Linking abstract analysis to concrete design: A hierarchical approach to verify medical CPS safety*

Anitha Murugesan[1]
anitha@cs.umn.edu

Oleg Sokolsky[2]
sokolsky@cis.upenn.edu

Sanjai Rayadurgam[1]
rsanjai@cs.umn.edu

Michael Whalen[1]
whalen@cs.umn.edu

Mats Heimdahl[1]
heimdahl@cs.umn.edu

Insup Lee[2]
lee@cis.upenn.edu

[1]Department of Computer Science and Engineering
University of Minnesota
200 Union St., Minneapolis, MN 55455, USA

[2]Department of Computer and Information Science
University of Pennsylvania
3330 Walnut St., Philadelphia, PA 19104, USA

ABSTRACT

Complex cyber-physical systems are typically hierarchically organized into multiple layers of abstraction in order to manage design complexity and provide verification tractability. Formal reasoning about such systems, therefore, necessarily involves the use of multiple modeling formalisms, verification paradigms, and concomitant tools, chosen as appropriate for the level of abstraction at which the analysis is performed. System properties verified using an abstract component specification in one paradigm must then be shown to logically follow from properties verified, possibly using a different paradigm, on a more concrete component description, if one is to claim that a particular component when deployed in the overall system context would still uphold the system properties. But, as component specifications at one layer get elaborated into more concrete component descriptions in the next, abstraction induced differences come to the fore, which have to be reconciled in some meaningful way. In this paper, we present our approach for providing a logical glue to tie distinct verification paradigms and reconcile the abstraction induced differences, to verify safety properties of a medical cyber-physical system. While the specifics are particular to the case example at hand – a high-level abstraction of a safety-interlock system to stop drug infusion along with a detailed design of a generic infusion pump – we believe the techniques are broadly applicable in similar situations for verifying complex cyber-physical system properties.

Keywords

Compositional verification, Model-based development, Medical cyber-physical systems

1. INTRODUCTION

Modern medical systems increasingly involve an array of interacting devices that communicate, coordinate and control therapy delivery to patients. Critical safety and efficacy properties of these cyber-physical systems (CPS) are dependent not only on the individual components' behaviors, but also on the overall CPS architecture that determines how these components are arranged and interconnected. This system decomposition, when formalized, can be profitably exploited in stringing together individual components' properties to establish top-level system properties. However, verification at the system level is likely to heavily abstract component behaviors down to what is essential for showing the system-level property. The component suppliers' responsibility is then to show that a concrete realization of a particular component indeed conforms to its abstract specification used to establish critical system properties.

Such a top-down compositional approach to verification is quite appealing – allocation of verification burden parallels allocation of development burden. But, as we descend down the system hierarchy, certain practical concerns surface that complicate the picture. First, different specification formalisms, modeling notations, and verification tools – the ones most appropriate for the component being verified – are likely to be employed, which necessitates some logical glue to attach the verification result of a lower-level component to its

*This work has been partially supported by NSF grants CNS-0931931 and CNS-1035715.

abstract higher-level representation. Secondly, as details get progressively elaborated, abstraction induced differences come to the fore which have to be somehow reconciled.

In this paper, we describe such a compositional approach to verify safety properties of a closed-loop generic patient-controlled analgesia infusion system. This prototypical system includes a patient blood oxygen level monitor whose output is constantly monitored by a supervisory controller that can automatically command the pump to terminate (pain medication) infusion, should the oxygen-level fall below a critical threshold.

In prior works [23, 21], we have separately and independently described two approaches to respectively verify (i) safety properties of the closed-loop system using timed automata models in UPPAAL that included a physiological model of the patient in the loop, and (ii) critical requirements of the infusion pump control software using a compositional assume-guarantee reasoning approach on an AADL system architecture with component behaviors elaborated in Simulink/Stateflow models. A natural question then is whether these two can be combined in some meaningful way such that the particular infusion pump when used as part of closed-loop system can be guaranteed to uphold critical safety properties. We describe our present work here that affirmatively answers this question. We highlight the key aspects of our approach for providing a logical glue to tie distinct verification paradigms and for reconciling the abstraction induced differences between the closed-loop system and the infusion pump system. While the specifics are particular to the case example at hand, we believe the approach is broadly applicable in similar situations for verifying complex CPS system properties.

The rest of the paper is organized as follows. In Section 1 we provide a brief overview of the the medical system case study and the Generic Patient Controlled Analgesic (GPCA) infusion pump. In Section 3 we describe the details of the hierarchical system modelling using different tools and notations - timed automata for the top level medical system, architectural design language to represent the hierarchical architecture of the system, and stateflow notation for modeling behavior of the GPCA components. In Section 4 we explain the individual model verification using tools appropriate for that modeling notation. We then demonstrate a formal hierarchical system verification of the medical system using the GPCA as a component. In the process of formally bridging the differences, we followed certain strategies and some limitations of the approach that we discuss in Section 5. Following a brief discussion of related work in Section 6 we finally conclude this paper, in Section 7, by summarizing our efforts, learning and our next steps in this research direction.

2. SYSTEM OVERVIEW

2.1 Closed Loop System

We considered the clinical scenario of Patient Controlled Analgesia (PCA). PCA is an approach to pain control in post-operative patients, widely used in modern hospitals. A liquid pain medication, typically an opiate such as morphine, is delivered by an infusion pump that is equipped with a request button. The pump delivers medication at a pre-defined low rate of infusion, known as basal rate. Often, the basal rate is insufficient to control pain. When the patient feels pain, he or she can press the request button. The pump then delivers an additional dose of medication, known as a bolus, at a higher rate of infusion.

