Architectural Modeling and Analysis for Safety Engineering

Danielle Stewart[1], 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
darren.cofer@rockwellcollins.com

Abstract. Model-based systems engineering (MBSE) methods and tools permit system-level requirements to be specified and analyzed early in the development process of airborne and ground-based systems. These tools can also be used to perform safety analysis based on the system architecture and initial functional decomposition.
Previously, Rockwell Collins and the University of Minnesota developed and demonstrated an approach to model-based safety analysis. New MBSE tools that incorporate assume-guarantee compositional analysis techniques provide the basis for greatly improving earlier approaches to safety analysis and can be used to ensure model consistency, correctness of assumptions, and better scalability.
Using AADL-based system architecture modeling and analysis tools as an exemplar, we extend existing analysis methods to support system safety objectives of ARP4754A and ARP4761. This includes extensions to existing modeling languages to better describe failure conditions, interactions, and mitigations, and improvements to compositional reasoning approaches focused on the specific needs of system safety analysis. We develop example systems based on the Wheel Braking System in SAE AIR6110 to evaluate the effectiveness and practicality of our approach.

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. Model-based systems engineering (MBSE) methods and tools now permit system-level requirements to be specified and analyzed early in the development process. 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 is especially critical as both airborne and ground-based software for aircraft operating in the National Airspace System (NAS) continues to grow in complexity.

Previously, Rockwell Collins and the University of Minnesota developed and demonstrated an initial approach to model-based safety analysis (MBSA) (CITE THIS). New MBSE tools that incorporate assume-guarantee compositional reasoning techniques provide the basis for greatly improving earlier approaches to safety analysis, and can be used to ensure model consistency, correctness of assumptions, and better scalability.

Using our system architecture modeling and analysis tools based on the Architecture Analysis and Design Language (AADL) as an exemplar, we extend existing analysis methods to support system safety objectives of ARP4754A and ARP4761. This includes extensions to existing modeling languages to better describe failure conditions, interactions, and mitigations, and improvements to compositional reasoning approaches focused on the specific needs of system safety analysis. We develop example systems based on the Wheel Braking System model in SAE AIR6110 to evaluate the effectiveness and practicality of our approach.

The goal of the AMASE project is to develop Model-Based Safety Analysis (MBSA) methods that will strengthen safety assurance by extending current system architecture modeling techniques and exploiting recent advances in formal compositional reasoning. Our approach will allow system and safety engineers to analyze system architecture models composed from heterogeneous software components that have been annotated with formally proved behavioral contracts. These models include fault behaviors and support assessment of both nominal and faulty system behaviors.

2 Background

The proposed model-based safety analysis program builds on a strong foundation of work performed by Rockwell Collins and the University of Minnesota, including the original papers on model-based safety analysis [10–12]. Recent work in architectural modeling, compositional verification using formal methods, and system assurance are key elements of this research effort. The remainder of this section provides a brief summary of these projects and shows how they support this work.

2.1 System Modeling for Safety Analysis

Using a model-based approach for safety analysis was first proposed by the University of Minnesota and Rockwell Collins in [12]. 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. Figure 1 provides an overview of this process applied to the wheel braking system example.

The approach extends the popular model-based development (MBD) paradigm to incorporate aspects relevant to safety analysis. In model-based development, the development effort is centered on a formal specification (model) of the digital system to be produced. This model can then be subjected to various types of analysis, including completeness and consistency analysis, model checking, and theorem proving [9]. One can then automatically and correctly generate the source code implementation from this specification [7, 14]. This generative aspect of the models is very important. The model is now a central artifact of the engineering process and it becomes cost-effective to construct accurate models of the system that can be used for multiple development activities. In addition, it ensures consistency between the analysis model and the system implementation that is fielded.

While MBD normally focuses on the software components of a system, MBSA may require consideration of mechanical components of the system as well as fault models that describe the behavior of the system in the presence of one or more faults. For models to be used both for safety analysis and generation of implementations, it must be possible to consider both a nominal model (without fault models) and a fault-extended model that contains fault models for components within the system. These fault-extended models can be considerably more complex than the nominal model. This added complexity typically obscures the actual non-failure system functionality making model creation, development, inspection, and maintenance difficult. In the absence of tool-support, the incorporation of the fault behaviors must be performed manually, leading to error-prone extensions of the model with fault behavior.

