2017 24th Asia-Pacific Software Engineering Conference

Toward Detection of Abnormal Behaviors in Timing and Security Requirements

Danielle Gaither, Hyunsook Do, and Barrett R. Bryant
Department of Computer Science and Engineering
University of North Texas
Denton, Texas USA
{dcg0063, hyunsook.do, barrett.bryant}@unt.edu

***Abstract*—Finding software system defects during the requirements analysis phase can yield significant savings of time and effort when compared to finding the same defects during development or testing phases. The requirements engineering field has helped bring about significant advances in the early detection of system defects. However, a relatively small amount of research has been done regarding the detection of abnormal system behaviors. This is especially true for non-functional requirements (NFRs), which include areas such as timing and security requirements. Our work proposes the beginnings of a domain-specific modeling language for requirements analysis, with a particular emphasis on detecting abnormal system behaviors. We also demonstrate a preliminary version of our approach on a real-time embedded system.**

I. Introduction

Automation has become an increasingly important part of people's lives in recent years. While innovations from the Internet of Things (IoT) have the potential to improve the quality of life for many people, it is also important to have assurances that such systems are safe and secure.

Unfortunately, security around such devices is often lax. For example, in October of 2016, malware-infected IoT devices, including cameras and DVRs, were used maliciously to initiate distributed denial of service (DDoS) attacks [1] that interrupted service to many popular websites, including Github, Twitter, and PayPal [2]. An attack on one particular website generated 665 Gbps of traffic, which is one of the largest DDoS attacks ever recorded [1]. It was later discovered that the malware's mechanism for taking control of a device was simply attempting to login by going through a list of commonly-used default usernames and passwords, which was sufficient to gain control of approximately 100,000 devices [3].

Many software defects can be traced to incorrectly specified requirements and such defects can result in unreliable software systems [4]. While it is not possible to detect every possible situation a system might encounter, well-specified requirements can reduce the risk of abnormal behaviors from a system [5].

To date, some researchers have proposed various approaches to detecting abnormal behaviors during the requirements analysis phase, but most attempts have focused on purely functional requirements [6], [7], and few researchers have considered non-functional requirements analysis, especially as a primary topic [8], [9]. However, non-functional requirements (NFRs) such as timing and security are important in part because they are the source of many quality-of-service (QoS) concerns, which in turn, are key to many service level agreements (SLAs). If such requirements are not met, a service provider could be put at significant financial or even legal risk for noncompliance with a contract. Further, for mission-critical systems, such as cars and aircrafts, NFRs are the most important and major concern, because if they do not function correctly as expected, the consequences could result in a safety or security hazard.

The primary goal of our work is to improve the capabilities of requirements analysis methods for detecting abnormal behaviors during the requirements analysis process, particularly with regard to NFRs such as timing and security requirements. To achieve this, we have devised an approach to modeling and analyzing timing and security requirements. Our approach creates a statechart model of the system from the requirements, which is grounded in an established formal semantics. The model is then simulated to detect any potential abnormal behaviors. If desired, the model can then be used to generate code.

To investigate the feasibility of our proposed approach, we have performed a case study on a real-world requirements document. The results indicate that our approach is capable of finding errors related to timing and security in a set of requirements. The approach can also help us recognize instances where requirements are not sufficiently well-defined. Because our approach provided promising results, we plan to improve and expand our approach to support for requirements modeling and abnormal behavior detection in existing requirements analysis tools as well as performing additional case studies and controlled experiments considering various types of embedded systems.

The rest of this paper is organized as follows. We provide relevant background information in Section II and outline our proposed approach in Section III. We then discuss completed and future work in Section IV through a feasibility study that uses a requirements document for an infusion pump. We offer conclusions and suggestions for future work in Section VI.

978-1-5386-3681-7/17 $31.00 © 2017 IEEE
DOI 10.1109/APSEC.2017.90

II. Background

In this section we provide background on two core elements of our approach: the causal component model [6] and statecharts [10].

