Rigorous Modeling and Analysis of Interoperable Medical Devices

Atif Mashkoor
Software Competence Center Hagenberg GmbH
Hagenberg, Austria
atif.mashkoor@scch.at

Johannes Sametinger
Johannes Kepler University
Linz, Austria
johannes.sametinger@jku.at

ABSTRACT
Medical Devices (MDs) are by definition safety-critical and increasingly also become security-critical when interoperating, i.e., when communicating in some form. Finding errors, inconsistencies, or vulnerabilities in MDs before deployment can significantly decrease costs, and increase quality and reliability. In this paper, we present a rigorous "correct-by-construction" approach for modeling and analyzing interoperating MDs by considering various abstraction levels, i.e., the functional, the safety, and the security level. The approach is illustrated using sample requirements of a hemodialysis device.

Author Keywords
Medical devices; modeling; analysis; Event-B; verification and validation; safety; security; hemodialysis.

ACM Classification Keywords
D.2.4 SOFTWARE ENGINEERING: Software/Program Verification: Model checking; I.6.4 SIMULATION AND MODELING: Model Validation and Analysis; J.3 LIFE AND MEDICAL SCIENCES: Medical Information Systems

1. INTRODUCTION

Modern medical devices (MDs) come with software and also provide some form of interoperability. For example, devices increasingly communicate health information. This communication is often wireless, but it is also done via the Internet. MDs like insulin pumps or pacemakers are about to transmit logs directly to physicians or hospitals, or receive modified settings and commands. According to a study by the West Health Institute [29], device interoperability with Electronic Health Records (EHRs) could save the U.S. health care industry USD 30 billions annually. Benefits of increased medical device interoperability include a reduction of manual information transfer, an integration of new medical sensors and actuators, and fewer transcription errors. Current interoperability barriers include increased development and testing costs, increased integration costs, and a lack of commonly accepted standards [29].

SpringSim 2016, April 3 - 6, 2016, Pasadena, CA, USA
MSM 2016 - Modeling and Simulation in Medicine

A typical interoperating MD is designed as a system of systems which, in turn, is composed of heterogeneous components. Manufacturers have to address several networking, dynamic and uncertain environmental constraints. By definition, MDs are safety-critical, i.e., any malfunctioning of the device may seriously harm the patient. Interoperability features necessitate the consideration of security issues as well, such that the proper functioning of the device is not affected by cybersecurity threats.

Security of MDs is different and more challenging than regular IT security [26]. In the past, MDs have been successfully hacked on several occasions. For example, unauthorized commands have been sent wirelessly to an insulin pump [14] and medical telerobots have been hijacked [4]. Another insecurity scenario of MDs has emerged in drug pumps that became exposed to a series of remotely exploitable vulnerabilities [31]. Attackers could take complete control of these pumps. The FDA's safety communication has issued a warning to device manufacturers and health care providers to put safeguards in place to prevent cyberattacks [7]. Deaths or injuries are not yet known, but hypothetical ramifications, like "ransomware" on MDs are obvious. Diminished integrity and availability of MDs include the inability to deliver timely and effective patient care [9].

The engineering of interoperable MDs requires high safety integrity levels (SILs) and strong assurances for their fitness for public use against safety hazards and cybersecurity threats. Considering safety and security requirements in the design of MDs increases the devices' reliability, confidentiality, integrity and availability. This ensure the continuous provision and protection of essential services and assets. Current development methods and testing techniques are not adequate for the high-confidence design and manufacturing of MDs [11]. The use of rigorous approaches for the safe development of MDs is highly recommended by standards and regulations [12, 13]. However, rigorous methods not only guarantee safe behavior of devices but also help in detecting potential vulnerabilities. Advantages of using rigorous methods to enhance security of MDs have already been demonstrated. Li et al. were able to discover several software vulnerabilities and bugs in a sample cardiac pacemaker scenario using formal verification methods [18].

The overall aim of this work is to propose a methodology for the integrated modeling of safety and security requirements of MDs. We show how a functional model encapsulating the core functionality of a MD can be supplemented with safety

and security concerns in a seamless way. The proposed rigorous process for MD development not only ensures verifiability and validity of functional, safety and security models but also maintains the notion of separation of concerns among the models.

