ICSE paper outline

Danielle Stewart*, Jing (Janet) Liu†, Michael W. Whalen*, Darren Cofer†, Mats Heimdahl*, Michael Peterson‡
*University of Minnesota
Department of Computer Science and Engineering
Email: {dkstewar, whalen, heimdahl}@umn.edu
†Rockwell Collins
Advanced Technology Center
Email: {Jing.Liu, Darren.Cofer}@rockwellcollins.com
‡Rockwell Collins
Commercial Systems Flight Controls Safety Engineering
Email: Michael.Peterson@rockwellcollins.com

***Abstract*—Risk and fault analysis are important activities that help to ensure that critical systems operate in an expected way. As critical systems become more dependent on software components, analysis regarding fault propagation through these software components becomes more important. The methods used to perform these analyses require understandability from the side of the analyst, scalability in terms of system size, and mathematical correctness in order to be sufficient proof that a system is safe. Determination of the events that can cause failures to propagate through a system as well as the affects of these propagations can be a time consuming and error prone process. In this paper, we describe a technique for determining these events with the use of a model checker and producing safety analysis artifacts that encode pertinant system safety information. We describe the technique of generating these safety artifacts, prove that our approach is sound, and describe its implementation in the Osate tool suite for AADL. We then present experiments in which we benchmark our approach in terms of scalability and the artifacts produced.**

I. Introduction

Risk and safety analyses are an important activity used to ensure that critical systems operate in an expected way. From nuclear power plants and airplanes to heart monitors and automobiles, critical systems are vitally important in our society. The systems are required to not only operate safely under nominal (normal) conditions, but also under conditions when faults are present in the system. Guaranteeing these properties when in the presence of faults is an active area of research and various safety artifacts are often required during the development process of critical systems. In these systems, software is an important component to assess in terms of safety. An area of interest in safety-critical system engineering and development is how failures in the system are propagated through software designs. It is a difficult problem to reason about failure propagations through software in a large scale critical system and automating the process can provide much needed insight into the system and potential problems that could be costly in terms of resources or even life.

For complex systems, safety analysis techniques can produce thousands of combinations of events that can cause system failure. Many of these events are propagated through software components. Determination of these events can be a very time consuming and error prone process. SAE Standard Aerospace Recommended Practice (ARP) 4761 provides general guidance on evaluating the safety aspects of a design [1] and from the design phase through to the detailed design phase, safety artifacts regarding fault propagation and effects are an important part of the assessment process.

In this paper, we describe a new technique for determining these events utilizing model checking techniques on a model of the system written in an architecture design language, AADL [2]. The model is annotated with assume-guarantee contracts for component level requirements on both hardware and software components using AGREE [3] and extended into a fault model using the Safety Annex [4], [5]. A benefit of using this approach for model development is that the faults are specified on a component level, e.g. hardware component, and the behavior of the system as a whole can be observed in the presence of this fault. In other words, it can be seen how the fault manifests itself and is propagated through the software components of the system.

The artifact produced utilizing this approach is a compositional fault tree in which probabilistic analysis is performed. At each layer of the system, a proof is produced showing the necessary model elements (including faults) that are required in order for the negation of the component requirement to be met. This is often referred to as a Top Level Event (TLE). In the end, the reasons for the fault propagation can be seen as well as the affects of such a propagation.

The rest of the paper is organized as follows. We first describe the architecture language and tool suites used for this approach in section II. We then describe the formalisms and methodology of the approach in section III followed by case studies and experimentation in section IV. Finally we discuss related work and conclusion in sections V and VI.

II. Background

This section provides a high level description of the pieces of the puzzle necessary in order to understand our approach. We describe the relevant algorithms and tools in order to make the later formalisms fit into the full picture.

A. Inductive Validity Cores

Given a complex model, it is often useful to extract traceability information related to the proof, in other words, which portions of the model were necessary to construct the proof. An algorithm was introduced by Ghassabani, et. al. to provide Inductive Validity Cores (IVCs) as a way to determine which model elements are necessary for the inductive proofs of the safety properties for sequential systems [6]. Given a safety property of the system, a model checker can be invoked in order to construct a proof of the property. The IVC generation algorithm can extract traceability information from that proof process and return the model elements required in order to prove the property. A short time later, this algorithm was refined in order to produce the minimal set of such IVC elements [7].