A. The Causal Component Model

The causal component model (CCM) was first proposed by Aceituna and Do [6]. The goal of CCM is to analyze a given set of requirements and detect potential abnormal system behaviors. A major contribution of this approach is that rather than considering a system as a flat interaction of system states, CCM decomposes a system into its components and considers the states of each individual component and how those states might interact with each other. This information, along with a list of undesired state combinations, is used to extract a set of transition rules, which are then analyzed to see if undesired states are reachable, and if so, whether it is possible for the system to recover from such states.

The original CCM tool was only concerned with functional requirements, but an enhanced version of the tool that is currently under development will also support modeling of non-functional requirements, since NFRs are of increasing importance among creators of software systems. Some of the reasons for this increased importance include performance concerns [11], timing concerns [12], and compliance concerns [13]. A key component of this work is to apply the CCM methodology to constructing statechart models.

B. The Use of Statecharts

Because our work is concerned with real-time embedded systems, it makes sense to use a modeling framework suited for that purpose. Statecharts were first proposed by Harel [10] as a way to overcome the limitations of the existing modeling tools of the time, such as the lack of precise timing semantics. This also coincided with a shift away from procedural programming and toward a more event-driven approach in the programming world at large. Some of the benefits of statecharts over simple finite state machines include a precise formal semantics, a mechanism for modeling hierarchy among states, support for concurrency, and the ability for different portions of the model to communicate with each other.

While more simple traditional graph-based tools are often used for requirements analysis, such tools typically do not support concurrency among states or allow for the connection of multiple levels of abstraction in the modeling process. The latter issue can be particularly helpful, as it can help mitigate state explosion, which is one of the most common problems facing any sort of state-based analysis method. Allowing for different levels of abstraction can immediately narrow the state space of a model at any given moment.

Further information about using a CCM-based approach with statecharts will be provided in Section III.

III. Proposed Approach

In this section, we describe the details of our approach, including a discussion of: (1) the system model and how it is constructed, (2) our proposed methodology for analyzing requirements.

Our model is focused on specific NFRs, particularly timing and security information. Statecharts have been widely used in verification of real-time embedded systems [14] due to the ease of adding timing information, the ability to execute states in parallel, and a hierarchical structure that often mimics the architecture of real-time embedded systems [15]. Figure 1 provides an overview of our approach.

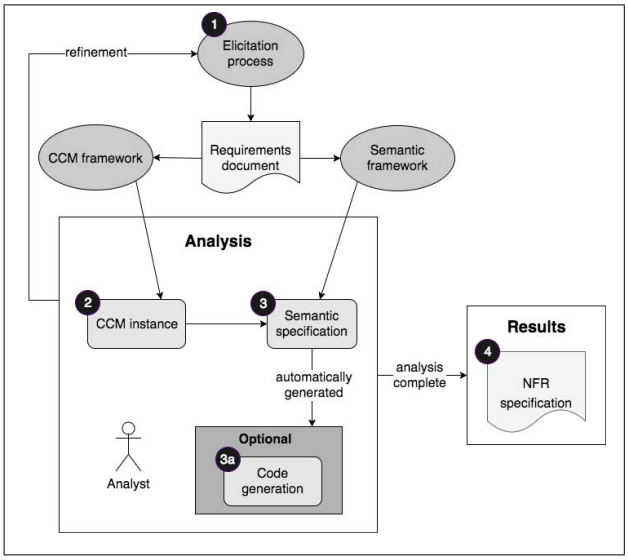

Fig. 1: An overview of the proposed approach [16].

A. The System Model

As Figure 1 shows, step 1 in our approach is eliciting system requirements from stakeholders. That information helps create the initial requirements document. We apply the CCM framework in step 2 to create a CCM instance, as described in Subsection II-A. Since our work is concerned with timing and security requirements, we also look at possible constraints on transitions, especially with regard to timing and security concerns.

Once the CCM instance is created, we then use that information in step 3 to create a statechart model of the system requirements using Yakindu Statechart Tools [17]. Yakindu is a statechart analysis platform that may be run either as a standalone application or as an Eclipse plugin [17]. The work described in this paper was performed using the standalone tool. The tool includes syntax checking and a built-in simulator for the statecharts, as well as the capability to generate code in C, C++, or Java. The code generator can also be customized to work with other languages, which is listed as step 3A in Figure 1.

