

# Compositional Formal Analysis Based on Conventional Engineering Models

**Tyler D. Smith** 

Adventium Labs, United States

<https://www.adventiumlabs.com>

tyler.smith@adventiumlabs.com

**Ryan Peroutka** 

Adventium Labs, United States

<https://www.AdventiumLabs.com>

ryan.peroutka@adventiumlabs.com

**Dr. Robert Edman**

Carnegie Mellon University, United States

<https://robertedman.com/>

redman@andrew.cmu.edu

---

## Abstract

Applications of formal methods for state space exploration have been successfully applied to evaluate robust critical software systems. Formal methods enable discovery of error conditions that conventional testing may miss, and can aid in planning complex system operations. However, broad application of formal methods has been hampered by the effort required to generate formal specifications for real systems. In this paper we present State Linked Interface Compliance Engine for Data (SLICED), a methodology that addresses the complexity of formal state machine specification generation by leveraging conventional engineering models to derive compositional formal state models and to generate formal assertions on the state machines. We demonstrate SLICED using the Virtual ADAPT model published by NASA and validate our results by replicating them using Simulink.

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata theory

**Keywords and phrases** State Modeling, SMV, Automata, Planning, Avionics

**Digital Object Identifier** 10.4230/LIPIcs...1

**Funding** This material is based upon work supported by the U.S. Naval Air Warfare Center, Aircraft Division under contract no. N68335-16-C-0506 and NASA Marshall Space Flight Development Center under contract No. 80NSSC19C0075. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Naval Air Warfare Center or NASA Marshall Space Flight Development Center. NAVAIR Public Release 2020-113. Distribution Statement A - “Approved for public release; distribution is unlimited.”

## 1 Introduction

Formal methods for system verification, particularly via *model checking* have been in use since the 1980s. A common use of model checking is to create a representation of the system under analysis as a *Finite State Machine (FSM)* with assertions to be verified written in *temporal logic*. A *model checker* is then used to evaluate the FSM against the temporal logic assertions, providing counterexamples for any assertions for which an exception exists. Several strategies have been developed for representing and exploring finite state machines, notably *symbolic model checking*, which represents sets of states as Boolean functions. Exploration of these functions can be accomplished via *Binary Decision Diagram (BDD)* manipulation, which treats the state model exploration as a graph traversal, or by *Bounded Model Checking (BMC)*,



© Adventium Labs;  
licensed under Creative Commons License CC-BY  
Leibniz International Proceedings in Informatics



LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which treats bounded length executions of the state model as a SATISFIABILITY (SAT) problems [5].

A detailed survey of the practice of model checking is beyond the scope of this paper (See [5] for an overview). However, a theme in model checking methodologies encountered by the authors is the need to translate or adapt one's system design to an FSM with temporal logic assertions. Many such system designs are already written as conventional engineering models (e.g., AADL, SysML, Simulink). <sup>1</sup> The SLICED methodology outlined in this paper provides a framework for generation of an FSM and associated assertions from conventional engineering models, as well as recommendations for addressing scalability concerns associated with the analysis of large engineering models.

## Background

SLICED originated as an analysis methodology for generating FSM and temporal logic assertions from Architecture Analysis and Design Language (AADL). AADL is a language for describing embedded software systems in semantically precise terms [1]. SLICED has subsequently evolved to also include generation of FSMs and temporal logic assertions from SysML and Simulink. SLICED relies on well-defined component *archetypes* to generate standardized *behavior* FSM for individual system components that can be assembled into a composite FSM (see Definition 1).

► **Definition 1.** *A component's **significant states** are states that affect or are affected by other components. In the context of SLICED, a component's **behavior** is a specification of its significant states, how external events affect its active significant state, and how its significant states or transitions between significant states affect other components.*

Advanced Diagnostic and Prognostic Testbed (ADAPT) is a testbed for evaluating electrical systems and injecting faults in a controlled manner. ADAPT consists of a power supply, batteries, sensors, and actuators. ADAPT has several hundred components [20]. NASA created Virtual ADAPT to provide a digital equivalent of ADAPT. Virtual ADAPT is a Simulink model of the majority of ADAPT [21].

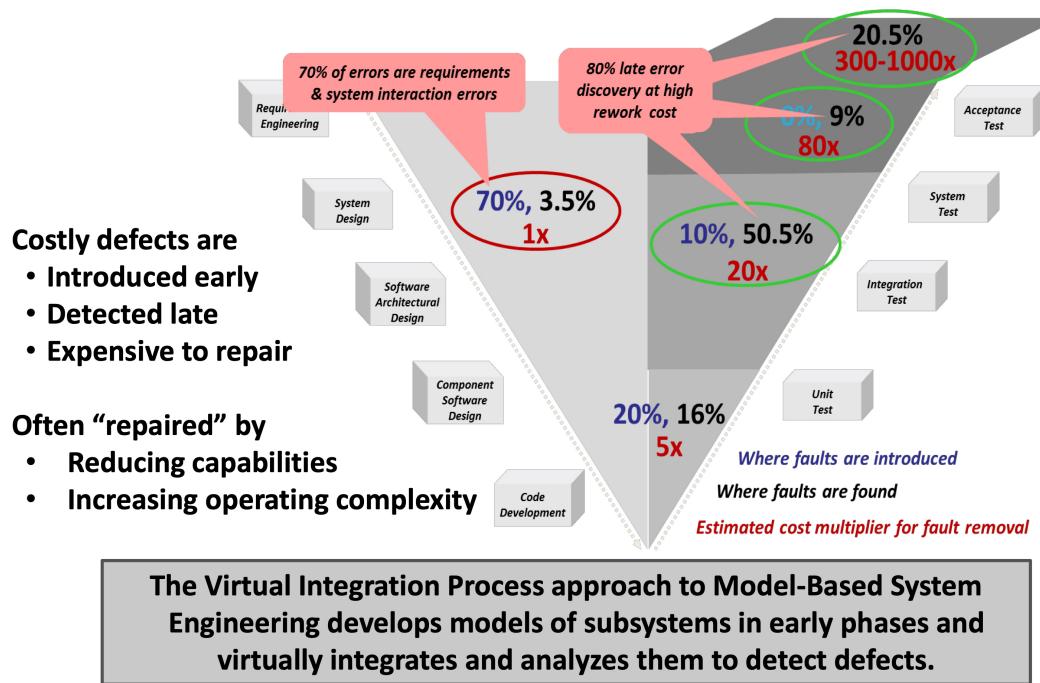
We exercise the SLICED methodology using the symbolic model checker NuSMV. NuSMV provides the ability to specify *Modules*, which are collections of variables and transitions analogous to subcomponents in a system design [11].

## Scope

This paper describes the SLICED methodology and how it applies to the Virtual ADAPT source model. We provide examples of the generated FSM as specified in the Symbolic Model Verification (SMV) language. We provide performance metrics generated from running the NuSMV solver on our generated FSM. We conclude with recommendations for future research.

---

<sup>1</sup> For the purpose of this paper, **conventional engineering models** are structured models that follow a well-defined semantic structure and describe stateful system components, but that do not explicitly provide FSMs or only provide FSMs specific components.



■ **Figure 1** Costs of incompatibilities over the lifetime of a project [22] [15] [3].

