, and *incremental number entry widgets*, which allow the user to modify an initial value by incrementing or decrementing it. Serial interfaces require a full numeric keypad, whereas incremental interfaces typically have two or four arrow keys (depending on the exact style of interaction). A detailed description of these number entry widgets is in [3]. Notable for hazard analysis is how entry errors can be corrected. In a serial interface, either numbers have to be re-entered or there must be a *delete* key. In contrast, in incremental interfaces, the whole point of the user interface is to adjust numbers, so correcting errors does not require a separate *delete* key, as correction is just a special case of adjustment.

4.2 Sources of information

The analysis is informed by the following sources of information:

- Domain knowledge developed within the CHI+MED project (www.chi-med.ac.uk). This knowledge results from the analysis of commercial infusion pumps [11, 12, 19, 22, 23, 24, 27], infusion pump logs [17], incidents involving infusion pumps [13, 20, 21, 31], current and best clinical practice in hospitals and home care [29], and workshops with pump manufacturers, users and clinicians [4, 5];
- Adverse event reports collected through the FDA's Manufacturer and User Facility Device Experience (MAUDE) database [34];
- Recommendations on infusion pump design [25];
- Previous hazard analysis on other components of the GIP [10, 36];
- International standards ANSI/AAMI HE75:2009 on human factors, and ISO 14971:2007 on risk management.

4.3 Hazard analysis results

Using the GIP-UI our PHA identified 60 potential design errors in number entry software that may cause use hazards. From the GIP-UI perspective, these design errors arise from individual components in the architecture, from a combination of these components, or from the communication (i.e., information flow) between these components. The full table is in the technical report [28].

Table 1 is a sample of this hazard analysis table. The table shows the considered hazards at the top: over-infusion (or under-infusion) where the patient receives more (or less) drug/nutrients than prescribed. Each row in the table identifies a use error that can lead to the over-/under-infusion hazards, as well as an underlying design error in infusion pump software that can cause this use error. A concrete example (as found in incident reports or our previous study) is also presented for each identified issue, to help understand the nature of the corresponding issue.

For illustrative purposes, we present design errors found to be commonly present in number entry software of infusion pumps currently on the market. These design errors were found in real marketed devices and may lead to severe clinical consequences, as they can potentially lead to situations where the pump is erroneously programmed with incorrect infusion parameters or pump settings without the user's awareness.

4.3.1 Potential design errors in Output Status Manager

Design errors in the Output Status Manager generally lead to inappropriate, inaccurate, or incorrect feedback to the user. As a result, the user may not be able to receive adequate information on the system's status, what user action has actually been registered, and what result has been accomplished. Common instances of this type of design errors are as follows:

Table 1 Sample of the developed hazard analysis table. The full table is in the technical report [28].

Hazards: overinfusion or underinfusion		
Use error	**Underlying design error**	**Example**
The user fails to edit the value	The pump displays incorrect values without the user's awareness	The device shows the last programmed value instead of the current value
The user fails to select the intended input field	The pump displays instructions to the user prematurely due to incorrect assumption on user action	The device requires a rate value but the display shows a notification message "Enter VTBI"
The user fails to read values or units correctly	The pump uses inappropriate fonts or formats to render values or units	The device renders fractional values without a leading zero (e.g., .9 instead of 0.9), or integer values with leading zeros (e.g., 09 instead of 9)
The user fails to enter the correct digits	The pump erroneously discards key presses without the user's awareness	The device registers the key sequence $\boxed{0}\,\boxed{.}\,\boxed{0}\,\boxed{1}$ without any warning or notification if the minimum legal rate is 0.1
...	...	...

- *Incorrect values are displayed without the user's awareness.* Pumps affected by this design error have an *ambiguous* display, i.e., the user is not able to tell the current system state from the observable information on the pump display. Consider the following error found in a marketed infusion pump. This pump displays the last programmed value or the last valid input value, instead of the current number entry value that the user enters. For example, it displays a value of 90 when the user starts the number entry task, since the last value programmed in the pump is 90. However, given that the user has not pressed any key yet, the current display value should be 0. This incorrect display value misleads the user in carrying out the rest of number entry. For instance, the user may press key $\boxed{1}$ in order to entering the number 901, while the actual value registered and displayed by the pump is 1. If not noticed, this can cause the user to incorrectly program the pump, and result in unexpected treatment to the patient. Our forensic analysis on this pump accredited the root cause of this error to initialization routines of the number entry system and routines for handling *illegal* input key sequences.
- *Instructions are displayed to the user prematurely due to incorrect assumption on user action.* Infusion pumps affected by this type of error usually display instructions that conflict with the ongoing actions taken by the user. A common cause of such errors is that the infusion pump incorporates routines that automatically validate the value entered by the user, even before the user actually finishes entering and confirms the value. Consider an instance of such errors found in a marketed infusion pump. The pump shows a notification message 'Enter Rate' when the user is expected to enter the rate value, and 'Enter VTBI' to instruct the user to enter the value of drug volume to be infused. However, if the user plans to enter a rate value of 109 ml/h but pauses for a

second after entering $\boxed{1}$ $\boxed{0}$, the pump erroneously makes an assumption that the user has finished entering the rate, registers the entered value as 10, and displays the message 'Enter VTBI' to prompt the user to enter the drug volume to be infused. In other words, design errors of this kind may cause *mode confusion* to the user: the users thinks the pump is in mode x (in this case, x is entering the rate) while it is actually in mode y (in this case, y is entering the volume to be infused). In fact, this type of error ('right data, wrong field') appears to be the most common use errors documented in infusion pump incident reports [8].

4.3.2 Potential design errors in Renderer

Design errors in the Renderer module typically introduce visualization issues in the device. For example, the device renders information erroneously or inconsistently, or fails to make appropriate display elements perceptible.

- *Inappropriate fonts or inappropriate formats are used to render numbers or units.* Example problems found in marketed devices include: fractional values are displayed without a leading zero (e.g., displaying .9 instead of 0.9); integer values are rendered with leading zeros (e.g., 09 instead of 9); and small decimal point (e.g., the decimal point is rendered as · instead of ●). Another example is with seven-segments displays [32]. Rendering values in seven-segment displays can easily cause the user to misread the values, as integer and fractional digits are hard to distinguish, and can result in out-by-ten errors.
- *Soft keys are incorrectly labeled without the user's awareness.* Soft keys are buttons that can be programmed to perform different functions during the pump's operation. Modern infusion pump usually implement soft keys as hard buttons placed on the sides of the screen, while soft button labels are displayed on the screen next to these hard buttons. We found it common for infusion pumps to display textual messages not intended to be soft button labels next to hard buttons, creating the illusion that they were. It is also common that infusion pumps erroneously render labels next to unused soft keys. Consider a pump with a soft key on the right of the screen, and another soft key on the left. Assume that these soft keys are aligned. If the pump renders 'Rate' aligned to the soft key on the left, and "2 ml/h" aligned to the soft key on the right, which soft key should be used to select the rate? The reasonable affordance[2] is that both soft keys be used for this purpose, as the two pieces of text are logically a single piece of information. The pump that we studied, however, disables one of these two keys, without providing any feedback when the user presses the disabled key. The likely clinical consequence of such errors is delay of treatment.

4.3.3 Potential design errors in Interaction Logic

Design errors in the Interaction Logic module generally result in incorrect human-computer interaction with buttons, keys, and displays. Typical causes of such errors include inappropriate or over-complicated procedures (i.e., sequence of actions) to interact with these widgets or the failure of implementing mechanisms for preventing or detecting user errors during interaction. Common instances of design errors in Interaction Logic include:

[2] Affordance is a property of objects that determines how the object can possibly be used [26].

- *Key presses by the user is erroneously discarded without the user's awareness.* Pumps affected may unexpectedly discard key presses and commit an out-by-ten error, which can potentially result in severe clinical consequences. For example, an infusion pump studied by us [3] enforces the constraint that all infusion rate values must be greater than or equal to 0.1. Thus, its number entry routines discards the third key press in the key sequence $\boxed{0}\boxed{.}\boxed{0}\boxed{1}$, as the value violates the constraint, and automatically sets the input rate as the minimum valuate 0.1. Moreover, the pump enforces that no fractional numbers are allowed for value above or equal to 100. Thus, the decimal point key press in the input sequence $\boxed{1}\boxed{0}\boxed{0}\boxed{.}\boxed{1}$ is discarded, and the erroneous value 1001 is registered. No warning or notification is provided to the user in either of these two cases. The root cause of this design error typically resides in the number entry routines for handling illegal input values (e.g., out-of-range or ill-formed values).
- *Values entered by the user are erroneously discarded without the user's awareness.* Pumps affected by this design error unexpectedly discard the value entered by the user if the input field is de-selected (e.g., the user selects another input field without confirming this entered value). The following is an example detected in a marketed infusion pump: the user first changes the infusion rate from its current value 91 mL per hour to 0.9 mL per hour, and then selects the VTBI field to edit without confirming the new infusion rate. As a result, the pump automatically discarded the new rate value and maintained the previous value, without any notification or warning. Design errors like this can cause the user to mistakenly configure the treatment to the patient, and result in serious clinical consequences.

4.3.4 Potential design errors in Non-standard Input Interpreter

Design errors in the Non-standard Input Interpreter generally result in human-machine interaction issues with touch-screen displays. A device with such errors typically requires the user to take an inappropriate or over-complicated sequence of actions to interact with the non-standard input widgets. It may also fail to implement necessary mechanisms to prevent user errors during the interaction. Common design errors in the Non-standard Input Interpreter include:

- *Legal gestures on touchscreens are erroneously discarded without the user's awareness.* Devices with this erroneous design may ignore legal gestures on input widgets. For example, slide gestures on scroll bars are erroneously ignored; press and hold gestures on virtual buttons and tap gestures on input fields are erroneously ignored. This design error can be associated with number entry routines that activate or select touchscreen input widgets. The likely consequence of this kind of errors is the delay of number entry.
- *Similar gestures on touchscreens are erroneously associated with logically different functions.* Pumps affected by this design error erroneously execute functions not intended by the user. This design error may be associated with number entry routines that activate or select touchscreen input widgets, such as virtual buttons and input areas. The following is an example error detected in a marketed pump. The user selected the infusion rate field and plans to enter a value. However, the pump's touchscreen registers a tap gesture outside the area of the selected rate input field or outside the virtual buttons for number entry, probably because the user accidentally taps outside the widgets or the touchscreen

[3] More information about this study can be found in [24].

is mis-calibrated. Thus, the input field is automatically de-selected, without any warning or notification for the user. Notably, this design error may aggravate other design errors such as those in the Interaction Logic, as values may be mistakenly confirmed or discarded because of accidental touch on the touchscreen.

4.4 Discussion

As shown in the PHA results, the GIP-UI architecture provides a basis for us to enumerate common errors in user interface design associated with number entry. It also facilitates the establishment of the cause-effect relationship between these errors and the use hazards. As demonstrated in our PHA, this cause-effect relationship can help device developers make their hypotheses and assumptions explicit when reasoning about user interface behaviors, and thus enable clear thinking about possible causal factors. Ultimately, it can assist developers in designing effective mitigation measures to address use hazards and design errors causing them.

The GIP-UI architecture and hazard analysis results presented in this paper can be used by manufacturers as an independent reference to challenge the safety of their own user interface design. Alternatively, manufacturers can use our work as the starting point to perform more comprehensive hazard analysis on their products (in this case, manufacturers should populate the GIP-UI architecture with design details specific to their products).

It is worth pointing out that, even though our PHA identified a substantial set of use hazards and their root causes [4], it is by no means exhaustive. In fact, a PHA is only the first round of hazard analysis applied to a design, and its results constitute an initial yet informative inventory for subsequent detailed hazard analysis or the initial design of risk mitigation measures. In order to identify all potential hazards (and related causes) in complex systems, the best practice is to employ a combination of systematic hazard analysis techniques, such as Failure Mode and Effect Analysis (FMEA) [9] or Systems-Theoretic Accident Modeling and Processes (STPA) [18], in a complementary manner [14].

5 Related work

Although there are many examples of conceptual/abstract architectures that can be used for hazard analysis, few of them are designed to support the identification of use hazards and their potential causal causes in medical devices.

In [2, 1], an architecture is developed for a STPA-based hazard analysis for a radiotherapy system. The architecture describes the functions of five sub-systems of the Paul Scherrer Institute's experimental ProScan proton therapy system. Usability issues were considered in the study. However, this architecture was not designed to explore potential causal factors of use hazards, and some information on usability issues were obtained through workshops with domain experts. Similarly, in [30], a system architecture is developed for a STPA hazard analysis of pacemakers. Even though pacemakers do not have a user interface, the analysis considered a wider, system-level perspective that included users and patients. Users are modeled as 'human controllers' guided by mental models. Use hazards, however, were not in the scope of the analysis. The architecture presented in this paper can be used together with analysis techniques such as STPA. It provides information that can be used as a basis to

[4] The full hazard tables are in [28].

populate the *controller* and *controlled process* components of the system architecture with details necessary to analyze the potential causes of use hazards.

In [15], design errors related to barcoded medication administration systems are explored. These systems are commonly used together with infusion pumps and other medical devices to identify patients, clinicians, and drugs. The analysis in [15] revealed how defective barcoding systems encouraged workarounds that could potentially lead to severe clinical consequences. These results complement the results of our hazard analysis, in that they explore issues in the design of an alternative number entry system that can be installed on infusion pumps.

Other works studying use hazards in medical devices usually overlook design issues. For instance, in [35], training and clinical procedures are identified as potential causes of use hazards. Consider a use hazard where an incorrect patient profile is selected. The established causal relationship identifies nurses being unclear about the available profiles, and therefore better training is suggested as a mitigating measure. Even though training and clinical procedures may be contributing factors of use hazards, this approach to hazard analysis provides little or no insights about how an infusion pump can be (re-)designed to assure safety under existing training and clinical procedures.

6 Conclusions

A generic user interface architecture, GIP-UI, has been presented to facilitate the identification and reasoning of use hazards in infusion pumps. Its applicability to this end has been confirmed through a hazard analysis that involved known use hazards in marketed infusion pumps. The architecture was successfully used to reason about the cause-effect relations between these hazards and their causes (i.e., software design errors) commonly present in user interface designs.

The GIP-UI architecture can potentially be used as the basis for hazard analysis on use hazards, and for the establishment of safety requirements that ensure reasonable safety regarding device-user interaction. It is also worth noting that, even though the GIP-UI is designed for infusion pumps, the general idea behind it can be applied to other medical devices that rely on the same mechanism as defined in GIP-UI to interact with the users, and thus facilitate the assessment and assurance of their safety in device-user interaction.

Acknowledgment. This work is support by CHI+MED (Computer-Human Interaction for Medical Devices, EPSRC research grant [EP/G059063/1]). We thank Michael Harrison (Newcastle University) and Chris Vincent (University College London) for their valuable comments that helped us to improve our manuscript.

References

1 B. Antoine. *Systems Theoretic Hazard Analysis (STPA) applied to the risk review of complex systems: an example from the medical device industry.* PhD thesis, Massachusetts Institute of Technology, 2013.

2 A. Blandine, M. Rejzek, and C. Hilbes. Evaluation of stpa in the safety analysis of the gantry 2 proton radiation therapy system, 2012.

3 A. Cauchi, P. Curzon, P. Eslambolchilar, A. Gimblett, H. Huang, P. Lee, Y. Li, P. Masci, P. Oladimeji, R. Rukšėnas, and H. Thimbleby. Towards dependable number entry for medical devices. In *EICS4Med, 1st International Workshop on Engineering Interactive Computing Systems for Medicine and Health Care.* ACM Digital Library, 2011.

4 CHI+MED. Guidelines for number entry interface design (infusion devices). http://www.chi-med.ac.uk/researchers/bibdetail.php?docID=684, 2013.

5 CHI+MED. Personas and scenarios (infusion devices). http://www.chi-med.ac.uk/researchers/bibdetail.php?docID=685, 2013.

6 S. Conversy, E. Barboni, D. Navarre, and P. Palanque. Improving modularity of interactive software with the MDPC architecture. In *EICS2007*. ACM Digital Library, 2007.

7 ECRI Institute 2014 top 10 health technology hazards, 2014.

8 Association for the Advancement of Medical Instrumentation. Infusing patients safely: priority issues from the AAMI/FDA infusion device summit. http://www.aami.org/publications/summits/AAMI_FDA_Summit_Report.pdf, 2010.

9 P. L. Goddard. Software fmea techniques. In *Reliability and Maintainability Symposium, 2000. Proceedings. Annual*, pages 118–123. IEEE, 2000.

10 The Generic Patient Controlled Analgesia Pump Hazard Analysis. http://rtg.cis.upenn.edu/gip-docs/Hazard_Analysis_GPCA.doc.

11 M. D. Harrison, J. Campos, and P. Masci. Reusing models and properties in the analysis of similar interactive devices. *Innovations in Systems and Software Engineering, Springer-Verlag London*, 2013.

12 M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon. Automated theorem proving for the systematic analysis of interactive systems. In *5th International Workshop on Formal Methods for Interactive Systems (FMIS2013)*, 2013.

13 ISMP Canada. Fluorouracil incident root cause analysis report. http://www.ismp-canada.org/download/reports/FluorouracilIncidentMay2007.pdf.

14 P. Jones, J. Jorgens III, A. R. Taylor Jr., and M. Weber. Risk management in the design of medical device software systems. *Journal of Biomedical Instrumentation and Technology*, 36(4):237–266, 2002.

15 R. Koppel, T. Wetterneck, J. L. Telles, and B. Karsh. Workarounds to barcode medication administration systems: their occurrences, causes, and threats to patient safety. *Journal of the American Medical Informatics Association*, 15(4):408–423, 2008.

16 G. E. Krasner and S. T. Pope. A description of the Model-View-Controller user interface paradigm in the Smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

17 P. Lee, F. Thompson, and H. Thimbleby. Analysis of infusion pump error logs and their significance for health care. *British Journal of Nursing*, 21(8):S12–S22, 2012.

18 N.G. Leveson. Software challenges in achieving space safety. *Journal of the British Interplanetary Society*, 2009.

19 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.

20 P. Masci and P. Curzon. Checking user-centred design principles in distributed cognition models: a case study in the healthcare domain. In *Proceedings of the 7th conference on Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society: information Quality in e-Health*, USAB'11, pages 95–108, Berlin, Heidelberg, 2011. Springer-Verlag.

21 P. Masci, H. Huang, P. Curzon, and M. D. Harrison. Using pvs to investigate incidents through the lens of distributed cognition. In Alwyn E. Goodloe and Suzette Person, editors, *Proceedings of the 4th NASA Formal Methods Symposium (NFM 2012)*, volume 7226, pages 273–278, Berlin, Heidelberg, April 2012. Springer-Verlag.

22 P. Masci, R. Rukšėnas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, and H. Thimbleby. On formalising interactive number entry on infusion pumps. *ECEASST*, 45, 2011.

23 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.

24 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.* Springer Berlin Heidelberg, 2014.

25 National Patient Safety Agency (NHS/NPSA). Design for patient safety: A guide to the design of electronic infusion devices. `http://www.nrls.npsa.nhs.uk/resources/?EntryId45=68534`, March 2010.

26 D. A. Norman. *The design of everyday things.* Basic books, 2002.

27 P. Oladimeji, H. Thimbleby, and A. Cox. Number entry interfaces and their effects on error detection. In *Human-Computer Interaction – INTERACT 2011*, volume 6949 of *Lecture Notes in Computer Science*, pages 178–185. Springer Berlin Heidelberg, 2011.

28 P. Masci et al. A preliminary hazard analysis for the GIP number entry software. `http://www.eecs.qmul.ac.uk/~masci/works/GIP-UI-PHA.pdf`, 2014.

29 R. Rukšėnas, P. Curzon, A. Blandford, and J. Back. Combining human error verification and timing analysis: a case study on an infusion pump. *Formal Aspects of Computing*, pages 1–44, 2013.

30 Q. S. M. Song. *A system theoretic approach to design safety into medical device.* PhD thesis, Massachusetts Institute of Technology, 2012.

31 H. Thimbleby. Is it a dangerous prescription? *BCS Interfaces*, 84, 2010.

32 H. Thimbleby. Reasons to question seven segment displays. In *Proceedings ACM Conference on Computer-Human Interaction — CHI 2013*, pages 1431–1440. ACM, 2013.

33 US Food and Drug Administration, Center for Devices and Radiological Health (FDA/CRDH). *White Paper: Infusion Pump Improvement Initiative*, 2010.

34 US Food and Drug Administration (FDA). Manufacturer and User Facility Device Experience Database (MAUDE). `http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/PostmarketRequirements/ReportingAdverseEvents/ucm127891.htm`.

35 T. B. Wetterneck, K. A. Skibinski, T. L. Roberts, S. M. Kleppin, M. E. Schroeder, M. Enloe, S. S. Rough, A. S. Hundt, and P. Carayon. Using failure mode and effects analysis to plan implementation of smart IV pump technology. *American Journal of Health-System Pharmacy*, 63(16):1528–1538, 2006.

36 Y. Zhang, P. Jones, and R. Jetley. A hazard analysis for a generic insulin infusion pump. *Journal of diabetes science and technology*, 4(2):263, 2010.

An Approach to Integrate Distributed Systems of Medical Devices in High Acuity Environments

David Gregorczyk[1], Stefan Fischer[1], Timm Busshaus[1], Stefan Schlichting[2], and Stephan Pöhlsen[3]

1 University of Lübeck
Institute of Telematics
Ratzeburger Allee 160
23562 Lübeck
`{gregorczyk,fischer,busshaus}@itm.uni-luebeck.de`

2 Drägerwerk AG & Co. KGaA
Smart Software Solutions
Research Unit
Moislinger Allee 53-55
23558 Lübeck, Germany
textttstefan.schlichting@draeger.com

2 Dräger Medical GmbH
Research & Development
Moislinger Allee 53-55
23558 Lübeck, Germany
textttstephan.poehlsen@draeger.com

Abstract

This paper presents a comprehensive solution to build a distributed system of medical devices in high acuity environments. It is based on the concept of a Service Oriented Medical Device Architecture. It uses the Devices Profile for Web Services as a transport layer protocol and enhances it to the Medical Devices Profile for Web Service (MDPWS) to meet medical requirements. By applying the ISO/IEEE 11073 Domain Information Model, device data can be semantically described and exchanged by means of a generic service interface. Data model and service interface are subsumed under the Basic Integrated Clinical Environment Specification (BICEPS). MDPWS and BICEPS are implemented as part of the publically available openSDC stack. Performance measurements and a real world setup prove that openSDC is feasible to be deployed in distributed systems of medical devices.

1998 ACM Subject Classification J.3 Life and Medical Sciences, C.2.2 Computer-Communication Networks

Keywords and phrases Integrated Clinical Environment, Devices Profile for Web Services, ISO/IEEE 11073

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.15

1 Introduction

Modern operating rooms (ORs) and intensive care units (ICUs) are equipped with numerous medical devices delivered by different manufacturers. While the amount of devices continuously increased over time, interoperability has not been adapted in the same way [14]. However, interconnecting interoperable medical devices can improve patient safety and optimize clinical workflows by providing the right information at the right time, in the right amount, at the right location and in the necessary quality [21]. As a result it makes the clinical work much easier and saves money.

Typical use cases are characterized by medical device safety-interlocks, remote control or data exchange with clinical information systems (CISs). Safety-interlocked devices perform a reciprocal monitoring to increase patient safety. For example, stopping an infusion at a predetermined blood pressure value or to prevent intra-abdominal CO2 insufflation if the heart rate and blood pressure are unmonitored [7]. Remote control means that designated device parameters can be controlled by remote devices. For example, muting a monitoring device's alert system or regulating the power of an ultrasound cutting device from a central OR cockpit. CIS communication comprises on the one hand a medical device providing data for, and on the other hand a medical device consuming data from a CIS or any electronic medical record system. Here, the most prominent example is the aquisition of patient demographics and other patient related data to be used during surgery.

The IEEE refers to interoperability as the ability of two or more IT systems to exchange information and to utilize the information that has been exchanged [9]. As a consequence, to achieve interoperability it is both important to assure error resistent data transmission and correct data interpretation. Assurance of data interpretation is twofold. Syntactic interpretation offers consistent data exchange according to an underlying specification, whereby semantic interoperability is the ability to interpret information exchanged with other systems, and to make effective use of it [8].

Unfortunately, in current distributed systems of medical devices interoperability is commonly achieved by using proprietary, vendor-dependent communication protocols and middleware. Products like Storz OR1 [12] or Olympus ENDOALPHA [17] provide fully integrated ORs. However, these systems are restricted to specific vendors and product models. Furthermore, integration after deployment at the Point-of-care (PoC) is a cumbersome task since in medical applications there is typically no system integrator with expert-level technical knowledge available [13]. In summary, realizing the aforementioned use cases is cost-intensive and will lead to isolated applications.

To enable *heterogeneous* interoperability, we introduce a future-proof, open and efficient architecture, protocol stack and middleware which is designed to satisfy functional and non-functional requirements on distributed systems of medical devices. The remainder of this paper is structered as follows. The second section illustrates related technologies and research projects. Section 3 describes the underlying conceptual model, requirements and a brief protocol overview. The overall conceptual model is described in Section 4. Sections 5 and 6 introduce the implementation and evaluation of the system. Our work is concluded in Section 7 which also gives an outlook on future work.

2 Related Work

In the past substantial standardization effort and several research projects have been carried out on medical device interoperability. The most popular standard is the ISO/IEEE 11073 (x73) [10]. It is separated into series 11073-1xxxx to 11073-7xxxx, of which the first three are the most important ones. The first part defines fundamentals for all subsequent parts, containing language elements, a nomenclature and an object-oriented Domain Information Model (DIM). The second part describes message exchange patterns between medical devices referring to the upper application layers of the ISO/OSI model. Physical interfaces are described as part of the third serie. However, most often only the DIM and nomenclature part are referenced due to the fact that underlying transport protocols do not

Table 1 Functional requirements (left) and non-functional requirements (right).

Plug & Play	**Risk Management**
Discovery & binding	Safe communication
Device capability description at runtime	Access control
Standardized protocols and open data access	Trust establishment between participants
Communication (1-1, 1-n, n-m)	**Privacy of patient-related data**
Event notification	**Latency in milliseconds**
Data reporting	
Remote control	

support the needs of current distributed systems of medical devices. As of today, x73 is rarely implemented by medical device manufacturers.

Another important work was done by Goldman et al. from the United States as part of the Medical Device "Plug-and-Play" Interoperability Program (MDPnP). They have published the ASTM F2761-1:200 Integrated Clinical Environment (ICE) standard [3]. It defines functional elements for PoC related IT systems, especially focusing on communication of patient data and on equipment command and control. Though the ICE standard gives sophisticated information on conceptual system design, no concrete technical specifications and implementations have been created yet.

There is also much research done in Germany. All roads run together in the OR.NET project [19] that began in September 2012. It combines and consolidates the concepts of predecessor projects to develop a proposal for a new standard called Open Surgical Communication Protocol (OSCP). Foundational predecessor projects are SOMIT FUSION [23], SOMIT OrthoMIT [24], Smart.OR [22], TiCoLi [26], TeKoMed [25] and DOOP [5]. All of these projects propose an architecture based on the idea of a Service-oriented Medical Device Architecture which is described in Section 3. The OASIS standard *Devices Profile for Web Services* serves as the fundamental transport protocol providing TCP/IP and UDP transport bindings, decentralized service discovery and eventing capabilities.

3 SOA for Medical Devices

The middleware presented in this paper follows the principles of the well-known Service-oriented Architecture (SOA) paradigm [15]. Service providers offer their capabilities by means of machine-readable service descriptions and publish them to a service directory. Service consumers can discover these services by using the service directory. Afterwards, they dynamically bind to suitable service providers and invoke their operations.

If SOA principles are applicated on device communication, it is called a Service-oriented Device Architecture (SODA) [4]. In a SODA, services encapsulate both a device's functionality and physical user interface. They are then called device services. In addition to the regular SOA request-response model, publish-subscribe systems are used to transmit information between service providers and consumers. In publish-subscribe systems the communication direction is reversed such that service providers start communicating with service consumers.

If a SODA is applied to a distributed system of medical devices, it is defined as a Service-oriented Medical Device Architecture (SOMDA). Device services are then called medical device services. Middleware systems which implement a SOMDA have certain functional and non-functional requirements to satisfy. Requirements have been acquired by Dräger and are listed in Table 1. Since SOMDA is an abstract concept, implementation directives are

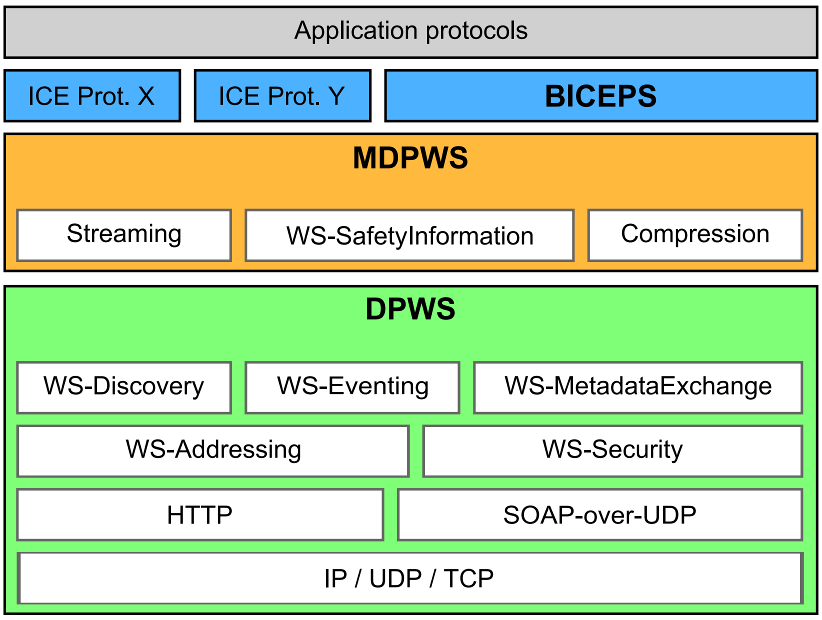


Figure 1 The openSDC protocol stack.

required to get a corresponding middleware up and running. The middleware introduced in this paper is based on Web Services and is called openSDC [18]. Figure 1 shows the protocol stack that is implemented in openSDC.

The most general protocol is the Devices Profile for Web Services (DPWS) [6]. A Web Services profile contains a certain set of specifications and defines appropriate constraints to eliminate protocol ambiguities. DPWS was first invented by Microsoft to interconnect network devices and PCs in a plug-and-play-like fashion. In particular, it was proposed to automatically detect network printers and download suitable drivers. DPWS comes with decentralized service discovery, eventing capabilities and is based on open standards. Hence, it meets the features of a SODA and commonly fulfills the requirements *Plug & Play* and *Communiction (1-1,1-n,n-m)* listed in Table 1.

DPWS is not sufficient to meet the complete set of requirements for a SOMDA. Therefore, openSDC implements a special DPWS dedicated to medical software: the Medical Devices Profile for Web Services (MDPWS). It provides streaming capabilities, error-resistent data transmission and compression options. Besides MDPWS, the Basic Integrated Clinical Environment Protocol Specification (BICEPS) provides a domain-specific protocol to offer generic and extensible device access and modeling. MDPWS and BICEPS are described in detail in the next section.

4 Conceptual Design

A central aspect of the work presented in this paper is the capability to be extensible for future communication needs. For this purpose a layered protocol stack has been developed. As shown in Figure 1, it is separated into three different layers. In conjunction with DPWS, MDPWS builds the transport layer to securely transmit messages. BICEPS introduces an extensible message information model (MIM) and access services to facilitate medical device interoperability. With changing network conditions, MDPWS can be replaced by upcoming, improved or redundant transport protocols, without altering the domain and message model.

4.1 MDPWS

MDPWS is based on DPWS version 1.1 [6]. DPWS became an OASIS standard in June 2009 is used by openSDC to provide Web Services-based features like:

- Service-oriented interaction
- Service discovery using WS-Discovery [16]
- Service description using WSDL [2]
- Publish-Subscribe using WS-Eventing [1]

One major requirement of openSDC is to be extensible for future communication needs. DPWS (and the underlying Web Services technology) is designed for extensibility. But it is an enabling technology rather than a comprehensive communication specification. Hence, DPWS only provides basic features which are constrained and enhanced by MDPWS to enable safe communication in distributed systems of medical devices in high acuity environments. The extended features of MDPWS are explained in the following subsections. In some cases, fundamental knowledge of DPWS may be beneficial.

4.1.1 Liveliness

To assure that a medical device is still active, MDPWS defines to send a directed Probe message as specified in DPWS.

4.1.2 Streaming

A typical use-case in PoC scenarios is the transmission of vital parameters by using waveforms. To enable efficient waveform transmission using Web Services, MDPWS includes SOAP-over-UDP and WS-Streaming. WS-Streaming is part of the MDPWS specification and has not been published yet. It defines a policy to embed descriptive information on streams provided by a Web Service. WS-Streaming does not explicitly define stream management and transport, but provides the capability to announce the existence and type of a stream.

4.1.3 Safety and Security

Any communication protocol dealing with distributed systems of medical devices has to avoid impairment of patient safety. Therefore, remote control mechanisms have to be at least single fault safe. Moreover, another risk control measure is the exchange of a remote invocation context.

Single fault safety can typically be met by providing dual channel transmission. To remotely modify device parameters, invocation information is sent redundantly over two independent channels. The second channel might be separated in time or representation. In time means that the second channel is established temporally after the first channel has been transmitted. The drawback of separation in time is that it requires stateful services and double transmission costs. Separation in representation is achieved by providing both channels in a single message. The service provider detects failures, for example, by means of an invalid checksum. Figure 2 illustrates the dual channel transmission in accordance to separation in representation. Data is given to the middleware of the service consumer by means of two input objects. The middleware serializes and deserializes one input object and compares the result to the second object. This implicates that parser end serializer are working correctly. The serialized object is transmitted to the service provider. On the

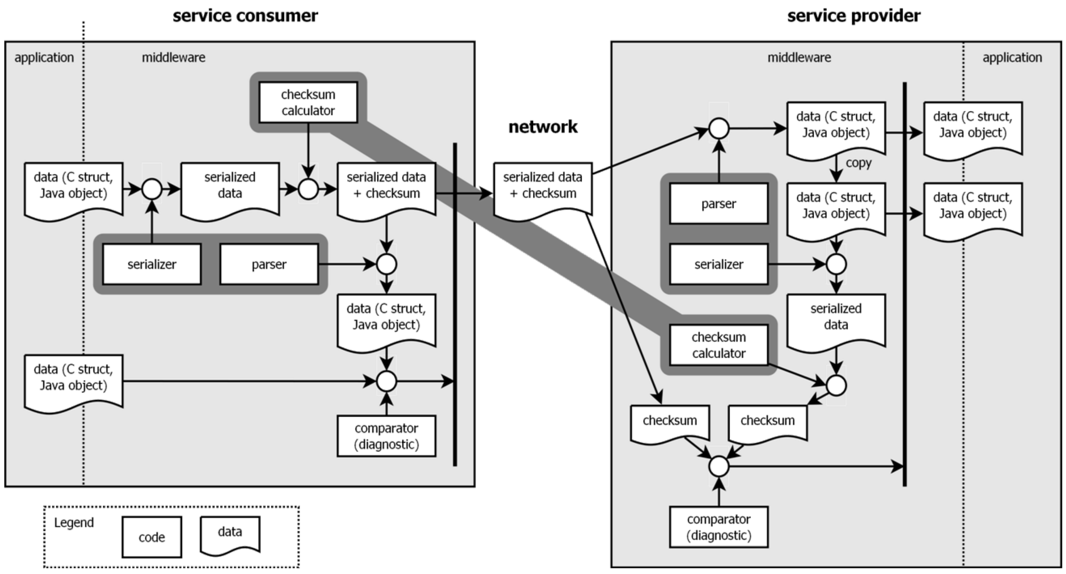


Figure 2 Dual channel transmission using separation in presentation.

provider side, the parsing process and the data duplication generating the second channel are controlled. In summary, this guarantees that data is not compromised when being transmitted from one process to another. More information on dual channel transmission is given in [20].

A remote invocation context is also called a safety context. The service consumer needs to know which information is required by the service provider to transmit a remote invocation context. If this knowledge is available to the service consumer, it can append context information to the service invocation message. An example for a safety context is the value of a setting an operation is applied to. To enable dual channel transmission and safety contexts, WS-SafetyInformation has been specified as part of MDPWS. It is not standardized yet.

4.1.4 Security

Besides patient safety, access control, integrity and confidentiality are major security goals. Only authorized service consumers should have remote access to devices. Patient data should be secured by using encryption. MDPWS provides capabilities to meet the aforementioned requirements. It includes a Public Key Infrastructure (PKI) to gain authorization capabilites using X.509 certificates. If access control is needed, it is handled on the level of individual security principles using the PKI. MDPWS specifies that a service provider may control access to a service by HTTPS with mutual authentication. This also applies to WS-Eventing services.

MDPWS specifies that a X.509 certificate is issued to a service provider's Universally Unique Identifier (UUID) [11]. The same is prescribed to service consumers. Authorization is enabled by using a Device Access Control List (DACL) which contains subject identities of security principals and associated granted access rights. In order to be able to handle groups of security principals, DACL provides group subject identities which are used to grant access to, for example, device types or devices of certain manufacturers. In order to ensure decentralization, devices have to maintain root certificates of the manufacturers for every device they are communicating with. Another solution is to get a separated DACL which is signed by the clinical operators to verify digital signatures of communication partners. So far, devices of a dedicated manufacturer trust every other decvice of the same manufacturer.

To ensure confidentiality and integrity of messages, MDPWS states that HTTPS should be applied. SOAP-over-HTTPS may also be used. Since SOAP-over-UDP is used to transmit anonymous streaming data, no security mechanism has been defined by MDPWS.

4.2 BICEPS

BICEPS specifies a MIM and access services for the domain of distributed medical devices. It is based in large parts on the x73 DIM as described in Section 2 and forms a minimal set of generic functionality and messages to facilitate interoperability and extensibility. A core component of x73 and even BICEPS is the Medical Device Information Base (MDIB). It represents an object-oriented model to encapsulate managed medical objects. Managed medical objects are best known as physiological patient data, device configuration data, or remote invocation operations.

4.2.1 MIM

Communication messages exchanged in distributed systems of medical devices contain state data about clinical measurements of a patient or the device associated with a patient. Moreover, remote invocation commands might also be transmitted. To enable interoperability, medical devices have to exchange meta-information about state data as well as contextual information that describes in which context state data has been acquired. Such information is described in the BICEPS MIM. It provides two parts: communication message definitions and the MDIB component. For the sake of brevity the communiction messages are not described in this paper. Basically, for every service request and response, and even for every notification, BICEPS-MIM defines a dedicated message payload.

The MDIB in turn is divided into a descriptive part and a state part. The descriptive part holds information on a device's structure, services and metrics, and provides coded values. Coded values allow to characterize object types by referencing a coding system with a version and code identifier. By means of coded values communication partners are able to interpret transmitted data, for example, the unit of a measurement value. Since every managed medical object is equipped with a coded value, they can be semantically interpreted. On the other side, the state part holds the content data that a medical device can deliver. It should be noted that both parts of the MDIB can change during runtime.

A top level overview of the BICEPS-MIM MDIB descriptive part is given in Figure 3. A Medical Device System (MDS) is an abstract representation of a physical device that exposes its capabilities as a medical device service. In accordance to x73 it is depicted as an HydraMDS and may contain multiple Virtual Medical Devices (VMDs). A VMD is a representation of a sub-system of a MDS. It may in turn contain multiple channels. Channels refer to a group of metrics, whereby metrics are abstract representations of measurement values, settings or status items. BICEPS-MIM specifies by default numeric values, textual values and sample arrays of numeric values. Nevertheless, it can be extended to any type of value. MDSs, VMDs and channels can be assigned with an optional alert system. It detects alert conditions and may signal them by means of alert signals. Alert signals are in most cased displayed visually or acousticly.

Another part of the model in Figure 3 is the Service Control Object (SCO). It comprises remote invocation capabilities. This includes affected objects and Quality of Service (QoS)

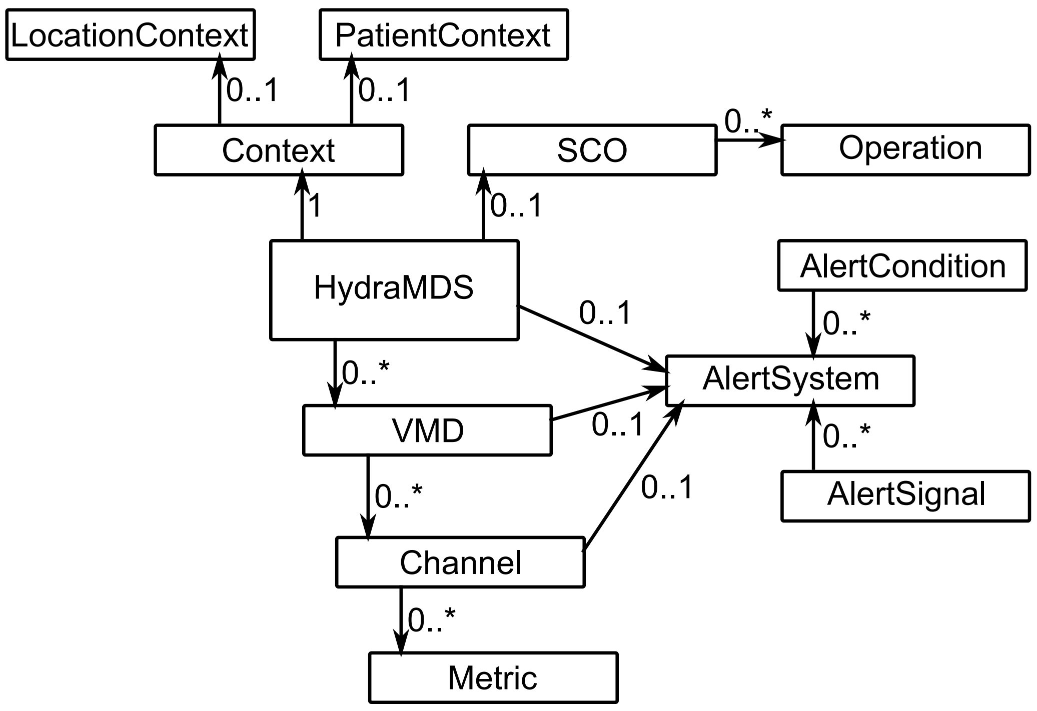


Figure 3 Top level overview of the BICEPS-MIM MDIB descriptive part.

parameters. There are different operation types to offer several set methods, non-generic method calls or remote control calls. The Context element represents the context the underlying MDS is currently working in. This context is designated by patient or location information.

4.2.2 Service interface

To get remote access to managed medical objects, x73 specifies a generic service interface called CMDISE. In a slightly different way this interface is also provided by BICEPS. It contains a generic get service to request data, a generic set service to manipulate data, a dedicated waveform service to transmit, for example, vital signs, a protected health information (PHI) service to request or set patient related information in a secured manner, and an event report service to enable publish-subscribe data retrieval. Figure 4 depicts the differences between CMDISE and BICEPS services. While x73 separates SET and ACTION, BICEPS merges them to a single service. BICEPS omits CREATE and DELETE since they provide extended functionality in terms of allowing to manipulate device memory. If CREATE and DELETE are required in sophisticated scenarios, they can later be implemented by BICEPS' extensibility mechanism. Waveform and PHI are extra services provided by BICEPS. Finally, EVENT REPORT is mapped from x73.

Figure 4 illustrates a coarse-grained service interface. BICEPS uses numerous operations grouped together to build GET, SET, Waveform, PHI and EVENT REPORT. GET is separated into:

- **GetMDIB:** retrieval of the full MDIB including descriptive and state part
- **GetMDDescription:** retrieval of the MDIB descriptive part
- **GetMDState:** retrieval of the MDIB state part

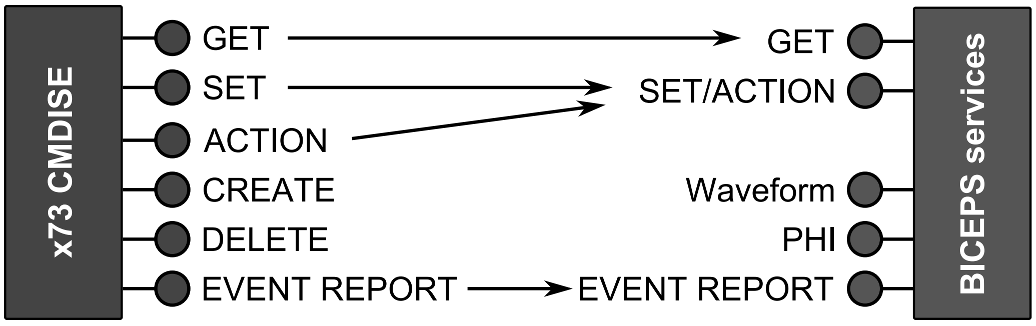


Figure 4 Coarse-grained mapping of Common Medical Device Information Service Element (CMDISE) and BICEPS service interfaces.

SET is characterized by:

- **SetValue:** sets a numeric metric
- **SetString:** sets a string metric
- **SetRange:** sets the range of a numeric metric
- **Activate:** executes remote control

EVENT REPORT is the capability to retrieve notifications about either changes of the descriptive part or the state part. The EVENT REPORT service is divided into

- **MetricReports:** notification of metric changes
- **AlertReports:** provides alert events
- **ContextReport:** notifies when context changes
- **OperationInvokedReport:** since operation calls are queued, this notification provides operation progress information
- **MDSCreatedReport:** notifies if a MDS appears to be available for access
- **MDSDeletedReport:** notifies if a MDS disappears and is no longer active
- **ObjectCreatedReport:** notifies on any object creation events
- **ObjectDeletedReport:** notifies on any object deletion events

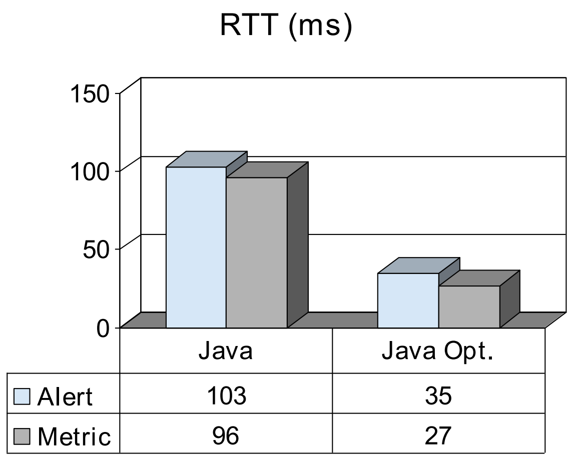
There is no definition on how to subscribe to notifications. BICEPS states that the underlying transport protocol has to support subscription management.

Waveform is an optional service that defines an interface to retrieve a stream for real-time sample array metrics. There is only a waveform stream manifested in BICEPS even without specifying means to subscribe to it. BICEPS states that the underlying transport protocol has to support subscription management.

Like Waveform, PHI is also an optional service. It allows retrieval and modification of patient data. Due to privacy reasons this service is separated from other BICEPS services. Hence, it can separately be protected.

Besides the generic service interface medical devices might offer remotely invokable operations that are not defined as part of the BICEPS-MIM / BICEPS services. For this purpose a non-generic operation descriptor can be defined in the descriptive part of the MDIB. It facilitates non-generic, proprietary service calls to be represented by a coded value. By using the non-generic operation descriptor, service calls are made accessible through the MDIB.

Finally, BICEPS does not define any QoS requirements, but provides extensibility points to embed QoS requirements for a transport protocol.



	Java	Java Opt.
☐ Alert	103	35
▨ Metric	96	27

Figure 5 Maximum round trip time performance measurements for openSDC using JRE6's default (Java) and optimized (Java Opt.) environment.

5 Implementation

BICEPS and MDPWS have been released as part of openSDC [18] and can be downloaded and used for free. OpenSDC serves as a reference implementation for vendors of medical devices, but is not intended to be used in clinical trials, clinical studies or in clinical routine.

The openSDC project uses a modified version of the WS4D JMEDS stack [27] to gain DPWS functionality. The modified JMEDS stack is called JDPWS. The remaining components of the openSDC stack implement the BICEPS and MDPWS protocols. It allows either a contract-first approrach by defining the MDIB in a XML file and load it into the framework, or a code-first approach by building the MDIB in Java code.

6 Evaluation

To get an impression on how stable and powerful openSDC is, performance measurements have been made using a Java JRE6. A client (2x2 GHz) requests metric values and alerts twice a second. The device (1x1.1 GHz) responds with 276 metric values and 80 alerts. Additionally, 10 waveform frames containing at least 20 values are sent out at 200 ms intervals.

The first measurement is the maximum round trip time (RTT) to deliver an alert and a numeric metric. It is depicted in Figure 5. When using Java with its default settings, openSDC causes a maximum RTT of 103 ms to post an alert and 96 ms to post a metric. By optimizing the JRE using Java's environment properties, open SDC is on average 69 percent faster. Figure 6 shows the average and maximum framerate when transmitting a waveform. In this scenario, only optimized Java has been tested. With an average framerate of 199 frames per second, openSDC's Web Service Streaming is fast enough to build continuous 1-dimensional signals.

In addition to the previously mentioned simple performance measurements, openSDC has been tested with real world medical devices as part of a public project workshop in December 2013. Figure 7 gives an idea of the demonstration setup. Participating devices were a Möller-Wedel operating microscope, a Localite navigation system, a Dräger monitoring device, a Söring ultra sound surgery device and an Olympus documentation system.