The statechart model is then simulated to detect possible abnormal behaviors. In the event abnormal behaviors are found, the process is refined through further discussion with system stakeholders. The refinement process can be repeated

as needed until stakeholders are satisfied, yielding a complete NFR specification in step 4.

Currently, our model is built manually from a CCM instance. Our goal is to integrate our approach with the enhanced CCM analysis tool that is in development. Since the models themselves are expressed as XML files, we are exploring automated analysis methods using XML parsers. A sample system model using our current approach is shown in Figure 2.

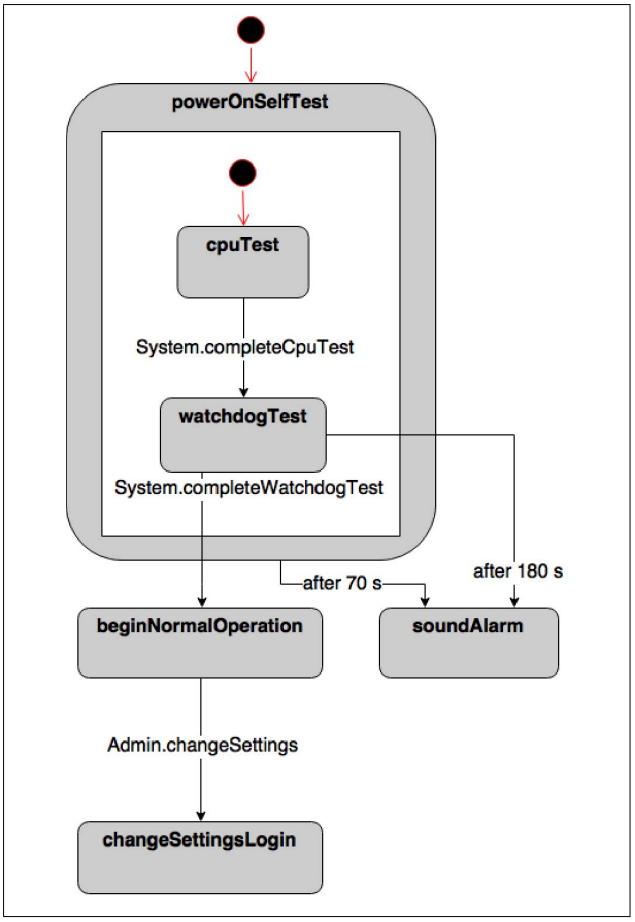

Fig. 2: A statechart representation of the system's key operations.

Figure 2 is a statechart representation of the requirements listed in Figure 3 and is described in detail in Section IV. In short, when the infusion pump is powered on, a Power On Self Test (POST) that consists of a series of smaller tests is initiated. These smaller tests are not necessarily completed in any particular order. If any of the tests fails, an alarm is sounded. Otherwise, the pump may begin normal operations.

Once the pump begins normal operations, it is possible to change certain settings on the pump. However, only a pump administrator, and not the user, should be able to change the settings. The requirements document does not provide further detail about what might be involved in the process of changing the pump settings.

B. Requirements Refinement

It is rare that any given set of requirements is completely correct on the first attempt. Therefore it makes sense for any process involving requirements to support refinement of the requirements as changes are made. As Figure 1 demonstrates, as each round of analysis is completed, the analysts return to the stakeholders for clarification and update the requirements accordingly. This process is demonstrated in our case study in Section IV.

IV. Feasibility Study

To evaluate our proposed approach, we performed a feasibility study using a requirements document for a generic infusion pump [18]. The pump and its requirements document are part of a larger project attempting to improve the design and requirements specifications of embedded medical devices. NFRs can specify timing and dosage information, as well as various security properties. Patients rely on pumps like this to administer important, sometimes even life-saving, medication, so the consequences of error can be dire.

The Generic Infusion Pump is part of a larger research-oriented project [19]. However, relatively little of the work is concerned with requirements *per se*. Much of the research is concerned with aspects of user interaction [20] or user interface [21].

Although other requirements documents are available for this system, our focus is on analysis of NFRs, so we chose a safety requirements document as our starting point. The focus of our analysis will be on a simplified subset of the requirements described in Figure 3.

