Architectural Modeling and Analysis for Safety Engineering

Danielle Stewart[1], Michael W. Whalen[1], Darren Cofer[2], and Mats P.E. Heimdahl[1]

[1] University of Minnesota
Department of Computer Science and Engineering
`whalen, dkstewar, heimdahl@cs.umn.edu`
[2] Rockwell Collins
`darren.cofer@rockwellcollins.com`

Abstract. Architecture description languages such as AADL allow systems engineers to specify the structure of system architectures and perform several analyses over them, including schedulability, resource analysis, and information flow. In addition, they permit system-level requirements to be specified and analyzed early in the development process of airborne and ground-based systems. These tools can also be used to perform safety analysis based on the system architecture and initial functional decomposition.

Using AADL-based system architecture modeling and analysis tools as an exemplar, we extend existing analysis methods to support system safety objectives of ARP4754A and ARP4761. This includes extensions to existing modeling languages to better describe failure conditions, interactions, and mitigations, and improvements to compositional reasoning approaches focused on the specific needs of system safety analysis. We develop example systems based on the Wheel Braking System in SAE AIR6110 to evaluate the effectiveness and practicality of our approach.

Keywords: Model-based systems engineering, fault analysis, safety engineering

1 Introduction

System safety analysis techniques are well established and are a required activity in the development of commercial aircraft and safety-critical ground systems. However, these techniques are based on informal system descriptions that are separate from the actual system design artifacts, and are highly dependent on the skill and intuition of a safety analyst. The lack of precise models of the system architecture and its failure modes often forces safety analysts to devote significant effort to gathering architectural details about the system behavior from multiple sources and embedding this information in safety artifacts, such as fault trees.

While model-based development (MBD) methods are widely used in the aerospace industry, they are generally disconnected from the safety analysis process itself. Formal model-based systems engineering (MBSE) methods and tools [3, 7, 8, 20, 21, 25] now permit system-level requirements to be specified and analyzed early in the development process. These tools can also be used to perform safety analysis based on the system architecture and initial functional decomposition. Design models from which aircraft systems are developed can be integrated into the safety analysis process to help guarantee accurate and consistent results. This integration is especially important as the amount of safety-critical hardware and software in domains such as aerospace, automotive, and medical devices has dramatically increased due to desire for greater autonomy, capability, and connectedness.

Architecture description languages, such as SysML [10] and the Architecture Analysis and Design Language (AADL) [1] are appropriate for capturing system safety information. There are several tools that currently support reasoning about faults in architecture description languages, such as the AADL error annex [17] and HiP-HOPS for EAST-ADL [6]. However, these approaches primarily use *qualitative* reasoning, in which faults are enumerated and their propagations through system components must be explicitly described. Given many possible faults, these propagation relationships become complex and it is also difficult to describe temporal properties of faults that evolve over time (e.g., leaky valve or slow divergence of sensor values).

In earlier work, University of Minnesota and Rockwell Collins developed and demonstrated an approach to model-based safety analysis (MBSA) [13,15,16] using the Simulink notation [19]. In this approach, a behavioral model of (sometimes simplified) system dynamics was used to reason about the effect of faults. We believe that this approach allows a natural and implicit notion of fault propagation through the changes in pressure, mode, etc. that describe the system's behavior. Unlike qualitative approaches, this approach allows uniform reasoning about system functionality and failure behavior, and can describe complex temporal fault behaviors. On the other hand, Simulink is not an architecture description language, and several system engineering aspects, such as hardware devices and non-functional aspects cannot be easily captured in models.

This paper describes our initial work towards a behavioral approach to MBSA using AADL. Using assume-guarantee compositional reasoning techniques, we hope to support system safety objectives of ARP4754A and ARP4761. To make these capabilities accessible to practicing safety engineers, it is necessary to extend modeling notations to better describe failure conditions, interactions, and mitigations, and provide improvements to compositional reasoning approaches focused on the specific needs of system safety analysis. These extensions involve creating models of fault effects and weaving them into the analysis process. To a large extent, our work has been an adaptation of the work of Joshi et. al to the AADL modeling language.

To evaluate the effectiveness and practicality of our approach, we developed an architectural model of the Wheel Braking System model in SAE AIR6110. Starting from a reference AADL model constructed by the SEI instrumented with qualitative safety analysis information [9], we added behavioral contracts to the model. In so doing, we determine that there are errors related to (manually constructed) propagations across components, and also an architecture that contains single points of failure. We use our analyses to find these errors.

