Safety Annex for the Architecture Analysis and Design Language

Danielle Stewart*, Jing (Janet) Liu†, Darren Cofer†, Mats Heimdahl*, Michael W. Whalen*, Michael Peterson†
*University of Minnesota, Department of Computer Science
{dkstewar, heimdahl, mwwhalen}@umn.edu
†Collins Aerospace
{jing.liu, darren.cofer, michael.peterson}@collins.com

***Abstract*—Model-based development tools are increasingly being used for system-level development of safety-critical systems. Architectural and behavioral models provide important information that can be leveraged to improve the system safety analysis process. Model-based design artifacts produced in early stage development activities can be used to perform system safety analysis, reducing costs, and providing accurate results throughout the system life-cycle. In this paper we describe an extension to the Architecture Analysis and Design Language (AADL) that supports modeling of system behavior under failure conditions. This *Safety Annex* enables the independent modeling of component failures and allows safety engineers to weave various types of fault behavior into the nominal system model. The accompanying tool support uses model checking to propagate errors from their source to their effect on top-level safety properties without the need to add separate propagation specifications. Our tools are also able to compute minimal cutsets for these errors to produce faults trees familiar to safety engineers and certification authorities. We describe the Safety Annex, illustrate its use with a representative example, and discuss and demonstrate the tool support enabling an analyst to investigate the system behavior under failure conditions.**

I. Introduction

System safety analysis is crucial in the development life cycle of critical systems to ensure adequate safety as well as demonstrate compliance with applicable standards. A prerequisite for any safety analysis is a thorough understanding of the system architecture and the behavior of its components; safety engineers use this understanding to explore the system behavior to ensure safe operation, assess the effect of failures on the overall safety objectives, and construct the accompanying safety analysis artifacts. Developing adequate understanding, especially for software components, is a difficult and time consuming endeavor. Given the increase in model-based development in critical systems [10], [26], [29], [32], [35], leveraging the resultant models in the safety analysis process holds great promise in terms of analysis accuracy as well as efficiency.

In this paper we describe the *Safety Annex* for the system engineering language AADL (Architecture Analysis and Design Language), a SAE Standard modeling language for Model-Based Systems Engineering (MBSE) [2]. The Safety Annex allows an analyst to model the failure modes of components and then "weave" these failure modes together with the original models developed as part of MBSE. The safety analyst can then leverage the merged behavioral models to propagate errors through the system to investigate their effect on the safety requirements.

In previous work, we have extended AADL to include formal *assume-guarantee* contracts describing the behavior of systems and components [21]. These contracts describe the nominal behavior of the system, and can be used to formally verify system requirements through compositional reasoning applied to the each hierarchical layer in the AADL model. The Safety Annex builds on this approach by adding language constructs for specifying the possible behaviors of systems and components in the presence of faults. When a fault is triggered in the model, its nominal contract is replaced by the specified failure contract and the resulting impact on the system can be analyzed using a model checker. This focus on the behavior of the system in the presence of failures is fundamentally different from the capabilities provided by the existing AADL Error Model Annex [23], and enables powerful new approaches to system safety analysis.

Determining how errors propagate through software components is currently a costly and time-consuming element of the safety analysis process. The use of behavioral contracts to capture the error propagation characteristics of software component without the need to add separate propagation specifications (*implicit* error propagation) is a significant benefit for safety analysts. In addition, the annex allows modeling of dependent faults that are not captured through the behavioral models (*explicit* error propagation), for example, the effect of a single electrical failure on multiple software components or the effect hardware failure (e.g., an explosion) on multiple behaviorally unrelated components. Furthermore, we will describe the tool support enabling engineers to investigate the correctness of the nominal system behavior (where no failures have occurred) as well as the system's resilience to component failures. We illustrate the work with an example drawn from the civil aviation domain.

Our work can be viewed as a continuation of work conducted by Joshi et al. where they explored model-based safety analysis techniques defined over Simulink/Stateflow [36] models [15], [30]–[32]. Our current work extends this work to provide new modeling and analysis capabilities. For example, the Safety Annex supports compositional verification and exploration of the nominal system behavior as well as the sys-

tem's behavior under failure conditions. Related work includes the AADL Error Annex (EMV2) [23], COMPASS [11], and AltaRica [7], [39]. Our approach differs from AADL EMV2 in that we leverage the behavioral modeling for implicit error propagation. We provide compositional analysis capabilities that are not available in COMPASS. The Safety Annex is fully integrated in a model-based development process and environment unlike a stand alone language like AltaRica.

The main contributions of the AADL Safety Annex are:

- close integration of behavioral fault analysis into the *Architecture Analysis and Design Language* AADL, which allows close connection between system and safety analysis and system generation from the model,
- support for *behavioral specification of faults* and their *implicit propagation* (both symmetric and asymmetric) through behavioral relationships in the model, in contrast to existing AADL-based annexes (HiP-HOPS [18], EMV2 [23]) and other related toolsets (COMPASS [11], Cecilia [6], etc.),
- additional support to capture binding relationships between hardware and software and logical and physical communications,
- computation of all minimal fault combinations that can cause violation of the safety properties to be compared to qualitative and quantitative objectives as part of the safety assessment process, and
- guidance on integration into a traditional safety analysis process.

II. Preliminaries

One of our goals is to transition the tools we have developed into use by the safety engineers who perform safety assessment of avionics products. Therefore, we need to understand how the tools and the models will fit into the existing safety assessment and certification process.

A. Safety Assessment Process

ARP4754A, the Guidelines for Development of Civil Aircraft and Systems [42], provides guidance on applying development assurance at each hierarchical level throughout the development life cycle of highly-integrated/complex aircraft systems. It has been recognized by the Federal Aviation Administration (FAA) as an acceptable method to establish the assurance process. The safety assessment process is a starting point at each hierarchical level of the development life cycle and is tightly coupled with the system development and verification processes. It is used to show compliance with certification requirements and for meeting a company's internal safety standards.