The paper is structured as follows: In Section 2, we provide background information on the employed rigorous method Event-B. The proposed methodology is presented in Section 3. In Section 4, we present a case study, where we discuss example functional, safety, and security requirements of a hemodialysis device. The paper is concluded in Section 5.

2. METHOD

Event-B is a rigorous method for system-level modeling and analysis [1]. Its notation is based upon standard first-order predicate logic and set theory. It uses the notion of refinement to support the correct-by construction approach for development by representing systems at various abstraction levels and using mathematical proofs to ensure consistency between the refinement levels.

2.1 Modeling

A specification in Event-B consists of states and events. A state is a mapping between names and values constrained by invariants. Models describing static and dynamic parts of the state are split into *Contexts* and *Machines*, respectively. The invariant is a first-order formula on the state. It defines *legal states* as the state's values where it holds. The invariant is constructed as the conjunction of smaller formulas called *axioms* and *invariants* in contexts and machines, respectively.

Events model the modifications of the state. Formally, they are guarded substitutions. Guards are first order formulas on the state and can also contain free variables called *event parameters.* Substitutions change some values, all at the same time. An event can be *fired* when its guard is true; the substitutions use the parameters' values which made the guard true. There is a special *INITIALISATION* event which must establish the first legal state.

2.2 Refinement

The notion of refinement plays the pivotal role in Event-B. Refinement is an idea of gradually introducing details and complexity into a specification which transforms into a concrete specification from an abstract one. A refinement relation between two Event-B specifications can be defined by:

- adding new invariants; this amounts to reinforce the properties of interest and to constrain the domain,
- adding new variables and constants unconnected to previous ones; this amounts to introduce new concepts and properties,
- adding new variables and constants implementing previous ones; this amounts to the usual data-refinement which goes from abstract to concrete data-structures,
- adding new events; this amounts to decompose an abstract "large" event into several concrete "smaller" events. This is called a change in observation level in [21].

Refinement definitions are embedded in the framework at two levels. The language provides users with a syntax to express the refinement relationship (*refines* for *machine* and *extends* for *context*) and the abstraction function (*with* clause in events).

2.3 Composition

The notion of composition is used to structure specifications through the interaction of independent (sub-) specifications. Composition allows for building specifications by merging various sub-specifications benefiting from their properties and proofs. The interaction among sub-specifications is usually occurred by shared states, shared events, or a combination of both. Sub-specifications interaction must be verified to comply with the desired behavioral semantics of the overall system.

2.4 Verification

The formal semantics of an Event-B model is based on two simple ideas: (1) a state with actual values must exist. We must prove that the *INITIALISATION* event can be fired and that it leads to a legal state, and (2) the firing of any event when its guard is true and the state is legal leads to another legal state. The structure of the language allows these ideas to be cast as a set of small logical formulas, called *Proof Obligations* (POs), that must be proven to hold. POs can be automatically generated; many of them can also be discharged automatically using available provers.

When a specification is refined, we also need to prove the abstract-refinement relationship between the two specifications. This amounts to prove that (i) the refinement invariant preserves the abstract invariant, (ii) the refined specification must not deadlock before its abstraction, otherwise the refined specification might not achieve what the abstract specification had previously required, and (iii) newly introduced events must not lead to divergence, i.e., not allowing the events of the abstract specification to hold. These properties can be expressed as POs. When all POs of all refinements in a chain have been discharged, we are guaranteed that the properties expressed as invariants at the most abstract level are preserved at the most concrete level. In order to verify composition, we must additionally ensure that composed events are feasible and composition invariant is preserved in the composed machine.

