7-2014

From Requirements to Code: Model Based Development of a Medical Cyber Physical System

Anitha Murugesan

Mats Heimdahl

Michael Whalen

Sanjai Rayadurgam

John Komp

See next page for additional authors

Recommended Citation

Anitha Murugesan, Mats Heimdahl, Michael Whalen, Sanjai Rayadurgam, John Komp, Lian Duan, BaekGyu Kim, Oleg Sokolsky, and Insup Lee, "From Requirements to Code: Model Based Development of a Medical Cyber Physical System", . July 2014.

Joint event of the Fourth Symposium on Foundations of Health Information Engineering and Systems (FHIES) and the Software Engineering in Healthcare (SEHC) workshop (FHIES/SEHC '14), Washington, D.C. , July 17-18, 2014.

From Requirements to Code: Model Based Development of a Medical Cyber Physical System

Abstract
The advanced use of technology in medical devices has improved the way health care is delivered to patients. Unfortunately, the increased complexity of modern medical devices poses challenges for development, assurance, and regulatory approval. In an e ort to improve the safety of advanced medical devices, organizations such as FDA have supported exploration of techniques to aid in the development and regulatory approval of such systems. In an ongoing research project, our aim is to provide effective development techniques and exemplars of system development artifacts that demonstrate state of the art development techniques.

In this paper we present an end-to-end model-based approach to medical device software development along with the artifacts created in the process. While outlining the approach, we also describe our experiences, challenges, and lessons learned in the process of formulating and analyzing the requirements, modeling the system, formally verifying the models, generating code, and executing the generated code in the hardware for generic patient controlled analgesic infusion pump (GPCA). We believe that the development artifacts and techniques presented in this paper could serve as a generic reference to be used by researchers, practitioners, and authorities while developing and evaluating cyber physical medical devices.

Disciplines
Computer Engineering | Computer Sciences

Comments
Joint event of the Fourth Symposium on Foundations of Health Information Engineering and Systems (FHIES) and the Software Engineering in Healthcare (SEHC) workshop (FHIES/SEHC '14), Washington, D.C. , July 17-18, 2014.

Author(s)
Anitha Murugesan, Mats Heimdahl, Michael Whalen, Sanjai Rayadurgam, John Komp, Lian Duan, BaekGyu Kim, Oleg Sokolsky, and Insup Lee

From Requirements to Code: Model Based Development of A Medical Cyber Physical System[⋆]

Anitha Murugesan[1], Mats P.E. Heimdahl[1], Michael W. Whalen[1], Sanjai Rayadurgam[1], John Komp[1], Lian Duan[1], Baek-Gyu Kim[2], Oleg Sokolsky[2], Insup Lee[2]

[1] Department of Computer Science and Engineering, University of Minnesota, 200 Union Street, Minneapolis, MN 55455,USA
{anitha,heimdahl,whalen,rsanjai,komp,lduan}@cs.umn.edu
[2] Department of Computer and Information Science, University of Pennsylvania,3330 Walnut Street, Philadelphia, PA 19104, USA
{baekgyu,sokolsky,lee}@cis.upenn.edu

Abstract. The advanced use of technology in medical devices has improved the way health care is delivered to patients. Unfortunately, the increased complexity of modern medical devices poses challenges for development, assurance, and regulatory approval. In an effort to improve the safety of advanced medical devices, organizations such as FDA have supported exploration of techniques to aid in the development and regulatory approval of such systems. In an ongoing research project, our aim is to provide effective development techniques and exemplars of system development artifacts that demonstrate state of the art development techniques.
In this paper we present an end-to-end model-based approach to medical device software development along with the artifacts created in the process. While outlining the approach, we also describe our experiences, challenges, and lessons learned in the process of formulating and analyzing the requirements, modeling the system, formally verifying the models, generating code, and executing the generated code in the hardware for generic patient controlled analgesic infusion pump (GPCA). We believe that the development artifacts and techniques presented in this paper could serve as a generic reference to be used by researchers, practitioners, and authorities while developing and evaluating cyber physical medical devices.

1 Introduction

Cyber Physical Systems (CPS) systems are physical systems whose operations are monitored and controlled by digital – often networked – computers. Advances in medical device technology, especially sophisticated software controls, has

[⋆] This work has been partially supported by NSF grants CNS-0931931 and CNS-1035715 improved health care delivery. Nevertheless, the increased use of software has lead to new failure modes for complex medical devices, for example, the numerous recalls on infusion pumps – a medical device used for controlled drug delivery to patients – indicate that they have posed serious health hazards [3]. Hence, along with the advances in technology comes the need for effective development and regulatory practices to ensure the safety of such systems. In an effort to improve the safety of advanced medical devices, organizations such as FDA have supported initiatives to aid in the development and regulatory approval of such systems, for example, the Infusion Pump Initiative [3]. Our aim is to contribute to these initiatives by defining and demonstrating effective development techniques, and providing archetypes of system development artifacts of a medical device.