2 Example: Wheel Brake System

As a preliminary case study, we utilized the Wheel Brake System (WBS) described in [2] (previously found in ARP4761 Appendix L). This ficticious aircraft system was developed to illustrate the design and safety analysis principles of ARP4754A and ARP4761. The WBS is installed on the two main aircraft landing gears and is used during taxi, landing, and rejected take off. Braking is either commanded manually using brake pedals or automatically by a digital control system with no need for the pedals (autobrake). When the wheels have traction, the autobrake function will provide a constant smooth deceleration.

Each wheel has a brake assembly that can be operated by two independent hydraulic systems (designated green and blue). In normal braking mode, the green hydraulic system operates the brake assembly. If there is a failure in the green hydraulics, the system switches to alternate mode which uses the blue hydraulic system. The blue system is also supplied by an accumulator which is a device that stores hydraulic pressure that can be released if both of the primary hydraulic pumps (blue and green) fail. The accumulator supplies hydraulic pressure in Emergency braking mode.

Switching between the hydraulic pistons and pressure sources can be commanded automatically or manually. If the hydraulic pressure in the green supply is below a certain threshold, there is an automatic switchover to the blue hydraulic supply. If the blue hydraulic pump fails, then the accumulator is used to supply hydraulic pressure.

In both normal and alternate modes, an anti-skid capability is available. In the normal mode, the brake pedal position is electronically fed to a computer called the Braking System Control Unit (BSCU). The BSCU monitors signals that denote critical aircraft and system states to provide correct braking function, detect anomalies, broadcast warnings, and sent maintenance information to other systems.

2.1 Nominal System Model

The WBS AADL model of the nominal system behavior consists of mechanical and digital components and their interconnections, as shown in Figure 1. The following section describes this nominal model from which the fault model was generated.

Wheel Braking System (WBS) The highest level model component is the WBS. It consists of the BSCU, green and blue hydraulic pressure lines (supplied by the green pump and blue pump/accumulator respectively), a Selector which selects between normal and alternate modes of hydraulic pressure, and the wheel system. The WBS takes inputs from the environment including PedalPos1, AutoBrake, DecRate, AC_Speed, and Skid. All of these inputs are forwarded to the BSCU to compute the brake commands.

Braking System Control Unit (BSCU) The BSCU is the digital component in the system that receives inputs from the WBS. It also receives feedback from the green and blue hydraulic lines and two power inputs from two separate power sources. The BSCU is composed of two command and monitor subsystems each powered independently from separate power sources. The pedal position is provided to these units and when skidding occurs, the command and monitor units will decrease the pressure to the brakes. The