## 2 Motivation

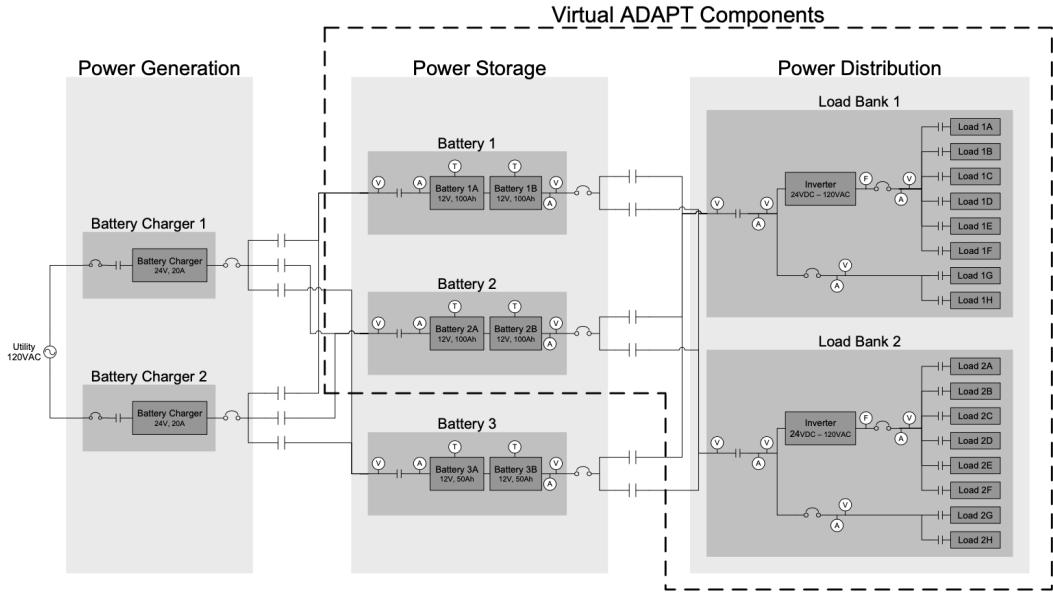
### Software Integration Cost

Software integration is a major cost driver in avionics development. Integration problems often go undetected until late in the development process, when the impact to cost and schedule to fix such issues is much higher (as shown in Figure 1).

### Limitations of Current Approaches

A variety of efforts in recent years have leveraged Model Based Systems Engineering (MBSE) to reduce the risk of software integration, such as Future Airborne Capability Environment (FACE) [16]. These efforts have garnered industry momentum toward modeling as a key element of systems engineering. However, system faults associated with the underspecified *behavior* of integrated components remain problematic. For example, in 1999 the Mars Polar Lander is believed to have crashed because of unspecified behavior of Hall Effect sensors on its landing legs [18]. The behavior was known to the landing leg engineers, but was not communicated to the software engineers. Integrated system behavior can also manifest in more subtle errors, such as timing errors in software integration due to inconsistent communication paradigm assumptions in software components found in a recent U.S. Army Study [8].

Formal modeling of component behavior has potential to address these issues. Formal modeling of the Mars Polar Lander landing legs could have communicated the legs behavior and enabled automated detection of integration errors in the interaction between the landing legs and software. Formal analysis can uncover unlikely system configurations that may be difficult to evaluate using conventional testing methods. However, there are two significant



**Figure 2** ADAPT components, with components modeled in Virtual ADAPT highlighted.

factors that complicate application of formal methods that must be addressed before formal methods can be viable in avionics systems engineering workflows. First, formal methods are outside of the expertise of many systems engineers. Although training can bridge this gap, tools and methodologies that reduce the effort required to employ formal methods could provide a low-cost-high-return alternative to extensive training. Second, current applications of formal methods analysis are not geared toward *integration* of components from different organizations or teams.

### Contributions of this Paper

The objective of this SLICED project is to make formal methods capabilities accessible to systems engineers by leveraging existing formal methods tools, MBSE environments, and models. This paper describes how SLICED addresses two core limitations of the state of art using a *compositional* approach to formulating formal methods specifications in terms of standardized *component archetypes* and a methodology for generating formal specifications and assertions from conventional engineering models. This paper also explores methods for creating formal assertions to check both for potential error conditions and to generate plans for error recovery. This paper describes a case study demonstrating a novel approach translating a Simulink model to modularized NuSMV state machine specifications. This paper uses the NASA Virtual ADAPT model for evaluation. The overall architecture for ADAPT and the elements implemented in Virtual ADAPT are shown in Figure 2 from [21].

Our approach in SLICED is based on the Architecture-Centric Virtual Integration Process (ACVIP). ACVIP provides early detection of integration errors through model integration [9]. Our first automated implementation of the SLICED methodology was based on AADL, which provides a lexicon for clearly and consistently describing embedded software components [1]. The most common language for ACVIP modeling is AADL, which is an embedded computing systems modeling language. Although the approach described in this paper does not use AADL directly (ADAPT is provided by NASA in Simulink, not AADL),

### 3 RELATED WORK

---

the component prototypes used to establish interface boundaries in ADAPT (e.g., battery and relay) are defined with intent and granularity informed by types in AADL (e.g., thread and bus). Simulink models provide a test platform for simulation-based evaluation. SLICED takes a Simulink model, abstracts the inner complexity of its components, creates an FSM, and evaluates that FSM to generate scenarios for evaluation via simulation.

The notation in this paper follows that used by Biere et al. who describe an FSM as a Kripke structure  $M = (S, I, T, L)$  and

- $S$  is a set of states.
- $I$  is a set of initial states.
- $T$  is a set of transitions between states.
- $L$  is a labeling of states with atomic propositions that hold in each state.

A path  $\pi$  in this structure is a sequence of states representing on execution of the system [5].

#### Research Hypotheses

There is an inherent loss of fidelity that occurs when translating from a highly detailed Simulink model to an abstract behavior model. This loss is acceptable because the objective of SLICED is not to perform a precise simulation of the system (Simulink already does that); rather, the objective of SLICED is to *generate* scenarios that can be evaluated using precise simulation. Claim 2 states this notion formally. There are error conditions that SLICED may not find, but those it does find are replicable in Simulink.<sup>2</sup>

▷ **Claim 2.** Let  $P$  be a source model in some language that can be executed or simulated. Let  $R$  be the set of system states possible in  $P$  and  $r$  in  $R$  be a single system state. Let  $M$  be a Kripke model decomposed from  $P$ . A *faithful* decomposition of  $P$  to  $M$  will yield a set of states  $S$  in which each state  $s$  maps to one or more states  $r$  in  $R$ . A path  $\pi$  on  $M$  is called replicable if there exists a sequence of states  $\Phi$  in  $R$  for which each state  $s_n$  maps to one or more states  $r_{j\dots k}$  and  $s_{n+1}$  maps to one or more states  $r_{j+1\dots k}$ . An error path  $\pi$  found in a FSM generated by decomposition from  $M$  is replicable via simulation or execution of  $P$ .

## 3 Related Work

### Related Work on Component Interoperability

The need to modularize software and hardware development to foster interoperability has driven a variety of research and standardization efforts. Notably, the FACE Technical Standard standardizes the concept of a Unit of Portability, a software component whose interface is defined using a standardized modeling language [16]. The well-defined boundary of FACE Units of Portability enables compositional analysis earlier than would otherwise be possible [25]. Similarly, the Hardware Open Systems Technologies (HOST) standard defines an approach for modular hardware components [14].

---

<sup>2</sup> “Error” as used in Claim 2 and throughout this paper refers to a deviation from a desired system state [2].

### Related Work on Application of Formal Methods

A variety of efforts have explored the potential of formal methods application to avionics engineering. The Defense Advanced Research Projects Agency (DARPA) High-Assurance Cyber Military Systems (HACMS) project successfully generated code from a formally verified model of assume-guarantee contracts [13]. The HACMS project used a tool called Assume Guarantee Reasoning Environment (AGREE) developed by Collins Aerospace that generates formal, invariant-oriented specifications from AADL models. AGREE generates the formal specification from the model, but is not capable of reasoning about infinite time conditions and requires that the user explicitly state assumptions and guarantees about system conditions. Vestal describes application of finite state automata to embedded software scheduling [27]. Boddy et al. generated a processor schedule using a formal problem specification derived from an AADL model [7]. Boddy's approach treats existing properties in an AADL model such as periodic thread deadlines as constraints and uses them to express a scheduling problem in terms of constraints derived from the model.