In this paper, we describe an end-to-end model based approach for developing medical device software; we outline our approach, elaborate on our experiences, challenges, and lessons learned when formulating and analysing requirements, modeling the system, performing formal verification, generating code, and executing the generated code to control an infusion pump. We use a Generic Patient Controlled Analgesia Infusion Pump (GPCA) system – a type of infusion pump – to illustrate our approach. We believe that the development techniques and the artifacts presented can serve as a generic exemplar to be used by researchers, practitioners, and authorities while developing and evaluating these type of devices. All the artifacts discussed in this paper are available at http://crisys.cs.umn.edu/gpca.shtml.

The paper is organized as follows. Section 2 gives an overview of our model based development approach followed by Section 3 providing an overview of the GPCA system – the case example that is used throughout the paper. Section 4 describes the requirements analysis and modeling efforts. Section 5 provides an overview of the verification strategies including a discussion on how our specific GPCA implementation is analysed in context of a closed loop medical system. Section 6 discusses the hardware-code integration challenges. Finally Section 7 concludes with a brief summary of our work and the steps to the future.

2 A Model Based Development Approach

We advocate a model based approach to the development of CPS software. Model-based development allows the unambiguous capture of requirement, architecture, and design detail, and enables the extensive use of tools and automation. This allows for greatly reduced manual effort while maintaining or improving the capability of fault finding and enhancing quality of the system [14, 15].

To ascertain that a system is safe and effective, it is imperative to precisely specify its requirements. In practice, however, the requirement are rarely (if ever) well known at the onset of a project. Instead, the initial set of requirements form a basis for a proposed solution (an initial systems architecture or design). In our case, we advocate modeling to evaluate design alternatives and explore the desired system. In effect, the models are proposed solutions to the problem at

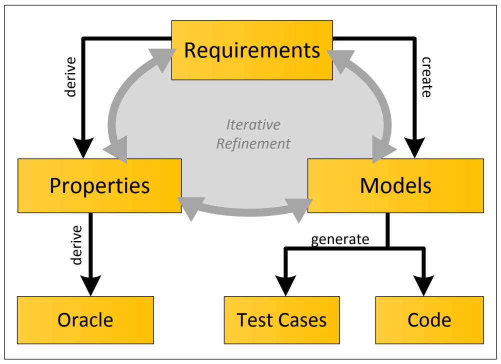

Fig. 1. Approach Overview

hand (the requirements); the models are early prototypes of a proposed system that help visualize and analyse the problem as well as the proposed solution.

Models and requirements exist in a symbiotic relationship; through iterative evolution they contribute to each other's improvement. While constructing models of a system – by exploring the solution domain – an engineer adds architectural and design detail not specifically stated in requirements, architectural and design information that helps clarify existing requirement as well as discover missing ones [6, 27]. Similarly, as requirements are modified and added, the models evolve to accommodate the new constraints. Throughout the process, while modeling, one may find that the design cannot meet the system-level requirements (for example, the requirements may be unrealizable or there is no cost effective solution). This may lead to the imposition of constraints on the system's operational environment or a renegotiation of the system-level requirements. Nuseibeh identified this virtuous cycle as the Twin Peaks model [20]. If the modeling notation is formal, various verification techniques can be used to determine if formalized requirements (that is, desirable model properties) are satisfied in the model; a failed verification (the model does not meet the requirements) points to a problem that must be addressed through a modified model, modified requirements, or a further constrained operating environment. Miller et al. first advocated such a process in the context of developing critical avionics systems [14].

In addition to the advantages during the early development phases, a formal model-based approach can be leveraged throughout the life-cycle. Using commercial tools, the models and requirements can be used for code generation, test case generation, and used as oracles during testing. In the following sections we will discuss the various steps in the process and share the experiences we gained from developing artifacts for the Generic Patient Controlled Analgesia Infusion Pump (GPCA).

3 GPCA System Overview

Infusion pumps are medical CPS used for controlled delivery of liquid drugs into a patient's body according to a physician's *prescription*, a set of instructions that governs the plan of care for that individual. Modern infusion pumps have evolved to include several features for different drug delivery modes, programmability, notifications and logging. For instance, most pumps deliver the drug at a constant (and usually low) rate for an extended period of time, called *basal* delivery mode. Special types of pumps such as Patient-controlled analgesia (PCA) pumps are equipped with a feature that allows patients to self-administer a controlled amount of drug – a *patient bolus* mode may be activated to deliver prescribed additional drug, typically to alleviate acute pain. The pumps also have the capability to monitor and notify the clinicians when exceptional conditions such as air embolism occur, when they are in use.

