¡¡¡¡¡¡¡ HEAD

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
200 Union Street
Minneapolis, MN, 55455, USA
`whalen, dkstewar, heimdahl@cs.umn.edu`
[2] Rockwell Collins
Advanced Technology Center
400 Collins Rd. NE
Cedar Rapids, IA, 52498, USA
`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. Model-based systems engineering (MBSE) methods and tools Add citations: us, Trento folks, UPenn folks 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 [] and the Architecture Analysis and Design Language (AADL) [] 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 [] and HiP-HOPS for EAST-ADL []. 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, Rockwell Collins and the University of Minnesota developed and demonstrated an approach to model-based safety analysis (MBSA)Add citations using the Simulink notation Add Simulink citation. 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 benchmark our work, we have developed architectural models for the Wheel Braking System model in SAE AIR6110 to evaluate the effectiveness and practicality of our approach. Starting from a reference AADL model constructed by the SEI instrumented with qualitative safety analysis information [], we add behavioral contracts to the model. In so doing, we determine that there are errors related to (manually constructed) propagations across components, and also an architectural decomposition that allows for single points of failure. We use our analyses to find and fix these errors.

The rest of the paper is organized as follows...

2 Safety Assessment Process

COMPLETELY STOLEN FROM THE SAFECOMP-05 PAPER! Either note it or modify it I added a citation. I think this late in the game we might as well just cite it.

The overall safety assessment process that is followed in practice in the avionics industry is described in the SAE standard ARP 4761 [3]. Our summary in this section is largely adopted from ARP 4761.

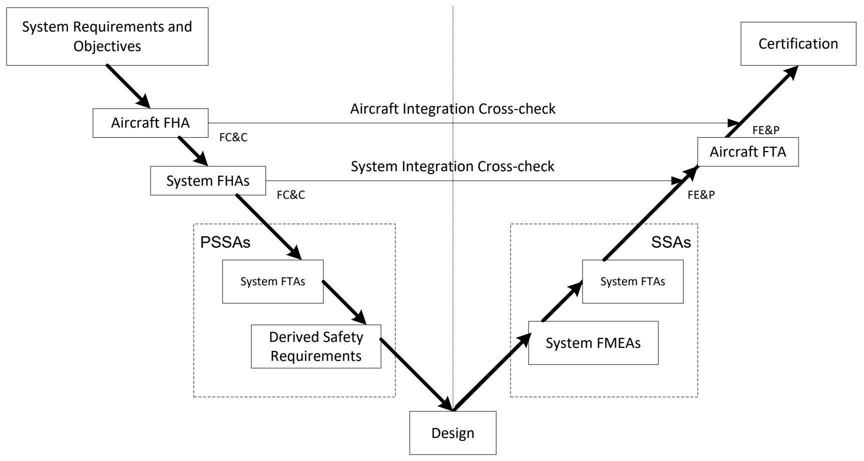

System Requirements and Objectives
Certification
Aircraft Integration Cross-check
Aircraft FHA
FC&C
FE&P
Aircraft FTA
System Integration Cross-check
System FHAs
FC&C
FE&P
PSSAs
SSAs
System FTAs
System FTAs
Derived Safety Requirements
System FMEAs
Design

Fig. 1. Traditional “V” Safety Assessment Process

Figure 1 from [10] shows an overview of the safety assessment process as recommended in ARP 4761. The process includes safety requirements identification (the left side of the “V” diagram) and verification (the right side of the “V” diagram), that support the aircraft development activities. An aircraft level Functional Hazard Analysis (FHA) is conducted at the beginning of the aircraft development cycle, which is then followed by system level FHA for individual sub-systems. The FHA is followed by Preliminary System Safety Assessment (PSSA), which derives safety requirements for the subsystems, primarily using Fault Tree Analysis (FTA). The PSSA process iterates with the design evolution, with design changes necessitating changes to the derived system requirements (and also to the fault trees) and potential safety problems identified through the PSSA leading to design changes. Once design and implementation are completed, the System Safety Assessment (SSA) process verifies whether the safety requirements are met in the implemented design. The system Failure Modes and Effects Analysis (FMEA) is performed to compute the actual failure probabilities on the items.

