A Comparative Study into Architecture-Based Safety Evaluation Methodologies using AADL's Error Annex and Failure Propagation Models

Lars Grunske and Jun Han

Faculty of ICT, Swinburne University of Technology

Hawthorn, VIC 3122, Australia, Tel: +61 3 9214 4758

E-mail: {lgrunske,jhan}@swin.edu.au

Abstract

Early quality evaluation and support for decisions that affect quality characteristics are among the key incentives to formally specify the architecture of a software-intensive system. The Architecture Analysis and Description Language (AADL) with its Error Annex is a new and promising architecture modeling language that supports analysis of safety and other dependability properties. This paper reviews the key concepts that are introduced by the Error Annex, and compares it to the existing safety evaluation techniques regarding its ability in providing modeling, process and tool support. Based on this review and the comparison, its strengths and weaknesses are identified and possible improvements for the model-driven safety evaluation methodology based on AADL's Error Annex are highlighted.

1 Introduction

The development of safety critical systems demands assurance that the system does not pose harm for people and the environment even if some system components fail [32, 34]. The related assurance process, known as safety analysis, consists of a risk and hazard analysis phase. The aim of risk analysis is to identify potential hazards that can occur during the system's lifetime and to determine their tolerable hazard probabilities (THP) or rates (THR) [18, 32]. A combination of a formally specified hazard condition together with its tolerable hazard probability/rate is a precondition to formulate a safety requirement. More specifically, a safety requirement is formulated as the negation of a hazard condition, combined with the hazard's THP/THR [32]. A comprehensive list of these safety requirements is the final outcome of the risk analysis process.

The set of all safety requirements, which are identified in the risk analysis process, become an input of the hazard analysis process. The aim of this process is to evaluate whether a system design meets its safety requirements. Traditionally, manual methods like Fault Tree Analysis (FTA) [26, 46] and Failure Modes and Effects Analysis (FMEA)

[25] are used to create evidence that the system fulfils its safety requirements. Besides these traditional methods, model-driven safety analysis techniques[1] have gained the increasing attention of researchers and practitioners [16]. Such model-driven approaches (applied in the architecture design phase) are used to automatically produce Fault Trees and FMEA tables based on an architecture design specification annotated with information about the failure behavior of the architectural components. Example languages for these annotations are: Failure Propagation and Transformation Notation (FPTN) [12, 13], Component Fault Trees (CFTs) [31], State Event Fault Trees (SEFTs) [20, 30], Fault Propagation and Transformation Calculus (FPTC) [48] and the Tabular Failure Annotation of the HiP HOPS methodology [39, 40, 41]. Some of these architecture-based model-driven safety evaluation techniques have been applied in industrial case studies (e.g.[23, 38]). However, most of these techniques are still only used in research contexts.

A recently proposed model-driven safety analysis technique is based on the Architecture Analysis and Description Language (AADL) [10] and its Error Annex. Due to the success of the AADL in developing embedded systems in the avionic and automotive domains, safety evaluation using AADL's Error Annex has attracted much interest and attention, with the prospect of becoming a de facto standard for model-driven safety evaluation at the architectural level. As such, it is prudent and timely to have a close examination of its overall capabilities in supporting practitioners' carrying out safety analysis.

This paper investigates the suitability of AADL's Error Annex as the basis of a model-driven safety evaluation methodology and compares it to the existing model-driven safety evaluation techniques, with the aim to identify its strengths and weaknesses and propose improvements. In fact, the comparison also identifies the strengths, weak-

---

[1] According to Lisagor et al. [33], model-driven safety analysis techniques can be distinguished into approaches that use failure logic modeling [13, 14, 20, 31, 36, 40] and fault injection experiments [4, 18, 21, 24, 28, 43]. This paper describes only approaches that use failure logic modeling.

IEEE computer society

nesses and possible improvements for the existing techniques. From the standpoint of the users or practitioners, the comparison considers the following three perspectives: modeling support, process support and tool support.

The specific contributions of this paper are two fold: (a) the paper presents and reviews the modeling concepts of AADL's Error Annex, thus giving a short introduction to the modeling of error behavior in AADL and (b) the paper compares the model-driven safety evaluation methodology based on AADL's Error Annex with the current model/driven safety evaluation methodologies.

The rest of the paper is organized as follows: Sections 2 and 3 introduce the running example of a fire alarm system and provide some background on existing model-driven architecture-based safety evaluation techniques. Section 4 introduces the modeling concepts of AADL's Error Annex. In the main part of this paper, Section 5, the safety evaluation methodologies based on AADL's Error Annex and existing safety evaluation models are compared. Their strengths, weaknesses and possible improvements are identified. Finally, Section 6 concludes the paper.

2 Running Example - Fire Alarm System

To introduce the concepts of AADL's Error Annex and to compare it with the existing failure propagation models, a fire alarm system is used as a running example. This system contains one or more smoke sensors, a software-based control unit including its executing hardware platform, a set of sprinkler actuator and a watchdog component that tests the software-based control unit at regular intervals. For simplicity reasons no bus systems or detailed hardware platform components, such as computer memory, I/O devices or hard disks, are modeled. The specific architecture of the system is depicted in Figure 1.

All the ports in this architecture are directed event ports where the initiating event comes from the environment when a fire occurs (`fire_breaks_out`). Once such a fire occurs a smoke sensor will detect the fire and the control unit will activate a set of sprinkler devices. Furthermore the watchdog component continually (e.g. every minute) initiates requests to the control unit via the `are_you_alive` port and awaits responses (`i_am_alive`) within a defined time interval (e.g. a second). If the control unit does not respond within this time interval the watchdog component resets the hardware platform via the `reset` port.