### Related Work on Model Checking

This paper uses NuSMV, a symbolic model checker that supports Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) model checking [11]. A variety of prior efforts use NuSMV. For example, Szpyrka et al. describe a process for generating NuSMV specifications from Petri Nets [26]. Most closely related to this project, Meenakshi et al describe a similar methodology for translating Simulink models to NuSMV in [19]. However, the methodology described by Meenakshi et al is focused on naive NuSMV generation from a given Simulink model.

### Related Work on Compositional State Modeling

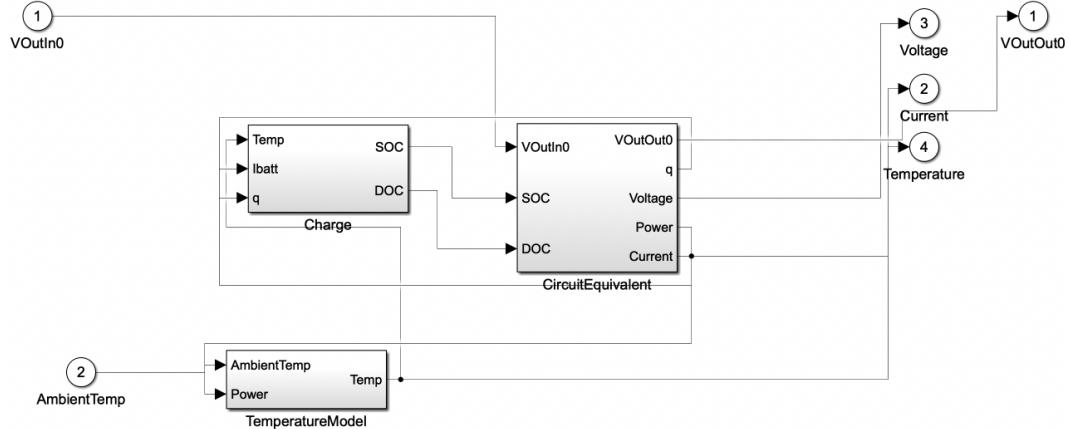
Ranganath et al. described an approach to modeling communication patterns of medical devices [24]. Ranganath's approach aims to reduce the computational burden of reasoning about integrated systems by abstracting details behind standardized interfaces. Beurdouche et al. used a compositional approach to modeling the state of Transport Layer Security (TLS) communication, creating a composite state machine by assembling state machines of individual components [4].

### SLICED as Associated to Related Work

The aim of SLICED and the focus on this paper is on generation of *abstract, interoperable* specifications to allow formal analysis of systems composed of models from multiple parties. SLICED is an extension of the interface specification approaches of FACE and HOST with an aim of enabling analysis akin to that performed by Beurdouche. SLICED uses a modularization approach similar to the communication patterns described by Ranganath, abstracting the patterns into standardized components akin to those specified by AADL.

## 4 System Model

SLICED takes a compositional approach to state modeling, requiring that the user provide state machine specifications for each component that describe its behavior as manifest at its logical boundary. Applied to a Simulink model, this means the user treats individual blocks



**Figure 3** Battery 1 in the ADAPT model. SLICED treats the battery as a black box, tracking only the fault status of the battery and the power draw.

(e.g., a battery) as black boxes, describing their behavior only in terms of what goes in to the component and what comes out.

For the ADAPT example application of SLICED, we defined five types of components, each with behavior specified at their boundary. Doing so enabled us to abstract the inner complexity of the components, losing model fidelity but enabling formal analysis. Listing 1 shows an example of a SLICED module abstracting the Simulink specification shown in Figure 3. The states listed in Listing 1 are taken from a secondary specification provided by NASA with the ADAPT model.<sup>3</sup>

```

1 MODULE Battery(output1, output2, capacity)
2 VAR
3   state : {nominal, dead, underRepair};
4 DEFINE
5   supplyingPower := (state = nominal);
6   draw := (output1.draw + output1.draw);
7 ASSIGN
8   init(state) := nominal;
9
10  next(state) := case
11    (draw > capacity) : dead;
12    ((state = dead) & (draw = 0)) : underRepair;
13    ((state = underRepair) & (draw = 0)) : nominal;
14    TRUE : state;
15  esac;
```

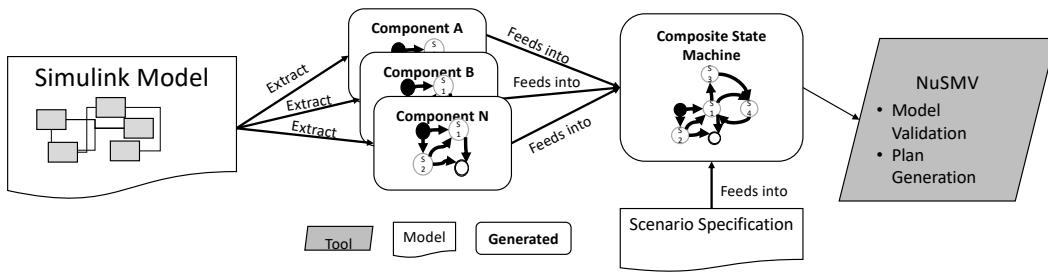
**Listing 1** Battery module specified in NuSMV

SLICED combines these state specifications with global system configuration (such as processor schedules, message routing, or intercomponent dependencies) to create a *composite state machine* (this data flow is shown in Figure 4).

► **Definition 3.** A *composite state machine* is a deterministic finite automata formed from the powerset construction of independent component state machines and system configuration.

By limiting the scope of each component's state machine to its behavioral interface, SLICED can minimize the effective state space that it must analyze. The limited scope of

<sup>3</sup> <https://github.com/nasa/VirtualADAPT/blob/master/MATLAB/ADAPTCOMPONENTS.m>



**Figure 4** SLICED Architecture

each component's state machine reduces the risk required to reconfigure an integrated system design, for example by adding or removing components as shown in Subsubsection 6.3.1, because the connections between components are well defined and clearly specified. A naive deterministic finite automata construction from multiple state machines results in the powerset construction of all of their component states, as described in Definition 3. Our definition of the composite state machine is similar to the *Interface Featured Time Automata* described by Cledou et al. [12].

A core element of the SLICED hypothesis is that discretized events in a system model can yield results that are useful for a real system. Use of a discrete clock enables us to model timed automata (strictly, *ordered* automata) where assumptions about the duration of events can be applied. A further branch of study, the application of statistical methods to timed automata, is beyond the scope of this paper.

### Approaches to Finite State Machine Generation

There are several potential levels of fidelity in translation of an electrical systems model (such as Virtual ADAPT) to a formal specification. All of these approaches treat individual components as black boxes and construct a formal specification as an assembly of black-box modeled components. Each level adds fidelity to the black box models, progressively them from “black” to “white.”

1. **State only translation:** Make no assumptions about the relationships between components. Model component states only as expressed verbatim in the source model.
2. **State and connection translation:** Assume that error states propagate along communication paths described in the source model. Assume some source model states are “good” and some are “bad” and that good and bad state properties propagate (e.g., if a component is in a bad state, anything connected to it is also in a bad state).
3. **State and connection translation with directionality:** Same as State and Connection Translation, but assume directionality in connections can be determined from the source model (that is, bad component state only propagates to *downstream* components.) This assumes that cycles *are* allowed, but that connections are directed.
4. **State and connection translation with directionality and discrete signals:** Instead of treating connections as binary on/off, treat connections as capable of holding one of a fixed set of values (e.g., value from 0...9).
5. **Discrete models of component behavior:** Model the internals of each component in the model, discretizing and bounding the values but otherwise fully translating its behavior to a formal specification.