A. Timing Requirements

A benefit of statecharts is that timing information is easy to implement, and we have done so in our statechart model in Figure 2. This makes timing requirements easier to analyze, and they are the first type of requirements we wish to discuss. The system begins with a power on self-test (POST) that must be completed within a certain time limit. The test includes a CPU test and a watchdog test. Although the document refers to further details regarding the watchdog test, such details are never specified. Therefore, they are ignored at this stage of analysis. Naturally, the maximum running time of a particular task is the sum of the maximum running time of each of that task's components. Note that the requirements specify that POST should take no more than 1 minute 10 seconds. However, note that the watchdog test can take up to 3 minutes to raise a failure alarm. Because the Watchdog test is one of the components of the POST operation, this yields a conflicting set of requirements.

One of the benefits of statecharts is that they allow for composite states, or to put it another way, states within states. Each state must be reachable, either by some initial state, represented by the black circles, or by a transition from another state. It is possible to trigger a transition after a certain amount of time has elapsed, after a boolean condition has been met, or upon the occurrence of a particular event.

1) Power On Self Test (POST)
   a) On being powered on, the pump shall undergo a power on self-test (POST).
   b) The system shall perform POST for all devices possible without degrading normal operation.
   c) The POST shall take no longer than 1 minute 10 seconds.
   d) The POST shall consist of the following tests:
      i) CPU test
      ii) Watchdog test
   e) Any failure of a test step during POST shall abort the remaining test steps and generate the appropriate alarm for the failure.
2) Watchdog test
   a) Each task involved in the pump delivery shall have a watchdog timer or counter associated with to catch and stop run-away, or stalled processes.
   b) The watchdog timer shall interrupt the pump if it ceases/suspends normal operation, or does not respond to user input for 90 seconds.
   c) The watchdog timer shall check that each of the other tasks has responded within the last 90 seconds.
   d) If any task does not respond to a watchdog test for more than 3 minutes, a watchdog alarm shall be raised.
   e) A watchdog test shall be performed by calling a low-level driver and shall generate a watchdog test failure alarm upon failure.
3) Pump settings
   a) The user/patient shall not be able to change the default settings.
   b) The defaults shall only be modified or configured by a pump administrator.
   c) The administrator screens shall be protected by a secure login/password.
4) Infusion requirements
   a) If the current volume of the drug reservoir is 0 ml, and an infusion is in progress, an alarm shall be issued.
   b) When an occlusion occurs, the pump shall stop flow and alarm as quickly as possible.

Fig. 3: Simplified requirements of the Generic Infusion Pump (extracted from [18]).

The clock starts as soon as the **power on self test** state is entered and updates every second. Per the requirements, if 70 seconds elapse without the tests successfully completing, an alarm is sounded. The same is true for the watchdog test, only for 180 seconds instead of 70. If both subtests succeed, i.e., it is possible to exit those substates without triggering the alarm, then the pump can begin normal operation. Otherwise, something has gone wrong during the testing process and an alarm is sounded.

As mentioned previously, our analysis method reveals a conflicting set of requirements. The overall self-test procedure sounds an alarm if it is not successful within 70 seconds, but the watchdog test can take up to 180 seconds to complete. This means that the watchdog test is never allowed to time out.

As shown in Figure 1, this error provides an opportunity for refinement. The analyst would return to the system stakeholders and ask for clarification. It might be possible to complete the watchdog test in less than 70 seconds, or perhaps the overall time limit needs to be raised to a limit greater than 180 seconds. In any case, a potential system bug has been averted in the requirements analysis phase, which costs significantly less in time and resources than catching the bug later in the process [22]. To resolve the conflict, we will assume that the stakeholders wish to raise the overall time limit for the self-test to 3 minutes 10 seconds to allow for execution time of the other tests. A listing of changes in requirements can be found in Table I.

B. Security Requirement

One benefit of requirements analysis is the ability to discover requirements that are ill-defined, as is the case with this set of requirements. It is worth noting that details of security implementations have not generally been considered during the requirements gathering phase until recently, but with lack of specification often leading to lax security implementations, we propose that creators of systems can ill afford to ignore the details of security implementations in the current environment. Questions to consider for this particular system include:

- What might the authentication procedure look like?
- What, if anything, might happen after a specified number of unsuccessful login attempts?

