Design Considerations for Modeling Modes in Cyber–Physical Systems

Anitha Murugesan, Sanjai Rayadurgam, Michael W. Whalen, and Mats P.E. Heimdahl
University of Minnesota

Editor's notes:
Safety critical systems such as cruise control in automotive systems and variable rate bolus in medical device infusion pumps introduce complexity and reduce the flexibility of incremental code modifications. This paper proposes a generic pattern to structure the mode logic such that additions, modifications, and removal of behaviors could be done in a quick and localized fashion without losing model integrity. The authors illustrate the proposed pattern using the infusion pump as a case study and describe a design pattern for the mode logic of reactive systems that allows for flexible, understandable, and maintainable models.
—Rahul Mangharam, University of Pennsylvania

Systems where the physical world interacts with—often distributed and networked—software are called cyber–physical systems (CPSs). The behaviors of such systems are typically defined in terms of modes—mutually exclusive sets of system behaviors [1]. The modes together with the rules defining when and how the system transitions between modes are commonly referred to as the system's mode logic. The mode logic forms a useful structuring concept that facilitates the design of CPSs in various domains such as medical devices, avionics, and automotive controls [2], [3]. However, the derivation and verification of the mode logic is challenging due to the plurality of modes and the complexity of the rules that govern the transitions. Often the problem is further exacerbated due to multiple orthogonal dimensions for partitioning the system behavior into modes, all of which may be necessary to adequately and succinctly describe the mode logic.

Several well-known issues associated with the mode logic, such as, correct handling of complex mode transitions, ensuring consistency across orthogonal mode classifications, and eliminating mode confusion and automation surprises for system operators have been studied in depth [4], [1]. Though modeling the mode logic early in the development cycle is generally regarded as beneficial in addressing such issues, there is not enough guidance to address the design concerns associated with such modeling. Challenges arise when considering the essential bookkeeping activities needed to manage various state variables, such as timers, which are invariably intertwined with the mode logic. Engineering considerations such as adaptability to change and reuse across multiple products in a product family further amplify this challenge. The difficulties involved in modeling the modal behavior of systems have not, in our opinion, been adequately addressed in the literature. Even the definitions of frequently used terms—*modes, features*, and *states*, to name a few—vary, are subjective, and often overlap, leading to some understandable confusion [5].

In this paper, we revisit our work previously presented in a workshop [6] and discuss the challenges faced during early modeling of the mode logic in CPSs using a generic patient controlled analgesia

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/MDAT.2015.2462112
Date of publication: 29 July 2015; date of current version: 01 September 2015.

2168-2356/15 © 2015 IEEE
Copublished by the IEEE CEDA, IEEE CASS, IEEE SSCS, and TTTC

(GPCA) infusion pump—a medical device—as a case example. Our goal was to model the mode logic of the infusion pump in a manner that exhibits conceptually clarity, flexibility, and maintainability. We propose a generic pattern to structure the mode logic such that additions, modifications, and removal of behaviors could be done in a quick and localized fashion without losing model integrity. We illustrate the proposed pattern using the infusion pump example in MathWorks' Stateflow modeling notation; however, it is broadly applicable for modeling the mode logic of CPSs as state machines in notations similar to Statecharts [7].

We believe the experiences we gathered in the process are likely to benefit practitioners engaged in similar modeling efforts. More broadly, we hope our effort becomes a catalyst for the modeling community to catalog solutions for various modeling problems and build a repertoire of modeling patterns.

GPCA description

We now describe a generic infusion pump system, which we use in the rest of this paper to illustrate the modeling concerns and patterns. Infusion pumps are medical CPSs used for controlled delivery of liquid drugs into a patient's body according to a physician's prescription. Modern infusion pumps typically provide a variety of infusion options for drug delivery. In basal infusion, the drug is delivered for an extended period of time, usually at a low constant rate. In bolus infusion, the drug is delivered at a higher rate for a short duration of time to address some immediate need or to increase the drug delivery according to some therapy regimen. There may be multiple bolus types. An intermittent bolus infusion may be used to deliver additional drug at prescribed time intervals during the infusion therapy. Further, in a patient-controlled analgesia infusion system, a patient bolus infusion may be activated to deliver additional drug in response to a patient's request for more medication, typically to alleviate acute pain. In clinician bolus infusion, the drug is delivered at an elevated rate in response to a clinician's request, say to infuse an elevated rate of drug at the beginning of infusion therapy. More advanced pumps provide additional options such as variable program that allow the physicians to choose a combination of these infusion types to support varying drug flow over time and based on patient requests. Such advanced therapies involve switching and/or layering infusion types. These pumps also have safety features designed to ensure that the device does not pose any hazard to the patient, that selectively allow or disallow specific infusion options based on the hazards detected.