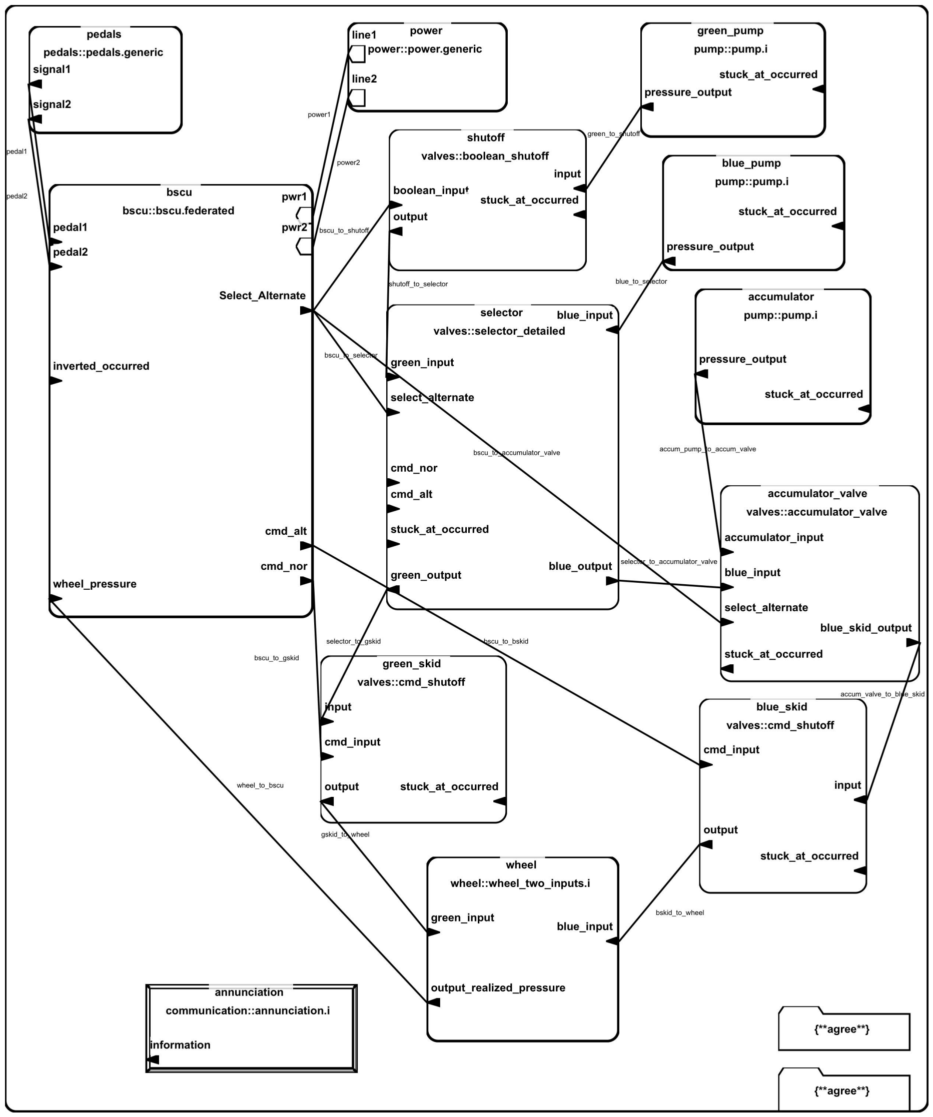

Fig. 1. AADL Simple Model of the Wheel Brake System

command unit regulates the pressure to the brakes in the green hydraulic line through the command cmd_nor. Computing this command requires both the brake requested power and the skid information. The command unit also regulates the pressure in the blue hydraulic line in order to prevent skidding which it does through the cmd_alt command. The monitor unit checks the validity of the command unit output.

The BSCU switches from normal to alternate mode (blue hydraulic system) when the output from either one of its command units is not valid or the green hydraulic pump is below its pressure threshold. Once the system has switched into alternate mode, it will not switch back into normal mode again.
Once the system has switched into alternate mode, it will not switch into normal mode again.

Hydraulic Pumps There are three hydraulic pumps in the system, green pump (normal mode), blue pump (alternate mode), and accumulator pump (emergency mode). Each pump provides pressure to the system and is modeled in AADL as a floating point value.

Shutoff Valve The shutoff valve is situated between the green pump and the selector. It receives an input from the BSCU regarding valve position and regulates the pressure coming through the green pipe accordingly.

Selector Valve The selector receives inputs from the pumps regarding pressure output and the BSCU regarding which mode the system is in. It will output the appropriate pressure from green, blue, or accumulator pump. An added requirement of the selector system is that it will only output pressure from one of these sources. Thus, the case of having pressure supplied to the wheels from more than one pump is avoided. The Selector takes the two pipe pressures (green and blue) as input, selects the system with adequate pressure and blocks the system with inadequate pressure. If both systems have pressure greater than the threshold, the AADL selects normal mode as the default.

Skid Valves The blue_skid and green_skid valves receive input from the selector as pressure coming through the respective pipes as well as input from the BSCU that commands normal or alternate mode. The skid valves will use these inputs to choose between the green or the blue pressure to send to the wheel.

2.2 Modeling Nominal System Behavior

In order to reason about behaviors of complex system architectures, we have developed a compositional verification tool for AADL models. Our tool, the *Assume-Guarantee Reasoning Environment* (AGREE) [8] is based on *assume-guarantee* contracts that can be added to AADL components. The language used for contract specification is based on the LUSTRE dataflow language [11]. The tool allows scaling of formal verification to large systems by splitting the analysis of a complex system architecture into a collection of verification tasks that correspond to the structure of the architecture.

We use AGREE to specify behavioral contracts corresponding to the behaviors expected of each of the WBS components. An example of a contract is shown in Figure 2.

```
annex AGREE {**

eq nominal_Select_Alternate : bool;

eq pedals_pressed: bool = (pedal1.val > 0.0) and (pedal2.val > 0.0);

eq skid_active: bool = (cmd_nor.activate_antiskid) or (cmd_alt.activate_antiskid);

eq commanded_pressure: bool = (cmd_nor.val > 0.0) or (cmd_alt.val > 0.0);

guarantee "If pedals pressed, no skid, and wheel pressure is nonexistant, then
          select alternate should be true" :
      (nominal_Select_Alternate =
       faults.historically(false -> not (pre(pedals_pressed and not(skid_active)) =>
          wheel_pressure.val > 0.0)));

guarantee "Pedals pressed and no skid and normal implies pressure commanded" :
       pedals_pressed and (not skid_active) => commanded_pressure;

guarantee "Alternate pressure and normal pressure don't occur simultaneously." :
       not(cmd_alt.val > 0.0 and cmd_nor.val > 0.0);

**};
```

Fig. 2. AGREE Contract for BSCU

3 Model-Based Safety Analysis

A model-based approach for safety analysis was proposed by Joshi et. al in [13–15]. In this approach, a safety analysis system model (SASM) is the central artifact in the safety analysis process, and traditional safety analysis artifacts, such as fault trees, are automatically generated by tools that analyze the SASM.