The verification is then completed through quantitative and qualitative analysis of the fault trees created for the implemented design, first for the subsystems and then for the integrated aircraft.

We propose to modify this traditional "V" process so that the lower level PSSA and SSA activities are performed based on a formal model of the system under consideration. Figure 2 shows the modified "V" diagram for model-based safety analysis. The shaded blocks are those activities that will be modified or added.

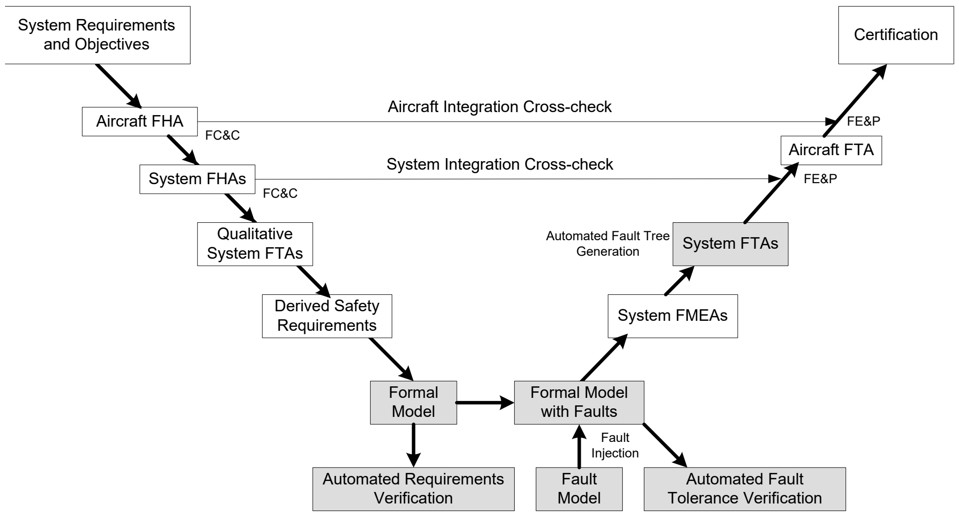

System Requirements and Objectives
Certification
Aircraft FHA
Aircraft Integration Cross-check
FE&P
FC&C
Aircraft FTA
System FHAs
System Integration Cross-check
FC&C
FE&P
Qualitative System FTAs
Automated Fault Tree Generation
System FTAs
Derived Safety Requirements
System FMEAs
Formal Model
Formal Model with Faults
Fault Injection
Automated Requirements Verification
Fault Model
Automated Fault Tolerance Verification

Fig. 2. Modified "V" Safety Assessment Process

As we can observe from Figure 2, the parts of the analysis that are primarily affected are at the bottom of the "V". The biggest difference is that the safety analysis activities at this level are now focused around a formal model of the system behavior, and that many of the artifacts of the safety analysis can be derived from this model. The idea is to try to pose the right verification questions to formal tools (such as model checkers and theorem provers) so that it is possible to derive the necessary safety analysis information. We then wish to turn the results of these analyses back into artifacts that can be easily understood and used by safety engineers.

3 Model-Based Safety Analysis

Audience should already know about model-based safety analysis. We want to discuss two "strands" of MBSA, one with explicit fault propagation of faults rather than system dynamics, and another based on faults propagating through their effects on dynamics.

A model-based approach for safety analysis was first proposed by Joshi et. al in [10–12]. 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. Figure include figure? provides an overview of this process applied to the wheel braking system example.

The contents and structure of the SASM differ significantly across different conceptions of MBSA. We can draw broad outlines between two different approaches: one in which *faults* are propagated between components explicitly and the analysis proceeds by determining the likelihood of faults to system boundaries, and another in which faults propagate by changing the system dynamics, which may cause the system behavior to visibly change. We call the first style the *explicit fault propagation* and the second style *implicit fault propagation* or *behavioral* propagation.