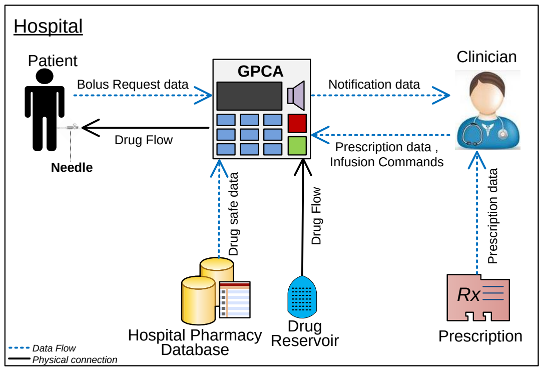

Fig. 2. GPCA Overview

Figure 2 shows an external intravenous Generic Patient Controlled Analgesia (GPCA) device in a typical usage environment, a hospital or a clinic. A clinician, a certified practitioner, operates the GPCA device: programs the prescription information, loads the drug, connects the device with the patient, and responds to notifications from the device. The patient receives the drug from the device through an intravenous needle inserted by the clinician. The patient can self administer prescribed amounts of additional drug by pressing a bolus request button accessible at the patient's bed. The GPCA also has an interface to the hospital pharmacy repository for accessing manufacturer provided drug information that is typically used to verify if the programmed therapy regimen is within drug safe limits. The GPCA has three primary functions: (1) deliver the drug based on the prescribed schedule and patient requests, (2) prevent hazards that may arise during its usage, and (3) monitor and notify the clinician of certain exceptional conditions encountered. In our work we focus on analysing the infusion pump's software that controls the drug infusion and raises alarms to notify the clinician when hazards are detected. Other infusion pump subsystems such as the user interface, sensors and actuators are out of scope of this work.

4 Model Based Requirements Analysis

As mentioned in Section 2, it is imperative to have a well defined set of requirement for a system that is discovered through deliberations between stakeholders. Effective requirements elicitation relies on an iterative process where the requirement domain and the solution space are explored concurrently [20]. In our work, early system modeling and analysis have served as crucial aids. Nevertheless, during the requirements and modeling efforts of the GPCA, several challenges – some unexpected, some expected – with developing requirements for CPS surfaced. First, in CPS, the system level requirements are expressed in the physical world; discovering, understanding, and capturing these requirements were unexpectedly challenging. Second, closely related to capturing system level requirements is the determination of what is considered the system under development and what is considered the environment of the system; determining this system boundary was an unexpected challenge. Finally, even a simple system contains several interacting features; properly understanding the patterns of interaction among features (the feature interaction problem) was an expected challenge where our modeling efforts were invaluable.

4.1 Requirements in the Continuous Domain

Understanding the interaction patterns between the system and its environment as well as between the various hardware and software components within the system is crucial for capturing CPS requirements. The behavior of both the hardware components as well as the environment is continuous and continual in nature. Frequently, we found that the requirements for a system were focusing on the ordering of events in the system (as is done with various temporal logics) and the real-time constraints on these events. That information, however, is not sufficient to capture the continuous and continual nature of the system. For example, the rate of change of a controlled variable, the time it takes for a controlled variable to settle sufficiently close to a set-point, and the cumulative errors built up over time may be of critical importance, concepts that we found were not stated explicitly as requirements. For example, let us consider a safety requirement of the GPCA,

> *"An over-infusion alarm shall be triggered if the flow rate of the drug in the infusion tubing is greater than X% of the prescribed flow rate for more than Y minutes"*

This requirement concerns the response to a flow rate of drug exceeding the prescribed value. Nevertheless, a closer examination of the problem reveals some ambiguity: Is it acceptable for the flow rate to exceed the threshold for Y consecutive minutes or does the time accumulate over the full infusion interval? Similarly, what is the cumulative effect of excess drug flow? For example, if the flow periodically exceeds $X\%$ for $Y - 1$ consecutive minutes and then comes back to normal for a short period of time, there is a possibility of over-infusion due the rate at which the drug is infused as well as the overall volume infused into the patient within a certain interval of time. We found behavioral modeling in the physical domain particularly useful in the elicitation, discovery, and refinement of such requirements.

Control System Modeling: To understand the control behavior of the system in the continuous domain, we developed control system models – traditionally used to evaluate various control strategies, tune the controllers, etc.

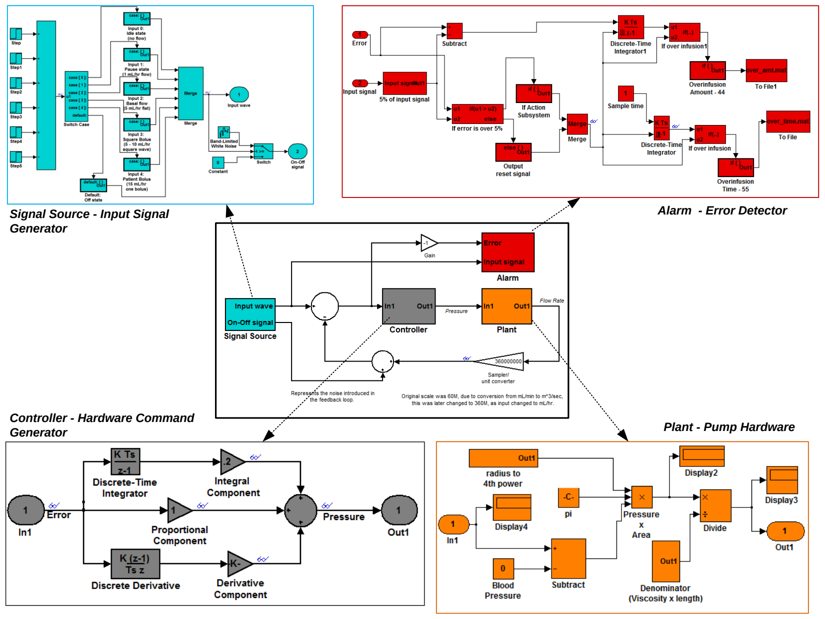

Fig. 3. Control System Model

For the GPCA, we developed these models using MathWorks Simulink [12] tools to better understand the requirements needed to adequately constrain the desired system behavior (Figure 3) [6]. Through these modeling efforts we had an opportunity to explore various system responses, investigate how a proposed system might behave in its intended environment, and thereby identify precise requirements for the system. For example, the overinfusion safety requirement is now refined and stated as,

"The software shall issue an over-infusion alarm if the flow rate of the drug in the infusion tubing is greater than X% of the prescribed flow rate for more than Y consecutive minutes or more than Z minutes cumulative during the infusion duration."

