Integrating Product-Line Fault Tree Analysis into AADL Models

Hongyu Sun¹, Miriam Hauptman¹, Robyn Lutz²

¹Department of Computer Science, Iowa State University
²Jet Propulsion Lab/Caltech
{sun, miriamh, rlutz}@cs.iastate.edu

Abstract

Fault Tree Analysis (FTA) is a safety-analysis technique that has been recently extended to accommodate product-line engineering for critical domains. This paper describes a tool-supported approach to integrate product-line FTA into AADL (Architecture Analysis and Design Language) models and associated AADL Error Models of a product line. With concrete models bound, architectural stage FTA allows automation in the fault tree generation and the analyses, which was impossible at requirement stage. A fault tree for a specific product is automatically pruned and adapted from the product-line FTA, which reduces effort and enhances consistency. The AADL Error-Annex-compatible format allows automated derivation of basic quantitative and cut set analyses for each product-line member to help identify and eliminate design weaknesses. The tool-supported capabilities described here enable comparisons among candidate new members to assist in design decisions regarding redundancy, safety features, and the evaluation of alternative designs. We evaluate these claims by an application to a product-line case study.

1. Introduction

Product-line engineering has been shown to improve product quality, reduce development cost, and provide improved time-to-market through systematic reuse of product-line assets, including shared requirements, architecture, code, and test suites [3]. A product-line is a set of systems developed by a single company that shares a common set of core requirements [21]. Product-line engineering (PLE) typically consists of two phases: domain engineering and application engineering [21]. During the domain engineering phase, developers identify the requirements shared among all the systems in the product line (commonalities) as well as the distinct requirements (variabilities) imposed on some of the systems. During the application engineering phase, each new product in the product line is built using the product-line assets previously produced. Some product lines are safety-critical, e.g., cardiac pacemakers, pilot cockpit displays, health-imaging systems, orbiting satellites, and weather-alert monitoring stations [5]. Here we are primarily interested in such product lines.

In our previous work we have shown how to perform a fault tree analysis for a product line and how this analysis can be adapted and reused when each new system is built in the product line [6, 7]. Fault Tree Analysis (FTA) was thus extended into Product Line Fault Tree Analysis (PLFT) to accommodate product-line engineering for safety critical domains [6]. To date, work on PLFT has mainly focused on safety analysis during the requirements phase of safety-critical systems.

An on-going problem in the product-line engineering of safety-critical software systems is how to capture and maintain the results of the safety analysis in the subsequent design architecture modeling effort. Integrating product-line safety analyses with Architecture Analysis & Design Language (AADL) architectural models [20] does not only extend the scope of early safety analyses into the architecture stage of development, but also enhances the usability of AADL on the development of the critical product lines. From previous experience, a report from a 2004 software product line workshop stated: “While primarily a design language, the AADL incorporates many concepts that are critical to successful product line deployment” [2]. Clegg et al. similarly note that an AADL-analyzable architectural model can help assure that critical non-functional requirements are
satisfied by the product-line architectural design and may help uncover additional potential hazards [4]. Feiler has provided a brief introduction to modeling families of systems in AADL [10]. We can see AADL has sufficient fundamental support for critical product line developments and is suitable for integrating more early analyses.

This paper describes a tool-supported approach to integrate product-line FTA into AADL models and associated AADL Error Models for a product line. In the domain engineering phase of our approach, an architectural stage PLFT is derived from the requirements stage PLFT by associating the PLFT with AADL system models, error models and commonality/variability properties. In the application phase, the fault tree for each product is automatically derived from the PLFT by pruning and adapting it according to the configuration of variabilities and local instance parameters. Here, a configuration of variability is defined as the variable requirements implemented into product line core assets in the form of variation points [12].