ARP4761, the Guidelines and Methods for Conducting Safety Assessment Process on Civil Airborne Systems and Equipment [41], identifies a systematic means to show compliance. Among the industry accepted safety assessment processes are Preliminary System Safety Assessment (PSSA) and System Safety Assessment (SSA). PSSA evaluates the system design and defines safety requirements. SSA evaluates the implemented system to show that safety requirements defined in the PSSA are in fact satisfied.

AADL provides a unifying framework for describing the system architecture for "performance-critical, embedded, real-time systems" [2]. From its conception, AADL has been designed for the design and construction of avionics systems. Rather than being merely descriptive, AADL models can be made specific enough to support system-level code generation. Thus, results from analyses conducted, including the new safety analysis proposed here, correspond to the system that will be built from the model.

An AADL model describes a system in terms of a hierarchy of components and their interconnections, where each component can either represent a logical entity (e.g., application software functions, data) or a physical entity (e.g., buses, processors). An AADL model can be extended with language annexes to provide a richer set of modeling elements for various system design and analysis needs (e.g., performance-related characteristics, configuration settings, dynamic behaviors). The language definition is sufficiently rigorous to support formal analysis tools that allow for early phase error/fault detection.

The Assume Guarantee Reasoning Environment (AGREE) [21] is a tool for formal analysis of behaviors in AADL models. It is implemented as an AADL annex and annotates AADL components with formal behavioral contracts. Each component's contracts can include assumptions and guarantees about the component's inputs and outputs respectively, as well as predicates describing how the state of the component evolves over time. AGREE translates an AADL model and the behavioral contracts into Lustre [27] and then queries a user-selected model checker to conduct the back-end analysis. The analysis can be performed compositionally following the architecture hierarchy such that analysis at a higher level is based on the components at the next lower level. When compared to monolithic analysis (i.e., analysis of the flattened model composed of all components), the compositional approach allows the analysis to scale to much larger systems [21].

In our prior work [45], we added an initial failure effect modeling capability to the AADL/AGREE language and tool set. We are continuing this work so that our tools and methodology can be used to satisfy system safety objectives of ARP4754A and ARP4761.

B. Model-Based Safety Assessment Process Supported by Formal Methods

We propose a model-based safety assessment process backed by formal methods to help safety engineers with early detection of the design issues. This process uses a single unified model to support both system design and safety analysis. It is based on the following steps:

1) System engineers capture the critical information in a shared AADL/AGREE model: high-level hardware and software architecture, nominal behavior at the component level, and safety requirements at the system level.

2) System engineers use the backend model checker to verify that the safety requirements are satisfied by the nominal design model.
3) Safety engineers use the Safety Annex to augment the nominal model with the component failure modes. In addition, safety engineers specify the fault hypothesis for the analysis which corresponds to how many simultaneous faults the system must be able to tolerate.
4) Safety engineers use the backend model checker to analyze if the safety requirements and fault tolerance objectives are satisfied by the design in the presence of faults. If the design does not tolerate the specified number of faults (or probability threshold of fault occurrence), then the tool produces counterexamples leading to safety requirement violation in the presence of faults, as well as all minimal set of fault combinations that can cause the safety requirement to be violated.
5) The safety engineers examine the results to assess the validity of the fault combinations and the fault tolerance level of the system design. If a design change is warranted, the model will be updated with the latest design change and the above process is repeated.

There are other tools purpose-built for safety analysis, including AltaRica [39], smartIFlow [28] and xSAP [8]. These tools and their accompanying notations are separate from the system development model. Other tools extend existing system models, such as HiP-HOPS [18] and the AADL Error Model Annex, Version 2 (EMV2) [23]. EMV2 uses enumeration of faults in each component and explicit propagation of faulty behavior to perform error analysis. The required propagation relationships must be manually added to the system model and can become complex and lead to mistakes in the analysis.

In contrast, the Safety Annex supports model checking and quantitative reasoning by attaching behavioral faults to components and then using the normal behavioral propagation and proof mechanisms built into the AGREE AADL annex. This allows users to reason about the evolution of faults over time, and produce counterexamples demonstrating how component faults lead to failures. Our approach adapts the work of Joshi et. al [32] to the AADL modeling language. Stewart, et. al provide more information on the approach [45], and the tool and relevant documentation can be found at: https://github.com/loonwerks/AMASE/.

III. Fault Modeling with the Safety Annex

To demonstrate the fault modeling capabilities of the Safety Annex we will use the Wheel Brake System (WBS) described in AIR6110 [1]. This system is a well-known example that has been used as a case study for safety analysis, formal verification, and contract based design [10], [14], [15], [30]. The preliminary work for the safety annex was based on a simple model of the WBS [45]. To demonstrate a more complex fault modeling process, we constructed a functionally and structurally equivalent AADL version of the more complex WBS NuSMV/xSAP models [15], as illustrated in Figure 2.

The WBS is composed of two main parts: the Line Replaceable Unit control system and the electro-mechanical physical system. The control system electronically controls the physical system and contains a redundant channel of the Braking System Control Unit (BSCU) in case a detectable fault occurs in the active channel. It also commands antiskid braking. The physical system consists of the hydraulic circuits running from hydraulic pumps to wheel brakes as well as valves that control the hydraulic fluid flow. This system provides braking force to each of the eight wheels of the aircraft. The wheels are all mechanically braked in pairs (one pair per landing gear). For simplicity, Figure 1 displays only two of the eight wheels.