"The software shall issue an over-infusion alarm if the volume of drug delivered in P consecutive minutes anytime during the infusion duration is more than Q% of the prescribed volume to be infused in that interval."

Through this modeling exercise, in addition to identifying precise requirements for the software, we were also able to identify categories of requirements that should be considered while capturing requirements for a cyber physical system, for example, accuracy, rise/drop time, rate of change, overshoot or maximum deviation, settling time, and cumulative error [6].

4.2 System and Component Scope

To precisely state requirements such as the requirements of the continuous domain, a clear-cut demarcation of the boundary between the system and its environment as well as between the system components is crucial. Scoping a system is a classic requirements engineering challenge and insufficient analysis among and between different engineering disciplines when determining system scope (precise demarcation of the system boundary) is a significant cause of CPS failures [10, 24]. The process of identifying the boundary is not restricted to the interface between the system and its environment; the hierarchical structure of the system under development (the system architecture) is also crucial. Most systems are developed as collections of communicating smaller, manageable sub-systems. To ascertain that the sub-systems – when composed – satisfy the overall system requirements, it is necessary that the designers express the structure (the system architecture, its components and connections) of the system and the appropriate requirements are allocated to each component.

Providing a well defined scope for the requirements is essential to maintain intellectual control of the the development and assurance efforts. For example, for the GPCA system, if we write requirements in terms of the *prescribed dosage* of a drug, we can postpone discussion of entry errors (the clinician entering the prescribed dosage into the pump is part of our system). On the other hand, from a development perspective, it is most likely preferable to write infusion pump requirements in terms of the *dosage entered at the pump interface*; the clinician is not part of the pump system and entry errors are part of its environment. Thus, the choice of scope does not only have implications for our development efforts (what is "in" our system and what is "outside" our system), but also has profound implications for the way the system is assured for safety.

To assist us in the process of scoping the GPCA system and establishing rigorous interfaces between the system and its environment as well as between system components we relied on architectural modeling. The scoping exercise forced us to consistently express requirements in terms of inputs and outputs at a particular level of abstraction.

Architectural Models: For the GPCA, we developed an architectural model, shown in Figure 4[3], that clearly defines its interfaces (input and output variables) [17]. We used the Architecture Analysis and Description Language (AADL) [23] for the modeling. For formal verification (discussed in Section 5), the GPCA turned out to be too large to be handled as a monolithic model; hence

[3] In figure 4, the connections between components are abstracted for visual clarity.

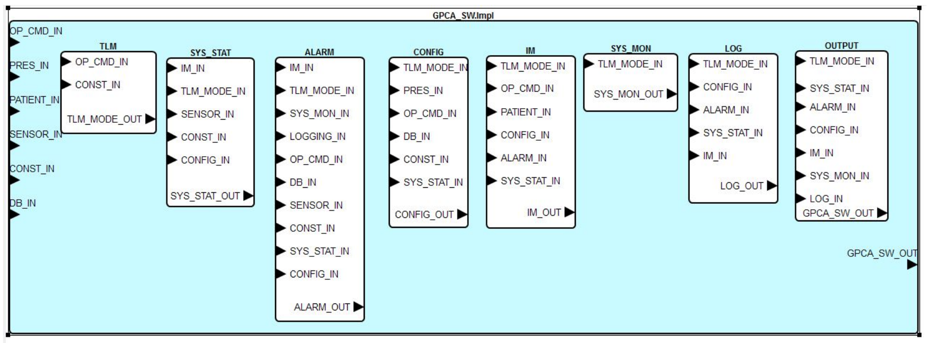

Fig. 4. Architectural Model

to manage complexity and maintain intellectual control, we decomposed it into smaller manageable components and defined an architecture that specified how the components are connected. The architecture also clearly specifies the interfaces of each component. This exercise typically occurs repeatedly over several layers of system abstraction; each component in the architectural model has its own set of interfaces, subcomponents, and component requirements. Rigorously defining the interfaces helped consistently scope all requirements by expressing them strictly in terms of the respective interface variables. The choice of the system boundary (the scope of the system) for a requirements effort is of course debatable; in our work we do not attempt to suggest an optimal scope. We would like, however, to emphasize that whatever scope boundary is chosen, it must be well defined and all requirements and environmental assumptions must be stated based on this system boundary [4].

4.3 Mode Logic Requirements

Although the architectural models helped to clarify the interfaces to ease the requirements definition burden, they do not help resolve the complexity of the behavioral requirements on the system. The dynamic behaviors of complex systems are frequently defined in terms of operational *modes*, which are frequently viewed to be mutually exclusive sets of system behaviors [11]. The modes together with the rules defining the transitions between them are called *mode logic* [8]. Derivation of precise requirements of the mode logic is challenging due to the plurality of modes and the complexity of the rules that govern the transitions. In the GPCA, understanding the mode logic of the various infusion delivery types, such as basal or bolus, was nontrivial and we again relied on modeling to illuminate and resolve the issues.