### SLICED Pseudocode

SLICED employs approach 4, **State and connection translation with directionality and discrete signals**: This is in contrast with the higher-fidelity translation presented by Meenakshi et al. [19]. The SLICED implementation is similar for most modeling languages (prior implementations have used AADL and SysML). The SLICED implementation for ADAPT and Simulink is as follows:

1. Iterate over all elements in the simulink model (using the library `org.conqat.lib.simulink`<sup>4</sup>) in a depth-first search.
2. For each leaf block, identify the *archetype* of the block (e.g., relay, battery, junction). Create a new MODULE in SMV for that block using a standard prototype.
  - a. The ADAPT Simulink model does not define component archetypes, so we relied on component naming conventions to differentiate components. For future work an AADL model of ADAPT could enable stronger semantic consistency in component definitions.
3. As the search ascends toward the root, continue creating MODULES for each block that has a corresponding archetype.
4. For each *line* contained in each block, add input or output parameters to the appropriate modules based on the start and end points of each line. This step is similar to the process of creating Reo connectors described in [12].
5. For each *line* that traverses multiple blocks, follow the line until a SLICED-abstracted component is found. When the SLICED-abstracted component is found, add it as an input or an output parameter as described in the previous step.

## 4.1 Subsystem Merging

An additional performance improvement strategy employed by SLICED is problem reduction based on *effective* communication. In software state analysis, SLICED uses intercomponent communication configuration to limit the size of the composite state machine by excluding composite states that are not reachable using a given communication configuration, similar to the “port linking” approach described by Cledou et al. [12]. As with Cledou et al., we treat all signals uniformly (that is, in a hardware model a component’s power draw is treated as a form of communication with the component supplying the power). SLICED uses a strategy similar to Shannon’s expansion to reduce the complexity of hardware model FSMs [17]. This paper focuses on the latter, as it is most applicable to the ADAPT model.

### 4.1.1 Subsystem Merging in ADAPT

In the ADAPT model there are four banks of actuators, two with two actuators and two with six actuators. Using our actuator module definition (Listing 2), this results in 16 total actuators, each with 3 states, increasing our effective state space by a factor of  $3^{16}$ . However, inspection of the model shows that the effective communication between the each bank of actuators and the batteries and relays all goes through a single connection (e.g., the breaker EY166 in Figure 5). The result is that there are a considerable numbers of composite states of the actuators that can be discarded *a priori* by collapsing the state space of six actuators into a single *effective* actuator state, shown in Listing 3, whose states are limited to the

---

<sup>4</sup> <https://www.cqse.eu/en/products/simulink-library-for-java/overview/>

available *effective* states of the subsystem. In the case of the breaker, EY166, there are  $3^6$  possible states when the subsystem is considered naively, but there are only 13 effective states of its interaction with other components (each actuator can draw 0, 1, or 2 units of power. There are 6 actuators, so power consumption of 0..12 is possible (thus drawlimit is set to 12 in Listing 3).

```

1  MODULE Actuator(input)
2  VAR
3      state : {nominal, nopower, faultyResistance};
4  DEFINE
5      draw := case
6          (!(input.supplyingPower) | (state = nopower)) : 0;
7          (state = nominal) : 1;
8          (state = faultyResistance) : 2;
9      esac;
10 ASSIGN
11     init(state) := nominal;
12

```

**Listing 2** Actuator module specified in NuSMV

```

1  MODULE MergedActuator(input, drawlimit)
2  VAR
3      draw : 0 .. drawlimit;
4

```

**Listing 3** Merged Actuator module specified in NuSMV

### 4.1.2 Subsystem Merging Formalism

In terms of Shannon’s expansion, we divide the system problem into two sub-problems at the boundary of the connection between the system and the parent system. This divide is possible because of the compositional structure of the FSM. Subsystems that interact only at the subsystem boundary (abstracting their inner components) must define functions that aggregate their internal state (such as the addition of power consumption described in Subsubsection 4.1.1). Dependencies between components are modeled as shared variables in the FSM. For example, the power draw of an actuator is represented as an integer variable that is shared with upstream relays, breakers, or power supplies. This approach to reduced subsystem representation is similar to the *Cone of Influence (COI)* feature of NuSMV, however COI only eliminates variables not reachable from a given assertion [10]. BMC analysis of the state model may likely have taken advantage of this feature of the model and is an opportunity for future study.

Any resulting error trace (for example, a error when the total draw is 10) can serve as input to a follow-up sub-analysis on the decomposed subsystem. The error trace for the top level error will give us the value on the communication channel between the top level system and the sub-system, which we then use to generate a *second* assertion, this time on the sub-system. Because the only interaction between the sub-system and the top level system is through the aggregation component, we can be confident that none of the  $3^6 - 13$  state combinations in the sub-system could have altered our top level assessment.

## 4.2 Assertion Generation

SLICED deals with two classes of assertions over the same problem space, both are classical applications of temporal logical applied by generation from conventional engineering models. First, *SLICED* generates *error discovery* assertions, which are assertions for which a

counterexample indicates a system specificaiton error. Error discovery assertions include those asserting that the system will *not* enter a bad state from a good state (safety) or that some expected state will eventually occur (liveness). SLICED can also generate *path discovery* assertions, which are assertions that the system will *not* enter a good state from a bad state. The former is useful for detecting potential system error conditions. The latter is useful for generating plans to restore a safe system state if a bad state has occurred.

► **Definition 4.** An Error Discovery Assertion *defines a condition to be avoided, such as “the system shall never enter an unsafe state.” A “solution” found by the solver for an error discovery assertion indicates an error in the design.*

► **Definition 5.** A Path Discovery Assertion *defines a condition to be reached if possible. A “solution” found by the solver for a path discovery assertion indicates a plan for return to a good state.*

### Safety and Liveness Assertion Generation

Failure assertions include both *safety* assertions (describing things that should not happen) and *liveness* assertions (describing things that should happen). SLICED generates assertions using information provided by the source model describing the expected states of each component. For example, when dealing with software components SLICED generates *liveness* assertions for periodic threads that must always reach a *final* state. Similarly, if the source model definition of a component describes an *error* state, SLICED generates a safety assertion that the error state will never be reached.

When evaluating timing properties, SLICED uses a cyclic clock, incrementing one tick for each step taken in the state machine evaluation. Ticks are represented at the Greatest Common Denominator (GCD) of timing properties of the design (e.g., for a system which expresses performance deadlines at a granularity of one millisecond, sliced uses one millisecond for a tick). SLICED uses a cyclic clock because all variables in a FSM must be finite. The maximum value of the clock is determined by the hyperperiod of all periodic elements in the source model (e.g., if the source model has threads with periods of 100, 200, and 300ms, SLICED will use a 600ms clock cycle). When the source model provides *deadlines* for performance, SLICED evaluates the component states against its cyclic clock.

For connections between components, SLICED treats each connection as a discrete variable or as a property of an existing variable. For connections with capacity constraints, SLICED treats the connection as a counting semaphore and creates assertions that restrict the number of messages it can contain according to its upper bound.

### Path Discovery Assertion Generation

To generate a path discovery assertion, SLICED expresses the initial (failed) state of the system in a FSM with its *initial* state set to include one or more errors. The FSM differs from that used to evaluate error discovery assertions because it leaves user actions specified as non-deterministic, in effect letting the solver take on the role of the user to manipulate the system at-will. In the FSM a *user action* is a transition whose guard relies on a user input event (e.g., flipping of a switch) where the source of that event is external to the modeled system. SLICED then generates an assertion describing the negation of the desired state, including both the original faulty component and of all of the other components in the system.