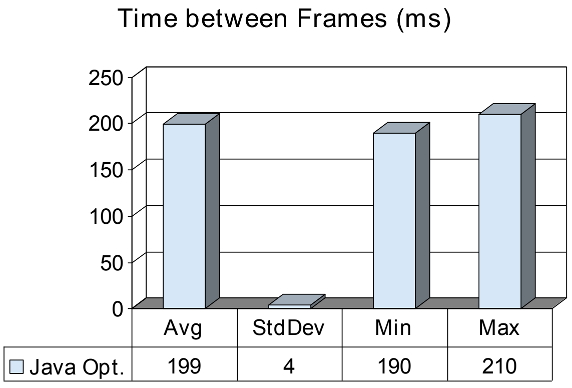


Figure 6 Maximum waveform framerate for openSDC using JRE6's optimized (Java Opt.) environment.

Different use cases have been introduced to prove the practical effect of openSDC. First, patient data arrived at the documentation system by fetching HL7 ADT messages. Afterwards, this data was automatically stored at the microscope and navigation system side using DPWS. Next, BICEPS was used to transmit vital signs from the patient monitor to the operating microscope to display health information. Moreover, it was applied to remotely control the ultrasound surgery device. By sending activation requests, the operating microscope was able to control cutting. The navigation system retrieved zoom and focus information from the operating microscope. Thereby, the navigation system was able to select proper views of the area currently navigated in.

In all scenarios openSDC provides immediate network responses confirming the measurements in Figure 5. Especially, there was no remarkable delay when remotely controlling the ultra sound activation. Therefore, the system is fast enough to remotely control devices where controlling actions have to be initiated and recognized by a human.

7 Conclusion and Future Work

In this paper, we have presented openSDC, a protocol stack which enables heterogeneous interoperability of medical devices. Based on a set of standards, most prominently including the Device Profile for Web Services (DPWS), we presented in detail the MDPWS protocols for message transport in medical environments and BICEPS which is used as an application protocol for service access. An openly available implementation of openSDC exists, and first evaluations show promising results.

We see future work mostly in three directions: First, BICEPS will be enhanced by extending its functionality through providing more plugins. Second, more clinical use cases will be described and show-cased. Finally, we are working on a seamless integration of openSDC with clinical IT systems. Especially authentication and authorization will be a major issue the solution of which will make the system much easier and more secure to use.

References

1 Don Box, Luis Felipe Cabrera, Craig Critchley, Francisco Curbera, Donald Ferguson, Steve Graham, David Hull, Gopal Kakivaya, Amelia Lewis, Brad Lovering, Peter Niblett, David Orchard, Shivajee Samdarshi, Jeffrey Schlimmer, Igor Sedukhin, John Shewchuk, Sanjiva

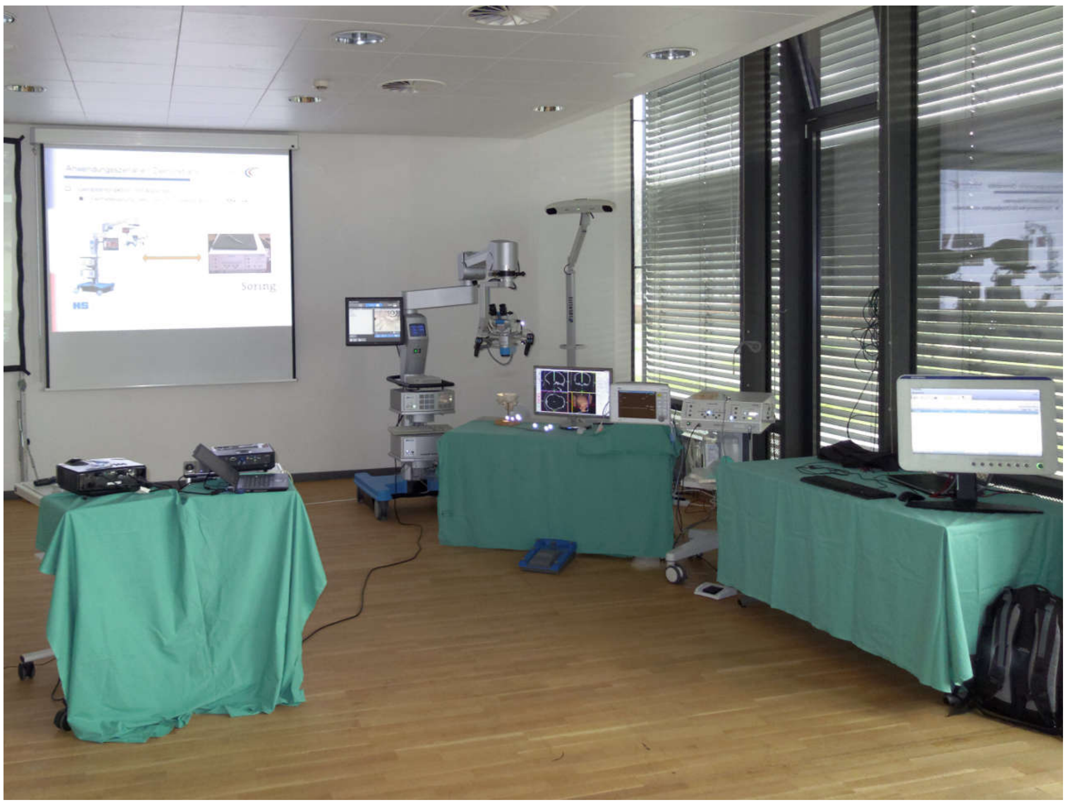


Figure 7 Image of an openSDC powered distributed system of medical devices at a project workshop in December 2013. F.l.t.r.: operating microscope, navigation system, patient monitor, ultrasound surgery device and documentation system.

Weerawarana, and David Wortendyke, editors. *W3C Member Submission: Web Services Eventing (WS-Eventing).* World Wide Web Consortium (W3C), July 2006.

2 Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, editors. *Note: Web Services Description Language (WSDL) 1.1.* World Wide Web Consortium (W3C), March 2001.

3 Committee F29.21 on Devices in the Integrated Clinical Environment. ASTM F2761 – 09 Medical Devices and Medical Systems – Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) – Part 1: General requirements and conceptual model. ASTM International, 2009.

4 Scott de Deugd, Randy Carroll, Kevin Kelly, Bill Millett, and Jeffrey Ricker. SODA: Service Oriented Device Architecture. *IEEE Pervasive Computing*, 5(3):94–96, c3, 2006.

5 DOOP project. `http://www.doop-projekt.de/`.

6 Dan Driscoll and Antoine Mensch, editors. *OASIS Standard: Devices Profile for Web Services Version 1.1.* OASIS, July 2009.

7 Julian M. Goldman. Medical Devices and Medical Systems – Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) – Part 1: General requirements and conceptual model. ASTM International, `http://www.mdpnp.org/uploads/F2761_completed_committee_draft.pdf`, 2008.

8 Healthcare Information and Management Systems Society – HIMSS. What is Interoperability? `http://www.himss.org/library/interoperability-standards/what-is`, April 2013.

9 IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries, January 1991.

10 ISO/IEEE. ISO/IEEE 11073: Health informatics – Point-of-care medical device communication, June 2004.

11 ITU International Telecommunication Union. Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components, September 2004.

12 Karl Storz. OR1. `https://www.karlstorz.com/cps/rde/xchg/SID-949FE9D0-512F111A/karlstorz-en/hs.xsl/522.htm`.

13 B. Larson, J. Hatcliff, S. Procter, and P. Chalin. Requirements specification for apps in medical application platforms. In *Software Engineering in Health Care (SEHC), 2012 4th International Workshop on*, pages 26–32, June 2012.

14 Kathy Lesh, Sandy Weininger, Julian M. Goldman, Bob Wilson, and Glenn Himes. Medical Device Interoperability-Assessing the Environment. In *Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability (HCMDSSMDPnP)*, pages 3–12, Cambridge, Massachusetts, USA, June 2007. IEEE Computer Society.

15 C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz, editors. *OASIS Standard: Reference Model for Service Oriented Architecture 1.0*. OASIS, October 2006.

16 OASIS. Web Services Dynamic Discovery (WS-Discovery) Version 1.1, 2009.

17 Olympus. ENDOALPHA. `http://www.olympus-europa.com/medical/en/medical_systems/products_services/systems_integration/productselector_service_solutions_8.jsp`.

18 openSDC Website. `https://sourceforge.net/projects/opensdc/`.

19 OR.NET. OR.NET | Projektwebsite. `http://www.ornet.org`.

20 Stephan Pöhlsen, Winfried Schöch, and Stefan Schlichting. A Protocol for Dual Channel Transmission in Service-Oriented Medical Device Architectures based on Web Services. In *3rd Joint Workshop on High Confidence Medical Devices, Software, and Systems & Medical Device Plug-and-Play Interoperability*, 2011.

21 Andreas Schweiger, Ali Sunyaev, Jan Marco Leimeister, and Helmut Krcmar. Toward Seamless Healthcare with Software Agents. *Communications of the Association for Information Systems (CAIS)*, pages 692–709, 2007.

22 smartOR project. `http://www.smartor.de/`.

23 SOMIT FUSION project. `http://www.somit-fusion.de/SF/`.

24 SOMIT orthoMIT. `https://www.vde.com/de/fg/DGBMT/Arbeitsgebiete/Projekte/Seiten/SOMIT-orthoMIT.aspx`.

25 TeKoMed project. `http://kosse-sh.de/projekte/tekomed/`.

26 TiCoLi. `http://www.iccas.de/ticoli/`.

27 Web Services for Devices (WS4D). JMEDS (Java Multi Edition DPWS Stack). `https://sourceforge.net/projects/ws4d-javame/`.

Simulations of the Cardiovascular System Using the Cardiovascular Simulation Toolbox

Gabriela Ortiz-León[1], Marta Vílchez-Monge[2], and Juan J. Montero-Rodríguez[3]

1 **Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica Cartago, Costa Rica**
gaby@itcr.ac.cr

2 **Escuela de Física, Instituto Tecnológico de Costa Rica Cartago, Costa Rica**
mvilchez@itcr.ac.cr

3 **Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica Cartago, Costa Rica**
jjmontero@itcr.ac.cr

Abstract

In the present document, six mathematical models of the cardiovascular system are studied and implemented in MATLAB® R2013a using an updated version of the Cardiovascular Simulation Toolbox proposed by O. Barnea at the Tel-Aviv University. All the mathematical models are based on electrical lumped-parameter analogies. The results of the simulations are compared with a list of expected hemodynamic parameters and contrasted with laboratory values.

1998 ACM Subject Classification I.6.4. Model Validation and Analysis

Keywords and phrases Biomedic, Cardiovascular, MATLAB, Simulation

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.28

1 Introduction

Simulation with lumped-parameter models is one of the traditional approaches to model the cardiac system. After the development of the Windkessel models by O. Frank in 1899 [6], several mathematical models based on equivalent circuits have been proposed. One of the efforts to build a complete set of equivalent models for cardiovascular simulations has been made by O. Barnea *et al.* [1] at the Tel-Aviv University. His team developed a Cardiovascular Simulation Toolbox for MATLAB® R14. This open-source tool is conformed by a set of 21 individual lumped-parameter electrical models. Complex models of the circulatory system can be achieved by interconnecting blocks and signals.

Barnea's library of models was no longer supported by the recent releases of MATLAB® because the original toolbox was built using the Power Systems Blockset™, and it was replaced by the SimPowerSystems™ module for Simulink®. In a previous publication [11] we showed our efforts to update Barnea's toolbox to the newer releases of MATLAB®, and published an updated version currently working in MATLAB® R2013a.

In this document, we implement six mathematical models in MATLAB® R2013a using this updated version of the Cardiovascular Simulation Toolbox, in order to obtain the hemodynamic parameters of healthy persons. The numerical results of this work are compared with reference hemodynamic parameters obtained at specialized laboratories [10] [7] [4]. The most important parameters used to compare simulations in this work are the cardiac output (CO), the blood flow or aortic flow (AQ) and the arterial pressure (AP).

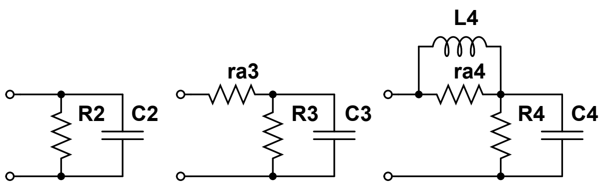


Figure 1 The Windkessel models of second, third and fourth order.

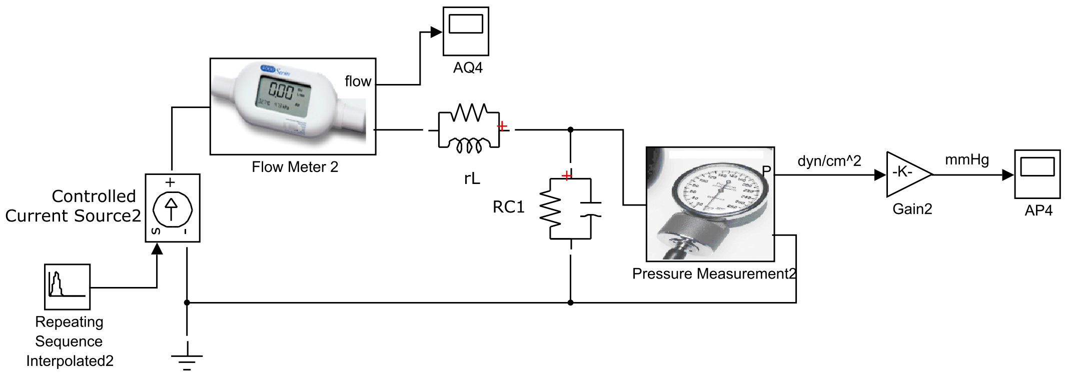


Figure 2 Windkessel model of fourth order, implemented in Simulink® using blocks from the Cardiovascular Simulation Toolbox [15].

1.1 Windkessel models (WK)

The first Windkessel model model was proposed by O. Frank in 1899 [6]. The simplest model is the second order Windkessel model (WK2) and includes a single resistor R and a single capacitor C that are the equivalent lumped parameters of the systemic circulatory system. On 1930 Ph. Broemser and O. Ranke [2] added a third circuit element r used to describe the input impedance of the aorta. This model is called the third order Windkessel model (WK3). The final model of four elements (WK4) was proposed to improve the dynamic response of the circuit and it includes an inductance L, describing the inertia of blood. The circuit representations of the Windkessel models are shown in Figure 1.

To simulate this model, the input flow of the left ventricle is required. We assumed a sinusoidal pulsing function, as described in [8]. The model calculates the mean arterial pressure (MAP), to calculate the cardiac output.

The models of Figure 1 are implemented in Simulink® using different blocks from the Cardiovascular Simulation Toolbox [15] and some extra blocks from the SimPowerSystems™ library. The block diagram of the fourth-order Windkessel model using Barnea's toolbox is presented in Figure 2.

The simulation results of the three Windkessel models are presented in Figure 3, which is a graphic describing the arterial pressure as a function of time. Constants required for the parameters of the elements are from M. Hlaváč [8].

Based on the results from Figur 3 the mean arterial pressure is calculated, and then divided by the total peripheral resistance (TPR) in order to obtain the mean aortic flow (MAQ). The cardiac output is calculated multiplying by 60. Results of these calculations are presented on Table 1 with the results of the other models.

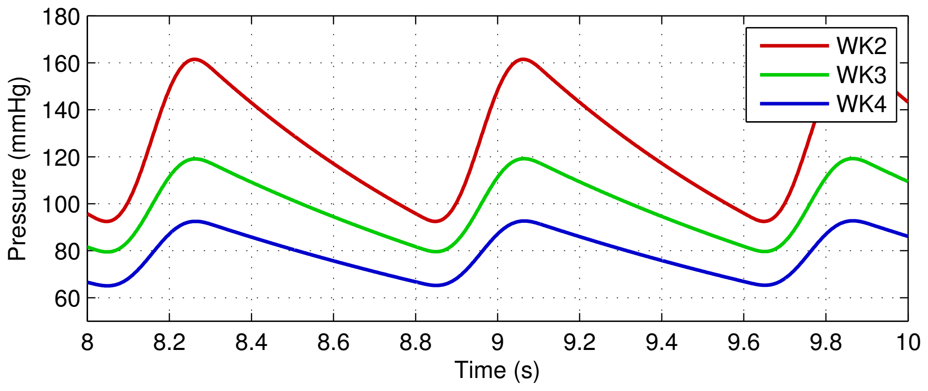


Figure 3 Arterial pressure as a function of time for the Windkessel models.

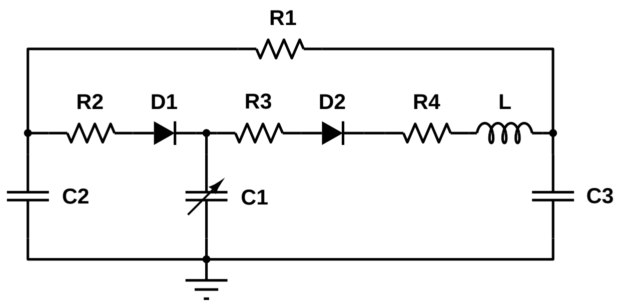


Figure 4 Circuit diagram for A. Ferreira model [5].

1.2 Lumped-parameter model of A. Ferreira

The model of A. Ferreira [5] is shown in Figure 4. The left ventricle C1 is described with a variable capacitor, because the walls of the heart chambers are elastics, and the elastance changes over time across the cardiac cycle. The left atria is described by the capacitance C2. The peripheral circulatory system is described by R1 and C3. Also the model includes the mitral valve, described by D1 and R2, and the aortic valve formed by D2 and R3. The resistor R4 models the input impedance of the aorta, and the inductance L describes the inertia of blood. Mitral valve and aortic valve are described with thyristors, switched with control signals to enable or block the blood flow in one direction.

To implement this model in Simulink® a new block was designed for the Cardiovascular Simulation Toolbox. The block is a variable non-linear capacitor, where the capacitance can be adjusted using an external elastance function. This elastance function is described by A. Ferreira *et al.* [5] and it is composed of exponential functions. The new block is based on the existent polynomial capacitor, but a multiplier was added and the polynomial block removed, in order to include the external signal that modulates the compliance of the block.

The model of A. Ferreira calculates the instantaneous arterial pressure in the same way as the Windkessel models, and then calculates the mean arterial pressure and the cardiac output. In this simulation, cardiac output has a value of 4.64 L/min. This value is included in Table 1 with the other results of this simulation.

The Simulink® implementation of the electrical model of A. Ferreira is presented in Figure 5. Simulation results for the model of A. Ferreira can be appreciated in Figure 6.

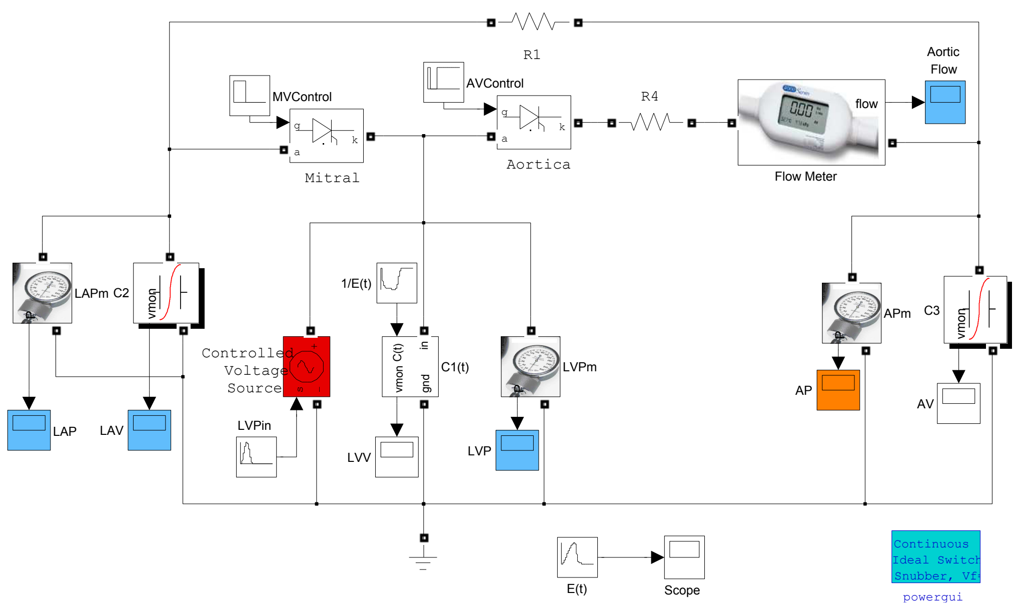


Figure 5 Simulink® implementation of the electrical model of A. Ferreira.

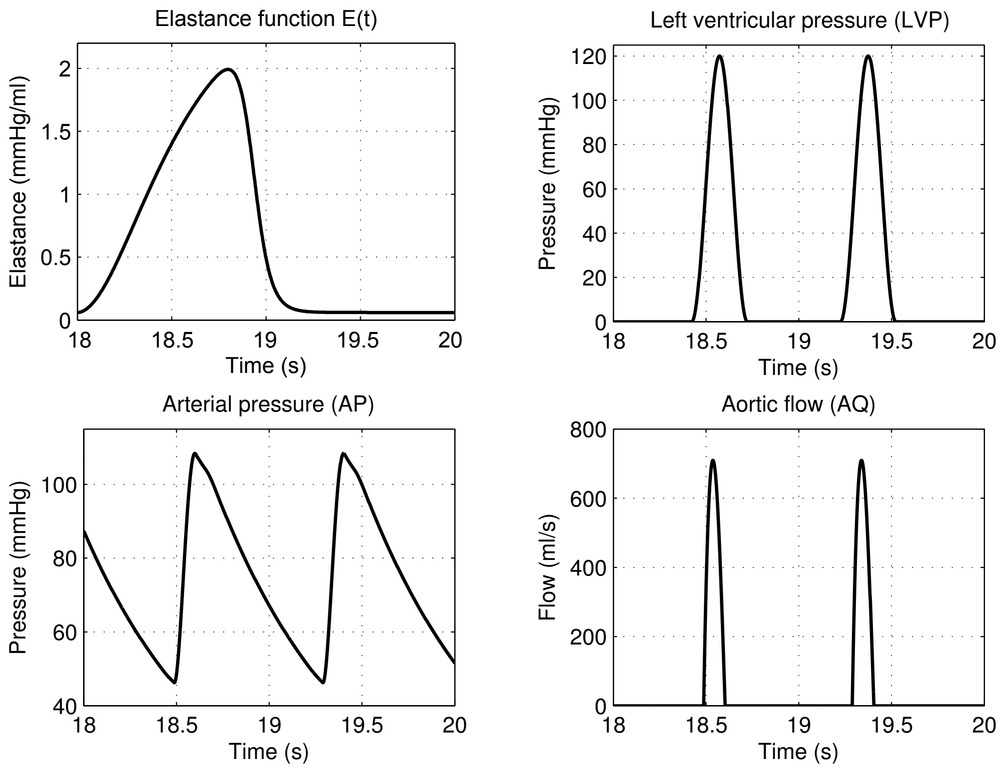


Figure 6 Simulation results of A. Ferreira model.

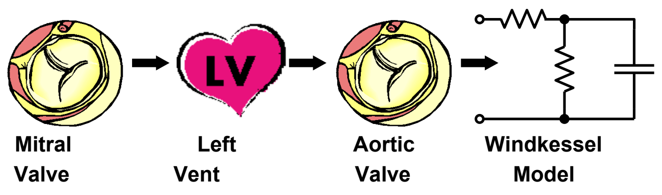


Figure 7 Block diagram of the Windkessel model with the left ventricle [15].

1.3 Windkessel model coupled to left ventricle (WK+V)

The main problem with the Windkessel models is that they require information about the flow generated by the left ventricle. In many simulations this flow is approximated by a sinusoidal representation that does not reproduce exactly the waveform of real blood flow, and this represents an error source. In order to improve the accuracy, this model simulates the left ventricle using the varying elastance model, and couples the generated flow to a third-order Windkessel model to obtain the final results of mean arterial pressure and cardiac output. The block diagram of this model is shown in Figure 7.

The Windkessel model coupled to the left ventricle (WK+V) is similar to A. Ferreira's model because it uses an elastance function to describe the left ventricle, and connects it to a third order Windkessel model. The difference is that the left ventricle is characterized by a third order polynomial elastance function instead of the exponential function of Ferreira's model. This simulation is included as an example of the Cardiovascular Simulation Toolbox and here is implemented in MATLAB® R2013a. The implementation of this model is shown in Figure 8. Simulation results for this mathematical model are shown in Figure 9.

1.4 Systemic and pulmonary circulation model (2A2V)

This model proposed by O. Barnea *et al.* [15] in the Cardiovascular Simulation Toolbox includes the systemic and pulmonary circulation, both atria, ventricles and the four cardiac valves. This model also simulates the oxygen saturation in blood across the sections of the cardiovascular system. The model calculates the pressure and volume at multiple points and the pressure-volume loop can be obtained by plotting the pressure and the volume of the left ventricle.

We used this model to obtain the hemodynamic parameters of a healthy person, considering a left-ventricular ejection fraction of 58.7% as suggested by Schlosser [14]. Adjusting the physical parameters of the atria and ventricles, we were also able to simulate the behavior of the cardiovascular system for a person suffering from systolic heart failure, with a left-ventricular ejection fraction of 24.6% [3].

Lumped-parameter models of the cardiovascular system in MATLAB® can be coupled with three-dimensional models, described in COMSOL® and simulated with finite element methods (FEM). Some examples of this coupling can be found in [12, 13].

The block diagram of the systemic-pulmonary model can be appreciated in Figure 10.

This model enables to obtain the pressure-volume loops for a healthy person, and also for a person suffering from systolic heart failure, as can be appreciated in Figure 11. The graphic from the left is obtained by running the default simulation of the model provided by O. Barnea, and it shows the cardiac cycle with the typical values expected for the subject. The complete set of parameters resulting from this simulation of the healthy system has been published in [11].

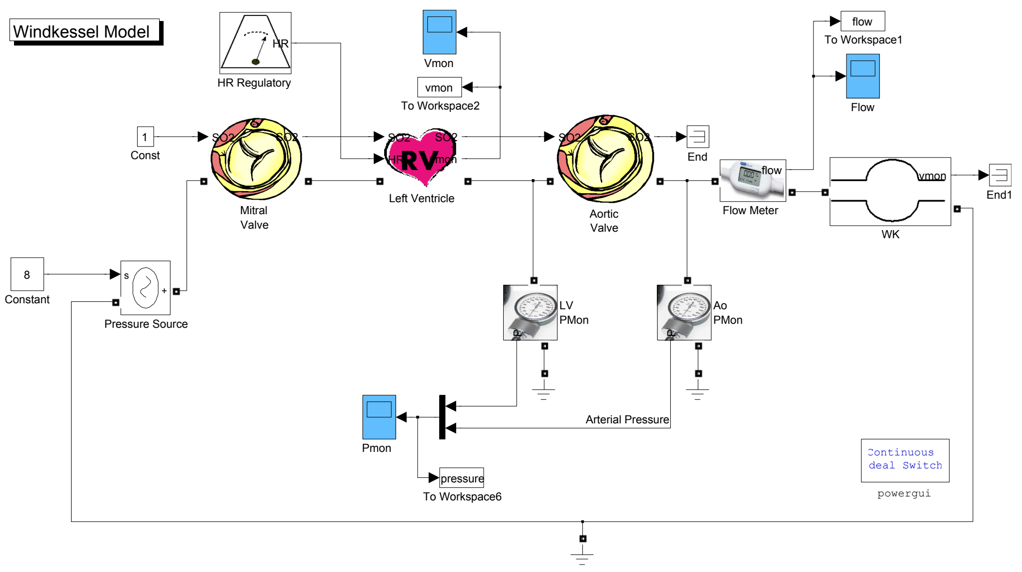


Figure 8 Simulation of the left ventricle coupled to a third order Windkessel model [1].

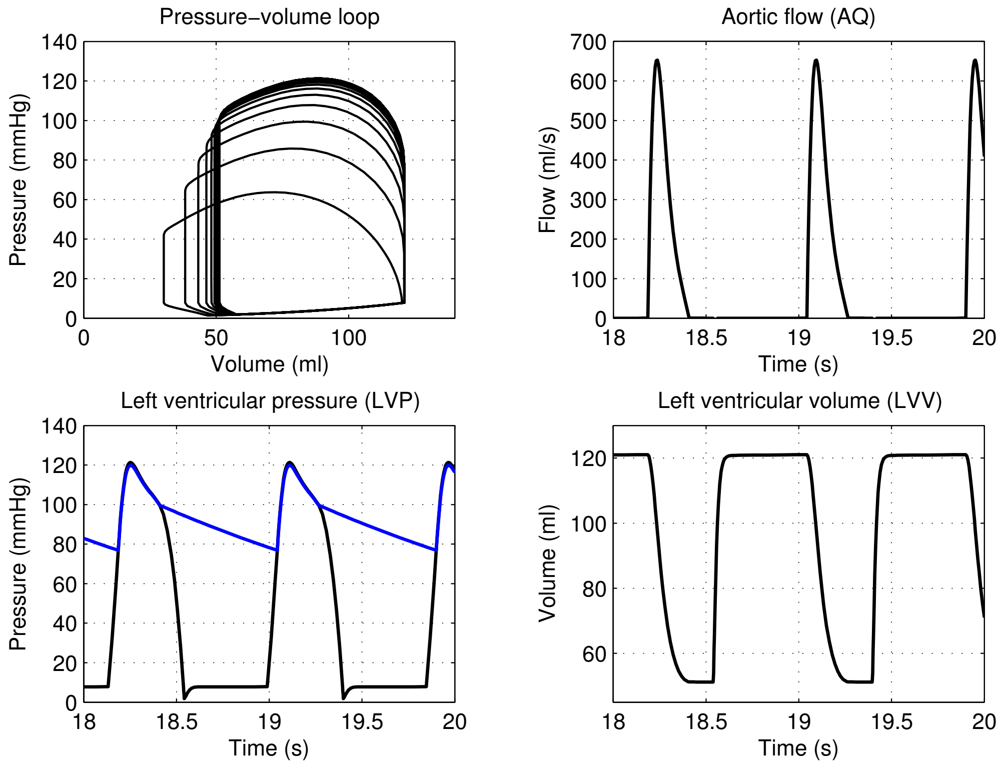


Figure 9 Simulation results of the WK+V model.

Table 1 Results of the simulation of six mathematical models of the cardiovascular system, implemented with blocks from the Cardiovascular Simulation Toolbox and the SimPowerSystems^{TM} toolbox.

Variable	WK2	WK3	WK4	WK+V	Ferreira	2V2A	Reference value
HR	75,00	75,00	75,00	70,00	75,00	75,00	48 – 105 [16]
CO	7,49	5,91	4,71	4,89	4,64	6,71	4.0 – 8.0 [4]
SV	99,87	78,80	62,80	69,81	61,87	89,45	60 – 100 [4]
MAP	124,83	98,54	78,49	-	-	116,83	70 – 105 [10]
SBP	161,49	119,25	92,76	119,86	108,39	143,89	90 – 140 [10]
DBP	92,50	79,60	65,23	76,87	46,24	68,35	60 – 90 [10]
LVEDV	-	-	-	121,00	-	152,47	65 – 239 [14]
LVESV	-	-	-	51,20	-	63,012	16 – 143 [14]
LVP	-	-	-	121,30	120,00	147,19	140 [9]
LVEf	-	-	-	57,69	-	58,67	59,2 ± 13,7 [14]
RVEDV	-	-	-	-	-	152,85	100 – 160 [10]
RVESV	-	-	-	-	-	78,92	50 – 100 [10]
RVEDP	-	-	-	-	-	40,12	15 – 25 [10]
RVESP	-	-	-	-	-	2,17	0 – 8 [10]
RVEf	-	-	-	-	-	48,37	40 – 60 [10]

In order to simulate the cardiovascular system of a person suffering from systolic heart failure, we proceeded to decrease the elastances of the heart chambers, and increase the effective resistance of the cardiac valves, to describe a heart that cannot eject blood with the same effectiveness as a regular heart. This hardening of the heart was done by adjusting the model parameters in Simulink. The complete set of output parameters has been published also in [11].

Comparing the two diagrams shown in Figure 11, it can be observed that, when the person has a medical condition, the volume of blood in the left ventricle tends to increase, because the heart cannot eject the same volume per beat (the stroke volume is reduced). The stroke volume can be calculated from the plots by subtracting the maximum and minimum volumes. It can be appreciated that the stroke volume for the healthy simulation is higher than the stroke volume for the systolic heart failure.

The models also enable the verification of medical devices such as a intra-aortic balloon or a ventricular assist device. Any device model can be developed and coupled to this simulation, to study and observe the changes of the hemodynamic parameters as the response of the body after the medical device implantation. We have developed a dummy VAD block, as seen in Figure 10, but further development is required.

Table 1 includes the simulation results for the six mathematical models studied in this document. The list of output parameters consists of the following: Heart Rate (HR, bpm), Cardiac Output (CO, ml), Stroke Volume (SV, ml), Mean Arterial Pressure (MAP, mmHg), Systolic Blood Pressure (SBP, mmHg), Diastolic Blood Pressure (DBP, mmHg), Left Ventricular End-Diastolic Volume (LVEDV, ml), Left Ventricular End-Systolic Volume (LVESV, ml), Left Ventricular Pressure (LVP, mmHg), Left Ventricular Ejection Fraction (LVEf, %), Right Ventricular End-Diastolic Volume (RVEDV, ml), Right Ventricular End-Systolic Volume (RVESV, ml), Right Ventricular End-Diastolic Pressure (RVEDP, mmHg), Right Ventricular End-Systolic Pressure (RVESP, mmHg) and Right Ventricular Ejection Fraction (RVEf, %).

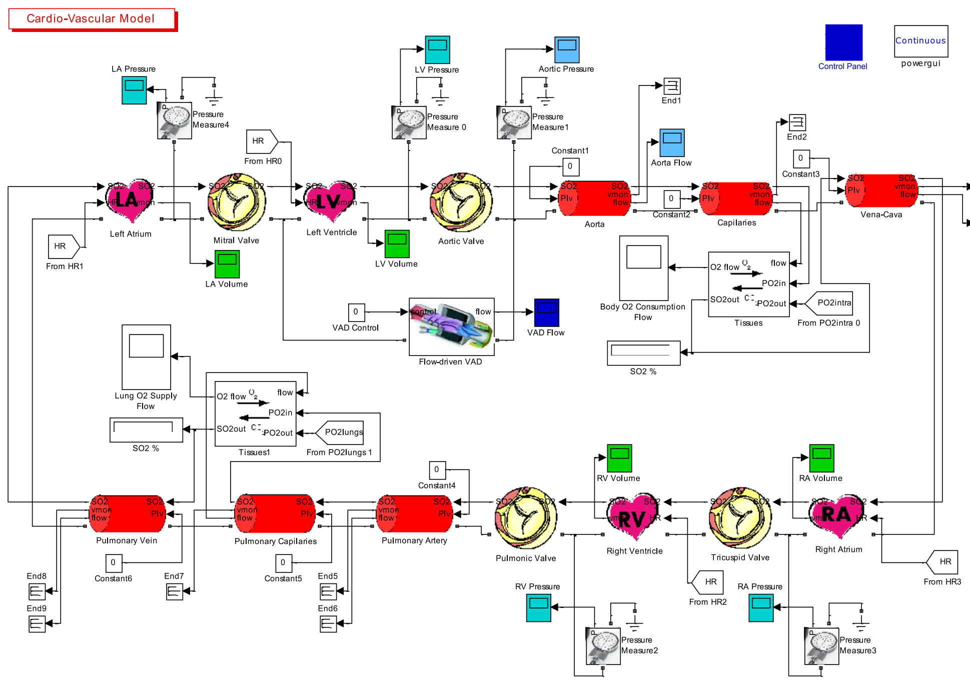


Figure 10 Block diagram of the complete circulatory system using the Cardiovascular Simulation Toolbox [15]. This model considers systemic and pulmonary circulation, coupled to the four chambers of the heart.

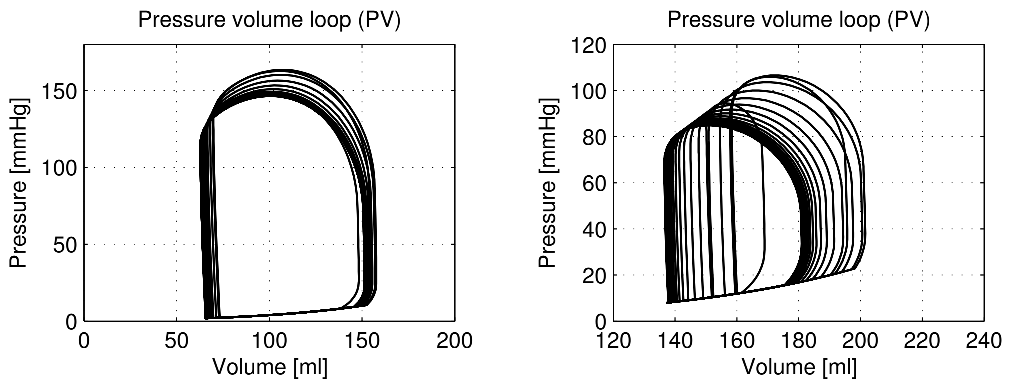


Figure 11 Pressure-volume loops for the left ventricle in the 2A2V circulatory model, in normal health conditions (left, LVEf=58.7%) and with systolic heart failure (right, LVEf=24.6%).

2 Analysis

Lumped-parameter electrical models of the cardiovascular system are an appropriate method to obtain several hemodynamic parameters of the circulation, and are a commonly used approach in many simulations.

Windkessel models of order 2, 3 and 4 are faster to produce results that the other models, and they require less computational power. These models are a good approximation to obtain the most general values and understand the basic behavior of the peripheral circulatory system, but they provide only the cardiac output and the arterial pressure.

The most versatile models are implemented with the Cardiovascular Simulation Toolbox because it presents excellent modularity and it is expandable with custom blocks. This characteristic permits to modify existent models in order to describe and simulate cardiac and circulatory diseases. The toolbox also can simulate existent lumped-parameter electrical models, such as the Windkessel models, the Ferreira model and many others.

We added a new block to the Cardiovascular Simulation Toolbox, describing a variable capacitor with an external compliance function C(t). This block was necessary to simulate Ferreira's model. The block enabled the use of the elastance function E(t) proposed on Ferreira's work.

Based on the results from Table 1 it can be observed that the 2A2V model calculates a higher number of parameters and it can obtain results for both ventricles. The other models have less precision and describe only the response of the systemic circulatory system to the input flow from the ventricle. The 2A2V model calculates the ejection fractions for both ventricles, and the results are compliant with the theoretical data.

3 Conclusions

We have implemented several models found in the literature using the Cardiovascular Simulation Toolbox from O. Barnea, in MATLAB® R2013a. The two models reviewed that can produce a pressure-volume loop are the WK+V and the 2A2V models. The other models do not generate sufficient information to calculate these diagrams.

In the Windkessel models, the comparison parameter is the cardiac output, and it can be appreciated how this value is close to the expected value when the order of the model is increased. These models are exact but they do not calculate any information about the behavior of the ventricles.

The numerical comparison of the models showed that the 2A2V simulation calculates the higher number of output parameters and has an adequate accuracy comparing with the expected laboratory values. The parameters of this model can be adjusted further to simulate illnesses and defects in the cardiovascular system, and several blocks can be added to the Cardiovascular Simulation Toolbox as they are required.

The models implemented in MATLAB® can be further improved by coupling the system with COMSOL® to increase numerical precision and produce realistic results.

We have used the 2A2V model to simulate the cardiovascular system of a healthy person, and also of a patient suffering from systolic heart failure, achieving a LVEf of 58.7% for the healthy cardiovascular system, and a LVEf of 24.6% for systolic heart failure, showing agreement with the expected parameters from Chatterjee *et al.* [3].

Acknowledgements. We want to thank the Vicepresidency of Research and Outreach (VIE, Vicerrectoría de Investigación y Extensión) from the Instituto Tecnológico de Costa Rica, for supporting and providing funding for this research, project number 5402-1360-3401.

References

1 Ofer Barnea. Open-source programming of cardiovascular pressure-flow dynamics using SimPower toolbox in MATLAB and Simulink. *Open Pacing Electrophysiol Ther J*, 3(1):6, 2010.

2 Ph. Broemser and O. Ranke. Ueber die Messung des Schlagvolumens des Herzens auf unblutigem Weg. *Zeitung für Biologie*, 90:467–507, 1930.

3 K. Chatterjee and B. Massie. Systolic and diastolic heart failure: differences and similarities. *Journal of cardiac failure*, 13(7):569–576, 2007.

4 Edwards Lifesciences. Normal hemodynamic parameters and laboratory values. Retrieved on April 9, 2014 from the webpage `http://www.edwards.com/`, 2011.

5 A Ferreira, M.A. Simaan, J.R. Boston, and J.F. Antaki. A Nonlinear State-Space Model of a Combined Cardiovascular System and a Rotary Pump. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 897–902. IEEE, 2005.

6 Otto Frank. Die Grundform des arteriellen Pulses. Erste Abhandlung. Mathematische Analyse. *Zeitschrift für Biologie*, 37:485–526, 1899.

7 Harrison's Practice. Normal Hemodynamic Parameters. Retrieved on April 9, 2014 from the webpage `http://www.harrisonspractice.com/`, 2010.

8 Martin Hlaváč. Windkessel model analysis in MATLAB. *Proc 2004 Student Electrical Engineering, Information and Communication Technologies, Brno 2004*, pages 1–5, 2004.

9 Lancashire & South Cumbria Cardiac Network. Normal & Abnormal Intracardiac Pressures. Retrieved on April 9, 2014 from the webpage `http://lane.stanford.edu/`.

10 LiDCO. Normal hemodynamic parameters. Retrieved on April 9, 2014 from the webpage `http://www.lidco.com/clinical/hemodynamic.php`, 2011.

11 Gabriela Ortiz-Leon, Marta Vilchez-Monge, and Juan J. Montero-Rodriguez. An Updated Cardiovascular Simulation Toolbox. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 1901–1904. IEEE, May 2013.

12 A. Quarteroni. Modeling the cardiovascular system—A mathematical adventure: Part I. *SIAM News*, 34(5):1–3, 2001.

13 A Quarteroni. Modeling the cardiovascular system—A mathematical adventure: Part II. *SIAM News*, 34(6):1–3, 2001.

14 Thomas Schlosser, Konstantin Pagonidis, Christoph U Herborn, Peter Hunold, Kai-Uwe Waltering, Thomas C Lauenstein, and Jörg Barkhausen. Assessment of left ventricular parameters using 16-MDCT and new software for endocardial and epicardial border delineation. *AJR. American journal of roentgenology*, 184(3):765–73, March 2005.

15 Liron Sheffer, William P Santamore, and Ofer Barnea. Cardiovascular simulation toolbox. *Cardiovascular engineering Dordrecht Netherlands*, 7(2):81–88, 2007.

16 K. Umetani, D.H. Singer, R. McCraty, and M. Atkinson. Twenty-four hour time domain heart rate variability and heart rate: relations to age and gender over nine decades. *Journal of the American College of Cardiology*, 31(3):593–601, 1998.

Adaptive Failure Detection and Correction in Dynamic Patient-Networks

Martin Ringwelski[1], Andreas Timm-Giel[1], and Volker Turau[2]

1 Institute of Communication Networks
2 Institute of Telematics
Hamburg University of Technology
Hamburg, Germany
`{martin.ringwelski,timm-giel,turau}@tuhh.de`

Abstract

Wireless sensors have been studied over recent years for different promising applications with high value for individuals and society. A good example are wireless sensor networks for patients allowing for better and more efficient monitoring of patients in hospitals or even early discharge form hospital and monitoring at home. These visions have hardly led research as reliability is and issue with wireless networks to be known error-prone. In life critical applications like health care this is not an aspect to be handled carelessly. Fail-safety is an important property for patient monitoring systems.

The Ambient Assistance for Recovery (AA4R) project of the Hamburg University of Technology researches on a fail-safe patient monitoring system. Our vision is a dynamically distributed system using suitable devices in the area of a patient. The data in the network is stored with redundancy on several nodes. Patient data is analyzed in the network and uploaded to a medical server.

As devices appear, disappear and fail, so do the services being executed on those devices. This article focuses on a Reincarnation Service (RS) to track the functionality of the processes. The RS takes suitable actions when a failure is detected to correct or isolate the failure. Checking of the nodes is done adaptively to achieve a good response time to failures and reduce the power consumption.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Wireless Sensor Networks, Fail-Safety, Health Monitoring, Failure Masking, Distributed Systems

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.38

1 Introduction

Until today, patients need to stay in the hospital after their treatment for monitoring their recovery or even for diagnosing. With wireless sensor networks (WSNs) these doctors would be able to monitor the health signals of the patients remotely, while they can be at their familiar environment. This would not only help reduce costs by reducing the occupied beds in the hospitals, but might also help people recover. People tend to recover better in their families and many infections happen in hospitals[1].

[1] See `http://www.bmg.bund.de/praevention/krankenhausinfektionen/fragen-und-antworten.html` (2014-02-10)

Using a WSN for patient monitoring rises several problems regarding the fail-safety of the system. Wireless links are volatile and nodes can fail. While those problems also affect other kinds of WSNs, in patient monitoring they can be life critical.

This work is part of the Ambient Assistance for Recovery (AA4R)[2] project of eight institutes at the Hamburg University of Technology. The project aims at building a patient monitoring system to support recovery of patients and relieve of hospital personal. This will be achieved by an uninterrupted use of technology from the ward of a hospital over a rehab center to the patients home. By this, we are closing the gaps between ambient assisted living in home care, telemonitoring and telediagnosis. Fail-safety is addressed at different subsystems such as the communication infrastructure or the sensors themselves. Further, the fail-safety is addressed at system-level. In this paper we primarily focus on fail-safety in the communication infrastructure.

The system is distributed using resources available in the area. We point out and model different sources of failure and discuss strategies to reduce them and the risk of a total failure of the system. To achieve that, we build a monitoring service for the system, called Reincarnation Service (RS). The RS discovers faults and takes actions to solve them.

In the next chapter, we review related work and show the differences to our approach. The vision of our scenario is introduced in the third chapter, followed by possible failures and failure rates. We discuss our approach and the expected improvements in the fifth chapter. In chapter six, we give a conclusion and discuss the future work.

2 Related Work

Chipara et al. [4] built and tested an IEEE 802.15.4 based wireless monitoring system for patient data. In contrast to our idea of monitoring the patient at home for longer periods, they only measured for up to three days during the patients stay at the hospital. The sensor data was limited to the pulse rate and the oxygen level of the blood. Data rates were only in the range of several bytes per minute. As the patients only moved inside the hospital and several relay points where installed to forward the data, the network reliability was not problematic in this study. The system was centralized and the nodes preconfigured. Problems of reliability in a distributed system were not investigated. The work of Ko et al. [7] invested a similar system.

Chen et al. [3] investigated a routing protocol for patient monitoring and fall detection at home. In case of emergency the system transmits the ECG and in-door position of the patient to first responders. Their protocol ensures a fast and reliable delivery of the data. The system is closed and no ambient sensors or other devices are used.

Preventing or avoiding faults is not possible in complex distributed systems. We plan to detect and correct or isolate faults in the system. To achieve high reliability, Herder et al. [5] described a Reincarnation Server in the Minix 3 operating system. This server periodically checks the state of the other services in the system and restarts them when they are broken. The described method is used on one machine and the checking time is constant. Nevertheless, we can use that idea of failure detection and correction and thereby increase the overall reliability of the system.

This approach can also be read out of the white paper of IBM [6] about self-healing, self-configuring and self-optimizing. They identified a four state healing loop consisting of

[2] `http://www.aa4r.org/`

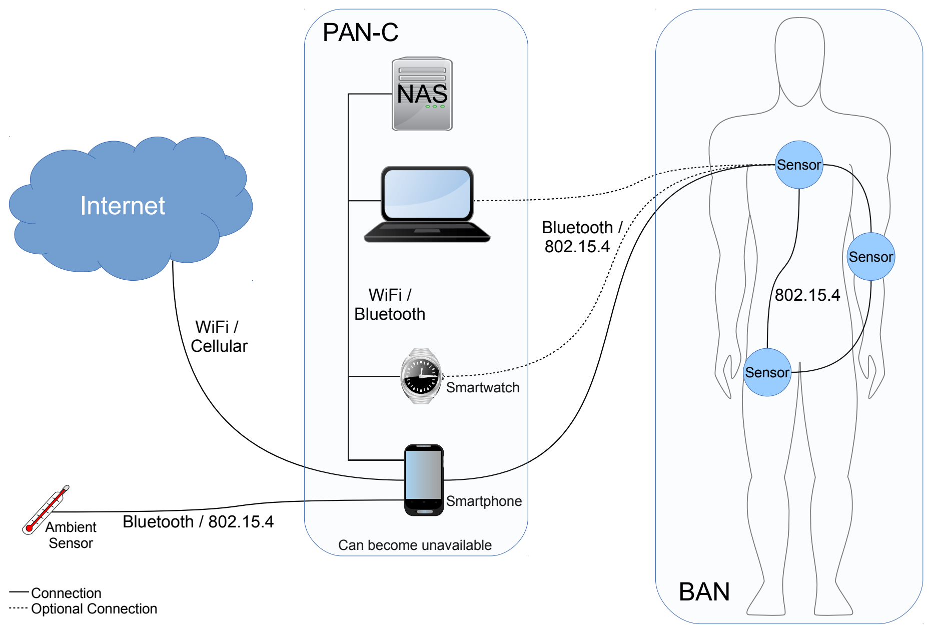


Figure 1 Network Architecture.

monitoring the state, detecting errors, analyzing the failure to plan the repair and executing the repair.