The contents and structure of the SASM differ significantly across different conceptions of MBSA. We can draw distinctions between approaches along several different axes. The first is whether models and notations are purpose-built for safety analysis (such as AltaRica [22], smartIflow [12] and xSAP [4]) vs. those that extend existing system models (ESM) (HiP-HOPS [6], the AADL error annex [24]). A second dimension involves the richness of the modeling languages used to represent failures. Most existing safety analysis languages only support model variables types drawn from small discrete domains (which we call *discrete*); the xSAP platform is a notable exception that allows *rich types*. Another dimension whether *causal* or *non-causal* models are allowed. Non-causal models allow simultaneous (in time) bi-directional failure propagations; currently only AltaRica [22] and smartIflow [12] allow this. Yet another dimension involves whether analysis is *compositional* across layers of hierarchically-composed systems or *whole-system*.

In this section, we will focus on the dimension of failure propagation, and contrast failure logic modeling (FLM) vs. failure effect modeling (FEM) [18]. In FLM, *failures* are propagated between components explicitly and the analysis proceeds by determining the likelihood of failures reaching system boundaries. In FEM, failures propagate by changing the system dynamics, which may cause the system behavior to visibly change. Our approach is an extension of AADL (ESM), richly-typed, causal, compositional, mixed FLM/FEM approach. We believe this is in a unique area of the trade space compared to other state-of-the-art MBSA approaches.

3.1 Failure Logic Modeling (FLM) Approaches

The FLM approach focuses on faults rather than constructing a model of system dynamics. We illustrate this approach with the AADL error model annex [24] that can be used to describe system behaviors in the presence of faults. This annex has facilities for defining *error types* which can be used to describe *error events* that indicate faults, errors, and failures in the system (the term *error* is used generically in the annex to describe faults, errors, and failures). The behavior of system components in the presence of errors is determined by state machines that are attached to system components; these state machines can determine error propagations and error composition for systems created from various subcomponents.

Error types in this framework are a set of enumeration values such as NoData, BadData, LateDelivery, EarlyDelivery, TimingError, and NoService. These errors can be arranged in a hierarchy. For example, LateDelivery and EarlyDelivery are subtypes of TimingError. The errors do not have any information (other than their type) associated with them. AADL includes information on the bindings of logical components (processes, threads, systems) and their communication mechanisms onto physical resources (memories, processors, busses), and the error annex uses this information to describe how physical failures can manifest in logical components.

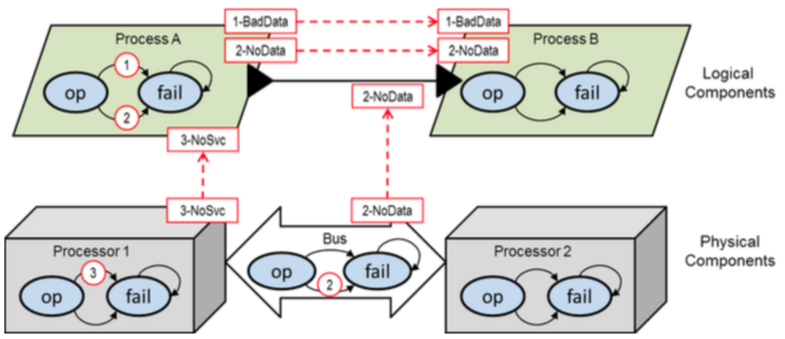

Fig. 3. Example of Error Model Information and Propagation

An example is shown in Figure 3 Errors are labeled with error types: 1-BadData, 2-NoData, 3-NoSvc. Failure events that can cause a component to fail are labeled with the corresponding error number. The error behavior of components is described by their state machines. Note that while all state machines in Figure 2 have two states, they can be much more complex. The dashed arrows indicate propagations describing how failures in one component can cause other components to fail. For example, failures in the physical layer propagate to failures in the associated logical components.

