Modal Abstraction View of Requirements for Medical Devices Used in Healthcare Processes

Heather M. Conboy
School of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
Email: hconboy@cs.umass.edu

George S. Avrunin
School of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
Email: avrunin@cs.umass.edu

Lori A. Clarke
School of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
Email: clarke@cs.umass.edu

***Abstract*—Medical device requirements often depend on the healthcare processes in which the device is to be used. Since such processes may be complex, critical requirements may be specified inaccurately, or even missed altogether. We are investigating an automated requirement derivation approach that takes as input a model of the healthcare process along with a model of the device and tries to derive the requirements for that device. Our initial experience with this approach has shown that when the process and device involve complex behaviors, the derived requirements are also often complex and difficult to understand. In this paper, we describe an approach for creating a modal abstraction view of the derived requirements that decomposes each requirement based on its modes, and thus appears to improve understandability.**

***Index Terms*—Requirement specifications, medical devices, healthcare processes, model checking, learning algorithms, modes.**

I. Introduction

Medical devices may be used in complex healthcare processes, which often involve concurrent activities and exceptional situations. Because of this complexity, it may be challenging to determine the device requirements accurately. Since healthcare processes may be life critical, such inaccuracies can have devastating consequences.

To address this problem, we are investigating an approach that automatically derives the device requirements from a model of the overall healthcare process along with the requirements for that process. The overall process model is a combination of a simple model of the device's behaviors and a model of how that device is used in the process. The derived requirements for the medical device, when adhered to, are sufficient to prevent any violations of the overall process requirements.[1] The derived requirements could then be used to suggest modifications to the behavior of the device, the process, or both to ensure that the overall process satisfies its requirements.

To automate the reasoning about an overall process model's behaviors and its requirements, our requirement derivation approach builds on interface synthesis methods (e.g., [1]), that were originally developed for software systems. This approach iteratively derives the requirements using model checking (e.g., [2]) and learning algorithms (e.g., [3]). Our initial work implemented a toolset to support this approach and evaluated it on two small case studies from the healthcare domain [4]. We observed that although the derived requirements provided useful insights about the devices, as the overall processes increased in complexity the requirements became less understandable. We also observed, however, that medical devices are often *modal*, meaning the device enables different behaviors depending on a few input settings. Thus, we are currently investigating the creation of a *modal abstraction view* of the derived requirements. This view decomposes the derived requirements based on the modes and, thus, seems to improve their understandability. This paper presents a high-level overview of our requirement derivation approach and the modal abstraction view.

The example shown throughout this paper is an infusion pump used to administer intravenous fluids and medications. A "smart" infusion pump would be programmed with a set of drug libraries giving the concentrations, dosing units, and dosing limits for the drugs in use in each particular area of the hospital. For instance, the drug library for an operating room typically allows a wider range of dosing limits than an intensive care unit. The clinician using the pump selects the drug, concentration, etc. and the pump alerts the clinician if the dose exceeds the limits in the library. One important requirement for any process in which an infusion pump is used is that a patient never be administered a drug overdose. This might be reflected as a requirement on the pump that if the selected dosage is outside the range, the pump must not administer that dosage to the patient. If the pump developers do not consider that a patient connected to a pump could be moved from one area in the hospital to another, then the device requirement that the pump must be reconfigured when it is moved might be overlooked. Or, perhaps more likely, even after carefully considering such alternative usages, the device requirement might contain some subtle errors. Note that an infusion pump is modal since the current drug library determines when the pump will issue alerts.

The remainder of this paper is organized as follows. The process-based requirement derivation approach and toolset are discussed in Section II, and the modal abstraction view is

[1]Although in this paper the approach is described from the perspective of deriving requirements for the device, this approach could also be applied from the perspective of deriving requirements for the process.

discussed in Section III. Section IV provides an overview of previous work on the derivation and visualization of requirements, and Section V summarizes our contributions and discusses some possible directions for future work.

II. Background

Our process-based requirement derivation approach takes as input an overall process model and one or more of its requirements. The overall process model is composed of a device model and a process model in which that device is used. The simple device model, however, allows a wide range of device behaviors, perhaps even some behaviors that violate the overall process requirements. Thus, this approach tries to produce a derived device requirement that restricts the behaviors of the device to prevent any violations of the overall process requirements. The ensuing paragraphs describe how we specify requirements, model processes, and derive requirements with an interface synthesis method that employs model checking and learning algorithms.