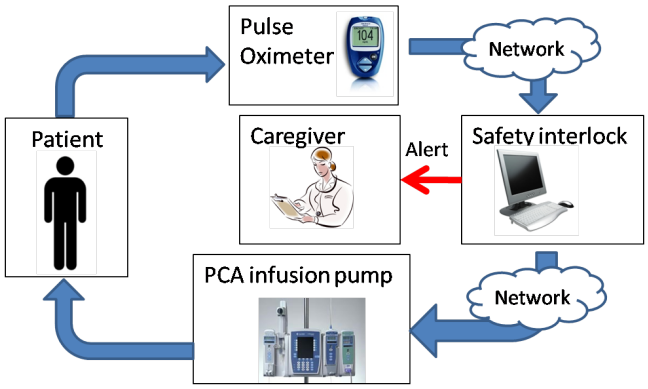

Figure 1: Overview of the closed-loop system

A serious side effect of opioid medication is that an overdose can lead to a respiratory failure that can result in the death of a patient. Sensitivity to the medication varies widely within the patient population, and can also be affected by the patient state. Therefore, the right balance between achieving pain control and avoiding overdose is hard to achieve. For this reason, PCA guidelines require the use of a sensor, typically a pulse oximeter, that would allow the clinician to detect the onset of respiratory problems. Pulse oximeters measure SpO_2, the blood oxygen saturation. Low oxygen saturation indicates that the patient is breathing strongly enough to supply lungs with oxygen. In the current practice, the caregiver manually adjusts the pump settings after the pulse oximeter raises an alarm. As the caregiver is typically responsible for a number of patients, there may be a delay before he or she responds to an alarm. For this reason, the quality of care may be improved by a closed-loop system illustrated in Figure 1, where a safety interlock device continuously monitors the pulse oximeter readings and stops the pump once a pre-set threshold is crossed.

2.2 Generic Patient Controlled Analgesic Infusion Pump

Infusion pumps have been involved in many incidents that have resulted in harm to the patient [1]. The US Food and Drug Administration (FDA), through its Infusion Pump Improvement Initiative [1] has sought to pro-actively promote the safe use of these devices by establishing additional requirements for infusion pump manufacturers.

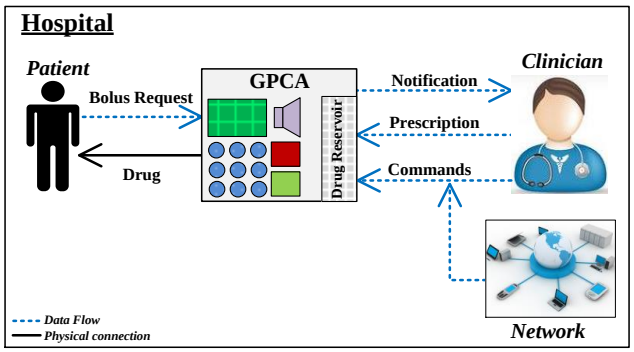

Figure 2: GPCA Device

In order to contribute to this initiative by providing an archetype of system development artifacts for a Generic Patient Controlled Analgesia Infusion Pump (GPCA) system, we modeled and verified an elaborate generic PCA infusion pump, as shown in Figure 2. Unlike the abstract PCA pump in the closed loop system, the GPCA we considered is richer in functionality, such as multiple modes of drug infusion, prescription validation, drug library, variety of system response to hazards depending upon their severity, visual and aural notifications, logging capability, routine system checks etc. At a high level, the GPCA primarily has three functions (1) Deliver the drug based on the prescribed schedule and patient requests, (2) Prevent hazards that may arise during its usage by monitoring and notifying the clinician of any hazardous conditions encountered and (3) Respond to the commands received.

3. MODELING

Systems are naturally constructed in hierarchies and models enormously help in the development process especially for critical systems. At higher levels of abstraction the concentration of modeling is to specify and analyse the overall system functions and the component details are heavily abstracted. But as the system is refined, the focus shifts from the overall system functions to the particular component functions. The notations used to represent the system at the higher levels of abstraction may not be appropriate at lower levels. Also some components of the system may be readily available to be reused. Hence it is natural to choose different notations to model and analyse the system at different layers of abstraction. However, in most cases the modelling and analysis efforts are kept isolated within the layers of abstraction and there is often an informal notion of correspondence. In this section we describe our approach of modeling systems at different levels of abstraction, starting from the closed loop system till the software of the GPCA, using varied modeling notations and our attempt to formally relate them, so that their analysis can be hierarchically connected.

3.1 Closed Loop System Modeling

The key patient safety questions that need to be answered for the closed loop system is whether the pump will always be stopped, and whether it will be stopped quickly enough. There are several delay factors in the control loop. First, human physiology imposes a delay from the time the drug concentration reaches a dangerous level and the time the effects of an overdose becomes apparent in the oxugen saturation levels. In modeling the physiology, we utilize a simplified model of pharmacokinetics of intravenous delivery of anesthetic drugs presented in [5]. Second, there is a delay in the pulse oximeter, which averages a number of samples in a variable-size window to produce a reading [6]. Sensed values and pump commands are transmitted over the network, adding to the delay. Finally, the controller itself adds a processing delay to issue a command to the pump.

In order to capture the essential functionality of the closed-loop system and perform analysis of timing properties of the system, we modeled the system components and interactions between them using UPPAAL [4]. An UPPAAL model is a collection of timed automata that can communicate using synchronous channels and shared variables. The architecture of the UPPAAL model follows closely that of the system in Figure 1. The PCA infusion pump is represented by three automata, which we discuss in more detail below. The model also includes an automaton representing patient physiology, an automaton for the pulse oximeter, an automaton capturing the logic of the safety interface, and a network automaton. For details of the model, we refer the reader to [23].

The three automata used to model the pump that correspond to the three essential properties of the pump. The first property states that *Infusion cannot begin until parameters of the infusion are configured and pump is started by the caregiver.* The second property specifies basal rate delivery. It states that *if the pump is not stopped, it delivers medication with at least the basal rate and otherwise the infusion rate is 0.* Finally, the third property specifies bolus delivery. It states that,

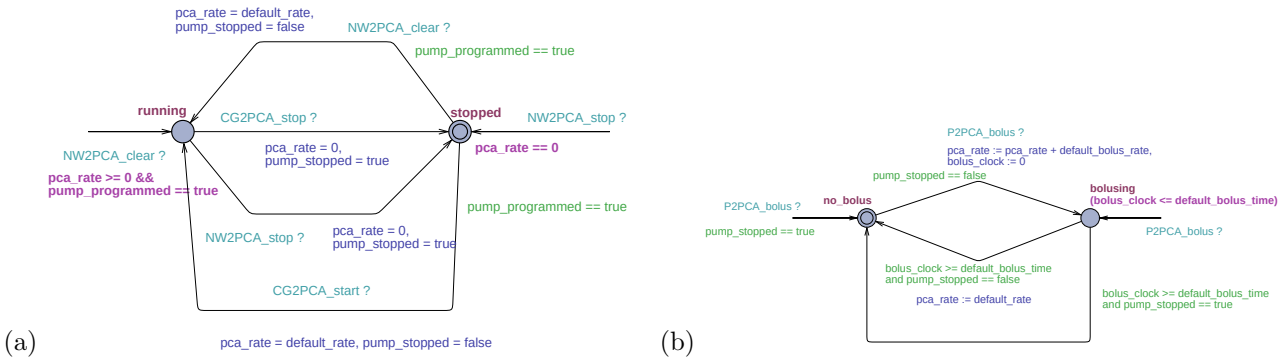

Figure 3: Infusion pump model in UPPAAL: (a) Basal infusion control (b) Bolus control

once the bolus is requested, medication is delivered at the bolus rate for the prescribed duration. The timed automata corresponding to the latter two properties are shown in Figure 3.