This work provides a tool-supported process for product-line fault tree analysis on a safety-critical product line during the architectural design phase. The architecture model is constructed in AADL format. The tool, a PLFT Eclipse plug-in, was developed for AADL in the Open Source AADL Tool Environment (OSATE) [11]. The AADL-PLFT plug-in contains three main features:

1) **Domain Engineering: PLFT Construction.** This feature partially automates the construction of the PLFT. It binds the fault tree with AADL system models, error models, and product-line requirements (commonalities and variations). The integration of the product-line fault tree with the AADL models enables fault-tree-based analyses of the models.

2) **Application Engineering: PLFT Management.** This feature uses the AADL-PLFT plug-in to load the PLFTs into an AADL annex for safety analysis on specific failure events in the system. This feature manages different types and versions of the PLFTs for each member in the product line.

3) **Application Engineering: PLFT Reuse.** This feature of the AADL-PLFT provides the safety analysis of the architectural design when a new member is built. It contains four functions: pruning a PLFT, adapting a PLFT, basic quantitative analysis and cut set analysis [15].

With concrete AADL system models and error models bound, architectural stage FTA allows automation in the fault tree generation from the PLFT and the analyses, which was impossible at requirement stage. A fault tree for a specific product is automatically pruned and adapted from the product-line FTA, which reduces effort and enhances consistency. The AADL Error-Annex-compatible format allows automated derivation of basic quantitative and cut set analyses for each product-line member to help identify and eliminate design weaknesses. The tool-supported capabilities described here enable comparisons among candidate new members to assist in design decisions regarding redundancy, safety features, and the evaluation of alternative designs. We evaluate these claims by an application to a product-line case study.

The remainder of the paper is organized as follows. Section 2 describes related work. Section 3 presents an overview of the approach and the AADL-PLFT plug-in and discusses its functionalities. Section 4 describes a case study application to a safety-critical product line, the Floating Weather Station (FWS) [21], and discusses how the analysis results can be used to evaluate safety-related aspects of the product line modeled in AADL. Section 5 provides concluding remarks.

### 2. Related Work

This section briefly describes related work in product-line fault tree analysis (PLFT) and the architectural analysis and design language (AADL). The reader is referred to [21] for additional background material on product line engineering.

#### 2.1. Product Line Fault Tree Analysis

FTA is a standard safety analysis technique to investigate the contributing causes of the hazard described in the root node of the fault tree [15]. It includes the following aspects:

1) **System Boundary Definition**
2) **Fault Tree Construction** [15]
3) **Qualitative Analysis.** This typically involves Cut Set Analysis [15] and other analyses related to the quality of the system.
4) **Quantitative Analysis.** This typically involves failure probability assessment and other analyses to help quantify the likelihood of the root node occurring. These analyses help the engineers to evaluate the safety of the design and perform tradeoff analyses among design options. Failure
probability assessment, although difficult, may be able to identify potentially troublesome software components for further development and testing [13]. For example, Goseva-Popstojanova et al. have shown how using dynamic models, such as Markov chain based models, to model the probability of a failure rather than using fixed values can reduce the uncertainty [13].

Because the quality of the FTA depends heavily on the expertise and experience of the analyst, constructing a fault tree is labor-intensive and time-consuming. Partially automated reuse of a product-line fault tree analysis across the members can significantly reduce the validation cost on the new members.

To enable the use of fault trees in product-line engineering, Dehlinger and Lutz introduced product-line fault tree analysis (PLFTA) [6]. This reduces the effort and cost to construct the fault trees for each new member in the product line. In PLFT, each leaf node is associated with a commonality or variability. The fault tree for each product in a product line can be derived from the product-line fault tree according to the commonality/variability attributes. Moreover, since the data in the PLFT have been reused repeatedly in the product line, they tend to be better verified and more accurate than a fault tree for a single system. A tool called PLFaultCat for PLFT helps in construction and analysis during the requirement development stage [6]. When new, unanticipated requirements occur, PLFaultCat provides tool support to assist the developer in updating the fault trees and propagating the changes into future products.