Requirements are commonly specified in natural language. To be used as the basis for formal reasoning, however, they need to be expressed in a mathematical formalism such as temporal logic or finite state automata (FSAs). Such formalisms are expressive enough to capture the interactions between a device and healthcare process. We chose to specify the requirements as FSAs because device developers are often familiar with FSAs and the derivation could be automated with a regular language learning algorithm such as the L^* algorithm [3], [5].

In the healthcare domain, special purpose notations have been developed to model medical guidelines and protocols (e.g., [6]). These healthcare process models are usually expressive enough to easily capture the normal behaviors but can be cumbersome when expressing aspects such as exceptional situations and intercomponent communications. Moreover, many of these healthcare process models do not have precisely defined semantics so they cannot be formally analyzed. For our requirement derivation work, we model the healthcare processes in the Little-JIL process modeling language [7] because it is both expressive and precisely defined.

Our process-based requirement derivation approach builds on interface synthesis methods. Such methods take as input a software component and a requirement of that component. These methods output an interface that captures the most general way to use that component without violating the given requirement. While some interface synthesis methods use a combination of learning and model checking (e.g., [1], [8]), as does our approach, other interface synthesis methods have been developed that are based on game theory or counterexample guided abstraction refinement (e.g., [1]). In theory, such methods have the same worst case complexity. But in practice, these methods vary widely with regard to the performance (in terms of space and time) and the requirements they derive (in terms of complexity). Our approach extends Beyer et al.'s interface synthesis method [1], which iteratively derives a requirement using the L^* learning algorithm and model checking. This method was selected because we have some expertise with this learning algorithm and could use the existing translation from Little-JIL to the FLAVERS model checker [2]. Our extension is similar to Giannakopoulou and Păsăreanu's extension [8] but differs in terms of the modeling language and model checker used. More details of the requirement derivation algorithm can be found in [4].

To illustrate the requirement derivation approach, consider the smart infusion pump described in the introduction. An important overall process requirement for any process that uses the infusion pump is to "never overdose" a patient. For this paper to clearly illustrate the proposed view, we describe a simplified pump example that involves only three modes, which will be enumerated in the next section. Our overall process model is composed of a pump model and an inpatient surgery process model in which the pump is used. The pump model allows a medical clinician to set the drug library for either the intensive care unit (ICU) or operating room (OR). Based on the current drug library, the pump will then issue alerts. The surgery process model first uses the pump in the OR and then may use the pump in the ICU. To use the pump, the clinician sets the drug library for the care area, enters a dose, and tries to start the pump. For the above "never overdose" requirement and overall process model, figure 1 shows the derived requirement as an FSA. This requirement informally states that the pump must be configured with the ICU drug library before that pump is used to administer infusions in the ICU, and similarly for the OR. We show this simple derived requirement to make the idea clear and for space considerations. When deriving requirements for examples larger than the example given here, we found that the resulting FSAs were often more difficult to understand.

To improve readability, the FSAs shown in the figures elide the violation state and all the transitions to that state. The violation state is a sink state, a non-accepting state with only self-loop transitions. Each derived FSA is total, meaning that every state has a transition for every event in the alphabet. Thus, whenever a state does not show a transition for a particular event, there is implicitly a transition on that event to the violation state. These violation transitions mean that the requirement disallows the occurrence of such an event.

Even with the elisions, the derived FSA can still be challenging to understand because the FSA provides a low-level representation of the interactions between the device and the healthcare process. Thus, we propose creating views of the derived requirements that incorporate high-level features such as abstraction or decomposition. This paper describes a *modal abstraction view* that decomposes the requirement based on its modes to improve understandability.

III. Modal Abstraction View

The modal abstraction view hierarchically decomposes a derived FSA into pieces based on the modes. Each piece describes a given mode's behaviors and the mode changes are represented as transitions among the pieces. For instance,

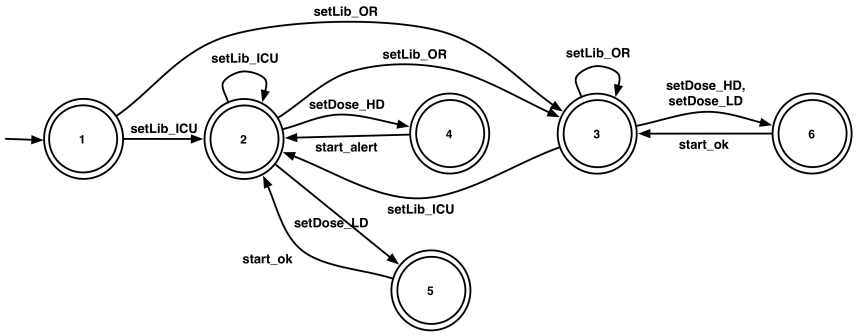

Fig. 1: Example derived requirement represented as an FSA

the pump behaviors can be decomposed into three modes: the pump not set for any drug library, one for the ICU drug library, and another for the OR drug library. Since viewers can now consider each piece separately, the cognitive load should be reduced. We developed a modal abstraction view tool to automatically create such a view.