2.5 Validation

Validation allows stakeholders to assess whether the specification is getting nearer to fulfill their requirements. While verification catches inconsistencies and reification mistakes, validation catches requirements errors. It enables stakeholders to imagine how the specification will actually work when implemented. Event-B has intuitive operational semantics that allow the execution/animation of a specification for validation purposes. The main principle behind validation of specifications through animation is as follows:

1. fire the INITIALISATION event
2. compute the set of enabled events: $S_{enabled}$

3. while ($S_{enabled} \neq \varnothing$)
   (a) pick one event, e from $S_{enabled}$
   (b) pick values for parameters which make the guard of e true
   (c) realize the assignments in the action part of e
   (d) compute the set of enabled events: $S_{enabled}$

The execution stops when no more event can be fired. It should be noted that the execution's halt may be seen as a correct behavior for some models (functional computations typically), but as an incorrect behavior for some others (deadlocks in control loops typically).

2.6 Tool Support

Rodin [2] is the main tool environment that supports modeling in Event-B. It is built upon the Eclipse platform which allows it to be extended by plug-ins. Rodin supports the specification of machines and contexts, their refinement, and their consistency checking by automatically generating POs. Composition is achieved through plug-ins such as parallel composition plug-in [28] or feature composition plug-in [10].

In Rodin, POs can be discharged either automatically, with the help of third-party theorem provers, or interactively. Additional verification tools such as model checkers and Satisfiability Modulo Theories (SMT) solvers can also be employed to analyze the consistency of Event-B specifications. Validation of a model is achieved by executing its specification using animation plug-ins such as ProB [17] or using simulations within JeB [30].

3. APPROACH

Modeling, design and development of MDs are complex engineering tasks. The degree of complexity requires a great amount of time, resources and attention for a device's development. However, proving that the device's operation is safe and secure is still a challenging quest. While guaranteeing the absence of mistakes in a piece of software is not always possible, even the identification of their presence is difficult. Traditional quality assurance techniques like code reviews or test case generation are also insufficient in this case, mainly for two reasons. First, most of the quality assurance techniques primarily work at the level of code and are not suitable for modeling and design [16]. Second, both certification regimes and the critical nature of the medical domain explicitly require employment of rigorous verification and validation approaches [6]. Additionally, the lack of domain knowledge of software engineers often makes the matter worse.

We present an approach where MDs are synthesized using an incremental refinement process synchronizing and integrating different views and abstraction levels of a device. Quality assurance is embedded in the model development process. All requirements of a device are supplied to a refinement-based development process that rigorously checks them for consistency and conformance. Every time a new requirement is specified, be it a general behavioral requirement or a specific safety/security requirement, it first undergoes an internal consistency check and then, additionally, it is also confirmed with the stakeholders whether this requirement indeed captures the desired behavior. The stakeholders, in this way, become part of the development process right from the start. Also, the chance is minimized that errors trickle down to the later stages of the development process. As shown in Fig. 1, our approach for the development of MDs consists of four major steps, i.e., requirements elicitation, formal specification, verification, and validation.

In the requirements elicitation step, various requirements of a device and domain are gathered and grouped into respective categories depending upon their nature. There may be several requirements categories. However, for interoperable MDs, requirements can broadly be classified into three categories, i.e., functional requirements, safety requirements and security requirements.

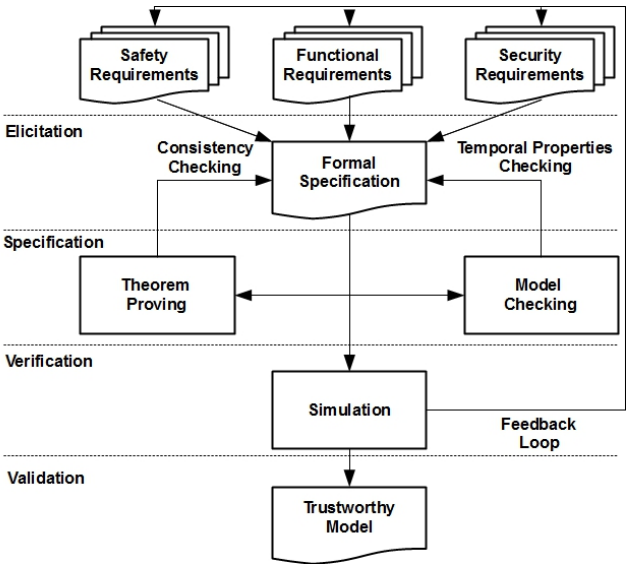

Figure 1. The rigorous development paradigm

In the formal specification step, requirements are translated into formal specifications using a stepwise modeling approach. During this process, requirements are carefully written with precision and clarity using mathematical and logical structures which are amenable to rigorous analysis to determine their correctness. We recommend to specify each category of requirements in a separate model. The functional model encapsulating the core functionality of the system can be supplemented by safety and security models. As MDs are by definition safety-critical, the formal safety model will be built on top of the core functional model. The security model will constitute the outer layer.