3.1.1 GPCA Architectural Modeling

As a part of another related initiative, we developed and analysed a generic PCA (GPCA) pump software model that is functionality superior to the PCA considered in the closed loop system model.

In the work described in this paper, our goal was to perform formalized hierarchical reasoning of the closed loop system as it refined to a functionally rich infusion pump model having the GPCA software that was already modeled. However, in order to reason in a hierarchical fashion, we need a hierarchical model, such that the abstractions are captured in the hierarchy. Hence we built an architectural model of an infusion pump using AADL. AADL notation supports descriptions of both hardware and software components and their interactions. AADL supports many of the constructs needed to model embedded systems such as processes, threads, devices (sensors and actuators), processors, buses, and memory. Furthermore, it contains an extension mechanism (called an *annex*) that can be used to extend the language to support additional features, such as requirements modeling. Also the GPCA software model was already modeled using AADL. Hence we choose AADL to build the hierarchical architectural model of an infusion pump.

The graphical representation of the architecture is shown in Figure 4. The top most system in AADL, `PCA_Pump`, have interfaces (inputs and outputs) that match the PCA interfaces of the closed loop system model (UPPAAL model). This was done so that there the level of abstraction of the `PCA_Pump` in AADL is same as the PCA of the closed loop system in UPPAAL and the properties of the `PCA_Pump` in AADL are always specified at the same level of detail as in the UPPAAL model.

While modeling the interfaces we group the individual inputs and outputs based on different sources: such as the `PCA_Pump` inputs, all infusion values from caregiver/clinician are grouped as `CAREGIVER_IN`; all commands to the pump are grouped as `PUMP_CMDS_IN` and patient bolus inputs are grouped as `PATIENT_IN`. This was done in order to simplify signal routing throughout the model. This grouping of inputs was consistently done for all components in all levels of system abstraction.

The architecture of the `PCA_Pump` component, consists of typical infusion device components such as sensors `(GPCA_HW_Sensors)`, a controller/software component `(GPCA_SW)` and actuators `(GPCA_HW_Actuators)`. The sensor component represents the device hardware that receive inputs to the device and sense exceptional conditions such as user interface, flow rate sensor etc. The actuator component represents the device hardware that is responsible for the drug flow out of the device and the device display. The software component is responsible for the system control. Since we already modeled a detailed software model, we reused the `GPCA_SW` model. In addition to the typical device components, we also created interface components (`GPCA_IP_Interface` and `GPCA_OP_Interface`) to match and route the inputs and outputs between the abstract `PCA_Pump` and concrete GPCA device.

From this point onwards, we refer to the high level `PCA_Pump` as the abstract model and detailed GPCA device model as the concrete model.

The concrete model has more inputs than the abstract `PCA_Pump` model hence the interface component matches some straightforward inputs, such as `Pump_Stop` of the abstract model to `Infusion_Cancel` of the concrete model, as well as suppress certain concrete model inputs so that the behaviours of the concrete model not required by the abstract model never occur. For example, the concrete model implements a functionality of suspending infusion in response to a `Infusion_Pause` command

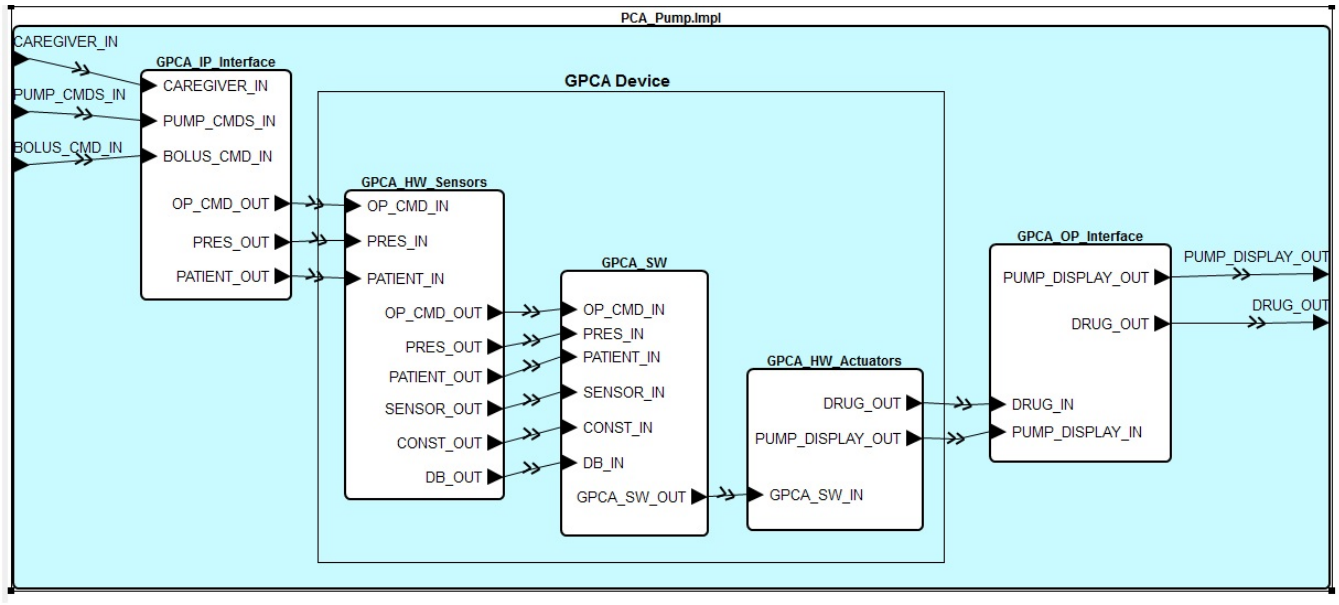

Figure 4: GPCA Device Architecture

from the clinician. However the abstract model does not specify this functionality. Hence the interface suppresses the `Infusion_Pause` command to the device. Similarly other inputs and system conditions of the concrete model that are not specified by the abstract model are abstracted by the interfaces.