In previous work [10], a behavior fault modeling language was constructed as an extension to the Lustre language [9]. Lustre is the kernel language of the popular model-based development tool SCADE [7]. In this approach, a safety engineer can model different kinds of fault behavior: e.g., stuck-at, ramp-up, ramp-down, and nondeterministic, and then ?weave? these fault models into the nominal model. The language for describing faults is extensible, allowing engineers to define a catalog of faults appropriate for their domain. In addition, the weaving process allows error propagation between unconnected components within a system model [11]. This allows consideration of physical aspects (e.g., proximity of components, shared resources such as power) that may not be present in a logical system model but can lead to dependent failures. In addition, it allows propagation of faults in the reverse direction of the model data flow. This can occur when physical components have coupling such as back-pressure in fluid systems or power surges in the opposite direction of communication through connected components. Finally, it is possible to create fault mediations to describe the output in the presence of multiple simultaneous faults.

In previous work, component-level modeling tools such as Simulink [14] and SCADE [7] were extended. In this work, we adapt this process to target system architecture models [2, 8] for safety analysis.

2.2 Safety Analysis Approach

A safety analysis system model can be used for a variety of simulations and analyses as shown in Figure 1. Modeling allows trivial exploration of *what-if* scenarios involving combinations of faults through simulations. For more rigorous analyses, static analysis tools, such as model checkers and theorem provers, can be used to automatically prove (or disprove) whether the system meets specific safety requirements.

The primary approach used for analysis in previous work was model checking. After creating the system model, a model checker was used to verify that safety properties hold on the nominal system, an idealized model of the digital controller and the mechanical system containing no faults. Once the nominal model is shown to satisfy the safety property, the behavior of the fault-extended model can be examined. In the approach described by Joshi et al. in [10–12], this analysis was performed by determining whether the property held for a given fault threshold: the maximum number of component faults to which the system is expected to be resilient.

This fault threshold is, in some sense, an approximation of the likelihood of component faults. It maps from the probabilistic *real world* potential for component failure into a non-probabilistic verification problem. Recent work [5] uses a more sophisticated approach involving minimum cut sets to describe the set of potential component failures that must be considered. Both approaches provide separation between the probabilistic aspects of the real world and the computational demands of formal analysis. This approach currently scales far better than direct use of probabilistic model checking tools such as PRISM [13], and will likely continue to do so in the future.

2.3 Compositional Verification of System Architectures

As part of the NASA Compositional Verification of Flight Critical Systems (CVFCS) project and the DARPA META and High Assurance Cyber Military Systems (HACMS) projects, we have constructed sophisticated compositional verification tools for reasoning about complex systems architectures. These tools [6] allow scaling of formal verification through splitting the analysis of a complex system architecture into a collection of verification tasks that correspond to the structure of the architecture. By decomposing the verification effort into proofs about each subsystem within the architecture, the analysis has been scaled to very large system designs [4]. In the case of the software for a complex medical drug infusion pump, a monolithic analysis of the design does not terminate in 24 hours, while the compositional approach completes in just over five minutes.

The approach naturally supports an architecture-based notion of requirements refinement based on assume-guarantee contracts. The properties of components necessary to prove a system-level property, including any assumptions about the component environment, in effect define the requirements for those components. The approach allows reuse of the verification that must already be performed on safety-critical software

components and enables distributed development by establishing the formal contracts for components that are used to assemble a system architecture. If we are able to establish a system property using only the contracts of its components, we have the means for performing virtual integration of the components. We can use the contract of each component as a specification for suppliers and have a great deal of confidence that if all the suppliers meet their specifications, the integrated system will work properly.

We have implemented this assume-guarantee mechanism for compositional verification as an extension of the AADL language derived from the safety property subset of the property specification language (PSL) [1]. The Assume-Guarantee Reasoning Environment (AGREE) is our tool for compositional verification of these contracts. Under the CVFCS project, we are currently adding automated bi-directional translation between AGREE contracts and implementation-level properties in languages such as C and Simulink. Initially, we will support translation into MATLAB properties for analysis using the Simulink Design Verifier and C assertions for analysis using source code model checkers (such as CMBC) or test- based verification. Since contracts are abstractions of component implementations, they provide a much more efficient way of representing heterogeneous components in the system model than translating the component models themselves.

3 New MBSA Capabilities

Our original MBSA work for NASA was based on MBD tools (such as Simulink and SCADE) that were available at the time [12]. However, these tools are really targeted at the design and implementation of software components, rather than at the system architecture level where most safety concerns arise.

Within the past five years there have been great advances in the capabilities of tools for modeling and analysis of at the system level, based on languages such as SysML [8] and AADL [2]. We use these new MBSE capabilities and extend them to implement the safety analysis methods needed for the design and certification of commercial aircraft systems. The system modeling tools that we plan to use are based on AADL, but they can import and export models from SysML.