This algorithm is of interest to us for the following reason. By letting the negation of the safety property of interest be our top level event, we can utilize the algorithm to find the set of minimal IVCs which will tell us the minimal model elements (including faults) necessary to prove that this top level event occurs. In other words, we will know the minimal set of events required in order to cause the occurance of the top level event. These model elements consist of the active faults present in the system and the contracts of components that were effected by the presence of the fault. This gives a trace of how the fault propagates through system components and how the software behaves in the presence of the faults.

In section III, we show the formal definitions of IVCs in detail.

Our approach utilizes a few tools in order to generate the artifacts of interest and a brief background will be helpful. and hence the rest of the background section consists of a brief description of AADL, AGREE, and the SOTERIA tools and languages.

B. Architecture Analysis and Design Language

We are using the Architectural Analysis and Design Language (AADL) [8] to construct system architecture models. AADL is an SAE International standard that defines a language and 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.

C. Compositional Analysis

Compositional analysis of systems was introduced in order to address the scalability of model checking large software systems. Monolithic verification and compositional verification are two ways that mathematical verification of component properties can be performed. In monolithic analysis, the model is flattened and the top level properties are proved using only the leaf level contracts of the components. On the other hand, 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 [9].

D. Assume Guarantee Reasoning Environment

The Assume Guarantee Reasoning Environment (AGREE) is a tool for formal analysis of behaviors in AADL models [9]. 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 [10] 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 [3].

E. Safety Annex for AADL

The Safety Annex for AADL is a tool that provides the ability to reason about faults and faulty component behaviors in AADL models [4], [5]. In the Safety Annex approach, formal assume-guarantee contracts are used to define the nominal behavior of system components. The nominal model is verified using AGREE. The Safety Annex weaves faults into the nominal model and analyzes the behavior of the system in the presence of faults. The tool supports behavioral specification of faults and their implicit propagation through behavioral relationships in the model as well as provides support to capture binding relationships between hardware and software compönents of the system.

F. SOTERIA

The Safe and Optimal Techniques Enabling Recovery, Integrity, and Assurance (SOTERIA) tool is used to perform safety analysis of Integrated Modular Avionics (IMA) systems [11]. In the SOTERIA project, a compositional modeling language was developed and this language is used as input in order to automatically synthesize the qualitative and quantitative safety analyses. The tool is compositional in that it requires safety aspects at each component level which enables the generation of compositional fault trees.

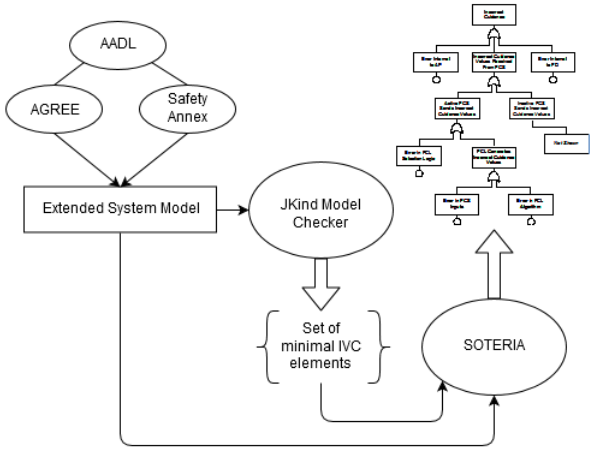

Fig. 1. Outline of the fault tree generation process

G. Fault Tree Analysis

A Fault Tree (FT) is a directed acyclic graph whose leaves model component failures and whose gates model failure propagation. The system failure under examination is the root of the tree and is called the Top Level Event (TLE). The Basic Events (BE) are the events that can occur in the system which lead to the TLE and in the graphical model, these correspond to the leaves. Figure 2 shows a simple example of a fault tree based on SAE ARP4761 [1]. In this example, the top level event corresponds to an aircraft losing all wheel braking. In order for this event to occur, all of the basic events must occur. This is seen through the use of the AND gate below the top level event. The gates in the fault tree describe how failures propagate through the system. Each gate has one output and one or more inputs. In Figure 2, the AND gate has three inputs and one output. The leaves of the tree represent the basic events of the system and in the case of this fault tree, these three events are also the *minimum cut set* for this top level event. The minimal cut set is the minimum set of basic events that must occur together in order to cause the TLE to occur. Finding these sets is important to FTA and has been an active area of interest in the research community since fault trees were first described in Bell Labs in 1961 [12].