3.1.2 GPCA Software Architecture

As mentioned earlier in the paper, we reused an existing model of the software (`GPCA_SW`). The software component (`GPCA_SW`) was by far the most complicated component. Hence developing it as one monolithic component was difficult to maintain as well as its verification was not scalable [21]. Hence it was decomposed further into sub-components as shown in Figure 5. To remove clutter from the figure, we do not show connections from the sub-components to the component boundary; the inputs and outputs of the subsystems are connected to the inports and outports of the system with the same name. This decomposition of the software was specified as an implementation of the `GPCA_SW` component similar to the decomposition of `PCA_Pump`. Once the `GPCA_SW` was decomposed into sub-components, it was tractable for analysis hence no further decomposition was necessary. For details of the model, we refer the reader to [21].

3.2 Behaviorial Modeling

The architectural models just defines the structure of components and their connections. They do not specify anything about the components behaviors. However, it was necessary to understand, analyse and verify behaviors elaborately. In our opinion, AADL is generally not the most suitable and preferred notations to model system behaviors. Hence for each of software's sub-component we modeled its detailed behavior using MathWorks Simulink [2] and Stateflow [3] tool. Simulink is a data flow graphical language for modeling and simulating dynamic systems (both the language and the tool are referred to as Simulink). Stateflow is a state-based notation used to model state machines and flow charts (again, Stateflow also refers to the tool). These tools are by far the most widely used notations in industry and suits our modeling needs well. These individual behavioral models allowed us to analyse and verify the behaviors in a manageable fashion. For details of the model, we refer the reader to [21].

4. VERIFICATION

Our strategy for verifying properties over models that span multiple abstraction levels is depicted in Figure 6 using the closed-loop infusion system as the case example. The main idea is to use the notation and tools appropriate for each abstraction level, decompose the single verification problem into distinct verification tasks for each level of abstraction, and finally tie together the results in a logical fashion that is amenable to manual review.

4.1 Verification Approach and Tools

For the verification effort of the GPCA model, we used three different model checking tools. For the timed automata model verification, we used UPPAAL model checker. For the architectural verification, we used the

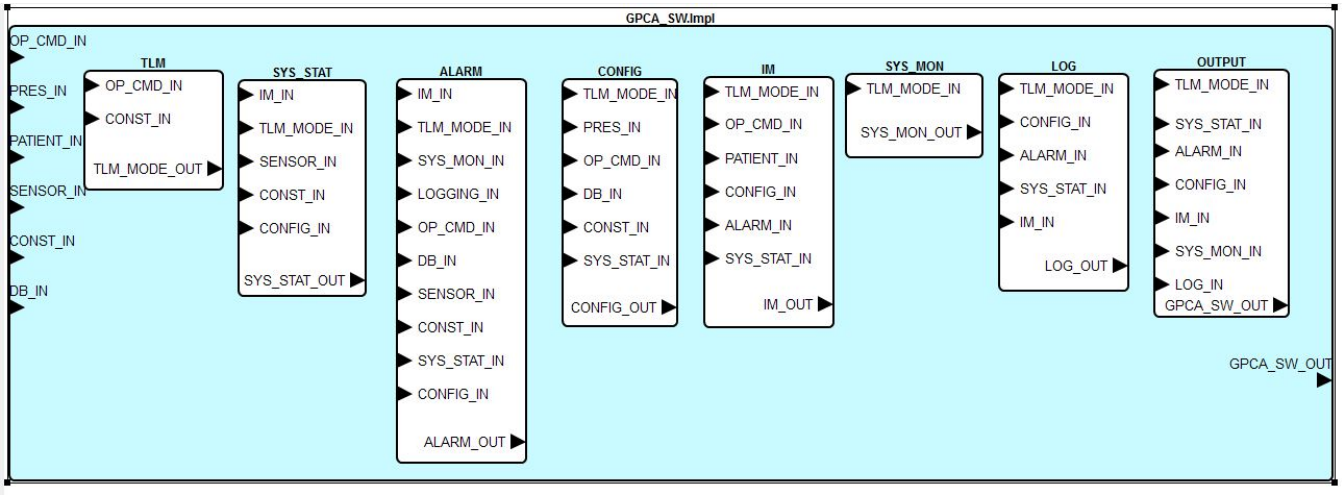

Figure 5: GPCA Software Architecture

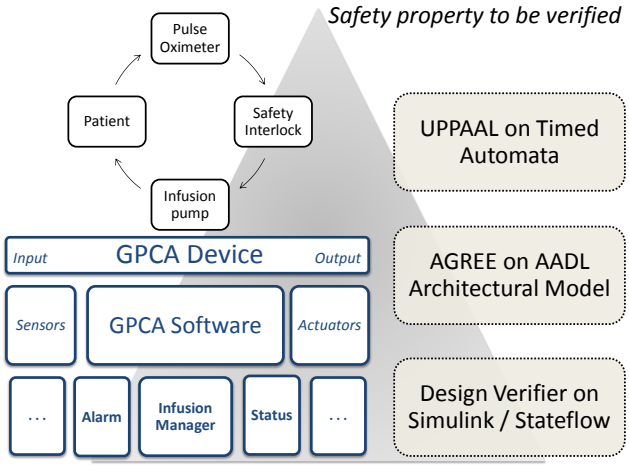

Figure 6: Multi-model verification

recently developed Rockwell Collins JKind tool[1]. For the behavioral model verification, we used the Simulink Design Verifier (SLDV) [19]. Both the JKind and SLDV tools use k-induction [24] algorithms implemented on top of a Satisfiability Modulo Theories (SMT) solver [12] to reason about infinite-state models involving real (rational) numbers and bounded or unbounded integers.

4.2 Closed Loop System Verification

We proved two essential closed-loop safety properties. The first one states that, once the patient's blood oxygen saturation level becomes lower than 90%, the pump will eventually be stopped, expressed as an UPPAAL query as `samplebuffer < 90 -> PCAbasal.stopped`. Here, `samplebuffer` is a shared variable that represents the true oxygen saturation. Note that the invariant of the `stopped` state in the `PCAbasal` automaton is `pca_rate==0`, implying that no drug is entering the patient. The second property takes into account the fact that the stopped pump can be restarted manually by the caregiver. We show that if the caregiver never restarts the pump when the patient is in danger, the patient eventually recovers, with the oxygen saturation levels returning to normal. This property is expressed as an UPPAAL query as `samplebuffer < 90 -> samplebuffer >= 91`.