Both issues address or have the potential to address some of security's core principles, namely the confidentiality and integrity of the system. However, the security requirement as specified in the document addresses neither issue. Indeed, the **changeLoginSettings** node is revealed as a potential dead end, as there are no transitions out of it.

Dead end nodes are not inherently undesirable. It is possible that developers may wish to use such nodes to portray error states, for example. However, let us assume in this case that a login screen to change the settings for a device is not one of those situations.

Once an undesired dead end node is detected, the opportunity for requirement refinement presents itself. Through our analysis, we can update the requirements to reflect more detailed information. Requirement 3 from Figure 3 can be updated to reflect the information in Table I. The specific changes to the requirements are indicated in boldface. A graphical representation of the updated statechart can be seen in Figure 4.

TABLE I: Original and updated requirements.

Previous Requirement	Updated Requirement
The POST shall take no longer than 1 minute 10 seconds.	The POST shall take no longer than **3 minutes** 10 seconds.
The administrator screens shall be protected by a secure login/-password.	The administrator screens shall be protected by a secure login/-password. 1) **If a login attempt is unsuccessful, the device will return to the login screen.** 2) **If there are three consecutive failed login attempts, the device alarm will sound.**

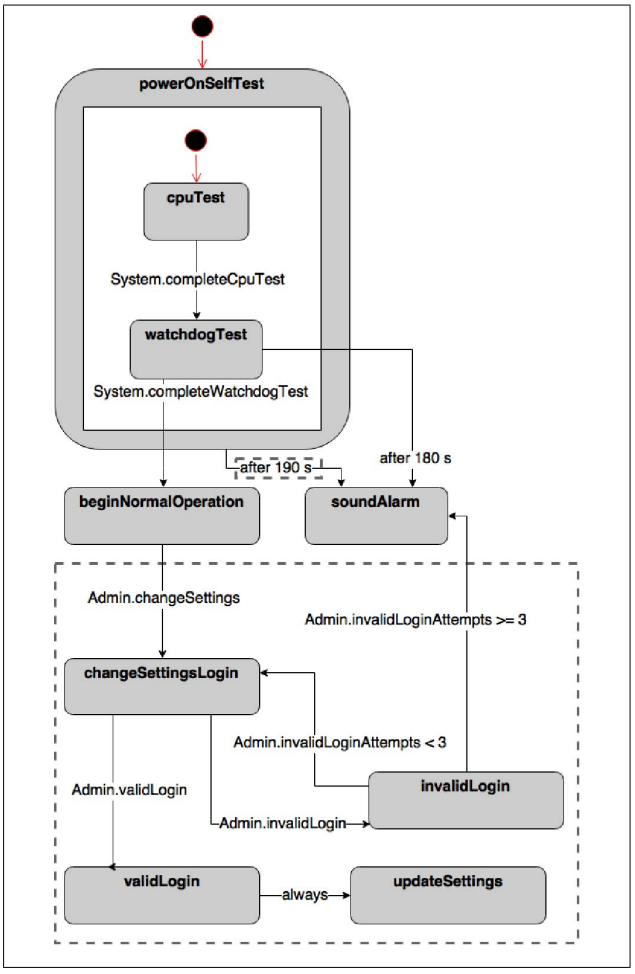

Fig. 4: A statechart representation of the system's key operations.

C. Limitations of Study

The work is in its early stages, so the findings presented here are preliminary. A more thorough study with an empirical comparison to existing analysis methods should yield more conclusive results.

Another limitation is the small size of the sample document. While there is work in progress to mitigate scaling problems such as state explosion, it is unknown if such mitigation strategies will be effective until they are attempted on larger requirements documents.

D. Summary of Study Findings

Through applying the CCM approach to building statecharts, we have created an approach to modeling both functional and non-functional requirements that can be analyzed in a systematic fashion. Our analysis method is capable of finding inconsistencies and incompleteness in a given set of requirements, as shown in our example. The work in this paper lays the groundwork for a formal semantics of a domain-specific modeling language created for this type of analysis. We believe our method is particularly well-suited to analysis of NFRs such as timing and security requirements, which are increasing in importance to builders of software systems.

