Safety Annex for Architecture Analysis and Design Language

Danielle Stewart[1], Jing (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. While model based development methods are widely used in the aerospace industry, it has been a recent development that these methods are applied to safety analysis. 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, 5, 6, 14, 15, 18]. 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.

The Safety Annex for Architecture Analysis and Design Language (AADL) provides the ability to reason about faults and faulty component behaviors in AADL models. In the Safety Annex approach, we use an assume-guarantee reasoning environment (AGREE) [6] to define the contracts of system components. The nominal model is then verified using compositional reasoning techniques. The Safety Annex then provides a way to inject faults into the nominal system model and watch the behavior of the component contracts. The Safety Annex provides a library of common fault node definitions that is customizable to the needs of an analyst. These faults are then defined for each of the necessary components of the system. The fault nodes support both deterministic and nondeterministic faults on the outputs of the components.

Related Work. There are tools purpose built for safety analysis such as AltaRica [16], smartIFlow [9] and xSAP [3]. These notations are separate from the safety analysis system model. Other tools extend existing system models (HiP-HOPS [4], the AADL error annex [17]). The Safety Annex provides a way to extend the AADL system model and more easily incorporate safety analysis techniques into the model development process. The Error Annex in AADL [13] primarily uses qualitative reasoning regarding faults. The faults must be enumerated and their propagations through system components must be explicitly defined. These propagation relationships become increasingly complex proportional to the complexity of the system in question. Furthermore, complex propagation relationships may be easily overlooked by the analyst and thus may not be explicitly described within the fault model. To this end, the Safety Annex provides a quantitative reasoning environment in which to describe faults and watch their propagations through a system model.

To a large extent, our work has been an adaptation of the work of Joshi et. al in [10–12] to the AADL modeling language and an extension of the work of Stewart et. al [19].

2 Functionality

In this section, we describe the main features and functionality of the Safety Annex.

2.1 Nominal Model

An AADL model of the nominal system behavior includes mechanical and digital components and their interconnections. This nominal model is then annotated with assume-guarantee contracts using the AGREE annex for AADL [6]. Using compositional verification techniques, the nominal model behavior can be seen in the absence of faults.

2.2 Fault Model

Once the nominal model behavior is defined and verified, the Safety Annex can be used on each of the system components that may be affected by faults. The faults are defined on each of the relevant components using a predefined customizable library of

fault nodes. This extended model is then analyzed and the behavior of the system in the presence of faults can be seen. Separation of the nominal model from the fault model is handled through a selection mechanism provided to the user. The nominal model behavior can be defined and verified without also running safety analysis.

2.3 Grammar and Syntax

To illustrate the grammar and syntax of the Safety Annex, we will use an example from the Wheel Brake System described in [1] and used in our previous work [19].

The fault library contains commonly used fault node definitions. An example of a fault node is shown in Figure 1. The *fail_to* node provides a way to input a failure value and if the fault is triggered, then the nominal component output value is overridden by the *fail_to* failure value.

```
node fail_to(val_in: real, alt_val: real, trigger: bool) returns (val_out: real);
let
    val_out = if (trigger) then alt_val else val_in;
tel;
```

Fig. 1. Fault Node Definition

In the WBS, the Selector valve component receives inputs from the hydraulic pressure pumps and a digital system component. The Selector will change modes of the system depending on the inputs it receives (see [1, 19] for more information). There are two main hydraulic lines that feed into the Selector, namely the green line (normal mode of operation) and the blue line (alternate mode). The output of the Selector component will either be the green or the blue hydraulic fluid. To motivate the description of the syntax, we will go through a fault on the Selector component that describes a nondeterministic value output on the blue hydraulic line as shown in Figure 2.

```
annex safety {**
    fault "Selector: blue output failed to non-deterministic value.": faults.fail_to {
        eq alt_value :real;
        inputs: val_in <- blue_output.val,
                alt_val <- alt_value;
        outputs: blue_output.val <- val_out ;
        duration: permanent;
    }
**};
```

Fig. 2. Safety Annex Fault Definition

The *fault statement* consists of a unique description string, the fault node definition name, and a series of *fault subcomponent* statements.

Inputs: The inputs in a fault statement are the parameters into the fault node definition. As shown in Figure 2, *val_in* and *alt_val* are the two parameters into the fault node. These are linked to the values found in *blue_output.val*, the output from the Selector component, and *alt_value*, a nondeterministic value defined within the Safety Annex. When the analysis is run, these values are passed into the fault node definition.