[1]Available at: https://github.com/agacek/jkind

4.3 Reasoning about GPCA Architectural Models with AGREE

AADL is supported by a growing number of tools, including tools that support editing and import/export of AADL models, as well as tools that allow one to analyze different aspects of the model—correctness of the connections, component resource usage within limits, etc. However, AADL does not have a built-in means of associating requirements with different components within the architecture, nor does it have support reasoning about requirements.

To formally argue that the system satisfies its requirements, assume-guarantee contracts [20] provide an appropriate mechanism for capturing the information needed from other modeling domains to reason about system-level properties. In this formulation, guarantees correspond to component requirements, and assumptions correspond to the environmental constraints that are used in verifying the component requirements. A contract specifies precisely the information that is needed to reason about the component's interaction with other parts of the system. Furthermore, the contract mechanism supports a hierarchical decomposition of the

verification process that follows the natural hierarchy in the system model.

In our work, we use AGREE framework [8] - a compositional reasoning framework based on assume-guarantee contracts. AGREE is a plugin to the OSATE AADL tool and adds support for requirements capture and formal verification of the architectural models. In AGREE, we use the past-time operator subset of *past-time linear temporal logic* (PLTL) [17] to specify contracts. The language is based on Property Specification Language (PSL) [16] and defines a Lustre language [14] flavor for the PSL Boolean layer expressions and definitions.

AADL distinguishes between a *system*, which describe the input/output interface of an AADL aggregate, and *system implementations*, which describe the internal structure of the system. Each system type may have several implementations. We define requirements contracts in a *system* because requirements are defined over the input/output interface of the component and should not be defined in terms of implementation details. However, we perform proofs at the *system implementation* level, where we can use the contracts of subcomponents and their architectural relationship to establish system level properties.

For each layer of the architecture, we establish, for each implementation of a system, that the implementation meets the requirements of the system defined in the layer. Transitively, we thus establish that the requirements of the top-level system are proved given that the properties of the lowest layer leaf-level components are true. The structure of contracts is the same for the subcomponents, though of course the interfaces and properties are specialized to the functionality of each subcomponent.

In order to formally link the closed loop PCA to the architectural models, we recaptured the closed loop PCA properties as AGREE contracts at the `PCA_Pump` level in AADL.

Lets take an example to explain the hierarchical steps of verification. A closed loop level property for the PCA informally states, *Infusion cannot begin until parameters of the infusion are configured and pump is started by the caregiver*. This requirement recaptured as AGREE contract is,

```
property no_infusion_start  =
  true -> pre(PUMP_DISPLAY_OUT.Current_System_Mode)=1
    and (not(CAREGIVER_IN.Infusion_Start
      and CAREGIVER_IN.Infusion_Programmed)) =>
        (PUMP_DISPLAY_OUT.Current_System_Mode = 1);}
```

In the above formalism, the `Current_System_Mode = 1` represents the current pump state (abstractly represented as mode 1) in which flow rate is always zero. The `pre(PUMP_DISPLAY_OUT.Current_System_Mode)` represents the value of the variable from the previous step. An astute reader might have noticed the "true->" in the property, which means the property is always true in the first step. This was intensionally done, since we noticed that using 'pre' in the first execution step returned random value. . Hence in order to avoid complication in specifying the property, we split the property into two, where we first specify that the pump always starts (that at the first execution step) in state at which flow is zero and then starting the second step, the above mentioned property holds. This was made just to make the property writing straightforward.

In order for the above property of `PCA_Pump` to hold, its components, should compositionally guarantee using their properties. The `GPCA_SW` being a component of `PCA_Pump` has much richer functionality than the functionalities specified by `PCA_Pump`. Hence state mapping between them was required to write `GPCA_SW` properties to guarantee `PCA_Pump` properties. Figure 7 shows a high level representation of the states and its mapping between the systems. The state representation of `PCA_Pump` was derived from the UPPAAL model and that of `GPCA_SW` was derived from its behavioral specifications. Similar to the state mapping, the transitions from and to these states were also mapped. We discuss the details of the mapping later in this paper.

In order to satisfy the `PCA_Pump` property, certain assumptions were required at the `PCA_Pump` level to match the abstractions. First of all, `CAREGIVER_IN.Infusion_Programmed` - specifies if a valid prescription is already programmed in the pump - was mapped to the concrete model variable `(PUMP_DISPLAY_OUT.Configured` > 0`)` that means the device holds a valid prescription. At the `PCA_Pump` level, these were still abstract concepts since the caregiver decides if the infusion is programmed by looking at the pumps's display and determining if it is configured. Hence we state this as an assumption at the `PCA_Pump` level.

Secondly, we mapped the `ON->Programmed` and `Stopped` of the abstract model to `IDLE` state of the concrete model hence the transitions `(ON->Programmed)->Running` triggered by `(CAREGIVER_IN.Infusion_Start and CAREGIVER_IN.Infusion_Programmed)` and the transition `Stopped -> Running` triggered by `(CAREGIVER_IN.Clear_Command)` were assumed to imply each other.

Thirdly, GPCA_IP_Interface maps the abstract model's `Infusion_Start` input signal with the concrete model's `Infusion_Initiate` command. Finally the `GPCA_SW` guarantees that,

```
property no_enter_therapy =
  true ->
    pre(GPCA_SW_OUT.Current_System_Mode) = 1
      and not(OP_CMD_IN.Infusion_Initiate
        and GPCA_SW_OUT.Configured > 0) =>
          (GPCA_SW_OUT.Current_System_Mode = 1);
```

There were a few other component guarantees that

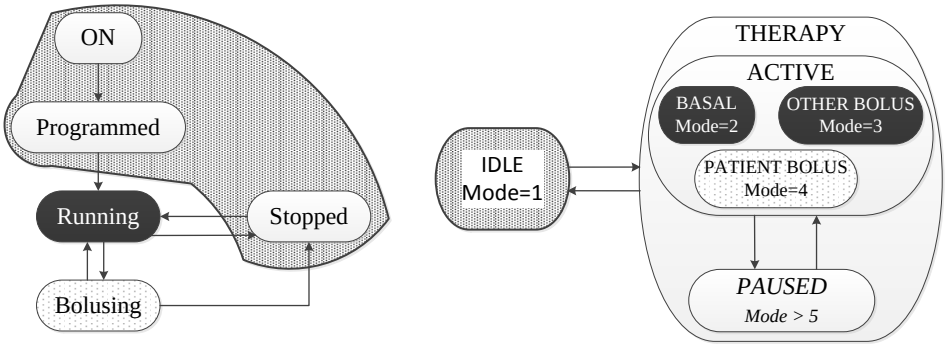

Figure 7: Abstract PCA and Concrete GPCA High level system state mapping

deal with mapping inputs and outputs that are not discussed here since its not a significant point of discussion. Hence when we compositionally verified the `PCA_Pump` property in AGREE, it was guaranteed using their component guarantees and system assumptions.