There are three operating modes in the WBS model:

- In *normal* mode, the system is composed of a *green* hydraulic pump and one meter valve per each of the eight wheels. Each of the meter valves are controlled through electronic commands coming from the active channel of the BSCU. These signals provide braking and antiskid commands for each wheel. The braking command is determined through a sensor on the pedal and the antiskid command is determined by the *Wheel Sensors*.
- In *alternate* mode, the system is composed of a *blue* hydraulic pump, four meter valves, and four antiskid shutoff valves, one for each landing gear. The meter valves are mechanically commanded through the pilot pedal corresponding to each landing gear. There are two ways the system can change into alternate mode. The selector can choose the blue circuit when the BSCU sends a system invalid signal, or the system can switch to the blue circuit when the selector detects lack of pressure in the green circuit.
- In *emergency* mode, the system mode is entered if the *blue* hydraulic pump fails. The accumulator pump has a reserve of pressurized hydraulic fluid and will supply this to the blue circuit in emergency mode.

The WBS architecture model in AADL contains 30 different kinds of components, 169 component instances, and a model depth of 5 hierarchical levels.

The behavioral model is encoded using the AGREE annex and the behavior is based on descriptions found in AIR6110. The top level system properties are given by the requirements and safety objectives in AIR6110. All of the subcomponent contracts support these system safety objectives through the use of assumptions on component input and guarantees on the output. The WBS behavioral model in AGREE annex includes one top-level assumption and 11 top-level system properties, with 113 guarantees allocated to subsystems.

An example system safety property is to ensure that there is no inadvertent braking of any of the wheels. This is based on a failure condition described in AIR6110 is *Inadvertent wheel braking on one wheel during takeoff shall be less than* 1.0×10^{-9} *per takeoff*. Inadvertent braking means that braking force is applied at the wheel but the pilot has not pressed the brake pedal. In addition, the inadvertent braking requires that power and hydraulic pressure are both present, the plane is not stopped, and the wheel is rolling (not skidding). The property

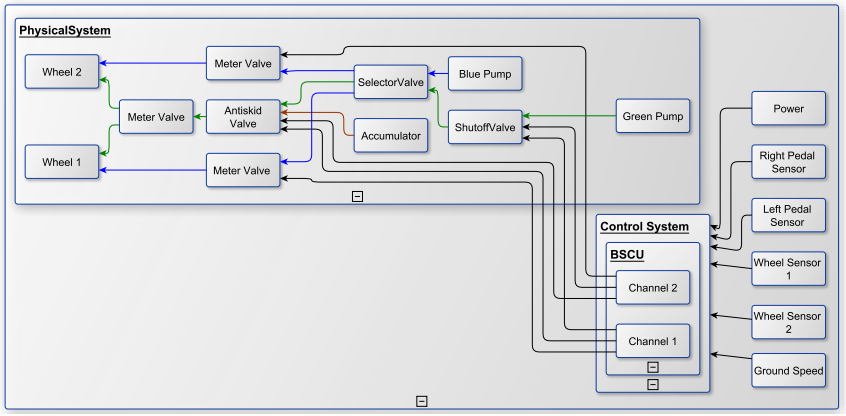

Fig. 1. Wheel Brake System

is stated in AGREE such that inadvertent braking does *not* occur, as shown in Figure 2.

```
lemma "(S18-WBS-0325) Never inadvertent braking of wheel 1" :
        true -> (not(POWER)
            or (not HYD_PRESSURE_MAX)
            or (mechanical_pedal_pos_L
            or (not SPEED)
            or (wheel_braking_force1 <= 0)
            or (not W1ROLL)));
```

Fig. 2. AGREE Contract for Top Level Property: Inadvertent Braking

A. Component Fault Modeling

The usage of the terms error, failure, and fault are defined in ARP4754A and are described here for clarity [42]. An *error* is a mistake made in implementation, design, or requirements. A *fault* is the manifestation of an error and a *failure* is an event that occurs when the delivered service of a system deviates from correct behavior. If a fault is activated under the right circumstances, that fault can lead to a failure. The terminology used in EMV2 differs slightly for an error: an error is a corrupted state caused by a fault. The error propagates through a system and can manifest as a failure. In this paper we use the ARP4754A terminology with the added definition of *error propagation* as used in EMV2. An error is a mistake made in design or code and an error propagation is the propagation of the corrupted state caused by an active fault.

The Safety Annex is used to add possible faulty behaviors to a component model. Within the AADL component instance model, an annex is added which contain the fault definitions for the given component. The flexibility of the fault definitions allows the user to define numerous types of fault *nodes* by utilizing the AGREE node syntax. A library of common fault nodes has been written and is available in the project GitHub repository [43]. Examples of such faults include valves being stuck open or closed, output of a software component being nondeterministic, or power being cut off. When the fault analysis requires fault definitions that are more complex, these nodes can easily be created and used in the model.

When a fault is activated by its specified triggering conditions, it modifies the output of the component. This faulty behavior may lead to the violation of the contracts of other components in the system, including assumptions of downstream components. The impact of a fault is computed by the AGREE model checker when the safety analysis is run on the fault model.

The majority of faults that are connected to outputs of components are known as *symmetric*. That is, whatever components receive this faulty output will receive the same faulty output value. Thus, this output is seen symmetrically. An alternative fault type is *asymmetric*. This pertains to a component with a 1-n output: one output which is sent to many receiving components. This fault can present itself differently to the receiving components. For instance, in a boolean setting, one component might see a true value and the rest may see false. This type of fault is modeled using the keyword *asymmetric*. For more information on fault definitions and modeling possibilities, we refer readers to the Safety Annex Users Guide [43].

As an illustration of fault modeling using the Safety Annex, we look at one of the components relevant to the inadvertent braking property: the brake pedal. When the mechanical pedal is pressed, a sensor reads this information and passes an electronic signal to the BSCU that then commands hydraulic pressure to the wheels.

One possible failure for this sensor is inversion of its output value. This fault can be triggered with probability 5.0×10^{-6} as described in AIR6110 (in reality, the component failure probability is collected from hardware specification sheets). The Safety Annex definition for this fault is shown in Figure 3. Fault behavior is defined through the use of a fault node called *inverted_fail*. When the fault is triggered, the nominal output of the component (*elec_pedal_position*) is replaced with its failure value (*val_out*).