We contrast these two styles below...

Regurgitate content from ”Model-Based Safety Analysis with smartIflow”

Some contrasting points: richness of dynamics. Most tools use formalisms that can only represent discrete quantities; we can do real-valued or large-domain integer dynamics

Also we are explicitly integrated with an architecture language

3.1 Explicit Fault Propagation Approaches

What is being done in AltaRica, HiPHOPS, AADL Error Annex...more here about primacy of faults as the mechanism of propagation between components. Steal from proposal.

The explicit propagation approach focuses on faults rather than constructing a model of system dynamics.

We illustrate this approach with the AADL error model annex [15] 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 errors in the system. 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.

An example is shown in Figure Where is this figure?!?. Errors are described by the red rectangles labeled with error types: 1-BadData, 2-NoData, 3-NoSvc. Error 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 red 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 [3], 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. The user must therefore model the system twice, specifying propagations in the models of the physical phenomena, and again using enumerations and propagation rules in state machines in the error model annex. 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 phenomenon and the behavior of the error model.

3.2 Implicit Fault Propagation Approaches (Behavioral MBSA)

Unfortunately, this is what is done, and done well in xSAP; they have a CAV paper on it; need to account for this.

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 like *nondeterministic*, *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 work, a behavior fault modeling language was constructed as an extension to the Lustre language [9]. Lustre is the kernel language of the popular model-based development tool SCADE [7]. 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 [11]. 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 as shown in Figure include figure?. Modeling allows trivial exploration of *what-if* scenarios involving combinations of faults through simulations. For more rigorous analyses, static analysis tools, such as model checkers and theorem provers, can be used to automatically prove (or disprove) whether the system meets specific safety requirements. The primary approach used for analysis in previous work was model checking. After creating the system model, a model checker was used to verify 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. In the approach described by Joshi et al. in [10–12], this analysis was performed by determining whether the property held for a given fault threshold: the maximum number of component faults to which the system is expected to be resilient.

This fault threshold is, in some sense, an approximation of the likelihood of component faults. It maps from the probabilistic *real world* potential for component failure into a non-probabilistic verification problem. Recent work [5] uses a more sophisticated approach involving minimum cut sets to describe the set of potential component failures that must be considered. Both approaches provide separation between the probabilistic aspects of the real world and the computational demands of formal analysis. This approach currently scales far better than direct use of probabilistic model checking tools such as PRISM [13], and will likely continue to do so in the future.

4 New MBSA Capabilities

REWRITE!

Previous research has been based on MBD tools (such as Simulink and SCADE) that were available at the time [12]. However, these tools are really targeted at the design and implementation of software components, rather than at the system architecture level where most safety concerns arise.

Within the past five years there have been great advances in the capabilities of tools for modeling and analysis of at the system level, based on languages such as SysML [8] and AADL [2]. We use these new MBSE capabilities and extend them to implement the safety analysis methods needed for the design and certification of commercial aircraft systems. The system modeling tools that we plan to use are based on AADL, but they can import and export models from SysML.

- We will improve the efficiency of MBSA methods by using MBSE tools that can perform compositional reasoning over complex system models. Using the assume-guarantee contract mechanism, these tools provide support for heterogeneous component models implemented in different languages (such as Simulink or C/C++).
- Our new analysis methods move away from traditional static safety analysis methods focused on probabilistic models (e.g., Fault Tree Analysis), to the direct modeling of potential failure mechanisms and the analysis of dynamic fault-mitigation strategies.
- Formal verification of system models provides increased assurance that these models are accurate and will produce correct results. The hierarchical structure of system architecture models supports analysis at varying levels of abstraction. Com-

positional analysis explicitly checks assumptions captured in component and subsystem contracts. Consistency and realizability checks [23] provide the ability to detect conflicting requirements between component and subsystem models.

We bridge the descriptions of errors in the error model annex with behavioral descriptions of components. We start from the error model notions of error types and state machines that describe transitions from nominal to error states. However, we then tie these nominal and error states to behavioral models of the components in question that describe how the faults manifest themselves in terms of the signals or quantities produced by the components. Now the behavioral models can provide implicit propagation of the faulty behaviors and the natural consequences of failures on component behavior will be manifested in the propagation of other component faults through the behavioral model.