Although the error model annex is very capable, it is not closely tied to the behavioral model of components or their requirements. For example, in the wheel braking system (WBS) example [23], it is possible that hydraulic system valves can fail open or fail closed. In fail closed, downstream components receive no flow and upstream pipes may become highly pressurized as a natural consequence of the failure. Physical models of these behavioral relationships often exist that can propagate failures in terms of the behavioral relationships between components. However, with the AADL error model annex, the propagations must be (re)specified and defined for each component. This re-specification can lead to inconsistencies between physical models and error annex models. In addition, the physical relationships between failures can be complex and may not be describable using enumeration values, leading to additional inconsistencies between the behavior of the physical phenomena and the behavior of the error model.

3.2 Failure Effect Modeling (FEM) Approaches

In a failure effect modeling approach, the analysis starts from a *nominal* model of the system that describes the system behavior when no faults are present. To perform safety analysis, we then also formalize the fault model. The fault model, in addition to common failure modes such as *non-deterministic*, *inverted*, *stuck_at* etc, could encode information regarding fault propagation, simultaneous dependent faults and fault hierarchies, etc. After specifying the fault model and composing it with the original system model, the safety analysis involves verifying whether the safety requirements hold in presence of the faults defined in the fault model.

In this approach, a safety engineer can model different kinds of fault behavior: e.g., stuck-at, ramp-up, ramp-down, and nondeterministic, and then *weave* these fault models into the nominal model. The language for describing faults is extensible, allowing engineers to define a catalog of faults appropriate for their domain. In addition, the weaving process allows error propagation between unconnected components within a system model [14]. This allows consideration of physical aspects (e.g., proximity of components, shared resources such as power) that may not be present in a logical system model but can lead to dependent failures. In addition, it allows propagation of faults in the reverse direction of the model data flow. This can occur when physical components have coupling such as back-pressure in fluid systems or power surges in the opposite direction of communication through connected components. Finally, it is possible to create fault mediations to describe the output in the presence of multiple simultaneous faults.

A safety analysis system model can be used for a variety of simulations and analyses. Modeling allows trivial exploration of *what-if* scenarios involving combinations of faults through simulations. The current AADL tool suite contains a graphical symbolic simulator that allows for forward and back-stepping through different failure scenarios. In addition it contains a test-case generator that can automatically generate such scenarios. For more rigorous analyses, we can use model checking tools to automatically prove (or disprove) whether the system meets specific safety requirements. As we will demonstrate on the WBS, an engineer first verifies that safety properties hold on the nominal system, an idealized model of the digital controller and the mechanical system containing no faults. Once the nominal model is shown to satisfy the safety property, the behavior of the fault-extended model can be examined to examine its resilience to faults.

4 Architectural Failure Effect Modeling for the WBS

We illustrate our FEM approach on the Wheel Braking System. Starting from the nominal model described in Section 2.1, we first determine whether a given safety property of interest holds on a fault-free instance of the model. We then extend the model with faults and determine whether the property continues to hold under reasonable fault scenarios.

The initial safety property to be proven determines whether the system will apply pressure to the wheels when commanded to do so:

```
If pedals are pressed and no skid occurs, then the brakes
will receive pressure.
```

Using the reference AADL model constructed by the SEI [9] extended with AGREE contracts describing system behaviors, this property proves immediately. From this point, we focus our attention on component failures and how this will affect the top level property of the system.

We would like to specify different component failure modes. These failure modes can be triggered by some internal or propagated fault. In order to trigger these faults, additional input was added to the AADL model for each fault that can occur within a nominal model component. This consists of two types:

- *fail_to* fault: This type of fault accounts for both nondeterministic failures and stuck-at failures. The components that are affected by this fault include meter valves and pumps. This fault can be used to describe both digital and mechanical errors. Examples of digital failures include a *stuck_at* failure for the command subsystem in the BSCU component, which causes the command unit to become stuck at a previous value. An example of a mechanical failure would be a valve stuck open (or closed).
- *inverted_fail* fault: This type of fault will be used on components which contain boolean output. It will simply take boolean input, negate it, and output the negated value. An example of this is the selector. In the nominal model, input to the selector consists of a boolean value *select_alternate* value from the BSCU.

These faults can be easily encoded in AGREE as shown in Figure 4. The failures simply return an alternate value (for *fail_to*) or invert the input value (for *inverted_failure*) when a failure occurs.

While modeling faults, the duration of the fault must also be taken into account. The AGREE tools allow a great deal of flexibility in terms of how faults are defined and their duration. For the purposes of this model, we currently consider only *transient* and *permanent* faults, where transient faults occur for an instant in time (e.g., a single-event upset) and a permanent fault persists for the remainder of the system execution.