There are two main types of FTA that we differentiate here as *qualitative* analysis and *quantitative* analysis. In qualitative analysis, the structure of the fault tree is considered and the cut sets are a way to indicate which combinations of component failures will cause the system to fail. On the other hand, in quantitative analysis the probability of the TLE is calculated given the probability of occurance of the basic events.

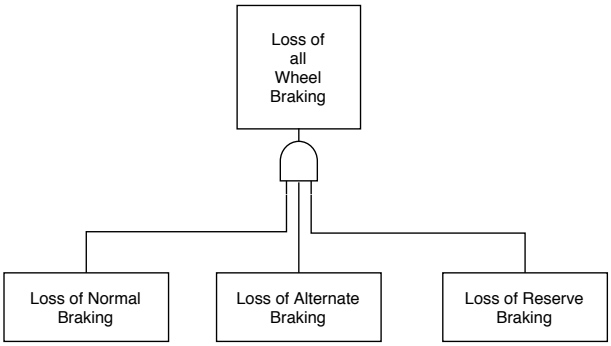

Fig. 2. A simple fault tree

The formal definition of a fault tree is provided in section III and more details regarding quantitative and qualitative FTA is explained there as well.

The general outline of our process is as follows and is shown in Figure 1 (Note: the fault tree shown in Figure 1 is generic and the text is irrelevant). The AADL system model is annotated with behavioral contracts using AGREE and then extended with faults using the Safety Annex. These three things together make the extended system model as shown in the figure. The extended system model is given to the JKind model checker [13] which outputs the minimal sets of IVC elements for each layer of the architecture. These IVC elements and pertinant details of the extended system model is given to the SOTERIA tool whose output is the graphical fault tree representation of the input.

III. Methodology

Given a complex model, it is often useful to extract traceability information related to the proof, in other words, which portions of the model were necessary to construct the proof. To this end, an algorithm was developed that efficiently computes the *inductive validity cores* (IVC) within a model necessary for the proofs of safety properties for sequential systems [14].

A. Preliminaries

Given a state space S, a transition system (I, T) consists of the initial state predicate $I : S \to \{0, 1\}$ and a transition step predicate $T : S \times S \to \{0, 1\}$. Reachability for (I, T) is defined as the smallest predicate $R : S \to \{0, 1\}$ which satisfies the following formulas:

$$\forall s.I(s) \Rightarrow R(s)$$
$$\forall s, s'.R \wedge T(s, s') \Rightarrow R(s')$$

A safety property $\mathcal{P} : S \to \{0, 1\}$ is a state predicate. A safety property $\mathcal{P}$ holds on a transition system (I, T) if it holds on all reachable states. More formally, $\forall s.R(s) \Rightarrow \mathcal{P}(s)$. When

this is the case, we write $(I,T) \vdash \mathcal{P}$. Following Ghassabani, et. al. [14], we formalize IVCs as follows.

Definition 1. *Inductive Validity Core*
Let (I,T) be a transition system and let $\mathcal{P}$ be a safety property with $(I,T) \vdash \mathcal{P}$. Then $S \subseteq T$ is an inductive validity core for $(I,T) \vdash \mathcal{P}$ iff $(I,S) \vdash \mathcal{P}$.

Definition 2. *Minimal Inductive Validity Core*
An inductive validity core S for $(I,T) \vdash \mathcal{P}$ is minimal iff $!\exists S'.S' \subset S \ni (I,S') \vdash \mathcal{P}$.

A fault tree is a directed acyclic graph (DAG) consisting of the node types *events* and *gates*. An event is an occurance within the system, typically the failure of a subsystem down to an individual component. Events can be grouped into *basic events* (BEs), which occur independently, and *intermediate events* which occur dependently and are caused by one or more other events. The event at the top of the tree, the *top level event* (TLE), is the event being analyzed. This event models the failure of the system (or subsystem) under consideration. The gates represent how failures propagate through the system and how failures in subsystems can cause system wide failures. The two main logic symbols used are the Boolean logic AND-gates and OR-gates. An AND-gate is used when the undesired top level event can only occur when all the lower conditions are true. The OR-gate is used when the undesired event can occur if any one or more of the next lower conditions is true. This is not a comprehensive list of gate types, but we focus our attention on these two common gate types.

To formalize a fault tree (FT), we use $GateTypes = \{And, Or\}$. Following Ruijters, et. al. [15], we formalize FT as follows.

