Translating SysML Activity Diagrams for nuXmv Verification of an Autonomous Pancreas

Orion Staskal, Josh Simac, Logan Swayne, Kristin Y. Rozier
Iowa State University
Iowa, USA
ostaskal@gmail.com, simac.josh@gmail.com, loganswayne@gmail.com, kyrozier@iastate.edu

***Abstract*—Model Based Systems Engineering (MBSE) provides a single platform capable of defining complex, multidisciplinary systems, but commonly-used tools such as Systems Modeling Language (SysML) lack the ability to formally validate and verify these systems. Symbolic model checking operates on system models of similar levels of abstraction to SysML, providing a push-button technique for ensuring the possible behavior set always obeys temporal requirements, e.g., for safe operation. We propose a translation method from SysML activity diagrams to the popular symbolic model checker nuXmv to enable their formal verification in four main steps: `main` module definition, submodule definition, activity diagram organization, and activity diagram translation. We apply this process to the Autonomous Artificial Pancreas System (AAPS) as a trade study. We then verify and validate the AAPS nuXmv model against a set of specifications derived from the AAPS safety requirements.**

***Index Terms*—MBSE, SysML, nuXmv, Cameo, Activity Diagram, Model Checking**

I. Introduction

As systems continue to become larger and more complex, so do the methodologies needed to design them. One increasingly used methodology is Model Based Systems Engineering (MBSE). MBSE provides a single platform capable of defining large, complex, multidisciplinary systems at various levels of abstraction. MBSE supports the development of requirements, design, and analysis of entire systems through all phases of the life cycle [1]; we focus on MBSE using Systems Modeling Language (SysML) [2]. As the expectations of the capabilities of modern systems have increased, so has the need to provide robust validation and verification, especially in safety-critical systems like medical devices.

The Center for Devices and Radiological Health (CDRH) division of the FDA aims to provide “safe, effective, and high-quality medical devices” through regulation, formally capturing a firm’s removal or correction of incompliant devices through recalls [3]. According to the Medical Device Recall Report for FY2003 to FY2012, medical device software is the leading cause of medical device recalls [4]. Medical device software plays a critical role in the safety of medical devices. This results in systems that are both safety-critical and highly dependent on software to execute, creating an ideal use case for model checking to provide formal system validation and verification.

Artifacts for reproducibility including all models and specifications appear at temporallogic.org/research/SESS2022

We focus on validation and verification of the Autonomous Artificial Pancreas System (AAPS). The AAPS is a hybrid closed-loop insulin pump system designed for patients with Type 1 diabetes and is similar to commercially available devices like the Tandem T:slim X2 insulin pump with Control-IQ technology and the Medtronic MiniMed 770G insulin pump system. The AAPS consists of the Continuous Glucose Monitor (CGM) and the AAPS module. The AAPS module houses the system software and includes supplies of insulin and glucagon. The goal of the AAPS is to maintain the patient’s blood glucose level within a predefined range, through the administration of insulin or glucagon while minimizing Human-Machine Interaction (HMI) through sensing and automation. The system functions in a cycle as follows: the CGM measures the blood glucose level of the patient, the CGM automatically transmits the blood glucose level to the AAPS module, then the AAPS control algorithm determines if the patient requires insulin or glucagon administration and how much to administer, the AAPS then waits for the next CGM measurement. This process repeats while the system is operational. The AAPS also supports connectivity to wearable devices, fitness applications, Blood Glucose Meter (BGM), medical databases, and medical care providers.

We used the nuXmv symbolic model checker version 2.0.0 [5] to validate and verify the AAPS model. nuXmv provides synchronous finite-state system analysis, taking as input a system model and set of operational requirements, and returning either a proof that the system always upholds the requirements or a counterexample detailing a system trace where it does not. Our translation takes advantage of the Boolean and enumeration SMV modeling language variable types [6].

Our translation from SysML to SMV also relies upon the nuXmv expression definition constructs: constant expressions, basic expressions, and simple and next expressions [6]. Since the AAPS requirements either represent system invariants or temporal pattern we can draw on timelines, we encapsulate them precisely and unambiguously using Linear Temporal Logic (LTL) [7].

We propose a process for translation from SysML to nuXmv encompassing the SysML state machine and activity diagrams used in the AAPS SysML definition. We demonstrate the process by formally validating and verifying the AAPS and release all artifacts necessary for formal verification and validation of any SysML system primarily defined by state machine diagrams and activity diagrams.