Outputs: The outputs of the fault definition correspond to the outputs of the fault node. The fault output statement links the component output (*blue_output.val*) with the fault node output (*val_out*). If the fault is triggered, the nominal value of *blue_output.val* is overridden by the failure value output by the fault node *val_out*.

Duration: The duration of the fault in this example is permanent.

Equation Statements: Equation statements support deterministic or nondeterministic types. An example of nondeterministic equation statement is shown in Figure 2. For more details on equation statements, see [6].

3 Architecture and Implementation

The architecture of the safety annex plugin is shown in Figure 3. 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 [6]. 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.
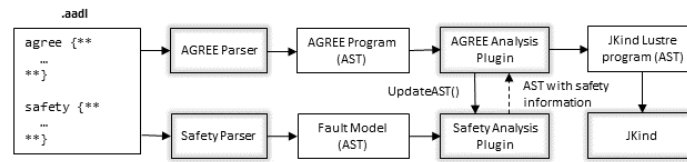

Fig. 3. Safety Annex Plug-in Architecture

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 4. 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, `assertions` 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. 4. 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

We updated the WBS model [19] by capturing components' faulty behaviors in Safety Annex. The fault nodes defined in a common fault library were invoked in components' Safety Annex. This allows the components' nominal behaviors and faulty behaviors

be modeled separately. At the top-level AADL component, the maximal number of faults that can be active at any point in execution was defined in the Safety Annex. The AGREE contracts at the top-level component were verified using AGREE, with the "Perform Safety Analysis during AGREE Verification" option selected, so the tool auto-wove the nominal and faulty behaviors before feeding to the back-end model checker.

In this example, the top level contract "Pedal pressed and no skid implies brake pressure applied" was verified valid against a system implementation when there was at most one fault active during execution, but invalid when there were more than one fault. The counterexample indicated that both Selector's outputs failed to non-deterministic values due to the faults introduced.

We also demonstrated on the Quad-Redundant Flight Control System (QFCS) model [2] that the Safety Annex can be used to introduce faulty behaviors to test existing fault response or mitigation logic: the system level properties failed when un-handled faulty behaviors were introduced.

Lastly, we explored using Safety Annex to introduce more complicated faults at the system level on a simplified QFCS model with cross-channel communications:

- Byzantine faults [7] were simulated by creating one-to-one connections from the source to multiple observers so disagreements were introduced by injecting faults to individual outputs. A system-level property failed due to the fault on the baseline model, but did not fail on the model with Byzantine fault handling protocol added. Using Safety Annex like this can test a system's vulnerability to Byzantine faults and consequent mitigation mechanisms as well.
- Hardware dependent faults were simulated by injecting faults to hardware components (physical layer) to affect their data outputs (logical layer), and consequently failing the software components bound to the hardware components. The relationship between the hardware components' outputs and the software components' inputs were specified in AGREE as part of the system's nominal behavior.

5   Conclusions & Future Work

Acknowledgements

References

1. AIR 6110. Contiguous Aircraft/System Development Process Example, Dec. 2011.
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. 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 *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 533–539, 2016.

4. 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.

5. A. Cimatti and S. Tonetta. Contracts-Refinement Proof System for Component-Based Embedded System. *Sci. Comput. Program.*, 97:333–348, 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. K. Driscoll, H. Sivencrona, and P. Zumsteg. Byzantine Fault Tolerance, from Theory to Reality. In *SAFECOMP*, volume 2788 of *LNCS*, pages 235–248, 2003.

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

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, 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.

12. A. Joshi, M. Whalen, and M. P. Heimdahl. Automated Safety Analysis Draft Final Report. Report for NASA Contract NCC-01001, August 2005.

13. 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.

14. 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.

15. 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.

16. 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 Proceedings Volumes*, 46(22):127 – 132, 2013.

17. SAE AS 5506B-3. Aadl annex volume 1, Sept. 2015.

18. 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.

19. D. Stewart, M. Whalen, D. Cofer, and M. P. Heimdahl. Architectural Modeling and Analysis for Safety Engineering. In *In Proceedings of IMBSA2017: 5th International Symposium on Model Based Safety and Assessment*, 2017.