However, the abstractness of the fault tree at the requirement stage limits its ability on the analysis and the automation. AADL-PLFT plug-in takes the advantage of the concrete system models at the architectural design stage to utilize automated fault tree generation and analyses. The AADL and PLFT marriage also enables more analyses on design decisions.

2.2. Architecture Analysis & Design Language

Architecture Analysis & Design Language (AADL) is used for the specification, analysis and automated integration of real-time, critical systems [20]. It has been used to predict and verify critical properties in several case studies, including Schedulability Analysis on Complex Avionics System (CAS) [1], Dependability Analysis [17], a Generalized Stochastic Petri Net (GSPN) model derived automatically from the AADL model, Mode-Driven Fault Tolerance (MDFT) Design [19], AADL, HOOD and UML2.0 Integration for mission critical systems [8], Security Level Analysis and Memory/Buffer Analysis [14] on a queuing system, Safety Level and Reliability Analysis [14] using the MetaH tool partitioning analyzer to partially verify the consistency of the safety levels, and Deadline, Priority Check based on Rate-Monotonic Analysis (RMA) [9]. Additional information on constructing an AADL model can be found in [20]. Feiler describes the modeling of product lines or families in AADL in [10] and provides details on modeling component variations in AADL. However AADL does not provide sufficient early analyses on safety properties for the critical systems. The introduction of PLFT into the AADL improves the utilization of AADL in the safety critical domains.

3. Approach

This section introduces the features of the AADL-PLFT plug-in and describes how to utilize them. It then gives an overview of the safety analysis approach with the AADL-PLFT plug-in.

3.1. AADL-PLFT Plug-in Overview

The AADL-PLFT Plug-in depends on the AADL and the AADL Error Model Annex. The AADL Error Model Annex is a standardized extension to the core AADL language standard and supports fault modeling in AADL [18]. Each component implementation is associated with one error model. The AADL-PLFT plug-in is developed for AADL in the Open Source AADL Tool Environment (OSATE) [11] in Eclipse. The plug-in contains three main features (Figure 1).

![Figure 1. Three features of AADL PLFT Plug-in](image)

3.2. Domain Engineering: PLFT Construction Feature

Figure 2 shows an overview of the PLFT construction. Examples of each of these elements are provided in the case study in section 4. The following
steps show how to construct a PLFT for the AADL model:
1) The requirements–phase PLFT XML file is an output of PLFaultCat. The AADL-PLFT plug-in can read an XML PLFT file generated from PLFaultCat [6]. The AADL-PLFT plug-in automatically enumerates all components in the system, all failure events in the error model and all the commonalities/variabilities (CVs).
2) To bind the fault tree to an analyzable error model, the designer constructs an AADL error model for the system, and binds the nodes in the fault trees to the specific error model. In addition, some special error models as environment and operator error models may need to be constructed. The PLFT will be simulated in the AADL Annex with the probability based on the error model event’s frequency or its probability when the planned simulation capabilities in AADL become available. The analyst can also assign a multi-instance relation tag to any fault tree node. This multi-instance tag can expand the fault tree node if more than one implementation exists for a component.
3) The construction of the commonality/variability (CV) file translates the text-based commonality and variability requirements into an AADL-compatible format and adds commonality/variability properties to the AADL system models.

**Figure 2. AADL-PLFT Plug-in PLFT Construction Feature Overview**

After an AADL PLFT is constructed, the PLFT can be written into a PLFaultCat compatible XML file with extra tags. Thus, if any inaccuracy in the description or logic of an AADL PLFT is found, the analysts can use PLFaultCat to “recycle” this PLFT. The compatibility between PLFaultCat and the AADL-PLFT makes this process adaptable to change. The resulting AADL-PLFT can be read and edited by PLFaultCat. It can also be loaded into the AADL Error Annex for analysis.