The WBS fault model expressed in the Safety Annex contains a total of 33 different fault types and 141 fault instances. The large number of fault instances is due to the redundancy in the system design and its replication to control 8 wheels.

```
annex safety {**
  fault SensorPedalPosition_ErroneousData
    "Inverted boolean fault" : faults.inverted_fail {
       inputs: val_in <- elec_pedal_position;
       outputs: elec_pedal_position <- val_out;
       probability: 5.0E-6 ;
       duration: permanent;
  }
};
```

Fig. 3. The Safety Annex for the Pedal Sensor

B. Implicit Error Propagation

In the Safety Annex approach, faults are captured as faulty behaviors that augment the system behavioral model in AGREE contracts. Unlike AADL EMV2, no explicit error propagation is necessary since the faulty behavior itself propagates through the nominal behavior contracts of the non-failed system components. The effects of any triggered fault are manifested through analysis of the AGREE contracts.

To illustrate key differences between implicit error propagation in the Safety Annex and explicit propagation in EMV2, we use a simple example derived from the WBS, shown in Figure 4. In this simplified WBS system, the physical signal from the Pedal component is detected by the Sensor and the pedal position value is passed to the Braking System Control Unit (BSCU) components. The BSCU generates a pressure command to the Valve component which applies hydraulic brake pressure to the Wheels.

In the EMV2 approach (top half of Figure 4), the "NoService" fault is explicitly propagated through all of the components. These fault types are essentially tokens that do not capture any analyzable behavior. At the system level, analysis tools supporting the EMV2 annex can aggregate the propagation information from different components to compose an overall fault flow diagram or fault tree.

The Safety Annex is used to add failure behaviors to the Sensor and Valve components and to specify the fault hypothesis (in this case, a single fault). When a fault condition is triggered in the analysis, the output behavior of that component is produced by its failure contract instead of its nominal contract. For our simplified example, if the Sensor component fault condition is triggered, the result is a "stuck at zero" error. The contract of the BSCU receives a zero input and proceeds as if the pedal has not been pressed. This will cause the top level system contract to fail.

C. Explicit Error Propagation

Failures in hardware (HW) components can trigger behavioral faults in the system components that depend on them. For example, a CPU Failure may trigger faulty behavior in the threads bound to that CPU. In addition, a failure in one HW component may trigger failure in other HW components located nearby, such as overheating, fire, or explosion in the containment location. The Safety Annex provides the capability to explicitly model the impact of hardware failures on other faults, behavioral or non behavioral. The explicit propagation to non behavioral faults is similar to that provided in EMV2.

To better model faults at the system level dependent on HW failures, a fault model element is introduced called a *hardware fault*. Users are not required to specify behavioral effects for the HW faults, nor are data ports necessary on which to apply the fault definition. Users specify dependencies between the HW component faults and faults that are defined in other components, either HW or SW. The hardware fault then acts as a trigger for dependent faults. This allows a simple propagation from the faulty HW component to the SW components that rely on it, affecting the behavior on the outputs of the affected SW components.

D. Fault Analysis Statements

The Safety Annex also allows the user to specify constraints on the occurence of faults in the system.

- The *max fault hypothesis* specifies the maximum number of faults that can be active at any point in the analysis. This is analogous to restricting the cutset to a specified maximum number of terms in the fault tree analysis in a traditional safety analysis.
- The *probabilistic fault hypothesis* specifieds that only faults whose probability of simultaneous occurrence is above the given threshold should be onsidered. This is analogous to restricting the cutsets to only those whose probability is above the specified value.

In the former case, we assert that the sum of the true *fault__trigger* variables is at or below some integer threshold. In the latter, we determine all combinations of faults whose probabilities are above the specified probability threshold, and describe this as a proposition over *fault__trigger* variables.

E. Fault Activation

With the introduction of dependent faults, active faults are divided into two categories: independently active (activated by its own triggering event) and dependently active (activated when the faults they depend on become active). The top level fault hypothesis applies to independently active faults. Faulty behaviors augment nominal behaviors whenever their corresponding faults are active (either independently active or dependently active).

IV. Tool Architecture and Implementation

The Safety Annex is written in Java as a plug-in for the OSATE AADL toolset, which is built on Eclipse. It is not designed as a stand-alone extension of the language, but works with behavioral contracts specified using the AGREE AADL annex [21]. The architecture of the Safety Annex is shown in Figure 5.

AGREE contracts are used to define the nominal behaviors of system components as *guarantees* that hold when *assumptions* about the values the component's environment are met. When an AADL model is annotated with AGREE contracts and the fault model is created using the Safety Annex, the model is transformed through AGREE into a Lustre model [27]