Once the informal requirements have been translated into a formal specification, the next step is to make sure that the requirements conform to verification standards, i.e., requirements are consistent and verifiable. During this process, it is determined that a specification conforms to some precisely expressed properties that the model is intended to fulfill such as safety and security conditions.

Two well-established formal verification approaches are theorem proving and model checking [5]. While the former refers to reasoning about defined properties using a rigorous mathematical approach, the latter is the process of exploration of the whole state space of a model to verify dynamic properties. Both deductive theorem proving and model checking are important for proving the consistency of MDs. While theorem proving is helpful in ensuring safety constraints (nothing bad happens) of the system, model checking is effective in verifying temporal constraints (eventually something good happens) of the system such as liveness and fairness properties.

Once requirements models have been specified and verified, the next step to consider is their validation. Validation is a process where it is established by examination and provision of objective evidence that the stakeholders' requirements have been captured correctly and completely in the requirements specification document. Verification alone is not sufficient to guarantee correctness of the model because it does not check whether the specification documents the requirements from the viewpoints of stakeholders.

For validation purposes, we propose to animate the formal specification in order to simulate its behavior to stakeholders. Animation is a process to demonstrate the fundamental operations of a specification using a dynamic and interactive graphical display. This technique is well-suited for making a quick mental image of the model even for non-technical domain experts who are not so familiar with complex mathematical and logical formulas.

4. HEMODIALYSIS CASE STUDY

Hemodialysis (HD) is a treatment for kidney failure that uses a MD to send a patient's blood through a filter, called a dialyzer, for extracorporeal removal of waste products. The blood is taken through the arterial access of the patient's body. The blood then travels through a tube that takes it to the dialyzer. Inside the dialyzer, the blood flows through thin fibers that filter out wastes and extra fluid using dialysate, a chemical substance that is used in HD to draw fluids and toxins out and to supply electrolytes and other chemicals to the bloodstream. The cleaned blood is then recycled back to the patient through the venous access. A vascular access lets large amounts of blood flow continuously during HD treatments to filter as much blood as possible per treatment. A specific amount of blood is conducted through the device every minute. The working principle of HD devices is depicted by Fig. 2.

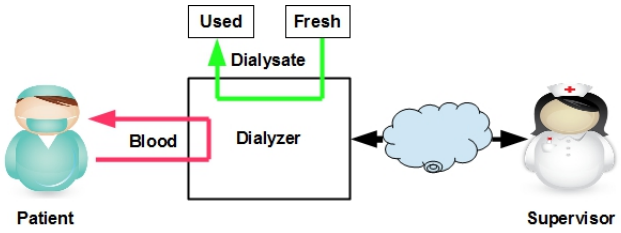

Figure 2. Working principle of hemodialysis devices

HD devices conventionally operate in a standalone mode. Patients come to a medical facility, get connected to the device and let dialysis be performed. Due to the involved complexity of the dialysis process, the resulted medical treatment is monitored by a professional caregiver, who has to be present locally, for treatment compliance and desired output. In order to demonstrate the interoperability feature, we proceed on the assumption that our HD device will provide an option of remote supervision, as depicted in Fig. 2 by the cloud between supervisor and dialyzer. This feature will be available at the cost of additional security precautions.