In this paper, we consider the software of a GPCA pump in which the programmable infusion types or infusion modes are basal, intermittent, and patient bolus. The following sections describe the challenges in modeling such a variety of infusion types with an eye toward flexible and straightforward addition, modification, and removal; it seems quite likely that clinical innovations will necessitate such changes to the mode-logic model of the GPCA software.

Modeling concerns

Models plays a central role throughout the lifecycle by serving as a concrete representation of the conceptual links between the various artifacts produced during development. Hence, it is essential that the models be conceptually clean, easy to understand, maintainable, and amenable to formal analysis [8]. Maintainability requires that the constructed model should be easy to modify for extensions and contractions [9]. This is particularly useful while modeling systems early in the development cycle, to identify interesting behavioral interactions that may have to be clarified with domain experts. Further, for modeling a family of closely related products—say, a product line of infusion pumps—it must be easy to adapt to the specifics of any given member of the product family.

When describing the GPCA software behaviors, we use the terms "modes" and "features." Features are visible aspects of the system from a user's perspective that can be individually enabled [10]. A mode is an exclusive collection of system behaviors from the system's perspective. While the definition of these terms and the distinction between them are of considerable interest and subject to debate, we avoid such discussions here. For the purposes of this paper, we refer to the available infusion types of the GPCA as features. For example, basal, patient bolus and intermittent infusions are features that are available in the device and the physician can choose and program a prescription that specifies the necessary parameters for these features. The resulting system behavior is modal—at any given instant there is a specific infusion type being delivered, such as basal or patient bolus. The mode logic defines how

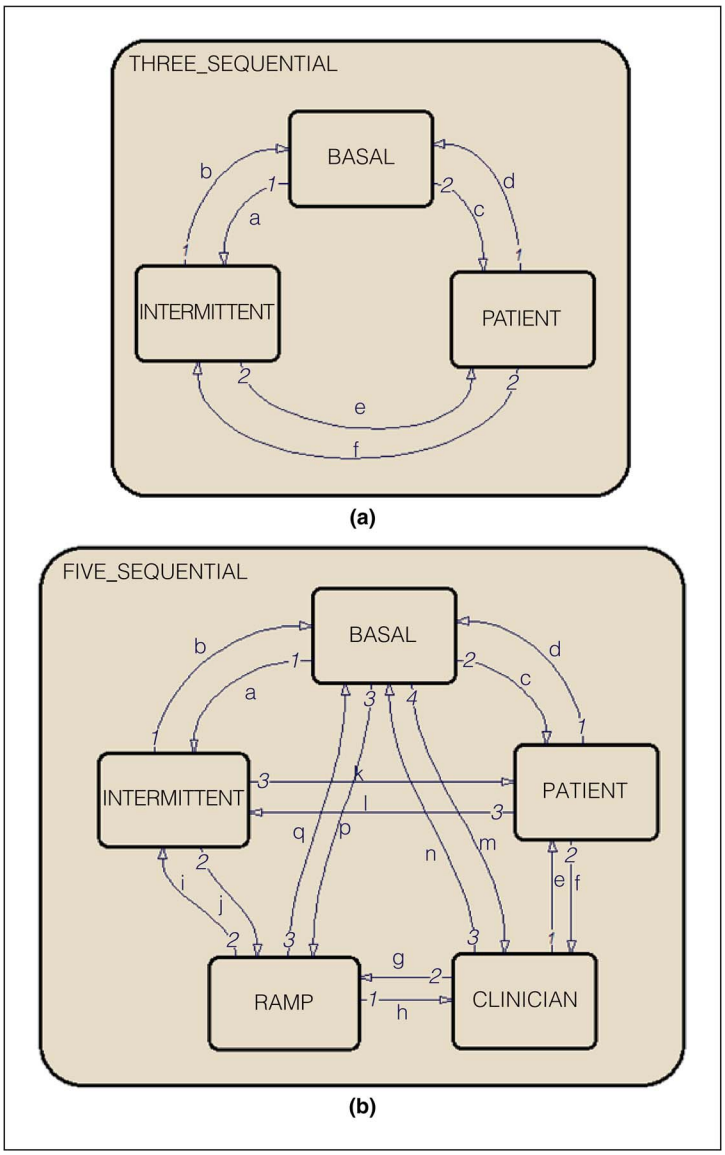


Figure 1. Sequential structure of the mode logic: (a) three features; and (b) five features.