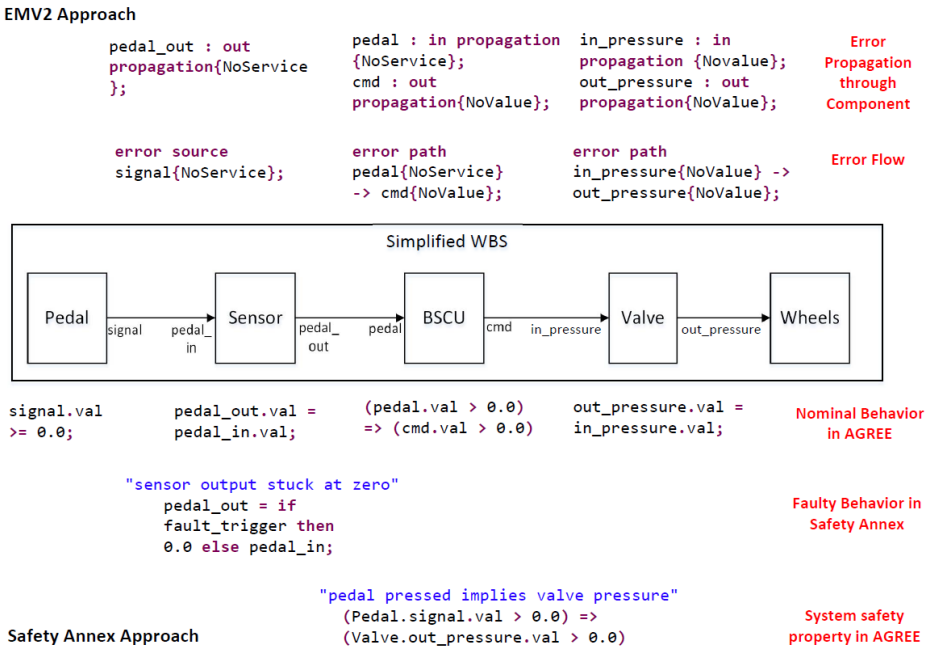

Fig. 4. Differences between Safety Annex and EMV2

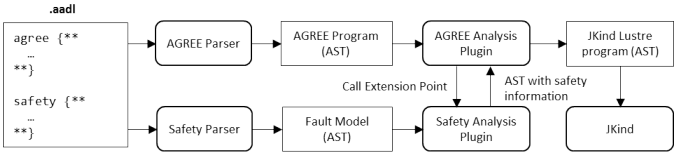

Fig. 5. Safety Annex Plug-in Architecture

containing the behavioral extensions defined in the AGREE contracts for each system component.

When performing fault analysis, the Safety Annex extends the AGREE contracts to allow faults to modify the behavior of component inputs and outputs. An example of a portion of an initial AGREE node and its extended contract is shown in Figure 6. The left column of the figure shows the nominal Lustre pump definition is shown with an AGREE contract on the output; and the right column shows the additional local variables for the fault (boxes 1 and 2), the assertion binding the fault value to the nominal value (boxes 3 and 4), and the fault node definition (box 5). Once augmented with fault information, the AGREE model (translated into the Lustre dataflow language [27]) follows the standard translation path to the model checker JKind [24], an infinite-state model checker for safety properties.

There are two different types of fault analysis that can be performed on a fault model. The Safety Annex plugin intercepts the AGREE program and add fault model information to the model depending on which form of fault analysis is being run.

Verification in the Presence of Faults: This analysis returns one counterexample when fault activation per the fault hypothesis can cause violation of a property. The augmentation from Safety Annex to the AGREE program includes traceability information so that when counterexamples are displayed to users, the active faults for each component are visualized.

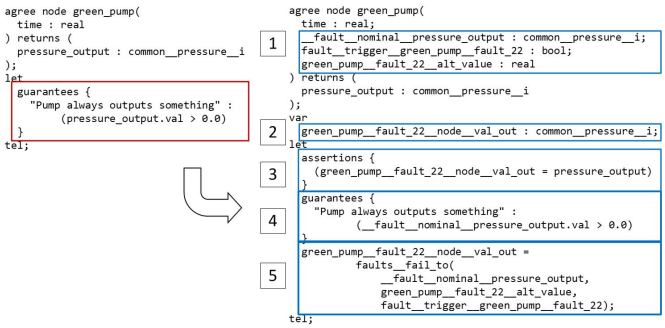

Fig. 6. Nominal AGREE Node and Extension with Faults

Generate Minimal Cut Sets: This analysis returns all minimal cut sets that lead to the violation of a property, transformed from a full enumeration of all minimal set of model elements necessary for the inductive proofs of the property [4], [25].

V. Analysis of the WBS Model

In this section we describe the nominal model analysis and the fault analysis for the WBS example.

A. Nominal Model Analysis

Before performing fault analysis, users should first check that the safety properties are satisfied by the nominal design model. This analysis can be performed monolithically or compositionally in AGREE. Using monolithic analysis, the

S18-WBS-R-0321: Loss of all wheel braking during landing or RTO shall be less than 5.0×10^{-7} per flight.
S18-WBS-R/L-0322: Asymmetrical loss of wheel braking (Left/Right) shall be less than 5.0×10^{-7} per flight.
S18-WBS-0323: Inadvertent braking with all wheels locked shall be less than 1.0×10^{-9} per takeoff.
S18-WBS-0324: Inadvertent braking with all wheels shall be less than 1.0×10^{-9} per takeoff.
S18-WBS-0325-wheelX: Inadvertent braking of wheel X shall be less than 1.0×10^{-9} per takeoff.

TABLE I
SAFETY PROPERTIES OF WBS

contracts at the lower levels of the architecture are flattened and used in the proof of the top level safety properties of the system. Compositional analysis, on the other hand, will perform the proof layer by layer, starting from the top, decomposing the larger proof into smaller problems. For a more comprehensive description of these types of proofs and analyses, see additional publications related to AGREE [3], [20]

The WBS has a total of 13 safety properties to be verifed at the top level. The subcomponents implementing the system each have their own assumptions and guarantees specified. The top level properties are shown in Table I. Contract S18-WBS-0325-wheelX has eight instances, one for each wheel. The behavioral model in total consists of 36 assumptions and 246 guarantees.

B. Fault Model Analysis

This subsection describes the options for fault model analysis using the Safety Annex and discusses the analytical results obtained.

1) Verification in the Presence of Faults: Max N Analysis: Using a max number of faults for the hypothesis, the user can constrain the number of simultaneously active faults in the model. The faults are added to the AGREE model for the verification and the model checker attempts to prove the top level properties given these constraints. If this cannot be done, a counterexample in generated showing which of the faults (N or less) are active and which contracts are violated.