Finite State Machine Models: To analyze the GPCA infusion mode logic requirements and behaviors, we modeled it as a finite state machine, using MathWorks' Simulink and Stateflow [12]. A portion of the model of the GPCA infusion mode logic is shown in Figure 5. In our endeavor to model the mode logic [16], we identified requirements and modeling patterns and requirement scenarios not apparent before the modelling efforts.

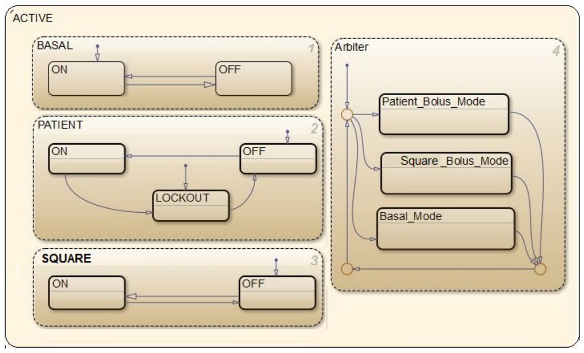

Fig. 5. Infusion Mode Logic of GPCA

For example, a statement from one of the versions of the GPCA software's requirements document reads:

> *"A patient bolus shall take precedence over a square bolus. At the completion of the patient bolus, the square bolus shall continue delivery."*

This requirement may be taken to mean that the patient bolus simply overrides the programmed square bolus infusion as illustrated in Figure 6(a) or the square bolus is suspended until the patient bolus is delivered and resumed thereafter as illustrated in Figure 6(b). Clearly, these alternatives influence how much drug a patient can receive over a bolus interval. Note here that both may be acceptable from a safety perspective; there may, however, be clinical differences making one approach more desirable than the other. The modeling efforts help identify and resolve such tradeoffs.

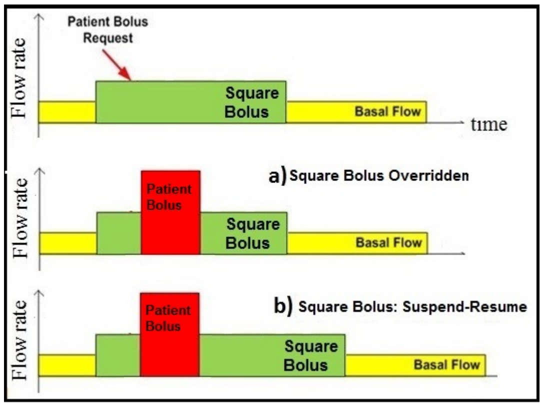

Fig. 6. Mode Interaction Patterns

In addition, the modeling efforts helped identify the major operational modes and an overall conceptual structure of the GPCA behavior. This structure was later used to reorganize the natural language requirement to be conceptually cleaner and help understandability and readability. For example, in the GPCA natural language requirements document, we organized the requirements in a hierarchical structure reflecting the model structure of the system; the requirements of the parent level modes are also applicable to all the child modes. For example, all requirements applicable for the Active mode are also applicable in the Basal mode (submode of Active). This organization enhanced clarity and reduced the repetition of a requirement in multiple places within the same document.

5 Formal Verification

As mentioned previously, complex systems are hierarchically constructed. For example, the GPCA software was decomposed into several components (captured in the AADL architectural model), requirements allocated to each component, and the detailed functionality of each component captured in the behavioral model (captured using Simulink and Stateflow). Thus, to verify the system as a whole, one needs a verification chain to help determine (1) whether or not the composition – based on the architecture – of the component level requirements is adequate to establish the system-level requirements, (2) if the detailed specification of a component's behavior satisfy the component-level requirements, and (3) whether the system operating in its intended environment will meet the safety constraints defined in the physical world. In this section, we summarize our verification strategy. A more detailed account is available in previous publications [19].

5.1 Compositional Verification

To formally prove that a system decomposed into an architecture satisfies its system level requirements requires a compositional argument involving (i) the component behaviors and (ii) the assumptions about the system's environment [5]. To perform compositional verification of the GPCA, we used AGREE (Assume Guarantee Reasoning Environment) – a compositional verification framework developed for AADL verification [2]. AGREE is based on assume-guarantee reasoning [13], that provides an appropriate mechanism for formally capturing the component requirements, and assumptions to verify system requirements. The AGREE framework is a plugin to the OSATE environment (the same environment used to model the AADL architecture).

In the GPCA architectural model, we defined requirements as assume-guarantee contracts for each component, as well as for the system as a whole. The AGREE tool automatically verifies if the component contracts in the specified architecture satisfy the system requirements.