To the best of our knowledge no previous research has verified a safety-critical medical device in nuXmv from this type of SysML model, and existing translations don't fully cover state machine and activity diagrams. Some papers present details regarding SysML state machine diagram [8], [9] and block definition diagram [10], [11] conversion to NuSMV for model checking, but do not cover activity diagrams. Other papers discuss the translation of activity diagrams to probabilistic models, then utilize other types of verification tools such as the PRISM probabilistic, explicit model checker [12], [13], [14]. Applying formal methods to safety-critical medical devices is a burgeoning area of research [15], [16]. We aim to expand the use of formal medical device verification via extending SysML translations to include state machine diagram and activity diagrams and enable scalable verification via symbolic model checking.

The remainder of the paper is organized as follows. The SysML model, including a description of the AAPS system diagram types, appears in Section II. Section III provides the methodology we designed to translate the SysML state machine diagram and activity diagrams to nuXmv. Example automata and an example activity diagram further illustrate the system and translation execution. Verification and validation of the model, including example LTL specifications, appear in Section IV. Section V discusses lessons learned. Section VI concludes with avenues for future work in medical device verification.

II. Autonomous Artificial Pancreas System SysML Model

The AAPS system was developed in SysML. SysML consists of nine different types of diagrams; the AAPS uses two main diagram types: activity diagrams and state machine diagrams. The activity diagrams hold most of the usable information in the AAPS SysML model. Each activity diagram describes a small system action, and the combination of all activity diagrams accurately characterizes all possible system actions. The activity diagrams are independent of each other, and only connected through other diagram types. Typically, SysML block definition diagrams connect the activity diagrams [17]; however, the AAPS SysML model connects the activity diagrams through the state machine diagram, which defines the higher-level system states.

In addition to these two main diagram types, the AAPS model also contains a detailed set of requirements. We use these requirements to generate the formal specifications required to prove the model functioned properly.

III. Translation from SysML to nuXmv

We use a four-step translation process to translate the SysML model into nuXmv.

SysML to nuXmv Translation Process

1) `Main` Module Definition: Use the SysML state machine diagram to define the `main` module in nuXmv
2) Submodule Definition: Define submodules from the SysML state machine diagram
3) Activity Diagram Organization: Categorize activity diagrams into corresponding modules
4) Activity Diagram Translation: Convert activity diagram information into nuXmv

A. `Main` Module Definition

The overall SysML state machine diagram is the highest-level overview of the AAPS. We use this diagram to generate the automaton shown in Figure 1.

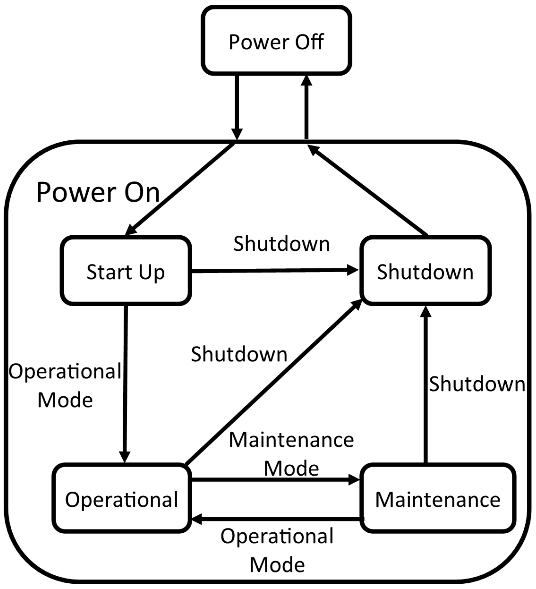

Fig. 1. Full system automaton built from the system state machine diagram

The state machine diagram shows the high-level system behavior and state switching between the different modes within the AAPS. Figure 1 corresponds exactly to the system modes in the state machine diagram. This automaton then gives a baseline for writing the `main` module, the high-level control module that defines transitions between the submodules in nuXmv. We define operational and maintenance modes as individual submodules in nuXmv due to their additional complexity, while the `main` module defines Start Up and Shutdown.

The AAPS `main` module has an enumerated `mode` variable with `StartUp`, `Operational`, `Maintenance`, and `Off` fields to define which mode the system was in. Boolean request variables denote the state of each of the mode transitions; a request must be active for the system to switch modes in the next time step. Step III-A translates only the system modes and transitions; we add all other definitions in later steps.