3 A Brief Review of Architecture-Based Safety Evaluation Methods

This section briefly introduces existing architecture-based safety evaluation methods. Specifically, the methods based on the modular failure propagation models [16] are investigated, since these models form the conceptual foundation for AADL's Error Annex. They include: Failure
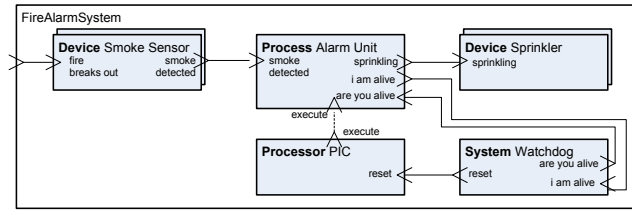

Figure 1. Fire Alarm System Architecture

Propagation and Transformation Notation (FPTN) [12, 13], Component Fault Trees (CFTs) [31], State Event Fault Trees (SEFT) [20, 30], Fault Propagation&Transformation Calculus (FPTC) [48] as well as the HiP HOPS methodology [39, 40, 41]. Since this section can only describe each approach briefly, the interested reader should consult the original literature for a detailed description of each approach.

FPTN. The Failure Propagation and Transformation Notation [12, 13] is the first modular and graphical approach for the specification of the failure behavior of architectural elements. As a result, this notation was a significant driving force for the later developed and more sophisticated models.

The Failure Propagation and Transformation Notation can be best described by an example (Figure 2). Generally, FPTN-modules contain a set of standardized sections. In the first section (the header section) an identifier (ID), a name and a safety integrity level (SIL) are specified. The second section specifies the propagation of failures, transformation of failures, generation of internal failures and detection of failures (including its probabilities). Therefore, this section enumerates all failures in the environment that can affect the component and all failures of the component that can affect the environment. These failures are denoted as incoming and outgoing failures and can be classified by a failure categorization[6] (e.g. reaction too late (tl) or too early (te), value failure (v), commission (c) and omission (o)). In the example in Figure 2, the incoming failures are `Smoke_detected:o`, `Smoke_detected:c`, and `Smoke_detected:tl` and the outgoing failures are `Sprinkling:o` and `Sprinkling:c`. The propagation and transformation of failures are specified inside the module with a set of equations or predicates (e.g. for propagation: `Sprinkling:c = Smoke_detected:c` and for transformation `Sprinkling:o= Smoke_detected:tl ||Smoke_detected:o`). These statements basically mean that for this component commission failures are directly propagated, whereas an omission failure can be caused by a too late or an omission failure of the `Smoke_detected` input. A component can also generate failures (e.g. via a `Software Fault`) or handle an existing failure (e.g.

284

via a `Safety Mechanism`). For both constructs it is necessary to specify the occurrence probabilities.
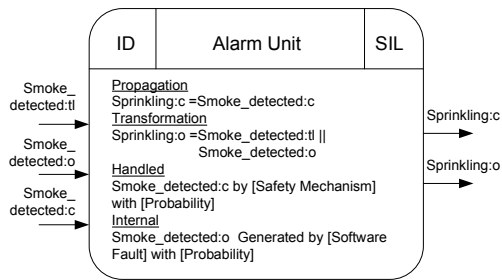

Figure 2. Example FPTN-Module

FPTN-modules can be nested hierarchically. Thus, FPTN is a hierarchical notation that allows the decomposition of the evaluation model based upon the system architecture. If a FPTN-module contains embedded FPTN-modules the incoming failures of one module can be connected with the outgoing failures of another module. Such a connection can be semantically interpreted as a failure propagation between these two modules. For the evaluation of an FPTN-module a fault tree is constructed for each outgoing failure based on the predicates specified inside the FPTN-module. As a result of this interpretation, a FPTN-module can be interpreted as a forest of fault trees, where the leaf nodes and their probabilities are extracted from the failure generation and handling section inside the FPTN-module.

HiP-HOPS. The Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) methodology [39, 40, 41] uses a textual notation that is called Tabular Failure Annotation (TFA) to specify the failure behavior of architecture elements in a set of commercial tool environments, e.g. Matlab-Simulink or Simulation X. To determine these Tabular Failure Annotations an extension of the Failure Modes and Effects Analysis [8] called Interface Focused FMEA (IF FMEA) [40] is used. Based on these annotations various analysis techniques are proposed. As an example fault trees in Fault Tree+ format [27] can be generated and can be analyzed for minimal cut sets to identify critical points of failures. Furthermore, as presented in [41], Failure Modes and Effect Analysis tables can be generated based on an analysis of the minimal cut-sets. In practice, HiP-HOPS has been successfully applied to many complex real-world systems in companies such as Volvo [38] or Daimler Chrysler [40]. An example of a tabular failure annotation for the control unit of the fire alarm system is given in Figure 3 on the next page.

CFTs. Component Fault Trees (CFTs) [31] are a modular version of traditional Fault Trees [26, 46]. Similar to traditional Fault Trees, CFTs use Fault Tree gates such as AND, OR, and M-out-of-N gates. Furthermore CFTs use input and output failure ports (graphically depicted as open triangles) and internal fault events (graphically depicted as circles). Internal fault events are similar to basic events in traditional Fault Trees and input and output failure ports are used to describe possible points for failure propagation. Figure 4 depicts a CFT for the alarm unit of the fire alarm case study.
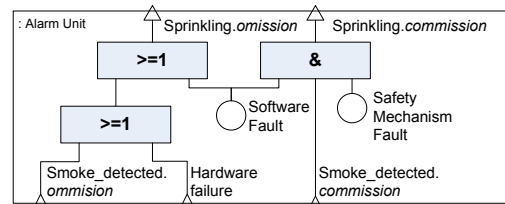

Figure 4. Component Fault Tree Example

CFTs allow the definition of a partial Fault Tree for each output failure port. These fault trees can be evaluated as a function of the input failure ports and the internal fault events. CFTs are typed and can be instantiated in different projects. To be used at an architectural level, [19] presents an approach that hierarchically constructs a system-level CFT based on an architecture specification where all architecture elements are annotated with low-level CFTs. To identify possible failure propagations between components, their possible dependencies are investigated and input and output failure ports are matched based on their names and types. The procedure for constructing hierarchical CFTs has been extended in [15] in order to analyze SaveCCM models [2].