5.2 Behavioral Verification

Knowing that the architecture and component requirements meet the system level requirement, it remains to be shown that the detailed component specifications meet the component requirement. As mentioned in the previous section, the component behaviours were modeled using MathWorks Simulink and Stateflow tools. Hence, using MathWorks Simulink Design Verifier (SDV) [12] – a plug-in tool for formal verification – one can verify that the behavioral models satisfy requirements modeled as synchronous observers. Ideally, the component requirements captured in AGREE would be reused as synchronous observers in the verification of the component behaviors. Unfortunately, the automatic translation from AGREE contracts to a notation acceptable in the Simulink Design Verifier has not been completed. Therefore, we manually translated the AGREE contracts to Embedded MATLAB. Embedded MATLAB was chosen over other possible notations (such as Simulink or Stateflow) for the observers since it is quite similar to the AGREE contracts making the translation easy. In the process of capturing the observers, we developed a structuring pattern for the Simulink and Stateflow models that would allow the independent evolution of the detailed behavioral models and the requirements captured as observers [26]. Given that the models of the GPCA had been decomposed into manageable pieces (through a combination of the compositional verification in AGREE and the component verification in SDV) scalability of the verification was not a problem and the overall correctness with respect to the system level requirements could be automatically established.

5.3 Verification in Context

Although one through compositional verification can demonstrate that the GPCA meets its requirements, this is no guarantee that the GPCA will work as intended in its operational environment. Thus, determining how the GPCA will perform when infusing a drug into a patient is of critical importance.

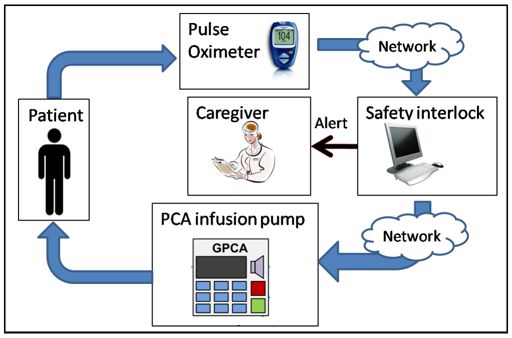

Fig. 7. GPCA in Closed Loop System

In prior work, we verified a collection of safety properties of a closed-loop medical system in which an infusion pump is used to deliver opioid medication. A serious side effect of opioid medication is that overinfusion of the drug can lead to respiratory failure in patients. Hence a sensor, such as a pulse oximeter, used to detect onset of respiratory problems with patients, and a safety interlock device, that monitors the sensor readings, is used to command the pump to stop once a certain sensor reading (threshold) is crossed.

The closed loop system considered a highly abstract PCA pump [21] and was modeled as a timed automaton in UPPAAL [1] whereas the GPCA discussed in this paper is a detailed model of a realistic infusion pump with rich functionality captured in AGREE and Simulink/Stateflow. To demonstrate that the GPCA satisfies the closed loop real-time safety requirements, it is necessary to demonstrate that the GPCA (expressed in AGREE) satisfies the PCA requirement (captured in UPPAAL). For example, a closed loop system level requirement for the PCA informally states, *Infusion cannot begin until parameters of the infusion are configured and pump is started by the caregiver.* This requirement (continuous time), that was modeled as an automaton in UPPAAL, was recaptured as AGREE contracts as shown below. The first contract specifies that, at startup the system shall not be in the infusing mode and the second contract specifies that if `Infusion_Start` and `Infusion_Programmed` commands are not issued by the caregiver, the system shall not be in infusing mode. In the second contract `pre(PUMP_DISPLAY_OUT.Current_System_Mode)` indicates the value of the system mode in the previous execution step. These contracts were compositionally verified using the detailed GPCA's models by devising an informal abstraction mapping between the models [18].

```
property start_up_behaviour =
   is_start_up => (PUMP_DISPLAY_OUT.Current_System_Mode != "Infusing");

property no_infusion_start =
 true -> pre(PUMP_DISPLAY_OUT.Current_System_Mode) != "Infusing" and
   not(CAREGIVER_IN.Infusion_Start and CAREGIVER_IN.Infusion_Programmed)
     => (PUMP_DISPLAY_OUT.Current_System_Mode != "Infusing");
```

When attempting to perform compositional reasoning in a situation involving multiple modeling formalisms, verification paradigms, and associated tools, care must be taken to ensure that results from one formalism can be used in another formalism. To perform this verification, we employed a semi-formal approach for determining whether the results in UPPAAL could be composed with the results obtained in AGREE.

6 GPCA Implementation

The goal of a development project is of course to produce executable code that safely can be deployed on the target platform, in our case, a working infusion pump. Given our verified models, we relied on Simulink Coder [12] to automatically generate C code from the models. (The Simulink Coder is not a trusted code generator; thus, the generated code must be subjected to rigorous conformance testing demonstrating that no faults were introduced in the code generation. This testing is outside the scope of this paper but in can be fully automated as extensively discussed in our previous work [7, 22, 25]) Although the process of code generation from the model is straightforward, installing the software on the actual hardware can be a challenge. Below we discuss the challenges we faced while integrating the device with the generated code and discuss the approaches we took to address them.