B. Submodule Definition

The AAPS state machine diagram also provides information on the activities within the operational and maintenance modes. We use this additional information to develop the automata in Figures 2 and 3. We model both automata as submodules within the `main` module in nuXmv.

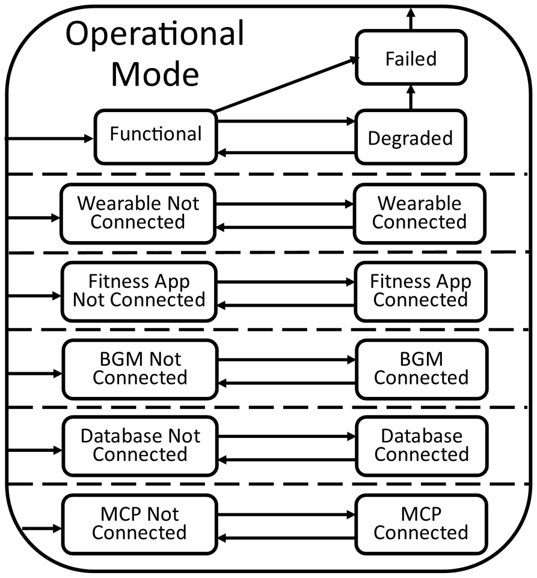

Fig. 2. Operational mode automaton showing the functional variable and the five external device connections

The operational mode consists of a functional flag to determine the AAPS state and five external device connections. The `OperationalMode` module controls the state of the functional variable and the status of each external device. We define the `Functionality` enumerated variable with four states: `Functional`, `Degraded`, `Failed`, and `Off`. We define each external device as a separate submodule within operational mode according to their respective activity diagrams.

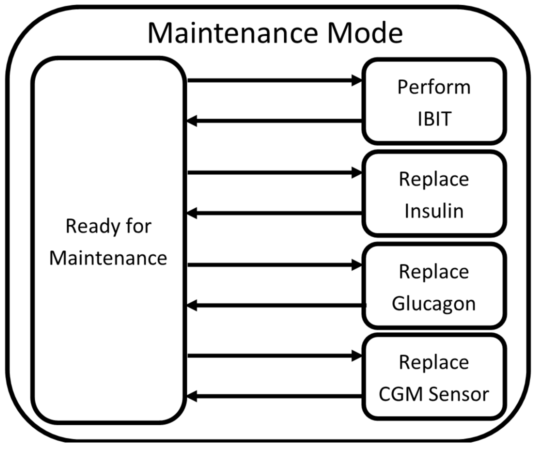

Fig. 3. Maintenance mode automaton showing the four maintenance activities and the ready state

Maintenance mode consists of four maintenance tasks and a ready state. We define a `TaskRequest` enumerated variable to control the transitions into and out of the maintenance tasks and a `MaintenanceTask` enumerated variable to set the current state of maintenance mode. Each individual maintenance task is a submodule within maintenance mode.

C. Activity Diagram Organization

We define 12 different modules from the state machine diagram: Main, OperationalMode, MaintenanceMode, CGMReplace, GlucagonRefill, IBIT, InsulinRefill, BGM, FitnessApp, MCP, MedicalDB, and WearableDevice. The OperationalMode and MaintenanceMode modules are inside of the Main module within nuXmv. The maintenance task modules are inside of the MaintenanceMode module, and the connected device modules are inside of the OperationalMode module within nuXmv. We assigned 42 of the 44 original SysML activity diagrams to a module that best characterized its functionality. We omit the two unassigned activity diagrams because they contribute little to the understanding and functionality of the final model.

D. Activity Diagram Translation

Once we assign each activity diagram to a module, we then translate the diagrams into the SMV modeling language. The activity diagram translations vary widely; some were intensive to model and required changes across several modules, while others required changing only a single variable. This difference in modeling intensity is related to the scope of the activity diagrams. The simpler activity diagrams were low level and only impacted a small area of the system, while some of the activity diagrams, such as *Respond to Critical System Failure* impacted several modules.

The *Share Data with MCP* activity diagram is one of the simpler activity diagrams; see Figure 4. It details the process of receiving a patient data request from the medical care provider (MCP), authenticating the request, and accepting the request for data or rejecting the request. We model this entire diagram using a single enumerated variable called `PatientData` in the MCP module in nuXmv. Table I lists the possible values of `PatientData`.

