Refinement Maps for Insulin Pump Control Software Safety Verification

Eman M. Al-qtiemat*, Sudarshan K. Srinivasan*, Zeyad A. Al-Odat*, Sana Shuja†
*Electrical and Computer Engineering, North Dakota State University,
Fargo, ND, USA
†Department of Electrical Engineering, COMSATS University,
Islamabad, Pakistan
Emails: *eman.alqtiemat@ndsu.edu, *sudarshan.srinivasan@ndsu.edu, *zeyad.alodat@ndsu.edu,
†SanaShuja@comsats.edu.pk

***Abstract*—Refinement-based verification is a formal verification technique that has shown promise to be applicable for verification of low-level real-time embedded object code. In refinement-based verification, both the implementation (the artifact to be verified) and the specification are modeled as transition systems, which essentially capture the states of the system and transitions between the states. A key step in the verification process is the construction of a refinement map, which is a function that maps implementation states onto specification states. Construction of refinement maps is most often done manually and requires key insights about how the implementation and specification behave. In this paper, we develop refinement maps for various safety properties concerning the software control operation of insulin pumps. We then identify possible generic templates for construction of refinement maps as a first step towards developing a process to construct refinement maps in an automated fashion.**

Keywords–Formal verification; safety-critical devices; Refinement maps; Refinement-based verification.

I. INTRODUCTION

One of the key issues in designing safety-critical embedded systems such as medical devices is software safety [1]. For example, infusion pumps (a medical device that delivers medication such as pain medication, insulin, cancer drugs etc., in controlled doses to patients intravenously) has 54 class 1 recalls related to software issued by the US Food and Drug Administration (FDA) [2]. Class 1 means that the use of the medical device can cause serious adverse health consequences or death.

Despite the fact that testing is the dominant verification technique currently used in commercial design cycles [3], testing can only show the presence of faults, but it never proves their absence [4]. Alternate verification processes should be applied to the software design in conjunction with testing to assure system correctness and reliability. Formal verification can address testing limitations by providing proofs of correctness for software safety. Intel [5], Microsoft [6] and [7], and Airbus [8] have successfully applied formal verification processes.

Refinement-based verification [9] is a formal verification technique that has been demonstrated to be effective for verification of software correctness at the object code level [10]. To apply refinement-based verification, software requirements should be expressed as a formal model. Previously, we have proposed a novel approach to synthesize formal specifications from natural language requirements [11], and in a later work, we have also addressed timing requirements and specifications [12].

Our verification approach is based on the theory of Well-Founded Equivalence Bisimulation (WEB) refinement [9]. In the context of WEB refinement, both the implementation and specification are treated as Transition Systems (TSs). If every behavior of the implementation is matched by a behavior of the specification and vice versa, then the implementation behaves correctly as prescribed by the specification. However, this is not easy to check in practice as the implementation TS and specification TS can look very different. The specification states obtained from the software requirements are marked with atomic propositions (predicates that are true or false in a given state). The implementation states are states of the microcontroller that the object code program modifies. As such, the microcontroller states includes registers, flags, and memory. The various possible values that these components can have during the execution of the object code program gives rise to the many millions of states of the implementation. To overcome this difference, WEB refinement uses the concept of a refinement map, which is a function that provided an implementation state, gives the corresponding specification state. Historically, one of the reasons that refinement-based verification is much less explored than other formal verification paradigms such as model checking is that the construction of refinement maps often requires deep understanding and intuitions about the specification and implementation [13]. However, once a refinement map is constructed, the benefit is that refinement-based verification is a very scalable approach for dealing with low-level artifacts such as real-time object code verification. This paper studies refinement maps corresponding to formal specifications related to infusion pump safety and proposes possible generic refinement map templates, which is the first step toward automating the construction of refinement maps.

The remainder of this paper is organized as follows. Section II summarizes background information. Section III details related work. Section IV describes the refinement maps and refinement map templates. Conclusions and direction for future work are noted in Section V.

II. BACKGROUND

This section explores the definition of transition systems, the definition of refinement-based verification, and the synthesis of formal specifications as key terms related to our work.

A. Transition Systems

As stated earlier, transition systems (TSs) are used to model both specification and implementation in refinement-based verification. TSs are defined below.

 ISBN: 978-1-61208-755-9

Definition 1: A TS $M = \langle S, R, L \rangle$ is a three tuple in which S denotes the set of states, $R \subseteq SXS$ is the transition relation that provides the transition between states, and L is a labeling function that describes what is visible at each state.