6.1 Device Set Up

Our target platform was a modified commercial infusion pump developed in the GPCA reference implementation project [9]. The experimental platform is a Baxter PCA infusion pump and is equipped with sensors and actuators that are necessary to verify infusion scenarios. The pump-motor (Figure 8-(1)) is used to generate forces to move the loaded syringe downward, so that drug flows into a patient's body through the intravenous needle. The low and empty reservoir switches (Figure 8-(2) and Figure 8-(3)) are used to detect if the remaining volume of drug in the syringe is too low or empty respectively. The patient bolus request button (Figure 8-(4)) enables the patient to request an additional bolus dose. The door sensor (Figure 8-(5)) is used to detect whether the door of the pump is opened or closed in order to prevent accidental removal of syringes during infusion. The buzzer (Figure 8-(6)) is used to raise an audible signal when certain alarming condition occurs. These sensors and actuators are interfaced with a micro-controller board where the generated code from the model runs.

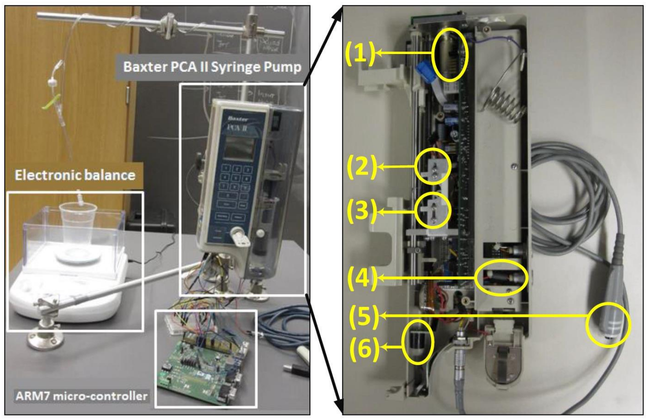

Fig. 8. Test setup with the Baxter Hardware
(1) Pump Motor (2) Low Reservoir Switch (3) Empty Reservoir Switch (4) Door Sensor (5) Patient Bolus Request Button (6) Buzzer

6.2 Hardware Abstractions and Code Clarity

The formal models describe the functionality of the system but abstract away hardware details such the signals needed to communicate with devices such as motors or buttons. Such abstraction is desirable to make the models device and hardware independent allowing us to target multiple physical infusion pumps with our generated code. On the other hand, it necessitates manually developed device dependent drivers to be provided with the generated code; thus, introducing a source of potential faults. Nevertheless, although in this project they were not developed, verified and trusted device drivers could be developed using the same techniques discussed throughout this paper.

Surprisingly, one of the most challenging aspects of the implementation effort was the task of understanding the generated code and identifying the appropriate signals within the model and connect them to the device drivers. This problem can be largely attributed to our unfamiliarity with the code generator; simple tools support to assist in this endeavour would have been helpful and will be developed for our future case studies.

6.3 Testing the Code

Testing of the software on the target platform (as opposed to in the host environment on a workstation) brought new – although not entirely unexpected – challenges and insights. For example, we found that the variable recording the volume of remaining drug in the drug reservoir occasionally (and inexplicably) indicated that the reservoir had been refilled although no such action had been taken. In the model, this variable was defined as an unsigned 8-bit integer value; in the code, this variable was decremented below zero and a wrap-around occurred. One may wonder why this problem was not identified in any of the formal verification performed on the GPCA models. Although the problem led to unexpected behaviors, none of the behaviors violated the safety requirement that were the focus of the verification efforts. Note here that the Simulink Design Verifier has the capability of checking for under and overflow, in our initial verification efforts this capability was simply not used. Flaws such as this point to an inherent limitation of formal verification – the verification will only guarantee compliance with the properties actually used in the verification. Methods and techniques to help ensure that the set of requirements used in the verification is sufficient is an area needing further investigation.

7 Summary and Conclusion

In this paper we presented a model based development technique for developing critical cyber physical medical device systems. We illustrated our approach using a Generic Patient Controller Analgesia (GPCA) pump as a case study. While illustrating the approach ranging from natural language requirements to executable software, we also discussed some of the challenges we faced at the various stages in the project.

The intent with the GPCA development was as an exercise to develop and demonstrate effective development techniques and make the artifacts created in the process (requirements, models, verification properties, executable code, etc.) available publicly as exemplars. In the course of the project, there were several unexpected challenges, for example, the difficulty of defining the scope of a system (what is part of the system and what is part of the environment) and properly defining requirement for the continuous and real-time aspect of a system. To overcome these problems, we adopted a model based approach relying on the development of numerous models to address the challenges at hand: continuous control models to help clarify requirement in the physical domain, timed automata to verify closed loop properties of the infusion system, architectural models in AADL to help clarify system boundaries and explore the solution space, formal models of component contracts to verify that the architecture met system requirement, and detailed behavioral models of the components for verification and code generation. The generated code was integrated on multiple infusion pump hardware platforms to validate that the approach was feasible.

Following these efforts, we in this project will continue to develop the formal verification capabilities, explore the challenging questions related to rigorously and in an automated fashion test software in an embedded target environment, and explore how to perform verification in the inherently stochastic environment present in any cyber physical system.