Our implementation will use the CometOS framework [12]. CometOS allows us to write code that can be simulated in Omnet++ and compiled for testbed hardware. This reduces the risk of failures during the migration of the software.

3 Scenario

The AA4R project aims at people recovering from different kinds of incidents. From a broken leg over to heart attack. These people need to stay in hospitals to monitor their recovery and ensure they are not suffering a fall-back. Another scenario are people coming to the hospital without knowing what is wrong. They also need to stay in the hospital to monitor their health values and be able to make a diagnosis. We want those people to be able to go home and continue with their lives. Our monitoring system will assist patients in their recovery and the physicians in their diagnosis. It will help prevent dangerous situations and call help in emergency situations.

The system we envision consists of a Body-Area-Network (BAN) of sensors, collecting vital information, ambient sensors and other External Devices (EDs), building a Personal-Area-Network (PAN). EDs, like smartphones, smartwatches or laptops, are used for the PAN-Controller (PAN-C). Figure 1 shows the network architecture of the Patient-Network (PN) including the BAN, EDs building the PAN-C and an ambient sensor. The sensors in the BAN might have direct connections to several EDs, but only only one connection will be preferred, depending on the stability of the connection and the resources of the device. On the other end of the Internet will be a medical server, which stores the information of the patients under pseudonyms for diagnosis by physicians.

The PAN-C is a distributed system consisting of different services in the network. A service will be delegated to the most suitable device, determined by their available resources and connections. Those services are:

- **Forwarding**: We want to send the data to a medical server. The server can automatically analyze the data and prepare relevant graphs for physicians or present raw data for doctors to monitor the values or figures remotely. This service needs to be done by a device having a reliable Internet connection.
- **Storing**: Storing the data that is collected in the BAN and other ambient sensors needs to be done redundantly by several devices. The data is needed for analyzing the health status in the network and to maintain the state and integrity of the system. Not only the current, but also the past data of the medical sensors is important to understand the situation of the patient. This service becomes critical when lacking an Internet connection.
- **Plausibility**: This service looks for anomalies in the sensor data to detect drifts or other failures of sensors. To achieve that, we need to have past data available.
- **Inclusion**: This service can be described as the coordination service. New devices for services of the PAN-C need to be found and included in the network. Also, ambient sensors that might be helpful to analyze the patient data, e.g. room temperature and humidity, need to be included. This service can be compared to the Membership Service in the work of Rodrigues et al. [11].
- **Control**: The control service allows medical applications to use possible actuators in the BAN. A patient with diabetes can get his insulin automatically or a patient with strong pain can get his analgesics. This service also needs to eliminate the risk of overdosing the patient.

Services on the EDs run as virtual machines to be separated from the rest of the devices system. This also enables us to simply copy the service to a different device and switching the responsibility to the new device.

Having several devices responsible for the services in the network reduces the risk, that all devices fail at once, but it increases the risk that devices running important services fail. Also consistency of data and decision are critical points in the network. Failures of services need to be detected to take measures to isolate and try to repair the failures. For that, a Reincarnation Service (RS) will run on every device to monitor the state of the connected devices.

The overlying application should not be concerned with failures of devices in the network. Only if a service can not be fulfilled by any other device or compensated by another service, the application should be informed to take actions.

4 Failures

As this system is responsible for a human being, a failure of one single component could lead to an undiscovered critical situation of the patient and thereby to death. That is why we need to analyze the possible faults in the system. We identified seven fault categories:

- **No Internet connection**: The data needs to be transmitted to a medical server for remote monitoring by a doctor. If the connection fails, the data needs to be temporarily stored locally and transmitted when a connection comes up again. For diagnosis, this will be a sufficient solution. In emergency situations, this problem can be life critical. Depending on the previous condition of the patient, a timeout might be used to trigger an emergency call on the server.

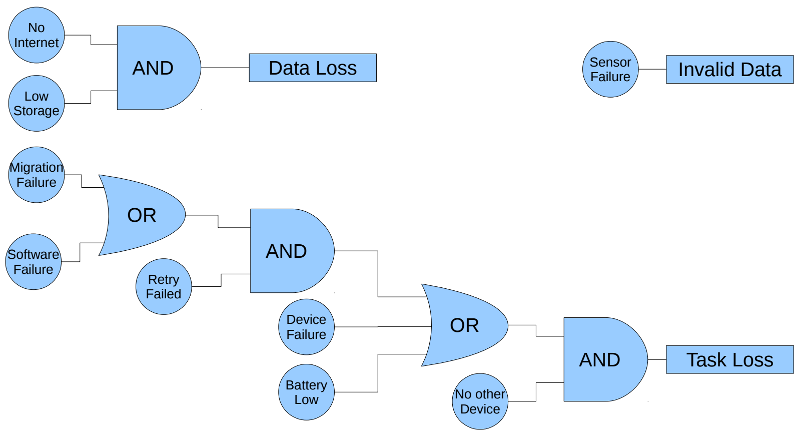


Figure 2 Fault Tree.

- **Low Storage**: The last data should always be held on nodes in the network for a better analysis of the current patient state. In case of a lost Internet connection the data needs to be kept even longer, so that the storage may get sparse.
- **Migration failure**: The state of the network needs to be known by every device. During transfer of a service from one device to another, inconsistencies in the distributed information might arise.
- **Sensor failure**: Sensors might drift, have an offset or even stop working. Those errors are not further considered in this work, as they are part of the analysis that other members of the AA4R project are working on.
- **Device failure**: Any device can randomly fail because of bad production or wear-out.
- **Software failure**: We can not guarantee an error-free program. An error may also be caused by a memory shortage. In this case it might work again after a restart.
- **Battery low**: As we are using wireless devices running on battery, any device can run low on battery. This failure can be easily foreseen.

The fault tree in Figure 2 shows the impact of the given failures. Only if one of the three subsystems fails and we encounter data loss, service loss or a sensor failure, the application must be informed. Otherwise the faults can be masked for the application, be it through a state-machine or a primary-backup approach [8]. Invalid data will be handled by the analysis service, which might just discover a drift or switch to a redundant sensor, if available.

Masking failures allows the application to concentrate on its service rather than having to deal with error handling. It is still possible for faults to occur, thus fault handling can not completely left out.

4.1 Expected Failure Rates

Failure rates, hazard rates or hazard functions are names for the expected failures at a given time. In contrast to the probability density function $f(t)$, which depicts the overall probability of failures per time, the hazard function $h(t)$ depicts this probability under the assumption, that no error happened before. It is the fraction of the probability density function by the survival rate, whereas the latter is $S(t) = 1 - F(t)$.

For electronic devices Nowlan and Heap [10] proposed a hazard function with an infant mortality and a constant failure rate. The infant mortality is caused by incapable devices, program or migration errors. Those errors are more likely to happen at the beginning of the lifetime of a device. Devices that survive the infant mortality phase are not likely to suffer from those errors. The constant failure probability is due to battery drain, interfering failures or devices moving out of reach. Those errors can happen at any time. A wear-out mortality is not included, as the system is not expected to run tens of years. We can also expect that the patient is taking care of his or her smart phone battery.

The infant mortality can be described with the Weibull distribution (equation 1), with a $0 < \mathrm{k} < 1$ [13]. k is the shaping factor, where $\mathrm{k} < 1$ results in a decreasing failure rate over time. λ is the scaling parameter. $\mathrm{f_{im}(t)}$ describes the error probability at a certain time, (2) is the cumulative error probability and (3) shows the hazard function:

$$\mathrm{f_{im}(t)} = \mathrm{k}\lambda_{\mathrm{im}}^{\mathrm{k}}\mathrm{t}^{\mathrm{k}-1}e^{-(\lambda_{\mathrm{im}}\mathrm{t})^{\mathrm{k}}} \tag{1}$$

$$\mathrm{F_{im}(t)} = 1 - e^{-(\lambda_{\mathrm{im}}\mathrm{t})^{\mathrm{k}}} \tag{2}$$

$$\mathrm{h_{im}(t)} = \mathrm{k}\lambda_{\mathrm{im}}^{\mathrm{k}}\mathrm{t}^{\mathrm{k}-1} \tag{3}$$

Failures by interfering signals or devices moving out of range can occur at any time. Those failures are not more likely to happen at the beginning or after some time. The constant failure rate can be described by an exponential distribution. Equations (4), (5) and (6) show the probability density function, the cumulative distribution and the hazard function of the exponential distribution:

$$\mathrm{f_c(t)} = \lambda_{\mathrm{c}}e^{-\lambda_{\mathrm{c}}\mathrm{t}} \tag{4}$$

$$\mathrm{F_c(t)} = 1 - e^{-\lambda_{\mathrm{c}}\mathrm{t}} \tag{5}$$

$$\mathrm{h_c(t)} = \lambda_{\mathrm{c}} \tag{6}$$

The hazard functions can be easily combined by addition to get the system hazard function in equation (7). For any given interval $[\mathrm{t_a}; \mathrm{t_b}]$ we can calculate the probability of a failure, under the assumption that no failure occurred before time $\mathrm{t_a}$, with the equation (8):

$$\mathrm{h(t)} = \mathrm{k}\lambda_{\mathrm{im}}^{\mathrm{k}}\mathrm{t}^{\mathrm{k}-1} + \lambda_{\mathrm{c}} \tag{7}$$

$$\mathrm{P_{fail}(t_a, t_b)} = \frac{\mathrm{F(t_b) - F(t_a)}}{\mathrm{1 - F(t_a)}} = 1 - \frac{e^{-\lambda_{\mathrm{c}}\mathrm{t_b} - (\lambda_{\mathrm{im}}\mathrm{t_b})^{\mathrm{k}}}}{e^{-\lambda_{\mathrm{c}}\mathrm{t_a} - (\lambda_{\mathrm{im}}\mathrm{t_a})^{\mathrm{k}}}} \tag{8}$$

Figure 3 depicts sample hazard functions for equations (3), (6) and (7). The used example values are later discussed in section 5.1.1.

5 Counter-measures

To achieve fail-safety, the system needs to be in a stable state, although inevitable faults may occur. For that, we need to detect the failures as fast as possible and take measures to

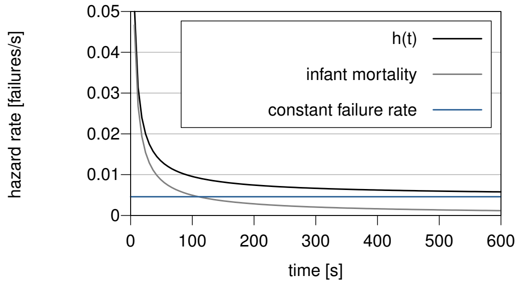


Figure 3 Sample of hazard functions for equations (3), (6) and (7) with $k = 0.2$, $\lambda_{\text{im}} = 0.9335\frac{1}{s}$ and $\lambda_c = 0.0046\frac{1}{s}$.

solve them. The system must not rely on one single node. Also, even if a service or data is lost, the system needs to be aware of the situation and keep on working with its limited capabilities.

There are no metrics for fail-safety, but we can use the Mean-Time-To-Failure (MTTF), Mean-Time-To-Recovery (MTTR) and the fraction of components allowed to fail, before the system becomes unstable. Those figures are used to describe the reliability.

For the different kind of failures, we have to take specially assembled counter-measures:

- **Link Quality Prediction**: By predicting the quality of a link, we can take measures to switch a service from a node to another one, before it is disappearing. This would extent a Link-Quality-Estimator (LQE) to a Link Quality Predictor.
- **Redundant storage**: As nodes with important information can disappear without warning, we need to have redundant data in the network. Acedaǹski et al. [1], Nguyen et al. [9] and Rodrigues et al. [11] have analyzed distributed network storage with random linear network coding. Nevertheless they can not assure that data is available after some time. Depending on the sensors data-rate and the number of devices and their capacities, data needs to be deleted eventually. We want to use an approach that assure data availability and prioritizes the data by importance, so only dispensable data gets lost.
- **Reincarnation Service**: For detecting failures in software and loss of connection, a Reincarnation Service (RS) checks the state of the devices and processes. In case of failure, the process can be restarted or a substitute can be found.

The RS itself also needs to be distributed. Every device running a service has a RS responsible for the services on that device. This ensures reliability of services on the devices. Figure 4 shows the layered software architecture on two connected devices. Services that are not present on a device, due to limited resources, can be used from another device. The medical application itself does not know, where the services are hosted. The RS is responsible for the services and the application to be running and available.

As devices can also fail or disappear, the device that has include another device, that means it has delegated a service to that device, is responsible for that. The Reincarnations Service has to check the Reincarnation Service on the other device.

5.1 Detecting Failures

To detect failures, processes and devices need to be checked periodically. The question remains how often a check should be performed. Too frequent checks will drain the battery and cause congestion, too few lead to long unrecognized failures which can cause other failures.

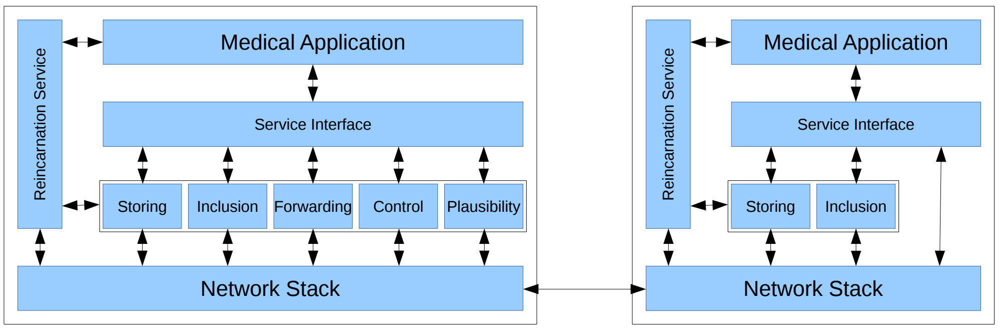
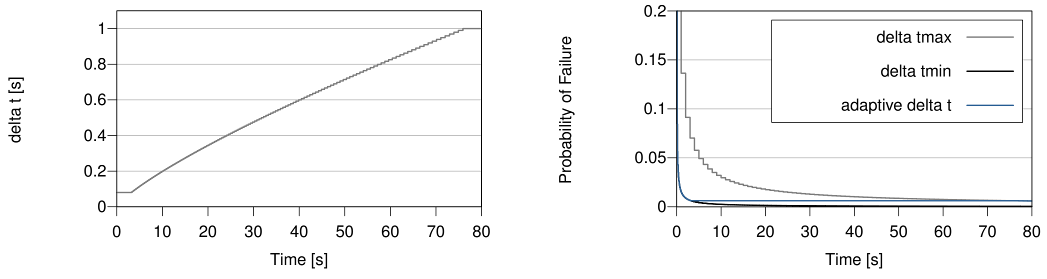


Figure 4 Software Architecture, showing two devices offering different services



(a) Adaption of Δt **(b)** Expected failures per Interval

Figure 5 Influences of Δt.

Setting the probability of failures in an interval to a fixed value fail_{max}, we can calculate the next time we need to check if the service is still alive. By that, we achieve an adaptive testing of the process, depending on the expected failure rate. But solving that equation is not trivial. We use the Newton method to approximate the next time to check (equation 9):

$$0 = \lambda_c \Delta t + {\lambda_{im}}^k \left((t + \Delta t)^k - t^k \right) + ln(1 - P_{\text{fail}_{\text{max}}})$$

$$\Delta t_{next} = \Delta t_{min} + \frac{\lambda_c \Delta t_{min} + {\lambda_{im}}^k \left((t + \Delta t_{min})^k - t^k \right) + ln(1 - P_{\text{fail}_{\text{max}}})}{-\lambda_c - k{\lambda_{im}}^k (t + \Delta t_{min})^{k-1}} \qquad (9)$$

If that equation results in a Δt_{next} less than Δt_{min}, Δt_{next} is set to Δt_{min}. To make sure that a failure does not stick undetected for a long time, it is best to also set $\Delta t_{next} < \Delta t_{max}$. Figure 5a shows a sample curve for an adaptive Δt, whereas Figure 5b depicts the failure probability during the different intervals for fixed Δts and the adaptive one from Figure 5a.

After detecting a failure, other processes must be tested again, regardless of their next checking time. This way, we can check if the failure affected other processes or was introduced by the failure of another process.

5.1.1 Calculating the parameters

To create a fitting curve, we need to know the parameters k, λ_c and λ_{im}. k and λ_{im} will be set to fixed values. They can be estimated in advance by empirical values of failures.

For every device that was already used for the PAN-C, we can adaptively compute λ_c with an exponentially weighted moving average (EWMA) filter. As for a constant failure rate, the MTTF is the inverse value of λ_c. We can use the last time to failure to update our estimation. Devices that are known to stay in the network for longer times will be checked less frequently than devices that disappear after short times.

5.1.1.1 Example

Assume we have a smartphone with a MTTF of two years, the battery holds for 24 hours and we expect the device to stay in the network for 8 hours. If we experience a migration failure, it is expected to occur in the first second. If the software has a failure we expect it to occur in the first minute. The MTTF of the whole system can be calculated by [2]:

$$\frac{1}{\mathrm{MTTF}} = \sum \frac{1}{\mathrm{MTTF}_{\mathrm{subsystem}}} .$$

We do that separately for the infant mortality failure and for the failures with constant failure rate. The migration failure and the software failure are part of the infant mortality, with an MTTF of about 0.9836 s. The MTTF for this is the expected value of the Weibull distribution, which is calculated by:

$$\mathrm{E}(X) = \frac{1}{\lambda_{\mathrm{im}}}\Gamma(1+\mathrm{k}) \quad \Longrightarrow \quad \lambda_{\mathrm{im}} = \frac{\Gamma(1+\mathrm{k})}{\mathrm{MTTF}} .$$

Assuming $\mathrm{k} = 0.2$, we calculate $\lambda_{\mathrm{im}} = 0.9334\ \frac{1}{s}$. The lower we set k, the more we decrease the influence of infant mortality failures to the long term failure rate.

λ_c is simply the inverse of the MTTF for the device, the battery and the device availability. By that we get $\lambda_c = 4.63 \cdot 10^{-5}\ \frac{1}{s}$. The Round-Trip-Time (RTT) between two neighbor nodes in IEEE 802.15.4 can be estimated with 20 ms (including MAC retries and Back-offs), so $\Delta t_{\min}$ should not have a lower value. To also allow other packets in the network to freely flow, we set $\Delta t_{\min} = 80$ ms. $\Delta t_{\max}$ is set to one second, so we are still able to catch failures after at least that time.

Those calculated values were used in the example Figures 5a and 5b. The maximum failure probability per interval was set to $\mathrm{fail}_{\max} = 0,62\%$.

When the smartphone disappears from the network after two hours, λ_c will be recalculated for the next time. The experienced time to failure would result in a $\lambda_{c,\exp} = 1.39 \cdot 10^{-4}\ \frac{1}{s}$. A new λ_c will then be calculated by:

$$\lambda_{c,\mathrm{new}} = \alpha\lambda_{c,\mathrm{old}} + (1-\alpha)\lambda_{c,\exp} .$$

5.2 Defeating Failures

When failures occur, it is important to understand their cause. When the device failed, the service needs to be switched to another device, but if it was a random failure it might be sufficient to restart the process. On the other hand, if the service program for that device is error-prone, the service needs to be updated.

Of cause, switching the service to another device is not an option if no other is available. If the device is still available and no other is capable of fulfilling it, the service needs to be restarted. Otherwise, if the device is not available anymore, the service must be set on hold.

The checking of devices and services by the RS must not only rely on a simple echo ping, but must ask for the devices and services state. Services might need to do a simple job, to see, if the service is still working properly. Answers by devices need to inherit the battery status, link quality, RAM usage and a version of the information used on that device. With this information, we can derive a probable cause for the failure. We can see if the information is out of date, the processes ran out of ram, the link went weak or if the device went out of energy.

The gathered information about the device can also influence the next checking time. A device responsible for the Internet connection with a bad link quality to the WiFi should be tested more often, than with a good link quality.

Another way to defeat failures is to have redundancy. We already mentioned to store data redundantly on the devices of the network, but we can also have several devices fulfilling the same service, as it is done on aircrafts. But in contrast to aircrafts, we have a dynamic system with changing devices and can only do that, when spare resources are available. The up-lying applications have to make their requests through the RS, unknowing where the services are fulfilled. That way failures can be masked from the applications and only get an error, when a service is not able to run anymore.

5.3 Expected Improvements

By implementing this adaptive checking behavior, we expect to keep the probability of undetected failures below a set maximum failure probability $P_{\mathrm{fail}_{\max}}$, while also not keeping the network occupied with checking packets and having a low energy consumption. Network capacities force us not to stick to that optimum, when the failure rate would expect a checking time below the round-trip-time. The adaption by an EWMA filter will make sure, that the assumed hazard rate will reflect the experienced disappearing of a device.

With the RS we also mask failures of devices from applications. Only when service can not be fulfilled anymore, the applications will be informed and can take actions on their own.

6 Conclusion and Future Work

In this work we presented our vision of a fail-safe dynamic Patient-Network for health monitoring. To achieve fail-safety, among other counter-measures we propose an adaptive checking of the functionality of the components by a Reincarnation Service (RS). Instead of a fixed interval to check the components, we use an interval corresponding to the expected failure-rate. By that, we hope to keep the probability of undiscovered failures below a set threshold, while not flooding the network with checking packets and also keeping the energy consumption low.

Those measures for a fail-safe distributed health monitoring system, have not been implemented and tested yet. We want to build an Omnet++ simulation to validate our assumptions. We also need to further investigate the possible steps to be taken, when a failure is discovered.

In this paper we only focused on an the adaptive checking of processes and devices. We have not discussed how to check the functionality of a process. It is also our aim to investigate in a fail-safe distributed storage and the prediction of disappearing nodes with LQE.

Other works of the AA4R project will focus on the security of the system, analysis of the data and validation of the model.

References

1 Szymon Acedański, Supratim Deb, Muriel Médard, and Ralf Koetter. How good is random linear coding based distributed networked storage. In *Proceedings of the WINMEE, RAWNET and NETCOD 2005 Workshops*, Riva del Garda, Italy, April 2005.

2 A. Birolini. *Quality and reliability of technical systems: theory, practice, management.* Springer, 1997.

3 Shyr-Kuen Chen, Tsair Kao, Chia-Tai Chan, Chih-Ning Huang, Chih-Yen Chiang, Chin-Yu Lai, Tse-Hua Tung, and Pi-Chung Wang. A reliable transmission protocol for zigbee-based wireless patient monitoring. *Information Technology in Biomedicine, IEEE Transactions on*, 16(1):6–16, Jan 2012.

4 Octav Chipara, Chenyang Lu, Thomas C. Bailey, and Gruia-Catalin Roma. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *The 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*, pages 155–168, Zurich, Switzerland, November 2010. ACM.

5 Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, and Andrew S. Tanenbaum. Minix 3: A highly reliable, self-repairing operating system. *SIGOPS Oper. Syst. Rev.*, 40(3):80–89, July 2006.

6 IBM Corp. *An architectural blueprint for autonomic computing.* IBM Corp., USA, October 2004.

7 JeongGil Ko, Tia Gao, R. Rothman, and A. Terzis. Wireless sensing systems in clinical environments: Improving the efficiency of the patient monitoring process. *Engineering in Medicine and Biology Magazine, IEEE*, 29(2):103–109, March 2010.

8 Sape Mullender, editor. *Distributed Systems.* ACM, New York, NY, USA, 1989.

9 Kien Nguyen, Thinh Nguyen, Y. Kovchegov, and Viet Le. Distributed data replenishment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(2):275–287, 2013.

10 F. Stanley Nowlan and Howard F. Heap. Reliability-centered maintenance. Technical Report AD-A066-579, United Airlines and Office of Assistant Secretary of Defense, December 1978.

11 Rodrigo Rodrigues, Barbara Liskov, Kathryn Chen, Moses Liskov, and David Schultz. Automatic reconfiguration for large-scale reliable storage systems. *IEEE Transactions on Dependable and Secure Computing*, 9(2):146–158, March 2012.

12 Stefan Unterschütz, Andreas Weigel, and Volker Turau. Cross-platform protocol development based on omnet++. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 278–282, ICST, Brussels, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

13 M. Xie and C.D. Lai. Reliability analysis using an additive weibull model with bathtub-shaped failure rate function. *Reliability Engineering & System Safety*, 52(1):87 – 93, 1996.

Challenges and Opportunities in Design of Control Algorithm for Artificial Pancreas

Mahboobeh Ghorbani and Paul Bogdan

University of Southern California
Electrical Engineering Department
`{mahboobg,pbogdan}@usc.edu`

Abstract

With discovery of the insulin, Type-1 diabetes converted from a fatal and acute to a chronic disease which includes micro-vascular complications which range from Kidney disease to stroke and micro-vascular complications such as retinopathy, nephropathy and neuropathy. Artificial pancreas is a solution to improve the quality of life for people with this very fast growing disease in the world and to reduce the costs. Despite technological advances e.g., in subcutaneous sensors and actuators for insulin injection, modeling of blood glucose dynamics and control algorithms still need significant improvement. In this paper, we investigate challenges and opportunities for development of efficient algorithm for designing robust artificial pancreas. We discuss the state of the art and summarize clinical and in silico assessment results. We contrast conventional integer order system approach with a newly proposed fractal control and summarize its benefits.

1998 ACM Subject Classification G.1 [Mathematics of Computing]: Numerical Analysus, J.3 [Computer Application]: Life and Medical Science-Health

Keywords and phrases Cyber-Physical Systems, Artificial Pancreas, Optimal and Model Predictive Control, Fractal Model Predictive

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.49

1 Introduction

In healthy individuals, the alpha and beta cells of the pancreas regulate the blood glucose concentration to around 80 mg/dl. For people suffering from Type-1 diabetes mellitus (T1DM), which is one of the fastest growing diseases globally, there is little or no endogenous insulin production, leaving the body unable to lower blood glucose without exogenous insulin. The impact of the intensive insulin therapy was not revealed up until the publication of results of Diabetes Control and Complication Trial (DCCT) in 1993 [26]. The CDDT involved a comparison of conventional therapy (one or two daily insulin injections and a daily monitoring of blood glucose or urine) and intensive insulin therapy and concluded that intensive therapy resulted in lower mean blood glucose values and significantly reduced complications (retinopathy, nephropathy and macro-vascular disease). The risk of complication is directly related to glycated hemoglobin known as HbA1c. OGrady et al. find that tighter blood glucose levels achievable with a closed-loop artificial pancreas (AP) results in Medicare savings of 1.9 billion over 25 years with improved quality of life (QOL) [24]. A schematic view of a closed-loop artificial pancreas is shown in Fig. 1, which is mainly composed of three parts:

Continuous time blood glucose measurement (CGM): The knowledge of glucose concentration in blood is a key aspect in the quantitative understanding of the glucose-insulin system and in diagnosis and treatment of diabetes. By the ability of CGM devices to provide glucose readings in real time, engineers can exploit signal processing and control theory to be used in designing efficient artificial pancreas. Besides the improvement of hardware part of CGM devices, a vast amount of research has devoted to address denoising, prediction and alert generation [13, 1, 5, 22, 17, 3, 6].

Control algorithm and safety layer: The main component of AP is the control algorithm that determines the right insulin injection rate based on CGM data to prevent hyperglycemia and hypoglycemia. Design of a QOL-aware AP is a challenging task since it requires building accurate mathematical model of glucose-insulin kinetics. Algorithms often include a safety layer as a supervisory module that constraints insulin delivery. This layer may monitor and limit insulin on board (the insulin delivered but yet to exert its action) or maximum insulin rate or may suspend insulin delivery at low glucose levels or when glucose is decreasing rapidly. In this paper, we overview these challenges and control strategies employed so far.

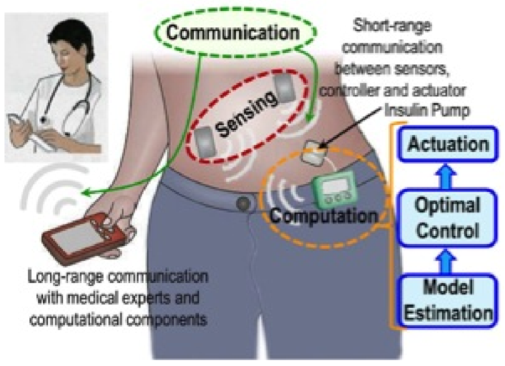

Communication
Short-range communication between sensors, controller and actuator
Insulin Pump
Sensing
Actuation
Computation
Optimal Control
Model Estimation
Long-range communication with medical experts and computational components

Figure 1 Systematic view of artificial pancreas [23].

Insulin injection device: The essential function of AP is the Insulin delivery. Insulin pumps, if inserted in a proper closed-loop system allow automatic insulin delivery. There are several technologies that can perform this task: an intra-venous route, subcutaneous insulin infusion (SCII) or intaperitoneal insulin delivery. Continuous subcutaneous insulin infusion (CSII) uses a portable electromechanical pump to mimic nondiabetic insulin delivery as it infuses at preselected rates normally a slow basal rate with patient-activated boosts at mealtime.

2 Control related challenges and constraints

Blood glucose (BG) regulation requires control algorithm to determine the best insulin injection over time. They have been tested in-silico and clinically over time and improved over the years. In this section, we first address the main challenges and constraints in designing efficient control algorithm. Next, we explain different control algorithms proposed so far and compare their performance.

Non-negligible delay in glucose measurement and between insulin injection and absorption: After administration of a subcutaneous bolus of rapid acting insulin analogues, the maximum BG lowering effect may occur after up to 90–120 min. This time lag is often not accounted for design of control algorithm. Patients treated with insulin pump are warned against stacking caused by the administration of a series of correction boluses. The same principle applies to closed-loop systems. In order to prevent hypoglycemia, high glucose levels have to be brought within normal range slowly during closed-loop delivery. Methods to assess the impact of injecting insulin (e.g. the one proposed in [7]) are highly needed in order to protect against insulin overdosing. Two alternative insulin delivery routes, intraperitoneal (IP) and technosphere insulin (TI) showed faster pharmacokinetic characteristics that can improve the design of future AP systems. Design of the AP using these fast acting alternative routes may enhance BG regulation by reducing actuation delays, especially during mealtime.

Asymmetric risk for low and high BG levels: The ultimate objective of any AP is to improve QOL and minimize complications resulting from poor blood glucose control. Toward this end, one should note to the asymmetric risk associated to high BG levels. Low BG levels are acutely risky as they can result in altered mental state, seizures and coma. Meanwhile, high BG levels increase the risk of chronic complications such as retinopathy, nephropathy and cardiovascular disease.

Irreversible action of insulin: Only positive amount of the injected insulin is possible and it cannot be collected back from the patients blood. An alternative to deal with this problem is to use bihormonal treatment [12] consisting of injecting glucagon and insulin. However, this also increases the problem space and complexity.

Meal detection/estimation: Meal dynamics can have a significant disturbance effect on BG level. In a fully closed-loop mode, insulin is delivered on the basis of glucose excursions only, without information about timing or meal size. In a less ambitious configuration that uses meal announcement, the closed-loop system is informed about meal size, and may generate advice on prandial insulin bolus.

Alternatively, control algorithms can automatically increase insulin delivery based on the carbohydrate content of the meal. A hybrid approach is characterized by administration of a small pre-meal priming bolus or administration of a fixed bolus and delivering the remaining insulin through the closed-loop operation [9].

Time dependency of control requirements: An important challenge in development of artificial pancreas is that overnight treatment requires slow acting insulin injection while post-prandial control requires rapid and aggressive insulin delivery to control BG.

On the other hand, exercise of moderate intensity increases the risk of hypoglycemia [32]. Exercise announcement or heart rate monitoring to suspend insulin during closed-loop delivery may be another effective method to control glucose levels during exercise. Pre-emptive carbohydrate intake or dual hormone treatment with glucagon might be needed to fully eliminate the risk of exercise-related hypoglycemia as responses to exercise are highly variable. To sum up, BG control is a time dependent process and this should be taken into account in order to have a safe and efficient AP.

Variability of model parameters: Up to 4 times inter-subject variability in rapid-acting insulin analogue pharmacokinetics has suggested with occasionally as much as 50 % intra-subject variability [12]. Within subject variability of insulin needs includes both day-to-day and hour-to-hour variations in insulin sensitivity owning to circadian and diurnal cycles, dawn phenomenon (an abnormal early morning increase in BG concentration), acute illness, stress and a delayed effect of alcohol intake. Basal insulin needs are generally lower in young individuals compared with older ones. Also, since overnight control requires regulation based on mild control actions while postprandial regulation is characterized by prompt and energetic correction, timely control effect should take place.

3 Control algorithms for BG level regulation

In this section we present two main groups of controller for BG level regulation namely proportional-integral-derivative and model predictive controller.

3.1 Proportional-Integral-Derivative (PID) controller

PID controller is a generic control loop feedback mechanism widely used in industrial control. The PID control algorithm for artificial pancreas adjusts the insulin delivery rate by assessing glucose excursions from three viewpoints: the departure from the target glucose level (the proportional component), the area under the curve between measured and target glucose levels (the integral component) and the rate of change in measured glucose levels (the derivative component). Some controllers include only a subset of the components (e.g. a proportional-derivative [27]).

To better understand the intuition behind using PID controller in the control algorithm for artificial pancreas, one should note that dose of insulin is directly related to the proportional error (P) (current glucose minus target glucose) since a patient with higher glucose level needs more insulin rather than one with lower glucose level. Moreover, in two patients with the same glucose level but with different rate of glucose increase, the one with higher increase rate should get a higher dose of insulin and this justifies the derivative element (D). To understand the role of Integral element (I), it should be noted that for two patients with the same current glucose level and no change in the very recent minutes, the one with more hours spending in high BG level (thus, having more integral error) needs more insulin due to the fact that this is a sign on insulin resistivity. Steil et al. have shown the normal healthy pancreas displays proportional, derivative and integral dynamics [14]. They argue that the abrupt step increase in glucose causes a rapid rise in pancreatic insulin release, which is called first phase response and is related primarily to the derivative component. Slower rise in insulin is called the second phase response, which corresponds to proportional term and persists as long as glucose is elevated. There is also an integral component employed in the second phase since insulin secretion after 3 hours of elevated glucose at a fixed level is greater than insulin secretion after only 1 hour at the same glucose level.

Equation 1 shows the the components of control signal ($u(t)$) that is the amount of insulin injection rate as a function of $e(t)$ which is the difference between BG level and the reference value. The K_p, K_i and K_d parameters can be assigned by learning algorithms that have been discussed in control related textbooks. Optimizing using PID controller needs tuning of the controller by some methods, like the ones proposed in [16] and [2].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

PID approach has inherent limitations due to time lags in glucose sensing and insulin action. Several studies have investigated this approach and achieved some improvement over conventional PID approach. For example, Weinzimer et al. in [29] have tested PID algorithm for insulin injection in 17 adolescences. They have tested both fully closed-loop and hybrid closed-loop (with pre-meal priming bolus) and show the addition of small manual priming bolus doses of insulin given 15 min before meals improves glycemic excursions. A different study by Renard et al. in [10] proved the feasibility of intraperitoneal insulin delivery for artificial beta cell and supported the need for further study since subcutaneous insulin delivery from a portable pump encountered delays and variability in insulin absorption. They evaluated their proposed method in a clinical study on eight T1DM patients while the time spent in 4.4–6.6 mmol/l was the primary end point. Another study in [12] uses both insulin and glucagon to prevent hypoglycemia encountered in PID algorithm with only insulin as the treatment.

3.2 Model Predictive Controller

Model Predictive Control (MPC) is a general optimization framework that can involve many different types of models and objective functions. The MPC approach is at the front of current research into closed-loop systems. It acceptably accommodates delays associated with insulin absorption and can also easily account for meal intake and prandial insulin boluses by the patient. The other advantage of model predictive control paradigm is the fact that it can account for variability since the model parameters can be personalized. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots into account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot. MPC has the ability to anticipate future events and can take control actions accordingly.

The vital ingredient of MPC is a model that links insulin delivery and meal ingestion to glucose excursions. This model can be physiological and account for fundamental processes regulating glucose levels or a black box model that disregards insights but learns the insulin glucose relationship via formal pattern recognition technique. They both can benefit from a wide range of mathematical models of glucoregulatory system. It is therefore clear that proper models of glucose and insulin kinetics as well as models that can be used to predict near-future metabolic behavior are mandatory. Minimal models (describing the key components of system functionality) and maximal models (nonlinear, high order models) are reviewed by Cobelli in [6].

A general MPC problem formulation, which includes optimization objective (Equation 2), glucose-insulin dynamical model (Equation 3) and initial value, glucose state and insulin control constraints (Equation 4) can be written as follows.

$$min_{u(t)} \int_0^{t_f} F(g(t), u(t))\, dt \tag{2}$$

$$\frac{dg(t)}{dt} = a_G\, g(t) + b_G\, u(t) \tag{3}$$

$$g(t=0) = g_0, u_{min} \leq u(t) \leq u_{max}, g_{min} \leq g(t) \leq g_{max} \tag{4}$$

where $g(t)$ denotes the BG level and $u(t)$ denotes the amount of insulin injected at time t which should be determined by solving the optimization problem; a_G and b_G are coefficients representing the impact of injected insulin on the BG dynamics. Also, t_f represents the finite horizon of the control problem which is usually 2h to 4h prediction window that corresponds to the bulk duration of action of a rapid acting insulin analogue such as aspart, lispro and glulisine. $g_{ref}(t)$ is the time dependent glucose reference value that can be chosen depending on the current state to avoid hypoglycemia or hyperglycemia. Initial condition is addressed by including g_0 which is the initial glucose level. Finally, u_{min} and u_{max} are the minimum and maximum allowed insulin amounts to be injected and g_{min} and g_{max} are the lower and upper bounds on the glucose level. $F(g(t)$ is a generic form for all possible cost functions. But, it is usually desired to minimize a summation form including both the distance to the reference glucose value and insulin injection effort. MPC has shown to be suitable for multivariate nonlinear systems such as the human body and it significantly gives better performance than PID control with patient-specific tuning. Several variations of MPC have been proposed in the literature. We briefly categorize them as follows:

Linear model predictive control (LMPC): There are several research studies that use linear model predictive controller. The work presented in [21] was the first in silico trial for linear model predictive approach which also showed better performance of MPC rather than PID controller in terms of limiting the oscillation of glucose levels. Research efforts presented in [31] and [9] were the first clinical investigations of linear model predictive algorithms in artificial pancreas that reported the superiority of using this approach over PID controller. in [21] Magni et al. in present an unconstrained MPC where the model is a linearization of a nonlinear parameters.

Researchers extend the MPC by defining new types of objective function and additional features to the problem formulation. Heusden et al. proposed using a priori patient characterization and fitting a linear control relevant model around the control point [20]. They also defined a new cost function named zone model predictive in contrast to previous studies in which only the distance to a target reference point is considered as the cost function. They consider a range for the BG level as the objective of the optimization and define the cost function as the minimum distant to the preferred zone. They have verified the robustness of the algorithm in silico and showed that the hypoglycemia is completely avoided even after meal disturbance. Lee et al. in [15] use new meal size estimation algorithm to the integrated AP and show how its performance is better than MPC-only case.

Non-linear model predictive (NMPC): Hovorka et al. in [28] present a nonlinear model and Bayesian techniques to estimate parameters in simulation studies. Clinical studies were performed under fasting conditions based on measurements that were delayed by 30 min to mimic the time lag associated with a sensor. The authors performed overnight studies using an algorithm and transferring results to a pump at 15 min intervals. The major result was a reduction in nocturnal hypoglycemia compared to standard pump treatment. Zarkogianni et al. also use a nonlinear model-predictive control for prediction of BG and control algorithm [19]. They have shown the usefulness of using this nonlinear MPC in silico for different meal profiles, fasting conditions, inter-patient variability and intraday variation.

Fractal model predictive control (FMPC): In spite of significant amount of work in PID and MPC, the complexity of BG dynamics has not been fully addressed. For instance BG is time dependent process that is influenced by various factors (meal size, exercise, psychological state, etc.). This has prompted a comprehensive multifractal investigation of BG dynamics [23] from publicly available data set [33]. The authors have shown how using fractional order controller leads more robust control over conventional integer order model predictive controller. They formulate the BG dynamics as a time dependent fractional order control problem and report the feasibility of implementation of fractional controller in hardware and report their results in terms of area and speed in field programmable gate array (FPGA). We compare the impact of applying fractional order controller to the conventional first order derivative controller. Fig 2 shows the outcome of applying both types of controllers to bring to some reference value which is $100mg/dL$ in this case. Unlike the expectation of integer order controller the final glucose value at the end of control horizon is much lower than the one expected.

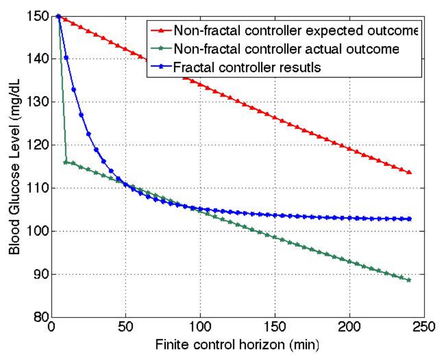


Figure 2 Performance of fractal and non-fractal MPC.

3.3 Assessment of Control performance

The ultimate goal of any closed-loop artificial pancreas controller is to minimize the complications resulting from poor BG control. Research studies that have evaluated closed-loop systems lasted at most several days. In these studies, *time when glucose is in the target range* is the most widely used metric to assess closed-loop performance. On the other hand, target glucose range differs in overnight and fasting condition (3.9–8.0 mmol/l) versus post-prandial condition (up to 10 mmol/l). The low BG index can be helpful in quantifying the duration and extent of hypoglycemia and other measures to assess severity of hypoglycemia and hyperglycemia have been proposed such as Grade score [25]. To sum up, in spite of existence of some FDA approved simulation environments ([4] and [8]), there is still significant need for establishing unified simulation sequences and defining precise criteria to compare different control algorithms. The same problem exists with the clinical studies in which there are no unified clinical conditions to be able to compare performance of different control algorithm.

4 Conclusions and future work

The ultimate goal of any medical cyber physical system is to use technology to increase the QOL for people. In type-1 diabetes mellitus, which is one of the fastest growing diseases globally, the patient's pancreas is not able to release insulin endogenously. As a result, the patient needs exogenous insulin in order to control BG to reduce acute and chronic complications. Recent technological advances have led to a paradigmatic shift in diabetes treatment by offering automatic and semi-automatic systems to replace traditional procedures to improve the QOL for diabetic people and let them forget about their disease.

Despite very advanced technologies in sensing and actuation technology, there is still a huge gap to fill for designing a robust AP, which comes from lack of accurate mathematical models and robust control algorithm. In this paper, we present main challenges and problems to be addressed in design of AP. Then, we present the state of the art control algorithms for closed-loop AP, which is mainly, composed of PID and model predictive control groups. As discussed in the paper, even with application of model predictive controller, which is proved to perform better than PID controller, clinical tests only prove simple situations e.g. over night or after meal conditions and more sophisticated glycemic control during meals and exercise is still challenging.

Future directions in research for developing more accurate mathematical models and control algorithm include investigation and application of the recently investigated time dependent fractional model for BG on more comprehensive data set for control purpose and also investigating this state of the art model for hormone levels e.g. insulin and glucagon. This is especially valuable for dual hormone closed-loop system [12, 11, 18], which is proven to be effective only when the predictions of the hormone levels are accurate [30]. More importantly, incorporating this mathematical model in state of the art software simulations, which are reference for evaluation of several control algorithms.

References

1 DB. Keenan at al. Continuous glucose monitoring considerations for the development of a closed-loop artificial pancreas system. *Journal of diabetes science and technology*, 5(6):1327–1336, 2011.

2 A. Ali et al. Pid controller tuning for integrating processes. *ISA transactions*, 49(1):70–78, 2010.

3 A. Ouattara et al. Blood glucose variability: a new paradigm in critical care? *Anesthesiology*, 105(2):233–234, 2006.

4 BP. Kovatchev et al. In silico preclinical trials: a proof of concept in closed-loop control of type 1 diabetes. *Journal of diabetes science and technology*, 3(1):44–55, 2009.

5 C. Chen et al. Recent advances in electrochemical glucose biosensors: a review. *RSC Advances*, 3(14):4473–4491, 2013.

6 C. Cobelli et al. Diabetes: models, signals, and control. *IEEE Reviews in Biomedical Engineering*, 2:54–96, 2009.

7 C. Ellingsen et al. Safety constraints in an artificial pancreatic β cell: An implementation of model predictive control with insulin on board. *Journal of diabetes science and technology*, 3(3):536–544, 2009.

8 CD. Man et al. Meal simulation model of the glucose-insulin system. *IEEE Trans on Biomedical Engineering*, 54(10):1740–1749, 2007.

9 D. Bruttomesso et al. Closed-loop artificial pancreas using subcutaneous glucose sensing and insulin delivery and a model predictive control algorithm: preliminary studies in padova and montpellier. *Journal of diabetes science and technology*, 3(5):1014–1021, 2009.

10 E. Renard et al. Closed-loop insulin delivery using a subcutaneous glucose sensor and intraperitoneal insulin delivery feasibility study testing a new model for the artificial pancreas. *Diabetes Care*, 33(1):121–127, 2010.

11 FH. El-Khatib et al. Adaptive closed-loop control provides blood-glucose regulation using dual subcutaneous insulin and glucagon infusion in diabetic swine. *Journal of Diabetes Science and Technology*, 1(2):181–192, 2007.

12 FH. El-Khatib et al. A bihormonal closed-loop artificial pancreas for type 1 diabetes. *Science Translational Medicine*, 2(27):27ra27–27ra27, 2010.

13 G. Sparacino et al. “smart” continuous glucose monitoring sensors: On-line signal processing issues. *Sensors*, 10(7):6751–6772, 2010.

14 GM. Steil et al. Modeling β-cell insulin secretion-implications for closed-loop glucose homeostasis. *Diabetes technology & therapeutics*, 5(6):953–964, 2003.

15 H. Lee et al. A closed-loop artificial pancreas using model predictive control and a sliding meal size estimator. *Journal of diabetes science and technology*, 3(5):1082–1090, 2009.

16 J. Cho et al. Cascade control strategy for external carbon dosage in predenitrifying process. *Water Science & Technology*, 45(4-5):53–60, 2002.

17 J. Kildegaard et al. Sources of glycemic variability—what type of technology is needed? *Journal of diabetes science and technology*, 3(4):986–991, 2009.

18 JR. Castle et al. Novel use of glucagon in a closed-loop system for prevention of hypoglycemia in type 1 diabetes. *Diabetes Care*, 33(6):1282–1287, 2010.

19 K. Zarkogianni et al. An insulin infusion advisory system based on autotuning nonlinear model-predictive control. *IEEE Transactions on Biomedical Engineering*, 58(9):2467–2477, 2011.

20 KV. Heusden et al. Control-relevant models for glucose control using a priori patient characteristics. *IEEE Transactions on Biomedical Engineering*, 59(7):1839–1849, 2012.

21 L. Magni et al. Model predictive control of type 1 diabetes: an in silico trial. *Journal of diabetes science and technology*, 1(6):804–812, 2007.

22 LBEA. Hoeks et al. Real-time continuous glucose monitoring system for treatment of diabetes: a systematic review. *Diabetic Medicine*, 28(4):386–394, 2011.

23 M. Ghorbani et al. A cyber-physical system approach to artificial pancreas design. In *Proc of IEEE/ACM/IFIP CODES*, page 17, 2013.

24 MJ. O’Grady et al. Changes in medicare spending for type 1 diabetes with the introduction of the artificial pancreas. *New York (NY): JDRF*, 2011.

25 NR. Hill et al. A method for assessing quality of control from glucose profiles. *Diabetic medicine*, 24(7):753–758, 2007.

26 P. Reichard et al. The effect of long-term intensified insulin treatment on the development of microvascular complications of diabetes mellitus. *New England Journal of Medicine*, 329(5):304–309, 1993.

27 PG. Jacobs et al. Development of a fully automated closed loop artificial pancreas control system with dual pump delivery of insulin and glucagon. In *Proc of IEEE EMBC*, pages 397–400, 2011.

28 R. Hovorka et al. Five-compartment model of insulin kinetics and its use to investigate action of chloroquine in niddm. *American Journal of Physiology-Endocrinology And Metabolism*, 265(1):E162–E175, 1993.

29 SA. Weinzimer et al. Fully automated closed-loop insulin delivery versus semiautomated hybrid control in pediatric patients with type 1 diabetes using an artificial pancreas. *Diabetes care*, 31(5):934–939, 2008.

30 SJ. Russell et al. Efficacy determinants of subcutaneous microdose glucagon during closed-loop control. *Journal of diabetes science and technology*, 4(6):1288–1304, 2010.

31 WL. Clarke et al. Closed-loop artificial pancreas using subcutaneous glucose sensing and insulin delivery and a model predictive control algorithm: the virginia experience. *Journal of diabetes science and technology*, 3(5):1031–1038, 2009.

32 DCCT Research Group et al. Epidemiology of severe hypoglycemia in the diabetes control and complications trial. *The American journal of medicine*, 90(4):450–459, 1991.

33 Diabetes Research in Children Network (DirecNet) Study Group et al. Accuracy of the glucowatch g2 biographer and the continuous glucose monitoring system during hypoglycemia. experience of the diabetes research in children network (direcnet). *Diabetes Care*, 27(3):722, 2004.