States are marked with Atomic Propositions (APs), which are predicates that are true or false in each state. The labeling function maps states to the APs that are true in every state. An example TS is shown in Figure 1. Here S = {S1, S2, S3, S4}, R = {(S1, S2), (S2, S4), (S4, S3), (S3, S4), (S3, S2), (S1, S3)} and, $L(S2)$ represents the atomic propositions that are true for the S2 state.

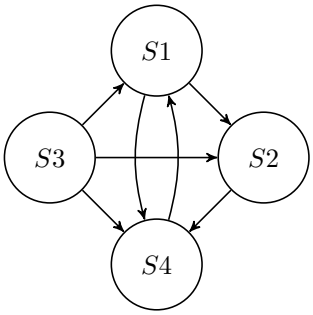


Figure 1. An example of a transition system (TS).

B. Refinement-Based Verification

Our verification process is based on the theory of Well-Founded Equivalence Bisimulation refinement. A detailed description of this theory can be found in [9]. Here, we give a very high-level overview of the key concepts. As stated earlier, WEB refinement provides a notion of correctness that can be used to check an implementation TS against a specification TS. One of the key features is that WEB refinement accounts for stuttering, which is the phenomenon where multiple but finite transitions of the implementation can match a single transition of the specification. This is a very key feature because the control code implements many functions and only some of these functions maybe relevant to the safety property being verified. Therefore, the code maybe doing a number of things that do not relate to the property and will therefore be stuttering a lot w.r.t. the specification. Another key feature of WEB refinement is refinement maps, which is the focus of this work. Refinement maps are functions that map implementation states to specification states. There is a lot of flexibility in how refinement maps can be defined. This allows for low-level implementations to be verified against high-level specifications.

Definition 2: (WEB Refinement): Let $M = \langle S, R, L \rangle$, $M' = \langle S', R', L' \rangle$, and r: S $\rightarrow$ S'. M is a WEB refinement of M' with respect to refinement map r, written $M \approx r\ M'$, if there exists a relation, B, such that $\langle\ \forall$ s $\in$ S :: sB(r.s)$\rangle$ and B is a WEB on the TS $\langle$ S $\uplus$ S', R $\uplus$ R', L $\rangle$, where L.s = L'(s) for s and S' state and L.s = L'(r.s) otherwise.

C. Synthesis of Formal Specifications

Our approach for development and study of refinement maps is based on the formal TS specifications. We have developed a previous approach to transform functional requirements into formal specifications [11]. Since this work is closely tied to the prior work, we briefly review it here. The transformation procedure is as follows: The first step of computing the TSs is to extract the APs from the requirements. We have developed three Atomic Proposition Extraction Rules (APERs) that work on the parse tree of the requirement obtained from an English language parser called Enju. A high-level procedure for specification transition system synthesis has been proposed to compute the states and transitions using the resulting list of APs under expert user supervision. Figure 2 summarizes the main steps of the synthesizing procedure.

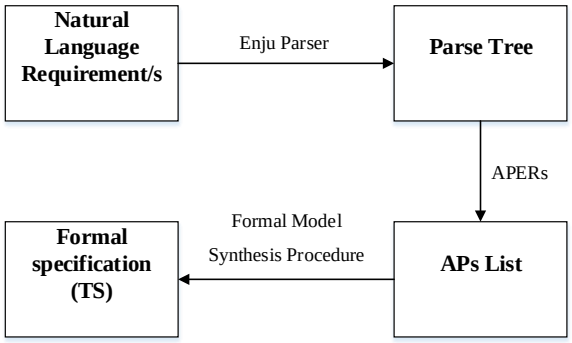


Figure 2. Formal Model synthesis procedure for Functional Requirements.

III. Related Work

This section summarizes a few works on applying refinement processes to get more concrete specifications and refinement-based verification. None of these works are applied to insulin pump formal specifications as our work. To the best of our knowledge, these are the most related state of art in this area of study.

Klein *et al.* [14] introduced a new technique called State Transition Diagrams (STD). It is a graphical specification technique that provides refinement rules, each rule defines an implementation relation on STD specification. The proposed approach was applied to the feature interaction problem. The refinement relation was utilized to add a feature or to define the notion of conflicting features.