CFTs have been used in many industrial projects (e.g. [23] reports their use at Siemens) and furthermore the windows-based tool ESSaReL [9] supports graphical specification and efficient evaluation of CFTs via probabilistic evaluation and minimal cut set analysis.

SEFTs. In contrast to the other models, State Event Fault Trees are a formalism that distinguishes states (that last over a period of time) from events (sudden phenomena, in particular state transitions). Syntactically, SEFTs are a visual formalism that extends Component Fault Trees with probabilistic finite state models [30]. As a result, an architectural element is modeled with a set of states (some of them can be seen as error states) and probabilistic transition between these states. As shown in the examples in Figure 5, states are graphically represented as rounded rectangles and events as solid bars. In SEFTs, transitions can be casually triggered by another event, exponentially distributed or deterministically delayed. Based on the distinction between states and events, novel fault tree gates can be supported, including the one used by Dynamic FTs [3].

Similar to CFTs, SEFTs can be structured hierarchically and ports are used to describe the interactions of an archi-

285

Output Failure Mode	Description	Input Deviation Logic	Component Malfunction Logic	λ (failure/ hour)
Sprinkling.c	CommissionFailure: control unit activates the sprinkler when its not needed	(Smoke_detected.c | Smoke_detected.v)	Software fault or controller malfunction	$1.0x10^-5$ $5.0x10^-7$

Figure 3. Example of a HiP-HOPS tabular failure annotation

tectural element with its environment. In addition to the standard event-based failure port, state ports can be used to probe if an architectural element is in a specific state (see the state output port of the "sprinkling" state in Figure 5). To construct a SEFT for an architecture specification where SEFTs are attached to each of its elements, a method is described in [20] that identifies inter-component relations based on name-matching of the state and event ports as well as the data and control flow specified in the architecture.
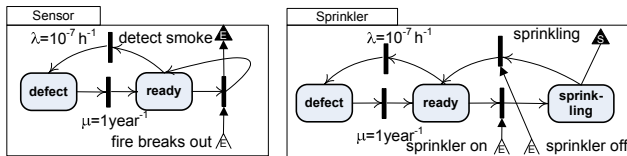

Figure 5. SEFT - Smoke Sensor and Sprinkler

FPTC. One limitation of the previously described failure propagation models is their inability to handle cyclic data- and control-flow structures in the system architecture. This is a major limitation of these techniques as many real-world control systems contain closed feedback loops. To overcome this problem, Wallace [48] has recently proposed the Fault Propagation and Transformation Calculus (FPTC). This calculus tries to solve the problem of cyclic dependencies in failure propagation models by using fixed-point evaluation techniques. However, the practicability of this calculus in industrial applications remains to be tested.

4 Safety Evaluation with AADL and its Error Annex

This section, first introduces the general concepts of the AADL and specifies the running example in AADL. Then, the specific concepts to define an annotation in terms of AADL's Error Annex are presented.

4.1 Introduction to the AADL

AADL components are specified through component types and component implementations [10]. A component type defines the component's interface in terms of inter-action points (data, event and event data ports) with other components and externally observable properties. A component type can be defined as one of the three component categories: application software, execution hardware and composite system (system). In the application software

category an AADL component can be a thread, thread group, process, data or subprogram. The execution hardware category is further refined into processor, memory, device (e.g. a sensor or actuator), and bus components.

```
system implementation FireAlarmSystem.impl
    subcomponents
        HW: processor PIC;
        AU: process AlarmUnit;
        WD: system WatchDog;
        SS: device SmokeSensor;SP: device Sprinkler;
    connections
        C0:    event port fire_breaks_out -> SS.fire_breaks_out;
        C1:    event port SS.smoke_detected -> AU.smoke_detected;
        C2:    event port AU.sprinkle -> SP.sprinkle;
        C3:    event port AU.i_am_alive -> WD.i_am_alive;
        C4:    event port WD.are_you_alive -> AU.are_you_alive;
        C5:    event port WD.reset -> HW.reset;
    properties
        Actual_Processor_Binding => reference HW applies to AU;
end FireAlarmSystem.impl;
```

Figure 6. Fire Alarm System in AADL

A component implementation defines the internal structure of a component. It may declare a set of sub-components and defines how the ports of the sub-components are connected as well as what application software is deployed on which execution hardware. Furthermore, a component implementation may define a set of operational modes of the components and transitions between these modes. Additionally, simple property sets such as security-levels, scheduling parameters, etc. and complex annotations defined in annex libraries can be attached to components types and component implementations. As an example, the component implementation of the fire alarm system as given in Section 2 is specified in Figure 6. In this specification, the AADL section on subcomponents defines the used components, and the section on connections specifies which of their input and output ports are connected. Finally, the section properties defines that the process AU is executed on the processor HW.

4.2 AADL's Errror Annex

AADL's Error Annex provides the capability to annotate AADL components with dependability related information called AADL error models. Similar to architectural AADL models, error models have two levels of description: an error type level and an error instance/implementation level.

