ICSEA paper outline

Danielle Stewart
University of Minnesota
Department of Computer Science and Engineering
Email: dkstewar@umn.edu

Jing (Janet) Liu
Rockwell Collins
Advanced Technology Center
Email: Jing.Liu@rockwellcollins.com

Michael W. Whalen
University of Minnesota
Department of Computer Science and Engineering
Email: whalen@umn.edu

Darren Cofer
Rockwell Collins
Advanced Technology Center
Email: Darren.Cofer@rockwellcollins.com

***Abstract*—Risk and fault analysis are important activities that help to ensure that critical software systems operate in an expected way. As critical systems become larger, the methods used to perform both formal verification and fault analysis require scalable algorithms. Compositional verification uses the idea that the correctness of a system may be determined from the correctness of its components. This has shown to be an efficient way of verifying system safety properties. A prominant artifact of fault analysis is a fault tree: a graphical representation of the failure modes of a critical system. In this paper, we describe a new approach in which fault trees can be automatically generated for a system model using compositional reasoning and formal verification techniques. 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 analysis is an important activity used to ensure that critical software systems operate in an expected way. From nuclear power plants and airplanes to heart monitors and automobiles, critical software systems are vitally important in our society. Fault Tree analysis (FTA) is a prominant technique used to graphically represent the derivation of the failure logic of a system and models how component failures propagate through a system [33]. System safety analysis techniques such as FTA are well-established and are a required activity in the development of safety-critical systems.

A Fault Tree (FT) is a directed acyclic graph whose leaves model component failures and whose gates model failure propagation. perhaps a figure here of a simple FT example? 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 *cut sets* are a way to indicate which combinations of component failures will cause the system to fail. In quantitative analysis, failure probabilities are considered for the fault trees.

In this paper, we describe a new technique for calculating minimal cut sets and constructing compositional fault trees for AADL models [1] annotated with assume-guarantee contracts [2] and extended into a fault model using the Safety Annex [37], [38].

For compositional probabilistic fault analysis, we are going to employ the online enumeration of Minimal Inductive Validity Cores (MIVCs) [18], [20] to build a fault dependency graph that shows the causal relationship between leaf level faults and the violation of requirements or safety properties at each architectural level. We will then feed that fault dependency graph to the Safe and Optimal Techniques Enabling Recovery, Integrity, and Assurance (SOTERIA) tool [35] to synthesize a fault tree with top level failure probabilities and minimal cutsets computed.

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

We provide here a brief description of AADL, AGREE, and the SOTERIA tools and languages as background for the reader.

A. Architecture Analysis and Design Language

We are using the Architectural Analysis and Design Language (AADL) [16] to construct system architecture models. AADL is an SAE International standard [1] that defines a language and provides a unifying framework for describing the system architecture for "performance-critical, embedded, real-time systems" [1]. 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.

B. Assume Guarantee Reasoning Environment

The Assume Guarantee Reasoning Environment (AGREE) [15] 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 [22] 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 [2].

C. Safety Annex for AADL

The Safety Annex for AADL [37], [38] is a tool that provides the ability to reason about faults and faulty component behaviors in AADL models. 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.

D. SOTERIA

The Safe and Optimal Techniques Enabling Recovery, Integrity, and Assurance (SOTERIA) tool [35] to perform safety analysis of Integrated Modular Avionics (IMA) systems. 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.

III. Methodology

The details of implementation and formalisms.

A. Preliminaries

1) 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. 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 [19].

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 $P : S \to \{0, 1\}$ is a state predicate. A safety property P holds on a transition system (I, T) if it holds on all reachable states. More formally, $\forall s.R(s) \Rightarrow P(s)$. When this is the case, we write $(I, T) \vdash P$. Following Ghassabani, et. al. [19], we formalize IVCs as follows.

Find the correct way to add definitions, lemmas, etc. to this format.

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

Definition 1: Minimal Inductive Validity Core: An inductive validity core S for $(I, T) \vdash P$ is minimal iff $!\exists S'.S' \subset S \ni (I, S') \vdash P$.

2) Fault Trees: 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 following gates are often used in fault trees but this list is not comprehensive.

AND Output occurs if all of the input events occur.
OR Output occurs if any of the input events occur.

There are obviously other forms of gates in fault trees (XOR, VOTING, INHIBIT, etc). I can define all of the gates that the Soteria model will be generating. That seems to make the most sense. Unless it really doesnät matter all that much. We can discuss.

To formalize a fault tree (FT), we use $GateTypes = \{And, Or, ...\}$. Following Ruijters, et. al. [34], 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.
- $T : G \rightarrow GateTypes$ is a function that describes the type of each gate.
- $I : G \rightarrow P(E)$ 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 $\pi_F : P(BE) \times E \vdash \{0, 1\}$ 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 $\pi_F(S) = 1$.

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.

Can I say finding ALL min IVCs is the same as finding ALL MCSs? I have to make sure.

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 P_{tle} be the top level event (TLE). By the definition for IVC, we know that $S \subseteq T$ is an IVC for $(I, T) \vdash P_{tle}$ iff $(I, S) \vdash P_{tle}$. In this case, we can quickly see that $\pi_F(S) = 1$, i.e. given the set $S \subseteq T$ of failed model elements, the top level event occurs. Since S is minimal IVC, it follows that $\pi_F(M) = 0$ for any $M \subset S$ and hence the set of minimal IVCs is equivalent to the set of MCSs.

IV. Case Studies

Case studies or tool comparisons.

V. Related Work

A model-based approach for safety analysis was proposed by Joshi et. al in [24]–[26]. 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) [17] and HiP-HOPS for EAST-ADL [13] 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) [7]. 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 [8], [11]. 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 [14], [30], MRMC (Markov Reward Model Checker) [27], [29], and RAT (Requirements Analysis Tool) [32]. The safety analysis tool xSAP [5] can be invoked in order to generate safety analysis artifacts such as fault trees and FMEA tables [6]. 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 [23] 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 [23]: "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) [21] 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 [14], PRISM (Probabilistic Symbolic Model Checker) [28], or the MRMC probabilistic model checker [27].

AltaRica [4], [31] 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 [9]. Failure states are defined throughout the system and flow variables are updated through the use of assertions [3]. 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 [5], [9], [12]. 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 [10].

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 [37]. To access the tool plugin, users manual, or models, see the repository [36].

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

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

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

[13] 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.

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

[15] 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.

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

[17] 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.

[18] E. Ghassabani, A. Gacek, and M. W. Whalen. Efficient generation of inductive validity cores for safety properties. *CoRR*, abs/1603.04276, 2016.

[19] E. Ghassabani, A. Gacek, and M. W. Whalen. Efficient generation of inductive validity cores for safety properties. *CoRR*, abs/1603.04276, 2016.

[20] 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.

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

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

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

[31] 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.

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

[33] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15-16:29–62, 5 2015.

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

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

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

[37] D. Stewart, J. Liu, M. Whalen, D. Cofer, and M. Peterson. Safety Annex for Architecture Analysis Design and Analysis Language. Technical Report 18-007, University of Minnesota, March 2018.

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