### 4.2.1 SLICED Formalism

For each component in the source model, we generate a state machine  $\psi$ . Using actions  $A$  on or between these components as specified in the source model (e.g., as connections), we assemble a composite state machine  $\Psi$ .

#### State machine semantics

The formal semantics of a state machine in *SLICED* are a 6-tuple  $\psi = (S, A, T \subseteq \sigma \oplus \psi \oplus A, s_0 \in \psi, E \subseteq \psi, d \in S)$ .

$A$ —is a set of events, or Actions,

$S$ —is a set of states for the component,

$T \subseteq \sigma \oplus \sigma \oplus A$ —is the set of labeled transitions,

$s_0 \in S$ —is a designated initial state,

$\prod \subseteq \Sigma$ —is a distinguished set of error states.

We use a shorthand for transitions of  $S$ , writing  $s \xrightarrow{a} s'$  for an element  $t \in T$ .

#### Composite State Machine

The composite state machine is built from both the behavioral model associated with each component and connections between them. The composite state machine is the analytic core of *SLICED* and describes the behavior of the entire system.

Formally, the composite state machine is a timed Büchi automaton built by viewing each component as running independently with the addition of a global clock. In the case of ADAPT, which does not have specific timing constraints used by *SLICED*, *SLICED* assumes events happen in discrete time steps. Transitions between states are inter-component connections, supporting human-in-the-loop state changes through non-determinism. For example, user-settable relays have non-deterministic open and closed state transitions to account for user actions. Using powerset construction, we can build a Deterministic Finite Automata (DFA) to model the behavior of the entire system [23]. In the context of a Simulink model, a transition is an event that changes the significant state of a component as defined by its component archetype. The naïve powerset construction is exponentially larger than the Nondeterministic Finite Automata (NFA) for any individual component.

Formally, given a collection of input  $n$  behavioral state machines  $\psi^1, \dots, \psi^n$ , using a common data model  $A$ , denoted  $\psi^i = (S^i, A, T^i \subseteq \sigma^i \oplus S^i \oplus A, s_0^i \in S^i, E^i \subseteq S^i, d \in S^i)$ , the composite state machine is constructed as:

$$\Psi = (\oplus \psi_i) \oplus A$$

$c$  = a clock

$$T = \left\{ \begin{array}{l} (\dots, s, \dots) \xrightarrow{a} (\dots, s', \dots) \text{ for } s \xrightarrow{a} s' \text{ a transition of } S^i \\ \text{and } S \text{ a time slice of } S^i \end{array} \right\}$$

$$s_0 = (s_0^1, s_0^2, \dots, s_0^n, S_0) \in \Psi$$

## State Space Size

The full state space of the naïve composite state machine for a model the scale of ADAPT is combinatorially large, making it impractical to analyze. By using component abstractions with simplified behavior specifications, *SLICED* dramatically reduces the size of the relevant state space.

We achieve further reduction in the state space size by using subsystem aggregation functions table to collapse large subsystem state spaces for top-level system analysis, then expand them to generate detailed traces for particular conditions.

## 5 Evaluation Model

### Virtual ADAPT

Virtual ADAPT is a Simulink formulation of a physical testbed representing a spacecraft’s electrical power system. It allows for the injection of faults such as malfunctioning relays and sudden spikes in electrical resistance so that the system’s response can be studied. Users can interact with Virtual ADAPT in real time through the GUI flippable switches or programmatic alteration of most model parameters.

Simulink models are represented as .mdl files which define a hierarchical structure of components and the connections between them through notions of subsystems, lines (same level connections), and ports (cross-level connections). Additionally, Virtual ADAPT includes MATLAB .m files describing fault states and state-transition logic for relevant components.

### Model Statistics

Virtual ADAPT contains 11991 blocks, including the top level block.<sup>5</sup> However, Virtual ADAPT is a hierarchical model and 73% of those blocks are in the lower 7 of Virtual ADAPT’s 13 levels of block and sub-block containment (3197 blocks in the top 6 levels).<sup>6</sup>

## 6 Experimentation Evaluation

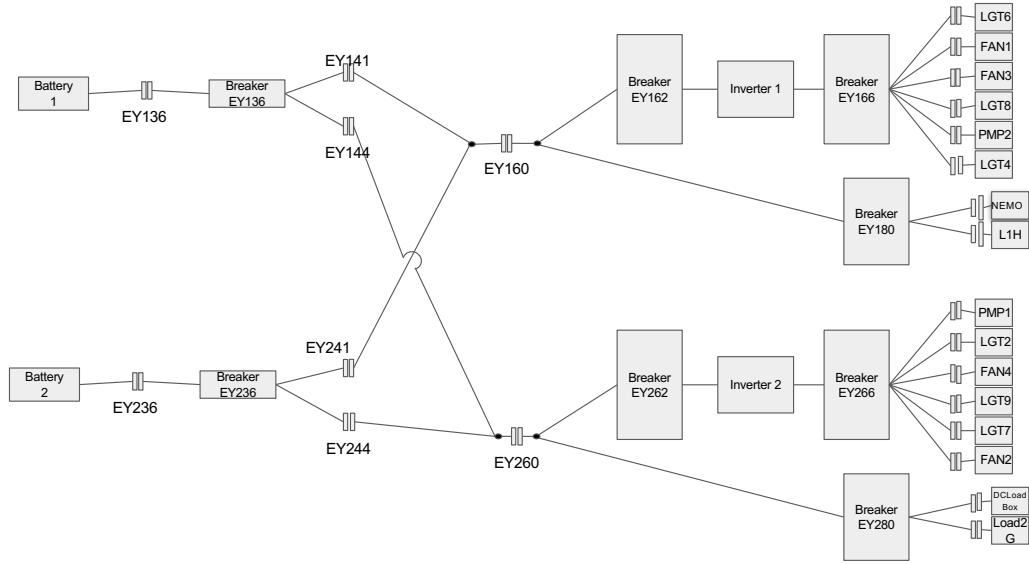
### 6.1 Experiment Overview

The objective of our experiment was to validate our approach to reducing complex system models to computationally tractable formal state specifications through application of a set of well-defined component archetypes. We determined success by whether we could run formal analysis on the generated state model and that a result found for a success or error discovery assertion in a SLICED model would also manifest in the Virtual ADAPT Simulink model.

---

<sup>5</sup> We did not include blocks related to fault injection in this count. Including fault injection brings the total block count to 14705.

<sup>6</sup> The top level block for this is the block named VirtualADAPT/VirtualADAPTv1



**Figure 5** ADAPT Components as Reduced for Analysis by SLICED.

## 6.2 Application of SLICED to ADAPT

### Core ADAPT Components

Virtual ADAPT identifies 97 of its high-level Simulink subsystems as "ADAPT Components," many of which have specified fault conditions.<sup>7</sup> These components fit into several categories: batteries, inverters, load banks, loads, circuit breakers, relays, and sensors.

Aside from loadbanks and sensors, each component is taken to be a state machine with its states and transitions straightforwardly deduced from each type's well-defined fault modes and inferred functionality. Additionally, this abstraction hides the detail associated with the electrical power transfer by describing component's electric states with discrete notions of power supply and consumption.

SLICED is concerned with behavioral interfaces. We identified 52 of the 602 "SubSystem" blocks in the top 6 levels of Virtual ADAPT as having modelable behavioral interfaces (we identified an additional 2 at the 7th level). We determined the applicability of blocks via inspection of their names and uses in ADAPT, for example by identifying any block of type "SubSystem" with a name containing "battery" as behaving according to the Battery behavior prototype.