286

An error model at the type level, defines a set of error states. These error states can also describe error free states. An error model type further defines a set of error events and its occurrence probabilities, if known. These error events can be normal error events as well as other events such as repair events. The occurrence probability of an error event can be static (defined by the language keyword `fixed`), exponentially distributed (defined by the language keyword `poisson`) or can have a user-defined non-standard distribution (`nonstandard`). Examples of interesting non-standard distributions are Weibull distributions that can be used to describe the error behavior of most software and hardware components [5]. However, despite the fact that the language definition allows user-defined distributions, this language feature has as far as the authors know not been used.

```
package FireAlarmErrorLib
public
annex error_model {**
  error model SensorErrorModel
  features
    error_free:    initial error state;
    unavailable, babbling:    error state;
    smoke_detected_omission:    out error propagation
    {Occurrence => fixed 1};
    smoke_detected_commission:    out error propagation
    {Occurrence => poisson 1E-3};
    fail_stop, fail_babble:    error event;
  end SensorErrorModel;

  error model implementation SensorErrorModel.Standard
  transitions
    error_free -[fail_stop]-> unavailable;
    error_free -[fail_babble]-> babbling;
    unavailable -[out smoke_detected_omission]->
    unavailable;
    babbling -[out smoke_detected_commission]-> babbling;
  end SensorErrorModel.Standard;
...
**};
end FireAlarmErrorLib;
```

Figure 7. Fire Alarm System error model

An error model implementation declares the error transitions between error states. These transitions can be triggered by internal error events or by error propagations from external components. (Note that most existing safety evaluation methods [12, 13, 19, 40] use the term failure propagations instead of error propagations.)

Each AADL error model can be stored in a library and can be (re)used for different AADL components. Consequently an AADL error model can describe the error behavior of a set of similar components. As examples, several different error models for simple hardware and software components have been defined in [11]. Figure 7 shows an extract of the error model annex library `FireAlarmErrorLib` for our running example. Specifically, the error model for a sensor is defined. This er-

```
device implementation SmokeSensor.simple
...
annex error_model {**
  Model => FireAlarmErrorLib::SensorErrorModel.Standard;
  Occurrence => poisson 1E-7 applies to error fail_stop;
  Occurrence => poisson 1E-7 applies to error fail_babble;
**};
end SmokeSensor.simple;
```

Figure 8. Linking the Error and AADL model

ror model contains three states, one for the correct behavior of the sensor (`error_free`) and two error states (`unavailable` and `babbling`). In the error state `unavailable` the sensor omits to send the message `smoke_detected` in case a fire occurs and in the error state `babbling` the sensor falsely sends the message `smoke_detected`. In the first case the error is propagated to the environment with a fixed probability of one, meaning that the senor will always omit to send the message. In the second case the wrong message is sent with an exponentially distributed probability with the rate 10^{-3} per second. The transitions between the error states are defined in the implementation of the error model `SensorErrorModel.Standard` (Figure 7).

As shown in Figure 8, error models from the error annex library can be annotated or associated to an architectural component `Model => FireAlarmErrorLib:: SensorErrorModel.Standard;`. When an architectural component is annotated with an error model it is possible to further tailor it with specific information such as occurrence probabilities for error events. In this example the occurrence rates for internal error events `fail_stop` and `fail_babble` have been defined. By using the same statement, already defined properties can be overwritten.

To analyze an AADL error model there are currently two approaches available. The first approach automatically translates an error model into a standard fault tree [29]. The second approach generates Generalized Stochastic Petri Nets (GSPNs) from error model specifications and uses existing GSPN tools for quantitative analysis [44].

4.3 Modeling of Error Propagation, Error Masking and Error Filtering

To construct an error model of a system component that contains subcomponents, the propagation of errors between these subcomponents needs to be defined. For the AADL error annex the error propagations are resolved via name matching of incoming and outgoing error propagations in combination with predefined abstract dependency rules between AADL components. A set of nineteen predefined dependency rules is specified in [11]. Examples of such dependencies are the error propagation from a hardware platform (processor, memory, bus, etc.) to its software com-

ponents (processes, threads, etc.) or between two software components that communicate via a connection between in-going and outgoing ports. Based on these error propagation rules, a set of inheritance rules is defined in [11] to automatically construct an error models for a hierarchical component.

An interesting and novel feature of AADL error annex models is the ability to define `Guard_In` and `Guard_Out` rules. These rules can be used to filter incoming and outgoing error propagations. Furthermore, error propagations can be renamed in order to allow a name matching between two incompatible error models. In the fire alarm system guards can be used to implement a voting strategy. For example, let us refine the architecture defined in Figure 6 and use three smoke sensors (`SS1`, `SS2`, and `SS3`) and an alarm unit with three input ports (`smoke_detected1`, `smoke_detected2` and `smoke_detected3`). In this case an incoming `smoke_detected_commission` error propagation can be masked with a "two out of three" voting strategy for the input port `smoke_detected1` of the alarm unit with the following `Guard_In` rule:

```
Guard_In => smoke_detected_commission when
  smoke_detected1[smoke_detected_commission] and
  (smoke_detected2[smoke_detected_commission] or
   smoke_detected3[smoke_detected_commission])
applies to smoke_detected1;
```

Similar rules have to be defined for the input ports `smoke_detected2` and `smoke_detected3` respectively.

5 A Comparison with Existing Failure Propagation Models

This section compares the safety evaluation methodology based on AADL's Error Annex with the existing architecture-based safety evaluation methods. This comparison is carried out based on the experience gained in modelling the fire alarm case study and a survey of the currently available documentation for each of the methods. The results presented in this section shall help safety engineers to identify the strengths and weaknesses of the approach based on AADL's Error Annex as well as other approaches, and to select and possibly combine the most appropriate architecture-based model-driven safety evaluation techniques. Furthermore recommendations are given on how to improve the safety evaluation methods.

The comparison is carried out from three perspectives, i.e., modeling support, process support, and tool support. Each perspective considers a set of relevant factors that are used to compare the different approaches. The final result of the comparisons is presented in three tables (Tables 1, 2, and 3) that are discussed in the following subsections.