V. Related Work

While some researchers have worked in the area of NFR analysis, most have focused on aspects other than requirements analysis and abnormal behavior detection. Filieri et al. used probabilistic formal models to reason about the potential availability of a system based on NFR information and possible environmental changes [8]. Mairiza et al. created a framework for analyzing NFRs, but it was only concerned with detecting and resolving conflicts among the requirements themselves. Although the authors argue that some conflict between NFRs is inevitable, a decision framework can be established not only to minimize conflicts between requirements, but also to determine how to prioritize different types of NFRs [9].

It is also important to examine the role that statecharts have played in similar work. Statecharts have been used by others with regard to real-time embedded systems. Ghezzi et al. [23] created a verification environment that assumes statecharts will evolve over time, as is often the case. Guo et al. [24] used statecharts to simulate not only medical devices, but also situations that healthcare professionals may encounter in their work. Their approach involved transforming statecharts into UPPAAL models for model checking. However, a limitation of UPPAAL is that it does not support substates in its automata [24]. Kleinmann and Wool [25] studied the use of statecharts in anomaly detection for control systems. They were specifically concerned with state explosion and the high rate of false positives from existing methods. Kleinmann and Wool used statecharts as a way of narrowing the state space, as well as devising a method for selecting the appropriate state space at a given moment.

There has also been relevant work in the area of model-driven requirements analysis. Kim et al. [26] used a model-based framework to deal with timing properties, but their work is not concerned with off-nominal behaviors. Aceituna and Do [6] performed work in off-nominal behavior detection, but

their approach was limited to functional requirements. Others have also done interesting work using a UML profile called MARTE, which is designed for modeling real-time embedded systems [27]. Saadatmand, Cicchetti, and Sjödin [12] combined MARTE with SysML to analyze security requirements, but their approach was limited to the verification of said properties. Suryadevara et al. [28] use MARTE for modeling real-time embedded systems, but their work is limited to model checking of timed properties.

VI. Conclusions and Future Work

In this paper, we proposed a model-based approach for detecting abnormal behaviors of requirements, particularly focusing on timing and security requirements, which have not to date received the attention by the researchers that functional requirements have. We believe that statecharts are a worthwhile choice as a basis for our approach. At the same time, we recognize the limitations of the current tools, so we have combined statecharts with a systematic approach to building our statecharts and have demonstrated the validity of our approach with a feasibility study.

Although we are in the early stages of this work, we already see significant potential for building on it. For example, we would like to be able to model safety requirements along with timing and security requirements. We are also working to evaluate our approach on more complex sets of requirements. Another avenue for exploration is integration with existing requirements patterns and tools.

Acknowledgment

This work was supported, in part, by NSF CAREER Award CCF-1564238 to the University of North Texas.

References

[1] M. Kan, "An IoT botnet is partly behind Friday's massive DDOS attack — PCWorld," 2016. [Online]. Available: http://www.pcworld.com/article/3134056/hacking/an-iot-botnet-is-partly-behind-fridays-massive-ddos-attack.html

[2] B. Chacos, "Major DDoS attack on Dyn DNS knocks Spotify, Twitter, Github, PayPal, and more offline — PCWorld," 2016. [Online]. Available: http://www.pcworld.com/article/3133847/internet/ddos-attack-on-dyn-knocks-spotify-twitter-github-etsy-and-more-offline.html

[3] Symantec Security Response, "Mirai: what you need to know about the botnet behind recent major DDoS attacks — Symantec Connect Community," 2016. [Online]. Available: https://www.symantec.com/connect/blogs/mirai-what-you-need-know-about-botnet-behind-recent-major-ddos-attacks

[4] J. C. Knight, "Software challenges in aviation systems," in *21st International Conference, SAFECOMP*. Springer Berlin Heidelberg, 2002, pp. 106–112.

[5] N. G. Leveson, "The role of software in spacecraft accidents," *Journal of Spacecraft and Rockets*, vol. 41, no. 4, pp. 564–575, 2004.

[6] D. Aceituna and H. Do, "Exposing the Susceptibility of Off-Nominal Behaviors in Reactive System Requirements," in *Proceedings of the IEEE International Requirements Engineering Conference*, vol. 23. IEEE, 2015, pp. 136–145.