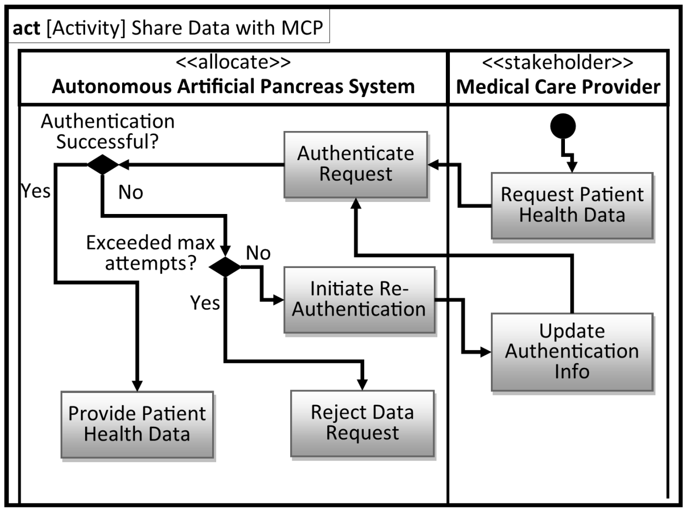
act [Activity] Share Data with MCP
<<allocate>>
Autonomous Artificial Pancreas System
<<stakeholder>>
Medical Care Provider
Authentication Successful?
Authenticate Request
Request Patient Health Data
Yes
No
Exceeded max attempts?
No
Yes
Initiate Re-Authentication
Update Authentication Info
Provide Patient Health Data
Reject Data Request

Fig. 4. SysML Share Data with MCP activity diagram example

The MCP module needs to connect to the operational mode module to exchange patient data; `PatientData` gets set to `None` by default if the MCP is not connected. If nothing is currently happening (`PatientData` is `None`), the variable

TABLE I
Patient Data Enumeration

Value Name	Description
None	No state of the activity diagram is currently happening
Requested	The MCP has received a request
Authenticating	Currently authenticating the request
Provided	The MCP accepted the request, provided data
Rejected	The MCP rejected the request

can remain in that state; a new request switches this variable to `Requested`. After spending one time step in `Requested`, it switches to `Authenticating`. The authentication part of the activity diagram is self-looping, meaning the nuXmv module always has the option to stay in that state. This means that for every step spent in the `Authenticating` mode, the model can choose between three different options: `Authenticating`, `Provided` or `Rejected`. If the data is provided, or the request is rejected, the variable will then return to the `None` state, and the process may start over again.

We continually tested our model against the written specifications to ensure correctness. We also used other methods of checking the model such as the nuXmv functions `check_fsm` and `print_reachable_states` to ensure the model did not have any deadlock states and ensure that the number of states was within reason. In the final model, no deadlock states exist, and the number of reachable states is within the expected bounds, i.e., comparable to the original SysML specification. The total number of reachable states for the AAPS nuXmv model is 10977 out of 1.83459e13 possible states. This is expected due to the large number of enumerated variable combinations that are not allowed due to the SysML model definition. There were several points during model development where the `check_fsm` function returned a deadlock state, revealing an incorrect transition or variable assignment somewhere in our model. To fully verify and validate the model requires specification checking [18].

IV. Verification and Validation of Model

A. Specification Debugging via Universal Model

The *universal model* [19] contains all possible current and future assignments for each variable; we utilized this model in nuXmv to check if the written specifications are satisfiable. A given LTL specification ϕ is satisfiable if and only if the negation $\neg\phi$ produces a counterexample representing a satisfying assignment when checked against the universal model. We checked all LTL specifications, their negations, and the conjunction of specifications for satisfiability, per the specification debugging practice introduced in [19]. This ensures that every specification is possible, not a tautology, and that all specifications can hold in the same system at the same time.

B. Specifications

We encapsulate the AAPS requirements as 29 LTL specifications [20] specifications and debug them. We also validate our model with 11 LTL and 94 CTL specifications. Our debugging effort includes code inspection and checking for satisfiability using the universal model. Model checking revealed spurious counterexamples caused by mistakes in the specification set. These are unexpected "passing" results that triggered further specification debugging. One such result was a trace where the system entered a degraded state in operational mode and switched to maintenance mode. At this point, the system was allowed to switch back to operational mode in a degraded state. This trace occurred because SysML does not specify a prioritization of tasks to be performed. We insert an assumption that the necessary maintenance tasks are to be completed and the degraded flag resolved before switching to operational mode.