4.1 Analysis of Faulty Models

The following is a short summary of the failures defined in the fault model.

```
node fail_to(val_in: real, alt_val: real, fail_occurred: bool) returns (val_out: real);
let
   val_out = if (fail_occurred) then alt_val else val_in;
tel;


node inverted_fail(val_in: bool, fail_occurred: bool) returns (val_out:bool);
let
  val_out = if fail_occurred then not(val_in) else val_in;
```

Fig. 4. AGREE Definition of a *fail_to* and *inverted_failure* Faults

– Valves and Pumps: All valves and pumps have the possibility of a *fail_to* fault. This includes green pump, blue pump, accumulator, and the shutoff valves.
– The selector can also have a digital *fail_to* fault regarding the inputs from BSCU commanding to use normal or alternate means of pressure along with an *inverted_fail* fault which would change the boolean value that commands antiskid to activate.

Given our understanding of the WBS, our assumption was that any single permanent fault could be introduced into the system and the pilot would still be able to command brake pressure. However, our analysis tools returned a counterexample to the property, and upon examination, the structure of the reference model was insufficient to guarantee the property.

The first issue was *feedback*; the reference model did not have a sensor to determine pressure after the selector valve. This means that a single failure of (for example) the blue or green antiskid valve cannot be detected by the BSCU (see Figure 1), and it cannot route around the failure. In order to address this, we added a pressure sensor to the wheel that communicates with the BSCU to detect lack of pressure at the wheel.

After adding a sensing apparatus to the wheel, the analysis generated another counterexample due to a single failure of the selector valve. In the reference model, there is a single selector component that takes as inputs the green pump, the blue pump, and the accumulator. A single failure in this component can lead to no pressure along either of the two outgoing pressure lines. To solve this issue, we removed the accumulator from the selector and added an accumulator valve. This component takes in the blue pressure from the selector and the accumulator pressure. It also takes in a *select_alternate* flag from the BSCU. The output of the accumulator_valve goes directly to the blue_skid component and is either the blue or the accumulator pressure.

Finally, our BSCU is currently structured to always fail-over from the green system to the blue system but never the reverse. Because of this choice (which matches the AIR6110 document), it is also necessary to guarantee that *select_alternate* is false until a failure occurs in the system; otherwise, a single failure in the blue anti-skid valve can cause the system to fail to provide pressure. This asymmetry is something that could be revisited in future work.

Even after making these three changes to the model, the original property still does not prove. At issue is that the sensing of a no-pressure situation is not instantaneous; there is a delay for this information to reach the BSCU and be acted upon to switch to the alternate braking system. In our current timing model for the system, the feedback to the BSCU involves a delay, but the BSCU and valves can react. Thus, we weaken our top-level property to state that if the brakes are pressed for two consecutive time instants, then pressure will be provided to the wheels:

```
If pedals are pressed in the previous state and pressed
in the current state and no skid occurs, then the brakes
will receive pressure.
```

5 Discussion

We have used the WBS model as a vehicle to experiment with different modeling and fault representation ideas, and to get a feel for the scalability of our approach. We started from the reference AADL model [9] to attempt to contrast our FEM approach using AGREE contracts vs. the FLM-based approach that was already part of this model. Part of this was driven by curiosity as to whether important faults might be caught by one approach and missed by the other, and to contrast the two styles of analysis.

During the process of defining and injecting faults, subtle issues of the system structure and behavioral interactions became much clearer. The idea that the system must use the green side until a failure occurs was unexpected. In addition, the extensions to the model were driven by the counterexamples returned by the tools. The approach quickly and precisely provided feedback towards aspects of the system that were not robust to failure. The researcher who produced the model (Danielle) was not involved in earlier MBSA work and had no prior exposure to the WBS model and yet was able to relatively quickly construct a fault-tolerant model. The fact that these holes in the reference model perhaps means that the behavioral approach can be better at drawing attention to certain kinds of failures.

On the other hand, the utility of the safety analysis is driven by the "goodness" of the properties. Our one example property is clearly insufficient: for example, it is not possible to detect faults related to over-pressurization or misapplication of the brakes when no braking is commanded. Of course, any complete analysis should have properties related to each hazardous condition. The approach is foundationally a top-down analysis (like fault trees) rather than a bottom up approach (like a FMEA / FMECA). In addition, if properties are mis-specified, or the system dynamics are incorrectly modeled, then properties may verify even when systems are unsafe. The explicit propagation approach of the FLM techniques force the analyst to consider each fault interaction. This too is a double-edged sword: when examining some of the fault propagations in the reference model, we disagreed with some of the choices made, particularly with respect to the selector valve. For example, if no select alternate commands are received from the BSCU, then both the green and blue lines emit a *No_Service* failure.