[7] P.-H. Cheng, J.-M. Fu, and L.-W. Chen, "Knowledge transfer of software tool development for functional requirements analysis," *Computer Applications in Engineering Education*, vol. 24, no. 1, pp. 131–143, jan 2016.

[8] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, vol. 24, no. 2, pp. 163–186, mar 2011.

[9] D. Mairiza, D. Zowghi, and V. Gervasi, "Utilizing TOPSIS: a multi criteria decision analysis technique for non-functional requirements conflicts," in *First Asia Pacific Requirements Engineering Symposium, APRES 2014*, Didar Zowghi and Zhi Jin, Eds. Auckland: Springer Berlin Heidelberg, 2014, pp. 31–44.

[10] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[11] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "Non-functional requirements in architectural decision making," *IEEE Software*, vol. 30, no. 2, pp. 61–67, mar 2013.

[12] M. Saadatmand, A. Cicchetti, and M. Sjödin, "UML-based modeling of non-functional requirements in telecommunication systems," in *The Sixth International Conference on Software Engineering Advances (ICSEA 2011)*, Barcelona, 2011, pp. 213–220.

[13] J. Bhatia, T. D. Breaux, and F. Schaub, "Mining Privacy Goals from Privacy Policies Using Hybridized Task Recomposition," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 3, pp. 22:1–22:24, 2016.

[14] A. David, M. Oliver Möller, and W. Yi, "Formal verification of UML statecharts with real-time extensions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2306, 2002, pp. 218–232.

[15] H. Giese and S. Burmester, "Real-time statechart semantics," Software Engineering Group, University of Paderborn, Tech. Rep., 2003.

[16] D. Gaither, "Verification of Non-Functional Requirements Using Formal Semantics," To appear in proceedings of MODELS 2017 Doctoral Symposium, Tech. Rep., 2017.

[17] itemis AG, "Yakindu Statechart Tools," 2016.

[18] D. E. Arney, R. Jetley, I. Lee, A. Ray, O. Sokolsky, and Y. Zhang, "Generic Infusion Pump hazard analysis and safety requirements Version 1.0," University of Pennsylvania Department of Computer and Information Science, Tech. Rep., 2009.

[19] Generic Infusion Pump Research Project, "The Generic Infusion Pump (GIP)," 2015. [Online]. Available: https://rtg.cis.upenn.edu/gip/

[20] R. Ruksenas, P. Masci, M. D. Harrison, and P. Curzon, "Developing and verifying user interface requirements for infusion pumps: a refinement approach," in *Electronic Communications of the EASST Proceedings of the 5th International Workshop on Formal Methods for Interactive Systems (FMIS 2013) systems*, 2013, pp. 1–15.

[21] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon, "Demonstrating that medical devices satisfy user related safety requirements," in *4th International Symposium on Foundations of Healthcare Information Engineering and Systems (FHIES2014)*, Washington, DC, 2014, pp. 1–16.

[22] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[23] C. Ghezzi, C. Menghi, A. Molzam Sharifloo, and P. Spoletini, "On requirement verification for evolving statecharts specifications," *Requirements Engineering*, vol. 19, no. 3, pp. 231–255, sep 2014.

[24] C. Guo, S. Ren, Y. Jiang, P.-L. Wu, L. Sha, and R. B. Berlin, "Transforming medical best practice guidelines to executable and verifiable statechart models," in *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, apr 2016, pp. 1–10.

[25] A. Kleinmann and A. Wool, "A Statechart-Based Anomaly Detection Model for Multi-Threaded SCADA Systems," in *International Conference on Critical Information Infrastructures Security*. Springer, Cham, 2016, pp. 132–144.

[26] B. Kim, L. Feng, L. T. Phan, O. Sokolsky, and I. Lee, "Platform-specific timing verification framework in model-based implementation," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDAA, 2015, pp. 235–240.

[27] Object Management Group, "The Official OMG MARTE Web Site," 2012. [Online]. Available: http://www.omg.org/omgmarte/

[28] J. Suryadevara, C. Seceleanu, F. Mallet, and P. Pettersson, "Verifying MARTE/CCSL Mode Behaviors Using UPPAAL," in *11th International Conference, Software Engineering and Formal Methods (SEFM 2013)*. Madrid: Springer Berlin Heidelberg, 2013, pp. 1–15.