To accomplish this, we use AADL and the error model annex to describe faults, and to use the AGREE contract specification language to describe behavioral models. This requires extensions to AGREE to define fault models that describe how different faults manifest themselves in changes to output signals. It also requires changes to the error annex. The conditions under which faults occur will become richer such that they describe not just propagation of enumerations from other components, but also valuations of input signals.

An example is shown in Figure 3. Errors are described by the red rectangles labeled with error types: 1-BadData, 2-NoData, 3-NoSvc. Error 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 3 have two states, they can be much more complex. The red 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.

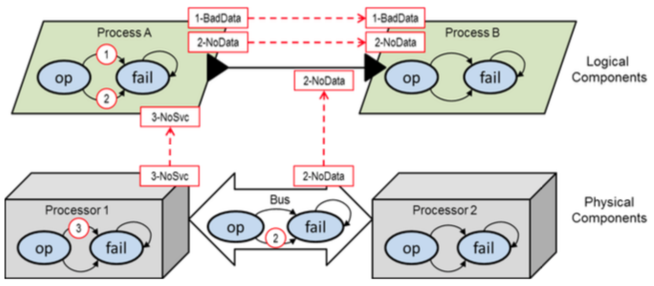


Fig. 3. Example of Error Model Information and Propagation

5 ARP4761: Wheel Brake System

Move the generic description to a section right after the "safety" section along with the ARP figure. Leave the modeling that we did in AADL here. That way we can reference it in the MBSA sections.

As a preliminary case study, we utilized the Wheel Brake System described in ARP 4761 - Appendix L [3]. The informal wheel brake system diagram is shown in Figure **??** and is taken from the ARP 4761 document. The Wheel Brake System is installed on the two main landing gears and is used during taxi, landing, and rejected take off. Braking is either commanded manually using brake pedals or automatically 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 is operated by two sets of hydraulic pistons which operate independently. One set is operated by the Green hydraulic line which is used in Normal braking mode. The second system (Alternate) is operated by the Blue hydraulic line which is used when the Normal system fails. The Alternate system is also supplied by an Accumulator which is a device with built up pressure that can be released if both of the two primary pumps (Blue and Green) fail. The accumulator supplies the Alternate system in case of Emergency braking mode.

Switching between the hydraulic pistons and sources can be done automatically or manually. If the Normal pressure (Green) is below a certain threshold, there is an automatic switchover to the Blue supply. If the Blue pump fails, then the Accumulator is used for 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 braking 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.

5.1 Nominal System Modeling

A formal specification of the nominal system model consists of mechanical and digital components and their interconnections. Figure 4 illustrates how the wheel braking system is modeled using Architecture Analysis and Design Language (AADL). The following section describes this nominal model from which the fault model was generated.

Wheel Braking System (WBS) The highest level component is the WBS. It consists of a digital control unit, the BSCU, and Normal and Alternate hydraulic pressure lines (supplied by 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.

