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*—This paper describes a new methodology with tool support for model-based safety analysis. It is implemented as a *Safety Annex* for the Architecture Analysis and Design Language (AADL). The Safety Annex provides the ability to describe faults and faulty component behaviors in AADL models. In contrast to previous AADL-based approaches, the Safety Annex leverages a formal description of the nominal system behavior to propagate faults in the system. This approach ensures consistency with the rest of the system development process and simplifies the work of safety engineers. The language for describing faults is extensible and allows safety engineers to weave various types of faults into the nominal system model. The Safety Annex supports the injection of faults into component level outputs, and the resulting behavior of the system can be analyzed using model checking through the Assume-Guarantee Reasoning Environment (AGREE).**

I. Introduction

System safety analysis techniques are well-established and are a required activity in the development of safety-critical systems. Model-based systems engineering (MBSE) methods and tools based on formal methods now permit system-level requirements to be specified and analyzed early in the development process [5], [14]. While model-based development methods are widely used in the aerospace industry, they are only recently being applied to system safety analysis.

In this paper, we describe a *Safety Annex* for the Architecture Analysis and Design Language (AADL) [15] that provides the ability to reason about faults and faulty component behaviors in AADL models. In the Safety Annex approach, we use formal assume-guarantee contracts to define the nominal behavior of system components. The nominal model is then verified using the Assume Guarantee Reasoning Environment (AGREE) [14]. The Safety Annex provides a way to weave faults into the nominal system model and analyze the behavior of the system in the presence of faults. The Safety Annex also provides a library of common fault node definitions that is customizable to the needs of system and safety engineers. Our approach adapts the work of Joshi et. al in [21] to the AADL modeling language, and provides a domain specific language for the kinds of analysis performed manually in previous work [30].

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 system failures. It can serve as the shared model to capture system design and safety-relevant information, and produce both qualitative and quantitative description of the causal relationship between faults/failures and system safety requirements. Thus, the contributions of the Safety Annex and this paper are:

- Close integration of behavioral fault analysis into the *architectural 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* through behavioral relationships in the model, in contrast to existing AADL-based annexes (HiP-HOPS, EMV2) and other related toolsets (COMPASS, Cecilia, etc.),
- additional support to capture binding relationships between hardware and software and logical and physical communications, and
- guidance on integration into a traditional safety analysis process.

II. Background

Background information on fault trees, IVCs, and Soteria.

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 [].

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. [], 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 $S \subset T$ 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. [], 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 $\pi_F : P(BE) \times E \vdash \{0, 1\}$ 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: $C \subset BE$ is a cut set of FT F if $\pi_F(C) = 1$.

Definition 6: Minimal Cut Set: $C \subset BE$ 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 $S \subset T$ 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 $S \subset T$ 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 [19]–[21]. 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) [16] and HiP-HOPS for EAST-ADL [12] 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) [6]. 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 [7], [10]. 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 [13], [25], MRMC (Markov Reward Model Checker) [22], [24], and RAT (Requirements Analysis Tool) [27]. The safety analysis tool xSAP [3] can be invoked in order to generate safety analysis artifacts such as fault trees and FMEA tables [4]. 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 [18] 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 [18]: "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) [17] 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 [13], PRISM (Probabilistic Symbolic Model Checker) [23], or the MRMC probabilistic model checker [22].

AltaRica [2], [26] 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 [8]. Failure states are defined throughout the system and flow variables are updated through the use of assertions [1]. 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 [3], [8], [11]. 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 [9].

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 [29]. To access the tool plugin, users manual, or models, see the repository [28].

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

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[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.

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

[14] 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.

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

[16] 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.

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

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

[19] 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.

[20] 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.

[22] 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.

[23] 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.

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

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

[26] 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.

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

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

[29] 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.

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