### 3.3. Application Engineering: PLFT Management Feature

The AADL-PLFT plug-in can load any specific PLFTs into the AADL annex in order to perform safety analysis on the specific failure events of the critical system. This feature allows exploration and comparison of the effects among different failures and different versions of PLFTs on the product-line members’ design. Comparison between different failures can help reveal hidden tradeoffs that may exist between two failures or unanticipated coupling in the system.

### 3.4. Application Engineering: PLFT Reuse

F1~F4 (Figure 3) represent the four analysis feature, which will be explained in details later. R1 ~R4 are the results from the four features.

**Figure 3. AADL PLFT Plug-in PLFT Analysis feature overview**

#### 3.4.1. Pruning the PLFT

The plug-in reads the CV (Commonalities/Variabilities) requirements for the current AADL product model, and prunes the PLFT according to what the requirements are for this particular system in the product line. Since each product in a product line is a composition of CVs, after pruning the PLFT appropriately, the plug-in can produce the fault tree for the current product. A leaf node in the AADL PLFT can be expressed as Com (F, CV, M), where Com is a reference to an AADL component-type model, F refers to a Failure Event generated by the error model of the component, CV is a Boolean value indicating whether the
Commonality or Variability associated with this component exists in the current product, and M is a multi-instance tag value. M captures the relation among the redundant units of this type in the product. Its value indicates whether all redundant units must fail before a failure is considered to have occurred (“AND”) or whether a failure is considered to have occurred when any unit fails (“OR”).

The pruning process for a new member of the product line can then be described formally as:

\[
\text{If } CV = \text{True, Then } \text{Com} (F, CV, M) \rightarrow \text{Com} (F, M) \\
\text{If } CV = \text{False, Then } \text{Com} (F, CV, M) \rightarrow \text{False}.
\]

For example, if a feature is only available to customers in some systems in the product line, potential failures in that feature will be pruned out if the current system does not have this feature.

3.4.2. Adapting the PLFT. The PLFT plug-in reads in the implementations for all the component types, and adapts the PLFT according to the implementations.

A Component type can be an abstract description (e.g., Sensor). Its implementations can contain more details (e.g., Wind Sensor, Temperature Sensor). In adaptation, the PLFT plug-in updates the component type bindings in the product-line fault tree with their implementations.

The plug-in also adapts the PLFT according to its multi-instance relation tags. If one component type has more than one implementation or instance, the Boolean logic connecting those instances will be constructed according to its Multi-Instance Tag.

Thus, if \( \text{Impl} = \{\text{Impl0}, \text{Impl1} \ldots\} \) are the implementations for the component Com, F is a Failure Event generated by the error model of the component, and M is a multi-instance tag whose value is “AND” or “OR” as defined above, then the PLFT adaptation can be defined as:

\[
\text{If } |\text{Impl}| = 1, \text{ then } \text{Com} (F, M) \rightarrow \text{Impl0} (F) \\
\text{If } |\text{Impl}| > 1, \\
\text{If } M = \text{“AND”, then } \text{Com} (F, M) \rightarrow \text{Impl0} (F) \text{ AND Impl1} (F) \ldots \\
\text{If } M = \text{“OR”, then } \text{Com} (F, M) \rightarrow \text{Impl0} (F) \text{ OR Impl1} (F) \ldots
\]

3.4.3. Analyzing the PLFT. The AADL-PLFT Plug-in provides analysis abilities to investigate the impact of the design decisions on the safety properties of the system. For fault trees whose leaf nodes have been labeled with failure probabilities (e.g., in the tree imported from PLFaultCat), a quantitative analysis then calculates the probability of the fault tree’s root node failure event. In the PLFT approach, this quantitative analysis can be used to explore possible alternative variabilities (i.e., features) of envisioned new products in the product line. By comparing failure probabilities among several configurations, the designers can improve their understanding of how the choices, duplications or tradeoffs, affect the occurrence of the top events in the fault trees.