Using the 52 identified blocks, we generated a NuSMV specification including definitions of each of our re-usable component archetypes, as well as variables defining each of the 52 blocks according to their determined archetype. We implemented connections between the blocks using additional variables internal to each type definition. Figure 5 shows all of these components and their connections.

ADAPT provides connections describing both voltage and amperage on wires between components. In creating the NuSMV representation we combined these connections into a single "draw" connection representing a unitless measure of power consumption. As with our

<sup>7</sup> <https://github.com/nasa/VirtualADAPT/blob/master/MATLAB/ADAPTComponents.m>

other design abstractions, this choice results in a loss of fidelity but maintains the capacity to reason about the system in terms of relative supply and demand.

## 6.3 Results

### 6.3.1 Performance Results

#### Test Platform

We ran our example analysis on a 64 bit Windows 10 laptop with 24GB of RAM and an Intel i7 CPU running at 2.5 GHz. We used NuSMV version 2.6.0. We measured performance of the NuSMV execution using the PowerShell `Measure-Command` cmdlet, as shown in Listing 4.

```
1 Measure-Command -Expression { nusmv .\battery_repair_full.smv > out.txt }
```

**■ Listing 4** Powershell Invocation Example.

We added a simple assertion to the state model, asserting that the draw on Battery1 would be less than one.

```
1 LTLSPEC G(Battery1.draw < 1)
```

**■ Listing 5** Simple Assertion for Performance Timing.

#### Nominal Analysis

Analysis of the generated FSM with no performance improvements applied was computationally intractable, yielding no results after 12 hours.

#### Acutator Removal

We performed several analyses of reduced models, each time starting with the full SLICED-generated model and applying a reduction or adaptation. First, we removed 12 actuators and their associated relays, as shown in Figure 6. Analysis of this model took 3 minutes 8 seconds.

#### Battery Subsystem Removal

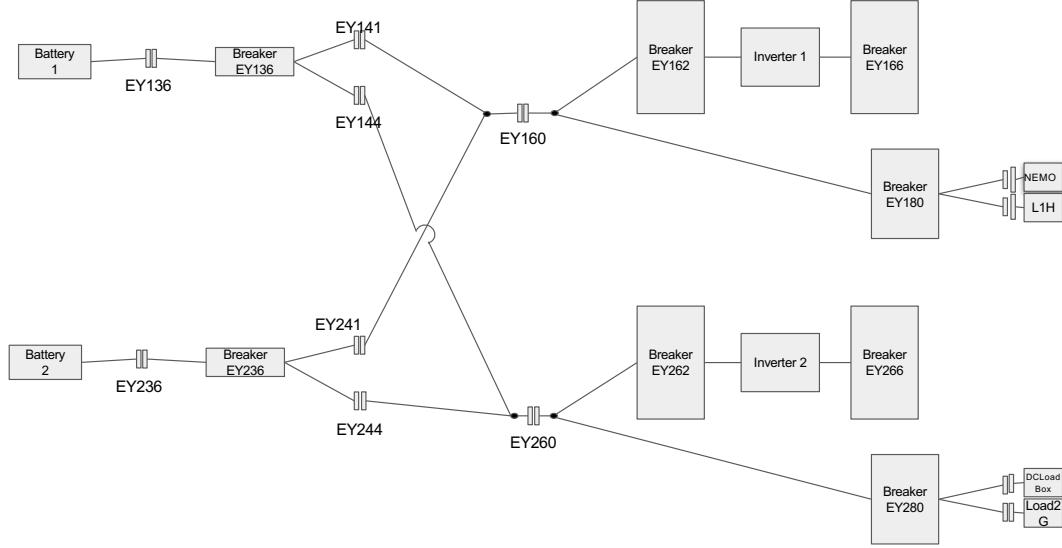
Second, we removed Battery2 and all of the 2-prefixed components, as shown in Figure 7. Analysis of this configuration took 8 minutes 34 seconds.

#### Smaller Actuator Removal

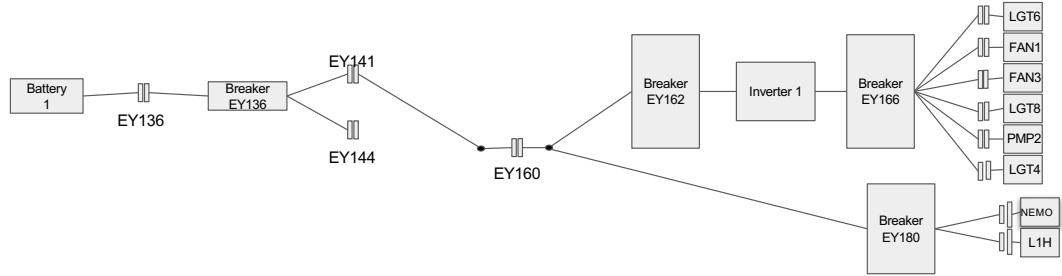
Third, we removed eight actuators and relays, as shown in Figure 8. Analysis of this configuration took 5 hours 25 minutes 59 seconds.

#### Actuator Subsystem Collapse

As discussed in Subsubsection 4.1.1 we merged the relays and actuators in two subsystems into abstractions representing only the possible power consumption of the subsystem (see Figure 9). Analysis of this configuration took 26 minutes 51 seconds. This result is of particular note because this reduction did not constitute a reduction in problem scope (as did the other reductions), yet it made the overall problem computationally tractable.



**Figure 6** ADAPT Components as generated by SLICED, with 12 actuators and relays removed.



**Figure 7** ADAPT Components as generated by SLICED, with the battery2 system and actuators removed.

This result will serve as a motivating factor for future research in performance-oriented composition of FSM and model checking problem specification based on traditional engineering models.

## 6.4 Validation

### 6.4.1 Error Detection

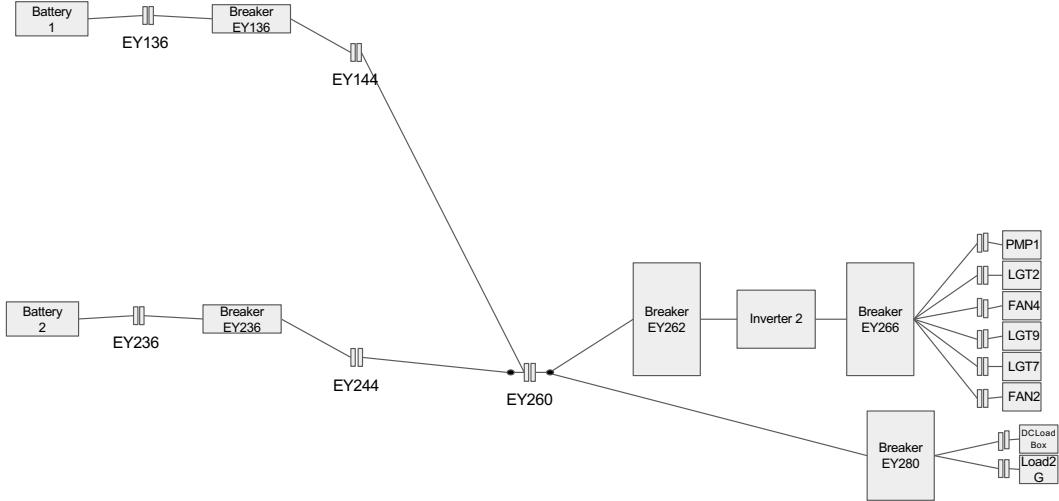
As described in Subsection 4.2, we generate assertions on the model based on component archetypes. For example, for breaker `CircuitBreakerEY162` we generate an assertion that `CircuitBreakerEY162` will remain connected (a safety assertion), as shown in Listing 6.

```
1 LTLSPEC G( CircuitBreakerEY166.state = connected )
```