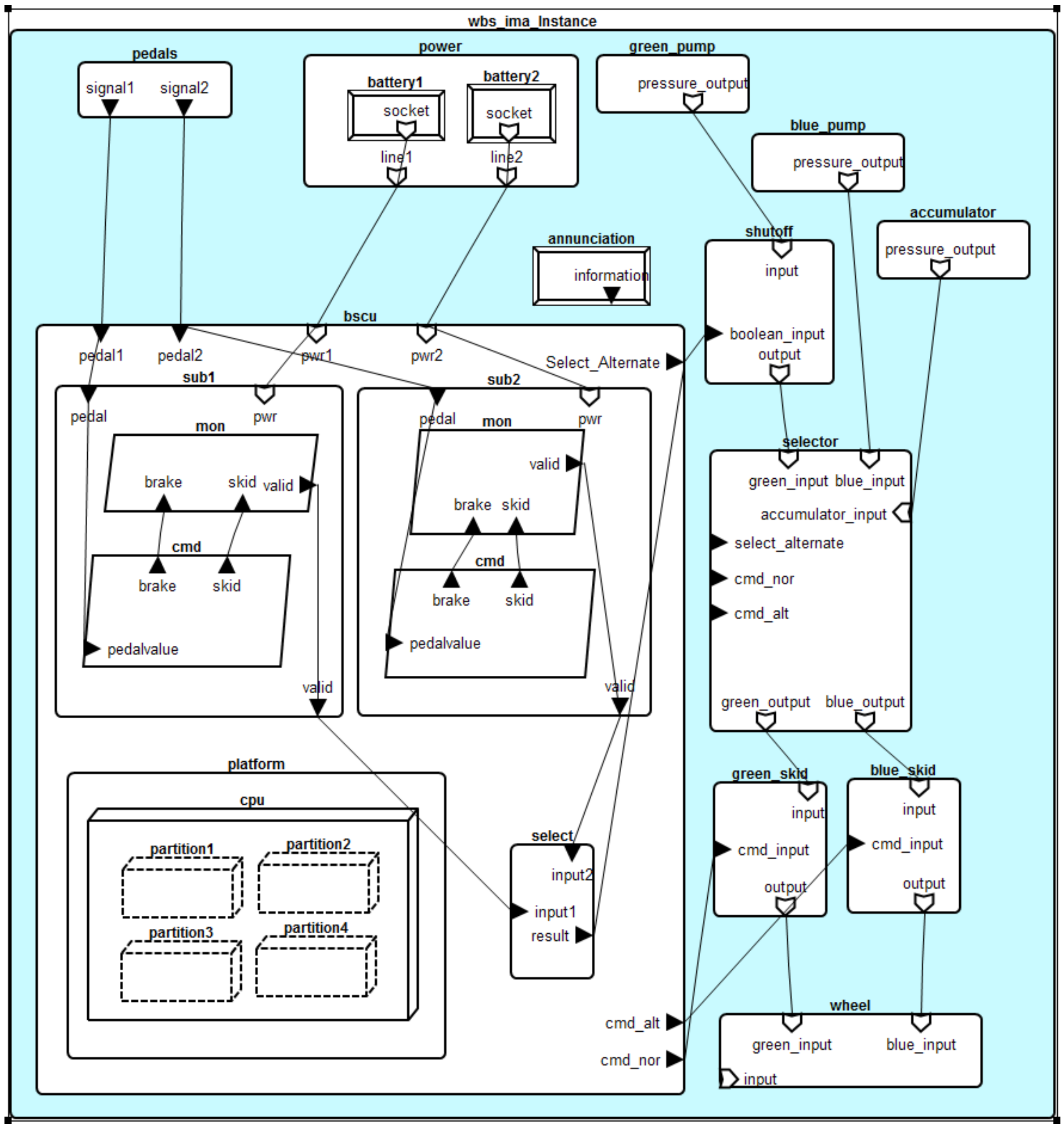


Fig. 4. AADL Simple Model of the Wheel Brake System

NOTE: selector::select_alternate is not connected (we fixed this in our model) maybe we could add the modified portion of the model into the figure

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 Normal and Alternate 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 maps into these units and when skidding occurs, the Command and Monitor units will decrease the pressure to the brakes.

The Command unit regulates the pressure to the brakes in the Normal 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 Alternate hydraulic line in order to prevent skidding which it does through the cmd_alt command. The Monitor unit monitors the validity of the Command unit output.

The BSCU switches to the Alternate hydraulic system (cmd_alt = true) under the following conditions:

– Output from both Command units are not valid
– The Green pump (Normal) is below threshold

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 a meter valve 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. It is unclear how the Selector valve operates if both incoming pipes have pressure greater than the threshold. The AADL model assumes that the default in this case is Normal mode.

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 off to the wheel.

6 Fault Modeling

In order to start from the error model notions of error types and look more closely at the state machines that describe transitions from the nominal state to the error state, we focus on the types of errors that can be present in the components of the system.