Automatic Resource Scaling for Medical Cyber-Physical Systems Running in Private Cloud Computing Architecture*†

Yong woon Ahn and Albert Mo Kim Cheng

Department of Computer Science, University of Houston
4800 Calhoun Road, Houston, Texas, U.S.A.
{yahn,cheng}@cs.uh.edu

Abstract

Cloud computing and its related virtualization technologies have become one of dominant trends to deploy software, compute difficult problems, store different types of data, and stream real-time video and audio. Due to its benefits from cost-efficiency and scalability to maintain server solutions, many organizations are migrating their server applications running on physical servers to virtual servers in cloud computing infrastructures. Moreover, cloud computing has enabled mobile and battery-powered devices to operate without strong processing power and large storage capacity. However, it is not trivial to use this trendy technology for medical Cyber Physical Systems (CPSs) which require processing tasks' requests to send instructions to the local actuator within specified deadlines. Since a medical CPS device monitoring a patient's vital signs may not have a second chance to recover from an erroneous state, achieving cost-efficiency with higher resource utilization in cloud computing may not be the ultimate goal to configure the healthcare IT infrastructure with medical CPS devices. In this paper, we focus on private cloud infrastructures with the fair resource sharing mechanism in order to run medical CPS applications. First, we introduce our medical CPS device model used for designing our cloud infrastructure following the Integrated Clinical Environment (ICE) standard developed by the Medical Device Plug-and-Play (MDPnP) project. Second, we investigate limitations to deploy CPS applications using existing auto-scaling mechanisms. Finally, we propose our novel middleware with a virtual resource sharing mechanism inspired by autonomic computing, and present its performance evaluation results simulated in the OpenStack private cloud.

1998 ACM Subject Classification C.2.4 Distributed Systems, C.3 Special-Purpose and Application-Based Systems, J.3 Life and Medical Sciences

Keywords and phrases Auto-Scaling, Cloud Computing, Medical Cyber-Physical System Device, Virtualization, Autonomic-Computing

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.58

1 Introduction

Cloud computing has become one of common technologies to provide unlimited computing experiences with small and battery powered mobile devices. Moreover, it provides other great advantages for deploying and maintaining server-side applications because of its flexibility to scale up and down computing and storage resources elastically. This flexibility is implemented by various hardware virtualization techniques which enable virtual machines (VMs) to be

* Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.
† This research is a part of the yPlatform project for medicine by Amerra Inc.

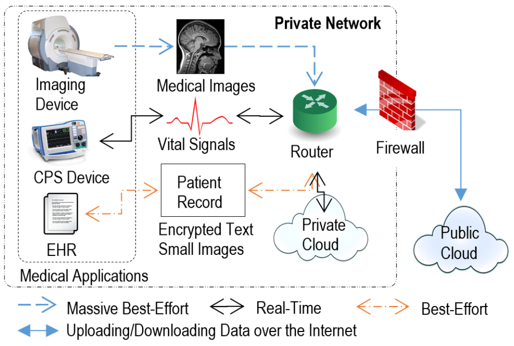


Figure 1 A hybrid cloud for medical devices and applications.

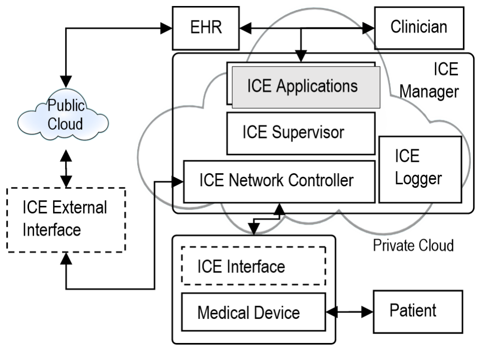


Figure 2 A hybrid cloud for medical CPS devices.

easily launched or terminated on demand to maintain the desirable Quality of Service (QoS) level for different types of common application. Like other IT areas, cloud computing is getting more attention from healthcare IT industries not only to reduce costs but also to improve patient care. By taking advantages from cloud computing, the healthcare cloud helps clinical environments remove their in-facility server rooms entirely by subscribing a public cloud Infrastructure as a Service (IaaS) or partially by deploying a private or hybrid IaaS [14]. Since medical records are generally very sensitive, and must be protected by highly secure physical facilities, multi-layered software or hardware network security systems, data encryption, and redundancy, operation of the in-facility server rooms can be more expensive and less secure to process and store medical data. More seriously, some medical imaging devices generate extremely massive data requiring huge data storages. To satisfy these requirements, a public IaaS could be a common solution because of its highly secure physical and virtual resources which also can be scaled up and down easily.

For non-real-time medical devices, public cloud computing solutions could be more suitable to provide healthcare IT systems by taking advantage of existing cloud computing solutions. However, these public and remote cloud solutions are too unpredictable to maintain the desirable QoS of medical CPSs because VM monitors used by their services basically do not know what type of medical application server is being loaded in a VM. Also, many uncertainties from transferring data over various networks would be serious issues in using public cloud solutions for medical CPSs. Although cloud computing service vendors provide Service Level Agreements (SLAs) covering cases where the user cannot access their subscribed computing and storage resources, they cannot guarantee that an application server responds to a source application within a specific task deadline. To overcome this limitation, a hybrid cloud architecture can be a solution. For non-real-time medical devices, we use VMs in a public cloud, and for medical CPS device, we use VMs in a private cloud locally located as shown in Figure 1. In this paper, we restrict a private cloud to operate as the in-facility server solution built with a stable network environment. No private cloud offered by public IaaS providers is considered as a private cloud in this paper. Each medical CPS device can transmit real-time tasks to this private cloud which can be specially configured to process real-time tasks with the highest priority. In this paper, we assume that each medical device can be interconnected via the ICE standard [4] which is one of the MDPnP projects [10] to support a cross-manufacturer medical device interoperability. Figure 2 shows data flows of CPS medical devices connected via ICE interfaces to deliver their real-time and life-critical tasks to ICE application servers running as a part of the ICE manager. The ICE supervisor is responsible for generating alarms to indicate that the required tasks cannot be processed with the current configurations of the ICE application servers. All ICE manager software and hardware components can be run in a private cloud, if these components can be emulated by virtualization technologies. If the medical device is a closed-loop CPS system, one or more ICE application servers are run as computing resources to process requests, and to respond action instructions to the source device. These ICE application servers also can provide user interfaces help clinicians check patient's states, and upload medical records to EHR systems as shown in Figure 2. Despite using a hybrid cloud solution, there are still possible issues to maintain the desirable QoS of medical CPS devices because most open sourced and commercialized private cloud solutions such as OpenStack [13], Eucalypus [12], and VMWare vCloud [16] have very similar auto-scaling mechanisms to adjust virtualized computing resources dynamically. Although these existing auto-scaling mechanisms are primary technologies to achieve the main goal to operate VMs on demand, these mechanisms are commonly performed by checking system performance metrics which human system administrators select. These system metrics can include the monitoring values of Virtual CPU (VCPU), memory, storage I/O, and network bandwidth. Although these metrics can represent a health status of each VM, they cannot represent whether all tasks are processed within their deadlines specified by medical CPS devices [9].

There are three major requirements to design and implement a private cloud for medical CPS devices using the ICE standard.

(a) A group of ICE application servers should always be ready to process all incoming real-time tasks from multiple CPS devices, and respond action instructions to the source CPS devices within specified deadlines.
(b) The private cloud must be designed to achieve a goal of higher physical and virtual resource utilization except any emergency case.
(c) Computing resources must be automatically adjusted by an implemented mechanism when a new CPS device is discovered, or one of connected devices increases bigger tasks.

These three requirements are essential, if we assume that the clinical environment adopts an ICE standard. Since computing powers of ICE application servers are generally configured before initializing the entire system, it requires that all ICE application servers always run to satisfy the worst-case scenarios even for not discovered medical CPS devices. In other word, there would be no advantage of the higher resource utilization by using cloud technologies. Also, even if the clinical environment adopts a private cloud, supporting the MDPnP standard would be another consideration. In this paper, we design middleware running in a private cloud infrastructure to provide a novel auto-scaling mechanism to preserve the cost-efficiency. Our middleware is performed independently in each VM with an ICE application server without modifying the ICE standard. Moreover, in order to implement an automatic service which can be self-optimized, we adopt the autonomic computing concepts to design our middleware [5].

The remainder of this paper is organized as follows. In Section 2, we introduce other researches working towards similar goals. Design and implementation of our solution are presented in Section 3 and 4. In Section 5, we evaluate our proposed middleware running in an OpenStack private cloud.

2 Related Work

For common real-time applications, S. Liu et al. proposed an on-line scheduling algorithm of real-time services for cloud computing in [7]. Their algorithm modifies the traditional utility accrual approach [3, 8] to have two different time utility functions (TUFs) of profits and penalties on executing tasks. One important assumption the authors made is that the timeliness with relative task deadlines would be a more realistic principle for most real-time applications than the absolute deadline guarantee for hard-real-time systems due to the nature of diverse network communication methods causing many uncertainties. Although their assumption about the timeliness is reasonable, this research does not consider cases of medical CPS devices which might be discovered at any time.

As we stated in the previous section, the most feasible solution to operate medical CPS devices in the cloud can be the auto-scaling mechanism to scale up the number of VMs to process real-time tasks without missing their deadlines, and to scale down to provide the cost-efficient and energy-saving physical data center alternative. In [9], M. Mao et al. proposed an auto-scaling mechanism considering task deadlines and budget constraints. Although their approach is based on deadline constraints to overcome downsides of the threshold-based stock auto-scaling mechanisms monitoring system metrics, a system administrator still has to adjust the configuration file manually whenever a new application needs to be connected. Also the authors did not consider cases with real-time medical applications which are possibly required to compress and transmit massive data to remote locations. To the best of our knowledge, our approach is the first attempt to adopt the autonomic computing concepts to operate the MDPnP environment.

3 System Design

In order to deploy medical CPS devices connected to a private cloud infrastructure, we first show common procedures to process CPS tasks. We assume that all CPS medical devices follow ICE standards to send and receive messages to and from an ICE supervisor, and we also assume that healthcare environments use a private cloud to configure the ICE manager.

3.1 Medical CPS Device

Figure 3 shows procedures to sample data from patient's body. In this paper, we focus on medical CPS devices which sample patient health condition data via various sensors, and compress them before transmitting to an ICE application server periodically. To interact between two different types of medical device, we use the device profile protocol introduced in [6]. This profile protocol is based on the ISO/IEEE 11073 Domain Information Model. In order to discover a new medical device, each device must send its profile protocol message to an ICE supervisor. This message includes device type, device health status, manufacture information, clock, device model, medical nomenclature, sampling period, event-trigger function, network interface, network protocol version, and so on. As Figure 3 shows, compressed data, Δ, would have a deadline, δ, which must be processed by the server before arriving the next period job to operate actuators. We call this data to the destined ICE application server real-time tasks. Also, if the medical CPS device did not receive this action command before sending the next period of sampled data, its device health status would become the *"fail"* state.

3.2 Proposed Middleware between an ICE Supervisor and ICE applications

In order to provide standard-compatible solutions for other existing ICE compliant systems, we do not change an ICE supervisor and any of its internal procedures. Our system must be

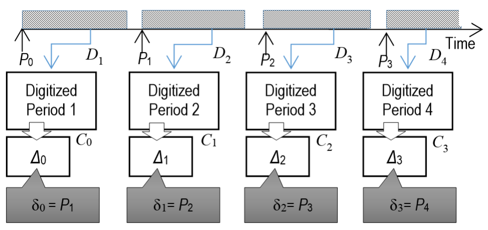


Figure 3 An example of real-time tasks collected by one sensor: P_i is the value of time starting i_{th} period, Δ_i is the number of time slots of the i_{th} compressed period, and D_i is the delay to digitize sampled or input original data for ith period.

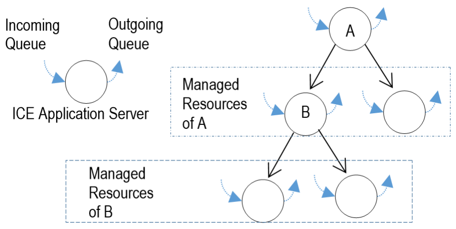


Figure 4 A parent VM with an ICE application server which has two VMs installing child ICE application servers which are dealt as managed resources in the autonomic computing concept.

run independently from the existing ICE manager. The ICE supervisor is responsible only to check whether the application servers and network interfaces can handle requests from newly discovered medical devices. If it detects any resource shortage or inappropriate profiles from the device, the supervisor indicates an alarm [4]. Our goal is to protect a medical device from being rejected due to shortage of computing resources when using a private cloud. To design our middleware to manage VMs running ICE application servers, we use autonomic computing concepts to implement the self-management with four essential attributes such as self-configuration, self-healing, self-optimization, and self-protection [5]. After initializing the ICE manager, our middleware must accumulate knowledge from the previous history of processing data and responding action instructions to decide whether the ICE supervisor needs more resources or not.

In our middleware design, each VM is a node in a tree data structure shown in Figure 4. Each node also can become an autonomic manager as shown in Figure 5. Our autonomic manager has managed resources which can be other ICE application servers running in different VMs. Our middleware is running in each VM and checking the health state of its ICE application server usages by counting how many tasks missed their deadlines for the predefined duration. There are four steps to accumulate knowledge and adjust managed resources in our autonomic manager. In the monitoring step, our autonomic manager collects managed resource states from its child VMs by checking the number of missed deadlines. In the analyzing step, the autonomic manager calculates the number of VMs for the next iteration, and sends a request to the next step to make a new plan such as task assignments for ICE application servers in child VMs, if it would improve system's overall utilization. In the planning step, the new plan is setup to launch or terminate one or more VMs, if it is requested by the previous step. Finally, in the executing step, the autonomic manager would execute this plan by calling private cloud management functions such as jClouds [2]. From these four steps, the managed resource can be adjusted by our autonomic manager running as middleware without human interventions. Also, since each middleware runs in different VM

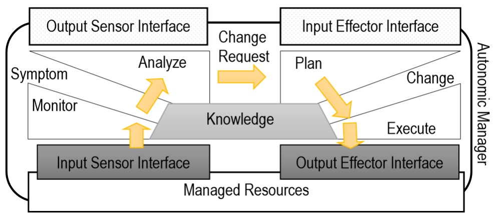


Figure 5 A generic architecture for autonomic manager.

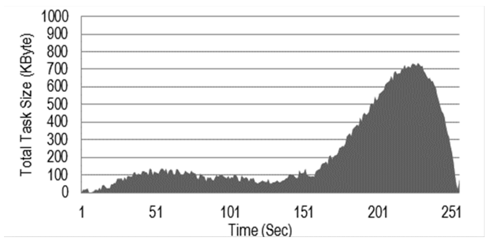


Figure 6 Real-time tasks used for the simulation.

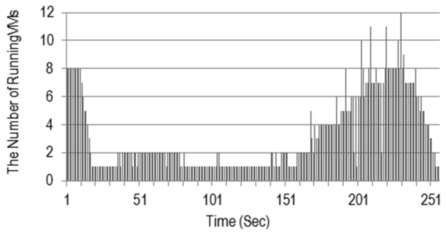


Figure 7 The number of VMs processing data.

independently, we can avoid any possible bottleneck when having one centralized coordinator to manage all VMs. To avoid any issue from having less knowledge at the beginning, the managed resources must be started with a sufficient number of VMs to process the worst case scenario, and gradually this number of VMs would decrease if no new medical device is discovered, or existing medical CPS devices stably transmit their real-time tasks periodically.

4 Implementation

We are implementing our system design on the OpenStack cloud which is one of the most well-known open source cloud infrastructure. Currently, this OpenStack environment only is used for medical CPS devices, and other medical applications use Amazon EC2 [1] public cloud. To implement our middleware approach, we wrote a medical CPS device simulator in Java. This simulator uses sampled ECG data from the MIT-BIH database [11] as its input, and sends compressed data to the ICE application servers periodically, and must receive action commands before sending other data to control its virtual actuator. In order to launch and terminate VMs, we use jClouds to control VMs.

5 Performance Evaluation

To evaluate our approach, we use the OpenStack Grizzly version running with Intel Xeon E3 Quad-Core CPU, 16 GB RAM, 500GB SAS HDD, two network interface cards, and Ubuntu server operating system. We use the Ubuntu 12.04 VM image [15] to run ICE application server simulators, and each VM is launched with 512Mbyte RAM, one VCPU, and no local storage. We assume that the ICE supervisor knows all current states of ICE application servers, and works well to coordinate tasks for every connected medical CPS device. We setup each ICE application server processes 100Kbyte ECG data per one second in the incoming task queue. Figure 6 shows our total workload receiving from medical CPS device simulators. At the beginning, we only have one medical device, but it increases its data size slightly between 40 and 80 seconds. This scenario can represent unknown network errors or

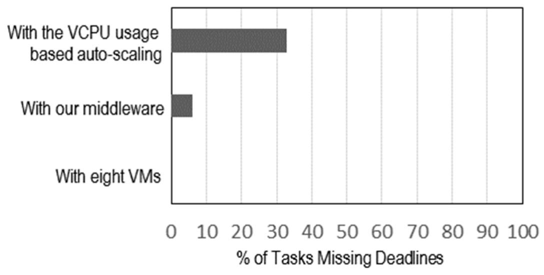


Figure 8 The percentage of subtasks missing deadlines.

possible cases intentionally increasing sensor's resolution. New medical devices are started to be discovered from 150 seconds, and at 210 seconds, all eight medical devices start sending tasks to request instructions. Tiny spikes shown on the slope mimic minor network errors such as congestions. Figure 7 shows the number of VMs running ICE application servers. When initializing our system, the ICE manager starts with eight VMs to prepare the worst case scenario receiving tasks from all eight devices. In our simulation, all eight VMs are initially structured as a balanced tree, and four autonomic managers in VMs have child VMs as managed resources. Until ten seconds, each autonomic manager checks its manager resources whether they process data or not. If they are not used to process tasks for this amount of time, the autonomic manager plans to terminate its managed resources. After ten seconds, it terminates its managed resources to achieve the higher server utilization. After 150 seconds, new medical devices are getting discovered and send tasks. Since our autonomic managers work independently from the ICE supervisor, the total number of VMs could be more than eight from our simulation for a moment. However, we believe that this issue can be fixed by enabling communications between our autonomic managers to balance the tree quickly. We compared our autonomic manager with the threshold-based auto-scaling mechanism which monitors VCPU usages. If it detects over 80% of VCPU usages, it launches a new VM in our simulation. Figure 8 shows the percentage of tasks missing their deadlines. As we can see, if we run eight VMs all the time without considering resource utilization and cost-efficiency, no deadline would be missed. However we lose the higher resource utilization of using cloud technologies. As using the VCPU usage based auto-scaling mechanism, ICE application servers missed deadlines of 240 tasks of total 735 subtasks as average values. But, after applying our mechanism, it only missed 45 subtasks even without a separated physical server only to run VMs. Our middleware improves 81.25% of the system reliability. We can see some tasks missed deadlines during the second catastrophic overflow after 150 seconds because of delays to launch new VMs. To overcome this drawback, we are adding workload prediction algorithms to our system to launch new VMs less frequently.

6 Conclusion and Future Work

Recently, cloud computing with virtualization technologies has become a big trend providing a new way to release software as a service and processing calculation-intensive tasks on remote VMs because it is very scalable, reliable, and cost-efficient with the on-demand computing model. Although most types of computing system and application can be migrated to cloud computing services, there are several serious issues when running medical CPS device applications. First, currently exiting cloud computing solutions with virtualization technologies do not have any special mechanism to support real-time and sensor-based applications. Second, to achieve the higher resource utilization when using the cloud, an auto-scaling mechanism is essential. However, existing performance metric based auto-scaling mechanism is not suitable to support medical CPS devices because of its inability to meet task deadlines. In order to support deadline-critical medical CPS devices following the MDPnP standard, we propose novel middleware running in a private cloud infrastructure with the ICE manager. This middleware with the autonomic manager uses self-management concepts from an autonomic computing architecture to develop our proposed auto-scaling mechanism for reserving virtual resources to meet timing constraints without human intervention. From our simulation, we have demonstrated that our approach can scale up when new devices are discovered. This research is still in progress as we develop more reliable workload prediction algorithms for already connected medical devices dynamically changing their task sizes. These prediction algorithms would enable our autonomic manager to launch new VMs even before the ICE supervisor requires more resources.

References

1 Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.
2 Apache jCloud. `http://jclouds.apache.org/`.
3 R. K. Clark. *Scheduling dependent real-time activities.* PhD thesis, Carnegie Mellon University, 1990.
4 ASTM F2761-2009. Devices in the Integrated Clinical Environment.
5 Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.
6 Tao Li and Jiannong Cao. Safety-ensured coordination of networked medical devices in mdpnp. Technical report, Hong Kong Polytechnic University, 2012.
7 S. Liu, G. Quan, and S. Ren. On-line scheduling of real-time services for cloud computing. In *Services (SERVICES'10), 2010 6th World Congress on*, pages 459–464, 2010. `http://dx.doi.org/10.1109/SERVICES.2010.109`.
8 C. D. Locke. *Best-effort decision making for real-time scheduling.* PhD thesis, Carnegie Mellon University, 1986.
9 M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *Proc. 11th IEEE/ACM Int'l Conf. Grid Computing (Grid'10)*, pages 41–48, 2010.
10 Medical Device Plug and Play Program. `http://www.mdpnp.org/`.
11 MIT-BIH Database Distribution. `http://ecg.mit.edu/`.
12 D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus opensource cloud-computing system. In *CCA'08: Cloud Computing and Its Applications, IEEE*, 2008.
13 OpenStack. `http://www.openstack.org/`.
14 B. Sotomayor, Ruben S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, Vol. 13, 2009.
15 Ubuntu Cloud Image. `http://cloud-images.ubuntu.com/precise/current/`.
16 VMWare vCloud. `http://www.vmware.com/products/vcloud-hybrid-service`.

Modeling of Reconfigurable Medical Ultrasonic Applications in BIP

Stefanos Skalistis and Alena Simalatsar

Rigorous System Design (RiSD) Laboratory
École Polytechnique Féréral de Lausanne (EPFL), 1015 Lausanne, Switzerland
`{stefanos.skalistis,alena.simalatsar}@epfl.ch`

Abstract

Medical ultrasonic imaging applications require high quality of images produced in real-time often with limited resources available. Deadlock-freedom and confluency must be guaranteed to ensure the correctness of the applications, while feasibility and optimality properties are required to provide the best Quality of Service (QoS) within available resources. In this paper we introduce BIP (Behavior-Interaction-Priority) framework components as main building blocks to model such applications in a correct-by-construction manner. Based on those components we model a reconfigurable multi-mode processing pipeline for ultrasonic imaging that supports QoS management by topology reconfiguration. Finally, as a proof of concept, we present a simple quality controller as a well-triggered component, which when combined with the processing pipeline can manipulate the quality of image processing.

1998 ACM Subject Classification D.1.3 Concurrent Programming, D.2.11 Software Architectures, F.1.1 Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Reconfigurable Pipelines, Quality of Service, Medical Ultrasonic Applications, Component-based System Design, Behavior-Interaction-Priority Modal Flow Graphs

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.66

1 Introduction

Ultrasonic imaging is widely used in medicine [7] as a diagnostic technique to provide static images (e.g., B-mode) and dynamic changes (e.g., based on Doppler effect). Static imaging provides visualization of muscles and internal organs, to capture their size, structure and any pathological lesions. Ultrasonic imaging based on Doppler effect [6] is widely used to visualize motion, in particular blood flow for diagnosis, such as blood clots, heart valve defects, aneurysms and many others. All these applications require high quality of images produced in real-time. Often ultrasonic devices are used in trauma and first aid cases as well as for remote diagnosis. This drastically limits the available resources for ultrasound computation algorithms, which requires Quality of Service (QoS) management. Moreover, deadlock-freedom and confluency must be guaranteed to ensure correctness of the computational and controlling algorithms.

B-mode ultrasonic imaging, chosen as a case-study in this paper, can be performed in different ways, also called modes or processing pipelines, which may achieve the output image with different quality characteristics and resource requirements. Thus, quality of final images depends first of all on the chosen processing mode. Moreover, the components of a chosen processing pipeline may perform the computation with different quality outcome. Components processing quality levels can be controlled by certain parameters set, e.g. the cutoff quality of a low-pass filter by adjusting the order of the filter. Based on that, we distinguish two approaches for QoS management driven by i) topology, and ii) components quality levels.

BIP (Behavior-Interaction-Priority) framework [1] provides essential means for rigorous system design. We employ a particular branch of BIP framework, namely well-triggered modal flows [3], which ensures correctness by construction and encompasses a synchronous computation model. Well-triggered modal flows requires no additional coordination with the BIP engine compared to the classical BIP. This is an important advantage, since additional coordination implies potential computational overhead, which may be critical for a system with limited resources. A well-triggered modal flow is composed of synchronized components, which successively perform computation steps. It defines the behavior of the system. Well-triggered modal flows are considered to be most fitting to model real-time multimedia systems.

In the frame of development of a scalable low-power, high-performance and trusted ultrasonic platform, feasibility, optimality and quality control become of utmost importance. In terms of timing, feasibility implies that no processing task must miss its deadline. While operating under this constraint, the system must make optimal use of its resources and time budget and at the same time provide the maximum possible quality for the produced images. Existing work [5] formulates and addresses the problem of QoS control of real-time multimedia systems in a feasible and optimal manner.

In this paper we present the model of a reconfigurable multi-mode ultrasonic application. The application is modeled as a modal flow graph composed of well-triggered components, that guarantees deadlock-freedom and confluence by construction, with basic QoS management. We define four essential types of the components required to build a reconfigurable multimedia application, namely *processing*, *buffer*, *accumulator* and *mode-selection* components. We also reason about the composition of such components showing that it results in a well-triggered composite component, which is essential for overall deadlock-freedom and confluence. Each component of this modal-flow comprises a configuration port that allows external component reconfiguration by a specific set of parameters sent from a separate controller component. The configuration ports are used for both structure reconfiguration by managing mode-selection components and buffer sizes as well as specific quality level control of processing elements.

The rest of the paper is structured as follows; Section 2 provides background information regarding BIP, modal flow graphs, and QoS. In Section 3 the problem is presented through our case study of a real-life ultrasonic application. Following, Section 4 describes our generic approach for modeling image processing applications, with the use of modal flow graphs, that provides QoS management and guarantees deadlock-freedom and confluence. Section 5 illustrates our approach on a case study, emphasizing the QoS management by application structure. Finally, in Section 6 we conclude and present ideas for future work.

2 Background

In this section we first present the essential background information, which starts with a general description of the BIP (Behavior-Interaction-Priority) framework. Then we talk about a specific part of this framework, namely modal flow components and graphs. We conclude the section with the description of QoS management technique for multimedia systems.

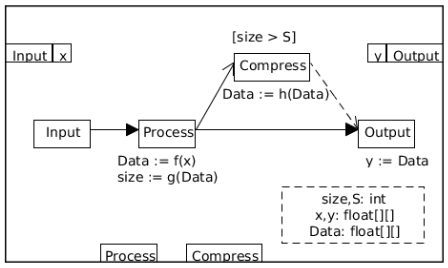


Figure 1 Example of Modal flow component.

2.1 BIP Framework

BIP [1] is a framework that provides essential means for rigorous design and modeling of heterogeneous systems. The BIP framework allows the modeling of systems as composition of atomic components by encompassing three layers: Behavior, Interaction and Priority. The behavior of each atomic component is described as 1-safe priority Petri-net extended with data and ports. The composition of these components is supported by the Interaction and Priority layers. Interactions between components, which are specified by connectors [2], are used to define the way systems are composed of components. Priorities are used to eliminate conflicts between interactions and thus restrict non-determinism.

In BIP the execution is driven by the BIP engine which has all the necessary information about the components, their connectors and the associated priorities. At every execution cycle, the engine receives information about the set of active ports for each of the atomic components. It then computes the set of interactions that have maximal progress and if there are more than one, picks one of them non-deterministically. The engine notifies the components of the chosen interaction and computes the associated data transfers. Each of the notified components then performs the associated transition and updates its state.

2.2 Modal Flow Graphs and Well-triggered Components

Modal flow components and modal flow graphs [3] are part of synchronous BIP and are used to model systems that are composed of synchronous components. Modal flow components are a particular class of 1-safe priority Petri-nets, extended with data and ports. As a result, modal flow components can directly be translated into 1-safe Petri-nets, following the semantics defined in [3]. These Petri-nets define the behavior of each component.

In modal flow components the dependency relations between events/actions are expressed using three kind of causal dependencies [3]:

- **Strong:** An event q strongly depends on p if the occurrence of p must always be followed by q. That is p and q can not happen independently.
- **Weak:** An event q weakly depends on p if the occurrence of p may be followed by q. That is either p happens or the sequence pq happens.
- **Conditional:** An event q conditionally depends on p if both p and q occur, then p must be followed by q. Otherwise p and q can occur independently.

In Figure 1, an example of a modal flow component is depicted. In the figure, solid arrows with filled arrowheads depict strong dependencies, solid arrows with normal arrowheads depict weak dependencies and dashed arrows depict causal dependencies. The rectangles represent ports and their associated data. This notation will be used in the rest of the paper.

In this example, a component is presented that can receive an input, process it and depending on the size decide to compress it, or not, before delivering the output. In Figure 1, it is depicted that *Output* strongly depends on *Process*, which, in turn, strongly depends on *Input*. This means that the component provides an output after processing, which, in turn, may occurs only after the component has received an input. It must be noted that this implies that the component, once the input is received, will obligatory process it and consequently produce the output.

Furthermore, *Compress* weakly depends on *Process*, which means that compression may occur after the processing. This depends whether the associated guard of that port, enclosed in square brackets, validates to true. Also, *Output* conditionally depends on *Compress*, that is if both occur then compression must happen before delivering the output. Finally, underneath each port the update functions are placed, which describe the associated computations.

Well-triggered components are modal flow components for which deadlock-freedom and confluence are guaranteed by construction iff the following constraints are met [3]:

- The causal dependency graph has no cycles.
- Each port has either strong or weak causes, but not both.
- Each port has at most a minimal strong cause.
- Each port that has strong causes, must have its guard true.

The example presented in Figure 1 is a well-triggered modal flow component. Interestingly, these constraints, e.g. graph acyclicity, are easy to check, either manually or automatically.

Well-triggered components can be composed based on interactions among their ports. The result of the composition is the modal flow graph that defines the behavior of the whole system. It must be noted that composition is a partial operation. This means that the composition of well-triggered components does not guarantee that the resulting modal flow graph will be well-triggered as well. Thus, in order to guarantee deadlock-freedom and confluence the constraints must be validated on the final modal flow graph.

As a result of confuency of well-triggered modal flow graphs, the existence of the BIP engine, that is present in classic BIP, is redundant and unnecessary. The engine is considered redundant since the confluent behavior of synchronous systems results in a deterministic execution of interactions. The engine is unnecessary as it introduces an extra processing overhead. This overhead originates from the fact that the BIP engine at each execution cycle computes the set of maximal interactions. Based on the confluent behavior of synchronous systems, this can be replaced by a single predefined scheduling of interactions out of all the possible ones. Finally, as a consequence of confluency in BIP components, code generation is possible.

2.3 QoS

There exist different approaches to systems design that address different levels of systems criticality. Currently, these systems are classified as safety-critical or best-effort systems. Safety-critical systems require high level of correctness, meaning no violation of critical constraints, e.g. timing constrains, when all the deadlines must be met. Engineering of such systems uses a conservative approach based on the worst-case execution time, which is often largely over-estimated and, therefore, implies not optimal or even redundant use of available resources. The best-effort systems are more relaxed in terms of critical constrains, where occasional miss of deadlines will not cause any hazardous outcomes. The design of such systems is mainly targeting efficient and optimal use of available resources.

Design of medical ultrasonic systems require meeting both critical and best effort properties. Such engineering approach is addressed in [5]. The authors proposed a method for fine grain QoS management of real-time applications, which allows the run-time adaptation of overall system behavior. The proposed approach provides control over three main properties:

- **Feasibility**, that is no deadline is missed;
- **Optimality**, that maximizes the use of available resources, e.g. provide the best QoS within specified resource constraints;
- **Smoothness** of quality levels, that is of particular importance to the multimedia applications.

Such QoS management considers a single-threaded process network application, which cyclically performs data transformation. Possible QoS levels and platform-dependent timing information of processing components must be provided as an input. The coordination of components execution is then controlled by a *controller* that monitors the progress of the computation within each cycle.

3 Reconfigurable Multimedia Systems

Generally, image processing and its applications follow the *input-process-output* paradigm. More specifically, an image processing application can be analyzed in several stages of computation, each of which receives the result of the previous computation stage as input, processes the input and delivers the output to the next computation stage.

This paradigm inherently enforces the components of an application to form processing pipelines. It is important to note that different pipelines may achieve their outputs with different quality characteristics, resource requirements and/or implementation. For example, in Section 3.1, different ultrasound imaging pipelines for B-mode are presented. Another typical example of that are pipelines that perform the required processing on raw images and then compress them, compared to pipelines that first compress the images and then perform the required processing [9]. The former, usually requires more resources but achieves better quality, while the latter requires less resources but results in degraded quality.

It is apparent that even if both pipelines deliver same outputs, their implementation may differ substantially since required processing actions performed on images have different nature.

To this end, we distinguish two different approaches for quality management:

- **QoS by structure**: In this approach, different quality outcomes are achieved based on the mode of processing pipeline (i.e. processing first, compression first, etc).
- **QoS by precision**: In this approach, having a concrete pipeline, different quality outcomes are achieved based on the parameters of the processing components (i.e. low compression-rate).

In this work we focus on providing a framework to model image processing pipelines that can be reconfigured by combining different modes in order to encompass QoS by structure.

3.1 Case Study

There exist several types of imaging applications that are based on ultrasound waves, including A-mode, B-mode (or 2D-mode), Doppler mode, Harmonic mode, and many others [4]. B-mode (brightness mode) ultrasound application is the most-known imaging technique due to its vast applicability in several diagnostic domains.

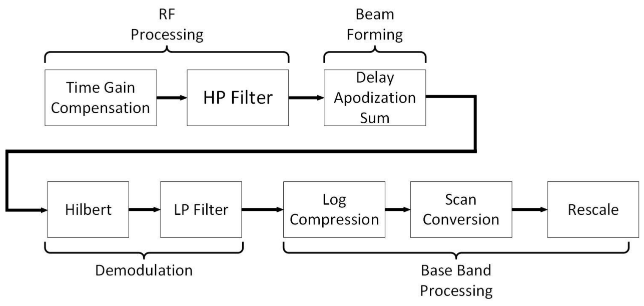


Figure 2 Baseline B-mode Processing.

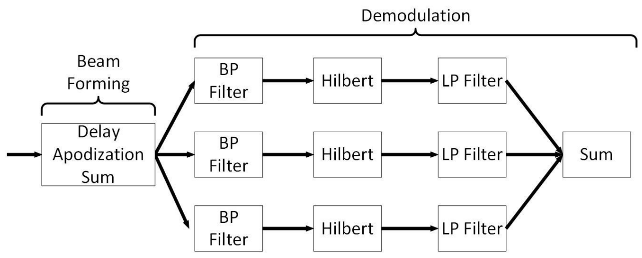


Figure 3 Frequency compounding.

In B-mode an array of transducers, called the probe, emits a beam of ultrasound waves and scans a plane through the body, which is then transformed into a two-dimensional image on a screen. There are variations of this technique that affect not only the way the retrieved signal is processed, but also the quality of the output image. Such variations include, but are not limited to:

- the mode of processing (baseline, in-phase and quadrature, frequency compounding, spatial compounding);
- the shape of the probe (linear, convex, phased array, etc);
- the type of the beam wave (planar, curved, etc);
- the angle of steering of the beam.

In this paper we focus on the algorithmic part of the variations of ultrasonic techniques, namely modes of processing. Generally, B-mode imaging consist of four processing stages; i) RF processing, ii) Beamforming, iii) Demodulation, iv) Baseband processing. These stages are comprised of several components, the ordering of which may slightly vary depending on the processing mode. More specifically, we consider the following modes:

1. *Baseline B-mode Processing*: This is the typical processing pipeline as depicted in Figure 2.
2. *Frequency Compounding*: In this mode multiple bands are separately demodulated and then summed. As a result, the output image has less high-frequency noise but also lower resolution. This mode differs from the *Baseline B-mode Processing* only in the Demodulation stage. The modified Demodulation is depicted in Figure 3.
3. *Spatial Compounding*: In this mode, instead of a single beam, several beams are emitted with different types (e.g. steered, curved, etc). Since multiple firings are used to reconstruct a single image, this procedure increases resolution. This mode differs from the *Baseline B-mode Processing* only in the Baseband stage. The modified Baseband processing is depicted in Figure 4.

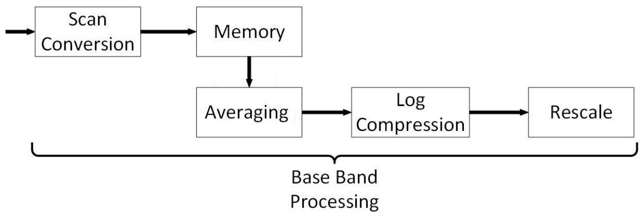


Figure 4 Spatial compounding.

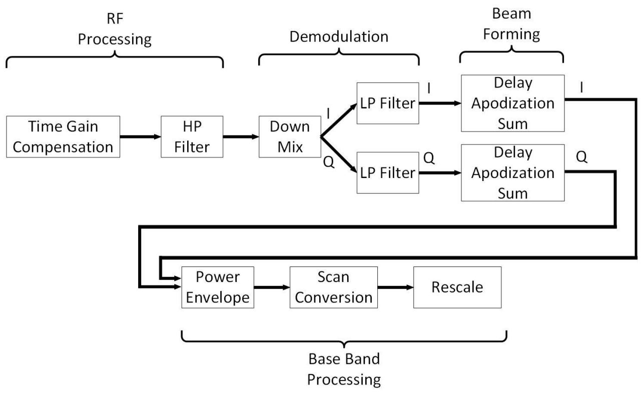


Figure 5 I/Q B-mode compounding.

4. *I/Q mode*: In this mode, in contrast with the aforementioned modes, the Demodulation stage occurs before Beamforming. The output quality depends on the subject under study (e.g. normal tissue, abdomen, etc.) as well as the configuration parameters of the components. The whole pipeline for this mode is depicted in Figure 5.

The quality of the resulting image depends on various aspects. It may depend on parameters set of each component. For example, a low-pass filter with higher filter order, i.e. cut-off quality, will result in a signal less contaminated with frequencies higher than cut-off frequency. Alternatively, the quality of the final image is also determined by the choice of the B-mode pipeline.

In this paper we exploiting the possibility to achieve optimal quality outcomes by reconfiguring the mode of operation. In this case study, we also focus on minimizing the program memory, that is to avoid redundant components where possible. The reason is that such over-provisioning may have a direct impact on implementation cost (i.e. in a FPGA implementation).

4 Components framework

In this section we present the framework of the classified components required to model ultrasound image processing that allow to provide QoS not only by structure, but also support QoS by precision.

We also show that their composition results in well-triggered components. This formal specification guarantees that the final pipeline, which is composed of these components, is deadlock-free and confluent.

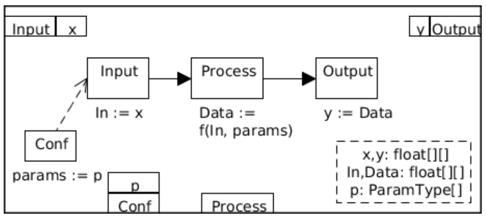


Figure 6 Processing Component.

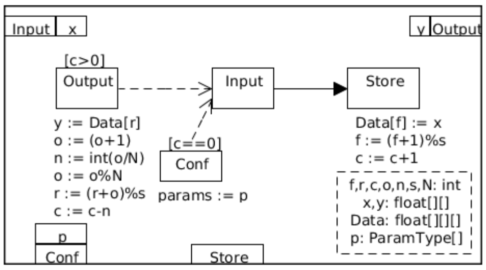


Figure 7 N-read Buffer Component of size s.

In order to model reconfigurable multimedia systems, we consider the following components:

- *Processing Components*: These components are the building blocks that follow the *input-process-output*. They are responsible for applying the necessary transformation to the input in order to get the desired output. They may have multiple inputs and/or outputs but to simplify we will refer them as input and output, respectively.
- *Memory Components*: These components are necessary for storing images between stages that produce multiple outputs which have to be processed separately. For this reason they must support multiple reads of the same value. They may have multiple inputs, but only a single output. There are several possible different types of memory components that can be modeled, such as FIFO, LIFO, etc. Following in this section, an N-read buffer is formally defined.
- *Accumulating Components*: These components are necessary for combining multiple images into a single one (e.g. different color channels). They may have multiple inputs, but only a single output. Following in this section, an N-write accumulator with a single input and a single-output is defined.
- *Mode-selection Components*: These components allow the reconfiguration of the pipeline. It has a single input and multiple outputs, one of which is active at any time, based on the mode.

Based on this components, it is possible to model complex pipelines that can be reconfigured and provide QoS by structure. In order to support QoS by precision, each of the aforementioned components must have a configuration port that will allow modification of the processing parameters.

4.1 Processing Component

In Figure 6, the main building block of a pipeline, the processing component, is presented as a well-triggered component. This component is responsible to receive an input, process it and finally output the result in one computation step. At the beginning of each computation step the component can receive new configuration parameters from the *Conf* port. It is assumed that the component is initialized with default parameters.

In ultrasound applications the processing component, receives at the *Input* port a 2-dimensional array that represents the image. Similarly, the component exposes the image at the *Output* port. The *Process* port is needed to signal that the processing has been performed. Finally, the *Conf* port is used to configure the component with appropriate parameters.

The strong dependencies between these ports enforce the *input-process-output* paradigm. The conditional dependency between the *Conf* port and the *Input* port implies that if both

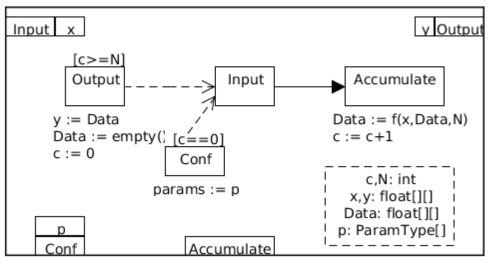


Figure 8 *N*-write Accumulator.

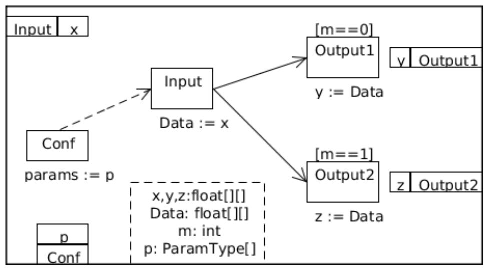


Figure 9 Mode-selector Component.

happen at same computation step, the configuration of the component must occur before the component receives the input. That means that a) the component can receive new configuration if there is no input to process; b) the component can receive the input and process it if there are no new configuration parameters; c) the component will receive first the new configuration parameters and then process the input based on these new parameters.

4.2 N-read Buffer Component

Figure 7 presents a buffering element, of size s as a well-triggered component. Following the standard notation, this component receives an image from the *Input* port and stores it internally, by copying the image to the memory and adjusting the front pointer f and the element counter c. The *Output* port exposes the oldest image stored internally, and when the image is read N times, the rear pointer r and the element counter c are adjusted. To achieve that behavior the output counter o and the next index n are used. The *Output* port is active, i.e. can be executed, only when the element counter is greater than zero, that is there is at least one image stored. Similarly, the *Conf* port is active only when there is no image in the buffer.

Storing and retrieving an element from a buffer can occur in arbitrary orderings. It is assumed that the buffer can execute both in a single computation step. Based on this assumption, *Input* and *Output* can occur independently, but if they occur in the same computation step *Output* precedes *Input*.

Finally, based on this well-triggered component, an unbounded buffer can be modeled as well by simplifying the update functions for the position pointers f and r, $f{:=}f{+}1$ and $r{:=}r{+}1$ respectively. A typical 1-read buffer can also be modeled by eliminating the update functions for o, n and replace them with the value of 1.

4.3 N-write Accumulator Component

Similar in logic to the buffer component is the accumulator component presented in Figure 8. This component when it receives a new input, it accumulates (e.g sum, average, select min/max, etc.) the new input with all the previously received inputs. When it has received N inputs, it outputs the result and empties the data in order to receive new inputs.

4.4 Mode-selector Component

Another important component required to model multi-mode processing pipelines is that of the mode-selector. The role of this component is to direct the received input to the correct output and thus change the processing pipeline.

In Figure 9, a selector component that supports two modes is presented. In this component the *Output1* and *Output2* ports weakly depend on the *Input* port. This means that the input can be received without producing any output, in the case when the selected mode is not valid. As in all previous cases the *Conf* precedes the *Input* port for the aforementioned reasons.

As with the processing component, the mode-selector component can be extended in multiple outputs, in a similar manner.

4.5 Composition of components

The framework components presented earlier, namely processing, buffer, accumulator and mode-selector components, are well-triggered. In order to construct deadlock-free processing pipelines from such components we have to reason if their composition results in well-triggered components as well. Composition is performed by merging the interacting ports and inheriting the dependencies from all the interacting ports. To check that the result is still well-triggered the constraints presented in Section 2.2 should hold.

Following, a descriptive reasoning is presented, regarding that the result of the composition of the framework components is well-triggered. A more formal proof can be found in [8].

Processing – Processing: Combining two processing components results in a well-triggered component if these are connected in series, that is the output of the former becomes the input of the latter. The resulting component actually performs the two processing steps sequentially, and is well-triggered.

Processing – Mode-Selector: Combining a processing component with a mode-selector component, in terms of connecting the respective output and input, results in a well-triggered component, since the mode-selector has no strong causes and no cycle is created.

Processing – Buffer: The same holds for combining the output port of the processing component with the input port of a buffer. Their composition is well-triggered, as the processing component has only strong causes and the input port of the component has only one strong cause and no weak causes.

Processing – Accumulator: Similarly, combining the output of a processing component with the input of an accumulator produces a well-triggered component. The resulting graph has no cycles and the accumulator component has only one strong cause and no weak causes.

Accumulator – Mode-Selector: The composition of an accumulator component with a mode-selector component, by connecting their respective output and input ports, results in a well-triggered component. Although, the output port of the accumulator has a guard, the mode-selector has no strong causes and the resulting graph has no cycles.

Accumulator – Processing: Combining, on the other hand, the output port of the accumulator component with the input port of a processing component is not straightforward. The output port of the accumulator has a guard, but this does not violates the constraints, since when the two ports are merged, the resulting port has no strong causes. Thus, the resulting component is a well-triggered component.

Buffer – Processing: Similarly, combining the output port of the buffer component with the input port of a processing component results in well-triggered component, as after the combination the resulting port has no strong causes.

Mode-Selector – Processing: In the same manner, combining the output port of a mode-selector component with the input port of a processing component results in a well-triggered component despite the guards.

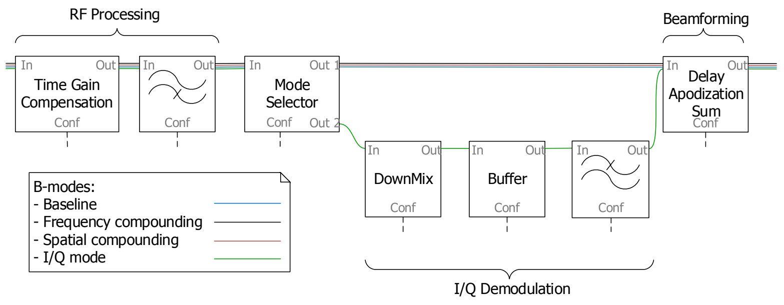


Figure 10 RF Processing, Demodulation(I/Q mode), Beamforming.

Mode-Selector – Buffer: Finally, the same holds for combining the output port of a mode-selector component with the input port of a buffer. Their composition is well triggered, as no cycles are created and the guards belong to ports with no strong causes.

It must be noted that there are several possible ways to combine these framework components. Nevertheless, the most interesting combinations, that can be used in processing pipelines, are those presented above.

5 QoS by Topology Reconfiguration

QoS by topology reconfiguration concerns the management of quality solely through the reconfiguration of the pipeline structure and not the configuration of its components. Figures 10-12 depict three consecutive parts of the reconfigurable pipline that consolidates the B-mode pipelines presented in Section 3.1.

This consolidated pipline enables quality management by merely altering only the topology of the pipeline based on the mode of operation, that is without changing the parameters of the processing components.

Every component of this pipeline is represented by one of the framework components, namely mode-selector, buffer, accumulator and processing component, that were defined in Section 4.

Each component has an In (resp. Out) port that corresponds to *Input* port (resp. *Output*) as defined for the processing components. Interactions between components are depicted with solid lines connecting participating ports. Each component has a *Conf* port that can be used for QoS managment by precision by altering the processing parameters of the components. Apparently, parameters adjusted through the *Conf* port for the mode-selector, accumulator and buffering components do not affect the quality per-se, as they do not perform any kind of processing.

In Figure 10, the first two components perform the RF processing, which is the same for all modes. After that, for all modes except from I/Q, the Beamforming is computed (by the delay-apodization-sum component). As stated in Section 3.1, in I/Q mode the Demodulation occurs before Beamforming. This is depicted in the lower branch (green line) in Figure 10 where the mode-selector component is used to switch among the modes. Further, that branch rejoins the normal pipeline in order for Beamforming to be performed.