At this point, the satisfaction of the properties were only until the GPCA Device components level. As discussed in section 3.1.2, the `GPCA_SW` was further decomposed into components. Hence the properties guaranteed by the `GPCA_SW` for the `PCA_Pump` has to be guaranteed by its sub-components. We performed another level of AGREE reasoning in which now the `GPCA_SW` is the top level system that has properties to be guaranteed (similar to the `PCA_Pump`) by its components (`TLM, ALARM, IM...`) guarantees. For example, the property of the `GPCA_SW` (`property no_enter_therapy`)), is guaranteed by Infusion Manager(`IM`), Configuration Manager (`CONFIG`), and Output (`OUTPUT`) component guarantees. There were not as many assumptions and abstraction mapping required for the property guarantee between the `GPCA_SW` and its components since they were already done as a part of another initiative. For more details on the verification between the `GPCA_SW` and its components, we refer the reader to [21].

4.4 SLDV

The required properties of the software components captured in AGREE used to guarantee the `GPCA_SW`, which in turn were used to guarantee the `PCA_Pump`. However, in order to verify if the software components captured in AGREE indeed hold, we used the detailed behavioral model of each of these components built using Simulink/Stateflow and verified using Simulink Design verifier that they satisfy their properties. The Simulink Design Verifier requires all properties to be specified as Boolean expressions in one of the available MATLAB notations. Hence in our work, we recapture the required component properties in embedded Matlab - a subset of the MATLAB computing language that supports efficient code generation for deployment in embedded systems. For more details on the behaviorial modeling verification, we refer the reader to [21].

5. DISCUSSION

In total, 7 `PCA_Pump` properties were proved in AGREE that correspond to the abstract timed automata model of the PCA. In order to guarantee these 7 system properties, there were 7 `GPCA_IP_Interface` properties and 11 `GPCA_SW` properties along with some auxiliary properties of the `GPCA_HW_Sensors`, `GPCA_HW_Actuators` and `GPCA_OP_Interface` to pass through the input and output signals. Similarly the 11 `GPCA_SW` properties were in turn guaranteed by 4 properties in `ALARM`, 11 properties in `IM` and 6 properties in `CONFIG`, which are the sub-components of `GPCA_SW`.

5.1 Matching Abstractions

To ensure that the infusion pump used in the closed-loop system model *simulates* the concrete GPCA model, the differences in the two abstractions had to be reconciled by formally matching states, transitions and inputs. The concrete GPCA model has many inputs, states and transitions that are not represented in the abstract model. We addressed this by associating concrete infusion modes with abstract states of the infusion pump, introducing a non-deterministic input in the abstract model and adding environmental assumptions in the concrete model.

First we established a mapping between states in the abstract model of the infusion pump and the infusion modes of the concrete GPCA model, as show in Figure 7. This enables restating properties specified in terms of flow-rate constraints at the abstract level as properties in terms of the infusion modes. The different GPCA infusion modes have specific associated flow-rates that can be immediately verified for conformance to flow-rate constraints. In this mapping, the abstract states `Stopped`, `ON` and `Programmed` are asso-

ciated with the IDLE mode represented by the predicate Current_System_Mode = 1; Running is associated with BASAL and OTHER BOLUS modes represented by Current_System_Mode $\in \{2, 3\}$; and BOLUSING is associated with PATIENT BOLUS represented by Current_System_Mode = 4. Note that there is no abstract state associated with the PAUSED mode of the GPCA, which denotes a temporary halt to infusion due to certain exceptional conditions identified by the alarm sub-system. Certain environmental assumptions, to be discussed shortly, are sufficient to guarantee that this mode is not reached for the scenarios considered.

Second, the abstract closed-loop system modeled the "stop" command to the pump as an exclusive input from the supervisory control for oxygenation-level based safety-interlock. However, realistically the operator may command the GPCA to stop at any time – one may view the operator as an external monitor for exceptional conditions, analogous to the internal monitor represented by the ALARMS component. Unlike infusion pump alarms, which are excluded from consideration at the abstract level, the "stop" command was included in the original abstract model, albeit for a restricted purpose. By allowing the "stop" to occur non-deterministically, one can easily simulate an externally commanded infusion stoppage. The Pump_Stop input signal was made non-deterministic in the abstract model to effect this change.

The GPCA monitors various exceptional conditions as modeled by the ALARMS component and responds appropriately, some of which would include a temporary pause to infusion. These alarms may be triggered by sensor inputs such as *air-in-line* or internally computed conditions such as *empty-reservoir*. Due to this, even a simple abstract property, *Once the pump is started, unless it is stopped, the drug flows at least at basal flow rate*, does not hold for the concrete model, because, the property presupposes the absence of infusion-ending triggers other than the stop command. In the closed-loop system, the abstract infusion pump model rightly disregards such details; at the abstract level, one is interested in establishing essential system properties using a simplified notion of an infusion pump. The responsibility for stopping infusion for any reason can be pushed out to the environment – in this case the pump operator can be assumed to command infusion stoppage as necessary. Therefore, when relating abstract infusion pump properties to a concrete realization of the pump, it is reasonable to interpret those as being prefixed with a standard caveat: "Absent exceptional pump conditions that internally trigger stoppage of infusion ...". Of course, what these exceptional conditions are would vary with particulars of the specific pump under consideration. In the process of mapping the abstract infusion pump model and its properties to the concrete GPCA, absence of these internally triggered exceptional conditions is captured as a system assumption, (no_cancel_implies_no_stop_conditions). Such assumptions must then be discharged using properties of the environment in which the GPCA operates. In the present case, one has to manually review these assumptions to ascertain that those are acceptable caveats to the property of interest. This is a desirable outcome, for it clearly calls attention to what those exceptional conditions could be for the specific GPCA device.