Definition 3. *Fault Tree*
A FT is a 4-tuple $F = \langle BE, G, T, I\rangle$ consisting of the following components.

- *BE is the set of basic events*
- *G is the set of gates with $BE \cap G = \emptyset$. We write $E = BE \cup G$ for the set of elements.*
- *$T : G \to GateTypes$ is a function that describes the type of each gate.*
- *$I : G \to P(E)$ describes the inputs of each gate. We require that $I(G) \neq \emptyset$.*

The graph formed by $\langle E, I\rangle$ is a directed acyclic graph with a unique root TLE which is reachable from all nodes.

Definition 4. *Semantics of a Fault Tree*
The semantics of FT F is a function $\pi_F : \mathcal{P}(BE) \times E \vdash \{0,1\}$ where $\pi_F(S,e)$ indicates whether e fails given the set S of failed BEs. It is defined as follows.

- *For $e \in BE$, $\pi_F(S,e) = e \in S$.*
- *For $g \in G$ and $T(g) = And$, let $\pi_F(S,g) = \wedge_{x \in I(g)} \pi_F(S,x)$*
- *For $g \in G$ and $T(g) = Or$, let $\pi_F(S,g) = \vee_{x \in I(g)} \pi_F(S,x)$*

The interpretation of the TLE t is written as $\pi_F(S,t) = \pi_F(S)$. If the failure of S causes the TLE to occur, we write $\pi_F(S) = 1$.

In qualitative analysis, cut sets and minimal cut sets provide information about the vulnerabilities of a system in terms of its basic events. A *cut set* is a set of components that together can cause a system to fail. A *minimal cut set* is a cut set which contains the minimum number of basic events required in order to cause the TLE to occur. More formally, these are defined as follows.

Definition 5. *Cut Set*
$C \subseteq BE$ is a cut set of FT F if $\pi_F(C) = 1$.

Definition 6. *Minimal Cut Set*
$C \subseteq BE$ is a MCS if $\pi_F(C) = 1 \wedge \forall C' \subset C.\pi_F(C') = 0$. In other words, a minimal cut set (MCS) is a cut set of which no subset is a cut set.

Given the formalisms defined previously, we show that finding the minimal IVCs is equivalent to finding the MCS.

Theorem 1. *Finding all minimal IVCs is equivalent to finding the set of all MCSs*

Proof. Let $T = \{f_1, f_2, ..., f_n\}$ be the set of model elements corresponding to faults for the components and let $\mathcal{P}_{tle}$ be the top level event (TLE). By the definition for IVC, we know that $S \subseteq T$ is an IVC for $(I,T) \vdash \mathcal{P}_{tle}$ iff $(I,S) \vdash \mathcal{P}_{tle}$. In this case, $\pi_F(S) = 1$, i.e. given the set $S \subseteq T$ of failed model elements, the top level event occurs. Since S is minimal IVC, for any $M \subset S$, $(I,M) \nvdash \mathcal{P}_{tle}$. Hence it follows that $\pi_F(M) = 0$ for any $M \subset S$. □

B. Quantitative analysis

Quantitative analysis methods derive numerical values for fault trees. One of these calculations that is of particular interest to the safety engineering community is the probability of the occurance of the top level event (TLE). This value is of interest because in certain critical systems, the top level properties must be proved safe within a certain probabilistic threshold [1]. The next section describes probability theory and provides a description of how they are used to calculate the probability of the TLE. These definitions are based on the ones given in [15].

I will add the probability formalisms here later. For now, I just want to clarify my findings regarding the proof that Mike was talking about yesterday.
Assume events occur independently.

Or gate:
Probability is defined as follows:
$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.
Assume that the probability of events A, B is quite small (this is called the Rare Event Approximation). Then the term $P(A \wedge B)$ will also be quite small and hence is an "error term". If we drop this term from the calculations, we end up with the approximated probability being: $P(A) + P(B) \geq P(A) + P(B) - P(A \wedge B)$.
This is clearly a conservative estimate. It is also shown in the Fault Tree Handbook that this error is less than 0.1 I believe. I will have to review that again, but there is a bound on this error that has been previously proven.