The plug-in also provides a cut set analysis capability to show the set of minimum cut sets of the fault tree. The minimum cut sets are the smallest combinations of leaf node events that will trigger the root-node failure. An example of a minimum cut set is: \( \{\text{ImplA} (F0), \text{ImplB} (F1), \text{ImplC} (F2)\} \). If any failure in this set can be prevented, the root node in the tree will not occur.

The minimum cut sets resulting from a cut set analysis can be too numerous to analyze manually. For example, in the example product line in this case study, the number of minimum cut sets were as high as 1029 for product line fault tree. The AADL PLFT plug-in can identify those failure events exist in many minimum cut sets. This feature is useful because, by eliminating them, a large number of minimum cut sets are eliminated. If a basic event is eliminated, e.g., by introducing a new algorithm, changing the error model’s internal logic or using alternative component implementations, the PLFT is also changed, and the PLFT plug-in analysis must be run again. When elimination is impossible, reducing the failure rate of those single events will be the most efficient way to mitigate the probability of the root-node hazard from occurring.

This section has described an approach to construct, manage and reuse the PLFT for an AADL architectural model using an AADL-PLFT plug-in. Analysis results show how the architectural configuration of a system can affect safety properties and how the PLFT-driven analysis can provide insights into alternatives for achieving a safer configuration with respect to a particular PLFT failure event. An AADL Error Annex compatible fault tree format will also allow the AADL Error Annex to simulate the fault tree within the error models when the dynamic simulation capability has been released by the AADL developers [18].

4. Case Study

This section describes an application of the integrated Product Line Fault Tree Analysis with the
AADL system and error models to a small case study. The Floating Weather Station (FWS) is a product line described by Weiss and Lai in [21]. It has been used to illustrate and, to some extent, to benchmark analysis techniques in previous publications [6, 7, 21]. Each FWS in the product line is a buoy floating in the sea with sensors that monitor weather conditions. The most common sensors for a FWS are an anemometer (wind speed sensor) and a thermometer. The FWS’s job is to collect weather information from sensors, store them in the database and periodically send reports through a radio transmitter.

All FWS in the product line share certain required features, or commonalities, such as monitoring wind speed and transmitting data. However, there are some variabilities in the product line so that the FWS can vary according to customers’ needs. For example, some have extra features to monitor ocean wave spectra or wind direction. Some FWS are also equipped with an emergency switch for nearby sea accidents so that a sailor able to reach a FWS can summon help [6].

The weather information from the FWS is required to be highly available and accurate. The FWS is a safety-critical product since for navigation and weather purposes, missing or incorrect weather data can contribute to accidents, including the loss of human life and property.

The architecture stage development of the FWS project requires the decision model and the PLFT from its requirement stage development. The workflow is shown in Figure 4 and illustrated in details in subsections from 4.1 to 4.4.

The architecture stage development of the FWS project requires the decision model and the PLFT from its requirement stage development. The workflow is shown in Figure 4 and illustrated in details in subsections from 4.1 to 4.4.

4.1. CV file construction

To construct the commonality and variability file (CV file) in the AADL format, we used the existing text-based specification of the FWS product-line commonalities and variabilities from the requirements phase provided in [6].

We represented the CV product-line requirements as a property set in AADL. Each commonality or variability property was associated with one or more Boolean values. The default value for commonalities (e.g., every FWS has a wind sensor) is set to true. For alternative variabilities (e.g., high-resolution sensor or low-resolution sensor), the default choice is set to true and the other choices are set to false. The default value for all other variabilities (e.g., the SOS feature) is set to false. An excerpt of the FWS CV file shows the format and default values:

```
property set CVAnalysisResults is
    --- Commonalities
    c1 the fws shall represent wind speed measurements in knots
    co_speedUnitKnots => true;
    --------- Variabilities
    v1 the fws may represent wind speed measurements in KPH and MPH besides knots
    va_speedUnitMPH_KPH => true;
end CVAnalysisResults;
```

4.2. AADL system model construction

An AADL system model has two kinds of models. One is the type model, which represents the functional interfaces of the component. It describes the external view of the system. The other is the implementation model, which contains the detailed contents of the component in terms of subcomponents, properties, etc [20]. It describes the internal view of the system. Note that, a type model can have more than one implementation model, which means that several implementations can share the same external interfaces.