Finally, in the concrete model, there are more more modes of active drug infusion than the ones specified in the abstract model as shown in Figure 7. Originally the abstract model's property was, *when the system is started and not stopped, system infuses at BASAL flow rate (when there is no patient bolus request)*. But the concrete model switches between BASAL and OTHER BOLUS based on the prescription. There were multiple ways to reconcile the differences. A straightforward approach is to assume that there is no prescribed OTHER BOLUS. However, this is not a reasonable assumption for the scenarios under consideration – the essential system properties must hold for any infusion scenario. Therefore, we adopted an alternative route: the abstract property was modified to state that system infuses at no less than the BASAL flow rate". With an assumption that any bolus infusion delivers drug at a rate higher than the BASAL flow rate, which is typically true of the problem domain, this change allows the possibility of multiple drug delivery modes with different flow rates even when the abstraction includes only a few specific modes explicitly.

Apart from these changes, some auxiliary properties were required to guarantee that the concrete model's states and transitions that were not represented by the abstract model are not reachable given the system assumptions and the abstraction mapping. For example, the concrete model has an additional state PAUSED shown in Figure 7. Thus, an auxiliary property to show that PAUSED is never reached under the given environmental assumptions is needed.

In summary, the goal of this matching process augmented with property verification is to justify the following claims:

1. Every abstract state, input and output in the abstract infusion pump model has some matching counterparts in the concrete model
2. States, inputs and outputs of *interest* in the concrete GPCA have some matching counterparts in the abstract model
3. Transitions between such matched states on matched inputs in the concrete GPCA model have a corresponding matching transition in the abstract model and the respective outputs match

```
-- UPPAAL Property :if the pump is not stopped, it delivers medication with at least the basal rate
property infusion_continue  =
   true -> (pre(PUMP_DISPLAY_OUT.Current_System_Mode) >= 2 and pre(PUMP_DISPLAY_OUT.Current_System_Mode) <= 4) and
            not(PUMP_CMDS_IN.Pump_Stop) =>
                (PUMP_DISPLAY_OUT.Current_System_Mode >= 2 and PUMP_DISPLAY_OUT.Current_System_Mode <= 4);
guarantee "infusion_continue":infusion_continue;

-- Map modes with flow rates.
guarantee "Mode 1":(PUMP_DISPLAY_OUT.Current_System_Mode = 1) => (DRUG_OUT.Drug_Flow_Rate = 0);
guarantee "Mode 2":(PUMP_DISPLAY_OUT.Current_System_Mode = 2) =>
                        (DRUG_OUT.Drug_Flow_Rate >= CAREGIVER_IN.Normal_Infusion_Rate);
guarantee "Mode 3":(PUMP_DISPLAY_OUT.Current_System_Mode = 3) =>
                        (DRUG_OUT.Drug_Flow_Rate >= CAREGIVER_IN.Normal_Infusion_Rate);
guarantee "Mode 4":(PUMP_DISPLAY_OUT.Current_System_Mode = 4) =>
                        (DRUG_OUT.Drug_Flow_Rate = CAREGIVER_IN.Bolus_Infusion_Rate);
....
....
-- Maps pump stop input of abstract model to Infusion cancel of concrete model
guarantee "Pump Stop means Infusion cancel" : (PUMP_CMDS_IN.Pump_Stop <=> OP_CMD_OUT.Infusion_Cancel);
....
....
-- sensor conditions that cause infusion to be stopped.
eq sensor_conditions_that_cause_pause:bool =
    (SENSOR_OUT.Battery_Depleted or SENSOR_OUT.RTC_In_Error or SENSOR_OUT.CPU_In_Error or
        SENSOR_OUT.Memory_Corrupted or SENSOR_OUT.Pump_Too_Hot or SENSOR_OUT.Watchdog_Interrupted or
            SENSOR_OUT.Temp or SENSOR_OUT.Humidity or SENSOR_OUT.Air_Pressure or
                SENSOR_OUT.Air_In_Line or SENSOR_OUT.Occlusion or SENSOR_OUT.Door_Open);

-- when there is no infusion cancel, then there is no sensor conditions causing infusion stop occurs.
property no_cancel_implies_no_stop_conditions =
        not(OP_CMD_IN.Infusion_Cancel) => not(sensor_conditions_that_cause_pause);
assume "no_cancel_implies_no_stop_conditions" :no_cancel_implies_no_stop_conditions;
....
....
-- If there is no infusion cancel then there is no condition stopping infusion.
property no_cancel_implies_no_stop_conditions =
  not(OP_CMD_IN.Infusion_Cancel) =>
                not(infusion_end_conditions or any_alarms or OP_CMD_IN.Infusion_Inhibit);
assume "no_cancel_implies_no_stop_conditions" :no_cancel_implies_no_stop_conditions ;

-- If the system is in ACTIVE, and if there is no conditions to stop infusion,system will be in ACTIVE.
property in_active =
    true ->
        (pre(GPCA_SW_OUT.Current_System_Mode) >= 2 and pre(GPCA_SW_OUT.Current_System_Mode) <= 4) and
            not(OP_CMD_IN.Infusion_Cancel) =>
                (GPCA_SW_OUT.Current_System_Mode >= 2 and GPCA_SW_OUT.Current_System_Mode <= 4);
guarantee "in_active":  in_active;
```

Figure 8: Portions of AGREE properties at different layers of system abstraction

4. Unmatched concrete inputs do not occur given certain environmental assumptions

5. Unmatched concrete states are not reachable under those environmental assumptions

Informally speaking, taken together these claims allow us to see the abstract infusion pump model as a stand-in (simulation) for the concrete GPCA model under the given environmental assumptions. Thus safety properties that are established for a system model that uses the abstract infusion pump as a component, in a way that satisfies the environmental assumptions made in the matching process, will be upheld when the concrete GPCA infusion pump is substituted in place of the abstract pump.

5.2 Limitations

Our current approach to verification is restricted in several ways. First, AGREE currently only handles synchronous architectural models in which execution proceeds in a deterministic discrete sequence of steps. Second, AGREE can verify only *invariants*, so liveness properties, cannot be specified in AGREE. In our experience, this is not as severe a limitation as it may seem, since most systems are concerned with *bounded liveness* in which an action must occur within a time interval that can be written in AGREE.

The current analysis tools use *rationals* to model the behavior of real numbers; However, most software is implemented using floating point numbers. This can

lead to unsoundness in our analysis of software that uses floating point arithmetic. Also, AGREE does not support trigonometric or non-linear functions. These can be approximated in some cases, but many of the interesting numeric properties of systems simply cannot be specified.