the enabled features interact to produce the system behavior for each mode.

There are three main challenges that have to be effectively addressed when representing the mode logic of the GPCA such that it is understandable, flexible, and maintainable. A simple view of different system modes as states with transitions to represent mode changes aligns well with a user's view of the system, but leads to inflexible designs that hinder changes to the mode logic. The bookkeeping logic of internal state variables such as timers and flags are inextricably linked to the states; this complicates the task of making changes as our system understanding evolves. Also, the behaviors of the individual modes have both variety and similarity that may need to be captured and exploited to avoid redundancy within models. Hence, designing an understandable model that is flexible and maintainable is not a trivial task.

In the sequel, we elaborate on these concerns and possible solutions through our experience of modeling the GPCA using MathWorks' Stateflow which is widely used in the industry. It is a state-based notation that augments finite state automata with hierarchical and parallel states, and data variables, and is appropriate for modeling the mode logic of reactive systems such as the GPCA.

Modeling GPCA mode logic

During requirements analysis, the various modes of drug infusion of GPCA were visualized from the user's perspective, as shown in Figure 1a. In our initial attempt to model these modes, we arranged the modes as sequential states with their respective timers and flags contained within them. This sequential state design has been commonly used in modeling the mode logic of various CPS application such as aircraft collision avoidance system and pacemakers. Although it appeared superficially intuitive and corresponded well with our mental models, complications emerged as we attempted to make changes to the mode logic. For example, Figure 1b visually shows the complexity involved while adding new modes to the existing mode logic. While a highly competent engineer may meticulously handle changes while retaining model integrity, it became evident that the design is not easily adaptable for evolution and maintenance.

Another concern was flexibility in the design for accommodating changes in behavioral requirements. For example, a version of a GPCA software requirement reads:

> When an intermittent bolus is in progress and a patient bolus is requested, the intermittent bolus is suspended until the patient bolus is delivered and resumed after the patient bolus.

This may be interpreted as the current infusion being stopped, the new infusion taking place, then the previous infusion restarted from where it left off as illustrated in Figure 2b. Such a behavior may be in conflict with a different requirement that determines