5.1 Modeling Support

Modeling support concerns with the ability of system safety engineers in specifying the error/ failure behavior of an architectural element and the error/ failure propagation between dependent architectural elements. The main factor is the underlying modeling formalism for the safety evaluation models and its semantics. Furthermore, to assess advanced features the following factors have also been considered: the ability of graphically or textually specifying safety evaluation models, the ability of reusing safety annotations, the ability of modeling all relevant dependencies at the architectural and error model levels, the ability of masking, filtering and renaming error/ failure propagations and the ability of modeling interactions between operational modes and the error model. The outcome of the comparison is presented in Table 1.

An investigation of the underlying modeling concepts clearly highlights the fact that the languages are similar and have been evolved as a family of languages based on the initial work of Fenelon et al. [12, 13]. The basic aim of all these languages is to characterize the failure logic of individual components in a way that enables the automatic synthesis of complete safety evaluation models for a system built with these components [16, 33]. However differences can be found in the ability to model state-dependent behavior. The earlier models (FPTN, FPTC, HiP-HOPS, and CFT) only describe purely event-based failure behavior, whereas SEFT and AADL's Error Annex include states as well. From a semantic perspective the purely event-based models describe fault-tree-like structures characterized as simple logical formulae. The state/event-based models have a translational semantics that maps the safety evaluation model into stochastic processes. Specifically, SEFTs are mapped to Deterministic and Stochastic Petri Nets (DSPNs) [20, 30] and AADL's Error Annex is mapped to Generalized Stochastic Petri Nets (GSPNs) [44]. Details of the both versions of stochastic petri nets can be found in [35]. The main difference between the purely event-based and state/event-based models is the expressive power. State/event-based models allow modeling of failure behavior close to specification of the expected system behavior which is normally expressed with state-based specification formalisms like Statecharts [22]. However, the increased expressiveness also implies a higher computational cost when analyzing these models. Purely event-based models can still be analyzed with analytic or numerical methods even for complex models [23, 39], whereas for state/event-based models simulations are often the only means to analyze them [30, 44]. Consequently, to select an appropriate formalism for a specific project the trade-off between expressibility and analyzability has to be considered.

An investigation of the other comparison factors shows that AADL's Error Annex matches or exceeds the model-

Method	Fundamental Modeling Formalism	Graphical/Textual Modeling	Reuse of Safety Evaluation Annotations	Modeling of Architectural Dependencies	Masking, Filtering, and Renaming of Error/ Failure Propagation	Modeling of Interaction between Errors and Operational Modes
AADL+ Error Annex	State/Event-Based	Textual modeling (including XML based representation) + Graphical modeling of the architecture in AADL	Support to build Error Model Annex Libraries containing reusable standard error model types incl. inheritance [11]	Implicit definition of architectural dependencies via 19 dependency rules	Supported via the `Guard_In` and `Guard_Out` Rules [11, 44]	Supported via the `Guard_Transition` Rules [11, 44]
FPTN& FPTC	Purely Event-Based	Graphical and textual modeling	Not supported	Should be specified in the underlying architectural model	Possibility to specify detection mechanisms [12]	Not supported
HiP-HOPS	Purely Event-Based	Textual representation in the tabular failure annotations [40]	Reuse possible with failure propagation patterns [49]	Is specified in the architectural model (e.g. Matlab/Simulink)	Extra modeling for masking, filtering and renaming required	Supported only by a recent extension (PANDORA) [47]
CFT	Purely Event-Based	Graphical modeling (models are saved in a XML based representation)	CFTs are error types and can be instantiated multiple times	Should be specified in the underlying architectural model	Extra modeling for masking, filtering and renaming required	Supported only by a recent extension of CFTs [1]
SEFT	State/Event-Based	Graphical modeling (models are saved in a XML based representation)	SEFTs are error types and can be instantiated multiple times	Should be specified in the underlying architectural model	Extra modeling for masking, filtering and renaming required	Not supported

Table 1. Modeling support of architecture-based safety evaluation methods

ing capabilities of the earlier failure modeling formalisms which it builds on. In particular, the filtering and masking of error propagations via the `Guard_In` and `Guard_Out` rules in AADL's Error Annex is a powerful new modeling feature that deserves further investigation as well as inclusion into the other failure propagation models. Currently, to model this feature in the traditional failure propagation models, the filtering and masking rules have to be directly implemented in the safety evaluation model. The ability of modeling interactions between operational modes and the error model, is only supported by the AADL and its Error Annex and by CFT [1] and HiP-HOPS [47] extensions. The modeling of these interactions is an important feature for systems with multiple operational modes, e.g. aircrafts. This suggests that the traditional failure propagation models should be extended to provide this feature.

Based on the evaluation of the modeling support one can summarize that AADL's error annex provides a rich syntax that eases the specification of error annotations. However, this rich syntax could also lead to modeling errors if the semantic implications are not recognized by the safety engineer. On the other hand, a simpler formalism such as CFTs might lead to more complex annotations, but with possible less modeling errors since the safety engineer is familiar with all the modeling concepts.

5.2 Process Support

Our comparison from the process support perspective follows the steps that are needed to perform safety evaluations at an architectural level. First, the hazard conditions and safety requirements have to be identified and formally specified. Afterwards, the architecture and its dependencies have to be modeled and each architectural element has to be annotated with an error/failure model. Based on this information, an error/failure propagation model for a hierarchical component must be constructed in order to calculate the hazard probabilities with an appropriate tool.