Rabiah *et al.* [15] developed a reliable autonomous robot system by addressing A* path planning algorithm reliability issue. A refinement process was used to capture more concrete specifications by transforming High-Level specification into equivalent executable program. Traditional mathematical concepts were used to capture formal descriptions.Then, Z specification language was employed to transform mathematical description to Z schemas to get formal specifications. Z formal refinement theory was used to obtain the implementation specification.

Spichkova [16] proposed a refinement-based verification scheme for interactive real time systems. The proposed work solves the mistakes that rise from the specification problems by integrating the formal specifications with the verification system. The proposed scheme translates the specifications to a higher-order logic, and then uses the theorem prover (Isabelle) to prove the specifications. Using the refinement-based verification, this scheme validates the refinement relations between two different systems. The proposed design was tested and verified using a case study of electronic data transmission gateway.

 ISBN: 978-1-61208-755-9

A new approach that focuses on the refinement verification using state flow charts has been presented by Miyazawa *et al.* [17]. They proposed a refinement strategy that supports the sequential *C* implementations of the state flow charts. The proposed design benefited from the architectural features of model to allow a higher level of automation by retrieving the data relation in a calculation style and rendering the data into an automated system. The proposed design was tested and verified using Matlab Simulink SDK. Through the provided case study, the scheme was able to be scaled to different state charts problems.

Cimatti *et al.* proposed a contract-refinement scheme for embedded systems [18]. The contract-refinement provides interactive composition reasoning, step-wise refinement, and principled reuse refinements for components for the already designed or independently designed components. The proposed design addresses the problem of architectural decomposition of embedded systems based on the principles of temporal logic to generate a set of proof obligations. The testing and verification of the Wheel Braking System (WBS) case study show that the proposed design can detect the problems in the architectural design of the WBS.

Bibighaus [19] employed the Doubly Labeled Transition Systems (DLTS) to reason about possibilities security properties and refinement. This work was compared with three different security frameworks when applied to large class systems. The refinement framework in this work preserves and guarantees the liveness of the model by verifying the timing parameter of the model. The analysis results show that the proposed design preserves the security properties to a series of availability requirements.

IV. Refinement Maps and Refinement Map Templates

Figures 3-9 show the formal TS specification for 8 insulin pump safety requirements and the refinement map we have developed corresponding to each specification TS. The formal TS specifications were developed as part of our previous work in this area [11] [12]. As can be seen from the figures, each TS consists of a set of states and the transitions between the states. Also, each state is marked with the atomic propositions that are true in the state.

Our strategy for constructing the refinement maps is as follows. A specification state can be constructed from an implementation state by determining the APs that are true in the implementation state. If a specification has n APs, then we construct one predicate function for each AP. The predicate functions take the implementation state as input and output a predicate value that indicates if the AP is true in that state or not. Thus, the collection of such predicate functions is the refinement map.

We next discuss the refinement map for the specification in Figure 3. The safety specification from [20] is as follows: "The pump shall suspend all active basal delivery and stop any active bolus during a pump prime or refill. It shall prohibit any insulin administration during the priming process and resume the suspended basal delivery, either a basal profile or a temporary basal, after the prime or refill is successfully completed." The APs corresponding to this safety requirement are (1) BO: active bolus delivery; (2) BA: active basal delivery; (3) P: priming process; and (4) R: refill process. The refinement map however has to account for what is happening in the implementation code and relate that to the atomic propositions.

The predicate function for BO uses several variables from the code including NB: Normal Bolus and EB: Extended Bolus as there are more than one type of Bolus dose supported by the system. So the AP BO should be true if there is a NB or an EB. NB is only a flag that indicates that a normal bolus should be in progress. The actual bolus itself will continue to occur as long as a counter that keeps track of the bolus has not reached its maximum value. Therefore, for example for a normal bolus, we use a conjunction of NB and the condition that the NB counter (NB_c) is less than its possible maximum value (NB_m). We use a similar strategy for the extended bolus as well. This refinement map template works for all processes similar to a Bolus dosage delivery, such as basal dosage delivery, priming process, and refill process. Therefore, we term this refinement map template as "process template." For the basal dosage (BA AP) a number of basal profiles (BPs) are possible that accounts for BP_1 thru BP_n. TB stands for temporary basal. As can be noted from Figures 4-9, the process template accounts for a large number of predicate functions corresponding to APs.