In terms of scalability, the analysis time for counterexamples was on the order of 1-2 seconds, and the time for proofs was around 4 seconds, even after annotating the model with several different failures. Thus, we feel that the analysis is likely to scale well to reasonably large models with many component failures.

The analysis in this paper involved hand-annotating the models with failure nodes. This process is both schematic and straightforward: we define the AGREE contracts over internal *nominal output variables* and then define the actual outputs using the nominal output variables as inputs to the fault nodes like those in Figure 4. We are currently in the process of defining a fault integration language. Some aspects of the Error Annex could be directly relevant: the state machines describing leaf-level faults could easily be compiled into behavioral state machines that determine when faults occur. On the other hand, in a behavioral approach we need to be able to bring in additional quantities (inputs, parameters) to instantiate behavioral faults, and the two approaches have very different notions of propagation.

The xSAP tool [4] has an elegant extension language that allows for fault definition, selection between multiple faults for a component, and "global" dependent faults that can affect multiple components. The authors have used this support to construct a sophisticated analysis model for the WBS [5]. However, some useful aspects of fault modeling, such as global faults that are driven by the state of the model, appear to be hard to construct. For example, a pipe-burst failure can be seen as a global failure because it may cause unconnected components within the model to fail. On the other hand, the likelihood of failure in the real system is driven by the number of currently pressurized pipes in the system, which appears to be hard to define. We hope to allow for such conditional and model-driven failures in our fault definition language.

6 Conclusions & Future Work

In this paper, we describe our initial work towards performing MBSA using the AADL architecture description language using a failure effect modeling approach. Our goal is to be able to perform safety analysis on common models used by systems and safety engineers for functional and non-functional analyses, schedulability, and perhaps system image generation. To perform this analysis, we use existing capabilities within AADL to describe the structure of the system, and build on the existing AGREE framework for compositional analysis of components.

As part of our exploration, we are interested in examining the strengths and weaknesses of our FEM and the AADL Error Annex FLM-based approach. We believe that the FEM approach has advantages both in terms of brevity of specifications and accuracy of results, and can build on existing analyses performed for systems engineering. However, there are also risks in the FEM approach involving incomplete or misspecified properties.

We illustrated the ideas using architecture models based on the Wheel Braking System model in SAE AIR 6110 [2] and use this in the evaluation of our approach. Using assume-guarantee compositional reasoning techniques, we prove a top level property of the wheel brake system that states when the brake pedals are pressed in the absence of skidding, there will be hydraulic pressure supplied to the brakes.

Starting from the error model notions of error types, two main faults were defined: *fail_to* which will describe failures of valves and pressure regulators and *inverted_fail* which describes the failures occurring to components that output boolean values. Using the AADL behavioral model of the WBS, these permanent faults were tied into the nominal model in order to reason about how this model behaves in the presence of specific kinds of faults.

In order to demonstrate that the system was resilient to single faults, we modified the model to allow feedback from the wheel pressure to the BSCU. This changed the way the system responded to faults that were further downstream of the BSCU or Selector and created a chance for the system to switch to alternate forms of hydraulic pressure. We also reasoned about the initialization values of the system in regards to which mode is the starting mode. It is crucial for the system to begin in Normal mode in order to function successfully in the presence of faults. After model modification and a small weakening of our original property to account for feedback delay, the model does fulfill the top level contract even when a permanent fault of one of the high level components is introduced.

The current capabilities of AGREE are well-suited to specifying faults. Our approach allows for scalar types of unbounded integers and reals, as well as composite types such as tuples and structures. It is possible to model systems and reason about them in either discrete time or real-time. However, adding faults to existing components is cumbersome and can obscure the nominal behaviors of the model. We are currently examining several fault specification languages, giving special consideration to the xSAP modeling language.

Future research work will involve the continuation of development of the methods and tools needed to perform model-based safety analysis at the system architecture level. By introducing a common set of models for both nominal system design and safety analysis, we hope to reduce the cost of development and improve safety. Our hope is to demonstrate the practicality of formal analysis for early detection of safety issues that would be prohibitively expensive to find through testing and inspection. We will base this research on industry standard notations that are being used in airborne and ground-based avionics in order to ensure transition of this technology.

Acknowledgements

This research was funded by NASA AMASE NNL16AB07T.