**Listing 6** Breaker 162 Draw Assertion.

The SLICED model was useful in fault determination. For example, it was discovered that sufficient power draw through `CircuitBreakerEY162` could cause it to trip and become disconnected, as shown in the counter example in Listing 7.

```
1 -- specification G CircuitBreakerEY162.state = connected is false
```



**Figure 8** ADAPT Components as generated by SLICED, with the battery 1 subsystem actuators and relays removed.

```

2 -- as demonstrated by the following execution sequence
3 Trace Description: LTL Counterexample
4 Trace Type: Counterexample
5 -> State: 1.1 <-
6 CircuitBreakerEY162.state = connected
7 CircuitBreakerEY162.supplyingPower = TRUE
8 -> State: 1.2 <-
9 BankOne.draw = 11
10 CircuitBreakerEY162.draw = 11
11 -> State: 1.3 <-
12 CircuitBreakerEY162.state = broken

```

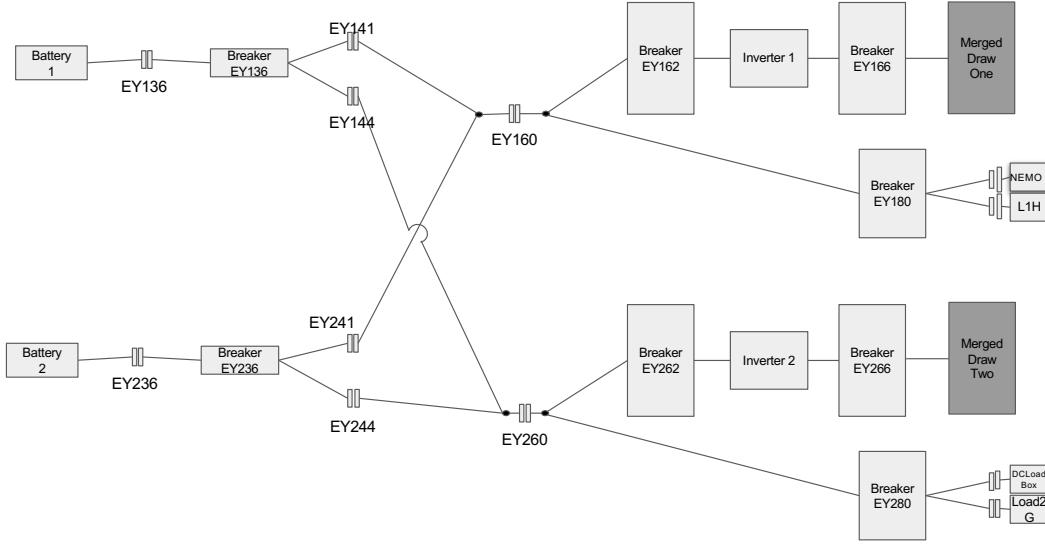
**Listing 7** Counterexample to the Breaker Draw Assertion (subset of full NuSMV output).

### Validation with Simulink

The Virtual ADAPT model is designed to allow user *injection* of error into the model.<sup>8</sup> The objective of SLICED is to find errors in the design as specified (without injection of specific errors), so to validate SLICED and Virtual ADAPT we manually introduced errors into the Virtual ADAPT model *specification* (instead of injecting them at runtime).

We replicated the example detection of a circuit breaker tripping on excessive load by changing an actuator's parameters in Virtual ADAPT to increase its power consumption (as described by the counterexample generated by NuSMV). Figure 10 shows a screenshot of Simulink's data viewer tracking the power draw from the actuator (ComputerPower:1) and the state of the circuitbreaker (CircuitBreakerEY162:3). Note that the state of the circuit breaker changes from connected (1) to disconnected (0) shortly after the increase in power consumption from the actuator.

<sup>8</sup> The ADAPT documentation uses the term “fault” however this paper uses definitions of “fault” and “error” from Avizienis et al., who define “fault” as the hypothesized cause of an error [2].



**Figure 9** ADAPT Components with two banks of actuators merged into their effective representations.

### 6.4.2 Fault Mitigation

#### Methodology

To exercise SLICED as a mechanism for generating error correction plans with ADAPT, we first used SLICED to generate a baseline ADAPT NuSMV specification, then removed some components to make it computationally tractable (as described in Subsubsection 6.3.1). Next, we manually modified that specification to set the initial state of a given component to an error state, for example by initializing Battery2 as dead (shown in Listing 8).

```

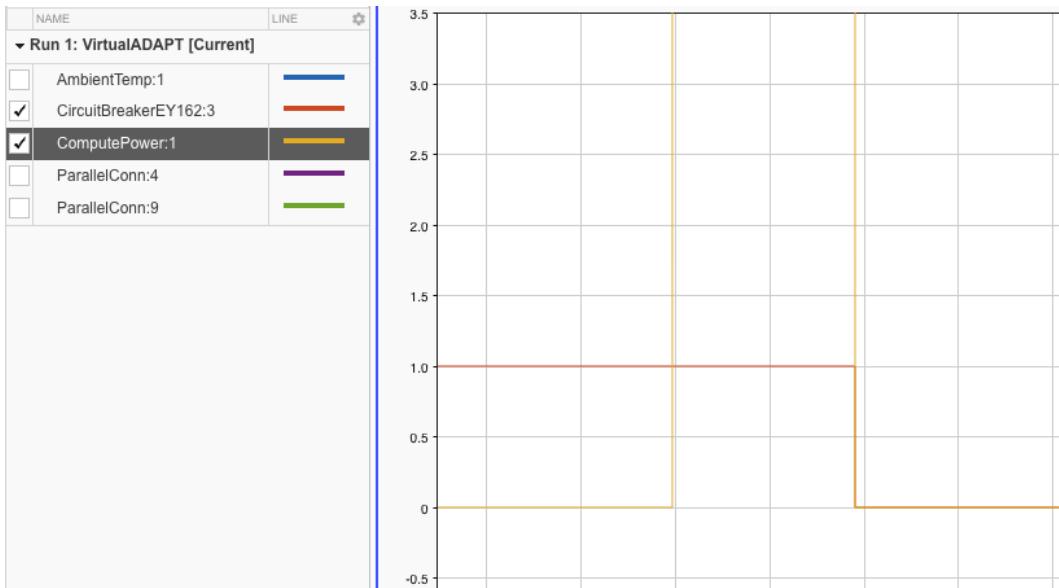
1 MODULE BatteryStartDead(output1, output2, capacity)
2 VAR
3   state : {nominal, dead, underRepair};
4 DEFINE
5   supplyingPower := (state = nominal);
6   draw := (output1.draw + output1.draw);
7 ASSIGN
8   init(state) := dead;
9
10  next(state) := case
11    (draw > capacity) : dead;
12    ((state = dead) & (draw = 0)) : underRepair;
13    ((state = underRepair) & (draw = 0)) : nominal;
14    TRUE : state;
15  esac;
```

**Listing 8** Battery module with dead initialization specified in NuSMV

Next, we specified an assertion claiming that “there is no way to return the system to a good state” such as the assertion in Listing 9.

```

1 -- NuSMV LTL assertion claiming that there is no future
2 -- state in which all of the listed
3 -- actuators are nominal and battery 1 is in a nominal state
4 -- with a power draw of two or less.
5 LTLSPEC G(
6   NEMO.state != nominal |
```



**Figure 10** Example of simulation Virtual ADAPT demonstrating a high power power draw tripping CircuitBreakerEY162.

```

7   L1H.state != nominal |
8   DCLoadBox.state != nominal |
9   Load2G.state != nominal |
10  Battery1.state != nominal |
11  Battery1.draw > 2 )

```

**Listing 9** Repair assertion for ADAPT

Finally, we ran NuSMV on the modified model. If a repair plan was possible, NuSMV generated a sequence of steps that could be taken by the user (recall that user-actionable operations are specified as non-deterministic in our model, so NuSMV can vary them at will to generate plan).