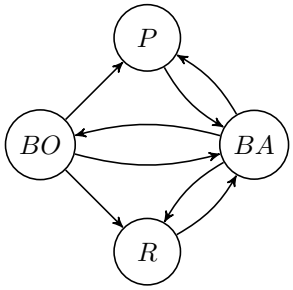
The second refinement map template is a simple one called the "projection template," which is used when the AP in the specification TS corresponds directly to a variable in the code. An example of the projection template can be found in Figure 4, where the User Reminder (UR) AP is mapped directly from a flag variable in the code that corresponds to the user reminder. A variation of this template is a boolean expression of Boolean variables in the code. An example of such an AP is the UIP AP in Figure 8.

The third refinement map template is called the "value change template," which is used when the AP is true only when a value has changed. An example use of this template can be found in Figure 4 for the CDTC AP. CDTC corresponds to the change in drug type and concentration and is true when the drug type or concentration is changed. For the drug type change, DT is the variable that corresponds to the drug type. The question here is how to track that a value has changed. The idea is to use history variables. HDT is a history variable that corresponds to the history of the drug type, i.e., the value of the drug type in the previous cycle. If HDT is not equal to DT in a code state, then we know the drug type has changed. The inequality of HDT and DT is used to construct the predicate function. For all the safety requirements analyzed, these three refinement map templates cover all the APs. Table 1 gives the expansions for all the abbreviations used in Figures 3-9, so that the corresponding refinement maps can be comprehended by the reader.

V. Conclusion and Future Work

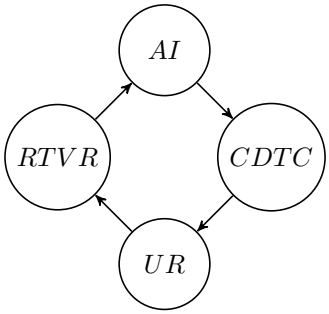
In this paper, we have developed refinement maps corresponding to the specification TSs of several infusion pump safety requirements. This is a first step in automating the construction of refinement maps. Our eventual goal is to develop a process for the construction of refinement maps. The refinement maps from this paper will be used as benchmarks to study and develop generic refinement map templates. Heuristics will be developed based on the output of the Enju parser to select a refinement map template for each atomic proposition. The development and testing of this process is part of future work.

 ISBN: 978-1-61208-755-9



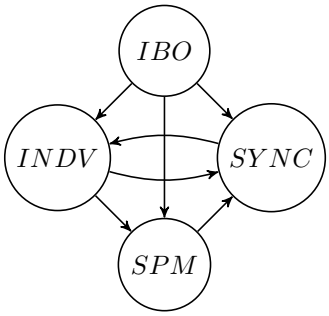
- **BO** = [NB $\wedge$ ($NB_c < NB_m$)] $\vee$ [EB $\wedge$ ($EB_c < EB_m$)]
- **P** = P $\wedge$ ($P_c < P_m$)
- **R** = R $\wedge$ ($R_c < R_m$)
- **BA** = [$BP_1 \wedge (BP_{1c} < BP_{1m})$] $\vee$ [$BP_2 \wedge (BP_{2c} < BP_{2m})$] $\vee \ldots \vee$ [$BP_n \wedge (BP_{nc} < BP_{nm})$] $\vee$ [TB $\wedge$ ($TB_c < TB_m$)]

Figure 3. A formal presentation of requirement 1.1.1 from [20] and the suggested refinement maps.



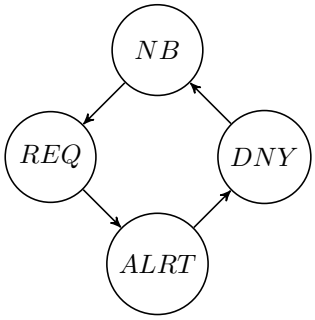
- **AI** = [$BP_1 \wedge (BP_{1c} < BP_{1m})$] $\vee$ [$BP_2 \wedge (BP_{2c} < BP_{2m})$] $\vee \ldots \vee$ [$BP_n \wedge (BP_{nc} < BP_{nm})$] $\vee$ [TB $\wedge$ ($TB_c < TB_m$)] $\vee$ [NB $\wedge$ ($NB_c < NB_m$)] $\vee$ [EB $\wedge$ ($EB_c < EB_m$)]
- **CDTC** = (DT $\neq$ HDT) $\wedge$ ($CDTC_c < CDTC_m$)
- **UR** = FLAG
- **RTVR** = (CRV $\neq$ HRV) $\wedge$ ($RTVR_c < RTVR_m$)

Figure 4. A formal presentation of requirement 1.1.3 from [20] and the suggested refinement maps.



