Safety Annex for AADL

Danielle Stewart[1], Janet Liu[2], Michael W. Whalen[1], and Darren Cofer[2]

[1] University of Minnesota
Department of Computer Science and Engineering
200 Union Street
Minneapolis, MN, 55455, USA
dkstewar, whalen@cs.umn.edu
[2] Rockwell Collins
Advanced Technology Center
400 Collins Rd. NE
Cedar Rapids, IA, 52498, USA
Jing.Liu, darren.cofer@rockwellcollins.com

Abstract. This paper describes a new methodology with tool support for model-based safety analysis. It is implemented as a new 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 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).

Keywords: Model-based systems engineering, fault analysis, safety engineering

1 Introduction

System safety analysis techniques are well established and are a required activity in the development of commercial aircraft and safety-critical ground systems. However, these techniques are based on informal system descriptions that are separate from the actual system design artifacts, and are highly dependent on the skill and intuition of a safety analyst. The lack of precise models of the system architecture and its failure modes often forces safety analysts to devote significant effort to gathering architectural details about the system behavior from multiple sources and embedding this information in safety artifacts, such as fault trees.

While model-based development (MBD) methods are widely used in the aerospace industry, they are generally disconnected from the safety analysis process itself. Formal model-based systems engineering (MBSE) methods and tools now permit system-level requirements to be specified and analyzed early in the development process [2, 4, 5, 13–15]. These tools can also be used to perform safety analysis based on the system architecture and initial functional decomposition. Design models from which aircraft systems are developed can be integrated into the safety analysis process to help guarantee accurate and consistent results. This integration is especially important as the amount of safety-critical hardware and software in domains such as aerospace, automotive, and medical devices has dramatically increased due to desire for greater autonomy, capability, and connectedness.

Architecture description languages, such as SysML [6] and the Architecture Analysis and Design Language (AADL) [1] are appropriate for capturing system safety information. There are several tools that currently support reasoning about faults in architecture description languages, such as the AADL error annex [12] and HiP-HOPS for EAST-ADL [3]. However, these approaches primarily use *qualitative* reasoning, in which faults are enumerated and their propagations through system components must be explicitly described. Given many possible faults, these propagation relationships become complex and it is also difficult to describe temporal properties of faults that evolve over time (e.g., leaky valve or slow divergence of sensor values). This is likewise the case with tools like SAML that incorporate both *qualitative* and *quantitative* reasoning [7]. Due to the complexity of propagation relationships, interactions may also be overlooked by the analyst and thus may not be explicitly described within the fault model.

This paper describes our initial work towards a behavioral approach to MBSA using AADL. Using assume-guarantee compositional reasoning techniques, we hope to support system safety objectives of ARP4754A and ARP4761. To make these capabilities accessible to practicing safety engineers, it is necessary to extend modeling notations to better describe failure conditions, interactions, and mitigations, and provide improvements to compositional reasoning approaches focused on the specific needs of system safety analysis. These extensions involve creating models of fault effects and weaving them into the analysis process. To a large extent, our work has been an adaptation of the work of Joshi et. al in [9–11] to the AADL modeling language.

2 Functionality

$\rightarrow$ Describe syntax and mechanics of the annex.

3 Architecture and Implementation

The architecture of the safety annex plugin is shown in Figure **??**. It is written in Java and is hosted in the OSATE AADL toolset [], which is in turn built on Eclipse []. It is not designed as a stand-alone extension of the language, but to work with existing behavioral information in the *Assume-Guarantee Reasoning Environment* (AGREE) AADL annex and tools []. AGREE allows *assume-guarantee* behavioral contracts to be added to AADL components. The language used for contract specification is based on the LUSTRE dataflow language [8]. The tool allows scaling of formal verification to large systems by splitting the analysis of a complex system architecture into a collection of verification tasks that correspond to the structure of the architecture.