To create the modal abstraction view, the user must specify the derived FSA and the mode change events. For the pump example, the mode change events are to configure the pump with either of the two drug libraries: $setLib\_ICU$ or $setLib\_OR$. Based on the mode change events, the derived FSA is automatically decomposed into a set of abstracted FSAs. Conceptually, there is one abstracted FSA to show the mode changes and one abstracted FSA for each mode to show that mode's enabled behaviors.

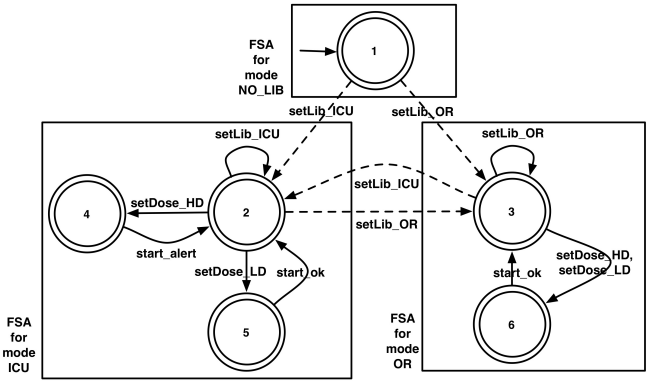

Fig. 2: Modal abstraction view of derived requirement

To illustrate, figure 2 shows the modal abstraction view of the derived requirement that was shown in figure 1. (This example is easy to understand without the modal abstraction, but space limitations prevent us from showing a larger example.) The figure shows the abstracted FSAs for each mode (designated as boxes for the subgraphs). In this figure, the no library mode is shown on the top and the ICU and OR modes are shown on the bottom. The abstracted FSA for the mode changes has three states corresponding to the subgraphs and the transitions among these three states annotated with the mode change events (designated as dashed lines). This view benefits from taking the modes in account because the larger FSA can be decomposed into smaller subgraphs and the mode changes can be highlighted.

Our modal abstraction view tool takes as input a derived FSA and a mapping from each mode change event to its corresponding mode. This tool essentially produces a mapping from each mode to the subgraph for that mode. The subgraph is represented by the set of states where the device is configured for that mode. This tool iterates through each state s in the FSA. If state s is the violation state or already contained in a subgraph, then the tool continues to the next state. When state s has an incoming transition labeled with a mode change event e, the tool looks up the corresponding mode m for event e and then creates a new subgraph g_m for mode m. First, subgraph g_m is created by finding all states reachable from state s by following outgoing transitions labeled with an event not in the set of mode change events. The tool next updates the output mapping to associate mode m with subgraph g_m. There is also special processing done for the start state of the FSA.

The resulting modal abstraction view seems promising. For the simplified pump example shown here, this view improves understandability in a small way. For the more complex examples mentioned in the background section, this view had a larger impact on understandability. To further investigate the modal abstraction view, the view could be extended to decompose each mode into sub-modes. Additionally, the modal abstraction view tool could highlight the states and transitions based on particular sub-processes (e.g., 'Administer ICU care' sub-process) or specific human participants (e.g., Nurse). The modal abstraction view currently builds on FSAs. We may be able to build on other requirement specifications such as modecharts [9], which provide support for transitions among subgraphs in addition to transitions among states.

IV. Related Work

Human factors research has long recognized the common problem of mode errors, where a user, believing that a modal system component is in one mode when the component is actually in another mode, uses the component inappropriately. For instance, the pump example described in the introduction illustrates a mode error. Crow et al. [10] apply model

checking techniques to aeronautics system models to generate counterexamples that demonstrate mode errors. The counterexamples can then be used to modify the behavior of the system. In our work, if an overall process model may violate the requirements being considered, then a derived requirement is produced. The derived requirement can then be used to suggest modifications to the behavior of the healthcare process, the device, or both.

Some automated requirement derivation approaches (e.g., [11]) take modes into account. Combefis et al. [11] investigate a requirement derivation approach that can track the modes to help identify mode errors. They do not create views that take advantage of the modes as our modal abstraction view does.

Previous research has investigated different visualizations of the component models or requirements to improve their understandability. Jahanian and Mok [9] developed modecharts to specify modal component models at a high-level of abstraction. They also describe how to automatically translate the component models into low-level source code. In a dual way, we take the low-level derived FSA and, based on the modes, automatically create the modal abstraction view represented as a set of high-level abstracted FSAs. Some of the automated requirement derivation approaches (e.g., [12]) take into account high-level abstractions such as interaction overview diagrams to decompose the derived component requirements to improve their understandability. We similarly take advantage of the modes to create the modal abstraction view.