Figure 11 depicts the Demodulation for Baseline, Frequency and Spatial compounding modes. In Baseline (blue line) and Spatial compounding (red line) the demodulation is performed through a hilbert-transformation followed by a low-pass filter. On the other hand,

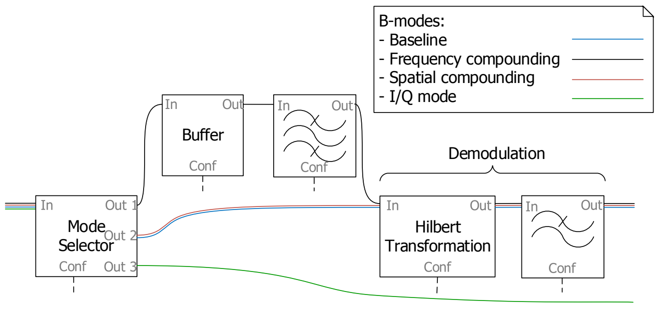


Figure 11 Demodulation (for all modes except I/Q).

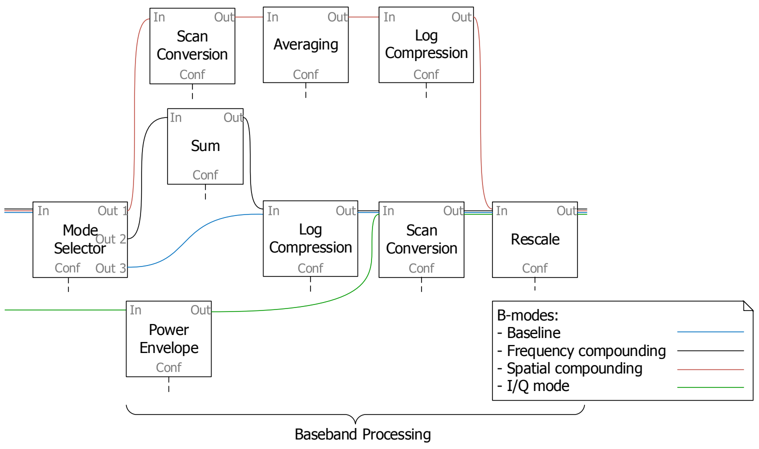


Figure 12 Baseband Processing (for all modes).

in Frequency compounding (black line) the image is passed multiple times through different band-pass filters, i.e spliting the image into several new images of different frequency-bands. The splitting is performed by using a buffer and reconfiguring the cut-off frequencies of the band-pass filter, rather than using multiple fixed filters, in order to reduce the number of components and thus reduce program memory. These images are then demodulated in the same manner as Baseline and Spatial compounding modes.

Figure 12 depicts the Baseband processing for all modes. In Spatial compounding, which is depicted on the top branch (red line), the multiple firings, which are required in this mode, are averaged after the scan-conversion. After averaging several images, the processing continues with log-compression followed by the rescaling so as to produce the final output. Similarly, in Frequency compounding (black line), the different images produced previously, by the band-pass filter, have to be summed-up. Then the same processing as with Baseline B-mode (blue line) is following. Finally, in the bottom branch, the I/Q signals are combined in the power-envelope and in the similar manner with the Baseline B-mode produces the final output.

It must be noted that the components Sum, Averaging and Power Envelope can not be modeled as processing components. Instead, they are modeled as accumulators. Similarly, the buffers present in this pipeline should be unbounded N-read buffers, where each them has an appropriate value of N.

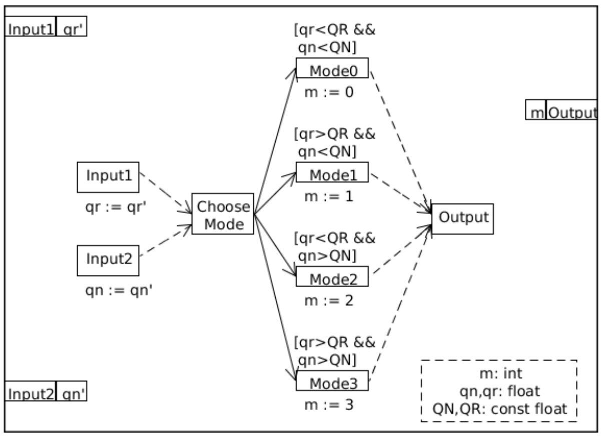


Figure 13 Simple QoS controller as well-triggered component.

This consolidated pipeline supports both QoS by structure based on the choice of a particular pipeline mode, and QoS by precision, where each component can be reconfigured separately. It also consists of well-triggered components, which can ensure deadlock-freedom and confluence. In Figures 10-12 there are some components with input ports belonging to more than one interaction, which is not allowed in modal flows in general. This can be resolved by manually implementing that part of the system. Of course, this may result in a non deadlock-free system and need to be further studied.

Finally, in Figure 13, a simple QoS controller is presented as a proof of concept that such a controller can be designed following the well-triggered paradigm. As such, the controller can be combined with the aforementioned pipeline and thus have a fully deadlock-free and confluent system that supports QoS management. This controller has two input ports, through which it receives the values *qr* (quality with respect to resolution) and *qn* (quality with respect to noise). The controller chooses the appropriate mode by comparing these values with the thresholds *QR, QN*. To make this more clear, the thresholds *QR, QN* distinguish the "*high*" and "*low*" quality for noise and resolution, respectively, while the input values are the desired quality levels. For example, if the requirements for quality with respect to resolution is "*high*" and the requirements for quality with respect to noise is "*low*", then the controller chooses mode *m=1*, which is then trasmitted to the mode-selector components that perform the choice of the porcessing mode.

6 Conclusion & Future Work

Ultrasonic imaging applications require high quality of images produced in real-time with limited resources available. In this context, feasibility, optimality and quality control are of significant importance, but the safety-critical nature of such applications requires guarantees that the system will be deadlock-free and confluent. We present an approach to model such applications using a synchronous computation model. Our approach is based on Modal Flow Graphs, which is a formalism that encompasses a synchronous computation model and guarantees by-construction deadlock-freedom and confluence provided the system satisfies some easy-to-check structural constraints.

There are two aspects of QoS management; *QoS by precision* is based on adjusting the parameters of some components of the computation chain, whereas *QoS by structure* is based on changing the topology of the computation chain. We have presented a model of the pipeline that consolidates four modes of ultrasound B-mode processing and provides quality control by structure through pipeline reconfiguration, as well as supports quality control by precision through the adjustment of computational parameters at the component level.

We have introduced framework components, which are well-triggered modal-flow components, that can be used to build reconfigurable multimedia pipelines. We have identified the conditions that must be satisfied by the interconnection structure among the components in order to preserve deadlock-freedom and confluence. With the case study we have demonstrated how the processing pipline of the ultrasoic application can be composed out of these framework components.

Finally, we have presented a simple QoS controller as a well-triggered component which when combined with a reconfigurable pipeline results in a fully deadlock-free and confluent system that supports QoS management by topology reconfiguration.

As part of on-going and future work, in the context of our case study, we are investigating parameters (e.g. variable cut-off frequencies of the filters, levels of saprcity of the computation matrixes, etc.) and constraints (power, time) that affect the QoS management. Based on that, we are planning to extend the quality controller to take into account parameters and constraints of the underlying platform and provide optimal use of resources.

Acknowledgements. The work described in this paper is part of the UltrasoundToGo project. The aim of this project is to develop a scalable low-power, high-performance, trusted platform for 3D portable ultrasound imaging systems.

References

1 Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, Joseph Sifakis, et al. Rigorous component-based system design using the BIP framework. *IEEE Software*, 28(3):41–48, 2011.

2 Simon Bliudze and Joseph Sifakis. The algebra of connectors: Structuring interaction in BIP. In *Proceedings of the 7th ACM/IEEE International Conference on Embedded Software*, EMSOFT'07, pages 11–20, New York, NY, USA, 2007. ACM.

3 Marius Dorel Bozga, Vassiliki Sfyrla, and Joseph Sifakis. Modeling synchronous systems in BIP. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT'09, pages 77–86, New York, NY, USA, 2009. ACM.

4 Richard SC Cobbold. *Foundations of biomedical ultrasound*. Oxford University Press, USA, 2007.

5 Jacques Combaz, Jean-Claude Fernandez, Thierry Lepley, and Joseph Sifakis. Qos control for optimality and safety. In *Proceedings of the 5th ACM International Conference on Embedded software*, pages 90–99. ACM, 2005.

6 Zahra Keshavarz-Motamed, Julio Garcia, Emmanuel Gaillard, Romain Capoulade, Florent Le Ven, Guy Cloutier, Lyes Kadem, and Philippe Pibarot. Non-invasive determination of left ventricular workload in patients with aortic stenosis using magnetic resonance imaging and doppler echocardiography. *PLoS One*, 9(1), 2014.

7 Sonia H. Contreras Ortiz, Tsuicheng Chiu, and Martin D. Fox. Ultrasound image enhancement: A review. *Biomedical Signal Processing and Control*, 7(5):419 – 428, 2012.

8 Stefanos Skalistis and Alena Simalatsar. Modeling of Reconfigurable Medical Ultrasonic Applications in BIP. Technical report, EPFL IC IIF RiSD, 2014.

9 Xiang Xie, GuoLin Li, ZhiHua Wang, Chun Zhang, DongMei Li, and XiaoWen Li. A novel method of lossy image compression for digital image sensors with bayer color filter arrays. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 4995–4998. IEEE, 2005.

A Domain Specific Language for Performance Evaluation of Medical Imaging Systems*

Freek van den Berg, Anne Remke, and Boudewijn R. Haverkort

Design and Analysis of Communication Systems, University of Twente
P.O. Box 217, 7500 AE, Enschede, The Netherlands
{f.g.b.vandenberg,a.k.i.remke,b.r.h.m.haverkort}@utwente.nl

Abstract

We propose iDSL, a domain specific language and toolbox for performance evaluation of Medical Imaging Systems. iDSL provides transformations to MoDeST models, which are in turn converted into UPPAAL and discrete-event MODES models. This enables automated performance evaluation by means of model checking and simulations. iDSL presents its results visually. We have tested iDSL on two example image processing systems. iDSL has successfully returned differentiated delays, resource utilizations and delay bounds. Hence, iDSL helps in evaluating and choosing between design alternatives, such as the effects of merging subsystems onto one platform or moving functionality from one platform to another.

1998 ACM Subject Classification B.8.2 Performance Analysis and Design Aids

Keywords and phrases Domain Specific Language, Performance Evaluation, Simulation, Model Checking, Medical Systems

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.80

1 Introduction

Medical imaging systems (MIS) are used to perform safety critical tasks. Their malfunctioning can lead to serious injury [1]. The safety is, among others, significantly determined by their performance, since imaging applications are time critical by nature. Predicting the performance of MIS is a challenging task, which currently requires the physical availability of such system in order to measure their performance. However, a model-based performance approach would allow to predict the system's performance already during early design and can thereby shorten the design cycle considerably.

Interventional X-ray (iXR) systems are MIS that dynamically record high quality images of a patient, based on X-ray beams. Design decisions in this domain are of various kinds, such as the possibility of merging of subsystems onto one platform, moving functionality from one to another platform, and assessing whether the system is robust against minor hardware changes. This paper investigates the use of a model-based approach to obtain insight in system performance.

We have decided to build iDSL, a domain specific language and toolbox for performance evaluation of Medical Imaging Systems, on top of MoDeST [8], which recently has been extended to support the modelling and analysis of Stochastic Timed Automata (STA) using PRISM [17] and UPPAAL [18] as well as discrete-event simulation using MODES. This

* This research was supported as part of the Dutch national program COMMIT, and carried out as part of the Allegio project under the responsibility of the Embedded Systems Innovation group of TNO, with Philips Medical Systems B.V. as the carrying industrial partner.

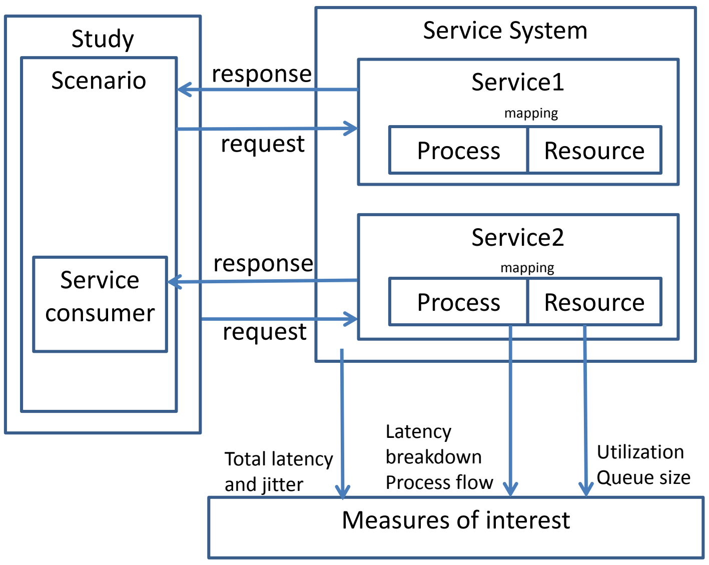


Figure 1 Conceptual model of a service system. Measures of interest are obtained using scenarios.

decision has been taken because of the expressiveness of STA and because MoDeST allows to use both analytical and simulation techniques.

We have designed the Domain Specific Language iDSL tailored towards MIS. iDSL adheres to the Y-chart philosophy [15], which separates the application from the underlying computing platform. It further uses hierarchical structures like the performance evaluation tool HIT [3]. And finally, iDSL can automatically generate design alternatives. We have constructed automated transformations from iDSL to different MoDeST model variants, each taking full advantage of the capabilities of the underlying evaluation tools, i.e., PRISM, UPPAAL and MODES. While these tools have been used widely for performance evaluation of embedded systems [13, 12, 16], to the best of our knowledge they have not been used for evaluating the performance of MIS. Finally, we use GraphViz [7] and GNUplot [20] to present performance outcomes graphically.

As for related work, [14, 19] apply model checking with UPPAAL on real-time medical systems to address safety. A study in which PRISM is used, addresses quantitative verification of Implantable Cardiac Pacemakers [5], which are time critical systems. [11, 21] evaluate the performance of iXR systems based on the Analytical Software Design (ASD) method.

The Octopus Toolset [2] provides various tools for the modelling and analysis of software systems in general, whereas iDSL is specifically designed for MIS. Earlier work [10, 22] proposed a simulation-based approach using POOSL [6], leading to average values.

This paper is further organized as follows. Section 2 describes the conceptual model of iDSL. Section 3 specifies the constructs and relations that constitute the iDSL language. Section 4 covers the functionality and usage of the iDSL tool. Section 5 concludes the paper.

2 Conceptual model

This section describes the conceptual model that forms the basis of iDSL (see also Figure 1).

A **service system**, as depicted in the upper right block, provides services to service consumers in its environment. A consumer can send a request for a specific service at a certain time, after which the system responds with some delay.

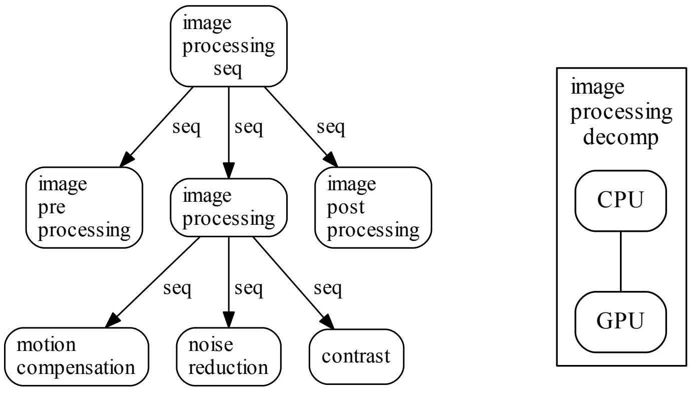


Figure 2 The IP ProcessModel (left) and IP ResourceModel (right) visualization are automatically generated from the iDSL code.

A **service** is implemented using a process, resources and a mapping, in accordance with the Y-chart philosophy [15]. A **process** decomposes high-level service requests into atomic tasks, each assigned to resources through the **mapping** (from which we abstracted in the figure). Hence, the mapping forms the connection between a process and the resources it uses. **Resources** are capable of performing one atomic task at a time, in a certain amount of time. When multiple services are invoked, their resource needs may overlap, causing concurrency and making performance analysis more challenging.

A **scenario** consist of a number of invoked service requests over time to observe the performance behaviour of the service system in specific circumstances. We assume service requests to be functionally independent of each other. That is, service requests do not affect each other's functional outcomes, but may affect each other's performance implicitly.

A **study** evaluates a selection of systematically chosen scenarios to derive the system's underlying characteristics. Finally, **measures of interest** define what performance metrics are of interest, given a system in a scenario. Measures can either be external to the system, e.g., throughput, latency and jitter, or internal, e.g., queue sizes and utilization.

3 Language constructs

We now demonstrate how to use iDSL by implementing an example Image Processing (IP) system. We have included the grammar of the iDSL language as reference at the end of the paper (see Figure 9). The iDSL language contains six sections, i.e., *Process*, *Resource*, *System*, *Scenario*, *Measure* and *Study*. The former three sections specify the functioning of the service system, whereas the latter three sections describe the way the system performance is assessed. iDSL transforms automatically into MoDeST [8] models and we therefore define its semantics in terms of MoDeST code. In what follows, we provide an iDSL instance per section, and the belonging MoDeST code that serves as semantics. In some cases, iDSL also provides an automatically generated visualization using GraphViz.

3.1 Process

A process decomposes a service into a number of atomic tasks, implemented in iDSL using a recursive data structure with layers of sub-processes. At the lowest level of abstraction, the

Table 1 Process: iDSL and MoDeST code.

iDSL Process code

```
Section Process
  ProcessModel image_processing_application
   seq image_processing_seq {
     atom image_pre_processing load 50
     seq image_processing {
       atom motion_compensation load 44
       atom noise_reduction load uniform(80 140)
       atom contrast load 134 }
     atom image_post_processing load 25 }
```

Generated MoDeST Process code

```
process image_processing(){
  motion_compensation(44);
  noise_reduction(Uniform(80,140));
  contrast(134) }
process image_processing_seq(){
  image_pre_processing(50);
  image_processing();
  image_post_processing(25) }
process image_processing_application_instance(){
  generator_image_processing_application?;
  image_processing_seq() }
```

Table 2 ResourceModel: iDSL and MoDeST code.

iDSL ResourceModel code

```
Section Resource
  ResourceModel image_processing_PC
    decomp image_processing_decomp  {
      atom CPU rate 2
      atom GPU rate 5 }
    connections { ( CPU , GPU ) }
```

Generated MoDeST ResourceModel code

```
process machine_call_GPU(real taskload){
  machine_GPU_start! {= sync_buffer=taskload =};
  machine_GPU_stop? }
process machine_GPU(){
  real taskload;
  machine_GPU_start? {= taskload=sync_buffer =};
  delay (taskload / 5 )
  machine_GPU_stop!;
  machine_GPU() }
```

atomic tasks each have a load, i.e., an amount of work, such as the number of CPU cycles.

The process for the example (Table 1 and Figure 2, left) combines *seq* and *atom* constructs. At its highest level, it consists of a sequential task that decomposes into an atomic task "pre-processing" with load 50, a sequential task "processing" and an atomic task "post-processing" with load 25. At a lower level, the sequential task "processing" consists of three atomic tasks named "motion compensation" with load 44, "noise reduction", and 'contrast" with load 134. The load of "noise reduction" is drawn from a uniform distribution on [80,140], at execution time.

In MoDeST, these hierarchies are implemented using layered processes, and the loads as parameters that are used later. The process is triggered via a generator through binary communications.

iDSL additionally supports the process algebraic constructs for parallelism (*par*), non-deterministic choice (*alt*), probabilistic choice (*palt*) and abstraction, as well as a mutual exclusion (*mutex*) to permit at most one process instance at a time on a certain process part.

3.2 Resource

In iDSL, a resource is defined as recursive hierarchical structure consisting of *decomp* and *atom* constructs, and a binary relation that defines which resources are connected.

The *decomp* construct is used to create decomposable resources, whereas the *atom* construct is used to specify atomic resources. They have a rate that specifies how much load they can process per time unit, e.g., the number of CPU cycles per second. Resources that are connected can perform operations in sequence for one process. The connections further enhance the way resources are visualized and enable high-level input validations.

We model the resource in our example as a composite resource (Table 2 and Figure 2, right). It consists of two atomic resources, i.e., a "CPU" with rate 2 and a "GPU" with rate 5. Additionally, the "CPU" and "GPU" are connected. In the MoDeST code, two processes per resource are created of which we have included the "GPU". A resource is implemented using binary communications to handle concurrency and a delay to represent the resource being in

Table 3 System: iDSL and MoDeST code.

iDSL System code

```
Section System
  Service image_processing_service
  Process image_processing_application
  Resource image_processing_PC
  Mapping assign { ( image_pre_processing, CPU )
    ( motion_compensation, CPU )
    ( noise_reduction, CPU )
    ( contrast, CPU )
    ( image_post_processing, GPU) }
```

Generated MoDeST System code

```
process motion_compensation(real taskload){
  machine_call_CPU(taskload) }
process image_post_processing(real taskload){
  machine_call_GPU(taskload) }
```

Table 4 Scenario: iDSL and MoDeST code.

iDSL Scenario code

```
Section Scenario
  Scenario image_processing_run
   ServiceRequest image_processing_service
     at time 0, 400, ...
   ServiceRequest image_processing_service
     at time dspace(offset), dspace(offset)+400,
```

Generated MoDeST Scenario code

```
process init_generator_image_processing_service
  () { delay (0)
      generator_image_processing_service() }
process generator_image_processing_service(){
  clock c; tau {= c=0 =};
  alt{
   :: generator_image_processing_application!
   :: delay(1) tau // time-out };
  when urgent(c >= (400-0) )
    generator_image_processing_service() }
```

use, i.e., processing a process. The self-recursion ensures that the resource runs forever. The delay is the quotient of the load and rate, e.g., CPU cycles divided by CPU cycles per second leads to seconds. The second process (with prefix machine_call) abstracts communications from the process layer. The MoDeST code reveals that concurrency is currently resolved using non-deterministic choices, in a non-preemptive manner.

3.3 System

A system consists of one or more services. In our example (Table 3), we construct an overall system with one service that combines the already defined process and resource (Figure 2). By defining an additional mapping, we connect them to form a service (Figure 3). In MoDEST, each mapping assignment results into a process that calls a resource.

3.4 Scenario

A scenario is defined as a bundle of services, on one system, that are individually requested over time (Table 4). The times of the requests are defined in terms of the first and second request, respectively 0 and 400 in the example here. Inter-request times are assumed to be constant, 400 in the example. To illustrate the modelling flexibility, we have added another set of service requests, including two *dspace* function calls that are constant within a design instance (to be explained later). In MoDeST, two processes handle the timing. The first (with prefix init) performs the initial delay once. The second then loops forever, with period of the inter-request time, triggering the process once per loop. When the service system fails to respond to a request immediately, a time-out occurs that drops the request.

3.5 Measure

Measures define what performance metric(s) one would like to obtain, given a system in a certain scenario. Different measures might call for different techniques to obtain them, e.g., simulation, model checking or numerical analysis. To illustrate our approach, we specified two measures (Table 5), based on two methods, i.e., MODES [8] based simulations and

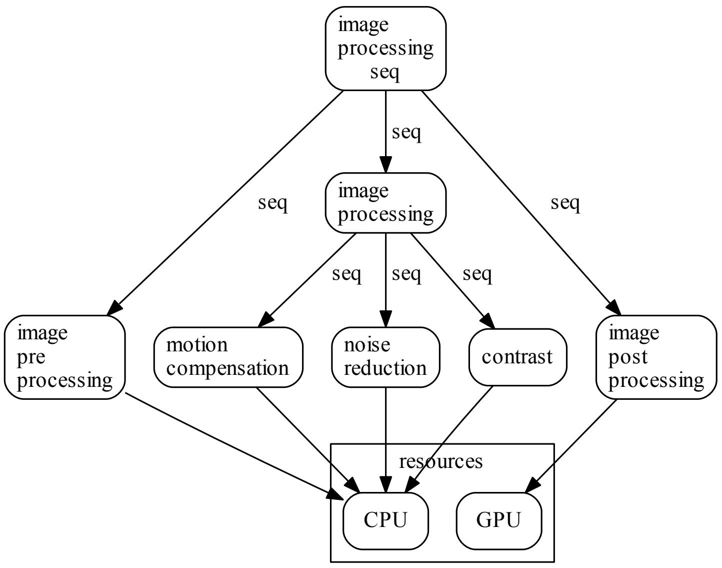


Figure 3 The IP Service visualization, which is automatically generated from the iDSL code.

UPPAAL [18] based model checking. The former uses Stochastic Timed Automata (STA) as its underlying model, while the latter uses Timed Automata (TA). We create specific MoDeST code (Table 5) for each case to combine the STA's expressiveness and the TA's model checking capability.

First, **simulations** provide response times, for a given number of simulations of a certain length. We use 1 run of length 280 in the example. Simulations additionally provide insight in resource utilizations and latency breakdowns. To eliminate non-determinism, we use an as soon as possible (ASAP) scheduler for time, and a uniform resolution for choice [9], which are fixed parameters that iDSL provides to MODES. The ASAP scheduler makes sure that whenever an action is possible, it is performed immediately. The uniform resolution selects one out of multiple actions to perform when their underlying distribution is not specified, with equal probabilities.

In MoDeST, we extend the already given code with both measurement points and properties, for both the latencies and utilizations. Each (sub)process is enclosed by a stopwatch to register a latency value, whereas an actual property retrieves this value for a single latency. Resources are augmented with a cumulative delay counter, retrieved by means of a property after some time, viz., an arbitrary 10000 in the example.

Table 5 Measure: iDSL and MoDeST code.

iDSL Measure code

```
Section Measure
  Measure ServiceResponse times
    using 1 runs of 280 ServiceRequests
  Measure ServiceResponse absolute times
    for any ServiceRequest
```

Generated MoDeST Measure code for MODES.

```
process image_processing(){
  tau {= stopwatch_image_processing = 0,
       image_processing_done = false =};
  ...
  tau {= image_processing_done = true,
       counter_image_processing++ =};
  tau {= image_processing_done = false =} }
property property_latency_image_processing =
  Xmax( stopwatch_image_processing |
      stopwatch_image_processing_done &&
      counter_image_processing==1 );
process machine_GPU(){ ...
  delay ( taskload/ 5 )
  tau {= util_counter_GPU+= ( taskload / 5) =};
  .. }
property property_utilization_CPU =
  Xmax ( util_counter_CPU/10000 | time==10000 );
```

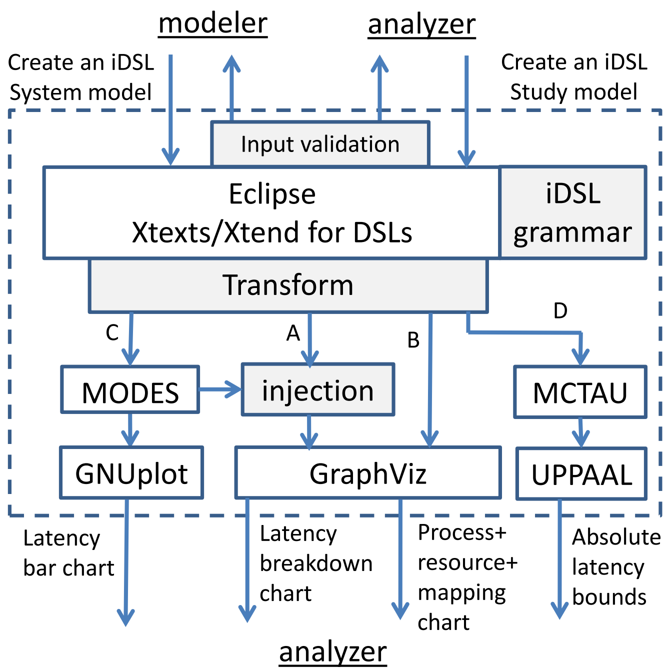


Figure 4 The iDSL tool chain overview. A modeller and analyser create an iDSL model based on the iDSL grammar. The iDSL tool transforms this model into MoDeST and GraphViz models, leading to performance measures to be evaluated by the analyser.

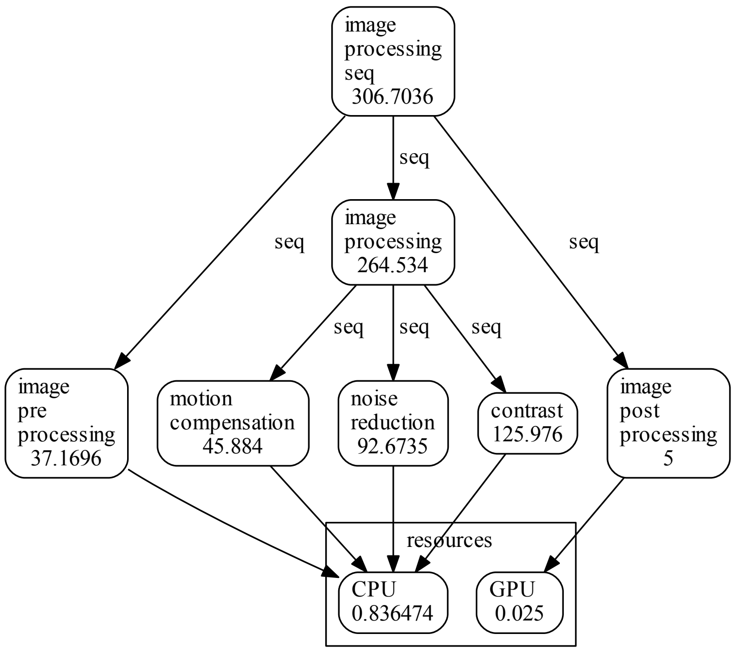


Figure 5 The latency breakdown chart and utilization (offset=0), based on MODES simulation results, which is automatically generated from the iDSL code.

Table 6 Binary search for bounds, pseudo code.

LB: Compute lower bounds, pseudo code

```
LB (lbound,ubound){
   if (ubound==lbound) return lbound
   check_value=(lbound+ubound)/2
   UPPAAL (p = probability(latency<check_value))

   if ( p=0 ) LB (check_value,ubound)
   else       LB (lbound,check_value) }
```

Compute lower bounds, execution trace

```
LB(0,1024)  -> LB(0,512)   -> LB(0,256) ->
LB(128,256) -> LB(128,192) -> LB(128,160) ->
LB(144,160) -> LB(152,160) -> LB(156,160) ->
LB(158,160) -> LB(159,160) -> LB(159,159) -> 159
```

Table 7 Study: iDSL and MoDeST code.

iDSL Study DSL code

```
Section Study
  Scenario image_processing_run
    DesignSpace
      (offset {0, 20, 40, 80, 120, 160, 200) }
```

Generated MoDeST Study code

```
real sync_buffer;
closed par{
 :: do{image_processing_application_instance()}
 :: do{image_processing_application_instance2()}
 :: init_generator_image_processing_service()
 :: init_generator_image_processing_service2()
 :: machine_CPU()
 :: machine_GPU() }
```

Second, **model checking** leads to the absolute minimum and maximum response times, given a system and scenario. It does not require parameters in the iDSL language, because its results are universal. The lower and upper bounds are *valid* when they are respectively lower and higher than all possible outcomes. They are *strict* when additionally the distance between them is minimal, i.e., the lower bound is the highest valid one and vice versa. iDSL can return bounds that are both valid and strict.

For model checking, iDSL "downgrades" STAs to TAs [9] automatically, thereby, replacing real numbers by integers, probabilistic choice and infinite distributions by non-deterministic choice, and removing some performance measuring variables to reduce the state-space size. For instance, the uniform function in the process (see Table 1), represented by a continuous probability function in STAs, becomes a non-deterministic, finite choice.

While TAs only support properties with boolean expressions, the absolute values cannot be retrieved using single properties. Therefore, we have equipped iDSL with a binary search algorithm that leads to a solution in $\mathcal{O}(\log(n))$, with n the size of the search range. The algorithm consists of two functions, i.e., a LB function to compute lower bound values and a UB function for higher bound values.

LB is a recursive function (Table 6, top) with two parameters, the lower and upper bound of the current range of values. The stop criterion, i.e., the lower and upper bound value are the same, ends the recursion by returning the lower bound value. Otherwise, the range is halved in two parts by taking the average value of the lower and upper bound. UPPAAL is queried with this value to determine in which half of the range the lower bound is located. A recursive call of LB then takes place using the right range half as parameter. The UB function operates in a similar fashion.

To illustrate the functioning of LB, we apply it on the case with one image processing system. We start by selecting the initial range of values. Since the algorithm is of $\mathcal{O}(\log(n))$ and the choice of n does therefore not affect the workload much, it is advised to overestimate the size of the range. Based on simulation results, we choose [0:1024] to be our initial range. The execution trace (Table 6, bottom) conveys 12 recursive calls before the final value 159 is finally obtained. This means that the one image processing system will never display a service response time smaller than 159, in the given scenario.

3.6 Study

Finally, a study forms a collection of scenarios that one would like to analyse in an automated manner. This is principally done by summing up one or more scenarios (Table 7). Individual MoDeST models are created for each scenario, which each contain a main parallel process to initiate all process model threads, generators and the resources involved.

We conclude with a design space, a shorthand way to specify a set of similar scenarios. In our example, we vary the starting time offset of one of the service-request sequences to be 0, 20, 40, 80, 120, 160 and 200. For this purpose, we create a design space in the study and enumerate the desired values. After this, the *dspace* function can be used for the offset parameter as done in the system section (Table 3). As a result, seven similar scenarios, one corresponding to each offset value, are created that vary where and only where the *dspace* function is used.

4 Tool and solution chain

This section covers the functioning of our iDSL tool (illustrated in Figure 4). iDSL requires two user roles to be fulfilled, the *modeller* and the *analyser*. The modeller constructs a model of a real system and the analyser specifies measures to perform. Execution of the model then generates artefacts with performance metrics to be investigated by the analyser.

4.1 Modelling

A modeller and analyser interactively create an iDSL instance in the Eclipse IDE for DSL Developers[1], adhering to iDSL's grammar. Input validation comprises syntax checking and advanced checks, e.g., for unique naming and non-circular definitions. Additionally, warnings and information boxes are displayed, e.g., when the design space is large. The modelling ends with the creation of a valid iDSL model.

4.2 Execution

Created iDSL models are then automatically transformed into two kinds of GraphViz specifications (Figure 4, A+B) and two kinds of MoDeST models (Figure 4, C+D). Transformations are written in Xtend and generate text output, based on iDSL instance constructs.

Some GraphViz specifications are performance unrelated and provide a visual presentation of the processes, resources and mappings of the system (as already shown in Figures 2 and 3). They are turned into a *process+resource+mapping chart* using the GraphViz tool.

The remaining GraphViz specifications have placeholders to contain performance numbers and form one input of the *injection* step. Some MoDeST models are executed in the MODES simulator and lead to latency and utilization numbers. The high-level latencies per instance are transformed into a *latency bar chart* by GNUplot.

The latencies at different process levels and utilizations form the second input of the *injection*. The remaining MoDeST models are executed in UPPAAL, via MCTAU [4], to obtain *absolute latency bounds*.

The injection step takes a GraphViz specification with placeholders and MODES performance numbers as input. By simply injecting the performance numbers at the right placeholders, a new GraphViz specification results. It is forwarded to the GraphViz tool and transformed into a *latency breakdown chart* (Figure 5). To illustrate the meaning of this

[1] `http://www.eclipse.org/downloads/packages/eclipse-ide-java-and-dsl-developers/junosr2`

chart, we show that the latency of a sequential process equals the sum of its sub-processes' latencies, e.g. for "image processing", the latency is (rounded off): $265 = 46 + 93 + 126$. Additionally, the utilization is the quotient of the busy time of a resource and the total elapsed time. Take for instance the "GPU" resource, which is only used for "image processing" and for 5 time units per service. Services are each invoked periodically every 400 time units. Therefore, the "GPU" has a utilization of $(5 + 5)/400 = 0.025$.

4.3 Analysis

Using the presented tool chain, iDSL offers the possibility to compare several design alternatives from various perspectives, in an automated manner. We proceed with discussing the results iDSL can generate. First, we discuss the results based on MODES simulations. After that, we review results obtained from model checking using MCTAU and UPPAAL.

Simulation results. We have defined a study with seven design alternatives for which iDSL automatically generates a latency breakdown chart and a latency bar graph. We present the ones for the offset=0 case (Figures 5 and 6). As can be seen in Figure 6, the latency varies highly. This is due to a high degree of concurrency, which forces the scheduler to make many concurrency resolving decisions that each increase the variability. We further see that the "noise reduction" and "contrast" processes contribute most to the latency, which stems directly from their large loads. Additionally, we have included a CDF with the latency times of the design alternatives altogether (Figure 7). It shows that when the offset is small and the level of concurrency larger, latency times become higher. For the highest offsets, no concurrency takes place.

Model checking results. We applied MCTAU on the case with one image processing system. The computation of the lower (Table 6, bottom) and upper bound leads to values 159 and 189, respectively. The difference of 30 between them is caused by the uniform distribution that is specified in the "noise reduction" process (Figure 8). As required by definition, all simulation outcomes fall within the absolute bounds.

5 Conclusion and future work

In this paper we presented iDSL, a domain specific language and toolbox for the performance evaluation of Medical Imaging Systems. iDSL automates performance analysis, for both model checking and simulations, and displays results visually. We have demonstrated the feasibility of our approach using a small example based on a real system, in which we investigated MIS with two concurrent image processing applications.

iDSL has successfully returned differentiated delay, utilization and bound values for a number of designs. In order to assess the scalability of iDSL, we will apply it on extensive cases of our industrial partner Philips, in the Allegio project[2]. This will put the expressiveness of the iDSL language to the proof and may lead to extensions to both the language and toolbox.

We are currently investigating whether we can add a transformation for probabilistic model checking. To support analysis further, we will extend iDSL to create graphs and diagrams that display information of multiple scenarios, services and simulation runs, and include GANTT charts.

[2] `http://redesign.esi.nl/research/applied-research/current-projects/allegio/`

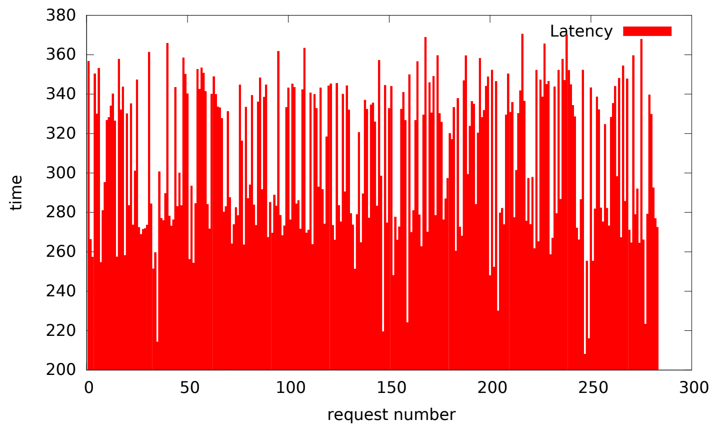


Figure 6 The MODES latency times bar graph (offset=0) for 280 service requests, which is automatically generated from the iDSL code.

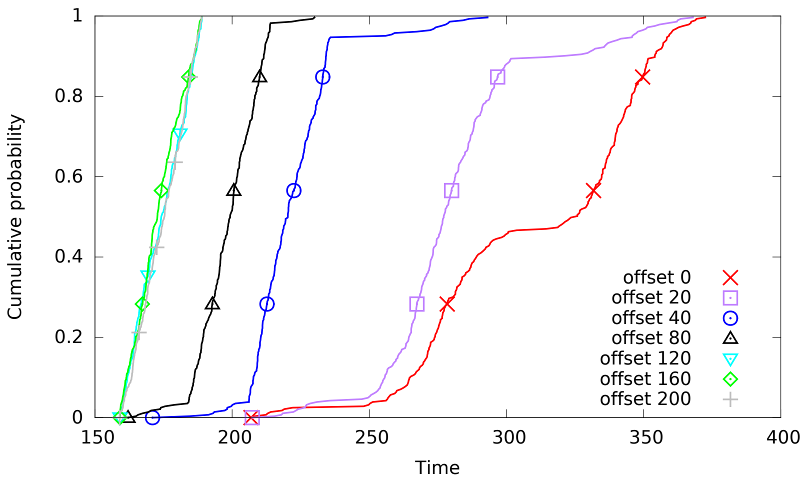


Figure 7 The cumulative distribution functions of latencies for seven design instances is automatically generated from the iDSL code.

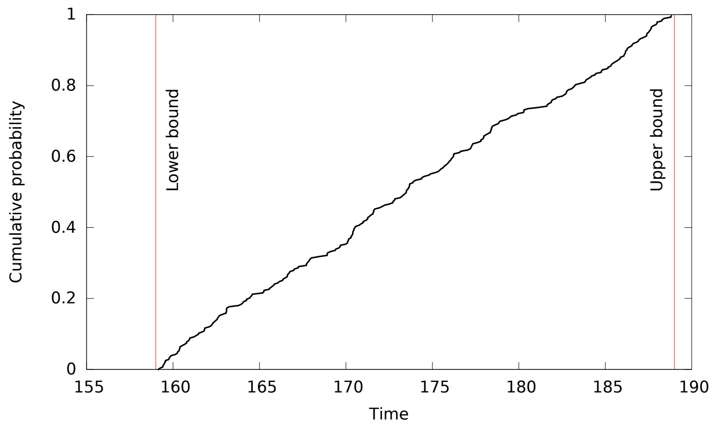


Figure 8 The absolute minimum and maximum bounds, and a CDF of the simulation outcomes is automatically generated from the iDSL code.

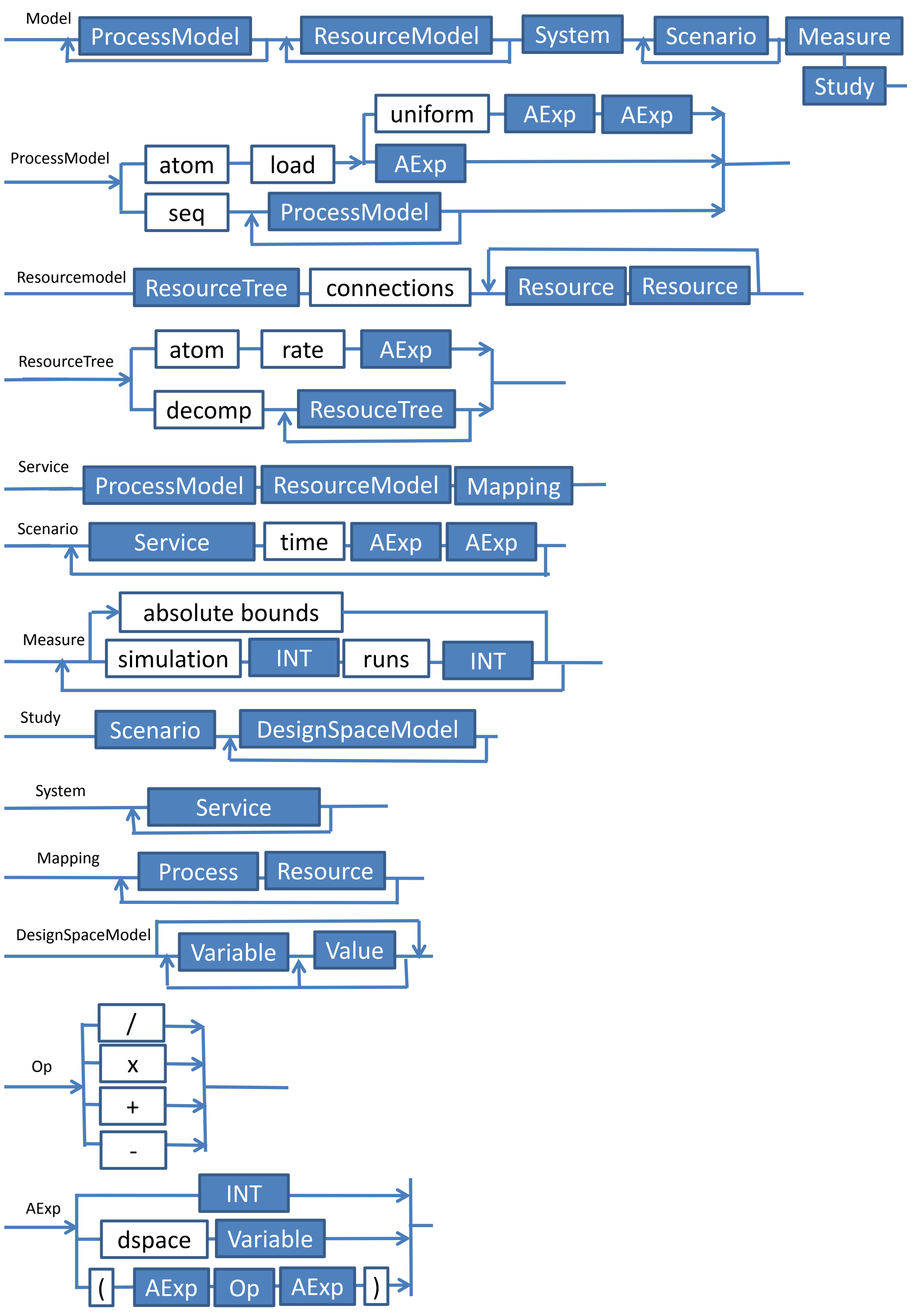


Figure 9 The grammar of iDSL's language as used in this paper. The grammar has the *Model* concept as its top-level node. It decomposes into of one or more *ProcessModels*, *ResourceModels*, a *System* (a set of *Services*), *Scenarios*, and a *Measure* and a *Study*.

Acknowledgements. We would like to thank Arnd Hartmanns of the MoDeST development team for his help and efforts made during the development of iDSL.

We would like to thank Arjan Mooij of the Allegio project for informing us about the binary search algorithm.

References

1 H. Alemzadeh, R. Iyer, Z. Kalbarczyk, and J. Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security & Privacy*, 11(4):14–26, 2013.

2 T. Basten, E. Van Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. De Smet, L. Somers, and E. Teeselink. Model-driven design-space exploration for embedded systems: the Octopus toolset. In *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6415 of *LCNS*, pages 90–105. Springer, 2010.

3 H. Beilner, J. Mater, and N. Weissenberg. Towards a performance modelling environment: News on HIT. In *Modeling Techniques and Tools for Computer Performance Evaluation*, pages 57–75. Plenum Press, 1989.

4 J. Bogdoll, A. David, A. Hartmanns, and H. Hermanns. MCTAU: Bridging the gap between modest and UPPAAL. In *Proc. 19th International SPIN Workshop on Model Checking of Software*, volume 7385 of *LNCS*, pages 227–233. Springer, 2012.

5 Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. Quantitative verification of implantable cardiac pacemakers. In *Proc. 33rd Real-Time Systems Symposium*, pages 263–272. IEEE, 2012.

6 Eindhoven University of Technology. Software/Hardware Engineering - Parallel Object-Oriented Specification Language (POOSL). http://www.es.ele.tue.nl/poosl/.

7 J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing*, volume 2265 of *LNCS*, pages 483–484. Springer, 2002.

8 E. Hahn, A. Hartmanns, H. Hermanns, and J.-P. Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2012.

9 A. Hartmanns. Model-checking and simulation for stochastic timed systems. In *Proc. 9th International Symposium on Formal Methods for Components and Objects*, volume 6957 of *LCNS*, pages 372–391. Springer, 2010.

10 S. Haveman, G. Bonnema, and F. van den Berg. Early insight in systems design through modeling and simulation. In *Proc. 12th Annual Conference on Systems Engineering Research*, 2014. To appear.

11 S. Hettinga. Performance analysis for embedded software design. *Master's thesis, University of Twente*, 2010.

12 G. Igna, V. Kannan, Y. Yang, T. Basten, M. Geilen, F. Vaandrager, M. Voorhoeve, S. de Smet, and L. Somers. Formal modeling and scheduling of datapaths of digital document printers. In *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *LCNS*, pages 170–187. Springer, 2008.

13 G. Igna and F. Vaandrager. Verification of printer datapaths using timed automata. In *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *LCNS*, pages 412–423. Springer, 2010.

14 Z. Jiang, M. Pajic, and R. Mangharam. Cyber-physical modeling of implantable cardiac medical devices. *Proc. of the IEEE*, 100(1):122–137, 2012.

15 B. Kienhuis, E. Deprettere, P. van der Wolf, and K. Vissers. A methodology to design programmable embedded systems. In *Embedded processor design challenges*, volume 2268 of *LCNS*, pages 18–37. Springer, 2002.

16 M. Kwiatkowska, G. Norman, and D. Parker. Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 15(11):1427–1434, 2007.

17 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: verification of probabilistic real-time systems. In *Computer Aided Verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

18 K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.

19 M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *Proc. 18th Real-Time and Embedded Technology and Applications Symposium*, pages 173–184. IEEE, 2012.

20 J. Racine. GNUplot 4.0: a portable interactive plotting utility. *Journal of Applied Econometrics*, 21(1):133–141, 2006.

21 R. Sadre, A. Remke, S. Hettinga, and B.R. Haverkort. Simulative and analytical evaluation for asd-based embedded software. In *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, volume 7201 of *LCNS*, pages 166–181. Springer, 2012.

22 F. van den Berg, A. Remke, A. Mooij, and B.R. Haverkort. Performance evaluation for collision prevention based on a domain specific language. In *Computer Performance Engineering*, volume 8168 of *LCNS*, pages 276–287. Springer, 2013.

A Safety Argument Strategy for PCA Closed-Loop Systems: A Preliminary Proposal[*]

Lu Feng[†], Andrew L. King, Sanjian Chen, Anaheed Ayoub[‡], Junkil Park, Nicola Bezzo, Oleg Sokolsky, and Insup Lee

Department of Computer & Information Science, University of Pennsylvania

Abstract

The emerging network-enabled medical devices impose new challenges for the safety assurance of medical cyber-physical systems (MCPS). In this paper, we present a case study of building a high-level safety argument for a patient-controlled analgesia (PCA) closed-loop system, with the purpose of exploring potential methodologies for assuring the safety of MCPS.