From the comparison given in Table 2 it becomes evident that all architecture-based safety evaluation methods, including the method based on AADL's Error Annex could improve their process support. What is needed is a clear description of all the steps involved to construct and analyze the safety evaluation models. Regarding identification and specification of hazard conditions and safety requirements, only CFTs and HiP-HOPS describe some simple methodological support with the Software Hazard Analysis and Resolution in Design (SHARD) [15] and Functional Failure Analysis (FFA) [40]. Based on these two techniques the safety engineers can identify which system-level output failure will lead to a hazardous situation where it only depends on the environment if an accident occurs. Generally, these hazard conditions can be formalized as simple logical formulae in CFTs and HiP-HOPS. In SEFTs, however, a full specification of the hazard conditions in real-time and probabilistic temporal logic (e.g. with specification patterns [17]) is possible [20]. The expressive power of specifiable hazard conditions in AADL's Error Annex has to be investigated in future research, however the authors expect a similar expressiveness as for the SEFTs.

Considering the second factor, AADL's Error Annex provides good support for modeling architecture specifications including architectural dependencies. Methodological support and guideline can be found in numerous CMU/SEI technical reports and tutorial style introductions [10, 11]. A similar support is provided only by the HiP-HOPS methodology, which can use architectures defined in Matlab-Simulink or Simulation X [39]. The other modeling formalisms are general purpose failure propagation languages and consequently they do not target specific architecture description language. As a result there is also no methodological support for modeling architecture specifications and architectural dependencies.

Considering the third factor, the safety evaluation approach with AADL's Error Annex does not provide clear

Method	Identification and Specification of Hazard Conditions & Safety Requirements	Architecture Specification including Architectural Dependencies	Identification of an Error Model of a Basic Architectural Components	Generation of Error Models for Hierarchical Components
AADL+ Error Annex	Not supported, a use of SHARD and Functional Failure Analysis (FFA) seams suitable	Good support for architectural modeling including architectural dependencies in AADL [10]	Not supported, using SHARD and IF-FMEA seams suitable	Generation of hierarchical error models based on name matching of incoming and outgoing error propagations via architectural dependencies
FPTN & FPTC	Not supported	General purpose language, no specific support for modeling architecture dependecies	Not supported	Generation of hierarchical error models based on name matching of incoming and outgoing error/failure propagations
HiP-HOPS	Simple identification with Functional Failure Analysis (FFA) of the top level component in system architecture	Support for modeling architecture specification in Matlab/Simulink [39]	Based on IF-FMEA [40]	Generation of fault trees [40] based on name matching of incoming and outgoing failure propagations via architectural dependencies
CFT	Hazard conditions can be identified with SHARD and specified directly in the CFT formalism [15]	General purpose language, limited support for architectural modeling in ROOM [19] and SaveCCM [15]	Based on SHARD [15] & IF-FMEA [19]	Generation of hierarchical CFTs based on name matching of incoming and outgoing failure ports with limited support of architectural dependencies (currently communication connection only)
SEFT	No support for the identification of hazard conditions, however once identified these hazard conditions can be directly specified in the SEFT formalism [20, 30]	General purpose language, no specific architecture specification language recommended	Not supported	Generation of hierarchical SEFTs based on name matching of incoming and outgoing failure ports with limited support of architectural dependencies (currently communication connection only)

Table 2. Process support of architecture-based safety evaluation methods

Method	Tool Description	Automatic Support for the Generation of Error Models of Hierarchical Components	Probabilistic Model Analysis (Tool Back-end)	Generation of Standard Fault Trees	Generation of FMEA tables
AADL+ Error Annex	OSATE (Eclipse Plugin)	Good support for the generation of error model for hierarchical AADL component [10]	Transformation to Generalized Stochastic Petri Nets (GSPN) & evaluation of the GSPN model [44]	Generation of fault tree for acyclic models [29], or via translation to AltaRica [37] & AltaRica model checking [4]	Currently not supported
HiP-HOPS	Matlab/Simulink extension [39]	Direct generation of fault trees for the system component	Probabilistic analysis of the fault trees	Generation of fault trees (in Fault Tree+ format) for acyclic models [27]	Generation of FMEA tables base on a minimal cutset analysis [41]
CFT	UWG3 & ESSaReL (Windows-based with drag and drop GUI) [9]	Manual tool guided generation of error models for hierarchical components	Probabilistic evaluation by translation of the CFTs into BDD [31]	Automatic flattening of CFTs to standard fault trees	Currently not supported, however FMEA table generation similar to [41] should be possible
SEFT	ESSaReL (Windows-based with drag and drop GUI) [9]	Manual tool guided generation of error models for hierarchical components	Transformation to Deterministic & Stochastic Petri Nets (DSPN) and simulation with TimeNet	Currently not supported	Currently not supported

Table 3. Tool support of architecture-based safety evaluation methods

instructions on how to create an error model for a specific architectural element. However, this is generally a weakness of all other investigated approaches. Although methods like SHARD [42] and IF-FMEA [40] exist to derive failure annotations with systematic "what-if" questions, an integration of these techniques into the architecture based safety evaluation methods is not standard. Finally, no significant differences can be found for the final factor as all methods support the generation of error models for hierarchical components.

5.3 Tool Support

Provision of tool support for the model-driven safety evaluation methods may be the most important aspect for practitioners. Even a good theoretical framework or error modeling language will not be used if there is no tool support to perform the safety analysis. Additionally, if we consider complex systems, "paper and pencil"-based evaluations are often impractical and will introduce in many cases more errors than they can find in the system under examination. The factors of consideration from this perspective are: support for automatic generation of error models for hierarchical software components, back-end support that allows the calculation of hazard probabilities, and the ability to

generate fault trees and FMEA tables as they are required for certification purposes. Since factors like ease of use and ease of modeling are subjective they are not included in the comparison. Furthermore, FPTN and FPTC do not currently have tool support available and consequently they are not considered in Table 3.