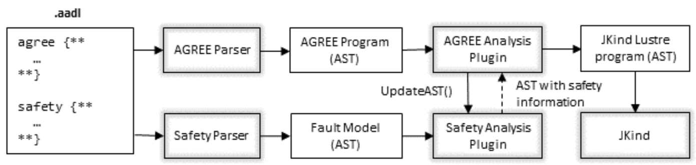

Fig. 1. Safety Annex Plug-in Architecture

AGREE contracts are normally used to define the nominal behaviors of system components as *guarantees* under *assumptions* about the values the component's environment will provide. The safety annex extends these contracts to allow faults to modify the expected behavior of component inputs and outputs. To allow for these kinds of extensions, AGREE implements an Eclipse extension point interface that allows other plug-ins to modify the generated AST prior to its submission to the solver. If the safety annex is enabled, these faults are added to the AGREE contract and, when enabled, override the normal guarantees provided by the component. An example of a portion of an initial AGREE node and its extended contract is shown in Figure 2. The `fault` variables and declarations are added to allow the contract to override the nominal behavioral constraints (provided by guarantees) on outputs. In the Lustre language, `asserts` are constraints that are assumed to hold in the transition system.

```
agree node green_pump(
  time : real
) returns (
  pressure_output : common__pressure__i
);
let
  guarantees {
    "Pump always outputs something" :
         (pressure_output.val > 0.0)
  }
tel;
```

```
agree node green_pump(
  time : real;
  __fault__nominal__pressure_output : common__pressure__i;
  fault__trigger__green_pump__fault_22 : bool;
  green_pump__fault_22__alt_value : real
) returns (
  pressure_output : common__pressure__i
);
var
  green_pump__fault_22__node__val_out : common__pressure__i;
let
  assertions {
    (green_pump__fault_22__node__val_out = pressure_output)
  }
  guarantees {
    "Pump always outputs something" :
           (__fault__nominal__pressure_output.val > 0.0)
  }
  green_pump__fault_22__node__val_out =
           faults__pressure_fail_to(
               __fault__nominal__pressure_output,
               green_pump__fault_22__alt_value,
               fault__trigger__green_pump__fault_22);
tel;
```

Fig. 2. Nominal AGREE node and its extension with faults

The top-level AADL component chosen for the analysis determines which faults are activated. We add an assertion that describes the user's fault hypothesis: either that a maximal number of faults can be active at any point in execution (often one or two), or that only faults whose probability of simultaneous occurrence is below some probability threshold. In the former case, we assert that the sum of the 'true' *fault_active* variables is under some integer threshold. In the latter, we determine all fault combinations of faults whose probabilities are above the specified probability threshold, and describe this as a proposition over *fault_active* variables.

Once augmented with safety information, the AGREE model follows the standard AGREE translation path to the model checker JKind **??**, which is an infinite-state model checker for safety properties. The augmentation includes traceability information so that when counterexamples are displayed to users, the active faults for each component are visualized.

4 Applications

$\rightarrow$ WBS, QFCS, FCC.

5 Conclusions & Future Work

Acknowledgements

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

References

1. AADL. Predictable Model-Based Engineering.
2. J. Backes, D. Cofer, S. Miller, and M. W. Whalen. Requirements Analysis of a Quad-Redundant Flight Control System. In K. Havelund, G. Holzmann, and R. Joshi, editors, *NASA Formal Methods*, volume 9058 of *Lecture Notes in Computer Science*, pages 82–96. Springer International Publishing, 2015.
3. D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lnn, and Y. Papadopoulos. Systems Modeling with EAST-ADL for Fault Tree Analysis through HiP-HOPS*. *IFAC Proceedings Volumes*, 46(22):91 – 96, 2013.
4. A. Cimatti and S. Tonetta. Contracts-Refinement Proof System for Component-Based Embedded System. *Sci. Comput. Program.*, 97:333–348, 2015.
5. D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In A. E. Goodloe and S. Person, editors, *Proceedings of the 4th NASA Formal Methods Symposium (NFM 2012)*, volume 7226, pages 126–140, Berlin, Heidelberg, April 2012. Springer-Verlag.
6. S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML.* Morgan Kaufman Pub, 2008.
7. M. Gudemann and F. Ortmeier. A framework for qualitative and quantitative formal model-based safety analysis. In *Proceedings of the 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering*, HASE '10, pages 132–141, Washington, DC, USA, 2010. IEEE Computer Society.
8. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language Lustre. In *In Proceedings of the IEEE*, volume 79(9), pages 1305–1320, 1991.
9. 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.

10. A. Joshi, S. P. Miller, M. Whalen, and M. P. Heimdahl. A Proposal for Model-Based Safety Analysis. In *In Proceedings of 24th Digital Avionics Systems Conference (Awarded Best Paper of Track)*, 2005.
11. A. Joshi, M. Whalen, and M. P. Heimdahl. Automated Safety Analysis Draft Final Report. Report for NASA Contract NCC-01001, August 2005.
12. B. Larson, J. Hatcliff, K. Fowler, and J. Delange. Illustrating the AADL Error Modeling Annex (V.2) Using a Simple Safety-critical Medical Device. In *Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '13, pages 65–84, New York, NY, USA, 2013. ACM.
13. A. Murugesan, M. W. Whalen, S. Rayadurgam, and M. P. Heimdahl. Compositional Verification of a Medical Device System. In *ACM Int'l Conf. on High Integrity Language Technology (HILT) 2013*. ACM, November 2013.
14. M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee. Model-Driven Safety Analysis of Closed-Loop Medical Systems. *Industrial Informatics, IEEE Transactions on*, PP:1–12, 2012. In early online access.
15. O. Sokolsky, I. Lee, and D. Clarke. Process-Algebraic Interpretation of AADL Models. In *Reliable Software Technologies - Ada-Europe 2009, 14th Ada-Europe International Conference, Brest, France, June 8-12, 2009. Proceedings*, pages 222–236, 2009.