According to the security scores introduced in [25], standalone HD devices have a high impact on a patient but a low exposure, as they do not communicate with other devices. Therefore, security precautions are not necessary. If, however, we plan to redesign the device for use under remote supervision, we get high exposure and need to take security concerns into account during design. Let us assume that we have a device in a connected environment where it is required to establish a connection to a medical facility in order to guarantee supervision and, if needed, corrective measures by a professional during the dialysis process. Many design decisions for such a device will have an influence on safety and security issues. For example, which side starts the process of establishing the connection? What happens if the connection breaks down? From a security perspective, questions like the following arise: How can we avoid man-in-the-middle attacks? Additionally, we have to introduce safety precautions for scenarios like the physical access of malicious persons to the room in the medical facility where HD treatments are supervised.

Security requirements relate to the device's confidentiality, integrity and availability, called the CIA triad. They are needed for the protection of essential services and assets. First, sensitive data about the patient and the HD treatment may be disclosed (confidentiality). Second, data on the HD device may be altered, resulting in a range of slightly to highly severe impacts on the patient (integrity). Third, the HD device may be rendered inoperable as safety precautions require that the device has to be under the supervision of a medical caregiver (availability).

4.1 Elicitation step

We start the modeling process of HD devices by first eliciting all the related requirements. Depending upon the nature of requirements, we group them into three categories: functional requirements, safety requirements and security requirements. The first category defines the basic functionality of the device. The second category elaborates the safety-critical nature of the device. The last category defines how the device reacts to cybersecurity threats.

The functional and safety requirements of HD devices are described in detail in [19]. An initial formal model of these requirements is also presented in [20]. However, as cybersecurity is an emerging phenomenon, currently no requirements document is available that details the security requirements related to HD devices. Despite this, we can still derive

a few high-level security requirements from the FDA standard [8] which in turn recommends the use of cybersecurity framework proposed by NIST [23]. The cybersecurity framework core describes five functions: identify, protect, detect, respond and recover. For the case study, we focus on identification and protection layers through authentication and access control.

A systematic approach to security requirements engineering is needed to avoid having general lists of security features and also to consider the attacker perspective [3]. Useful techniques include misuse/abuse cases and attack trees [24], threat modeling [27], risk assessment [22], and the security development life-cycle [15]. It is necessary to identify valuable assets and steps to safeguard these.

4.2 Specification step

Once requirements have been identified and categorized, we can proceed with their formal specification. Each category is specified in a distinct formal model. Although the Event-B method allows the specification of requirements belonging to different categories as refinements of the core functional model of the device, it is better to specify them in separate models to maintain the notion of separation of concerns among functional, safety and security requirements. We also distill the static elements of requirements from the dynamic elements and specify them in contexts and machines, respectively. Subsequently, we show how a representative example of each category can be formally modeled.

Functional model

The formal specification process starts by first taking functional requirements into account. The following example requires the dialysate temperature to remain in a specific range:

> **Requirement**: If the system is in the preparation phase and performs priming or rinsing or if the system is in the initiation phase, then the dialysate temperature shall remain between 33°C and 40°C.

We first initiate a context that defines requirement-related static data such as $\mathrm{SystemPhases}$ and $\mathrm{Operations}$. Then the corresponding machine of the requirement is specified. It contains several variables, invariants and events. Invariants $\mathrm{inv1}$ and $\mathrm{inv2}$ specify the related functional requirement.

inv1 systemPhase = Preparation $\wedge$ (operation = Priming $\vee$ operation = Rinsing)
$\Rightarrow$ dialysateTemperature $\geq$ 33 $\wedge$ dialysateTemperature $\leq$ 40
inv2 systemPhase = Initiation
$\Rightarrow$ dialysateTemperature $\geq$ 33 $\wedge$ dialysateTemperature $\leq$ 40

Safety model

After specifying the functional model of the device, we proceed with the specification of the safety model. The following example specifies a sample safety requirement:

> **Requirement**: If the system is in the preparation mode and performs priming or rinsing or if the system is in the initiation mode and if the dialysate temperature exceeds the maximum temperature of 41°C, then the software shall disconnect the dialyser from the dialysate within 60 seconds and execute an alarm signal.