We determined that our use of NuSMV discovered repair plans that were valid in the context of the SLICED-generated FSM (that is, they were legal sequences of transitions), but that our generated FSM did not facilitate constraints on the plan discovery so that other desired system constraints are maintained throughout execution of the plan. This meant that we could generate viable repair plans for simple problems (e.g., turn a switch back on), but that attempts to generate multi-step plans were often met with technically valid but practically infeasible results, such as rapidly switching relays on and off.

## 7 Conclusions

This paper described a methodology for generating FSM that formally describe the behavior of components in a system model. As a case study, this paper used the NASA Virtual ADAPT Simulink model as a data source from which to generate NuSMV specifications. We evaluated the performance of NuSMV on our generated FSMs and replicated findings from analysis of the FSM in Simulink.

## 7.1 Results

We described a process for creation of an FSM by identifying component archetypes in the source model, creating standardized state machines for each archetype, and assembling the final FSM from component state machines.

### Performance

We determined that naive generation of an FSM from components in the Virtual ADAPT model yielded a state machine too large for brute force exploration. We determined that further decomposition of the FSM into partial models enabled exploration.

### Error Detection

We described a process for generating assertions about components in a generated FSM. We demonstrated detection of a tripping circuit breaker, and replicated the trip in Simulink.

### Repair Plan Generation

We determined that the level of fidelity and detail generated by SLICED is appropriate and viable for generation of plans, but that the assertion language and counterexample capability we evaluated with NuSMV were not ideal for plan generation because the FSM we generated did not readily account system constraints when generating a plan.

### Future Work

The results of our experiment demonstrated that a formal model of abstract components derived from a high fidelity abstraction can provide results that are replicable in the source conventional engineering model. However, further work is required to verify our claim about the completeness of the SLICED methodology. In particular, the methodology for translating Simulink blocks to NuSMV used in this experiment was based on heuristics. Stronger arguments about the correctness of the formal model will require rigorous translation. Additional work should contribute toward improving the structure of generated models for subsystem separation and on identifying solver optimization approaches to take advantage of the generated models.

Additional work is also needed to refine the generation of repair plans from conventional engineering models, either by exploring new methods of problem specification, by using different analysis tools (e.g., different model checkers or dedicated planning tools), or both.

---

### References

- 1 *Architecture Analysis & Design Language (AADL)*, jan 2017. URL: <https://doi.org/10.4271/AS5506C>, doi:<https://doi.org/10.4271/AS5506C>.
- 2 Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.
- 3 Boehm Barry et al. Software engineering economics.
- 4 B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of tls. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552, May 2015. doi:[10.1109/SP.2015.39](https://doi.org/10.1109/SP.2015.39).

- 5 Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in computers*, 58(11):117–148, 2003.
- 6 Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saár. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911 – 938, 2012. In Commemoration of Amir Pnueli. URL: <http://www.sciencedirect.com/science/article/pii/S0022000011000869>, doi:<https://doi.org/10.1016/j.jcss.2011.08.007>.
- 7 Mark Boddy. Separation platform for integrating complex avionics (spica) final report. December 2013. URL: [https://nari.arc.nasa.gov/sites/default/files/Boddy\\_SPICA\\_PhaseI\\_FinalReport\\_r2.pdf](https://nari.arc.nasa.gov/sites/default/files/Boddy_SPICA_PhaseI_FinalReport_r2.pdf).
- 8 Alex Boydston, Peter Feiler, Steve Vestal, and Bruce Lewis. Joint common architecture (jca) demonstration architecture centric virtual integration process (acvip) shadow effort. <https://www.army.mil/e2/c/downloads/414601.pdf>, June 2015.
- 9 Alex Boydston, Peter Feiler, Steve Vestal, and Bruce Lewis. Architecture centric virtual integration process (acvip): A key component of the dod digital engineering strategy. 2019. URL: <https://www.adventiumlabs.com/our-work/publications/2019/architecture-centric-virtual-integration-process-acvip-key-component-dod>.
- 10 Roberto Cavada, Alessandro Cimatti, Charles Arthur Jochim, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Andrei Tchaltsev. Nusmv 2.4 user manual. *CMU and ITC-irst*, 2005.
- 11 Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: A new symbolic model verifier. In *International conference on computer aided verification*, pages 495–499. Springer, 1999.
- 12 Guillermina Cledou, José Proen  a, and Luis Soares Barbosa. Composing families of timed automata. In *International Conference on Fundamentals of Software Engineering*, pages 51–66. Springer, 2017.
- 13 Darren Cofer, Andrew Gacek, John Backes, Michael Whalen, Lee Pike, Adam Foltzer, Michal Podhradsky, Gerwin Klein, Ihor Kuz, June Andronick, Gernot Heiser, and Douglas Stuart. A formal approach to constructing secure air vehicle software. *Computer*, 51:14–23, 11 2018. doi:[10.1109/MC.2018.2876051](https://doi.org/10.1109/MC.2018.2876051).
- 14 Georgia Tech Applied Research Corporation. Hardware open systems technologies, February 2016.
- 15 D. Galin. *Software Quality Assurance: From Theory to Implementation*. Alternative Etext Formats. Pearson Education Limited, 2004. URL: <https://books.google.com/books?id=p5jDETUc2K8C>.
- 16 The Open Group. Technical standard for future airborne capability environment, edition 3.0, November 2017. URL: <https://publications.opengroup.org/c17c>.
- 17 Edmund M. Clarke Jr. Symbolic model checking with bdds.
- 18 Nancy Leveson. The role of software in spacecraft accidents. *Journal of Spacecraft and Rockets - J SPACECRAFT ROCKET*, 41, 07 2004. doi:[10.2514/1.11950](https://doi.org/10.2514/1.11950).
- 19 B Meenakshi, Abhishek Bhatnagar, and Sudeepa Roy. Tool for translating simulink models into input language of a model checker. In *International Conference on Formal Engineering Methods*, pages 606–620. Springer, 2006.
- 20 Ole J Mengshoel, Adnan Darwiche, Keith Cascio, Mark Chavira, Scott Poll, and N Serdar Uckun. Diagnosing faults in electrical power systems of spacecraft and aircraft. In *AAAI*, pages 1699–1705, 2008.
- 21 NASA. Virtualadapt, October 2017. URL: <https://github.com/nasa/VirtualADAPT/blob/master/docs/VirtualADAPT.pdf>.
- 22 Strategic Planning. The economic impacts of inadequate infrastructure for software testing. 2002.
- 23 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114, 1959. doi:[10.1147/rd.32.0114](https://doi.org/10.1147/rd.32.0114).

- 24 Venkatesh-Prasad Ranganath, Yu Kim, John Hatcliff, and Robby. Communication patterns for interconnecting and composing medical systems. volume 2015, 08 2015. doi:10.1109/EMBC.2015.7318707.
- 25 Tyler Smith, Rand Whillock, Robert Edman, Bruce Lewis, and Steve Vestal. Lessons learned in inter-organization virtual integration. In *Aerospace Systems and Technology Conference*, 2018. URL: <https://www.sae.org/publications/technical-papers/content/2018-01-1944/>, doi:<https://doi.org/10.4271/2018-01-1944>.
- 26 Marcin Szpyrka, Agnieszka Biernacka, and Jerzy Biernacki. Methods of translation of petri nets to nusmv language. In *CS&P*, pages 245–256, 2014.
- 27 Steve Vestal. Modeling and verification of real-time software using extended linear hybrid automata. In *NASA Langley Formal Methods Workshop*, 06/2000 2000.