And gate:
With the calculations of the and gate, this is where independence is really required. Using Bayes Rule, we have $P(A \wedge B) = P(B)P(A|B) = P(A)P(B|A)$ for conditionally dependent events and $P(A \wedge B) = P(B)P(A)$ for independent events. If we assume independence when the events are NOT independent, in the worst case scenario, we get something like B is completely dependent on A. Therefore $P(B|A) = 1$ and $P(A \wedge B) = P(B) \geq P(B)P(A)$. Not a conservative estimation. In the case of dependence, joint probability values would be required (acommon cause analysis).

Any other kind of gate uses combinations of these rules. These are not proofs that I made, I just looked at the logic of the operations. I am not sure how much we want to include in the paper. I assume a description is necessary, but a proof? No, this isn't a proof... This is just following definitions down a short little jaunt in the woods.

IV. Case Studies

To demonstrate the effectiveness of this approach and compare to state of the art existing tools, we describe two case studies.

A. Wheel Brake System

The Wheel Brake System (WBS) described in AIR6110 [16] is a well-known example that has been used as a case study for safety analysis, formal verification, and contract based design [17]–[20]. The preliminary work for the safety annex used a simplified model of the WBS [4]. In order to demonstrate scalability of our tools and compare results with other studies, we constructed a functionally and structurally equivalent AADL version of one of the most complex WBS NuSMV/xSAP models (arch4wbs) described in [17]. We refer readers to this publication for diagrams of the full arch4wbs model [17]. A simplified diagram taken from ARP4761 is shown in Figure 3.

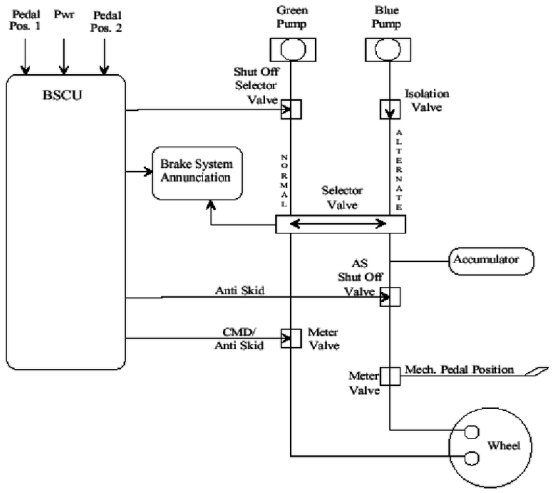

Fig. 3. Simplified model of the WBS from ARP4761

1) WBS architecture description: The WBS is composed of two main parts: the control system and the physical system. The control system electronically controls the physical system and contains a redundant Braking System Control Unit (BSCU) in case of failure. The physical system consists of the hydraulic circuits running from hydraulic pumps to wheel brakes. This is what provides braking force to each of the 8 wheels of the aircraft.

There are three operating modes in the WBS model. In *normal* mode, the system uses the *green* hydraulic circuit. The normal system is composed of the green hydraulic pump and one meter valve per each of the 8 wheels (shown as one wheel in Figure 3). Each of the 8 meter valves are controlled through electronic commands coming from the BSCU. These signals provide brake commands as well as antiskid commands for each of the wheels. The braking command is determined through a sensor on the pilot pedal position. The antiskid command is calculated based on information regarding ground speed, wheel rolling status, and braking commands.

In *alternate* mode, the system uses the *blue* hydraulic circuit. The wheels are all mechanically braked in pairs (one pair per landing gear). The alternate system is composed of the blue hydraulic pump, four meter valves, and four antiskid shutoff valves. The meter valves are mechanically commanded through the pilot pedal corresponding to each landing gear. If the system detects lack of pressure in the green circuit, the selector valve switches to the blue circuit. This can occur if there is a lack of pressure from the green hydraulic pump, if the green hydraulic pump circuit fails, or if pressure is cut off by a shutoff valve. If the BSCU channel becomes invalid, the shutoff valve is closed.

The last mode of operation of the WBS is the *emergency* mode. This is supported by the blue circuit but operates 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 model contains 30 different kinds of components, 169 component instances, a model depth of 5 hierarchical levels. The model includes one top-level assumption and 11 top-level system properties, with 113 guarantees allocated to subsystems. There are a total of 33 different fault types and 141 fault instances within the model. The large number of fault instances is due to the redundancy in the system design and its replication to control 8 wheels. An example property is to ensure no inadvertent braking of each of the 8 wheels. This means that if all power and hydraulic pressure is supplied (i.e., braking is commanded), then either the aircraft is stopped (ground speed is zero), or the mechanical pedal is pressed, or brake force is zero, or the wheel is not rolling.