We start the model by defining a context that contains the requirement-related static data such as alarm types. Then the corresponding machine of the requirement is specified. It contains several variables and invariants. Invariants $\mathrm{inv3}$ and $\mathrm{inv4}$ specify the related safety requirement.

inv3 systemPhase = Preparation $\wedge$ (operation = Priming $\vee$ operation = Rinsing)
$\wedge$ dialysateTemperature $>$ 41 $\Rightarrow$
dialyserState = {Dialysate $\mapsto$ DialyserDisconnected} $\wedge$
dialyserDisconnectionTime $<$ 60 $\wedge$ alarm = ALM1
inv4 systemPhase = Therapy $\wedge$ dialysateTemperature $>$ 41 $\Rightarrow$
dialyserState = {Dialysate $\mapsto$ DialyserDisconnected} $\wedge$
dialyserDisconnectionTime $<$ 60 $\wedge$ alarm = ALM2

Then we also specify two monitoring events to capture the safety requirement of the device because both events trigger different alarms. As shown in Fig. 3, the event $\mathrm{disconnectDialyserPreparation}$ is triggered when the system is in the preparation phase and the temperature of the dialysate rises to more than 41°C during the operation. However, if the system is in the initiation phase while the same thing happens, then the event $\mathrm{disconnectDialyserInitiation}$, shown in Fig. 4, is triggered.

Event disconnectDialyserPreparation
Where
systemPhase = Preparation $\wedge$ dialysateTemperature $>$ 41 $\wedge$
(operation = Priming $\vee$ operation = Rinsing) $\wedge$
dialyserState = {Dialysate $\mapsto$ DialyserConnected} $\wedge$
dialyserDisconnectionTime $<$ 60
Then
dialyserState := {Dialysate $\mapsto$ DialyserDisconnected} $\wedge$
alarm := ALM1 $\wedge$ dialyserDisconnectionTime := 0
End

Figure 3. Event $\mathrm{disconnectDialyserPreparation}$

Event disconnectDialyserInitiation
Where
systemPhase = Initiation $\wedge$ dialysateTemperature $>$ 41 $\wedge$
dialyserState = {Dialysate $\mapsto$ DialyserConnected} $\wedge$
dialyserDisconnectionTime $<$ 60
Then
dialyserState := {Dialysate $\mapsto$ DialyserDisconnected} $\wedge$
alarm := ALM2 $\wedge$ dialyserDisconnectionTime := 0
End

Figure 4. Event $\mathrm{disconnectDialyserInitiation}$

An additional event $\mathrm{dialyserDisconnecitonClock}$ is also specified to monitor the timing constraint of the requirement. The tick of the clock is modeled as $\mathbb{N}$ that increments $\mathrm{dialyserDisconnectionTime}$ by 1 (second).

Security model

Once both functional and safety models of the device are specified, we start specifying the security model. The following example specifies a sample security requirement:

> **Requirement**: Only an authorized user with administrative privileges can set the treatment parameters.

We first define a context that specifies the static data related to the requirement. It includes sets such as ConnectionStates with elements InitialState and LoggedState, Privileges with elements AdminPrivilege and MonitoringPrivilege, Users with elements Patient and CareGiver and SettingModes with elements Enabled and Disabled. The context also defines another constant UserPrivilege that states that every user of the device has a defined privilege.

Then we specify the security requirement as invariant of the machine by stating that the parameter setting mode will only be enabled for those users who have properly logged in to the system and also have administrative privileges. Invariant inv5 states this requirement.

inv5 $\forall$ users . users $\in$ Users $\wedge$ connectionState(user) = LoggedState $\wedge$
UserPrivilege(users)=AdminPrivilege $\Rightarrow$ settingMode(users)=Enabled

The security requirement-related behavior of the device is specified by the event shown in Fig. 5.

```
Event enableSettingMode
  Any
    user
  Where
    user ∈ Users ∧ connectionState(user) = LoggedState ∧
    UserPrivilege(user) = AdminPrivilege ∧
    settingMode(user) = Disabled
  Then
    settingMode(user):= Enabled
End
```

Figure 5. Event enableSettingMode

At the end of the modeling process, the functional, the safety and the security model which are specified independently of each other can be composed into a unified model by employing composition plug-ins. The plug-ins automatically create the composed model by merging the state and events of other models. Generated POs, when discharged, ensure the seamlessness and correctness of the composition process.