- We will improve the efficiency of MBSA methods by using MBSE tools that can perform compositional reasoning over complex system models. Using the assume-guarantee contract mechanism, these tools provide support for heterogeneous component models implemented in different languages (such as Simulink or C/C++).
- Our new analysis methods move away from traditional static safety analysis methods focused on probabilistic models (e.g., Fault Tree Analysis), to the direct modeling of potential failure mechanisms and the analysis of dynamic fault-mitigation strategies.
- Formal verification of system models provides increased assurance that these models are accurate and will produce correct results. The hierarchical structure of system architecture models supports analysis at varying levels of abstraction. Compositional analysis explicitly checks assumptions captured in component and subsystem contracts. Consistency and realizability checks [23] provide the ability to detect conflicting requirements between component and subsystem models.

In this section we describe some of the new capabilities that we will develop, including improvements related to the AADL Error Model Annex, the use of model-based assurance cases, and evaluation of system-level certification objectives related to system safety that can be satisfied using the proposed methods.

3.1 Fault and Behavioral Modeling

The importance of fault modeling for system and software architectures has been recognized by the architecture language community. For example, AADL has an error model annex [15] that can be used to describe system behaviors in the presence of faults. This annex has facilities for defining error types which can be used to describe error events that indicate errors in the system. The behavior of system components in the presence of errors is determined by state machines that are attached to system components; these state machines can determine error propagations and error composition for systems created from various subcomponents.

Error types in this framework are a set of enumeration values such as NoData, BadData, LateDelivery, EarlyDelivery, TimingError, and NoService. These errors can be arranged in a hierarchy. For example, LateDelivery and EarlyDelivery are subtypes of TimingError. The errors do not have any information (other than their type) associated with them. AADL includes information on the bindings of logical components (processes, threads, systems) and their communication mechanisms onto physical resources (memories, processors, busses), and the error annex uses this information to describe how physical failures can manifest in logical components.

An example is shown in Figure 2. Errors are described by the red rectangles labeled with error types: 1-BadData, 2-NoData, 3-NoSvc. Error events that can cause a component to fail are labeled with the corresponding error number. The error behavior of components is described by their state machines. Note that while all state machines in Figure 2 have two states, they can be much more complex. The red dashed arrows indicate propagations describing how failures in one component can cause other components to fail. For example, failures in the physical layer propagate to failures in the associated logical components.

Although the error model annex is very capable, it is not closely tied to the behavioral model of components or their requirements. For example, in the wheel braking system (WBS) example [3], it is possible that hydraulic system valves can fail open or fail closed. In fail closed, downstream components receive no flow and upstream pipes may become highly pressurized as a natural consequence of the failure. Physical models of these behavioral relationships often exist that can propagate failures in terms of the behavioral relationships between components. However, with the AADL error model annex, the propagations must be (re)specified and defined for each component. The user must therefore model the system twice, specifying propagations in the models of the physical phenomena, and again using enumerations and propagation rules in state machines in the error model annex. This re-specification can lead to inconsistencies between physical models and error annex models. In addition, the physical relationships between failures can be complex and may not be describable using enumeration values, leading to additional inconsistencies between the behavior of the physical phenomenon and the behavior of the error model.

We bridge the descriptions of errors in the error model annex with behavioral descriptions of components. We start from the error model notions of error types and state machines that describe transitions from nominal to error states. However, we then tie these nominal and error states to behavioral models of the components in question that describe how the faults manifest themselves in terms of the signals or quantities produced by the components. Now the behavioral models can provide implicit propagation of the faulty behaviors and the natural consequences of failures on component behavior will be manifested in the propagation of other component faults through the behavioral model.

To accomplish this, we use AADL and the error model annex to describe faults, and to use the AGREE contract specification language to describe behavioral models. This requires extensions to AGREE to define fault models that describe how different faults manifest themselves in changes to output signals. It also requires changes to the error annex. The conditions under which faults occur will become richer such that they describe not just propagation of enumerations from other components, but also valuations of input signals. For example, very high pressure in a pipe in the WBS model may lead to a pipe burst failure.

References

1. Ieee standard for property specification language (psl), 2005.
2. AADL. Predictable model-based engineering. http://www.aadl.info.
3. AIR 6110. Contiguous aircraft/system development process example, Dec. 2011.
4. 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.
5. M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei. Efficient anytime techniques for model-based safety analysis. In *Computer Aided Verification (CAV '11*, 2015.
6. 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.
7. Esterel Technologies. Scade suite product description. http://www.estereltechnologies.com.
8. S. Friedenthal, A. Moore, and R. Steiner. *A practical Guide to SysML.* Morgan Kaufman Pub, 2008.
9. 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.
10. 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.
11. 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.
12. 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.

13. 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.
14. MathWorks. The mathworks inc. simulink product web site. http://www.mathworks.com/products/simulink, 2004.
15. SAE AS 5506B-3. Aadl annex volume 1, Sept. 2015.