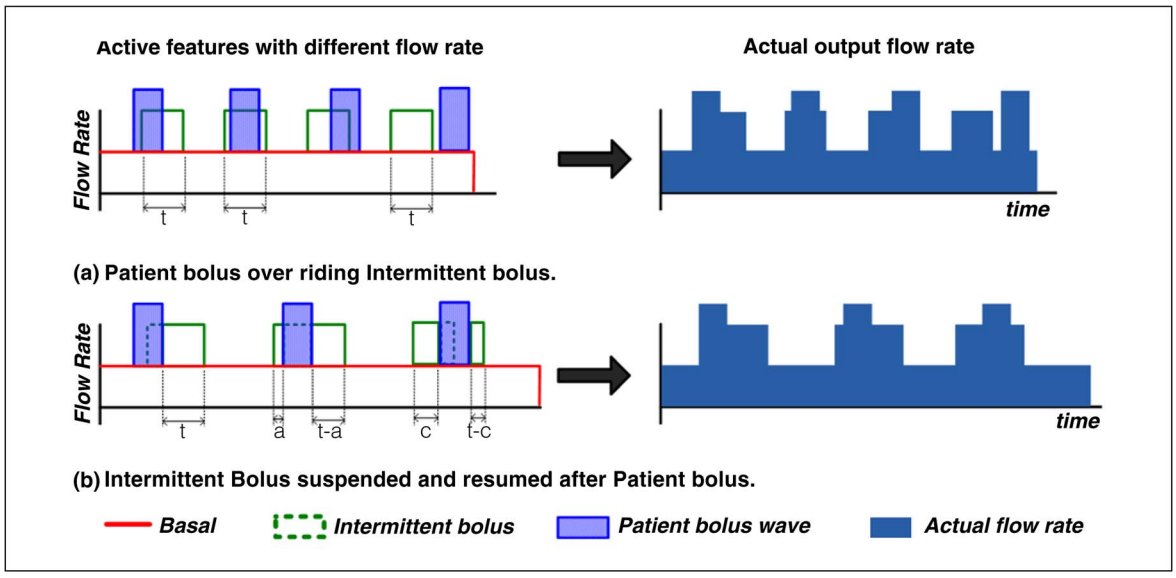


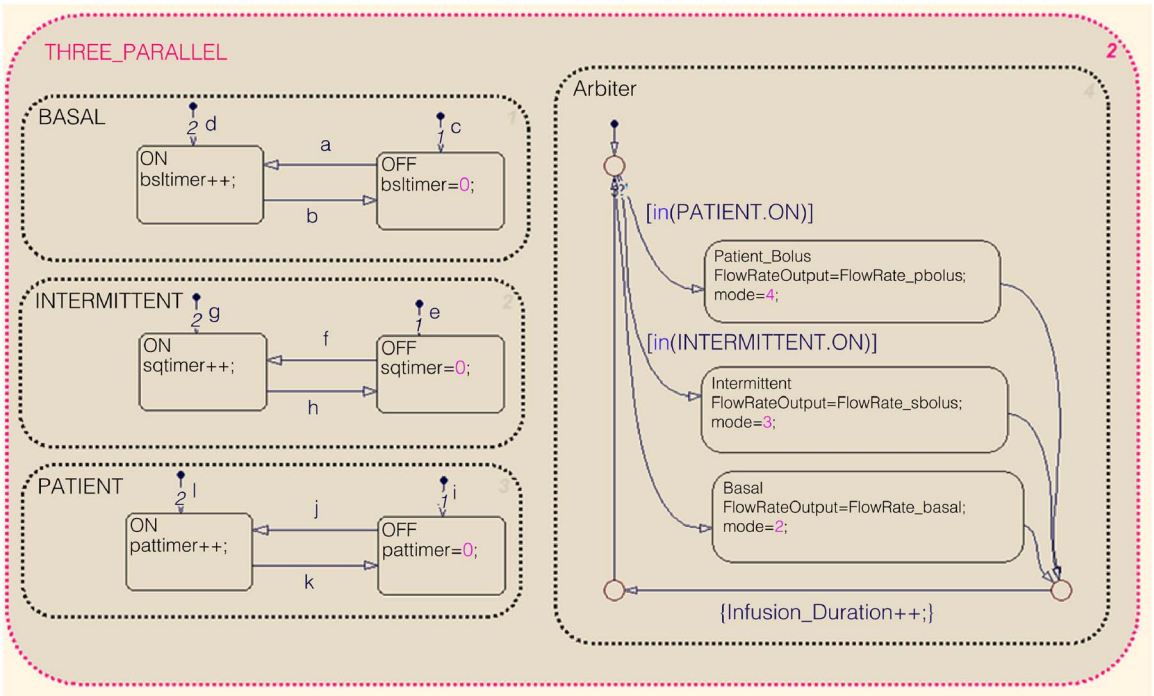
Figure 2. Variety of system behaviors.

the total duration of the infusion therapy. Alternatively, one can view the new infusion as simply overriding the current infusion rate (Figure 2a) and not increasing the total scheduled infusion time. Both solutions may be considered as viable (they may even be combined if therapies interact with each other in different ways).