V. Conclusions and Future Work

Our initial investigation of this process-based requirement derivation approach produced useful derived requirements. The investigation, however, identified limitations, most notably the understandability of the derived requirements and the scalability of the derivation approach. To try to improve the understandability of the derived requirements, this paper explores a modal abstraction view, a decomposition of the derived requirements based on the modes. To evaluate this view, we implemented a modal abstraction view tool and are exploring its impact on the understandability of the requirements generated by our derivation approach. The modal abstraction view seems to improve understandability and should be further evaluated.

One possible direction for future research is to explore creating alternative views of the derived requirements that abstract away or highlight certain aspects of the derived requirements to improve their understandability. For instance, a process-based view of a derived requirement could highlight the portions that correspond to specific sub-processes or to the activities performed by specific medical clinicians. The different views of the derived requirements would ideally complement each other to further improve understandability of the derived requirements.

To try to improve scalability, we plan to explore various optimizations of our derivation approach. Additionally, we plan to investigate incremental refinement of the healthcare process models based on previous derivations.

To better understand this derivation approach, more extensive evaluation is needed. Such an evaluation should consider a larger range of overall process requirements and healthcare processes. The devices we considered in our case studies are relatively small units, but the approach could also be applied to more complex devices or to larger software systems such as computerized order entry systems.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Award(s) IIS-1239334 and CNS-1258588. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. Beyer, T. A. Henzinger, and V. Singh, "Algorithms for interface synthesis," in *CAV*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Springer, 2007, pp. 4–19.

[2] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich, "Flow analysis for verifying properties of concurrent software systems," *ACM Trans. on Softw. Eng. Method.*, vol. 13, no. 4, pp. 359–430, 2004.

[3] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[4] H. M. Conboy, G. S. Avrunin, and L. A. Clarke, "Process-based derivation of requirements for medical devices," in *IHI*, T. C. Veinot, Ü. V. Çatalyürek, G. Luo, H. Andrade, and N. R. Smalheiser, Eds. ACM, 2010, pp. 656–665.

[5] R. L. Rivest and R. E. Schapire, "Inference of finite automata using homing sequences (extended abstract)," in *STOC*, D. S. Johnson, Ed. ACM, 1989, pp. 411–420.

[6] M. Peleg, S. W. Tu, J. Bury, P. Ciccarese, J. Fox, R. A. Greenes, R. Hall, P. D. Johnson, N. Jones, A. Kumar, S. Miksch, S. Quaglini, A. Seyfang, E. H. Shortliffe, and M. Stefanelli, "Comparing computer-interpretable guideline models: A case-study approach," *JAMIA*, vol. 10, p. 2003, 2002.

[7] A. G. Cass, B. S. Lerner, S. M. S. Jr., E. K. McCall, A. E. Wise, and L. J. Osterweil, "Little-JIL/Juliette: a process definition language and interpreter," in *ICSE*, C. Ghezzi, M. Jazayeri, and A. L. Wolf, Eds. ACM, 2000, pp. 754–757.

[8] D. Giannakopoulou and C. S. Pasareanu, "Interface generation and compositional verification in JavaPathfinder," in *FASE*, ser. Lecture Notes in Computer Science, M. Chechik and M. Wirsing, Eds., vol. 5503. Springer, 2009, pp. 94–108.

[9] F. Jahanian and A. K. Mok, "Modechart: A specification language for real-time systems," *IEEE Trans. Softw. Eng.*, vol. 20, no. 12, pp. 933–947, Dec. 1994.

[10] J. Crow, D. Javaux (University of Liege), and J. Rushby, "Models and mechanized methods that integrate human factors into automation design," in *HCI-AERO '00: Int. Conf. on Human-Computer Interaction in Aeronaut.*, K. Abbott, J.-J. Speyer, and G. Boy, Eds., Toulouse, France, September 27 - 29 2000, pp. 163–168.

[11] S. Combéfis, D. Giannakopoulou, C. Pecheur, and M. Feary, "A formal framework for design and analysis of human-machine interaction," in *SMC*. IEEE, 2011, pp. 1801–1808.

[12] J. Whittle and P. K. Jayaraman, "Synthesizing hierarchical state machines from expressive scenario descriptions," *ACM Trans. Softw. Eng. Method.*, vol. 19, no. 3, pp. 8:1–8:45, Feb. 2010.