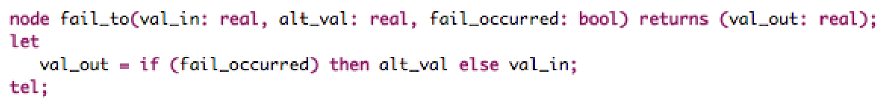
To begin with, we proved a top level property of the WBS using AGREE. This property states:

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

Using the AADL nominal model, this property proves. 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 error 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. Our simple fault model contains two types of faults:

- *fail_to* fault: This type of failure 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 modeled in such a way as to describe both digital and mechanical errors. Examples of digital errors include a *stuck_at* failure mode for the Command subsystems in the BSCU component. This causes the Command units to become stuck at a previous value. An example of a mechanical error that is captured by this fault is a valve stuck open (or closed). The definition in AGREE of this fault is shown in Figure 5.

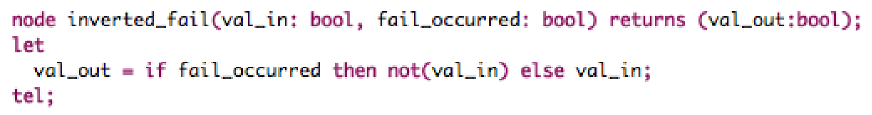


```
node fail_to(val_in: real, alt_val: real, fail_occurred: bool) returns (val_out: real);
let
   val_out = if (fail_occurred) then alt_val else val_in;
tel;
```

Fig. 5. AGREE Definition of a *fail_to* Fault

- *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 examples of this is the Selector. See Figure 6

While modeling these errors, the duration of the fault must also be taken into account. There are both permanent faults and transient faults. We defined an AGREE node



```
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;
tel;
```

Fig. 6. AGREE Definition of a *inverted_fail* Fault

called `historically` which will model the permanent faults based on whether transient faults are historically true.

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

6.1 AADL 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 AADL system, the assumption was that permanent faults could be introduced into the system and we would still be able to prove our top level property. Upon connecting the faults into the WBS, it was shown that this top level property could not be proven using these permanent faults. Upon further reflection, it was clear why that was the case.

In the AADL model, the Selector is situated between the pumps and the Blue_Skid and Green_Skid components. It's function is as follows: inputs are pump pressures (for Blue, Green, and Accumulator) and information from the BSCU regarding Normal or Alternate pressure. It selects between the Blue and the Green pump in this way. The output of the Selector is pressure of the Blue and Green pumps to the Blue_Skid and Green_Skid components respectively. The Blue_Skid and Green_Skid components also get a flag from BSCU. Blue_Skid receives a flag with cmd_alt while Green_Skid gets the command cmd_nor. This will cause those components to either output pressure (if their respective BSCU command is true) or output zero pressure (if the BSCU command is false). The pressure output from here goes directly to the wheel. For a clear picture of this organization, see Figure 4.

Here is a case when the fault is *fail_to* and it is a failure of the Green_Skid component. Previously in the system, the Selector received a command from BSCU to output Normal pressure (Green pump). This command was passed on to the Green_Skid component along with positive green pressure. If the Green_Skid component gets stuck at a value of zero for pressure, the output to the wheel is zero. The Blue_Skid component gets no pressure from the Selector (it has already selected green pressure) and thus outputs no pressure to the wheel.

Despite one transient failure, the tail end of the system is not designed to recover from this failure. A similar single point of failure is the Blue_Skid component and the Selector. These faults cause system failure for the same reason.

6.2 Strengthening the Nominal Model

To solve this issue, we added a component based on the Simulink model in [12]. This component, the Accumulator_Valve, 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.

An additional piece we added was output to the BSCU from the wheel. The pressure at the wheel is sent to the BSCU to readjust system mode if needed. For instance, if the pressure at the wheel is zero from both blue and green skid components, then the BSCU has a chance to send a *select_alternate* command over to the Accumulator_Valve and turn on the accumulator pressure. We assume at this point that we will only use accumulator pressure and not switch back over to Normal mode.