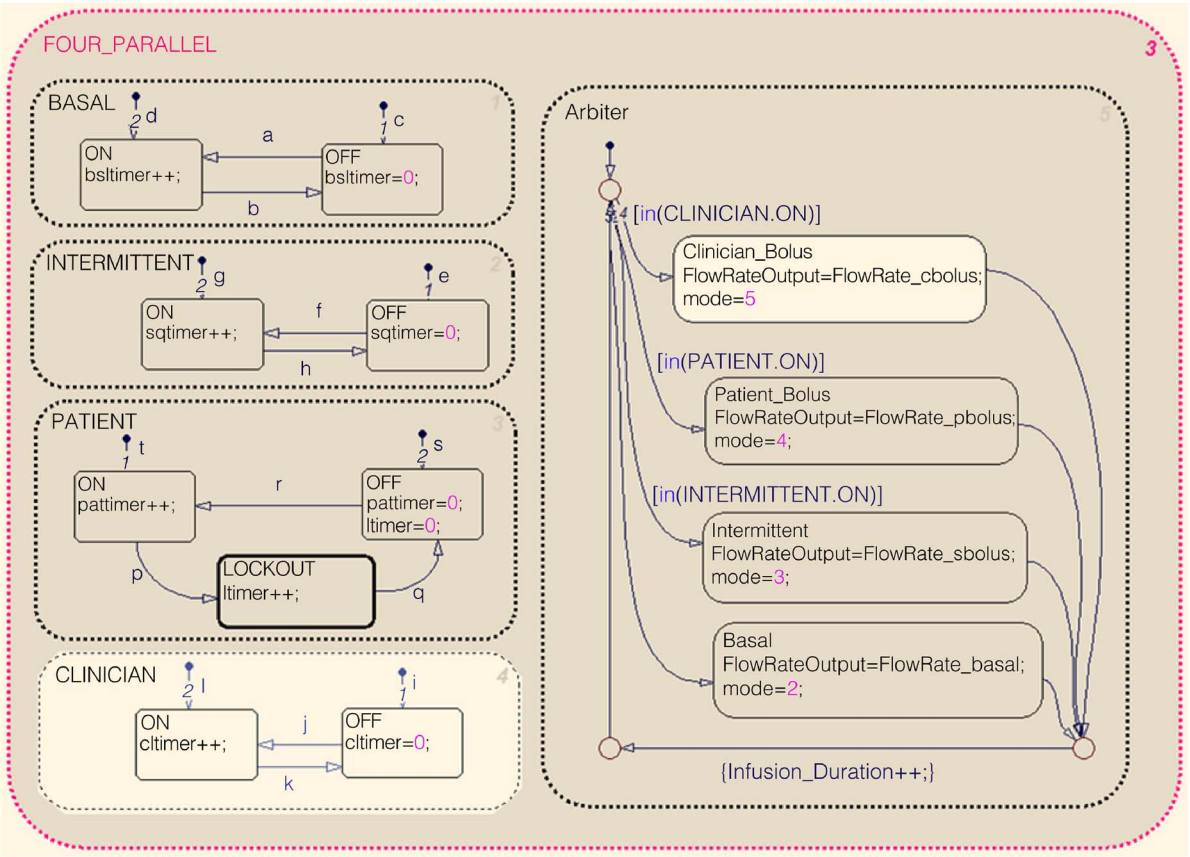
These alternatives have different consequences not only from the domain perspective but also from a modeling perspective. The mode logic for a suspend–resume-bolus behavior is different from the mode logic for a highest-rate-bolus behavior. During requirements analysis we want to be able to easily change the mode logic to simulate the alternatives and let the domain experts make the appropriate choice. As designers, we foresee that such behavioral requirements are likely to change and that there may be opportunities for reusing "generic" descriptions of modal behavior. This scenario is not limited to infusion pump system, but is likely to happen in other CPSs such as the cruise control system in automotive that has multiple cruising modes [3]. Decisions on whether the history of the previously active mode is necessary to resume to that mode is likely to change as the system understanding evolves. Accomplishing these changes with a sequential state machine for the mode logic is cumbersome because the associated bookkeeping and time management tends to be dispersed across multiple states and transitions.

An alternative approach is to design the system's modal behavior as a composition of multiple feature behaviors mediated and resolved by an arbiter, as shown in Figure 3a. In this structure, each feature is a self-contained state machine that has its own logic for turning on and off. The individual timers for each feature are managed within its state machine, and the overall system timer for therapy duration is commonly handled independently of the individual state machines. The transitions within each state machine are independent of the other parallel machines. The arbiter is a separate state machine that decides the modal behavior based on the individual feature behaviors; the arbiter is responsible for the prioritization and feature interactions to determine the overall system behavior manifested through the flow rate used for infusion and current infusion mode displayed to the clinician.