We created four benchmarks using the information in the SysML model to capture the level and type of functionality that the system achieves. The verification specifications are grouped and ordered based on system functionality. Table II lists the benchmark levels and a brief description of each level. We ordered the four benchmark levels by their importance to the functionality of the AAPS from low level safety-critical features to high level convenience features. If all the specifications in a benchmark pass, then the system is considered to complete that benchmark functionality. We consider the AAPS to meet minimum safety requirements if it passes the first two benchmarks, meaning that the system works properly in operational mode and responds appropriately if it enters a degraded or failed state.

TABLE II
System Benchmarks

Benchmark	Description
Safety-Critical	Medication properly administered while operational
Failure Safety	Proper responses to system failures
Mode Transitions	Pre-transition and post-transition conditions met
Connected Devices	Alert messages sent and received

C. Results

We present a total of 132 specifications that span both validation and verification of the model. The verification specifications verify that the AAPS requirements hold for the nuXmv translation of the model, and the validation specifications validate that the nuXmv model performs all the intended functions, i.e., matches the SysML model [21]. We model-checked 105 specifications to validate our nuXmv model, while the remaining 29 specifications verify the AAPS requirements. We wrote all our specifications to ensure that nuXmv returning a `true` value would validate or verify the respective part of the system. NuXmv provides a counterexample trace for `false` results, enabling us to find errors in the model and correct them.

1) Validation: Out of our set of 105 validation specifications, 18 test specific system behaviors. These specifications cover shutdown mode, maintenance mode, operational mode, and app connectivity. The other 87 specifications serve to ensure reachability for all valid variable assignments within the AAPS nuXmv model. Table III provides a summary of the validation results.

TABLE III
SUMMARY OF VALIDATION RESULTS

Specification Type	Number of Specs	Result
Shutdown Mode	3	PASS
Maintenance Mode	10	PASS
Operational Mode	2	PASS
App Connectivity	3	PASS
Valid Variable Assignment	87	PASS

We wrote validation specifications in tandem with the translation of the model from SysML. We first wrote specifications to validate the high level system aspects, writing the rest of the validation specifications when the low-level functionality was implemented in nuXmv. The following example is one of the system behavior specifications used to validate maintenance mode.

$\Box\neg\{(IBIT = Running) \wedge [(Refill\_Insulin = Running) \vee (Refill\_Glucagon = Running) \vee (Replace\_CGM = Running)]\}$

The given LTL formula states that IBIT shall never run while insulin needs refilling, glucagon needs refilling, or the CGM needs replacing. We created three other variations of this LTL formula to validate that the AAPS nuXmv model only allows one maintenance task to run at any given time.

1) $\Box\neg\{(Refill\_Insulin == Running) \wedge [(IBIT == Running) \vee (Refill\_Glucagon == Running) \vee (Replace\_CGM == Running)]\}$
2) $\Box\neg\{(Refill\_Glucagon == Running) \wedge [(Refill\_Insulin == Running) \vee (IBIT == Running) \vee (Replace\_CGM == Running)]\}$
3) $\Box\neg\{(Replace\_CGM == Running) \wedge [(Refill\_Insulin == Running) \vee (Refill\_Glucagon == Running) \vee (IBIT == Running)]\}$

We derived these specifications directly from the state machine diagram in the SysML model. In the state machine diagram, only one state can be active at any time unless otherwise stated. In maintenance mode, only one of the maintenance activities can be in progress at any given time. We then wrote the above specifications to validate that the assertion held true through our translation into nuXmv. Most of the other validation specifications ensure that states are reachable and that all variables are used. We wrote these validation checks in pairs for each variable.

$$\mathcal{E}\Diamond FailFlag \qquad \mathcal{E}\Diamond\neg FailFlag$$

The above specifications ensure that there exists at least one state in the system where the `FailFlag` variable is enabled, and at least one state where the `FailFlag` variable is disabled. If either of these specifications return false, the variable only has one possible assignment, and either the model behavior needs to be fixed, or the variable has no function and can be removed.