1998 ACM Subject Classification D.2.9 Management (Software quality assurance), J.3 Life and Medical Sciences, K.4.1 Public Policy Issues (Computer-related health issues, Human safety)

Keywords and phrases Medical Cyber-Physical Systems, Safety Argument, Assurance Cases, Patient-Controlled Analgesia Infusion Pump, Closed-Loop Systems

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.94

1 Introduction

Medical devices are increasingly used to deliver critical therapies. Because many devices are used to control the release of chemicals or energy into the patient, the safety of such devices are very important. In the United States, the Food and Drug Administration (FDA) must approve each medical device before it can be marketed. The purpose of this approval process is to ensure that each device meets an acceptable level of safety. The approval process presents challenges to all parties involved. If a company fails to obtain approval for a new device they will not be able to market it and will not be able to make a return on their investment. For the FDA considerable resources are devoted to analyzing submissions and determining if approval should be granted. Therefore, there is a need to effectively communicate and review the safety of medical device systems with a range of stakeholders (*e.g.,* medical device manufacturers and regulatory authorities). The assurance case, which is a method for expressing an argument about some properties of the system is a good way to justify the safety of medical device systems. In fact, the FDA issued a draft guidance [11] in 2010 suggesting that medical manufacturers of infusion pumps provide a safety assurance case with their pre-market submissions.

There are many challenges for both manufacturers and reviewers (*i.e.,* regulatory bodies) when it comes to effective application of the assurance case approach: for example, how can one ensure that the argument presented by an assurance case is valid (*e.g.,* logically consistent)? How can one justify the confidence of evidence used? How can one evaluate the sufficiency of an assurance case? Recently, research into assurance cases for medical

[*] This work is supported in part by NIH grant 1U01EB012470-01 and NSF grants CNS-1035715, IIS-1231547.
[†] Lu Feng is supported by James S. McDonnell Foundation 21st Century Science Initiative – Postdoctoral Program in Complexity Science/Complex Systems – Fellowship Award.
[‡] Anaheed Ayoub is currently employed by Mathworks.

devices has been increasing. For instance, Weinstock and Goodenough [12] discussed the safety case construction of generic infusion pumps; Jee *et al.* [4] constructed a safety case for a pacemaker; Ayoub *et al.* [2] proposed a safety pattern for model-based development, and applied it to a case study of generic Patient-Controlled Analgesic (PCA) infusion pump software.

Recent technological advancements impose additional challenges for assuring the safety of medical device systems. There is an emerging trend of network-enabled medical devices which can communicate and coordinate with each other during the treatment, forming medical cyber-physical systems (MCPS). New functionalities such as closed-loop continuous care, which was not possible with stand-alone devices, are now being developed. However, MCPS also bring new hazards (*e.g.,* network failure) to patient safety, adding more concerns for the safety argument in assurance cases.

In this paper, we consider a patient-controlled analgesia (PCA) closed-loop system, which is an example of MCPS, and build a high-level safety argument for it. The purpose of this case study is to explore potential methodologies for assuring the safety of MCPS. For the rest of the paper, we introduce the background of PCA closed-loop system in Section 2, present our safety argument in Section 3, and draw conclusions in Section 4.

2 Background: PCA Closed-Loop System

PCA infusion pumps are commonly used to deliver pain medication to patients who are experiencing high levels of pain due to serious physical trauma (*e.g.,* surgery). Patients often have different tolerance levels for pain and different reactions to the medication. Therefore, in addition to delivering opioids with a fixed schedule programmed by a caregiver, the PCA pump also allows the patient to request an additional dose of medication (called *bolus*) by pressing a button. A well-known hazard with opioid medication is that an overdose can cause respiratory failure, which may be fatal to patients [8]. There are some safety mechanisms built into modern PCA pumps. For example, a PCA pump can be programmed with limits on the number of doses it will deliver, which helps to avoid overdose no matter how often the patient pushes the bolus button. However, the existing safety mechanisms are not sufficient to protect patients in all clinical scenarios and a large number of adverse events involving PCA pumps have been reported [9]. The causes of patients receiving overdose include, but are not limited to, the following:

- the pump is misprogrammed,
- the wrong concentration of drug is loaded into the pump,
- a caregiver overestimates the maximum dose the patient can receive,
- PCA-by-proxy, *i.e.,* someone other than the patient presses the bolus button.

Obviously, there is still certain risk associated with the use of PCA pumps, to which we refer as the *residual* risk of standalone pumps.

To mitigate the overdose hazard, clinicians must monitor the patient's respiratory function through vital sign sensor readings (*e.g.,* blood oxygen saturation measured by a pulse oximeter). Then, if the patient entered respiratory distress, the caregiver would manually intervene to resuscitate the patient. Unfortunately the current practice is both error prone and burdensome for the clinician [3, 5].

Recently, the notion of a "closed-loop" PCA system has been proposed to ease the burden of clinicians by interconnecting the infusion pump, pulse oximeter, and a computer controller over a network. The controller would monitor the pulse oximeter readings and, when a problem is detected, automatically stop the infusion pump and alert the clinician.

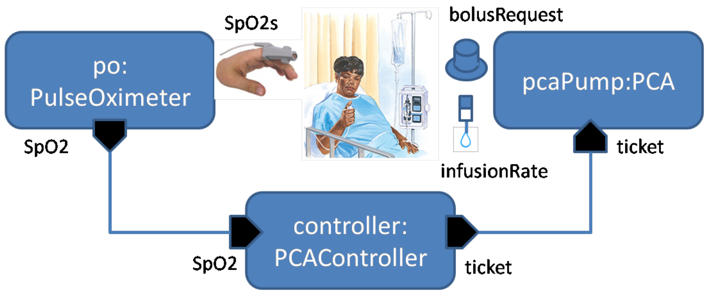


Figure 1 PCA closed-loop system overview (adapted from [7]).

Figure 1 shows the architecture and essential data flow of a PCA closed-loop system. A pulse oximeter receives physiological signals from a clip on the patient's finger and calculates the SpO_2 values (*i.e.,* the measure of blood oxygenation). The computer controller makes control decisions based on SpO_2 readings received from the pulse oximeter, and periodically issues a "ticket" to the infusion pump. Each ticket limits the bolus and basal time period that the pump can infuse before the patient could possibly be pushed into respiratory distress. If the network becomes disconnected for a long period, the pump would expire the current ticket and stop delivering pain medication to protect the patient from overdose. Unless the ticket expires or the pump is stopped by the controller, the infusion pump will continue to deliver opioids to the patient at the basal rate programmed by the caregiver. The patient may also occasionally press the button and request a bolus from the infusion pump. After the absorption of the opioid medication, the patient's respiratory state may become more depressed, which is reflected by the patient's blood oxygenation level. The safety of such a closed-loop system has been studied in [1, 10] via simulation-based analysis and formal verification.

3 Safety Argument

In this section, we develop a high-level safety argument for the PCA closed-loop system. Figure 2 shows our argument using the Goal Structuring Notation (GSN), a popular graphical notation for organizing and presenting safety argument (we refer readers who are unfamiliar with GSN to [6]).

The top-level *goal* (**G1**) is to show that "*The PCA closed-loop system is at least as safe as the stand-alone infusion pump, with respect to the overdose hazard*". Here, we assume that the closed-loop system is built on top of a stand-alone infusion pump whose safety has already been assessed in a separate safety argument, and the pulse oximeter's behavior is not affected by putting in the PCA closed-loop. This context is documented as **C1.1** in Figure 2.

To address **G1**, our strategy is to argue by risk-benefit analysis (**S1**), which is defined in the context **C1.2**. If the benefit brought by the closed-loop system outweighs its introduced risk, then we can assert that the goal **G1** is true. More specifically, the *benefit* refers to how much residual risk of the stand-alone pump can be mitigated by the closed-loop system.

Following strategy **S1**, we decompose **G1** into three sub-goals:

- **G2.1**: *The introduced risk due to hazards of closed-loop system is acceptable.*
- **G2.2**: *Some residual risk of the stand-alone infusion pump is adequately mitigated by the closed-loop system.*
- **G2.3**: *The benefit of closed-loop system outweighs its introduced risk.*

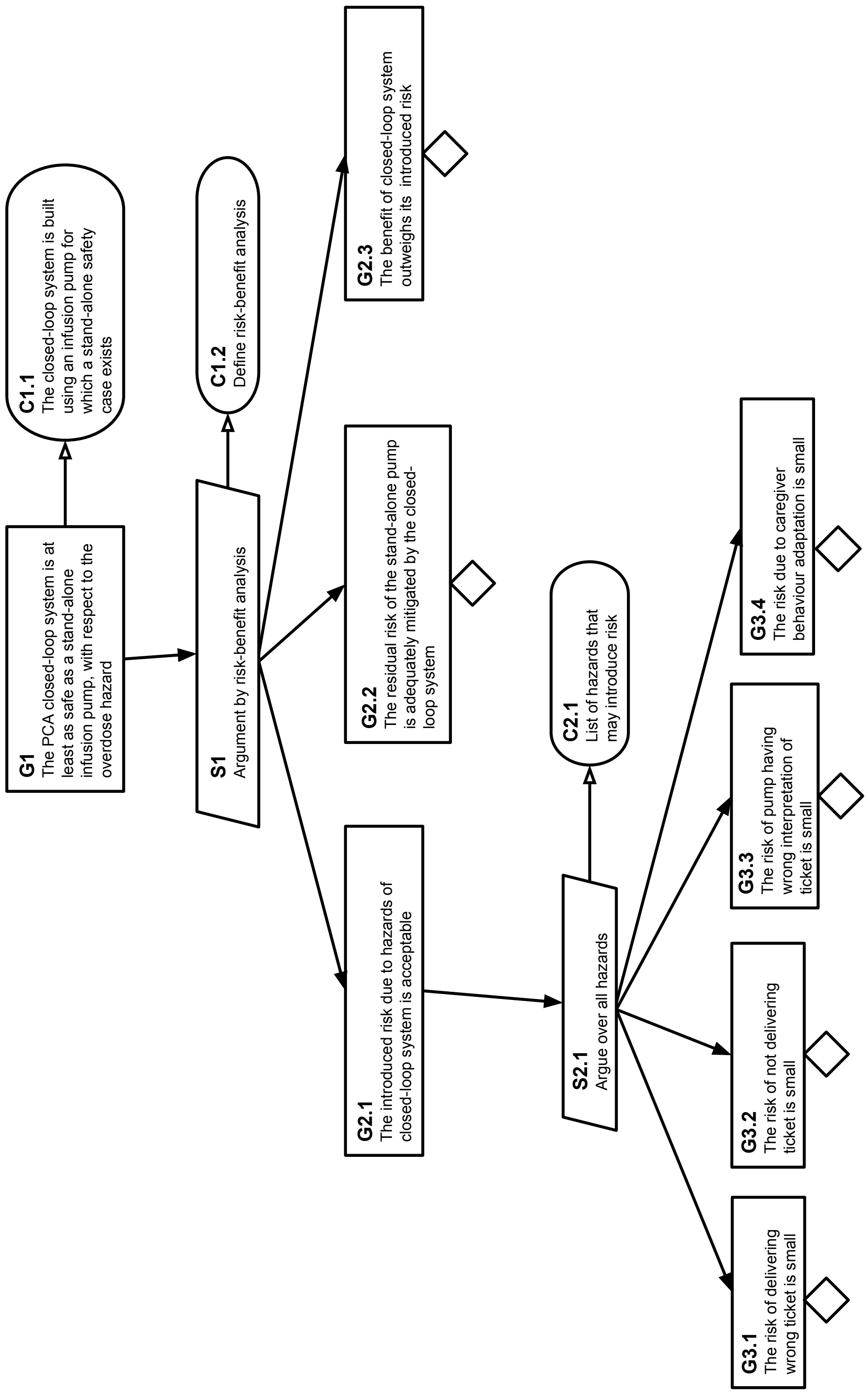

G1
The PCA closed-loop system is at least as safe as a stand-alone infusion pump, with respect to the overdose hazard
C1.1
The closed-loop system is built using an infusion pump for which a stand-alone safety case exists
S1
Argument by risk-benefit analysis
C1.2
Define risk-benefit analysis
G2.1
The introduced risk due to hazards of closed-loop system is acceptable
G2.2
The residual risk of the stand-alone pump is adequately mitigated by the closed-loop system
G2.3
The benefit of closed-loop system outweighs its introduced risk
S2.1
Argue over all hazards
C2.1
List of hazards that may introduce risk
G3.1
The risk of delivering wrong ticket is small
G3.2
The risk of not delivering ticket is small
G3.3
The risk of pump having wrong interpretation of ticket is small
G3.4
The risk due to caregiver behaviour adaptation is small

Figure 2 High-level safety argument for the PCA closed-loop system.

In Figure 2, we only further develop **G2.1** as an example, while keep **G2.2** and **G2.3** undeveloped (denoted by a *diamond* underneath the rectangle element). In the following, we elaborate on **G2.1** in more details and propose possible strategies for **G2.2** and **G2.3**.

The strategy (**S2.1**) for claiming goal **G2.1** is to argue over a list of possible hazards introduced by the closed-loop system, under the context (**C2.1**) that lists introduced hazards. This strategy leads to four sub-goals, each of which corresponds to a hazard of the closed-loop system. In Figure 2, these four goals (**G3.1-G3.4**) are not further developed. We briefly discuss their corresponding hazards as follows.

- (**G3.1**) *Delivering a wrong ticket to the infusion pump.* This hazard may be caused by incorrect controller computation, corruption of the message on the network, or incorrect sensor readings. We may argue that the risk of this hazard is small by providing formal verification evidence for the correctness of the controller algorithm. Another useful evidence is the verification of the infusion pump. If the pump correctly handles tickets arriving from the network interface, tickets cannot make the pump infuse when it would not be infusing in the stand-alone case, or infuse at a different rate. That is, at any time, the pump would be infusing at the same rate as it would be infusing in the stand-alone case, unless it has been stopped by an expired ticket. Therefore, a bad ticket would not cause more overdose than in the stand-alone case, if the pump handles the ticket correctly.
- (**G3.2**) *Not delivering a ticket to the pump.* Various reasons may cause this hazard. For example, the controller does not produce a ticket when it should, due to an incorrect implementation or incorrect sensor reading; or the calculated ticket is lost, due to disconnected network or other failures. In any case, the infusion will continue unmodified until the prescription runs out or the current ticket expires. Thus, this hazard would not introduce additional risk because the patient receives exactly the same amount of medication as in the stand-alone case.
- (**G3.3**) *The pump has a wrong interpretation of the ticket.* Recall that a ticket contains the maximum time period over which the infusion pump can infuse, a ticket that does not expire when it should due to the pump's wrong interpretation may lead to overdose. Similar to the argument for **G3.1**, we can provide the formal verification of the pump as evidence to show that the risk of hazard is small.
- (**G3.4**) *Caregiver behavior adaptation.* For example, due to the automation of closed-loop system, the caregiver may check the pump alarm state and assess the patient condition less frequently than in the stand-alone case. Or, the caregiver learns to assume that the system will self-correct and therefore applies more aggressive therapy. The argument about this hazard relies on the caregiver's training. Training materials and guidelines will be used as evidence. In additional, a sufficiently reliable new alarm system must be present to detect closed-loop system failure and notify caregivers.

We can argue goal **G2.2** in a similar way as for **G2.1**, that is, arguing over residual risk of the stand-alone pump that can be mitigated by the closed-loop system. As described in Section 2, the residual risks include, for example, the pump being misprogrammed, the wrong concentration of drug being loaded into the pump, a caregiver overestimating the maximum dose the patient can receive, or someone other than the patient pressing the bolus button. These hazards can be adequately mitigated in the closed-loop system due to the fact that, the controller would automatically monitor the patient's respiratory function via pulse oximeter readings and automatically stop the infusion pump whenever necessary to protect the patient from overdose.

Finally, goal **G2.3** takes a holistic view of benefit and risk of the closed-loop system. Essentially, we want to show that the benefit of the closed-loop system (*i.e.,* mitigating residual risk of the stand-alone pump) outweighs its introduced risk. A formal risk-benefit analysis report can be used as evidence to support this goal.

4 Conclusions

We have presented a high-level safety argument for a patient-controlled analgesia (PCA) closed-loop system, where an infusion pump, a pulse oximeter, and a computer controller are interconnecting over a network. The goal of the argument is to show that "*The PCA closed-loop system is at least as safe as the stand-alone infusion pump, with respect to the overdose hazard*", and the strategy is to argue by risk-benefit analysis. This case study has the potential of being generalized for other network-enabled medical devices. We hope to further explore this direction in the future. Ultimately, we would like to develop a safety argument pattern for closed-loop systems.

References

1 David Arney, Miroslav Pajic, Julian M Goldman, Insup Lee, Rahul Mangharam, and Oleg Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE Int'l Conf. on Cyber-Physical Systems*, pages 139–148. ACM, 2010.

2 A. Ayoub, B. Kim, I. Lee, and O. Sokolsky. A Safety Case Pattern for Model-Based Development Approach. In *NFM2012*, pages 223–243, Virginia, USA, 2012.

3 Rodney W. Hicks, Vanja Sikirica, Winnie Nelson, Jeff R. Schein, and Diane D. Cousins. Medication errors involving patient-controlled analgesia. *American Journal of Health-System Pharmacy*, 65(5):429–440, March 2008.

4 E. Jee, I. Lee, and O. Sokolsky. Assurance cases in model-driven development of the pacemaker software. In *4th International Conference on Leveraging Applications of Formal Methods, Verification, and Validation – Volume Part II*, ISoLA'10, pages 343–356, Berlin, Heidelberg, 2010. Springer-Verlag.

5 Joint Commission. Sentinel event alert issue 33: Patient controlled analgesia by proxy. `http://www.jointcommission.org/sentinelevents/sentineleventalert/`, December 2004.

6 Tim Kelly and Rob Weaver. The goal structuring notation – a safety argument notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004. `http://www-users.cs.york.ac.uk/tpk/dsn2004.pdf`.

7 Andrew L. King, Lu Feng, Oleg Sokolsky, and Insup Lee. Assuring the safety of on-demand medical cyber-physical systems. In *Proceedings of the 1st IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA'13)*, 2013.

8 P. E. Macintyre. Safety and efficacy of patient-controlled analgesia. *British Journal of Anaesthesia*, 87(1):36–46, 2001.

9 Teryl K. Nuckols, Anthony G. Bower, Susan M. Paddock, Lee H. Hilborne, Peggy Wallace, Jeffrey M. Rothschild, Anne Griffin, Rollin J. Fairbanks, Beverly Carlson, Robert J. Panzer, and Robert H. Brook. Programmable infusion pumps in icus: An analysis of corresponding adverse drug events. *Journal of General Internal Medicine*, 23(1):41–45, 2008.

10 Miroslav Pajic, Rahul Mangharam, Oleg Sokolsky, David Arney, Julian M. Goldman, and Insup Lee. Model-driven safety analysis of closed-loop medical systems. *IEEE Transactions on Industrial Informatics*, 2013.

11 U.S. Food and Drug Administration, Center for Devices and Radiological Health. *Guidance for Industry and FDA Staff – Total Product Life Cycle: Infusion Pump – Premarket Notification [510(k)] Submissions*, April 2010.

12 C. Weinstock and J. Goodenough. Towards an Assurance Case Practice for Medical Device. Technical report, CMU/SEI-2009-TN-018, 2009.

Evaluating On-line Model Checking in UPPAAL-SMC using a Laser Tracheotomy Case Study

Xintao Ma[1], Jonas Rinast[1], Sibylle Schupp[1], and Dieter Gollmann[2]

1 **Institute of Software Systems, Hamburg University of Technology**
21073 Hamburg, Germany
`{xintao.ma,jonas.rinast,schupp}@tuhh.de`

2 **Security in Distributed Systems, Hamburg University of Technology**
21073 Hamburg, Germany
`diego@tuhh.de`

Abstract

On-line model checking is a variant of model checking that evaluates properties of a system concurrently while deployed, which allows overcoming limitations of inaccurate system models. In this paper we conduct a laser tracheotomy case study to evaluate the feasibility of using the statistical model checker UPPAAL-SMC for on-line model checking in a medical application. Development of automatic on-line model checking relies on the precision of the prediction and real-time capabilities as real-time requirements must be met. We evaluate the case study with regards to these qualities and our results show that using UPPAAL-SMC in an on-line model checking context is practical: relative prediction errors were only 2% on average and guarantees could be established within reasonable time during our experiments.

1998 ACM Subject Classification D.2.4 Software Engineering – Software/Program Verification – Model Checking, F.1.1 Computation by Abstract Devices – Models of Computations – Automata, F.1.2 Computation by Abstract Devices – Modes of Computations – Online Computation, I.6.3 Simulation and Modeling – Applications, I.6.7 Simulation and Modeling – Simulation Support Systems, J.3 Life and Medical Sciences – Health

Keywords and phrases On-line Model Checking, Laser Tracheotomy, UPPAAL-SMC, Patient-in-the-loop

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.100

1 Introduction

In the medical domain not only the devices must operate reliably but also the safety of connected patients must be guaranteed. This requirement becomes more and more pressing with the increased development of patient-in-the-loop systems that monitor and treat patients autonomously and where a malfunction could seriously harm the patient. Model checking, a widely known technique to show that a system fulfills certain properties, might be an option to ensure safe operation of such systems, but is often not adequate in the medical context. Classic model checking relies on models that accurately predict the system state also in a distant future for all system components for reasoning about the system. When human physiology is involved such models are unavailable most of the time. For example, predicting the long-term behavior of the blood oxygen concentration of a human patient currently is infeasible since the present understanding of the processes within the human body only permit short-time predictions.

On-line model checking relaxes the need for accurate long-term models required by classic model checking. Instead of statically proving a property of the system on-line model checking yields guarantees that are only valid for a limited time using bounded model checking. To still provide a safety guarantee at all times the on-line model checking approach repeatedly evaluates the property to extend its period of validity indefinitely. This iterative process allows on-line model checking to dynamically adapt the underlying system model. Thus, the on-line model checking approach can employ measures to adjust the model parameters such that they match runtime observations if the model accuracy decreases significantly, and thus reestablish a consistent state. Accurate long-term models are no longer required as the dynamic adaptation of the short-term models to the current real-world situation may still yield the desired long-term guarantees.

In this paper we carry out an on-line model checking case study using the model checker UPPAAL and evaluate its statistical model checking module UPPAAL-SMC regarding its suitability for on-line model checking. UPPAAL is one of few available tools that have the potential to carry out automatic on-line model checking. Evaluating whether its performance in practice meets the real-time requirements imposed by on-line model checking is crucial to tapping its full potential. Therefore we encode the models of a previous on-line model checking case study that models a laser tracheotomy surgery with hybrid models for the model checker PHAVer using the timed automata formalization used by UPPAAL. This relation enables us to compare our results to the previous work and lets us focus on questions regarding UPPAAL-SMC's performance and suitability. We then carry out the on-line model checking process with our derived models using a prototype for automatic on-line model checking with UPPAAL. Next, the collected data on the accuracy of parameter prediction and the run-time performance is compared to the results of the original case study. Relative errors of SpO_2 estimations were on average about 2% which is slightly worse than the original case study. For performance, a verification step took on average about 50ms which is a significant improvement. As a general result it follows that it is practical to use UPPAAL in an on-line model checking context.

The rest of the paper is organized as follows: Section 2 introduces related work. Section 3 provides a short introduction to hybrid automata and their relation to UPPAAL-SMC. The on-line model checking approach and its implementation with UPPAAL is the topic of Section 4. Section 5 shows the analyzed case study and its on-line models. Section 6 provides our experiment results and an evaluation of those. At last, Section 7 summarizes the paper and gives ideas on future research.

2 Related Work

In general on-line model checking can be put into the context of self-adaptive software. More specifically, verification at runtime enables ways to produce self-adapting software systems [6]. Zhao et al. introduce the on-line model checking approach as a lightweight verification technique to reduce the state space explosion problem [18]. They argue that on-line model checking is significantly different from the runtime verification approach: in contrast to on-line model checking runtime verification operates directly on the execution trace without involving a system model. Thus, the approach is not capable of predicting property violations. Steering is a control-theoretic approach trying to resolve this drawback [10]. Li et al. apply on-line model checking to a laser tracheotomy surgery scenario to ensure the patient safety [14, 15]. They use the hybrid model checker PHAVer in combination with a custom implementation to carry out the model checking procedure. This is the reference case study we compare our UPPAAL implementation to. Bartocci et al. [2] and Chen et al. [8] deal with the model repair problem, a related approach that tries to adjust model parameters to satisfy system properties in case they are violated. However, here the adjustment goal for on-line model checking is not to satisfy a property but to ensure that the model does not deviate from the observed real-world state.

Regarding the implementation of on-line model checking, Bu et al. pursue the development of an on-line model checking tool set called BACH$_{\mathrm{OL}}$, which is based on the linear hybrid automaton model checker BACH, to facilitate verification of complex cyber-physical systems [4]. Furthermore, in earlier work we began implementing a framework for on-line model checking with UPPAAL [17]. This framework automatically performs the necessary state reconstruction for seamless model simulation and verification.

In the context of closed-loop medical systems, King et al. report on their experience with their Medical Device Coordination Framework (MDCF) when modeling a closed-loop medical system to control an infusion pump [13]. Their research focuses on the interoperability between medical devices. Arney et al. also analyze a patient-in-the-loop system [1]: they develop UPPAAL and MATLAB models to show in advance potential flaws in the control loop that endanger the patient. The development and verification of formal models for pacemaker systems is the topic of work by Chen at al. [7] and Jiang et al. [12].

3 Timed and Hybrid Automata

Both the models from the original case study our work is based on and our models use variations of finite state machines to represent the system. The original case study derives a *hybrid automata* model. In UPPAAL, however, modeling of hybrid automata is only possible with the statistical model checking extension UPPAAL-SMC as UPPAAL normally uses networks of *timed automata* as the underlying modeling formalism, a subset of hybrid automata.

A hybrid automaton, according to Henzinger, is a finite state automaton extended with a set of continuous variables [11]. Thus, a hybrid automaton may model discrete and continuous behavior. Such a hybrid automaton consists of the following parts:

- **Graph.** A finite directed multigraph (V, E) that models the topology of the discrete transitions with the locations V and the edges E.
- **Variables.** A finite set of variables $X = \{ x_1, \ldots, x_n \}$ valued in the reals ($\mathbb{R}$) together with its set of derivatives $\dot{X} = \{ \dot{x_1}, \ldots, \dot{x_n} \}$.
- **Condition Predicates.** Three labeling functions that assign predicates to locations $l \in V$: *init* assigns initial valuations, *inv* assigns an invariant condition, and *flow* assigns a flow condition that determines how variables evolve over time in a location. These are generally linear differential equations.
- **Guards.** A labeling function *guard* that assigns predicates to edges $e \in E$ that specify when a transition over an edge may be triggered.
- **Actions.** A labeling function *action* that assigns actions $a \in \Sigma$ to edges $e \in E$ that are performed when a transition over the edge is triggered.

In UPPAAL-SMC a hybrid automaton is defined in terms of the underlying timed automata definition such that most of the known UPPAAL features, e.g., synchronization, could be carried over. In UPPAAL, a timed automaton is defined as follows

▶ **Definition 1.** A *timed automaton* T is a tuple $T = \{ L, l_0, C, A, E, I \}$ where L is a set of locations, $l_0 \in L$ is the initial location, C is a set of real-valued clock variables, A is the action set, E is the set of edges of the form (l, a, g, R, l') where $l, l' \in L$, $a \in A$, g a predicate over C, and R a subset of C, and I is a function that assigns invariant predicates to locations $l \in L$.

In Definition 1 the set R on an edge is its reset set, i.e., the set of clocks that are set to certain values when a transition involving the edge is fired. Note that in UPPAAL a reset of a clock does not necessarily mean the clock is set to zero; any integer value is allowed. Furthermore, UPPAAL restricts predicates over C to conjunctions of terms that bound a clock or a difference of clocks by an integer.

Definition 1 contains nearly all the necessary components for modeling a hybrid automaton. When taking into account the renaming of components only the *flow* function defining the behavior of individual variables in a location can not be specified directly. Therefore, UPPAAL-SMC extends the timed automaton definition with an additional component F, which allows modeling of hybrid automata [9]. The function F corresponds to the *flow* labeling in Henzinger's hybrid automata and is called a *delay function*. It allows modification of the default delay function of UPPAAL-SMC, which advances all clocks synchronously at the same rate, in certain locations l by defining explicit rates for clocks: $x' = e$ where e only depends on the discrete part of the state. It follows that the transformation of hybrid models to UPPAAL-SMC models may be carried out if their *flow* function can be expressed by explicit rates. In this case study we determine the delay function by performing a linear regression (see Section 5 and Section 6).

For more information on UPPAAL, Behrman et al. provide a complete introduction [3]. The statistical model checking module, UPPAAL-SMC, is covered in the publication by Bulychev et al. [5].

4 On-line Model Checking

This section introduces the on-line model checking process in Subsection 4.1 and then provides details on the implementation aspects of it in Subsection 4.2.

4.1 The On-line Model Checking Approach

On-line model checking is a technique to apply classic model checking to domains where accurate modeling of a system may be infeasible. In such cases classical model checking, if based on approximate models, may yield seemingly satisfactory results. But in reality those properties can not be guaranteed because the model does not correctly reflect the system. On-line model checking overcomes the model inaccuracies by periodically adjusting the underlying model to the real-world values observed from the system.

Figure 1 and Figure 2 depict the relation of the state spaces of both approaches. Figure 1 shows the classical model checking approach where a single model of the system is constructed. Here, the model does not correctly model all aspects of the systems. Thus, the state space of the model is only a subset of the state space of the system and an actual trace of the system as shown by the arrow starting with the circle may leave the model state space. As only the state space of the model is checked for compliance with the requirements for the system there are cases where the model checking approach assures a system property but in reality that property may not be satisfied. Thus, when an exact model of a system is not available classical model checking does not yield reliable results.

In contrast, Figure 2 shows the situation when applying on-line model checking. Here, the model is adjusted periodically and a new model is generated based on the current system

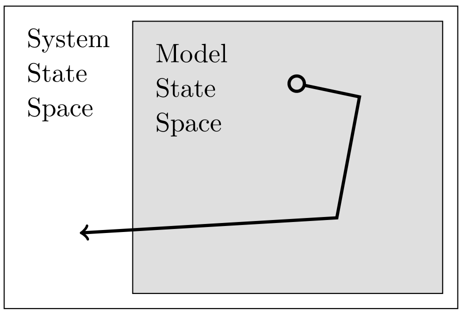


Figure 1 State Space in Classic Model Checking.

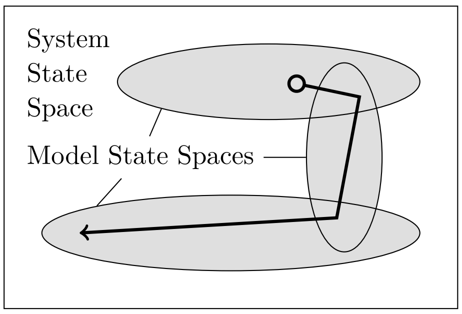


Figure 2 State Space in On-line Model Checking.

trace. The first benefit is that the concrete system state is always valid in the current model. Thus, leaving the model state space is impossible and a guarantee obtained from the model checker is always reliable although the models do not at all times accurately reflect the system. Consequently, the obtained guarantees only have limited periods of validity because of the limited model scope. The limited scope though is responsible for another benefit: the model state spaces in general are smaller and thus the model checking performance becomes better. It follows that with the on-line model checking approach the model checking technique can be applied to domains where models are likely to be inaccurate because model adjustments may overcome any inaccuracies.

▶ **Example 2.** As an example assume a light bulb is supposed to be switched on and off every five seconds. Experience shows that this kind of light bulb malfunctions after 1000 on-off-cycles. It is critical that the light cycle in the system never stops and thus we want to ensure a 30 second grace period to exchange the bulb when approaching the end of its lifetime. If the model is correct a simple calculation yields the time when a change is necessary. Now, assume that the real system does not switch the light on and off every five seconds but the switching delay varies unpredictably. Then verifying every 30 seconds that no malfunction occurs within the next minute would achieve the same 30-second grace period only if the model variables are updated with the current light state and the number of on-off-cycles that actually occurred.

4.2 Implementing On-line Model Checking

The implementation of on-line model checking of a real system can be divided into three phases, one before deployment of the system, and two during deployment:

- **Modeling.** During the modeling phase first the requirements for a system are specified. Then a model of the system is developed that allows reasoning about those requirements. Also, the initial state of the model is defined, i.e., the values with which the real system starts operation.
- **Verification.** In the verification phase a current system model is passed to the model checking engine together with the requirement properties and checked for compliance. In case the verification fails an emergency handling routine may be triggered to resolve the issue. Otherwise a guarantee is obtained that the requirements are fulfilled for a limited time bound T.
- **Adjustment.** In the adjustment phase the real system is observed and the previous model is adjusted accordingly to accurately represent the current and near-future states.

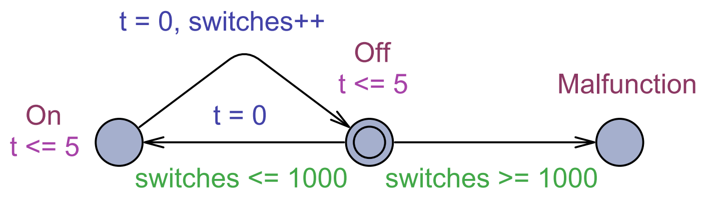


Figure 3 Model Checking Example System.

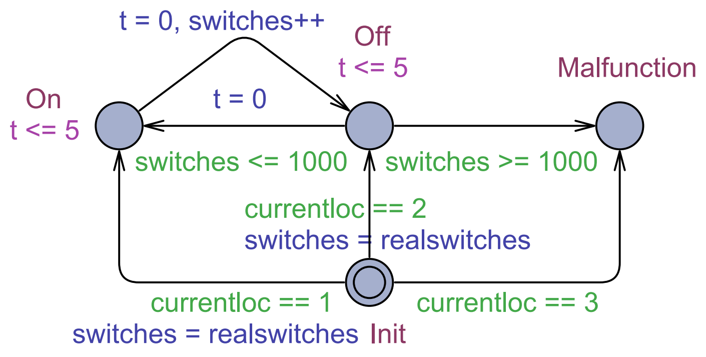


Figure 4 On-line Model Checking Example System.

The adjustment must be performed before the period of validity T runs out to ensure continuity of the guarantees. The newly created model is then forwarded to the next verification phase.

When implementing on-line model checking it is thus necessary that the model provides means for adjusting the model. In this paper we manually modify the basic system models to allow the necessary modifications because automatically providing these means induces a reconstruction problem of the previous system state [17] and solving the reconstruction problem introduces additional machinery, which would dilute the focus on UPPAAL's performance of this case study.

▶ **Example 3.** Recall Example 2. Figure 3 shows the basic UPPAAL model for the example system. The adaptation of the model is rendered possible by introducing two parameters, the performed number of switches and the current state of the lamp. They are passed during the adaptation phase from the real system to the model as the constant values `realswitches` and `currentloc`. The resulting on-line model is depicted in Figure 4.

5 A Medical Case Study

In this section we present an on-line model checking case study on a medical laser tracheotomy scenario and demonstrate the applicability of UPPAAL in such scenarios. In Subsection 5.1 we introduce laser tracheotomy in general and related safety requirements. The concrete UPPAAL system models derived from the case study by Li et al. [14] are the focus of Subsection 5.2.

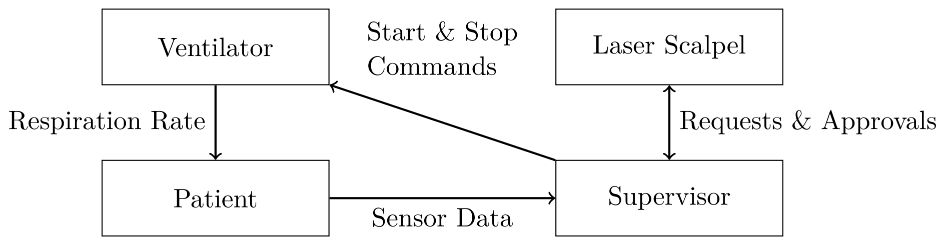


Figure 5 Laser Tracheotomy System [14].

5.1 Laser Tracheotomy

Tracheotomy is a surgery performed on patients that have problems breathing through their nose or mouth, e.g., when the tongue muscle falls back and blocks the air flow while sleeping. During the surgery a direct access to the windpipe of the patient is created, usually from the front side of the neck. Laser tracheotomy refers to the kind of tracheotomy where the access to the windpipe is created using a laser scalpel, a medical device capable of cutting tissue with focused light. Using laser for the cut has several benefits such as a greater precision and a reduction of blood loss due to blood vessels being closed immediately. However, during tracheotomy the laser also poses the threat of tissue burns in case the oxygen concentration in the windpipe of the patient is too high.

In this case study we want to ensure that the laser may only be triggered when an operation is safe. Additionally, as the patient is ventilated during the surgery, we want to ensure that the blood oxygen of the patient does not drop to dangerous levels, because ventilation needs to be suspended during cutting. Lastly, for convenience of the surgeon, an additional requirement is that once the use of the laser is approved the laser should emit for a minimum time such that the cut is not interrupted unnecessarily. The verification properties for the statistical model checking are given in $\text{WMTL}_{\leq}$ (Weighted Metric Temporal Logic) [5]:

- O_2 above threshold while laser emits
  - `Pr[<=100](<> O2 > Th_O2 && LaserScalpel.LaserEmitting)`
- SpO_2 below threshold while laser emits
  - `Pr[<=100](<> SpO2 < Th_SpO2 && LaserScalpel.LaserEmitting)`
- Laser stops emitting early

  -
    ```
    Pr[<=100](<> (O2 > Th_O2 || SpO2 < Th_SpO2) &&
              t_appr < Th_appr && LaserAppr == true)
    ```

These properties characterize unreachable states and thus the probabilities should be zero with the configured confidence.

5.2 System Modeling

The laser tracheotomy scenario described by Li et al. consists of four different components [14]:

- **Patient.** The patient under surgery characterized by current windpipe oxygen level (O_2) and blood oxygen level (SpO_2).
- **Ventilator.** The medical ventilator device regulating the patient's breathing rate during the surgery. The ventilator is characterized by the current height of the pressure cylinder.
- **Laser Scalpel.** The laser scalpel is used to cut the opening to the windpipe. The laser scalpel is characterized by whether or not the surgeon currently wants to operate the laser and if operation is allowed.

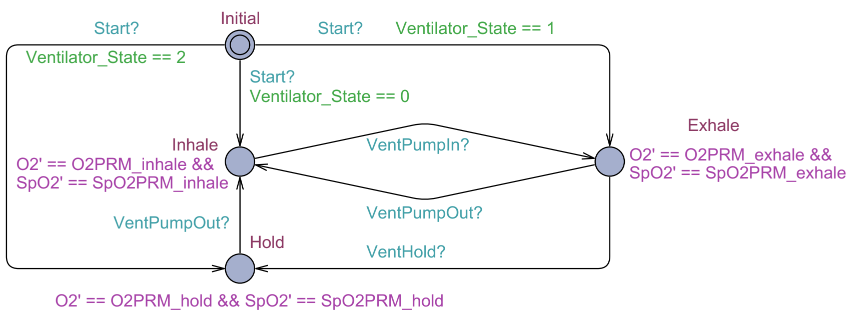


Figure 6 Patient UPPAAL Model.

- **Supervisor.** The supervisor is responsible for ensuring the safety requirements of the system as given in Subsection 5.1. The supervisor approves usage of the laser scalpel and operates the ventilator.

Figure 5 shows the connections of the system components with respective communication data. The ventilator regulates the respiration rate of the patient. The physiological signals of the patient are measured by sensors and forwarded to the supervisor. The supervisor analyzes the values and either approves usage of the laser scalpel requested previously and consequently stops the ventilator, or usage is prohibited and the ventilator continues normal operation. Additionally, when an approved cut is finished the ventilator starts operating again by instruction of the supervisor. We now discuss our UPPAAL models of the components for on-line model checking in more detail. All of the models were derived from the original hybrid models using the encoding from Section 4. The main difficulty in the transformation of the models is representing the continuous variables O_2 and SpO_2 in the hybrid models using clock variables in UPPAAL-SMC and ensuring correct system behavior using synchronization. The remaining parts are straight-forward because the graph components and transition constraints carry over directly due to the same finite state machine formalism.

5.2.1 Patient

The patient model consists of three locations plus the initialization location. The locations correspond to the patient inhaling and exhaling assisted by the ventilator and the patient exhaling without assistance when the ventilator is switched off. In the three locations the O_2 and SpO_2 values are predicted using a linear regression approach taking into account a history of 30 seconds (see Section 6).

5.2.2 Ventilator

The ventilator model also has three main locations, an initialization location and two intermediate locations for communication reasons. The main locations correspond to the ventilator pumping air into the patient, out of the patient, and not pumping at all. Here the current height of the ventilation cylinder, H_vent, is modified accordingly. Communication with the patient model is implemented such that the patient always inhales and exhales as enforced by the ventilator. Furthermore input from the supervisor model is accepted to enable and disable the ventilator.

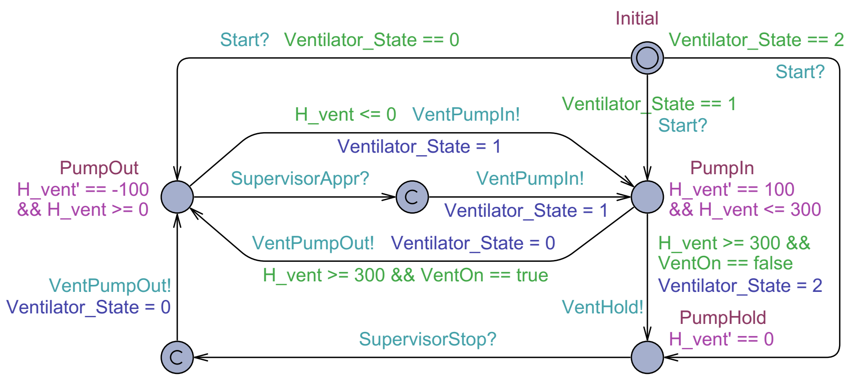


Figure 7 Ventilator UPPAAL Model.

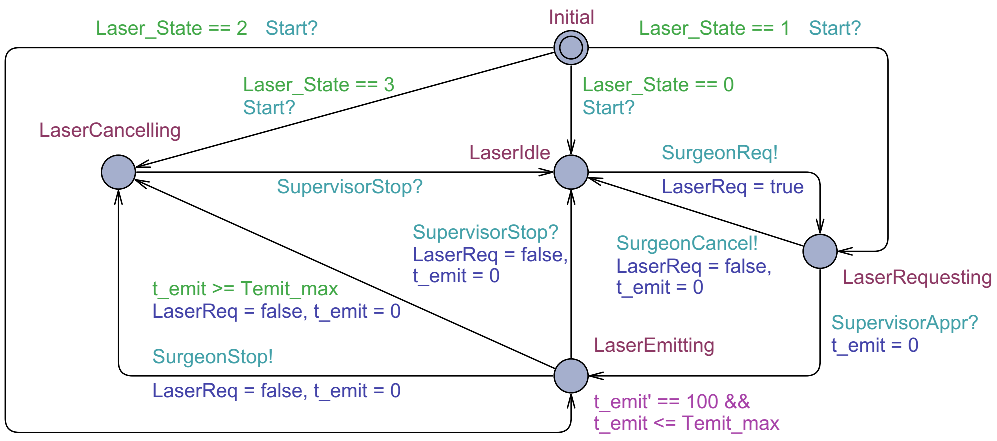


Figure 8 Laser Scalpel UPPAAL Model.

5.2.3 Laser Scalpel

The laser scalpel model represents the interaction between the surgeon and the laser scalpel. It uses four locations. The surgeon may send a request to the supervisor model to trigger the laser, which eventually gets approved. When the laser emits the surgeon can either switch the laser off or revoke the approval if conditions necessitate action. Communication thus takes part between the laser scalpel and the supervisor model. The surgeon inputs are left open meaning that any external input may be executed at any time during verification.

5.2.4 Supervisor

The supervisor model checks if the physiological parameters of the patient are within safe boundaries and approves the laser usage for a maximum duration. If any of the safety requirements gets violated the supervisor revokes its approval. The interesting part here is that the initialization part also checks if a safety requirement was violated. This behavior is necessary because when the model is adapted the O_2 and SpO_2 values may change, which might invalidate a previous approval. Also, when the supervisor approves usage of the laser the ventilator is put on hold.

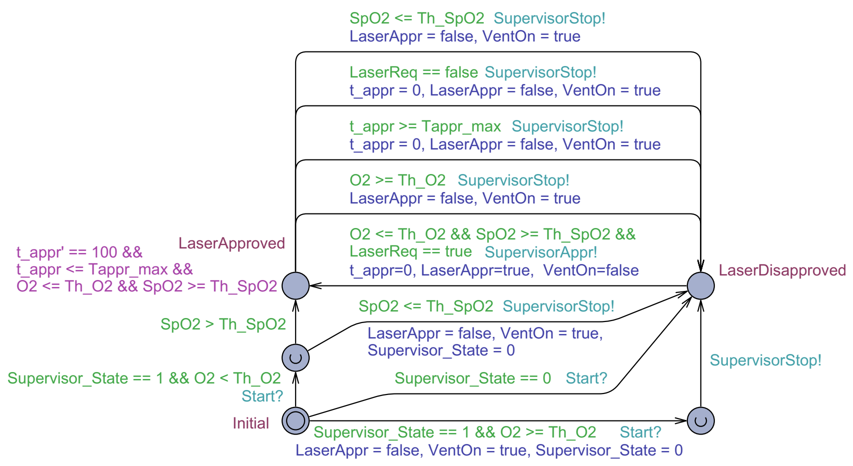

SpO2 <= Th_SpO2 SupervisorStop!
LaserAppr = false, VentOn = true
LaserReq == false SupervisorStop!
t_appr = 0, LaserAppr = false, VentOn = true
t_appr >= Tappr_max SupervisorStop!
t_appr = 0, LaserAppr = false, VentOn = true
O2 >= Th_O2 SupervisorStop!
LaserAppr = false, VentOn = true
O2 <= Th_O2 && SpO2 >= Th_SpO2 &&
LaserReq == true SupervisorAppr!
t_appr=0, LaserAppr=true, VentOn=false
LaserApproved
t_appr' == 100 &&
t_appr <= Tappr_max &&
O2 <= Th_O2 && SpO2 >= Th_SpO2
LaserDisapproved
SpO2 > Th_SpO2
SpO2 <= Th_SpO2 SupervisorStop!
LaserAppr = false, VentOn = true,
Supervisor_State = 0
SupervisorStop!
Supervisor_State == 1 && O2 < Th_O2
Start?
Supervisor_State == 0 Start?
Initial
Supervisor_State == 1 && O2 >= Th_O2 Start?
LaserAppr = false, VentOn = true, Supervisor_State = 0

Figure 9 Supervisor UPPAAL Model.

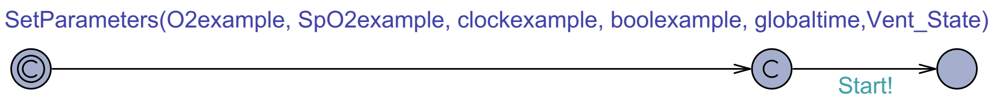

SetParameters(O2example, SpO2example, clockexample, boolexample, globaltime,Vent_State)
Start!

Figure 10 Initialization UPPAAL Model.

5.2.5 Initialization

Lastly, the initialization model has the purpose of initializing all constants that may have been adapted to real-world values during model adaptation. Using broadcast synchronization, a starting transition guarantees a common starting point for the whole model.

6 Experiments and Evaluation

To evaluate the on-line model checking approach with UPPAAL we carried out several experiments with our models. Real-world patient data necessary for the adaptation steps was extracted from the PhysioNet database, an open medical database offering a large collection of recordings of medical signals of various kind (http://www.physionet.org). Six different patient traces were assembled and used as a basis. Every patient trace was executed ten times yielding 60 experiments in total. Table 1 shows the PhysioNet data bases and the patient IDs of the data used. More information on the data can be found in the original thesis on this topic [16]. All experiments ran the system for 600 seconds where every three seconds a model adaptation and verification was performed. Thus, the workflow of every three-second cycle is as follows: first we adjust the O_2 and SpO_2 values in the model to the observed values. Then we try to verify the system properties for the next six seconds. And lastly, we evaluate the verification results such that if a property was not verified we derive that in three seconds an unsafe state occurs and thus emergency measures should be taken

Table 1 PhysioNet Databases and Patient IDs.

	Database	#1	#2	#3	#4	#5	#6
O_2 (CO_2)	MGF/MF	mgh077	mgh077	mgh089	mgh057	mgh019	mgh110
SpO_2	MIMIC v2	a45463	a45436n	439n	n10301n	a45611n	477n

Table 2 Relative Errors of O_2 and SpO_2 Estimation.

[%]	#1	#2	#3	#4	#5	#6
Min SpO_2	0	1.43	0.52	2.28	0.48	0.59
Max SpO_2	6.01	1.62	0.63	2.99	0.59	4.18
Avg SpO_2	1.59	1.54	0.60	2.82	0.54	2.27
Min O_2	0.6	18.0	12.3	16.8	11.3	8.3
Max O_2	66.0	23.2	14.0	20.5	15.3	10.5
Avg O_2	21.7	20.8	13.4	18.9	12.3	9.2

Table 3 Model Checking Execution Times.

[s]	Minimum	Maximum	Average
UPPAAL-SMC	0.033	0.32	0.047
PHAVer	0.571	1.445	0.727

before the unsafe state is reached. Note that if the models correctly predict the short-term behavior of O_2 and $Sp O_2$ and the supervisor strategy is effective such an emergency can not arise. The history window for the linear regression was 30 seconds in all cases. The confidence level for the statistical model checker was set to 99%. We evaluated three aspects of the approach: first we checked whether the safety requirements given in Subsection 5.1 are violated for any patient trace. Then we compared the relative errors of our O_2 and SpO_2 predictions to the values in the reference paper [14]. Lastly, we evaluated the execution times with a focus on the real-time requirements.

The first result is straightforward: during all experiments all three safety properties were satisfied at all times with the confidence level of 99%. Thus, our models seem to be accurate enough to predict the physiological parameters of the patient for a time bound of three seconds. Moreover, the supervisor strategy implemented in the models proves to be effective at preventing accidental tissue burns resulting from triggering the laser at inappropriate times.

Table 2 shows the relative errors of our parameter estimation. The SpO_2 estimates are very consistent and in general show a relative error of about 2%. These results are accurate enough to guarantee the safety of the patient with regards to the blood oxygen. In contrast, the estimation of windpipe oxygen is not that precise with an average relative error of about 16%. However, due to the supervisor strategy the safety of the patient is still guaranteed. Still, the laser could potentially be allowed to fire more often. Thus, a more sophisticated prediction strategy than linear regression is likely to yield better prediction results, which enable the supervisor to approve the use of the laser more often. Compared to the results of the original case study our SpO_2 results are slightly less accurate but still useful for safety statements. As the original case study does not specify exactly which patient traces were used as an experiment basis differences in the results may simply stem from the selection of different traces. For the O_2 results Li et al. provide no relative error results.

Table 3 shows the execution times of an adaptation step of the models and the following verification of the safety properties. Our experiments were carried out on a Macbook Pro 2.66 GHz with 4GB memory using iOS 10.6.8. In the experiments our approach took at worst 320 milliseconds for a cycle while in the original case study nearly 1.5 seconds elapsed. Unfortunately, the original case study does not specify the used hardware. Still, the approach using simulation of timed automata in UPPAAL-SMC for verification performs significantly better than the symbolic verification of hybrid automata in PHAVer. Thus, we assume the speedup can not be attribute only to differences in hardware, especially because our hardware is not on the top end. Looking at the absolute values with the hard real-time constraints of three seconds for one cycle in mind, using UPPAAL-SMC provides a performance advantage in practice. With execution times of about 10% of the real-time deadlines the implementation in a hard real-time system seems feasible.

7 Conclusion and Future Work

This paper presented the on-line model checking approach, a variant of model checking that allows reasoning about systems where accurate long-term models are unavailable. We implemented a medical laser tracheotomy case study using UPPAAL-SMC and used it to evaluate the on-line model checking approach in practice. The on-line model checking approach periodically adjusts the underlying system model to real-world values and analyzes the new models, e.g., for patient safety issues. The case study showed that this approach is capable of providing reliable safety guarantees even if the patient's physiological behavior is modeled only roughly using a simple linear regression approach when parameters are continuously adapted to the real-world values. Although this paper identifies on-line model checking as a useful technique to ensure safety of complex systems, further research is necessary to support this claim. Future research should focus on larger scale case studies and provide a unified approach including automatic adaptation interfaces to ease the development of systems that should be monitored using on-line model checking. Such an automatic adaptation interface would synthesize necessary means to adapt a model from a classical model checking model and execute the on-line model checking procedure to allow seamless simulation and verification of the system in question.

Acknowledgements. We thank the anonymous reviewers of MedicalCPS 2014 for references to related work and comments that helped to improve the presentation.

References

1 David Arney, Miroslav Pajic, Julian M. Goldman, Insup Lee, Rahul Mangharam, and Oleg Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems – ICCPS '10*, pages 139–148, Stockholm, Sweden, 2010. ACM New York, NY, USA.

2 Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C.R. Ramakrishnan, and Scott A. Smolka. Model Repair for Probabilistic Systems. In Parash Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *Lecture Notes in Computer Science*, pages 326–340. Springer Berlin Heidelberg, 2011.

3 Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on Uppaal 4.0. Technical report, Department of Computer Science, Aalborg University, Aalborg, Denmark, 2006.

4 Lei Bu, Dingbao Xie, Xin Chen, Linzhang Wang, and Xuandong Li. Demo Abstract: BACHOL – Modeling and Verification of Cyber-Physical Systems Online. In *Proceedings of the 3rd ACM/IEEE International Conference on Cyber-Physical Systems – ICCPS '12*, pages 222–222, Beijing, China, April 2012. IEEE.

5 Peter Bulychev, Alexandre David, Kim G. Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. In Herbert Wiklicky and Mieke Massink, editors, *10th Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL 2012)*, volume 85

of *Electronic Proceedings in Theoretical Computer Science*, pages 1–16, Tallinn, Estonia, July 2012.

6 Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, 2012.

7 Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. In *Real-time Systems Symposium (RTSS 2012)*, pages 263–272. IEEE, December 2012.

8 Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model Repair for Markov Decision Processes. In *Theoretical Aspects of Software Engineering (TASE 2013)*, pages 85–92. IEEE, July 2013.

9 Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical Model Checking for Stochastic Hybrid Systems. In *Electronic Proceedings in Theoretical Computer Science*, volume 92, pages 122–136, August 2012.

10 Arvind Easwaran, Sampath Kannan, and Oleg Sokolsky. Steering of Discrete Event Systems: Control Theory Approach. *Electronic Notes in Theoretical Computer Science*, 144(4):21–39, 2006.

11 Thomas A. Henzinger. The Theory of Hybrid Automata. In *Logic in Computer Science, 1996. LICS'96*, pages 278–292. IEEE, 1996.

12 Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and Verification of a Dual Chamber Implantable Pacemaker. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 188–203. Springer Berlin Heidelberg, 2012.

13 Andrew King, Dave Arney, Insup Lee, Oleg Sokolsky, John Hatcliff, and Sam 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 (SEHC 2010)*, pages 1–11. ACM, 2010.

14 Tao Li, Feng Tan, Qixin Wang, Lei Bu, Jian-Nong Cao, and Xue Liu. From Offline toward Real-Time: A Hybrid Systems Model Checking and CPS Co-design Approach for Medical Device Plug-and-Play (MDPnP). In *Proceedings of the 3rd ACM/IEEE International Conference on Cyber-Physical Systems – ICCPS '12*, pages 13–22, Beijing, China, April 2012. IEEE.

15 Tao Li, Qixin Wang, Feng Tan, Lei Bu, Jian-nong Cao, Xue Liu, Yufei Wang, and Rong Zheng. From Offline Long-Run to Online Short-Run: Exploring A New Approach of Hybrid Systems Model Checking for MDPnP. In *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability (HCMDSS-MDPnP 2011)*, 2011.

16 Xintao Ma. Online Checking of a Hybrid Laser Tracheotomy Model in UPPAAL-SMC. Master thesis, TU Hamburg-Harburg, December 2013.

17 Jonas Rinast, Sibylle Schupp, and Dieter Gollmann. State Space Reconstruction for On-Line Model Checking with UPPAAL. *VALID 2013, The Fifth Internation Conference on Advances in System Testing and Validation Lifecycle*, pages 21–26, 2013.

18 Yuhong Zhao and Franz Rammig. Online Model Checking for Dependable Real-Time Systems. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 154–161. IEEE, April 2012.

Integrating Safety Assessment into the Design of Healthcare Service-Oriented Architectures

Ibrahim Habli[1], Abdulaziz Al-Humam[1], Tim Kelly[1], and Leila Fahel[2]

1 **University of York, York, UK**
`{Ibrahim.Habli,aaah501,Tim.Kelly}@york.ac.uk`
2 **Calderdale and Huddersfield National Heath Service Foundation Trust, Halifax, UK**
`Lfahel@nhs.net`

Abstract

Most healthcare organisations are service-oriented, fundamentally centred on critical services provided by medical and nursing staff. Increasingly, these human-centric services rely on software-intensive systems, i.e. medical devices and health informatics, for improving different aspects of healthcare, e.g. enhancing efficiency through automation and patient safety through smart alarm systems. However, many healthcare services are categorised as high risk and as such it is vital to analyse the ways in which the software-based systems can contribute to unintentional harm and potentially compromise patient safety. This paper proposes an approach to modelling and analysing Service-Oriented Architectures (SOAs) used in healthcare, with emphasis on identifying and classifying potential hazardous behaviour. The paper also considers how the safety case for these SOAs can be developed in a modular manner. The approach is illustrated through a case study based on three services: ambulance, electronic health records and childbirth services.

1998 ACM Subject Classification K.4.1 [Public Policy Issues]: Computer-related health issues

Keywords and phrases Healthcare, Safety, Assurance, Service-Oriented Architecture

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.113

1 Introduction

Healthcare organisations are structured based on different, yet interdependent, services [1], e.g. emergency and urgent care services, cancer services and ambulance services. Many of these services are supported by software-intensive systems. These systems include medical devices [23] (e.g. infusion pumps) and networked health IT systems (e.g. distributed Electronic Health Record (EHR) systems). There is also an increased interest in improving the integration between the different healthcare services through enhancing software and data interoperability and standardising the interfaces between the health IT infrastructures and medical devices [2, 3, 4].

Despite their significant benefits, software-based services and systems can pose risks to patient safety [5, 6]. For example, between 2005 and 2009, the US Food and Drug Administration (FDA) received over 56,000 reports of issues related to the use of infusion pumps [7]. Many of the safety issues were traced to software defects. In the UK, the Medicines and Healthcare Products Regulatory Agency (MHRA) reported a continuous increase in medical device adverse incidents, totalling 9099 reports in 2009 [8]. The British Medical Journal (BMJ) also reported a significant increase in medical device recalls and warnings [9]. Given the criticality of certain software systems, e.g. EHR, assessing the extent to which the software behaviour contributes to safety hazards in healthcare services should be an integral part of the clinical risk assessment process and the overall clinical safety case [10, 11]. These safety hazards arise in clinical environments that are centred on the interactions between many different human, procedural and technological elements. Understanding and controlling the complex links between the software behaviour and the emergence of the clinical hazards (i.e. potential to cause preventable/unintentional harm) is a significant challenge. Addressing this challenge at the clinical level requires close collaboration between different stakeholders, primarily clinical experts, health scientists, safety analysts and system and software engineers. At the level of software systems, software engineers need to analyse failures within, and between, the software-intensive healthcare systems (e.g. incorrect dosage information provided automatically to an infusion pump by the EHR system [2]). Jointly with clinical experts, these engineers should also analyse how these failures can become hazardous behaviours once situated within a clinical environment (e.g. inaccurate blood pressure data due to physical factors such as the height of an IV bag [12]). Unfortunately, inadequate interaction between clinical experts and software engineers remains a major hurdle for achieving effective risk assessment of software-intensive healthcare services [13].

Importantly, assurance has to be provided that the risk of the software hazardous behaviours, identified in the clinical risk assessment process, has been adequately mitigated. Increasingly, this is being communicated in the form of a safety or assurance case [7, 10, 11]. Safety cases provide a reasoned and structured argument of how the available evidence, generated from testing and analysis, supports overall claims made about system safety. For a complex clinical environment, the overall safety case is not monolithic but compositional [14], comprising different safety arguments and evidence for the various healthcare services. These services, including systems and processes developed by different organisations, are clearly interdependent and so are their corresponding safety arguments [21].

This paper focuses on the safety assessment of services within healthcare Service-Oriented Architectures (SOAs). It presents a preliminary approach to modelling and analysing SOAs used in healthcare, with emphasis on identifying and classifying potential hazardous software behaviour. The approach is based on two existing modelling techniques for specifying individual services, and the processes connecting them, namely the Service oriented architecture Modelling Language (SoaML) [15] and the Business Process Modelling Notation (BPMN) [16]. This approach also builds on adapting two existing safety analysis techniques, namely the Functional Hazard Assessment (FHA) [17] and Software Hazard Analysis and Resolution in Design (SHARD) [18]. Further, the paper explores how the safety case for the SOA can be developed in a modular manner using the Goal Structuring Notation (GSN) [19]. The approach is illustrated through examples from an exploratory case study based on three services: ambulance, electronic health records and childbirth services.

The rest of the paper is organised as follows. An overview of the proposed SOA safety assessment approach is presented in Section 2, followed by a more detailed description of the safety analysis techniques in Sections 3 and 4. The safety analysis is illustrated in Section 5 using an exploratory case study. The nature and potential structure of a modular safety case for SOA are discussed in Section 6, followed by conclusions in Section 7.

2 SOA Safety Assessment

A service has been defined as "*a value delivered to another through a well-defined interface and available to a community*" [15]. The value that a service delivers, and the safety risks associated with it, can only be realised and understood in the sociotechnical setting of the

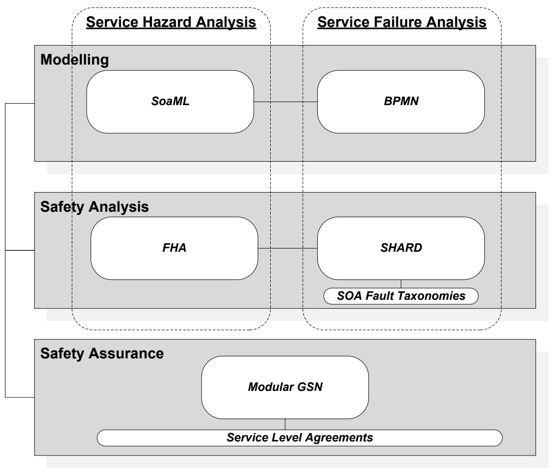


Figure 1 SOA Safety Assessment.

service (e.g. interactions between different types of services and processes). One approach to designing and representing this setting, especially for software-based services, is through an SOA. An SOA "*provides a paradigm for defining how people, organizations, and systems provide and use services to achieve results*" [15].

For healthcare services, the SOA paradigm can assist in the assessment of patient safety, in which accidents predominantly occur as a result of the interaction of different human, system and organisational behaviours. An overview of the SOA safety assessment approach proposed in this paper is depicted in Figure 1, and is briefly introduced in the rest of this section. A more detailed description of the safety analyses is provided in the next two sections.

Healthcare services, including interfaces and contracts between these services, are modelled in this approach using SoaML. The SoaML models depict a modular description of the healthcare SOA in which related contracts, interfaces and operations are encapsulated in, and provided by, self-contained services. In order to identify the hazards associated with services, we propose a variant of FHA, called Service Hazard Analysis (SHA), based on analysing three potential service deviations [17]: (1) *service not provided when required*, (2) *service provided when not required* and (3) *incorrect service*. The primary output of SHA is a set of safety requirements defined at the service level.

Next, in order to analyse the causes of the service hazards, identified using SHA, the detailed tasks implementing the SOA services and processes are modelled in BPMN, focusing particularly on the flow of information between interacting tasks. The SHARD safety analysis technique [18], which is a variant of the hazard and operability study (HAZOP) technique [20], is adapted to analyse the flow of information between the tasks represented within the SOA processes. We refer to this as Service Failure Analysis (SFA). The analysis and resulting failure modes are driven by the application of a set of guidewords: *omission, commission, early, late* and *incorrect value* [18]. Each of the failure modes is then associated with specific service faults linked to existing SOA fault taxonomies (i.e. to make the analysis SOA-specific [25]). The output of this analysis is a set of derived service safety requirements.

Finally, the above analyses are used as a core part of the safety evidence base to inform the structure of the overall safety argument for the SOA safety case. Given the modular nature of SOA, the safety argument is structured based on different, yet interrelated, argument modules [14]. Most of these argument modules correspond to the safety justification of a

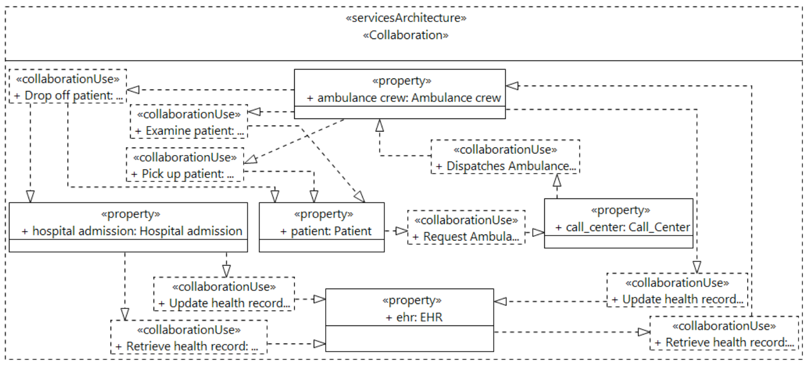

«servicesArchitecture»
«Collaboration»
«collaborationUse» + Drop off patient: ...
«property» + ambulance crew: Ambulance crew
«collaborationUse» + Examine patient: ...
«collaborationUse» + Pick up patient: ...
«collaborationUse» + Dispatches Ambulance...
«property» + hospital admission: Hospital admission
«property» + patient: Patient
«collaborationUse» + Request Ambula...
«property» + call_center: Call_Center
«collaborationUse» + Update health record...
«property» + ehr: EHR
«collaborationUse» + Update health record...
«collaborationUse» + Retrieve health record: ...
«collaborationUse» + Retrieve health record:...

Figure 2 ServicesArchitecture Model.

specific individual service. Typically, within SOA, services interact based on pre-defined Service-Level Agreements (SLAs) [22]. Similarly, the relationship between different argument modules can be specified using argument contracts [14]. The mapping between SLAs and the argument contracts drives the overall structure of the SOA safety argument and offers traceability between the design, represented in the SOA models, and the safety assurance, represented in the modular safety argument.

3 Service Hazard Analysis (SHA)

Safety analysis processes are centred on identifying, analysing and managing hazards. For healthcare SOA, identifying the hazards associated with the services is essential for defining the service safety requirements, which should influence the architectural design. In this paper, hazard identification and classification is carried out based on SHA, using a high-level representation of the SOA in SoaML. SHA consists of four steps:

1. **Identify a service:** high-level services are captured in SoaML *ServicesArchitecture* models. A *ServicesArchitecture* represents how *Participants* collaborate, by producing and consuming *Services* to achieve goals. Figure 2 shows an example *ServicesArchitecture* model that captures *Participants* (e.g. *ambulance crew*, *patient* and *hospital-admission*) and *Services* (e.g. *Request ambulance*, *Examine patient* and *Update health record*).
2. **Identify the service failure modes:** the modes or types of failures that are considered in this step are as follows: (1) *service not provided when required*, (2) *service provided when not required* and (3) *incorrect provision of service*. These are intended for use as prompts for identifying the different ways in which the service can fail. The use of these types requires the safety analyst to interpret the meaning and relevance of specific failures in the context of the service in hand (e.g. *incorrect drug dosage* has to be interpreted in the context of specific classes of drugs for specific conditions or combinations of conditions).
3. **Determine the safety effects and severity of each service failure mode:** the adverse consequences of the failure mode should be determined, taking into consideration different factors, e.g. the condition of the patient and other services and systems (not just the software systems). A safety classification should also be defined (e.g. *Catastrophic, Major, Considerable* or *Significant*), typically based on Hazard/Risk Matrices (HRM) defined in standards or by the healthcare organisation [10, 11]. In terms of determining the effects of service failures, a useful feature of SoaML *ServicesArchitecture* models is that they include a representation of service interaction and the *Participants* that use these services (i.e. making it easier for the analyst to trace the effects of a failure mode). For example,

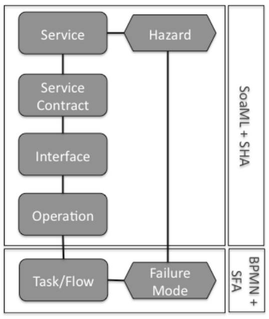


Figure 3 Link between SoaML and BPMN Models.

administrating certain drugs or treatments (e.g. radiotherapy) might have potential hazardous effects on both the patients and the caregivers (both represented as SoaML *Participants*).

4. **Provide recommendations or service safety requirements:** based on the identified failure modes and their classification, recommendations, preferably in the form of service safety requirements, should be generated, where the rigor/integrity with which the requirements need be met should be proportionate to the severity of the failures (i.e. higher degrees of severity requires more stringent integrity requirements and more rigorous processes) [24, 27]. SHA is implementation-independent and should be performed at the early specification and design stages of the SOA.

4 Service Failure Analysis (SFA)

Service hazards identified using SHA typically emerge from a combination of factors, whether human, organisational or technological. Understanding and analysing the causes of these hazards is an important step towards eliminating or reducing the risk of these hazards. In this section we introduce SFA to examine the detailed implementation of the services, and identify how failures, specifically interaction/information flow failures, can contribute to service hazards. SFA consists of five steps:

1. **Identify a flow between two tasks:** SFA, as proposed in this paper, is based on a behavioural model of the SOA represented in BPMN [16]. BPMN provides the ability to communicate and represent internal procedures, based on *Processes*, using a graphical and structured notation. Typically, these *Processes* represent workflows of connected *Tasks* (i.e. atomic *Activities*), grouped into *Swimlanes* (i.e. a container for organising *Activities*). Figure 4 shows an example BPMN process model, comprising connected *Tasks* that are organised into five different *Swimlanes* (e.g. *ambulance crew*, *patient* and *hospital-admission*). The SoaML model used for SHA (Section 3) and the BPMN model used for SFA, in this section, are linked are follows (Figure 3): Services in SoaML interact through defined *Contracts*. These *Contracts* have explicit *Interfaces* that offer a number of *Operations*. Each of these *Operations* is then linked to a *Task* in BPMN, enabling traceability between the different models at different abstraction levels. Further, each *Participant* in SoaML is represented as a *Swimlane* in BPMN. This step in SFA involves the selection of a link between two BPMN Tasks that captures a flow in terms of inputs, outputs, data, sequence and timing.
2. **Identify flow failure modes:** similar to Step 2 in SHA, this step uses a number of guidewords, this time based on SHARD, to determine the ways in which the selected

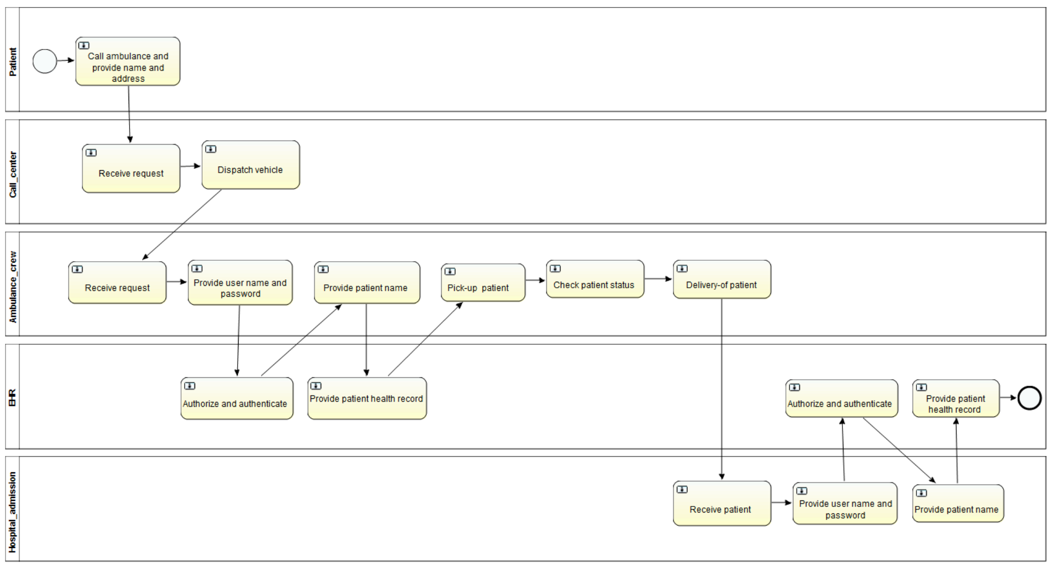

Patient
Call ambulance and provide name and address
Call_center
Receive request
Dispatch vehicle
Ambulance_crew
Receive request
Provide user name and password
Provide patient name
Pick-up patient
Check patient status
Delivery-of patient
EHR
Authorize and authenticate
Provide patient health record
Authorize and authenticate
Provide patient health record
Hospital_admission
Receive patient
Provide user name and password
Provide patient name

Figure 4 BPMN Model.

flow can deviate from its intended usage. Five guidewords are used here: *omission, commission, early, late* and *value* [18]. The failures derived from the use of each of these guidewords should be interpreted in the context of the SOA design.

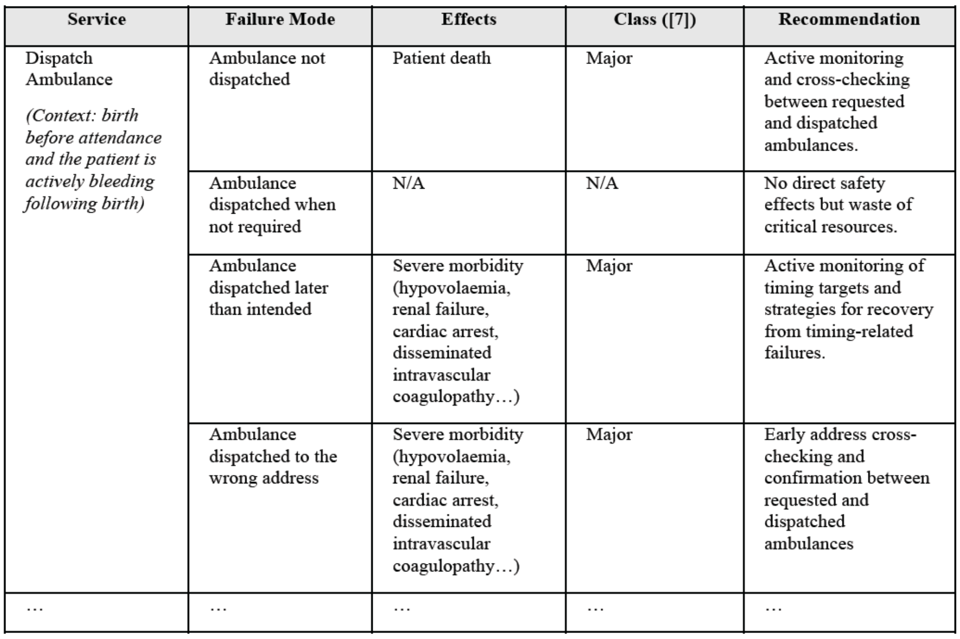
3. **Determine the potential causes of each flow failure mode:** causes can be a combination of technical, human and organisational events or conditions. For technical causes in particular, we use the SOA fault taxonomy developed by Bruning et al [25]. This fault taxonomy is well structured and categorises faults based on the SOA lifecycle phase in which they can emerge: (1) publishing, (2) discovery, (3) composition, (4) binding and (5) execution faults. The advantage of using these fault types is that they are SOA-specific. However, again, these fault types should be used as prompts or hints for safety analysts rather than as an exhaustive list of all possible SOA faults.
4. **Determine the potential effects of each flow failure mode:** the potential effects should be recorded and should be examined in terms of the contribution that they can make to the hazardous service failure modes, identified in SHA, or possibly the contribution that they might make to new hazardous behaviours (i.e. missed during SHA).
5. **Provide detailed safety requirements or design recommendations:** where a failure mode contributes to one or more hazards, one or more safety requirements should be defined to address the failure mode. Further, some design recommendations could be made for addressing the failure mode, e.g. based on existing safety tactics in the software architecture literature or SOA-specific dependability tactics (e.g. in [26], which are centred on the use of service redundancy, diversity, graceful degradation, monitoring and containment).

5 Exploratory Case Study

In this section, we illustrate the use of SHA and SFA using extracts from an exploratory case study, which is based on three healthcare services: ambulance, EHR and childbirth services.

A subset of these services is represented in the SoaML *ServicesArchitecture* model shown in Figure 2. It covers the services used (i.e. produced and consumed) from the point a phone

Table 1 SHA Results.



Service	Failure Mode	Effects	Class ([7])	Recommendation
Dispatch Ambulance *(Context: birth before attendance and the patient is actively bleeding following birth)*	Ambulance not dispatched	Patient death	Major	Active monitoring and cross-checking between requested and dispatched ambulances.
	Ambulance dispatched when not required	N/A	N/A	No direct safety effects but waste of critical resources.
	Ambulance dispatched later than intended	Severe morbidity (hypovolaemia, renal failure, cardiac arrest, disseminated intravascular coagulopathy…)	Major	Active monitoring of timing targets and strategies for recovery from timing-related failures.
	Ambulance dispatched to the wrong address	Severe morbidity (hypovolaemia, renal failure, cardiac arrest, disseminated intravascular coagulopathy…)	Major	Early address cross-checking and confirmation between requested and dispatched ambulances
…	…	…	…	…

call is made to request an ambulance (for a pregnant woman) to the point at which the patient is admitted to a hospital (labour ward). Another set of *ServicesArchitecture* models was created to capture subsequent stages e.g. fetal monitoring, first and second stages of labour, caesarean section and postnatal care (not discussed further in this paper).

Table 1 shows an extract from the SHA results when applied to the *Dispatch Ambulance* service. The analysis establishes the potential safety criticality of the *Dispatch Ambulance* service, based on the severity of the worst credible effects of the identified failure modes, leading to stringent safety requirements allocated to the *Dispatch Ambulance* service.

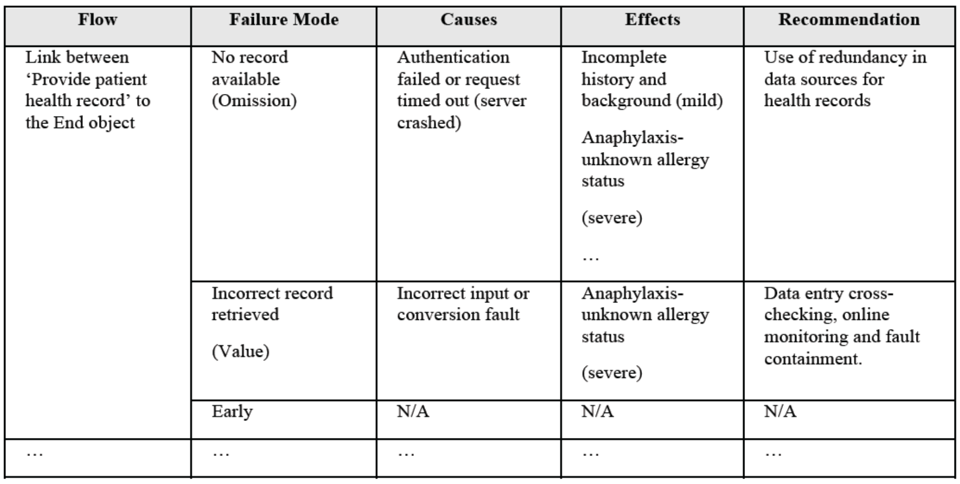
As can be observed from the results, the analysis is collaborative in nature, demanding inputs from both engineers (e.g. determining how the service can fail) and clinicians (e.g. assessing effects on patients). Figure 4 presents a more detailed model of the SOA using BPMN, with more emphasis of the SOA process. BPMN Tasks in this process are mapped into the *Operations* in the *Interfaces* provided by each SoaML *Service* while BPMN *Swimlanes* are mapped onto the SoaML *Participants*.

Considering each flow between the *Tasks*, SFA was applied to the BPMN model. A sample outcome is shown in Table 2, considering the last flow in the BPMN model, from *Provide patient health record* to the End object (i.e. admission to labour ward). The analysis shows how the lack of patient information from the EHR, and more seriously incorrect information, can potentially lead to adverse patient complications.

6 SOA Safety Cases

Establishing and justifying an acceptable level of confidence in the safety of software-based healthcare services will often require different safety arguments and evidence generated by different service owners or providers. One approach to representing this compositionality of the overall safety case for service-based systems is through modular GSN [19]. Modular GSN supports the definition of the safety case based on the composition of different, but

Table 2 SFA Results.



Flow	Failure Mode	Causes	Effects	Recommendation
Link between 'Provide patient health record' to the End object	No record available (Omission)	Authentication failed or request timed out (server crashed)	Incomplete history and background (mild) Anaphylaxis-unknown allergy status (severe) …	Use of redundancy in data sources for health records
	Incorrect record retrieved (Value)	Incorrect input or conversion fault	Anaphylaxis-unknown allergy status (severe)	Data entry cross-checking, online monitoring and fault containment.
	Early	N/A	N/A	N/A
…	…	…	…	…

interrelated, argument modules. When argument modules are composed a record of the agreement and consistency can be recorded using a safety case contract [14]. This contract "*contains definition of the relationships between two modules, defining how a claim in one supports the argument in the other*" [19].

Modularity in the definition of services lends itself to the concept of modular safety cases. Service owners or producers often rely on other services when guaranteeing and providing their own services. Similarly, claims in certain argument modules can only be said to be substantiated (the guarantee clause) if claims or evidence are available in other argument modules that offer sufficient support (the rely clause). Figure 5 shows a preliminary GSN modular structure for the safety case for the healthcare SOA considered in the case study in the previous section. The safety case has three categories of argument modules:

- *Top-Level Argument Module* includes a hazard-directed argument, which covers the main safety claims concerning the identified service hazards, including interaction hazards;
- *Ambulance Service Argument Module, EHR Service Argument Module, Hospital Admission Service Argument Module* and *Service Interactions Argument Module* include hazard mitigation arguments for the hazards posed by the different services and their interactions; and
- *Ambulance Crew and Call Centre Argument Module, EHR System Argument Module* and *Hospital Admission Argument Module* include detailed arguments concerning the systems and organisations responsible for implementing the healthcare services.

However, the safety case structure in Figure 5 and the case study description in Section 5 do not take into account that in larger regions (which include sub-regions), the same types of healthcare services can be offered by a variety of different ambulance service providers and hospitals. In such situations, the high-level argument structure in Figure 5, comprising the top-level hazard-directed argument module and hazard mitigation argument modules potentially need not change. Where the implementation of the provision of services changes, the bottom tier safety argument structure will need to change. For example, if a healthcare region was divided into two sub-regions, say east and west, with different hospitals, ambulance crews and call centres, then the safety arguments concerning the systems and organisations implementing the healthcare services would be different, taking into account the specific design and operational issues concerning these systems and organisations.

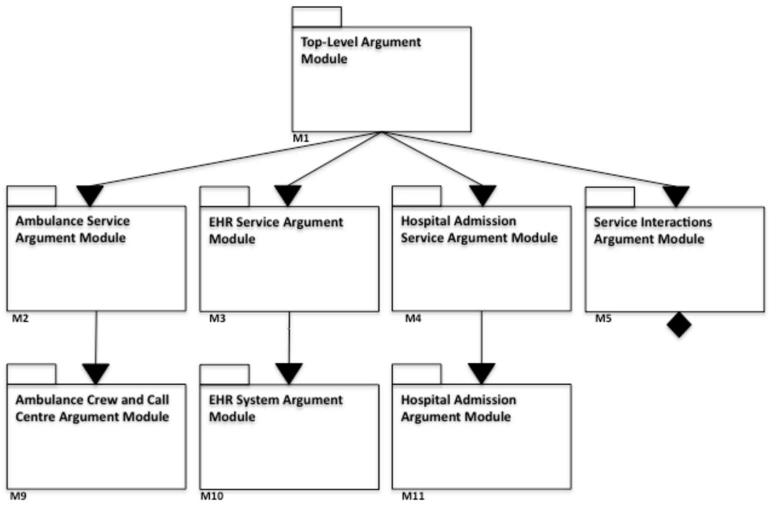

Top-Level Argument Module
M1
Ambulance Service Argument Module
M2
EHR Service Argument Module
M3
Hospital Admission Service Argument Module
M4
Service Interactions Argument Module
M5
Ambulance Crew and Call Centre Argument Module
M9
EHR System Argument Module
M10
Hospital Admission Argument Module
M11

Figure 5 SOA Safety Argument.

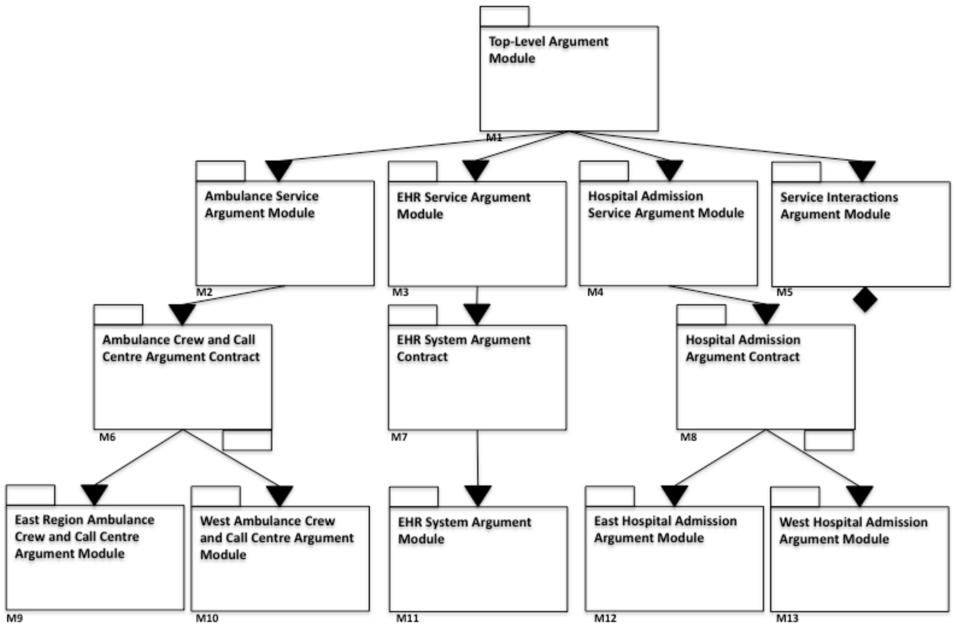

Top-Level Argument Module
M1
Ambulance Service Argument Module
M2
EHR Service Argument Module
M3
Hospital Admission Service Argument Module
M4
Service Interactions Argument Module
M5
Ambulance Crew and Call Centre Argument Contract
M6
EHR System Argument Contract
M7
Hospital Admission Argument Contract
M8
East Region Ambulance Crew and Call Centre Argument Module
M9
West Ambulance Crew and Call Centre Argument Module
M10
EHR System Argument Module
M11
East Hospital Admission Argument Module
M12
West Hospital Admission Argument Module
M13

Figure 6 SOA Safety Argument (including Contracts).

Figure 6 shows a modified representation of the safety case structure in Figure 5, taking into consideration the need for two different safety arguments (east and west) for each of these argument modules: *Ambulance Crew and Call Centre Argument Module* and *Hospital Admission Argument Module*. However, it can be noticed that the *Ambulance Service Argument Module* and *Hospital Admission Service Argument Module* are now supported by the *Ambulance Crew and Call Centre Argument Module* and *Hospital Admission Argument Module* (east and west) via two contract modules. This loose coupling between these argument modules can help in minimising the impact of change to the safety case when different systems and organisations are used to provide the services (e.g. a third sub-region is introduced).

7 Conclusions

This paper has presented a preliminary approach to integrating safety assessment into the design of healthcare SOA, covering three aspects: modelling using SoaML and BPMN, safety analysis using SHA and SFA (adapting FHA and SHARD) and safety assurance using modular GSN. An exploratory case study was also discussed, based on three services: ambulance, electronic health records and childbirth services. We are currently developing a tool-support platform for the above safety assessment approach in order to improve traceability between the design, safety analysis and safety assurance models and provide automated means for supporting the safety analysis process. We are also developing a set of modular safety argument patterns that analysts can use as a basis for structuring SOA safety arguments.

Finally, examining the safety impact of interactions between services and service providers remains a significant challenge, especially for healthcare services that span both primary and secondary care and cover more than one medical condition (e.g. care and treatment of diabetes in pregnancy which involves GPs, obstetricians, midwives, endocrinologists, and diabetes-specialist nurses). Our future work will examine means for identifying, modelling and analysing these interactions by integrating search-based technologies (e.g. simulation and model-checking) into the above SOA safety assessment approach.

References

1 Keen J. *What is a care pathway?* 4th International Workshop on Software Engineering in Health Care, Zurich, Switzerland, June 2012

2 Sokolsky, O., Lee, I., and Heimdahl, M. *Challenges in the regulatory approval of medical cyber-physical systems.* International Conference on Embedded Software, Taipei, Taiwan, October 2011

3 Arney, D., Goldman J. M., Whitehead, S. F., and Lee, I. *Synchronizing an X-ray and anesthesia machine ventilator: a medical device interoperability case study.* International Conference on Biomedical Electronics and Devices, Porto, Portugal, January, 2009

4 IEC. *IEC 80001-1:2010, Application of Risk Management for IT-Networks Incorporating Medical Devices – Part 1: Roles, Responsibilities and Activities.* IEC, October, 2010

5 Rakitin, R. *Coping with defective software in medical devices.* IEEE Computer. 39, 4, April 2006

6 Koppel, R., Metlay, J. P., Cohen, A., Abaluck. B., Localio, A. R., Kimmel, S. E., and Strom, B. L. *Role of computerized physician order entry systems in facilitating medication errors.* The Journal of Urology, March, 2005

7 FDA. *Total Product Life Cycle: Infusion Pump Premarket Notification 510(k) Submissions.* April 2010

8 MHRA. *Adverse Incident Reports 2009.* Device Bulletin DB2010 (03), 2009

9 Heneghan, C., Thompson, M., and Billingsley, M. *Medical device recalls in the UK and the device regulation process: retrospective review of safety notices and alerts*. BMJ, May 2011

10 Health and Social Care Information Centre. *Clinical Risk Management: its Application in the Manufacture of Health IT Systems*. ISB 0129, 2013

11 Health and Social Care Information Centre. *Clinical Risk Management: its Application in the Deployment and Use of Health IT Systems*. ISB 0160, 2013

12 Hofmann, R. *Modeling Medical Devices for Plug-and-Play Interoperability*. MS Thesis, MIT, 2007

13 Sujan, M., Koornneef, F., Chozos, N., Pozzi, S., and Kelly, T. *Safety cases for medical devices and health IT: involving healthcare organisations in the assurance of safety*. Health Informatics Journal, 18, 4, September 2013

14 Fenn, J., Hawkins, R., Kelly, T., and Williams, P. *Safety case composition using contracts: refinements based on feedback from an industrial case study*. Safety-Critical Systems Symposium, Bristol, UK, February 2007

15 OMG. *Service oriented architecture Modeling Language (SoaML) Specification*. Version 1.0.1, May 2012

16 OMG. *Business Process Model And Notation (BPMN)*. Version 2.0, 2011

17 SAE. *ARP4761. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. December 1996

18 Pumfrey, D. *The Principled Design of Computer System Safety Analyses*. PhD Thesis, The University of York, September 1999

19 GSN Standard Committee. *Goal Structuring Notation (GSN)*, [On-line]. `http::www.goalstructuringnotation.info`

20 Kletz T. *Hazop and Hazan*. 4th ed., Taylor and Francis, 2006

21 Brown, A., Fenn, J., and Menon. *Issues and considerations for a modular safety certification approach in a service oriented architecture*. IET International System Safety Conference, 2010

22 Keller, A., and Ludwig, H. *The WSLA framework: specifying and monitoring service level agreements for web services*. Journal of Network and Systems Management, 11, 1, March 2003

23 ISO. *ISO 14971:2012: Medical devices. Application of Risk Management to Medical Devices*. July 2012

24 Habli, I., Hawkins, and R., Kelly. *Software safety: relating software assurance and software integrity*. International Journal of Critical Computer-Based Systems (IJCCBS). 1, 4, November 2010

25 Bruning, S., Weissleder, S., and Malek. *A fault taxonomy for service-oriented architecture*. High Assurance Systems Engineering Symposium, Dallas, US, 2007.

26 Buckley, I., Fernandez, E.B., Anisetti, M., Ardagna, C., Sadjadi, M., and Damiani, E. *Towards pattern-based reliability certification of services*. On the Move to Meaningful Internet Systems, Hersonissos, Greece, 2011.

27 Hawkins, R., Habli, I., and Kelly, T. *The Principles of Software Safety Assurance*. 31st International System Safety Conference, Boston, USA, August 2013

Design Pillars for Medical Cyber-Physical System Middleware*

David Arney[1], Jeff Plourde[1], Rick Schrenker[1], Pratyusha Mattegunta[1], Susan F. Whitehead[1], and Julian M. Goldman[1,2]

1 **MD PnP Program, Massachusetts General Hospital, Boston, MA, USA**
`info@mdpnp.org`

2 **Harvard Medical School, Cambridge, MA, USA**
`jmgoldman@mgh.harvard.edu`

Abstract

Our goal is to improve patient outcomes and safety through medical device interoperability. To achieve this, it is not enough to build a technically perfect system. We present here our work toward the validation of middleware for use in interoperable medical cyber-physical systems. This includes clinical requirements, together with our methodology for collecting them, and a set of eighteen 'design pillars' that document the non-functional requirements and design goals that we believe are necessary to build a successful interoperable medical device system. We discuss how the clinical requirements and design pillars are involved in the selection of a middleware for our OpenICE implementation.

1998 ACM Subject Classification J.3 Life and Medical Sciences, D.2.1 Requirements/Specifications

Keywords and phrases Medical Device Interoperability, Clinical Requirements, Design Pillars, Requirements Elicitation, Validation

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.124

1 Introduction

Medical device interoperability has the potential to reduce healthcare costs, improve patient outcomes and improve patient safety. Achieving interoperability requires that medical devices (including software applications) and other equipment share the same information model and communication protocol. This enables applications to work with any source of compatible data regardless of the manufacturer or specific device type. In this paper, we concentrate on the communication protocol aspect of interoperability, in particular the role of middleware.

A system using separate medical devices is a distributed system. There is a long and rich history of work in the field of distributed systems that can directly inform the development of interoperable medical cyber-physical systems (MCPS). One broadly accepted tenet of this work is that network architecture can be broken down into a number of layers; this is perhaps most commonly illustrated by the Open Systems Interconnection (OSI) Seven Layer Model. Breaking network architecture into these layers allows designing and reasoning about

* This publication was made possible by grant number 1U01EB012470 from NIH/NIBIB and award number W81XWH-12-C-0154 from Department of Defense US Army Medical Research and Materiel Command. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH/NIBIB or the US Army or the Department of Defense.

them independently – a transport layer can operate on many different network and data link layers that in turn can work with a multitude of physical layers. Middleware is software that implements some of these middle layers between an application and networking hardware. There are a great number of middleware implementations with widely varying capabilities that implement various subsets of the seven layers. Choosing an appropriate middleware for a particular domain is thus a complex undertaking that requires an understanding of what applications need and expect from the network.

Systems Engineering similarly has a long history with many lessons that can inform MCPS development. The most important lesson here is that user needs must be used to validate system designs. Broadly stated, technical requirements are used for verification (that "the system was built right") and user requirements are used for validation (that "you built the right system"). A technically flawless system that does not satisfy user needs will not be used.

In this paper, we present two types of user requirements and discuss how they can be used to validate middleware for MCPS. The two types of requirements are Clinical Requirements, which capture the needs of clinicians, clinical engineers, biomedical engineers, and other medically-oriented users and Design Pillars, which are broad-scope non-functional requirements that we and our team of collaborators have formulated over ten years of building MCPS implementations.

2 Implementations of ASTM F2761

Interoperable MCPS follow the architecture described in ASTM F2761-09(2013) [7]. This standard does not call out a specific middleware, but includes high-level requirements for any interoperable clinical environment. We aim specifically to support 'plug-and-play' interoperability, including the composition of devices that may be used as by applications as components of the integrated system. Other existing standards such as ISO/IEEE 11073 specify all layers of the network stack, from physical to application. We reuse parts of the 11073 family, notably the terminology set (11073-10101) and parts of the information model in our OpenICE implementation [13], adapting them as needed for use with a middleware.

ASTM F2761 defines a functional architecture for interoperable medical systems, illustrated in Figure 1. Clinicians use applications that interact with medical devices and that run on a Supervisor. Patient-connected medical devices and other equipment connect to the system via adapters (for legacy devices) or a built-in ICE Equipment Interface. The Network Controller ties together devices with the Supervisor, and sends data to the Data

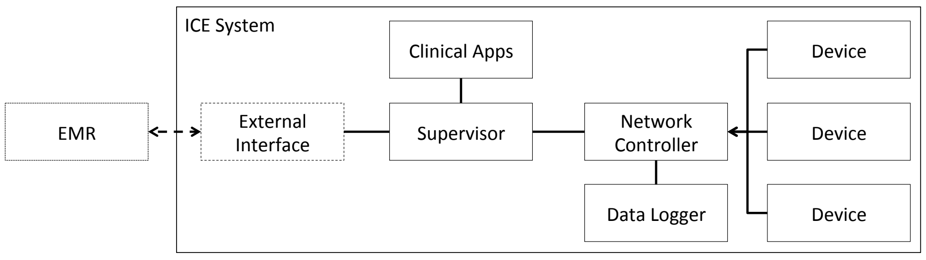


Figure 1 ICE Functional Architecture.

Logger, which functions analogously to the black-box recorder of an airplane. The ICE system communicates with outside resources like an electronic health record (EHR) system, physician order entry system, or pharmacy system through External Interfaces. Though not shown in Figure 1, the patient and clinicians are key elements of the system, which is intended explicitly to improve patient safety and outcomes. Middleware can be used to implement this functional architecture by taking on responsibilities of the Network Controller and, to a lesser extent, the Supervisor. With a sufficiently capable middleware, almost all functions of the network controller are subsumed into the middleware.

Over the last 10 years we have built in our lab numerous prototype medical distributed systems [9] utilizing a variety of connectivity solutions. We started by using approaches built on web services such as SOAP and industrial systems like MODBUS to synchronize an X-Ray exposure with an anesthesia machine ventilator [4] [8]. This was followed by an infusion pump safety interlock built on a deterministic, hard real-time network implemented on custom FPGA hardware [3]. We have done extensive work on patient-controlled analgesia pumps including formal analysis of pumps [5], formal analysis of systems [6] [12] [2], and closed-loop control [14]. In addition to publications, these systems were presented at medical conferences including HIMSS, the American Society of Anesthesiologists annual conference, the Society for Technology in Anesthesiology annual meeting, and other venues. Feedback gathered from clinicians at these venues has gone into each iteration and been included in the clinical requirements and design pillars. This body of work forms the basis for claiming the validity of the design pillars in Section 3.