We believe that this pattern results in a design that is modular, scalable, and easier to understand. For example, in the GPCA, all the logic for prioritization was grouped in the arbiter and the individual timer and transition logic is within the respective feature state machine itself. This makes it straightforward to incorporate changes or extend the model to include new features. To illustrate the extensibility afforded by this structure, the clinician bolus feature, highlighted in Figure 3b, was included with minor changes to the existing features and arbiter. The observant reader would also have noted the difference in the way the patient bolus feature is modeled compared to other features: the addition to the lockout state that prevents patient boluses from being



(a)



(b)

Figure 3. Parallel structure of the mode logic: (a) three features; and (b) four features.

administered too closely in time illustrates the flexibility in the design to accommodate the changes in system behavioral requirements as our understanding of the system evolves. Similarly, one can easily modify the model to include a suspend–resume behavior for the intermittent feature by adding a new state along with transitions for suspend and resume within the intermittent state machine and corresponding logic in the arbiter for the prioritization. The parallel structuring allows the design to be modular by aggregating the prioritization logic in the arbiter and the feature-specific behaviors within the respective state machines.

Complex features

In the GPCA, some bolus modes had additional constraints placed on their behavior. For example, as mentioned above, once a patient bolus infusion is completed, the feature should be locked out from further activation (temporarily disabled) for a preset amount of time so that the system cannot continually provide high dosage infusion. This is easily accommodated by having an additional lockout state and appropriate transitions to and from that state, within the patient bolus state machine. This may also be viewed as a substate of the off-state. Similarly, a feature may have to be temporarily suspended and then resumed at a later point; this can be seen as a refinement of the on-state into substates. Instances of such variety in feature behaviors and comparable approaches for modeling such behaviors can be found in other safety-critical domains such as flight guidance systems [2].

Analyzing such common behavioral patterns of features led us to expand the simple on–off feature state machine to a more advanced hierarchical machine pattern, as shown in Figure 4. Not all states in this pattern may be required for a single feature and, indeed, each GPCA feature was modeled with only a few of these states. However, we believe that this general pattern captures the common structure of such feature state machines that typically arise when modeling the mode logic. In our experience with the GPCA system, we found that tailoring the common pattern to suit the needs for modeling a specific feature was rather straightforward. It helps analyzing the mode logic and making changes to it without losing model integrity. Additionally, this common pattern helps handling the bookkeeping of timers and other state variables in a consistent fashion across the modes.

The parallel structure of the mode logic allowed inclusion of this complex feature state machine