Model definition and specification are two of the main takeaways from the validation specifications. The AAPS model was already a fully defined model and we did not find any major errors in the model. We did find some inconsistencies in the model that needed to be properly defined when fully verifying the AAPS. One of these inconsistencies was the exit behavior of maintenance mode. The IBIT activity diagram specifies that the system will return to operational mode if the IBIT passes with no critical failures. This conflicts with the general flow of the state machine diagram as well as the Perform AAPS System Maintenance, Enable Maintenance Activities, and Disable Maintenance Activities activity diagrams. These diagrams all show that upon finishing a maintenance activity, the system will not necessarily return to operational mode, but may stay in maintenance mode and wait for another maintenance request. We discovered this problem through the failure of a validation specification designed to test if operational mode is guaranteed after the IBIT completed if there was no shutdown request. When the specification failed, we reviewed the model and found the inconsistency. In this case, we updated the specification to match the model and the three other activity diagrams.

2) Verification: A total of 29 verification specifications, distributed between each of the system benchmarks defined in Table II verified the given AAPS requirements all hold over the AAPS as defined. Table IV shows the specification breakdown.

TABLE IV
SUMMARY OF VERIFICATION RESULTS

Benchmark	Number of Specs	Result
Safety-Critical	6	PASS
Failure Safety	7	PASS
Mode Transitions	11	PASS
Connected Devices	5	PASS

1) $\Box(((Mode = Operational) \wedge (PatientStatus == LowBS) \wedge \neg(Functionality == Failed)) \rightarrow (BloodSugarAdjustment == DeliverGlucagon))$
2) $\Box(((Mode = Operational) \wedge (PatientStatus == HighBS) \wedge \neg(Functionality == Failed)) \rightarrow (BloodSugarAdjustment == DeliverInsulin))$
3) $\Box(((Mode = Operational) \wedge (PatientStatus == Normal)) \rightarrow (BloodSugarAdjustment == None))$

These three example LTL formulas verify medication administration, the aspect of the system that is the most safety-critical. We intentionally designed the example LTL formulas to test the AAPS nuXmv model from the highest level possible. More specifically, the LTL formulas verify that when the system is operational, the patient receives glucagon when their blood sugar is low, insulin when their blood sugar is high, and nothing when their blood sugar is within the normal range. All 29 verification specifications passed, indicating that the system operates as originally designed.

V. DISCUSSION OF LESSONS LEARNED

This case study provides an example translation of a safety-critical medical system SysML model into nuXmv for formal verification and validation. Verification and validation allow the system developers to have confidence in the model before moving forward to future development. We were able to catch model inconsistencies and specify necessary system behaviors in the AAPS through this process.

Timing is one of the major difficulties when translating a model from SysML to nuXmv. SysML does not have any native support for timing or step characteristics, while nuXmv is inherently step-based with its states. We made several assumptions about the model to overcome this difficulty because we did not create the model ourselves. This timing information would be known if we were the original system designers. The timing definition is required to both translate the model and write the specifications. We wrote the specifications in tandem with the model translation to ensure that the timing assumptions were consistent across both aspects. We primarily dealt with these timing difficulties by using request variables, where a variable would activate to signal the action in the following time step.

Every SysML model will have differences depending on the system being modeled. In the AAPS model, the important system capabilities are located in the activity diagrams; however, this may not be the case for all systems. The proposed process will work to translate the state machine diagram and activity diagrams to nuXmv, but some functionality may be missed if a given model has important capabilities in other diagram types.

VI. Conclusion

This paper explored the formal verification and validation of a complex medical system SysML model through translation to nuXmv. SysML is widely used across the industry for modeling and system specification [22], making it a perfect starting point for formal model verification. The SysML semi-formal model is converted into nuXmv for formal system safety verification. nuXmv is the ideal tool for this, giving the ability to verify and validate all system safety specifications while providing plenty of modularity for any system.

This process is scalable to any system size due to the modularity provided. A further system breakdown may be required for a larger system, and a smaller system could use a more simplified approach. Symbolic model checking through nuXmv provides a more efficient approach for larger models and is more scalable than other explicit model checking methods [7]. The use of nuXmv modules also eases the additional system complexity.

This verification effort started as a project for an Applied Formal Methods course [23]; this effort demonstrates that engineers with an introductory background in formal methods can successfully apply our techniques.