V. Related Work

Who makes fault trees currently in our list of tools here? Who makes compositional fault trees? Why is our way better/different? Why is what we do important for AADL? AADL currently has a method to develop fault trees using error annex. Why is this better/different?

A model-based approach for safety analysis was proposed by Joshi et. al in [20]–[22]. 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 failure 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 [24] 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.

Closely related to our work is the model-based safety assessment toolset called COMPASS (Correctness, Modeling project and Performance of Aerospace Systems) [25]. 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 [26], [27]. 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 [28], [29], MRMC (Markov Reward Model Checker) [30], [31], and RAT (Requirements Analysis Tool) [32]. The safety analysis tool xSAP [33] can be invoked in order to generate safety analysis artifacts such as fault trees and FMEA tables [34]. 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, threads, and processes, appear to be missing, which means that the SLIM language may not be suitable as a general system design notation (ESM).

SmartIFlow [35] 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 failure propagations. On the other hand, the tools do not yet appear to scale to industrial-sized problems, as mentioned by the authors [35]: "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) [36] 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 [28], PRISM (Probabilistic Symbolic Model Checker) [37], or the MRMC probabilistic model checker [30].

AltaRica [38], [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. For this dialect the Safety assessment, fault tree generation, and functional verification can be performed with the aid of NuSMV model checking [40]. Failure states are defined throughout the system and flow variables are updated through the use of assertions [41]. 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 [33], [40], [42]. 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 [18].

VI. 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. 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. Next steps will include extensions to automate injection of Byzantine faults as well as automatic generation of fault trees. For more details on the tool, models, and approach, see the technical report [5]. 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] SAE ARP 4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," December 1996.

[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*, ser. LNCS, vol. 9058, 2015, pp. 82–96.

[4] D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl, "Architectural Modeling and Analysis for Safety Engineering," in *IMBSA 2017*, 2017, pp. 97–111.

[5] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson, "Safety Annex for Architecture Analysis Design and Analysis Language," University of Minnesota, Tech. Rep. 18-007, March 2018. [Online]. Available: https://www.cs.umn.edu/research/technical_reports/view/18-007

[6] E. Ghassabani, A. Gacek, and M. W. Whalen, "Efficient generation of inductive validity cores for safety properties," *CoRR*, vol. abs/1603.04276, 2016. [Online]. Available: http://arxiv.org/abs/1603.04276

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

[8] P. Feiler and D. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012.

[9] D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional Verification of Architectural Models," in *NFM 2012*, vol. 7226, April 2012, pp. 126–140.

[10] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Dataflow Programming Language Lustre," in *IEEE*, vol. 79(9), 1991, pp. 1305–1320.

[11] K. Y. Siu, H. Herencia-Zapana, P. Manolios, and M. Noorman, "Safe and Optimal Techniques Enabling Recovering, Integrity, and Assurance."

[12] C. Ericson, "Fault tree analysis - a history," in *Proceedings of the 17th International Systems Safety Conference*, 1999.

[13] A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The JKind Model Checker," *ArXiv e-prints*, Dec. 2017.

[14] E. Ghassabani, A. Gacek, and M. W. Whalen, "Efficient generation of inductive validity cores for safety properties," *CoRR*, vol. abs/1603.04276, 2016. [Online]. Available: http://arxiv.org/abs/1603.04276

[15] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," vol. 15-16, 05 2015.

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

[17] 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*, 2015, pp. 518–535.

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

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

[20] A. Joshi and M. P. Heimdahl, "Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier," in *SAFECOMP*, ser. LNCS, vol. 3688, 2005, p. 122.

[21] 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.

[22] 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.

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

[24] 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*, vol. 46, no. 22, pp. 91 – 96, 2013.

[25] 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.

[26] 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.

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

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

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

[30] 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*, ser. QEST '05. IEEE Computer Society, 2005.

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

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

[33] 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.

[34] 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.

[35] P. Hönig, R. Lunde, and F. Holzapfel, "Model Based Safety Analysis with smartIflow," *Information*, vol. 8, no. 1, 2017.

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

[37] 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)*, vol. 6806 of LNCS, 2011.

[38] 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*, vol. 46, no. 22, 2013.

[39] 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.

[40] 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*, vol. 98, 2011.

[41] 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.

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

[43] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson, "Safety annex for aadl repository," https://github.com/loonwerks/AMASE, 2017.