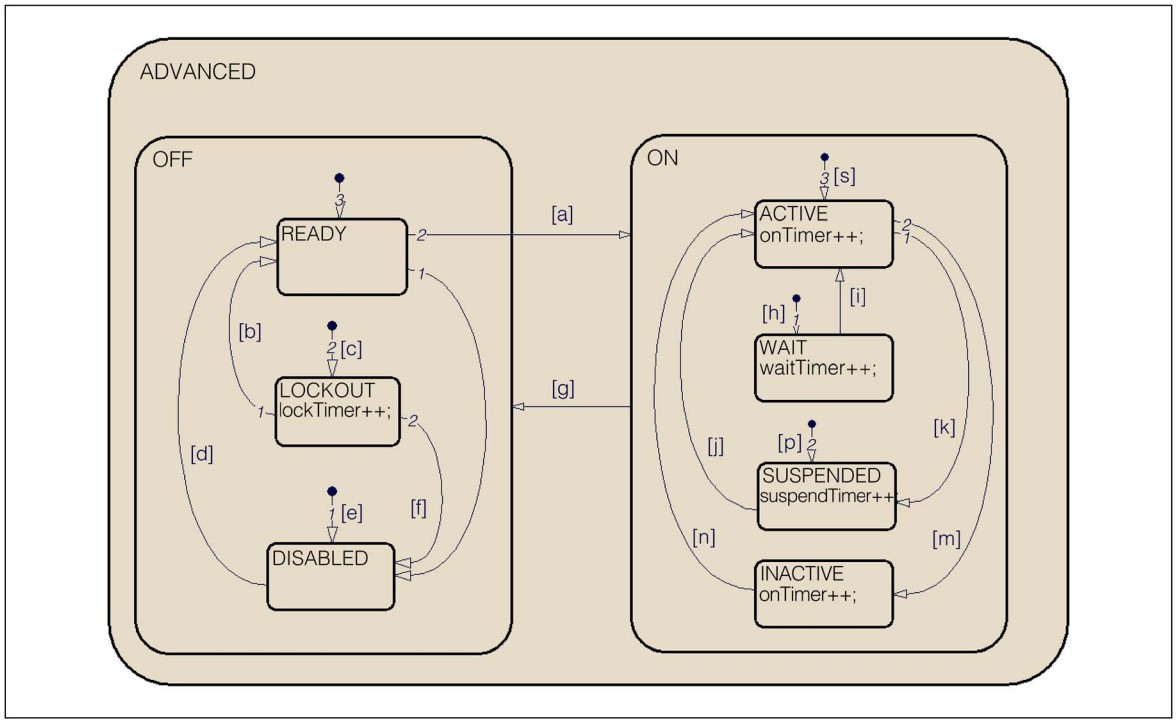


Figure 4. Complex features.

without affecting the way the other features are modeled. This easy inclusion/exclusion of features' states was one of the design goals of the model.

Related work and discussion

Among the alternatives, parallel structuring seems to be the most suitable pattern to model the mode logic for a wide range of CPSs. In our endeavor to model the mode logic for GPCA software, we identified challenging issues in terms of design understandability, flexibility, and manageability. While the sequential structure better matches the user's conceptual view of how operational modes work (mutually exclusive system states), it was clearly not suitable from the model's flexibility and manageability perspective. Other approaches to capture the mode logic include tabular formats such as Parnas tables [11] and SCR[12]. However, in general, we found that the graphical notations are more suitable for early modeling efforts to easily communicate to domain experts and to get clarity on requirements. Also, the tables for describing the mode logic become unmanageable with increase in the number of features and parameters that influence the mode logic. One could also capture the transition conditions in tables [2]. However, it could unnecessarily entangle the mode logic in multiple places which, in turn, reduces the understandability and flexibility of the model.

Alternative structuring patterns for the mode logic using state machines has been proposed by Miller et al. [2], where an event processing state machine resolves conflicting events which triggers feature transitions. Instead of allowing the features to transition into their individual states and an arbiter resolving their priorities (our approach), the event processing state machine restricts the possible transition of individual features by allowing/rejecting the events. While this is suitable to model systems where only one feature is active at a given time, our arbitration pattern helps decide the overall system behavior based on multiple, and more importantly orthogonal, enabled features. For example, in the infusion pump system, if there are alarming features that are orthogonal to the infusion features, the arbiter could be encoded to combine the behaviors of the system, in a conceptually simpler manner. On the other hand, the arbiter does not resolve conflicting simultaneous inputs. Thus, it is possible to combine the two, where event processing prioritizes the events before they are consumed by individual feature transitions while the arbiter resolves the conflicts after the individual feature transitions occurred.

While the proposed parallel structuring is flexible and maintainable, from a verification perspective, care has to be taken while specifying properties to verify the mode logic. When the modal behavior is expected to be mutually exclusive, a sequential pattern naturally captures it. But in the parallel pattern, the logic of deciding active features is handled by the arbiter. Verifying mutual exclusion is not necessarily straightforward; the behavior of the arbiter would have to be involved in specifying the property to be verified.

In this paper, we discussed our approach for modeling the mode logic of a medical device and the rationale for our model design choices. The challenges that had to be addressed and the modeling technique we employed seem to be common across several CPS domains. We believe that the modeling approach advocated in this paper could be applied as a pattern for representing the mode logic of reactive systems. While the focus of this work is on designing flexible, understandable, and maintainable models, in our future work, we also plan to assess the complexity of verification of these patterns. By identifying, documenting, and sharing such solutions, we hope this grows into a broader initiative toward a comprehensive catalog of CPS modeling patterns. ■

Acknowledgment

This work was supported in part by the National Science Foundation (NSF) under Grants CNS-0931931 and CNS-1035715. We would like to thank Dr. S. P. Miller of Rockwell Collins' Advanced Technology Center for starting us thinking about the challenges involved in modeling modes.

■ References

[1] A. Joshi, S. P. Miller, and M. P. Heimdahl, "Mode confusion analysis of a flight guidance system using formal methods," in *Proc. 22nd Digital Avionics Syst. Conf.*, 2003, doi:10.1109/DASC.2003.1245813.