4.3 Verification step

Verification of models has been performed through a combination of theorem proving, model checking and animation techniques.

Using theorem proving, we prove three kinds of properties. 1) Functional, safety and security requirements defined as invariants are maintained by behaviors of respective specifications. 2) Refinements of specifications are non-divergent, i.e., new events introduced in refinements do not take control forever by preventing events from abstract specifications from happening, and enabledness preserving, i.e., if an event is enabled at an abstract level then it is also enabled at the concrete level. 3) Composition of specifications is consistent, i.e., composed events are feasible and composition invariant is preserved by the behavior of the composed specification.

Using model checking, we ensure that legal states of models are reachable, specified formulas are satisfiable and models do not contain undesired deadlocks. Using animation, we check that system traces eventually reach their intended final states. Animation is particularly important to ensure requirements that cannot be easily specified and verified though theorem proving and model checking such as a therapy finishes successfully and the blood taken out of a body is finally put back to the body.

4.4 Validation step

For animation and model checking of our specifications, we have used the ProB tool that supports automated consistency checking of Event-B specifications via constraint solving techniques. Animation using ProB worked well. We created several behavioral scenarios and executed them accordingly to demonstrate the behavior of the device to stakeholders. Stakeholders then could visualize the requirements of the system and consequently approve, disapprove, or augment them.

The ProB tool can also be used as an assistant in finding potential invariant problems and their improvement by generating counterexamples whenever an invariant violation is discovered. It also helps in improving invariant expressions by providing hints for strengthening invariants each time an invariant is modified or a new proof is generated by the Rodin platform.

Animation may also be used for logging and playback, which are necessary features for monitoring, debugging, error tracking, and business intelligence. Animation, in this fashion, helps in finding root causes of failures when adverse events happen.

5. CONCLUSION

In this paper, we have demonstrated how the rigorous method Event-B and its supporting tool Rodin can be used to model and analyze functional, safety, and security aspects of MDs. As a case study, we have chosen a HD device that is used for the treatment of patients with kidney failure. We have shown the formal specification of functional and safety requirements. Additionally, we have supposed an interoperability scenario, where the supervisor is not present locally but performs duties from a remote location. This scenario poses security concerns. We have then shown an example of how to tackle these concerns by modeling and analysis.

Formal models provide a consistent and consolidated set of requirements. However, an enormous amount of technical details can have a negative impact on the comprehensibility of modeled requirements. We have, therefore, proposed to animate the behavior of the formal specification for inspection by stakeholders.

Formal modeling is a complex engineering task and there is no standard recipe for model development. Traditional development is performed in a linear sequence fashion. However, this approach is not suitable for overly complex cases such as HD devices. We have, therefore, classified and specified system's requirements into three distinct and independent models: functional, safety, and security, which are unified at the end of the modeling process.

Sometimes requirements belonging to different models may contradict each other. For example, a safety condition may

require that the medical procedure under performance is remotely supervised by a caregiver all the time so that the caregiver can intervene in case of a serious situation. However, a security requirement may suggest that in case of intrusion detection, the device can shutdown the network access completely or run in a minimal capability mode, i.e., send out the data such as sensor readings and event logs but not accept commands from the network. Tackling such challenging scenarios is a direction for future work.

Acknowledgement

The writing of this article is partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