References

1. G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems (revised lectures)*, volume 3185 of *LNCS*, pages 200–237, 2004.
2. Darren D. Cofer, Andrew Gacek, Steven P. Miller, Michael W. Whalen, Brian LaValley, and Lui Sha. Compositional verification of architectural models. In Alwyn E. Goodloe and Suzette Person, editors, *Proceedings of the 4th NASA Formal Methods Symposium (NFM 2012)*, volume 7226, pages 126–140, Berlin, Heidelberg, April 2012. Springer-Verlag.
3. FDA. White Paper: Infusion Pump Improvement Initiative. April 2010.
4. Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.
5. J. Hammond, R. Rawlings, and A. Hall. Will it work? [requirements engineering]. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE Int'l Symposium on*, pages 102 –109, 2001.
6. Mats Heimdahl, Lian Duan, Anitha Murugesan, and Sanjai Rayadurgam. Modeling and requirements on the physical side of cyber-physical systems. In *Second Int'l Workshop on the Twin Peaks of Requirements and Architecture*, May 2013.
7. Mats P.E Heimdahl, S. Rayadurgam, Willem Visser, George Devaraj, and Jimin Gao. Auto-generating test sequences using model checkers: A case study. In *3rd Int'l Workshop on Formal Approaches to Testing of Software (FATES 2003)*, 2003.
8. Anjali Joshi, Steven P Miller, and Mats PE Heimdahl. Mode confusion analysis of a flight guidance system using formal methods. In *Proceedings of 22nd Digital Avionics Systems Conf.(DASC'03)*, volume 1, pages 2–D. IEEE, 2003.
9. Baek Gyu Kim, A. Ayoub, O. Sokolsky, Insup Lee, P. Jones, Yi Zhang, and R. Jetley. Safety-assured development of the GPCA infusion pump software. In *Embedded Software (EMSOFT), 2011 Proceedings of the Int'l Conf. on*, pages 155–164, Oct 2011.
10. J.C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th Int'l Conf. on Software Engineering*, pages 547–550. IEEE, 2002.
11. Nancy Leveson, L. Denise Pinnel, Sean David Sandys, Shuichi Koga, and Jon Damon Reese. Analyzing software specifications for mode confusion potential. In *Proceedings of a Workshop on Human Error and System Development*, pages 132–146, 1997.
12. MathWorks Inc. Products. http://www.mathworks.com/products.

13. Ken L. McMillan. Circular compositional reasoning about liveness. Technical Report 1999-02, Cadence Berkeley Labs, Berkeley, CA 94704, 1999.
14. Steven P. Miller, Alan C. Tribble, Michael W. Whalen, and Mats P. E. Heimdahl. Proving the shalls: Early validation of requirements through formal methods. *Int. J. Softw. Tools Technol. Transf.*, 8(4):303–319, 2006.
15. Steven P. Miller, Michael W. Whalen, and Darren D. Cofer. Software model checking takes off. *Commun. ACM*, 53(2):58–64, 2010.
16. Anitha Murugesan, Sanjai Rayadurgam, and Mats Heimdahl. Modes, features, and state-based modeling for clarity and flexibility. In *Fifth Int'l Workshop on Modeling in Software Engineering*, May 2013.
17. Anitha Murugesan, Sanjai Rayadurgam, and Mats Heimdahl. Using models to address challenges in specifying requirements for medical cyber-physical systems. In *Fourth workshop on Medical Cyber-Physical Systems*, April 2013.
18. Anitha Murugesan, Oleg Sokolsky, Sanjai Rayadurgam, Michael Whalen, Mats Heimdahl, and Insup Lee. Linking abstract analysis to concrete design: A hierarchical approach to verify medical cps safety. In *Int'l Conf. on Cyber-Physical Systems (ICCPS) 2014*, April 2014.
19. Anitha Murugesan, Michael W. Whalen, Sanjai Rayadurgam, and Mats P.E. Heimdahl. Compositional verification of a medical device system. In *ACM Int'l Conf. on High Integrity Language Technology (HILT) 2013*. ACM, November 2013.
20. Bashar Nuseibeh. Weaving together requirements and architectures. *Computer*, 34:115–117, 2001.
21. M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee. Model-driven safety analysis of closed-loop medical systems. *Industrial Informatics, IEEE Transactions on*, PP:1–12, 2012. In early online access.
22. A. Rajan, M.W. Whalen, M. Staats, W. Deng, and M.P.E. Heimdahl. The Effect of Program and Model Structure on the Fault Finding Ability of MC/DC Test Suites. In *Proceedings of Int'l Symposium on Software Testing and Analysis (ISSTA)*, Submitted 2008. Available at http://crisys.cs.umn.edu/ISSTA08.pdf.
23. SAE. http://www.aadl.info/aadl/downloads/papers/aadllanguagesummary.pdf.
24. L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang. Cyber-physical systems: A new frontier. *Machine Learning in Cyber Trust*, pages 3–13, 2009.
25. Matt Staats, Gregory Gay, Michael W Whalen, and Mats P.E. Heimdahl. On the danger of coverage directed test case generation. In *15th Int'l Conf. on Fundamental Approaches to Software Engineering (FASE)*, April 2012.
26. M. Whalen, A. Murugesan, S. Rayadurgam, and M. Heimdahl. Structuring Simulink models for verification and reuse. In *Proceedings of the 6th Int'l Workshop on Modeling in Software Engineering*, 2014.
27. Michael W. Whalen, Andrew Gacek, Darren Cofer, Anitha Murugesan, Mats P.E. Heimdahl, and Sanjai Rayadurgam. Your what is my how: Iteration and hierarchy in system design. *Software, IEEE*, 30(2):54–60, 2013.