3 Design Pillars for Successful Interoperability

We have given the name Design Pillars to the set of non-functional requirements that summarize the approach that we believe is necessary to achieve safe, adoptable medical device interoperability. Other standards, including ISO 14971, IEC 60601, and FDA's guidance documents on risk management also include important guidance for interoperable systems. This list has a different focus, aiming to capture the normally unwritten goals and philosophy needed to achieve successful interoperability. These design pillars are a work in progress and we welcome additions, comments, and arguments about them.

No Silos. Work toward a concrete implementation of the ICE standard has matured to a point where harmonization is required. Silos of interoperability work that cannot successfully interoperate are self-defeating. By identifying the most important characteristics of a middleware for ICE systems we can begin the process of selecting the most appropriate foundation for the platform. Future work can then proceed on a new generation of clinical applications that operate within the scope of this platform.

Open Source. There must be an open source reference implementation. We will share our software code and documentation with the community, giving them the necessary tools to adapt and utilize our software, including commercial reuse. We will prefer dependencies (tools and software libraries) that are available with open source and little or no cost. We will demonstrate how a member of the broader community can easily make use of our work. Our development is done in an open repository. Documentation, the ticketing system, and bug tracking are all publicly visible. We encourage anyone interested to contribute to this effort. To demonstrate the feasibility of proposed solutions, prototype implementations are required. Such prototypes are most useful when they can be shared. This does not preclude closed-source implementations and commercialization once the conceptual use of a middleware to build a platform for ICE apps has been proven. An open source reference implementation permits other implementers to perform testing and reuse code as appropriate.

Existing Standards. Interoperability must be built on standards, utilizing existing software standards to the greatest possible extent. Where existing standards must be corrected, completed, or extended, the rationale must be documented.

Security. Medical systems inherently touch human lives and private information. ICE implementations must be secure to the greatest extent possible. Security in this domain encompasses a tremendous range. Most relevant to middleware selection are the needs for identification, authentication, and authorization of connected devices, clinical users, and patients. Information in transit and at rest must be secured with appropriate use of encryption.

There is an apparent tradeoff between security and usability. Security features must not slow down or prevent urgent clinical use.

Scalability and Extensibilit.y ICE implementations must scale gracefully. A platform that enables a revolution in bedside devices must scale to support the next generation of devices. Therefore even while we're building concrete prototypes with the current generation of medical devices we must anticipate a newer generation of devices that we expect will furnish higher resolution data streams. Software simulation should be used for initial stress testing, testing on hardware may also be necessary. The platform, supporting current generation devices, should exhibit a great deal of underutilized capacity. ICE encompasses data and control at scales from the bedside to the globe and must support integration at these scales.

High Availability. We are building software that must guarantee high availability. ICE supports the integration of multiple sources of patient data. Components that fail should be seamlessly replaced by redundant data sources or other components if they are available. Put another way, risk control measures need to take into account component malfunctions. ICE should support achieving single fault tolerance for applications. Dependability is availability plus reliability – i.e., it's there when you need it and won't break while you're using it.

Performance. Performance is another key to acceptance by the clinical community. Sluggish performance may be inconsequential in the laboratory setting but a poorly performing system in the clinic consumes a critical resource; the clinician's time. Poor performance can also encourage clinicians to marginalize the system; isolating the threat to their workflow. ICE implementations must support dynamic detection and reporting of performance degradation.

Visibility of runtime configuration. ICE implementations must surface the state of the system; for instance, the connection state of devices should be readily available to a user. When the system is in an undesirable state, for example lacking connectivity to a critical medical device, it is important that information be made available. A system operating with hidden states will never earn the confidence of clinicians, but neither will a system cluttered with unnecessary information. The platform must also allow for the plug-and-play assembly of medical devices and because of this the configuration at runtime is the only source of information about how the system is configured.

Generic Interface. Each component will share its data representation in common. Software shared in common among components will mediate all communication.

External Connectivity. ICE implementations must interface with external systems. Some examples of external systems include an EMR system, an eHealth eXchange (NwHIN), departmental systems (such as pharmacy), or network time protocol (NTP) servers inside or outside of the hospital or home.

Novel Applications. ICE implementations must enable the development of novel applications that run within their frameworks. The point of ICE is to enable new clinical applications to improve patient outcomes and safety.

Clinical Scenarios. Requirements for ICE implementations should be derived from publicly available clinical scenarios so that traceability of technical requirements can be maintained. Technical requirements must be linked to clinical requirements which are derived from clinical scenarios. Technical design will also be informed by those scenarios and linkages between design decisions and high-level clinical requirements must be documented.

Community Involvement. Developers of ICE implementations must maintain awareness of developments in other large-scale initiatives and relevant standards bodies. The linkages between external developments and implementation design decisions must be explicitly documented. Findings should be shared back with Standard Development Organizations where possible.

Forensic Data Logging. ICE implementations must create a credible log of all activity so that adverse events can be investigated in order to surface and trace root causes of faults in the distributed system. Every aspect of implementations must avoid any data pathways that may "sidestep" this logging (while balancing this with our need for scalability and security). Information known to bypass the data logger must be documented with a rationale.

Plug and Play. Components can be added to and removed from the system at any time. The system must dynamically determine and monitor the presence of components. In the interests of security, scalability, and performance components may be refused by the system for various reasons but this refusal must be surfaced per 3. Applications must handle the disappearance of required data and control sources or sinks and the appearance of new sources and sinks gracefully.

Regulatory Pathway. ICE implementations operate in a regulated space. The regulators vary geographically, but the need to demonstrate the safety and essential performance of ICE systems and components is universal. To achieve this, ICE implementations should be designed and implemented in such a way as to facilitate regulatory clearance. Following the other design pillars should ease regulatory burdens.

Industry Adoptability. The goal of ICE to achieve dramatic improvements in patient outcomes and safety can only be met if such systems are commercially available. To this end, ICE implementations (particularly open source implementations) should facilitate commercial reuse. At the same time, common networking pieces such as data representations must be shared and developed in common.

Human Factors. The user interface and other human factors issues need to be carefully designed and tested in realistic environments so that new hazards that are introduced are adequately controlled. For instance, when a device is operating as a component of a larger system, its front panel must display an indicator that it's under remote control.

4 OpenICE and Clinical Requirements

We have built an open source implementation of an interoperable medical device system based on the ICE standard. We call this implementation OpenICE, and it is available on SourceForge [13]. This implementation is built using the OMG standard middleware DDS [11]. Implementing using DDS has helped to inform our requirements, but our clinical requirements and design pillars are not tied to DDS. We have spent extensive time building implementations on a variety of platforms and we believe that DDS is a good (stable, well-supported, variety of implementations available, etc.) choice, but not the only choice. We hope that our approach is useful to anyone implementing an interoperable medical system on any platform.

An ideal middleware would support an abstract API that would permit many instantiations on varying hardware and software platforms. DDS approaches this ideal in that the OMG standard specifies an API that may be implemented in many ways. ICE implementations have a wide range of requirements, for instance for timing and latency. Creating an abstract API that will support scalability, reliability, fault tolerance, and safety analysis is is not trivial, and we believe that this is a promising direction for research.

Our middleware choice was, and continues to be, driven by a combination of our design pillars and our clinical requirements. The pillars and clinical requirements drive the technical requirements that the middleware must meet for a particular system instantiation. We validate that a middleware is appropriate for use in building interoperable MCPS such as an ICE implementation by evaluating it against the design pillars and its capability to support the needs documented in the clinical requirements. By documenting and assessing user needs we gain assurance that the system we build will be suitable for use in its intended environment.

Our approach is to allow clinical focus groups [10] to suggest clinical scenarios, which are captured either in person or through our prototype clinical scenario repository [1]. These scenarios then suggest clinical requirements, such as the samples shown in Figure 2. These clinical requirements imply technical requirements which are implemented to build a concrete system such as OpenICE. We use the technical requirements to verify the implementation, the clinical requirements and design pillars to validate the implementation, document gaps, and iterate. This waterfall development style description is overly linear, and it's important to realize that design and implementation are likely to iterate rapidly.

Clinical scenarios may document a situation where patient outcomes or safety could be improved by the use of interoperable devices. It is vital that the set of scenarios also include situations where a technical integration failure or lack of interoperability leads to patient harm, as well as situations where interoperability leads to new hazardous situations. Scenarios can be reflect an actual or imagined sequence of events that happened, or they can be constructed from an imaginable sequence of events derived from what policies and guidelines exist to prevent. In this work, we concentrate on the use of clinical requirements and their influence on middleware selection, rather than the process of moving from clinical scenarios to clinical requirements or from clinical to technical requirements.

The clinical requirements primarily represent the interactions of the system, including

SCR1: The ICE system shall be aware of the required frequency / accuracy / reliability of the incoming data for each parameter based on clinical significance, and shall choose the closest available frequency / accuracy / reliability on the device and provide this information to the clinician for review.

SCR2: If the device connected to the ICE system is not capable of providing the required frequency / accuracy / reliability of the incoming data for each parameter based on clinical significance, the ICE system shall choose the closest available frequency / accuracy / reliability on the device and provide this information to the clinician for review.

SCR3: The ICE System shall notify users when it loses connectivity with any of its components.

Figure 2 Sample Clinical Requirements.

constituent devices, with users including clinicians and the patient. Our clinical requirements have come from elicitation sessions, clinicians, hospital policies, existing documentation, ASTM F2761 Annex B, clinical care guidelines, nursing documentation, clinical specialists, incident reports, and other groups. Figure 2 contains a selection of clinical requirements that have direct implications for middleware selection. For instance, consider SCR 3 "The ICE System shall notify users when it loses connectivity with any of its components." These clinical requirements are written from the perspective of the clinical user, who may have little or no knowledge of how the system works; they are a form of black box requirements. This requirement could be implemented in a wide variety of ways. There are no requirements stated for timing, for how the notification should happen, or for which component should do the notification. Such specializations of the requirement follow from specific use cases and specific implementations. The specialization of SCR 3 will be very different for an ICE implementation intended to run only an application that sends data to an offline documentation archive versus an implementation intended to support running an application controlling a closed-loop infusion of a fast-acting drug. SCR 1 and SCR 2 may also raise the eyebrows of those experienced in real-time systems. The closest possible match may not be a very good match at all, which is why review is required, and any deviation may throw off carefully engineered timings. It is important to remember that these requirements capture clinical needs as voiced by clinicians. They are not technical engineering requirements, and they are subject to interpretation and change in building implementations. Validation that a given implementation satisfies the clinical requirement is inherently subjective. It is our intention in compiling these that they be relatively unambiguous and reflect clinical consensus. The clinical requirements shown in the examples are generic in the sense that they are meant to apply to all ICE systems.

5 Conclusion

We have presented an approach to validating middleware selection for MCPS using user needs as documented in design pillars and clinical requirements. We are using this approach in our development of our OpenICE implementation, and we believe that the user needs we document here will be useful for others who are working on medical device interoperability. Our design pillars are intended to support making interoperability a community activity. We want to be able to share interface code, test suites, and requirements with the whole interoperability movement, not just ideas.

Our goal is to improve patient outcomes and safety through interoperability. To achieve this, it is not enough to build a technically perfect system, even one that satisfies all of the clinical requirements. Our design pillars document the non-functional requirements and design goals that we believe are necessary to build a successful interoperable medical device system.

Acknowledgements. Our team at the MD PnP Program has had the pleasure of working with many generous and brilliant collaborators. We would like to thank Sandy Weininger and Yi Zhang at FDA CDRH and our collaborators on the Quantum Medical Device Interoperability project, namely Insup Lee, Oleg Sokolsky, Andrew King and the rest of the PRECISE team at the University of Pennsylvania, John Hatcliff and the members of the Medical Device Coordination Framework project at Kansas State, Lui Sha and his team at UIUC, Tracy Rausch, Wayne Saari and the DocBox crew, Dick Moberg, and Mike Robkin and all of Anakena Solutions. We've all spent many hours hotly debating issues around medical device interoperability. This paper represents some of our perspective, which has been greatly improved by our collaborators input. Oversimplifications and omissions are our own. We're looking forward to the next round of discussion.

References

1 Diego Alonso, Jeff Plourde, Sandy Weininger, and Julian M. Goldman. Web-based clinical scenario repository (CSR). In *Poster Presentation at the Society for Technology in Anesthesia Annual Meeting*, 2014.

2 Rajeev Alur, David E. Arney, Elsa L. Gunter, Insup Lee, Jaime Lee, Wonhong Nam, Frederick Pearce, Stephen Van Albert, and Jiaxiang Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (cara) infusion pump control system. *Software tools for technology transfer*, 5(4):308–319, 2004.

3 David E. Arney, Sebastian Fischmeister, Julian M. Goldman, Insup Lee, and Robert Trausmuth. Plug-and-play for medical devices: Experiences from a case study. *Biomedical Instrumentation & Technology*, 43(4):313–317, July 2009.

4 David E. Arney, Julian M. Goldman, Insup Lee, Ersel Llukacej, and Susan F. Whitehead. Use case demonstration: X-ray/ventilator. In *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007*, page 160, June 2007.

5 David E. Arney, Raoul Jetley, Paul Jones, Insup Lee, and Oleg Sokolsky. Formal methods based development of a PCA infusion pump reference model: Generic Infusion Pump (GIP) project. In *HCMDSS-MDPNP'07: Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, pages 23–33, Washington, DC, USA, 2007. IEEE Computer Society.

6 David E. Arney, Miroslav Pajic, Julian Goldman, Insup Lee, Rahul Mangharam, and Oleg Sokolsky. Toward patient safety in closed-loop medical device systems. In *Cyber-Physical Systems (ICCPS'10)*, April 2010.

7 ASTM F2761-09(2013). 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. `http://www.astm.org/Standards/F2761.htm`.

8 David E.Arney, Julian M. Goldman, Susan F. Whitehead, and Insup Lee. Synchronizing an x-ray and anesthesia machine ventilator: A medical device interoperability case study. In *BIODEVICES'09*, pages 52–60, January 2009.

9 Julian M. Goldman, Mike Jaffe, Dave Osborn, and Sandy Weininger. The Integrated Clinical Environment (ICE) Standard (ASTM F2761-09) – The First Ten Years. In *Poster Presentation at the Society for Technology in Anesthesia Annual Meeting*, 2014.

10 Julian M. Goldman, Susan F. Whitehead, and Sandy Weininger. Eliciting clinical requirements for the medical device plug-and-play (MD PnP) interoperability program. In *Anesthesia & Analgesia: Abstracts of Posters Presented at the International Anesthesia Research Society 80th Clinical and Scientific Congress*, March 2006.

11 Object Management Group. Data distribution service (DDS). `http://portals.omg.org/dds/`, March 2014.

12 Andrew King, David Arney, Insup Lee, Oleg Sokolsky, John Hatcliff, and Sam Procter. Prototyping closed loop physiologic control with the medical device coordination framework. In *2nd Workshop on Software Engineering in Health Care (SEHC'10)*, May 2010.

13 MDPnP Interoperability Program. OpenICE software repository. `http://mdpnp.org/MD_PnP_Program___OpenICE.html`, March 2014.

14 Carl F. Wallroth, Julian M. Goldman, Jurgen Manigel, Dave Osborn, T Roellike, Sandy Weininger, and Dwayne Westenskow. Development of a standard for physiologic closed loop controllers in medical devices. In *Poster Presentation at the World Congress of Anesthesiology*, 2008.

OR.NET – Approaches for Risk Analysis and Measures of Dynamically Interconnected Medical Devices

Franziska Kühn[1,2], Martin Leucker[1], and Alexander Mildner[3]

1 **Institute for Software Engineering and Programming Languages, University of Lübeck, Germany**
`{kuehn,leucker}@isp.uni-luebeck.de`
2 **Graduate School for Computing in Medicine and Life Science, University of Lübeck, Germany**
3 **UniTransferKlinik Lübeck, Germany**
`a.mildner@unitransferklinik.de`

Abstract

Nowadays, it lacks an open, standardized and dynamic interconnection of medical devices. All existing combinations of medical devices consist of isolated solutions with proprietary interfaces, as no common standards for networking and the exchange of data of medical devices exist. This situation leads to confusing operating rooms and inefficient operations. Thus, new strategies need to be developed for the authorization of dynamically interconnected medical devices. Primarily, those concern of an acquisition and methodical adaption of new requirements and risks resulting from this way of interconnection. The approach is to develop a method for a risk analysis for interconnected medical devices, which is structured modular and consists of a risk assessment of the standalone device and a risk analysis for the interconnection considering the risks involved in the transfer of functions. When interconnecting the medical devices the risk analysis of each of the devices is taken and they are compared by a gap analysis. Through this strategy it will be possible to realize a standard-compliant dynamic interconnection of medical products, which would be advantageous both for clinic operators and producers. This paper presents the current situation of the authorization of combined medical devices and proposes a strategy for the risk management of dynamically interconnected medical devices as a substantial part of the authorization.

1998 ACM Subject Classification J.3.3 Medical information systems

Keywords and phrases Modular Risk Analysis, Medical Device Interconnection, OR.NET, Development Method

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.133

1 Introduction

At present, the dynamic interconnection of medical devices poses as well legal as technical challenges, especially for clinic operators. On the one hand most medical devices the clinic operators would like to interconnect are not interoperable, on the other hand the connection of medical devices (which have not been authorized together) leads to a self-production by the clinic operator [3]. The clinic operators can choose all-in-one operating room solutions from certain manufacturers, but often expensive and elaborate custom integration projects are necessary. All of the possible combinations today consist of isolated solutions with proprietary interfaces and have a limited flexibility and interchangeability, as no common

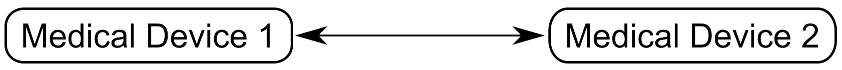


Figure 1 Risk analysis and conformity statement today.

standards for networking and the exchange of data of medical devices to each other and to adjacent IT-systems exist. This situation leads to confusing operating rooms and inefficient operations.

The OR.NET-project[1], funded by the federal ministry of education and research (BMBF), focuses on the safe, secure and dynamic interconnection of medical devices in the operating room and clinic. The project includes almost 50 project partners ranging over medical device manufacturers, clinic operators, standardisation organisations and research institutes. Besides the safe and secure interconnection of medical devices, the main goals are a standardised solution for the interoperability of all medical devices and the possibility of interconnecting arbitrary medical devices for clinic operators without taking responsibility for the resulting system.

This paper shows the authorization of combined medical devices nowadays, which is illustrated in Section 2. The content of Section 3 deals with our general idea of the authorization of dynamically interconnected medical devices. The strategy for the risk management of such systems as a substantial part of the authorization is approached in Section 4. Conclusively the paper discusses the advantages of a dynamic interconnection of medical devices.

2 Today's Situation of Authorisation

Figure 1 shows the idea of the procedure for obtaining an authorization of today's systems and combinations.

The procedure is generally based on the assumption that the whole system is known and the individual components have been developed for an interaction of each other. Either the person establishing the interconnection declares conformity and is responsible for the entire system or both producers of the medical devices allow the application with the other, precisely specified device and document this option in the intended use of their own medical device [4]. Approving all combinations of interest in advance is not possible because of the enormous number of possible combinations. Another problem is that once a component is replaced, a new conformity assessment procedure of the entire system must be worked through. These obstacles for a dynamic interconnection of medical devices, especially from different producers, lead to a large amount of not connected medical devices and thereby to a confusing amount of control elements and monitors in the operating room as shown in Figure 2.

3 Futures Authorization

A dynamic, open interconnection of medical devices with their hardware and software components is not yet implemented in today's authorisation processes. Thus, new strategies need to be developed for the authorization of such systems. Primarily, those concern of an acquisition and methodological adaption of new requirements and risks resulting from

[1] `http://www.ornet.org`

Figure 2 Operating room today.

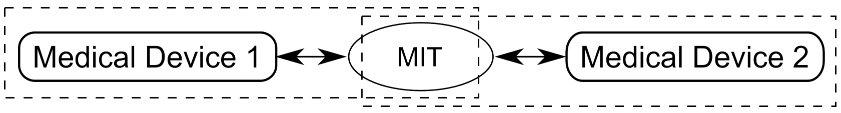


Figure 3 Authorization of dynamically interconnected systems.

this way of interconnection [2]. Figure 3 shows the idea of the procedure for obtaining an authorization of futures, dynamically interconnected medical device combinations. The devices are authorized without knowledge of each other but with a defined interface. They are connected by a safe medical IT-network (MIT), constructed and controled according to EN 80001 [1].

4 Approaches for Risk Analysis and Measures

The aim is to develop a method to consider risks of a dynamic interconnection without defining a specified connection partner. This method must be able to assess, evaluate, control and document the risks additionally occurring by an interconnection. The structure of the risk management is shown in Figure 4.

Risk analysis, -evaluation and -control is to be made separately for every medical device in the combination. The producers do not longer allow the interconnection with only one other, precisely specified medical product, but comply with a precisely defined interface. For this reason it is necessary to precisely define the interfaces of the medical devices and certify the respective medical devices including the interface specifications. The risk analysis of each device is structured modular and consists of a risk assessment of the standalone device (in accordance with today's practices) and a risk analysis for the interconnection. The risk analysis of the interconnection is developed by considering the risks involved in the transfer of functions of the medical device to another one or the takeover of functions of another medical device. These risks are transferred to a Failure Mode and Effects Analysis (FMEA) [5] and risk control measures are applied. The clinic operator is only responsible for a safe and reliable network conforming to EN IEC 80001 [1] and for meeting the producers demands on the network e.g. like a minimum bit rate and, if required, implement risk measures for the interonnection defined in the FMEA of a certain medical device. When interconnecting the

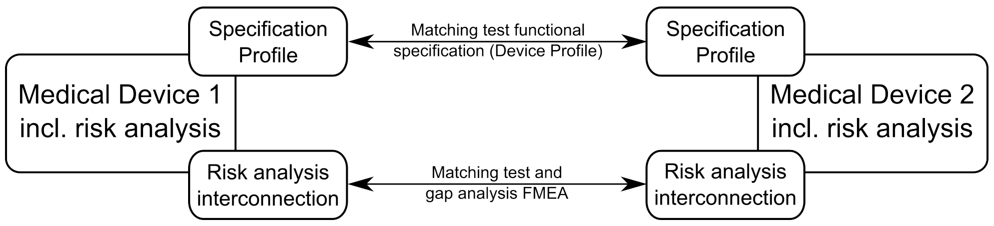

Specification Profile
Matching test functional specification (Device Profile)
Specification Profile
Medical Device 1 incl. risk analysis
Medical Device 2 incl. risk analysis
Risk analysis interconnection
Matching test and gap analysis FMEA
Risk analysis interconnection

Figure 4 Risk management for dynamic interconnections.

medical devices the FMEAs of each of the devices are taken and they are compared by a gap analysis. In that gap analysis the risks considered in the FMEAs for the interconnection are compared by the clinic operator. If there are risks considered only in one of the FMEAs it has to be checked if those mean additional risks also for the interconnected device. Only if this is the case additional risk measures have to be taken to enable a safe and secure interconnection.

5 Discussion

It lacks an open, standardized and dynamic interconnection of medical devices. This interconnection would be advantageous both for clinic operators and producers. The clinic operators could put together their ideal device combinations that would support their operation flow best. An expected simplification of work processes would also lead to monetary savings. In large medical technology companies the loss of proprietary interfaces would lead to savings potentials, enabled by a simplified authorization and less needed expert know-how. For small and medium businesses a standardized interconnection would open up new business areas and they could have a better chance in the market if their devices could interact with those of large producers. Simplified procedures and a reduced number of control elements would relieve the operating room staff and increase patient safety. Merging data sets from multiple devices would increase the quality of diagnoses and reduce the number of required monitors.

References

1 Janko Ahlbrandt, Johannes Dehm, Rainer Röhrig, Christian Wrede, and Michael Imhoff. Gemeinsames Positionspapier zur IEC 80001-1 – Risikomanagement für medizinische Netzwerke in der Intensiv- und Notfallmedizin. Technical report, DIVI, DGMBT, VDE, 2012.
2 International Electrotechnical Commission. EN IEC 60601-1:2006 – Medical electrical equipment – Part 1: General requirements for basic safety and essential performance, 2006.
3 Armin Gärtner. *Medizinproduktesicherheit Band 5: Medizinische Netzwerke und Software als Medizinprodukt.* TÜV Media GmbH, Cologne, Germany, 2010.
4 Council of the european communities. Council directive 93/42/eec of 14 june 1993 concerning medical devices (amended by directive 2007/47/ec of 5 september 2007, June 1993.
5 Dieter H. Müller Thorsten Tietjen, André Decker. *FMEA-Praxis – Das Komplettpaket für Training und Anwendung.* Carl Hanser Verlag, München, Germany, 2011.

Automated Verification of Quantitative Properties of Cardiac Pacemaker Software*

Marta Kwiatkowska and Alexandru Mereacre

Department of Computer Science, University of Oxford, UK
`{marta.kwiatkowska,mereacre}@cs.ox.ac.uk`

Abstract

This poster paper reports on a model-based framework for software quality assurance for cardiac pacemakers developed in Simulink and described in [3]. A novel hybrid heart model is proposed that is suitable for quantitative verification of pacemakers. The heart model is formulated at the level of cardiac cells, can be adapted to patient data, and incorporates stochasticity. We validate the model by demonstrating that its composition with a pacemaker model can be used to check safety properties by means of approximate probabilistic verification.

1998 ACM Subject Classification I.6.4 Model Validation and Analysis, J.3 Life and Medical Sciences

Keywords and phrases Pacemakers, Verification, Simulink

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.137

1 Introduction

Today's implantable medical devices are increasingly often controlled by embedded software and rigorous software design methodologies are needed to ensure their safe operation and to avoid costly device recalls. The natural models for medical devices, such as cardiac pacemakers [6], GPCA infusion pumps [8] and continuous glucose monitors [11], are stochastic hybrid systems: they involve discrete mode switching and nonlinear continuous flows, e.g., electrical signal or glucose level, while at the same time allowing for stochasticity that arises from randomness of the timing of events. Therefore, developing effective methodologies to provide safety assurance in this setting by means of *quantitative verification* is an important challenge.

Regarding cardiac pacemakers, a number of model-based frameworks have been proposed, to mention the Virtual Heart Model (VHM) of Jiang *et al.* [5, 7]. Though mainly intended for simulation and testing, its timed automata pacemaker model [6] has been verified using UPPAAL [10] against a *random heart model*. The random heart model can capture the timing delays between events, but is unable to model the stochasticity in the timing that is characteristic in a heart rhythm and varies from patient to patient. Following a suggestion in [6] that physiologically-relevant heart models are needed to establish the correctness of more complex properties for pacemakers, we earlier developed a realistic heart model that addresses this issue [1]. The model was adapted from a sophisticated model that generates multi-channel electrocardiogram (ECG) based on nonlinear ordinary differential equations (ODEs) due to Clifford *et al.* [4]. To transfer to our setting, where we need to consider that the pacemaker is implanted in the heart tissue, we convert external ECG signals into

* This work is supported by the ERC Advanced Grant VERIWARE, VERIPACE project and Oxford Martin School Institute for the Future of Computing.

action potential signals read by implanted sensors. A unique feature of the model of [1] is that the heart can probabilistically switch between normal and abnormal beat types, in a manner that can be learnt from patient data. We performed quantitative, probabilistic verification by analysing the composition of the pacemaker model of [6] and the heart model against typical correctness properties such as (i) whether the pacemaker corrects faulty heart beats, maintaining normal heart rhythm of 60-100 beats per minute (BPM), and (ii) that the pacemaker does not induce erroneous heart behaviours (that is, it does not overpace the heart unless necessary). These were implemented based on the probabilistic model checker PRISM [9] and MATLAB.

One of the shortcomings of the heart model in [1] is that it does not capture the electrical conduction system of the heart, and specifically the delays in the action potential signal as it is propagated from cell to cell.

This paper reports on a model-based framework for software quality assurance for cardiac pacemakers developed in Simulink and described in [3]. We instantiate the framework with a physiologically-relevant heart model built as a network of cardiac cells. The heart model is inspired by that of [7], except that we represent it as a network of input-output hybrid automata, instead of timed automata, and enhance it with stochasticity. The model enables the modelling of both diseased and normal rhythms, and can be adapted to exhibit random delays in the timing of events that are patient-specific. We implement the heart model in Simulink and validate it against the pacemaker models of [7], demonstrating basic safety properties of the pacemaker by means of probabilistic approximate model checking, with encouraging results. We also provide experimental results for advanced properties, including pacemaker mediated tachycardia correction and detailed analysis of energy consumption. This paper extends the results reported in [2]. The outcome of the research is a comprehensive model-based framework based on Simulink suitable for simulation, as well as quantitative verification, of pacemakers. The interested reader is referred to [3] for more detail.

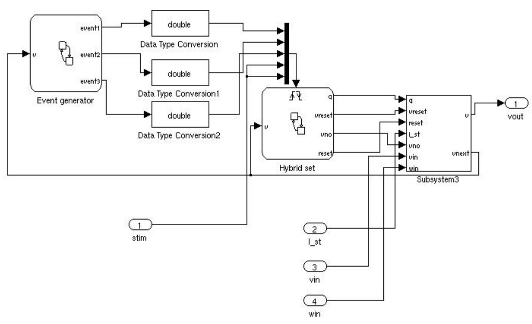
2 Quantitative Verification

We implement both the heart model and the pacemaker model in Simulink.

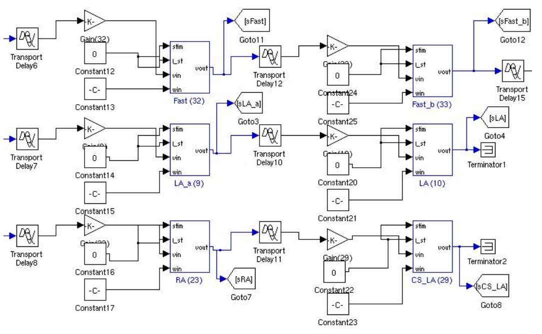
Fig. 1a shows the Simulink implementation of a cardiac cell. The cell is implemented by means of three Simulink blocks: *Event generator*, *Hybrid set* and *Subsystem*. The *Event generator* block is responsible to generate the input events to the cell. The *Hybrid set* implements the cell hybrid automaton model. The *Subsystem* block performs the integration procedure to compute the voltage level of the cell. Fig. 1b shows a network of six cells. Each cell block is composed from the three sub-blocks shown in Fig. 1a and connected to other cells through delay and gain components.

We run a set of experiments on a faulty heart to verify that the pacemaker restores the normal heart beat. In Fig. 2a we depict two signals. The first one (in blue) denotes the action potential generated by the *natural pacemaker cell* of the heart situated in the atrium. More precisely, we have three beats in six seconds, which is approximately 30 beats per minute. The number of heart beats is thus too slow and needs the intervention of the pacemaker. The second signal (in red) denotes the action potential from one of the cardiac cells situated in the ventricle. This is the signal which is captured and paced by the pacemaker.

The second set of experiments depicts the relation between different programmable parameters of the pacemaker and the battery charge. In Fig. 2b we depict the battery charge in a period of 1 min when the programmable parameters TAVI and TURI of the pacemaker are varied.

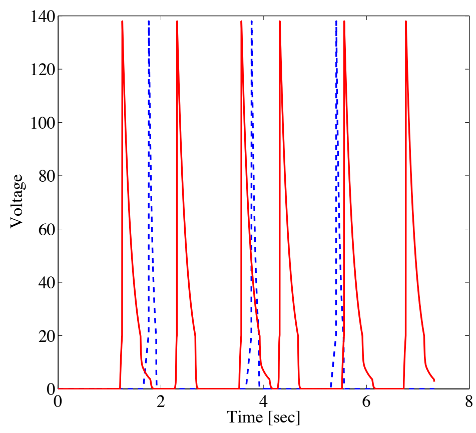


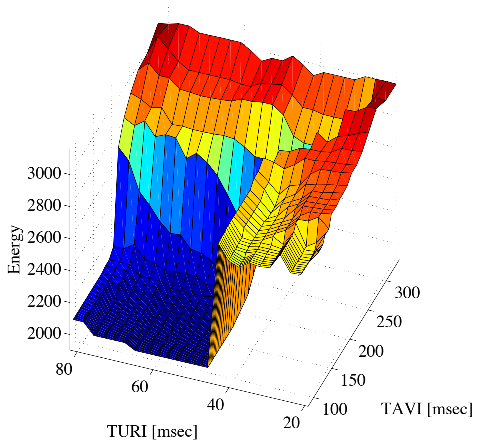
(a) Cell block



(b) Cell connection

Figure 1 Cardiac cell model.





t]

(a) Heart beat correction experiments **(b)** Battery charge

Figure 2 Experiments.

References

1 T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. *RTSS*, pp. 263–272. IEEE, 2012.

2 T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. A Simulink Hybrid Heart Model for Quantitative Verification of Cardiac Pacemakers. *HSCC*, pp. 131–136. IEEE, 2013.

3 T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers over Hybrid Heart Models. *Information and Computation*, Elsevier, 2013.

4 G. Clifford, S. Nemati, and R. Sameni. An Artificial Vector Model for Generating Abnormal Electrocardiographic Rhythms. *Physiological Measurements*, 31(5):595–609, May 2010.

5 Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam. Real-Time Heart model for implantable cardiac device validation and verification. *ECRTS*, pp. 239–248. IEEE, 2010.

6 Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and Verification of a Dual Chamber Implantable Pacemaker. *TACAS*, pp. 188–203, 2012.

7 Z. Jiang, M. Pajic, and R. Mangharam. Cyber-Physical Modeling of Implantable Cardiac Medical Devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.

8 B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. L. Jones, Y. Zhang, and R. P. Jetley. Safety-assured development of the GPCA infusion pump software. *EMSOFT'11*, pp. 155–164. ACM, 2011.

9 M. Kwiatkowska, G. Norman and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems *CAV'11*, pp. 585–591, 2011.

10 K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell, 1997.

11 S. Sankaranarayanan and G. E. Fainekos. Simulating Insulin Infusion Pump Risks by In-Silico Modeling of the Insulin-Glucose Regulatory System. *CMSB*, pp. 322–341, 2012.

Potential Advantages of Applying Assurance Case Modeling to Requirements Engineering for Interoperable Medical Device Systems*

Rick Schrenker[1], Jeff Plourde[1], Diego Alonso[1], David Arney[1], and Julian M. Goldman[1,2]

1 **MD PnP Program, Massachusetts General Hospital, Boston, MA, USA**
`{raschrenker,jplourde1,dalonso,darney}@mgh.harvard.edu`

2 **Harvard Medical School, Cambridge, MA, USA**
`jmgoldman@mgh.harvard.edu`

Abstract

This poster describes our initial work in applying assurance cases to the requirements engineering processes necessary in building interoperable medical device systems.

1998 ACM Subject Classification D.2.9 Management, K.6.3 Software Management, K.6.4 System Management

Keywords and phrases Assurance Case, Goal Structured Notation, Requirements Engineering, Interoperability, Medical Device

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.141

Overview

Habli and others have described the application of assurance case methods to support goal-oriented requirements engineering [3]. Atwood describes the application of goal-structuring notation (GSN) to make visible the domain knowledge and assumptions that support the validity of a specific requirement, noting the value of making this available for subsequent analysis on a requirement change or reuse across a system family [2].

Among the objectives of the CIMIT MD PnP Program is to develop one or more instances of systems that, using the same family of systems components and interfaces, satisfies the functional and non-functional needs of four distinct clinical scenarios [4] according to the criteria set forth in ASTM 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 [1].

Our challenges include, on identifying requirements that at the clinical use level appear to span more than one use case, to assure that when the requirement and its derived specifications continue to satisfy the high level need throughout decomposition into technical requirements and specifications. Similarly, on changes to a requirement, the impact of the change needs to be assessed across the set of clinical scenarios to which it contributes.

Our plan to apply GSN covers the following. We include examples of traceable documentation and requirements decomposition and discussion of all points:

* This publication was made possible by grant number 1U01EB012470 from NIH/NIBIB and award number W81XWH-12-C-0154 from Department of Defense US Army Medical Research and Materiel Command. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH/NIBIB or the US Army or the Department of Defense.

- Documentation of traceability from clinical scenario through validation.
- Decomposition of requirements (goals) across key functional and non-functional properties (sub-goals).
- Determination whether sharing of the above with stakeholders spanning the clinical, technical, and management domains enable clearer communication, e.g., to support decision making.
- Analysis of how well the tools we are evaluating in the MD PnP Program cover the breadth of requirements posed by the full set of clinical scenarios.
- Capture and analysis of the cost of application of GSN to our work.

We are also in the process of implementing a requirements management database system and are considering mapping between the GSN and database model to identify potential added value in the combination, e.g. improvement to requirements conflict identification processes.

References

1 ASTM F2761-09(2013). 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. `http://www.astm.org/Standards/F2761.htm`.

2 Katrina Attwood, Tim Kelly, and John Mcdermid. The use of satisfaction arguments for traceability in requirements reuse for system families: Position paper. In *University of Madrid, Madrid Spain*, pages 18–21, 2004.

3 Ibrahim Habli, Weihang Wu, Katrina Attwood, and Tim Kelly. Extending argumentation to goal-oriented requirements engineering. In *Proceedings of the 2007 Conference on Advances in Conceptual Modeling: Foundations and Applications*, ER'07, pages 306–316, Berlin, Heidelberg, 2007. Springer-Verlag.

4 MD PnP Interoperability Program. QMDI clinical scenarios. `http://www.mdpnp.org/MD_PnP_Program___Clinical_S.html`, March 2014.

Process-Oriented Analysis for Medical Devices

Vasiliki Sfyrla[1], Josep Carmona[2], and Pascal Henck[3]

1 **Viseo R&D, 4 Avenue Doyen Louis Weil, 38000 Grenoble France**
`vsfyrla@objetdirect.com`

2 **Universitat Politècnica de Catalunya, Calle Jordi Girona, 31, 08034 Barcelona, Spain**
`jcarmona@lsi.upc.edu`

3 **Fresenius VIAL, Le Grand Chemin 38590 Brézins, France**
`pascal.henck@fresenius-kabi.com`

Abstract

Medical Cyber Physical Systems are widely used in modern healthcare environments. Such systems are considered life-critical due to the severity of consequences that faults may cause. Effective methods, techniques and tools for modeling and analyzing medical critical systems are of major importance for ensuring system reliability and patient safety.

This work is looking at issues concerning different types of medical industry needs including safety analysis, testing, conformance checking, performance analysis and optimization. We explore the possibility of addressing these issues by exploiting information recorded in logs generated by medical devices during execution. Process-oriented analysis of logs is known as process mining, a novel field that has gained considerable interest in several contexts in the last decade.

Process mining techniques will be applied to an industrial use case provided by Fresenius, a manufacturer of medical devices, for analyzing process logs generated by an infusion pump.

1998 ACM Subject Classification B.3.3 Performance Analysis and Design Aids, B.5.3 Reliability and Testing, D.2.2 Design Tools and Techniques, I.2.6 Learning, I.5 Pattern Recognition

Keywords and phrases Process Logs, Process Mining, Discovery, Formal Analysis, Infusion Pump

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.143

1 Introduction

Medical Cyber physical systems couple tightly the cyber and physical parts to provide mission-critical services in clinical environments [6]. They can be found embedded in a wide range of medical devices from small size, including infusion pumps and pacemakers, to large equipment such as x-rays. Medical devices are widely used in modern healthcare environments improving effectiveness of the patient care. Nevertheless, medical devices enormous complexity may affect the quality and the reliability of the overall system, putting patients at risk.

Much consideration is being given to rigorous system design and analysis methods for improving the trustworthiness of medical devices and verifying safety properties. Ensuring safety and reliability has been studied for various medical devices including the pacemaker [5] and the PCA Infusion Pump [11]. In the latter case, model-driven engineering is proposed for establishing a safety assured implementation. The generic PCA reference model is translated from Simulink to Timed Automata and its safety requirements described in temporal logic formulae. Using the UPPAAL tool [4], requirements are verified for satisfiability.

In the medical devices industry, design tools such as Matlab/Simulink and Unified Modeling Language(UML) are used for modeling the embedded system. However, in spite of several contributions on defining semi-formal semantics for those specifications, they are still not considered formal languages. Formal analysis need to be conducted on unambiguous specification, thus transforming the underlying specification into a formal model is imperative. Depending on the complexity of the system, model transformation can be a time consuming, tedious and error-prone task.

This work proposes an alternative approach of analyzing and ensuring safety properties of medical devices. Based on the process logs generated from the execution of a device, it is possible to extract process related information e.g. process model or reconstruct the system model in a formal representation. This methodology also known as process mining [8], offers a wide range of analysis that can be performed on the generated model. It does not only provide the means to increase safety and reliability of a system, it also opens up new possibilities of performance analysis and optimization.

These aposteriori formal reasoning can complement existing analysis performed at the model at design level such as model checking. Extracting a formal model from generated logs, for already designed and implemented systems, may accelerate the desired analysis and be more time efficient.

The remainder of this paper, presents the basic idea of process mining and the benefits of applying process mining techniques for analyzing medical devices.

2 Process Mining

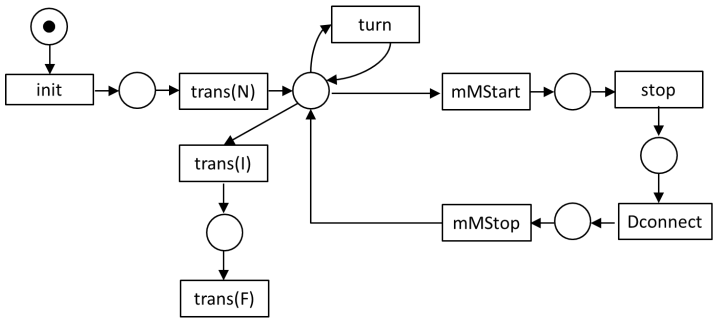
Process Mining [1] is a model-driven approach aiming at constructing process models based on available process logs stored by information systems. A process model is a formal representation of a process. It may represent several process information, like activity executions, actors, roles, resources, time, data, among others. Starting from a process log that represents the footprints left by executing a process, process mining techniques may derive a formal process model that represents the underlying process. For instance, a process model may focus on the control-flow perspective, i.e., the order of execution of activities within a process execution. Process models can be described in different process modeling languages such as BPMN, Petri Nets or Event Driven Process Chain.

A fundamental goal of process mining is the discovery and extraction of the model that describes the process underlying in the process log. The reconstruction of the model is the aim itself but process mining it is not limited to that. Conformance checking can be performed to compare the derived model with the process log and monitor possible deviations. Notice that while in discovery the only algorithm's input is the log, in conformance two inputs are considered: the log and the process model, the latter can be either obtained by a discovery technique or manually designed from an expert. Analysis might aim also at additional objectives including optimization of the process, testing for satisfiability of safety properties, performance analysis etc.

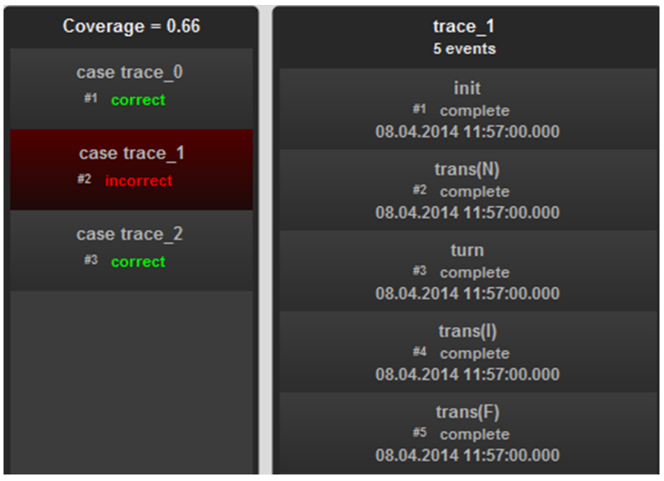
In contrast to data mining, process mining techniques focus on the process perspective, and hence the causal relations between different events of a process are identified. Process mining is applicable to systems (e.g. ERP systems or embedded systems) that record their behavior and produce process logs. Current research of process mining in the health-care domain is in early stages. In [9], process mining was used to obtain insights related to careflows for gynecological oncology patients. In [2] datasets of stroke patients have been analyzed to demonstrate how to construct models for a whole data set or for only aspects

Trace ID	Activity
0	init
	trans(N)
	turn
	mMStart
	stop
	Dconnect
	mMStop
	trans(I)
	trans(F)
1	init
	trans(N)
	turn
	trans(I)
	trans(F)
2	init
	trans(N)
	trans(I)
	trans(F)
...	...

(a)



(b)


Coverage = 0.66

case trace_0
#1 correct

case trace_1
#2 incorrect

case trace_2
#3 correct

trace_1
5 events

init
#1 complete
08.04.2014 11:57:00.000

trans(N)
#2 complete
08.04.2014 11:57:00.000

turn
#3 complete
08.04.2014 11:57:00.000

trans(I)
#4 complete
08.04.2014 11:57:00.000

trans(F)
#5 complete
08.04.2014 11:57:00.000

(c)

Figure 1 (a) Generated traces from an Infusion Pump, (b) Discovered Model, (c) Verifying property.

that are of interest for identifying for instance differences in treatment strategies between different hospitals. Finally, the work [3] presents a framework to support process mining in critical care that enables improvement for clinical guideline.

However, process mining in health care has not yet been considered within the context of medical devices and towards the analysis of embedded software. In this context, process mining techniques can be exploited for performing formal analysis and focusing on other aspects such as performance and optimization.

To give an impression of how process mining could be applied to the analysis of medical software, an example is illustrated in Figure 1. Figure 1(a) shows hypothetical logs generated by the execution of an infusion pump software. Several activities are performed at each trace 0, 1 and 2. The Petri Net in Figure 1(b) is discovered using the ILP algorithm of ProM tool [12]. ProM is a platform independent open source framework which supports a wide variety of data and process mining techniques in form of plug-ins. Using the LTL checker plug-in, the property *always_when_turn_then_eventually_stop* is verified. Figure 1(c) indicates the trace that violates the given property. This is due to the fact that action turn is not followed by stop.

3 Conclusions

We presented Process Mining, an approach for performing analysis using event logs generated by medical devices. Using process mining techniques, we can exploit the information recorded in the event logs.

We explore the possibility of conducting log-based verification [10] in order to prove satisfiability of safety properties. Applying log-based verification to already designed systems could be a faster way than model checking to prove properties. Manufacturers, such as Fresenius [7], could be particularly interested by such an approach. The main reason is that analysis can be performed without the effort of transforming existing models into the formal description compatible with a model checker.

However, there are many challenges associated with the approach of log-based verification. What are the interesting information to log and how to distinguish between good and bad events taking into account that resources in medical systems are limited. How many scenarios are needed to create logs and to what extent these logs cover the integrity of the model.

Future plans include working on the analysis of real logs generated by the execution of the Fresenius infusion pump. This work aims at extracting results that can complement the analysis of the model at design level, currently under investigation [13].

Acknowledgements. The authors would like to thank prof. I. Lee, prof. O. Sokolsky, prof. K. K. Venkatasubramanian and prof. E. Vasserman for their invaluable insights.

References

1 W. van der Aalst. *Process Mining, Discovery, Conformance and Enhancement of Business Processes*. Springer 2011

2 R. Mans, H. Schonenberg, G. Leonardi, S. Panzarasa, A. Cavallini, S. Quaglini, and W. van der Aalst. *Process mining techniques: an application to stroke care.*.

3 C. McGregor, C. Catley, and A. James. *A Process Mining Driven Framework for Clinical Guideline Improvement in Critical Care*. 2011

4 The UPPAAL tool: `http://www.uppaal.org/`

5 C. Li, A. Raghunathan, and N. K. Jha, *Improving the Trustworthiness of Medical Device Software with Formal Verification Methods*. Embedded Systems Letters, IEEE 2013.

6 A. Banerjee, K. K. Venkatasubramanian, T. Mukherjee, et al. *Ensuring Safety, Security and Sustainability of Mission-Critical Cyber Physical Systems*. 2011

7 Fresenius Healthcare Company: `http://www.fresenius-kabi.com/`

8 Process Mining Manifesto: `http://www.win.tue.nl/ieeetfpm/`

9 R. S. Mans et al. *Application of process mining in healthcare–a case study in a dutch hospital*. Biomedical Engineering Systems and Technologies. Springer 2009.

10 W. M. P. Aalst, H. T. Beer, and B. F. Dongen. *Process Mining and Verification of Properties: An Approach Based on Temporal Logic*. On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE. Springer

11 B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. Jones, Y. Zhang, and R. Jetley. *The safety-assured development of the GPCA infusion pump*. 2011

12 The ProM framework.`http://www.promtools.org/prom6/`

13 V. Sfyrla, S. Marcoux, and C. Vittoria *Formal Analysis of Fresenius Infusion Pump (FIP)*. 2013