In order for this to solve the problem, the original top level contract needed to be revisited:

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

As it is stated, the added components will not solve the problem. There must be stateful information encoded into this contract. Thus it is changed to:

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

It was also necessary to guarantee that *select_alternate* is false until an error occurs in the system. This was written as an AGREE guarantee stating that the initialization of *select_alternate* is false. It remains false until the system commands brake pressure (i.e. brake pedals are pressed), there is no skid occurring, and there is no realized pressure at the wheel. At this point, the system will enter Alternate mode and command pressure from the blue (or accumulator) pump. It will not return to Normal mode.

7 Discussion

Some of the goals of extending existing modeling languages in this research is to better describe failure conditions and to better understand interactions of system components in the face of faults. In the process of defining and injecting faults, some subtle issues of the system became far more clear. One of these was the necessity of a non-symmetric system in terms of the Alternate and the Normal hydraulic pressure. As the system is described, the Alternate system requires a separate functionality by means of the Selector and Accumulator valve. Without this separation between the Normal and Alternate systems, one permanent fault will cause failure in the system (namely in either of the skid components). Another failure condition that wasn't initially obvious was the initialization of the system to be in Normal mode. If the initial state of the system is Alternate mode, it cannot recover from a fault in the blue_skid component or the accumulator_valve component.

These failure conditions did not appear obvious by studying the organization of components or proving the nominal system requirements, but instead became clear when utilizing some of these compositional analysis techniques.

Another goal mentioned above was to better understand the interactions of system components in the face of faults. This understanding was improved during the process of our initial permanent fault introduction to the system. When a fault occurred in the skid components whose output goes directly to the wheel, the system had no time to select otherwise. The interaction between the BSCU and skid components (or wheel) was nonexistent. In order to rectify this lack of communication, further interaction was required between the wheel and BSCU components. By allowing the system one more step of computation, the system is able to respond to these later errors and correct accordingly. This realization came while observing the interaction between components in the face of faults.

Once these issues were addressed, we could successfully prove the top level contract in the case of one permanent fault in any of the high level components. When any one of these faults are present, the system can successfully respond to the fault and still provide pressure to the brakes.

8 Conclusions & Future Work

In this paper, we describe methods of system safety analysis techniques and some issues in capturing system safety information. We describe our initial work in extending modeling notations to better describe and analyze failure conditions in order to meet the specific needs of system safety analysis.

We begin with architectural models based on the Wheel Braking System model in SAE AIR6110 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. This is the nominal model. We then focus our attention on fault generation.

Starting from the error model notions of error types, two main faults were defined: *fail_to* which will describe errors of valves and pressure regulators and *inverted_fail* which describes the faults occurring to components that output boolean values. Using the AADL behavioral model of the WBS, these permanent errors 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 reason about the system, we extended the model to account for a lack of communication between components. 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.

What was seen is that the model does fulfill the top level contract even when a permanent fault of one of the high level components is introduced.

Future research work involves 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, this will not only reduce the cost of development but also improve safety by integration of these processes. 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.

References

1. Ieee standard for property specification language (psl), 2005.
2. AADL. Predictable model-based engineering. http://www.aadl.info.
3. AIR 6110. Contiguous aircraft/system development process example, Dec. 2011.
4. 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.
5. M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei. Efficient anytime techniques for model-based safety analysis. In *Computer Aided Verification (CAV '11*, 2015.
6. 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.
7. Esterel Technologies. Scade suite product description. http://www.estereltechnologies.com.
8. S. Friedenthal, A. Moore, and R. Steiner. *A practical Guide to SysML*. Morgan Kaufman Pub, 2008.
9. 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.
10. 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.
11. 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.
12. 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.
13. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *In Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of LNCS, 2011.
14. MathWorks. The mathworks inc. simulink product web site. http://www.mathworks.com/products/simulink, 2004.
15. SAE AS 5506B-3. Aadl annex volume 1, Sept. 2015.