A. Future Work

The methodology proposed opens a new avenue into the study of formal methods analysis in complex medical systems. The provided system translation technique can be further researched to include additional SysML elements such as a more robust requirement translation process. The provided technique is not currently easily workable into an automated tool for use with SysML models, but future revision of the process could yield a more straightforward and implementable process.

Acknowledgment

Thanks to Eric Villhauer for providing the AAPS model used in this paper. This work was supported in part by NSF CAREER Award CNS-1552934.

References

[1] Mann, *A Practical Guide to SysML: The Systems Modeling Language*. Kybernetes, 2009.

[2] M. Hause *et al.*, "The SysML modelling language," in *Fifteenth European Systems Engineering Conference*, vol. 9, pp. 1–12, 2006.

[3] "Center for devices and radiological health," 2022. Last accessed 10 April 2022.

[4] "Medical device recall report FY2003 to FY2012," 2013. Last accessed 10 April 2022.

[5] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker," in *CAV*, pp. 334–342, Springer, 2014.

[6] M. Bozzano, R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "nuXmv 2.0.0 User Manual," 2019. Last accessed 10 April 2022.

[7] K. Y. Rozier, "Linear temporal logic symbolic model checking," *Computer Science Review*, vol. 5, no. 2, pp. 163–203, 2011.

[8] M. Mahani, D. Rizzo, C. Paredis, and Y. Wang, "Automatic formal verification of SysML state machine diagrams for vehicular control systems," tech. rep., 2021.

[9] F. Mhenni, N. Nguyen, H. Kadima, and J.-Y. Choley, "Safety analysis integration in a sysml-based complex system design process," in *2013 IEEE International Systems Conference (SysCon)*, pp. 70–75, 2013.

[10] H. Wang, D. Zhong, T. Zhao, and F. Ren, "Integrating model checking with SysML in complex system safety analysis," *IEEE Access*, vol. 7, pp. 16561–16571, 2019.

[11] G. Caltais, F. Leitner-Fischer, S. Leue, and J. Weiser, "Sysml to nusmv model transformation via object-orientation," in *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*, pp. 31–45, Springer, 2016.

[12] Y. Jarraya, A. Soeanu, M. Debbabi, and F. Hassaine, "Automatic verification and performance analysis of time-constrained sysml activity diagrams," in *Engineering of Computer-Based Systems (ECBS)*.

[13] M. Debbabi, F. Hassaine, Y. Jarraya, A. Soeanu, and L. Alawneh, "Probabilistic model checking of SysML activity diagrams," in *Verification and validation in systems engineering*, pp. 153–166, Springer, 2010.

[14] S. Ouchani, O. A. Mohamed, and M. Debbabi, "A formal verification framework for SysML activity diagrams," *Expert Systems with Applications*, vol. 41, no. 6, pp. 2713–2728, 2014.

[15] R. Jetley, S. Purushothaman Iyer, and P. Jones, "A formal methods approach to medical device review," *Computer*, vol. 39, no. 4, pp. 61–67, 2006.

[16] R. P. Jetley, C. Carlos, and S. P. Iyer, "A case study on applying formal methods to medical devices: computer-aided resuscitation algorithm," *International Journal on Software Tools for Technology Transfer*, vol. 5, no. 4, pp. 320–330, 2004.

[17] S. Friedenthal and J. A. Wolfrom, "Modeling with SysML," in *INCOSE*, vol. 20, pp. 1847–1995, Wiley Online Library, 2010.

[18] S. Antoy and D. Hamlet, "Automatically checking an implementation against its formal specification," *IEEE Transactions on Software engineering*, vol. 26, no. 1, pp. 55–69, 2000.

[19] K. Y. Rozier and M. Y. Vardi, "LTL satisfiability checking," in *Model Checking Software (SPIN)*, vol. 4595 of *LNCS*, pp. 149–167, Springer-Verlag, 2007.

[20] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[21] J. S. Carson, "Model verification and validation," in *Proceedings of the winter simulation conference*, vol. 1, pp. 52–58, IEEE, 2002.

[22] A. Albers and C. Zingel, "Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of SysML," in *Smart product engineering*, pp. 83–92, Springer, 2013.

[23] K. Y. Rozier, "On teaching applied formal methods in aerospace engineering," in *Proceedings of the Formal Methods Teaching Workshop (FMTea) at the 3rd World Congress on Formal Methods*, vol. 11758 of *LNCS*, pp. 111–131, Springer, October 2019.