Driven by industrial cooperation, the described approaches generally have good tool support. One weakness however of most architecture-based safety evaluation models is the inability to generate FMEA tables. Only HiP-HOPS provides the means to extract the necessary information of FMEA tables by analyzing minimal cutsets of the generated fault trees [41]. An adaption of this technique would be a good extension to the tool support of AADL's Error Annex and the existing failure propagation models. A further suggestion for improvement in safety evaluation with AADL's Error Annex and all other approaches is to provide the ability to generate fault trees and FMEA tables for models that contain cycles. This could be done by extending the approach described by Wallace [48]. Finally, due to their state/event based syntax, SEFTs and AADL's Error Annex could be used to derive Dynamic Fault Trees [3] from the safety evaluation models.

6 Conclusion

The paper has presented a brief introduction to error modeling with AADL's error annex. Based on this introduction, its modeling capabilities as well as its process and tool support are compared with existing safety evaluation methods. This comparison has identified the strengths and weaknesses of the approach based on AADL's Error Annex as well as other approaches, and pointed out the general and approach-specific improvements for them. This also provides the basis for safety engineers to select the most appropriate approach and associated techniques for a given safety analysis task.

From the comparison it becomes evident that AADL's error annex provides rich syntactical support for error modeling, that is similar to, if not more expressive than, the existing failure propagation models. Some examples of useful syntactic features are the ability of masking, filtering, and renaming of error/ failure propagations as well as the ability of modeling interactions between errors and operational modes. Besides, the error model's interrelation with AADL is particularly beneficial. As identified in this paper, a weak aspect is the methodological support for the identification of error models including probabilities for error events. Existing failure propagation models attempt to use techniques like IF-FMEA [40] or SHARD [15] to generate safety annotations for specific components. Even for these techniques, however, clearer guidelines would still be beneficial. Another point for improvement is the tool support for the generation of FMEA tables, and for the generation of fault trees and FMEA tables for models with cycles.

In future work this comparison could be extended to other model-driven architecture-based safety evaluation methods. Specifically, a recent approach that extends the EAST ADL (http://www.east-eea.org/) with an error modeling capability should be considered [7]. EAST ADL, which has been designed by a consortium driven by the European automotive industry, is an architecture description language similar to AADL. EAST ADL's error model has been integrated as a proof-of-concept with the HiP-HOPS method [40] in order to analyze the error model. Finally, the comparison could be extended to include other recent concepts and techniques. For example, [45] describes an approach to generate fault trees for product lines based on an AADL product line model. Such a capability is not provided by any of the existing safety evaluation models.

References

[1] R. Adler, M. Förster, and M. Trapp. Determining Configuration Probabilities of Safety-Critical Adaptive Systems. In *21st International Conference on Advanced Information Networking and Applications (AINA 2007),*, pages 548–555. IEEE Computer Society, 2007.

[2] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The SAVE approach to component-based development of vehicular systems. *Journal of Systems and Software*, 80(5):655–667, 2007.

[3] J. Bechta-Dugan, S. Bavuso, and M. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–77, Sept. 1992.

[4] P. Bieber, C. Castel, and C. Seguin. Combination of fault tree analysis and model checking for safety assessment of complex system. In *Proceedings of the 4th European Depting Conference on Dependable Computing (EDCC)*, volume 2485 of *LNCS*, pages 19–31. Springer, 2002.

[5] A. Birolini. *Reliability Engineering: Theory and Practice*. Springer, third edition, 1999.

[6] A. Bondavalli and L. Simoncini. *Failure Classification with respect to Detection*. Esprit Project Nr 3092 (PDCS: Predictably Dependable Computing Systems), 1990.

[7] P. Cuenot, D.-J. Chen, S. Gérard, H. Lönn, M.-O. Reiser, D. Servat, R. T. Kolagari, M. Törngren, and M. Weber. Towards Improving Dependability of Automotive Systems by Using the EAST-ADL Architecture Description Language. In *Architecting Dependable Systems IV*, volume 4615 of *LNCS*, pages 39–65. Springer, 2006.

[8] Department of Defence. *MIL-STD-1629A, Procedures for Performing a Failure Mode, Effects and Criticality Analysis*. Washington, 1980.

[9] ESSaRel. Embedded Systems Safety and Reliability Analyser, The ESSaRel Research Project,Homepage: http://www.essarel.de/index.html, 2005.

[10] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis and Design Language (AADL): An Introduction. Technical report, CMU/SEI-2006-TN-011, 2006.

[11] P. H. Feiler and A.-E. Rugina. Dependability Modeling with the Architecture Analysis and Design Language (AADL). Technical report, CMU/SEI-2007-TN-043, 2007.

[12] P. Fenelon, J. McDermid, M. Nicholson, and D. J. Pumfrey. Towards integrated safety analysis and design. *ACM Computing Reviews,*, 2(1):21–32, 1994.

[13] P. Fenelon and J. A. McDermid. An integrated toolset for software safety analysis. *Journal of Systems and Software*, 21(3):279–290, 1993.

[14] H. Giese and M. Tichy. Component-based hazard analysis: Optimal designs, product lines, and online-reconfiguration. In J. Górski, editor, *Proc. of the 25th Int. Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 4166 of *LNCS*, pages 156–169. Springer, 2006.

[15] L. Grunske. Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. In C. Hofmeister, I. Crnkovic, and R. Reussner, editors, *Second Int. Conf. on Quality of Software Architectures, QoSA 2006,*, volume 4214 of *LNCS*, pages 199–213. Springer, 2006.

[16] L. Grunske. Early quality prediction of component-based systems - A generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[17] L. Grunske. Specification patterns for probabilistic quality properties. In Robby, editor, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 31–40. ACM, 2008.

[18] L. Grunske, R. Colvin, and K. Winter. Probabilistic model-checking support for FMEA. In *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007)*, pages 119–128. IEEE Computer Society, 2007.