6. RELATED WORK

Formal analysis of medical systems.

Modeling and analysis of closed-loop medical systems have been primarily studied in the context of diabetes care. Much attention is given to modeling patient physiology and design of algorithms for glucose control; see, for example, [26, 7]. However, we are not aware of any studies, where evaluation of the closed-loop system is tied to the modeling of medical devices that comprise the system. A closed-loop safety interlock for PCA infusion, similar to the one studied in this paper, has been proposed in [9], however, the authors do not show any analysis results. Similarly, the GPCA pump has been used in a number of case studies that involved a variety of formal methods to model different aspects of the pump behavior. In [18], code for a simple infusion controller has been generated from UPPAAL-verified code.

Multi-formalism modeling.

In the context of multi-formalism modeling, work has been done to develop frameworks to integrate models specified using different formalisms. For example, The Möbius approach [11], provides a comprehensive infrastructure to support interacting formalisms and solvers. However the formalisms and solvers are required to be described in terms of a predefined framework and only those formalisms that can be mapped to the tool's underlying predefined common semantics can be used. Similarly, The OsMoSys approach [25] and AToM^3 [10], requires the formalism (restricted to graph based formalisms) to be defined in terms of an XML based meta-language. The main drawback of these approaches is requiring the formalism to be definable/mappable using a common semantics/rules. This restricts the freedom of the designer to the model in the notation that most suitable for representing and analysing the system and its components.

An important aspect of relating models in different formalisms is the notion of time. Timed automata used to model the closed-loop system use continuous time, while behaviors of the AADL model rely on discrete time. The usual way of reconciling the discrepancy between the notions of time is through discretization. A number of techniques for discretization have been developed in the literature, such as [13, 15]. Our UPPAAL model satisfies the conditions put forth in [22], which guarantee that whenever a transition is enabled during an execution, it can occur at an integer-valued time instance. Thus, the model can be faithfully simulated in discrete time.

7. CONCLUSION

When systems are composed from sub-systems, properties of the system must be assessed based on its component properties and the composition. To reason about such systems, no single analysis and modeling method can successfully cope with all aspects of a system or its components. Hence multiple notations and formalisms are used. An approach to logically glue the diverse analysis of the system and its components is required to reason about the system properties.

In this paper, we considered compositional verification of a medical system at multiple levels of abstraction, with different formalisms used at each level. We were able to formally glue distinct verification paradigms by leveraging the system's hierarchical architectural decomposition. We showed how properties proved for the system components at the lower levels of abstraction can be used to validate the more abstract models, ensuring that properties proved at the higher levels of abstraction remain satisfied.

While techniques used in this paper are specific to the example at hand and to the formalisms used within the case study, we believe that this work can form the basis for a general, scalable and practical approach to layered verification of properties in complex cyber-physical systems. In order to fully realize the promise of this approach, we are currently working to make the approach more systematic and eliminate the *ad hoc* aspects we used in this work. We are also making the approach more general, allowing more formalisms be incorporated.

8. ACKNOWLEDGMENTS

We thank Andrew Gacek, Darren Cofer and John Backes at Rockwell Collins for developing and improving JKind and AGREE. We also thank Miroslav Pajic for helpful discussions on restructuring the model of the closed-loop system.

9. REFERENCES

[1] Infusion Pumps. via the world-wide-web: http://www.fda.gov/InfusionPumps.

[2] Simulink - simulation and model-based design. via the world-wide-web: http://www.mathworks.com/products/simulink/.

[3] Stateflow - environment for modeling state machines. via the world-wide-web: http://www.mathworks.com/products/stateflow/.

[4] G. Behrmann, A. David, and K. 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.

[5] B. W. Bequette, editor. *Process control: modeling, design, and simulation*. Prentice Hall, 2nd edition, 2003.

[6] V. Chan and S. Underwood. A single-chip pulsoximeter design using the MSP430. Technical Report SLAA274, Texas Instruments, Nov. 2005.

[7] C. Cobelli, C. D. Man, G. Sparacino, L. Magni, G. D. Nicolao, and B. P. Kovatchev. Diabetes: Models, signals, and control. *IEEE Reviews in Biomedical Engineering*, 2:54–96, 2009.

[8] D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional verification of architectural models. In A. E. Goodloe and S. Person, editors, *Proceedings of the 4th NASA Formal Methods Symposium (NFM 2012)*, volume 7226, pages 126–140, Berlin, Heidelberg, April 2012. Springer-Verlag.

[9] P.-A. Cortes, S. M. Krishnan, I. Lee, and J. M. Goldman. Improving the safety of patient-controlled analgesia infusions with safety interlocks and closed-loop control. In *Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, pages 149–150, 2007.

[10] J. De Lara and H. Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling. In *Fundamental approaches to software engineering*, pages 174–188. Springer, 2002.

[11] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius framework and its implementation. *Software Engineering, IEEE Transactions on*, 28(10):956–969, 2002.

[12] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts)*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.

[13] A. G6ll, A. Puri, and P. Varaiya. Discretization of timed automata. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 957–958, 1994.

[14] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[15] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proceedings of ICALP 1992*, volume 623 of *LNCS*, pages 545–558, 1992.

[16] IEEE. *IEEE Std. 1850-2005. Property Specification Language (PSL)*. IEEE, 2005.

[17] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.

[18] B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. Jones, Y. Zhang, and R. Jetley. Safety-assured development of the gpca infusion pump software. In *Proceedings of EMSOFT*, 2011.

[19] Mathworks Inc. Simulink Design Verifier. Via the world-wide-web: http://www.mathworks.com/products/sldesignverifier.

[20] K. L. McMillan. Circular compositional reasoning about liveness. Technical Report 1999-02, Cadence Berkeley Labs, Berkeley, CA 94704, 1999.

[21] A. Murugesan, M. W. Whalen, S. Rayadurgam, and M. P. Heimdahl. Compositional verification of a medical device system. In *ACM International Conference on High Integrity Language Technology (HILT) 2013*. ACM, November 2013.

[22] 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 *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2012)*, Apr. 2012.

[23] 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.

[24] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In W. A. H. Jr. and S. D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.

[25] V. Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis. The osmosys approach to multi-formalism modeling of systems. *Software and Systems Modeling*, 3(1):68–81, 2004.

[26] S. Weinzimer, G. Steil, K. Swan, J. Dziura, N. Kurtz, and W. Tamborlane. 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, May 2008.