The user can choose to perform either compositional or monolithic analysis using a max N fault hypothesis. In compositional analysis, the analysis proceeds in a top down fashion. To prove upper layer properties, the properties in the layer directly beneath that level are used to perform the proof. Users constrain the maximum number of faults within each layer of the model by specifying the maximum fault hypothesis statement to that layer. If any lower level property failed due to activation of faults, the property verification at the higher level can no longer be trusted because the higher level properties were proved based on the assumption that the direct sub-level contracts are valid. This form of analysis is helpful to see weaknesses in a given layer of the system.

In monolithic analysis the layers of the model are flattened, which allows a direct correspondence between all faults in the model and their effects on the top level properties. As with compositional analysis, a counterexample shows these N or fewer active faults.

2) Verification in the Presence of Faults: Probabilistic Analysis: With a probabilistic fault hypothesis, the analysis is performed with combinations of faults whose probability of occurrence is greater than the specified probability threshold. This is done by inserting assertions that allow those combinations in the Lustre code. If the model checker proves that the safety properties can be violated with any of those combinations, a counterexample will be generated. Probabilistic analysis done in this way must utilize the AGREE option for monolithic analysis.

To perform this analysis, it is assumed that the non-hardware faults occur independently and possible combinations of faults are computed and passed to the Lustre model to be checked by the model checker. The computation first removes all faults from consideration that are too unlikely given the probability threshold. The remaining faults are arranged in a priority queue $\mathcal{Q}$ from high to low probability of occurrence. Assuming independence in the set of faults, we take a fault with highest probability from the queue and attempt to combine the remainder of the faults in $\mathcal{R}$. If this combination is lower than the threshold, then we do not take into consideration this set of faults and instead remove the tail of the remaining faults in $\mathcal{R}$.

In this calculation, we assume independence among the faults, but in the Safety Annex it is possible to define dependence between faults using a fault propagation statement. After fault combinations are computed, the triggered dependent HW faults are added to the combination as appropriate. The dependencies are implemented in the *Verify in the Presence of Faults* options for analysis, but not yet implemented in the *Generate Minimal Cut Sets* analysis options.

3) Generate Minimal Cut Sets: Max N Analysis: To perform analysis, users specify the max N fault hypothesis statement in the top level of the system implementation to be verified, and select the *Generate Minimal Cut Sets* option to conduct the analysis. This gives cut sets of cardinality *less than or equal to* N. Generate MinCutSet analysis was performed on the Wheel Brake System and results are shown in Table II. Notice in Table II, the label across the top row refers to the cardinality (C) and how many cut sets of that cardinality were found. For the *Never loss of all wheel braking* property (R-0321) in the first row of Table II, there are 6 minimal cut sets of cardinality 1. These include faults on the selector and shutoff valves and pumps. These reflect single points of failure of the system by failure to supply hydraulic fluid to the wheels. Many of these could be mitigated by defining a feedback from the wheel to the control system. When braking is commanded in a certain mode of operation and no pressure is supplied, the control system could change modes of the system. This behavior was not incorporated into the WBS model, but could be considered for future design changes. There are no combinations of strictly two or three faults that can violate this property, but for $c = 4$ there exists a single cut set. These faults relate to the four meter valves that provide hydraulic pressure to the pairs of wheels. If all four meter

valves go out at the same time, this will result in a loss of all wheel braking.

The results from this table could be further explored to see how the multiple wheels, meter valves per wheel, and wheel brake subsystems interact to create large numbers of cut sets per cardinality. For instance, R-0324's contract references all 8 wheels and every subsystem pertaining to these 8 wheels contribute in a combinatorial increase in the number of cut sets.

Due to the increasing number of possible fault combinations at $N = 6$, the computational time increases quickly. The WBS analysis was only run to $N = 6$ for this reason.

4) Generate Minimal Cut Sets: Probabilistic Analysis: To perform analysis, users specify the probablistic fault hypothesis statement in the top level of the system implementation to be verified, and select the *Generate Minimal Cut Sets* option to conduct the analysis. This gives cut sets whose probability of simultaneous occurrence exceed the given threshold in the probability hypothesis.

The probabilistic analysis for the WBS was given a top level threshold of 1.0×10^{-9} as stated in AIR6110. The faults associated with various components were all given probability of occurrence compatible with the discussion in this same document.

As shown in Table III, the number of allowable combinations drops considerably when given probabilistic threshold as compared to just fault combinations of certain cardinalities. For example, one contract (inadvertent wheel braking of all wheels) had over a million minimal cut sets produced when looking at it in terms of max N analysis, but after taking probabilities into account, it is seen that only one combination of faults can violate this property.

C. Use of Analysis Results to Drive Design Change

We use a single top level requirement of the WBS: S18-WBS-0323 (Never indadvertent braking with all wheels locked) to illustrate how Safety Annex can be used to detect design flaws and how faults can affect the behavior of the system. This safety property description can be found in detail in Section III. Upon running max N fault analysis with $N = 1$, the pedal sensor fault (output inverted) was shown to be a single point of failure for this safety property.

Various strategies are possible to mitigate the problem, depending on the goals of the system, the architecture currently modeled, and the mitigation strategies that are desired. In the case of the pedal sensor in the WBS, redundant pedal sensors were added. Subsequent runs of the analysis demonstrated resilience to the failure of a single pedal sensor.

As can be seen through this single example, a system as large as the WBS would benefit from many iterations of this process. Furthermore, if the model is changed even slightly on the system development side, it would automatically be seen from the safety analysis perspective and any negative outcomes would be shown upon subsequent analysis runs. This effectively eliminates any miscommunications between the system development and analysis teams and creates a new safeguard regarding model changes.

For more information on types of fault models that can be created as well as details on analysis results, see the users guide located in the GitHub repository [43]. This repository also contains all models used in this project.

VI. Related Work

A model-based approach for safety analysis was proposed by Joshi et. al in [30]–[32]. 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 they propagate faults explicitly through user-defined propagations, which we call *failure logic modeling* (FLM) or through existing behavioral modeling, which we call *failure effect modeling* (FEM). The next is whether models and notations are *purpose-built* for safety analysis vs. those that extend *existing system models* (ESM).