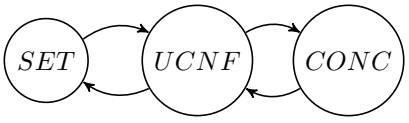
- **IBO** = [NB $\wedge$ ($NB_c < NB_m$)] $\vee$ [EB $\wedge$ ($EB_c < EB_m$)]
- **INDV** = [$BP_1 \wedge (BP_{1c} < BP_{1m})$] $\vee$ [$BP_2 \wedge (BP_{2c} < BP_{2m})$] $\vee \ldots \vee$ [$BP_n \wedge (BP_{nc} < BP_{nm})$] $\vee$ [TB $\wedge$ ($TB_c < TB_m$)] $\vee$ [NB $\wedge$ ($NB_c < NB_m$)] $\vee$ [EB $\wedge$ ($EB_c < EB_m$)]
- **SMP** = [P $\wedge$ ($P_c < P_m$)] $\vee$ [R $\wedge$ ($R_c < R_m$)]
- **SYNC** = INCAL $\wedge$ ($INCAL_c < INCAL_m$)

Figure 5. A formal presentation of requirement 1.8.2 and 1.8.5 from [20] and the suggested refinement maps.



- **NB** = NB $\wedge$ ($NB_c < NB_m$)
- **REQ** = REQ-FLAG
- **ALRT** = ALRT-FLAG
- **DNY** = CALL-FUNCT

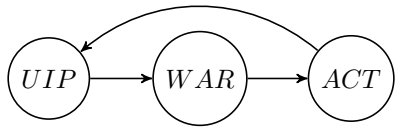
Figure 6. A formal presentation of requirement 1.3.5 from [20] and the suggested refinement maps.



- **SET** = CLRS $\vee$ [CHNS $\wedge$ ($CHNS_c < CHNS_m$)] $\vee$ RESS
- **UCNF** = FLAG
- **CONC** = [SETT $\wedge$ ($SETT_c < SETT_m$)] $\vee$ [CHNC $\wedge$ ($CHNC_c < CHNC_m$)]

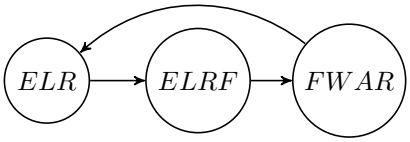
Figure 7. A formal presentation of requirement 2.2.2 and 2.2.3 from [20] and the suggested refinement maps.

 ISBN: 978-1-61208-755-9



- **UIP** = BG $\vee$ TBG $\vee$ INCR $\vee$ CORF
- **WAR** = FLAG
- **ACT** = CNFI $\vee$ [CHNI $\wedge$ $(CHNI_c < CHNI_m)$]

Figure 8. A formal presentation of requirement 3.2.5 from [20] followed by the suggested refinement maps.



- **ELR** = [EL $\wedge$ $(EL_c < EL_m)$] $\vee$ [LR $\wedge$ $(LR_c < LR_m)$]
- **ELRF** = ELF $\vee$ LRF
- **FWAR** = FLAG

Figure 9. A formal presentation of requirement 3.2.7 from [20] followed by the suggested refinement maps.

TABLE I. LIST OF ABBREVIATIONS

Abbreviation	Meaning
AI	Active Infusion
CDTC	Change Drug Type and Concentration
DT	Data Type
HDT	Historical Data Type
UR	User Reminder
RTVR	Reservoir Time and Volume Recomputed
CRV	Current Reservoir Volume
HRV	Historical Reservoir Volume
REQ-FLAG	Request Flag
CALL-FUNCT	Call-Function for Calculation
INCAL	Insulin Calculations
CLRS	Clear Settings
CHNS	Change Settings
RESS	Reset Settings
BG	Blood Glucose
TBG	Targeted Blood Glucose
INCR	Insulin to Carbohydrate ratio
CORF	Correction Factor
CNFI	Confirm Input
CHNI	Change Input
EL	Event Logging
LR	Log Retrieving
ELRF	Event Logging or Logging Retrieving Failure
ELF	Event Logging Failure
LRF	Logging Retrieving Failure

ACKNOWLEDGMENT

This publication was funded by a grant from the United States Government and the generous support of the American people through the United States Department of State and the United States Agency for International Development (USAID) under the Pakistan - U.S. Science & Technology Cooperation Program. The contents do not necessarily reflect the views of the United States Government.