REFERENCES

1. Abrial, J.-R. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T., Mehta, F., and Voisin, L. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer 12*, 6 (2010), 447–466.
3. Allen, J. H., Barnum, S., and Ellison, R. J. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley Longman, June 2008.
4. Bonaci, T., Herron, J., Yusuf, T., Yan, J., Kohno, T., and Chizeck, H. J. To Make a Robot Secure: An Experimental Analysis of Cyber Security Threats Against Teleoperated Surgical Robots. *CoRR abs/1504.04339* (2015).
5. Clarke, E. M., and Wing, J. M. Formal methods: state of the art and future directions. *ACM Comput. Surv. 28*, 4 (1996), 626–643.
6. Food and Drug Administration (FDA). Design Control Guidance For Medical Device Manufacturers, 1997.
7. Food and Drug Administration (FDA). *FDA Safety Communication: Cybersecurity for Medical Devices and Hospital Networks*. June 2013.
8. Food and Drug Administration (FDA). Content of Premarket Submissions for Management of Cybersecurity in Medical Devices: Guidance for Industry and Food and Drug Administration Staff, 2014.
9. Fu, K., and Blum, J. Controlling for cybersecurity risks of medical device software. *Communications of the ACM 56*, 10 (Oct. 2013), 35–37.
10. Gondal, A., Poppleton, M., and Snook, C. Feature composition - towards product lines of Event-B models. In *1st International Workshop on Model-Driven Product Line Engineering* (2009).
11. High Confidence Software and Systems Coordinating Group. High-confidence medical devices: Cyber-physical systems for 21st century health care. Tech. rep., Networking and Information Technology Research and Development Program, 2009.
12. International Electrotechnical Commission (IEC). IEC 60601-1:2005 Medical electrical equipment Part 1: General requirements for basic safety and essential performance, 2005.
13. International Electrotechnical Commission (IEC). IEC 62304:2006 Medical device software - Software life cycle processes, 2006.
14. Kaplan, D. Insulin pumps can be hacked. *SC Magazine* (Aug. 2011).
15. Kissel, R., Stine, K. M., Scholl, M. A., Rossman, H., Fahlsing, J., and Gulick, J. Security considerations in the system development life cycle, sp 800-64 rev.2. Tech. rep., National Institute of Standards & Technology, 2008.
16. Lamport, L. Who builds a house without drawing blueprints? *Commun. ACM 58*, 4 (Mar. 2015), 38–41.
17. Leuschel, M., and Butler, M. ProB: An Automated Analysis Toolset for the B Method. *Journal Software Tools for Technology Transfer 10*, 2 (2008), 185–203.
18. Li, C., Raghunathan, A., and Jha, N. K. Improving the Trustworthiness of Medical Device Software with Formal Verification Methods. *IEEE Embedded Systems Letters 5*, 3 (Sept. 2013), 50–53.
19. Mashkoor, A. The hemodialysis machine case study. Tech. rep., Software Competence Center Hagenberg GmbH, 2015.
20. Mashkoor, A. Model-driven development of high-assurance active medical devices. *Software Quality Journal* (2015), 1–26.
21. Mashkoor, A., and Jacquot, J.-P. Utilizing Event-B for Domain Engineering: A Critical Analysis. *Requirements Engineering 16*, 3 (2011), 191–207.
22. National Institute of Standards and Technology (NIST). Guide for Conducting Risk Assessments - NIST special publication 800-30 revision 1. Tech. rep., Sept. 2012.
23. National Institute of Standards and Technology (NIST). Framework for improving critical infrastructure cybersecurity, 2014.
24. Opdahl, A. L., and Sindre, G. Experimental comparison of attack trees and misuse cases for security threat identification. *Information and Software Technology 51*, 5 (May 2009), 916–932.
25. Sametinger, J., and Rozenblit, J. Security Scores for Medical Devices. Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - Volume 5: HEALTHINF (Rome, Italy, Feb. 2016), 533–541.
26. Sametinger, J., Rozenblit, J., Lysecky, R., and Ott, P. Security Challenges for Medical Devices. *Communications of the ACM 58*, 4 (Apr. 2015), 74–82.

27. Shostack, A. *Threat Modeling: Designing for Security*. John Wiley & Sons Inc, Apr. 2014.

28. Silva, R. Towards the composition of specifications in Event-B. *Electronic Notes in Theoretical Computer Science 280* (2011), 81 – 93.

29. West Health Institute. The Value of Medical Device Interoperability. Tech. rep., Mar. 2013.

30. Yang, F., Jacquot, J.-P., and Souquières, J. Proving the fidelity of simulations of Event-B models. *15th IEEE International Symposium on High-Assurance Systems Engineering* (2014), 89–96.

31. Zetter, K. Hacker Can Send Fatal Dose to Hospital Drug Pumps. *WIRED* (2015).