References

1. AADL. Predictable Model-Based Engineering.
2. AIR 6110. Contiguous Aircraft/System Development Process Example, Dec. 2011.
3. J. Backes, D. Cofer, S. Miller, and M. W. Whalen. Requirements Analysis of a Quad-Redundant Flight Control System. In K. Havelund, G. Holzmann, and R. Joshi, editors, *NASA Formal Methods*, volume 9058 of *Lecture Notes in Computer Science*, pages 82–96. Springer International Publishing, 2015.

4. B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli, and G. Zampedri. The xSAP Safety Analysis Platform. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 533–539, 2016.
5. M. Bozzano, A. Cimatti, A. F. Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta. Formal Design and Safety Analysis of AIR6110 Wheel Brake System. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 518–535, 2015.
6. D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lnn, and Y. Papadopoulos. Systems Modeling with EAST-ADL for Fault Tree Analysis through HiP-HOPS*. *IFAC Proceedings Volumes*, 46(22):91 – 96, 2013.
7. A. Cimatti and S. Tonetta. Contracts-Refinement Proof System for Component-Based Embedded System, journal = Sci. Comput. Program., volume = 97, pages = 333–348, year = 2015, url = http://dx.doi.org/10.1016/j.scico.2014.06.011, doi = 10.1016/j.scico.2014.06.011, timestamp = Fri, 30 Jan 2015 18:56:15 +0100, biburl = http://dblp.uni-trier.de/rec/bib/journals/scp/CimattiT15, bibsource = dblp computer science bibliography, http://dblp.org.
8. D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In A. E. Goodloe and S. Person, editors, *Proceedings of the 4th NASA Formal Methods Symposium (NFM 2012)*, volume 7226, pages 126–140, Berlin, Heidelberg, April 2012. Springer-Verlag.
9. J. Delange, P. Feiler, D. P. Gluch, and J. Hudak. AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment. Technical Report CMU/SEI-2014-TR-020, Software Engineering Institute: Carnegie Mellon University, October 2014.
10. S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML.* Morgan Kaufman Pub, 2008.
11. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language Lustre. In *In Proceedings of the IEEE*, volume 79(9), pages 1305–1320, 1991.
12. P. Hnig, R. Lunde, and F. Holzapfel. Model Based Safety Analysis with smartIflow . *Information*, 8(1), 2017.
13. A. Joshi and M. P. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In *SAFECOMP*, volume 3688 of *LNCS*, page 122, 2005.
14. A. Joshi and M. P. Heimdahl. Behavioral Fault Modeling for Model-based Safety Analysis. In *Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium (HASE)*, 2007.
15. A. Joshi, S. P. Miller, M. Whalen, and M. P. Heimdahl. A Proposal for Model-Based Safety Analysis. In *In Proceedings of 24th Digital Avionics Systems Conference (Awarded Best Paper of Track)*, 2005.
16. A. Joshi, M. Whalen, and M. P. Heimdahl. Automated Safety Analysis Draft Final Report. Report for NASA Contract NCC-01001, August 2005.
17. B. Larson, J. Hatcliff, K. Fowler, and J. Delange. Illustrating the AADL Error Modeling Annex (V.2) Using a Simple Safety-critical Medical Device. In *Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '13, pages 65–84, New York, NY, USA, 2013. ACM.
18. O. Lisagor, T. Kelly, and R. Niu. Model-based safety assessment: Review of the discipline and its challenges. In *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, pages 625–632, June 2011.
19. MathWorks. The MathWorks Inc. Simulink Product Web Site. http://www.mathworks.com/products/simulink, 2004.

20. A. Murugesan, M. W. Whalen, S. Rayadurgam, and M. P. Heimdahl. Compositional Verification of a Medical Device System. In *ACM Int'l Conf. on High Integrity Language Technology (HILT) 2013*. ACM, November 2013.
21. M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee. Model-Driven Safety Analysis of Closed-Loop Medical Systems. *Industrial Informatics, IEEE Transactions on*, PP:1–12, 2012. In early online access.
22. T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, and A. Rauzy. The AltaRica 3.0 Project for Model-Based Safety Assessment. *IFAC Proceedings Volumes*, 46(22):127 – 132, 2013.
23. SAE ARP 4761. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.
24. SAE AS 5506B-3. Aadl annex volume 1, Sept. 2015.
25. O. Sokolsky, I. Lee, and D. Clarke. Process-Algebraic Interpretation of AADL Models. In *Reliable Software Technologies - Ada-Europe 2009, 14th Ada-Europe International Conference, Brest, France, June 8-12, 2009. Proceedings*, pages 222–236, 2009.