REFERENCES

[1] B. Fei, W. S. Ng, S. Chauhan, and C. K. Kwoh, "The safety issues of medical robotics," Reliability Engineering & System Safety, vol. 73, no. 2, 2001, pp. 183–192.

[2] FDA, "List of Device Recalls, U.S. Food and Drug Administration (FDA)," 2018, last accessed: 2019-10-11. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfRES/res.cfm

[3] S. Quadri and S. U. Farooq, "Software testing-goals, principles, and limitations," International Journal of Computer Applications, vol. 6, no. 9, 2010, pp. 7–10.

[4] E. Miller and W. E. Howden, Tutorial, software testing & validation techniques. IEEE Computer Society Press, 1981.

[5] R. Kaivola et al., "Replacing testing with formal verification in intel coretm i7 processor execution engine validation," in Computer Aided Verification, 21st International Conference, CAV, Grenoble, France, June 26 - July 2, 2009. Proceedings, pp. 414–429. [Online]. Available: https://doi.org/10.1007/978-3-642-02658-4_32

[6] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "SLAM and static driver verifier: Technology transfer of formal methods inside microsoft," in Integrated Formal Methods, 4th International Conference, IFM, Canterbury, UK, April 4-7, 2004, Proceedings, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-3-540-24756-2_1

[7] K. Bhargavan et al., "Formal verification of smart contracts: Short paper," in Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. ACM, 2016, pp. 91–96.

[8] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, "Towards an industrial use of fluctuat on safety-critical avionics software," in International Workshop on Formal Methods for Industrial Critical Systems. Springer, 2009, pp. 53–69.

[9] P. Manolios, "Mechanical verification of reactive systems," PhD thesis, University of Texas at Austin, August 2001, last accessed: 2019-10-04. [Online]. Available: http://www.ccs.neu.edu/home/pete/research/phd-dissertation.html

[10] M. A. L. Dubasi, S. K. Srinivasan, and V. Wijayasekara, "Timed refinement for verification of real-time object code programs," in Working Conference on Verified Software: Theories, Tools, and Experiments. Springer, 2014, pp. 252–269.

[11] E. M. Al-qtiemat, S. K. Srinivasan, M. A. L. Dubasi, and S. Shuja, "A methodology for synthesizing formal specification models from requirements for refinement-based object code verification," in The Third International Conference on Cyber-Technologies and Cyber-Systems. IARIA, 2018, pp. 94–101.

[12] E. M. Al-Qtiemat, S. K. Srinivasan, Z. A. Al-Odat, and S. Shuja, "Synthesis of Formal Specifications From Requirements for Refinement-based Real Time Object Code Verification," International Journal on Advances in Internet Technology, vol. 12, Aug 2019, pp. 95–107.

[13] M. Abadi and L. Lamport, "The existence of refinement mappings," Theoretical Computer Science, vol. 82, no. 2, 1991, pp. 253–284.

[14] C. Klein, C. Prehofer, and B. Rumpe, "Feature specification and refinement with state transition diagrams," arXiv preprint arXiv:1409.7232, 2014.

[15] E. Rabiah and B. Belkhouche, "Formal specification, refinement, and implementation of path planning," in 12th International Conference on Innovations in Information Technology (IIT). IEEE, 2016, pp. 1–6.

[16] M. Spichkova, "Refinement-based verification of interactive real-time systems," Electronic Notes in Theoretical Computer Science, vol. 214, 2008, pp. 131–157.

 ISBN: 978-1-61208-755-9

[17] A. Miyazawa and A. Cavalcanti, “Refinement-based verification of sequential implementations of stateflow charts,” arXiv preprint arXiv:1106.4094, 2011.

[18] A. Cimatti and S. Tonetta, “Contracts-refinement proof system for component-based embedded systems,” Science of computer programming, vol. 97, 2015, pp. 333–348.

[19] D. L. Bibighaus, “Applying doubly labeled transition systems to the refinement paradox,” Naval Postgraduate School Monterey CA, Tech. Rep., 2005.

[20] Y. Zhang, R. Jetley, P. L. Jones, and A. Ray, “Generic safety requirements for developing safe insulin pump software,” Journal of diabetes science and technology, vol. 5, no. 6, 2011, pp. 1403–1419.

 ISBN: 978-1-61208-755-9