[2] S. P. Miller, A. C. Tribble, M. W. Whalen, and M. P. E. Heimdahl, "Proving the shalls: Early validation of requirements through formal methods," *Int. J. Softw. Tools Technol. Transf.*, vol. 8, no. 4, pp. 303–319, 2006.

[3] C. Nowakowski et al., "Cooperative adaptive cruise control: Testing drivers' choices of following

distances,'' California PATH Program, Inst. Transportation Studies, Univ. California Berkeley, Berkeley, CA, USA, 2010.
[4] S. P. Miller and J. N. Potts, ''Detecting mode confusion through formal analysis and modeling,'' NASA Contractor, Rep. NASA/CR-1999-208971, 1999.
[5] J. Bredereke and A. Lankenau, ''A rigorous view of mode confusion,'' in *Computer Safety, Reliability and Security*, vol. 2434. Berlin, Germany: Springer-Verlag, 2002, pp. 19–31, ser. Lecture Notes in Computer Science.
[6] A. Murugesan, S. Rayadurgam, and M. Heimdahl, ''Modes, features, state-based modeling for clarity and flexibility,'' in *Proc. Workshop Model. Softw. Eng.*, 2013, pp. 13–17, doi:10.1109/MiSE.2013.6595290.
[7] D. Harel, ''Statecharts: A visual formalism for complex systems,'' *Sci. Comput. Programm.,* vol. 8, no. 3, pp. 231–274, Jun. 1987.
[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. New York, NY, USA: Pearson, 1994.
[9] D. Parnas, ''Designing software for ease of extension and contraction,'' *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 2, pp. 128–138, Mar. 1979.
[10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, ''Feature-oriented domain analysis (FODA) feasibility study,'' DTIC Document, Tech. Rep., 1990.
[11] D. L. Parnas, ''On the criteria to be used in decomposing systems into modules,'' *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.
[12] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw, ''SCR*: A toolset for specifying and analyzing requirements,'' in *Proc. 10th Annu. Conf. Comput. Assurance*, 1995, pp. 109–122.

Anitha Murugesan is currently working toward a PhD at the University of Minnesota, Minneapolis, MN, USA. She has worked as a Software Engineer with Honeywell Technology Solutions, India, for six years. Her research interests are requirements analysis and formal verification of cyber–physical systems software. Murugesan has a BE in electrical and electronics engineering from Madras University, Chennai, India, an MTech in computer science and engineering from Vellore Institute of Technology, Vellore, India, and an MS in computer science from the University of Minnesota. She is a member of the IEEE and the Association for Computing Machinery (ACM).

Sanjai Rayadurgam is a Program Director at the Software Engineering Center, University of Minnesota, Minneapolis, MN, USA. His research interests are in software testing, formal analysis and requirements modeling, with particular focus on safety-critical systems development, where he has significant industrial experience. Rayadurgam has a BSc in mathematics from the University of Madras, Chennai, India, an ME in computer science and engineering from the Indian Institute of Science, Bangalore, India, and a PhD from the University of Minnesota at Twin Cities, Twin Cities, MN, USA. He is a member of the IEEE and the Association for Computing Machinery (ACM).

Michael W. Whalen is the Director of the Software Engineering Center, University of Minnesota, Minneapolis, MN, USA. His interests are in formal analysis, language translation, testing, and requirements engineering. He has developed simulation, translation, testing, and formal analysis tools for model-based development languages including Simulink, Stateflow, SCADE, and RSML-e. Whalen has a BA in computer science and mathematics from Luther College, Decorah, IA, USA and a PhD in computer science from the University of Minnesota. He is a member of the IEEE and the IEEE and the Association for Computing Machinery (ACM).

Mats P.E. Heimdahl is the Department Head and Professor of Computer Science and Engineering at the University of Minnesota, Minneapolis, MN, USA. His research interests are in software engineering, safety-critical systems, testing, requirements engineering, and automated analysis of specifications. Heimdahl has an MS in computer science and engineering from the Royal Institute of Technology (KTH), Stockholm, Sweden and a PhD in information and computer science from the University of California at Irvine, Irvine, CA, USA. He is a member of the IEEE, the Association for Computing Machinery (ACM), and PSIA.

■ Direct questions and comments about this article to Anitha Murugesan, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, Minnesota 55455, USA; anitha@cs.umn.edu.