For FEM approaches, there are several additional dimensions. One dimension involves whether *causal* or *non-causal* models are allowed. Non-causal models allow simultaneous (in time) bi-directional error propagations, which allow more natural expression of some failure types (e.g. reverse flow within segments of a pipe), but are more difficult to analyze. A final dimension involves whether analysis is *compositional* across layers of hierarchically-composed systems or *monolithic*. Our approach is an extension of AADL (ESM), causal, compositional, mixed FLM/FEM approach.

Tools such as the AADL Error Model Annex, Version 2 (EMV2) [23] and HiP-HOPS for EAST-ADL [18] are *FLM*-based *ESM* approaches. As previously discussed, given many possible faults, these propagation relationships require substantial user effort and become more complex. In addition, it becomes the analyst's responsibility to determine whether faults can propagate; missing propagations lead to unsound analyses. In our Safety Annex, propagations occur through system behaviors (defined by the nominal contracts) with no additional user effort.

The model-based safety assessment toolset COMPASS (Correctness, Modeling project and Performance of Aerospace Systems) [11] is closely related to our work. COMPASS is a mixed *FLM/FEM*-based, *causal compositional* tool suite that uses the SLIM language, which is based on a subset of AADL, for its input models [12], [16]. In SLIM, a nominal system model and the error model are developed separately and then transformed into an extended system model. This extended model is automatically translated into input models for the NuSMV model checker [19], [38], MRMC (Markov Reward Model Checker) [33], [37], and RAT (Requirements Analysis Tool) [40]. The safety analysis tool xSAP [8] can be invoked in order to generate safety analysis artifacts such as

Property	$c=1$	$c=2$	$c=3$	$c=4$	$c=5$	$c=6$	$c=7+$
R-0321	6	0	0	1	144	7776	-
R-0322	32	0	0	0	0	0	-
L-0322	32	0	0	0	0	0	-
0323	90	0	0	0	0	0	-
0324	8	3,401	6,800	66,472	435,358	1,892,832	-
0325-WX	20	0	0	0	0	0	-

TABLE II
WBS MINCUTSET ANALYSIS RESULTS FOR CARDINALITY c

Property	$c=1$	$c=2$	$c=3$	$c=4$	$c=5$	$c=6$	$c=7$	$c=8$
R-0321	0	0	0	0	0	0	0	0
R-0322	32	0	0	0	0	0	0	0
L-0322	32	0	0	0	0	0	0	0
0323	90	0	0	0	0	0	0	0
0324	0	1	0	0	0	0	0	0
0325-WX	20	0	0	0	0	0	0	0

TABLE III
WBS MINCUTSET RESULTS FOR PROBABILISTIC ANALYSIS

fault trees and FMEA tables [9]. COMPASS is an impressive tool suite, but some of the features that make AADL suitable for SW/HW architecture specification: event and event-data ports, properties, threads, and processes, appear to be missing, which means that the SLIM language may not be suitable as a general system design notation (ESM). A SLIM nominal model is extended by creating an error model. This is defined by type, implementation, and effect. Furthermore, the error model specifies outgoing and incoming propagations similar to the EMV2 error annex for AADL [23]. Outgoing error propagations report the error state to other components. If their error states are affected, the other components will have a corresponding incoming propagation [9], [11], [22]. This differs from our approach in that error propagations do not need to be specified and the propagations are performed behaviorally through the violation of contracts on connected components.

SmartIFlow [28] is a *FEM*-based, *purpose-built*, *monolithic non-causal* safety analysis tool that describes components and their interactions using finite state machines and events. Verification is done through an explicit state model checker which returns sets of counterexamples for safety requirements in the presence of failures. SmartIFlow allows *non-causal* models containing simultaneous (in time) bi-directional error propagations. On the other hand, the tools do not yet appear to scale to industrial-sized problems, as mentioned by the authors [28]: "As current experience is based on models with limited size, there is still a long way to go to make this approach ready for application in an industrial context".

The Safety Analysis and Modeling Language (SAML) [26] is a *FEM*-based, *purpose-built*, *monolithic causal* safety analysis language. System models constructed in SAML can be used used for both qualitative and quantitative analyses. It allows for the combination of discrete probability distributions and non-determinism. The SAML model can be automatically imported into several analysis tools like NuSMV [19], PRISM (Probabilistic Symbolic Model Checker) [34], or the MRMC probabilistic model checker [33].

AltaRica [7], [39] is a *FEM*-based, *purpose-built*, *monolithic* safety analysis language with several dialects. There is one dialect of AltaRica which use dataflow (*causal*) semantics, while the most recent language update (AltaRica 3.0) uses non-causal semantics. The dataflow dialect has substantial tool support, including the commercial Cecilia OCAS tool from Dassault [6]. For this dialect the Safety assessment, fault tree generation, and functional verification can be performed with the aid of NuSMV model checking [13]. Failure states are defined throughout the system and flow variables are updated through the use of assertions [5]. AltaRica 3.0 has support for simulation and Markov model generation through the OpenAltaRica (www.openaltarica.fr) tool suite.

Formal verification tools based on model checking have been used to automate the generation of safety artifacts [8], [13], [17]. This approach has limitations in terms of scalability and readability of the fault trees generated. Work has been done towards mitigating these limitations by the scalable generation of readable fault trees [14].

VII. CONCLUSION

We have developed an extension to the AADL language with tool support for formal analysis of system safety properties in the presence of faults. Faulty behavior is specified as an extension of the nominal model, allowing safety analysis and system implementation to be driven from a single common model. Both symmetric and asymmetric faulty behaviors are supported. This new Safety Annex leverages the AADL structural model and nominal behavioral specification (using the AGREE annex) to propagate faulty component behaviors without the need to add separate propagation specifications to the model. Implicit error propagation enables safety engineers to inject failures/faults at component level and assess the effect of behavioral propagation at the system level. It also supports explicit error propagation that allows safety engineers to describe dependent faults that are not easily captured using implicit error propagation. Generation of minimal cut sets collects all minimal set of fault combinations that can cause violation of the top level properties. For more details on the tool, models, and approach, see the Architectural Modeling

and Analysis for Safety Engineering (AMASE) project final report for NASA [44]. To access the tool plugin, users manual, or models, see the repository [43].