[19] L. Grunske and B. Kaiser. Automatic generation of analyzable failure propagation models from component-level failure annotations. In *Fifth International Conference on Quality Software (QSIC 2005), 19-20 September 2005, Melbourne*, pages 117–123. IEEE Computer Society, 2005.

[20] L. Grunske, B. Kaiser, and Y. Papadopoulos. Model-driven safety evaluation with state-event-based component failure annotations. In *8th Int. Symp. on Component-Based Software Engineering, CBSE 2005, Proc.*, pages 33–48, 2005.

[21] L. Grunske, P. A. Lindsay, N. Yatapanage, and K. Winter. An automated failure mode and effect analysis based on high-level design specification with behavior trees. In *5th Int. Conf. Integrated Formal Methods, IFM 2005, Proceedings*, volume 3771. Springer, 2005.

[22] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[23] R. Heilmann, S. Rothbauer, and A. Sutor. Component fault tree analysis resolves complexity: Dependability confirmation for a railway brake system. In F. Saglietti and N. Oster, editors, *SAFECOMP 2007*, volume 4680 of *LNCS*, pages 100–105. Springer, 2007.

[24] M. P. E. Heimdahl, Y. Choi, and M. W. Whalen. Deviation analysis: A new use of model checking. *Automated Software Engineering*, 12(3):321–347, 2005.

[25] IEC 60812. IEC (Intern. Elect. Commission), Functional safety of electrical/electronical/programmable electronic safety/related systems, Analysis Techniques for System Reliability - Procedure for Failure Mode and Effect Analysis (FMEA), 1991.

[26] IEC 61025. IEC (Intern. Elect. Commission) Fault-Tree-Analysis (FTA), 1990.

[27] Isograph. Isograph website (including faulttree+): http://www.isograph-software.com/, 2006.

[28] A. Joshi and M. P. E. Heimdahl. Behavioral fault modeling for model-based safety analysis. In *HASE 07*, pages 199–208. IEEE Computer Society, 2007.

[29] A. Joshi, S. Vestal, and P. Binns. Automatic Generation of Static Fault Trees from AADL Models. In *DSN Workshop on Architecting Dependable Systems*, Lecture Notes in Computer Science, page to appear. Springer, 2007.

[30] B. Kaiser, C. Gramlich, and M. Förster. State/event fault trees–a safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, 92(11):1521–1537, November 2007.

[31] B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*, pages 37–46, Adelaide, 2003.

[32] N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[33] O. Lisagor, J. A. McDermid, and D. J. Pumfrey. Towards a practicableprocess for automated safety analysis. In *Proceedings of ISSC 2006*, 2006.

[34] R. R. Lutz. Software engineering for safety: a roadmap. In *ICSE - Future of SE Track*, pages 213–226, 2000.

[35] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, New York, 1995.

[36] A. Mohamed and M. Zulkernine. On failure propagation in component-based software systems. In *Proc. of the 8th IEEE International Conference on Quality Software (QSIC'08)*, pages 402–411. IEEE Computer Society Press, 2008.

[37] K. Mokos, P. Katsaros, N. Bassiliades, V. Vasiliades, and M. Perrotin. Towards compositional safety analysis via semantic representation of component failure behaviour. In *Proc. of the 8th Joint Conference on Knowledge - Based Software Engineering (JCKBSE 08)*, page (to appear). IOS Press, 2008.

[38] Y. Papadopoulos and C. Grante. Evolving car designs using model-based automated safety analysis and optimisation techniques. *Journal of Systems and Software*, 76(1):77–89, 2005.

[39] Y. Papadopoulos and M. Maruhn. Model-based synthesis of fault trees from matlab-simulink models. In *2001 International Conference on Dependable Systems and Networks (DSN 2001)*, pages 77–82. IEEE Computer Society, 2001.

[40] Y. Papadopoulos, J. A. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Int. Journal of Reliability Engineering and System Safety*, 71(3):229–247, 2001.

[41] Y. Papadopoulos, D. Parker, and C. Grante. Automating the failure modes and effects analysis of safety critical systems. In *Int. Symp. on High-Assurance Systems Engineering (HASE 2004)*, pages 310–311. IEEE Comp. Society, 2004.

[42] D. J. Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, Department of Computer Science, University of York, 1999.

[43] J. D. Reese and N. G. Leveson. Software deviation analysis. In *Proceedings of the 19th International Conference on Software Engineering*, pages 250–261. ACM Press, 1997.

[44] A.-E. Rugina, K. Kanoun, and M. Kaâniche. A System Dependability Modeling Framework Using AADL and GSPNs. In *Architecting Dependable Systems IV*, volume 4615 of *LNCS*, pages 14–38. Springer, 2007.

[45] H. Sun, M. Hauptman, and R. R. Lutz. Integrating Product-Line Fault Tree Analysis into AADL Models. In *Tenth IEEE Int. Symp. on High Assurance Systems Engineering (HASE 2007)*, pages 15–22. IEEE Computer Society, 2007.

[46] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault Tree Handbook. U. S. Nuclear Regulatory Commission, 1996.

[47] M. Walker, L. Bottaci, and Y. Papadopoulos. Compositional temporal fault tree analysis. In F. Saglietti and N. Oster, editors, *26th International Conference on Computer Safety, Reliability, and Security SAFECOMP 2007*, volume 4680 of *LNCS*, pages 106–119. Springer, 2007.

[48] M. Wallace. Modular architectural representation and analysis of fault propagation and transformation. *Electr. Notes Theor. Comput. Sci.*, 141(3):53–71, 2005.

[49] I. Wolforth, M. Walker, and Y. Papadopoulos. A language for failure patterns and application in safety analysis. In *IEEE Conference on Dependable Computing Systems (DEPCOS08)*. IEEE Computer Society, 2008.