Acknowledgments. This research was funded by NASA contract NNL16AB07T and the University of Minnesota College of Science and Engineering Graduate Fellowship.

REFERENCES

[1] AIR 6110. Contiguous Aircraft/System Development Process Example, Dec. 2011.

[2] AS5506C. Architecture Analysis & Design Language (AADL), Jan. 2017.

[3] J. Backes, D. Cofer, S. Miller, and M. W. Whalen. Requirements Analysis of a Quad-Redundant Flight Control System. In *NFM*, volume 9058 of *LNCS*, pages 82–96, 2015.

[4] J. Bendík, E. Ghassabani, M. Whalen, and I. Černá. Online enumeration of all minimal inductive validity cores. In *International Conference on Software Engineering and Formal Methods*, pages 189–204. Springer, 2018.

[5] P. Bieber, C. Bougnol, C. Castel, J. P. Heckmann, C. Kehren, S. Metge, and C. Seguin. Safety Assessment with Altarica - Lessons Learnt Based on Two Aircraft System Studies. In *In 18th IFIP World Computer Congress*, 2004.

[6] P. Bieber, C. Bougnol, C. Castel, J.-P. H. C. Kehren, S. Metge, and C. Seguin. Safety assessment with altarica. In *Building the Information Society*, pages 505–510. Springer, 2004.

[7] P. Bieber, J.-L. Farges, X. Pucel, L.-M. Sèjeau, and C. Seguin. Model - based safety analysis for co-assessment of operation and system safety: application to specific operations of unmanned aircraft. In *ERTS2*, 2018.

[8] 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 *TACAS*, 2016.

[9] M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta. The compass 3.0 toolset (short paper). In *IMBSA 2017*, 2017.

[10] M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei. Efficient Anytime Techniques for Model-Based Safety Analysis. In *Computer Aided Verification*, 2015.

[11] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In *Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg, 2009.

[12] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Yen Nguyen, T. Noll, and M. Roveri. Model-based codesign of critical embedded systems. 507, 2009.

[13] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta. Symbolic Model Checking and Safety Assessment of Altarica Models. In *Science of Computer Programming*, volume 98, 2011.

[14] M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta. Formal safety assessment via contract-based design. In *Automated Technology for Verification and Analysis*, 2014.

[15] 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 *CAV 2015, Proceedings, Part I*, pages 518–535, 2015.

[16] M. Bozzano, A. Cimatti, M. Roveri, J. P. Katoen, V. Y. Nguyen, and T. Noll. Codesign of dependable systems: A component-based modeling language. In *2009 7th IEEE/ACM International Conference on Formal Methods and Models for Co-Design*, 2009.

[17] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *ATVA*, 2007.

[18] D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lönn, and Y. Papadopoulos. Systems Modeling with EAST-ADL for Fault Tree Analysis through HiP-HOPS*. *IFAC Proceedings Volumes*, 46(22):91 – 96, 2013.

[19] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2000.

[20] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional verification of architectural models. In *NASA Formal Methods Symposium*, pages 126–140. Springer, 2012.

[21] D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In *NFM 2012*, volume 7226, pages 126–140, April 2012.

[22] COMPASS Users Manual. http://www.compass-toolset.org/docs/compass-manual.pdf.

[23] P. Feiler, J. Hudak, J. Delange, and D. Gluch. Architecture fault modeling and analysis with the error model annex, version 2. Technical Report CMU/SEI-2016-TR-009, Software Engineering Institute, 06 2016.

[24] A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani. The JKind Model Checker. *CAV 2018*, 10982, 2018.

[25] E. Ghassabani, M. W. Whalen, and A. Gacek. Efficient generation of all minimal inductive validity cores. *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 31–38, 2017.

[26] M. Gudemann and F. Ortmeier. A framework for qualitative and quantitative formal model-based safety analysis. In *HASE 2010*, 2010.

[27] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language Lustre. In *IEEE*, volume 79(9), pages 1305–1320, 1991.

[28] P. Hönig, R. Lunde, and F. Holzapfel. Model Based Safety Analysis with smartIflow. *Information*, 8(1), 2017.

[29] P. Hönig, R. Lunde, and F. Holzapfel. Model Based Safety Analysis with smartIflow. *Information*, 8(1), 2017.

[30] 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.

[31] 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.

[32] 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*, 2005.

[33] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A markov reward model checker. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, QEST '05. IEEE Computer Society, 2005.

[34] 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.

[35] 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*, 2011.

[36] MathWorks. The MathWorks Inc. Simulink Product Web Site. http://www.mathworks.com/products/simulink, 2004.

[37] MRMC: Markov Rewards Model Checker. http://wwwhome.cs.utwente.nl/ zapreevis/mrmc/.

[38] NuSMV Model Checker. http://nusmv.itc.it.

[39] 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*, 46(22), 2013.

[40] RAT: Requirements Analysis Tool. http://rat.itc.it.

[41] SAE ARP 4761. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.

[42] SAE ARP4754A. Guidelines for Development of Civil Aircraft and Systems, December 2010.

[43] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson. Safety annex for aadl repository. https://github.com/loonwerks/AMASE, 2018.

[44] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson. Architectural Modeling and Analysis for Safety Engineering (AMASE) Final Report. https://github.com/loonwerks/AMASE/tree/master/doc, September 2019.

[45] D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl. Architectural Modeling and Analysis for Safety Engineering. In *IMBSA 2017*, pages 97–111, 2017.