For a product line, commonalities can be modeled as a system implementation model with all commonalities inside, called a basic configuration. All the other system implementations for members of the product line need to inherit this basic configuration model to implement the commonalities. The variabilities are then configured within these implementations.

In the FWS, one variant point is the input sensor type and number. According to the CV analysis, all FWS should have a wind speed sensor. Future products in the FWS product line can also have wind direction and temperature sensors, or a combination of the two. The wind direction and temperature sensors are optional (Boolean-valued) variabilities. Different kinds of sensors can inherit a general sensor type model. The
following excerpt from the sensor model shows how
the wind direction sensor (commonality) and the wind
speed sensor (variability) both extend the general
Sensor.

```
device Sensor
  features
  plug: requires bus access Wire;
  output: out data port Raw_Data;
end Sensor;
```

The wind speed sensor was modeled as:

```
device WindSpeedSensor
  extends Sensor
end WindSpeedSensor;
```

The other sensor was modeled as:

```
device WindDirectionSensor
  extends Sensor
end WindDirectionSensor;
```

Each sensor could also be either high-resolution or
low-resolution:

```
device implementation WindSpeedSensor.HighRes
  end WindSpeedSensor.HighRes;

device implementation WindSpeedSensor.LowRes
  end WindSpeedSensor.LowRes;
```

The number of the sensors, as well as other
components, can be configured in the system
implementation model which inherits all the
commonalities.

If the product line evolves in unanticipated ways,
new variabilities (i.e., new requirements) may need to
be added to the product line. In this case, components
can be added to (or, conversely, removed from) the
implementation model without affecting the external
interfaces (the system type model) and the
commonalities (basic configuration implementation).
If the external interfaces change, the type model must
be also updated.

4.3. Error models construction

Next, the FWS error models were bound with the
basic events (leaf nodes) in the product line fault tree.
To achieve this, we constructed an error model and one
or more error model implementations for each
component in the AADL system model.

In safety analysis of product lines (such as the FWS),
environmental and human effects are often considered.
Thus, some fault tree leaf node failures referred to
environmental events or operator actions. To enable the
binding between error models and fault tree leaf nodes,
we took advantage of the Markov-chain-based error
model to construct environmental and operational
models in the AADL error model annex.

A simple example of storm weather environment
model was constructed for the FWS use. Its
Corresponding Markov chain for the error model is
illustrated in Figure 5. The white bar represents the
fixed frequency and the black bar represents the
probability generated by the simulation function. When
the model is in the Storm state, it will output InStorm
event to the upper level error model.

```
Figure 5. Background Markov Chain for storm
error model
```

4.4. AADL-PLFT construction

There are two ways to build product-line fault trees
in the AADL-PLFT plug-in format: to use PLFaultCat
as a front-end or to directly construct the PLFT in the
AADL error model annex of the system model in the
PLFT plug-in format. Manual construction of fault
trees for large product lines tends to be impractical and
error-prone. The tool support provided by PLFaultCat
helped generate the product-line fault tree for the FWS
[6]. The product-line fault tree for the FWS was thus
saved by PLFaultCat as an XML file.

The PLFT plug-in took this XML file, the AADL
models constructed for the FWS (the AADL system
type, implementation models and the AADL error
models) and the CV file describing the FWS product-
line commonalities and variabilities as input. It
enumerates a list of the components in the AADL
system model, a list of the output events in the error
models and a list of the commonalities and variabilities
in the CV file. Then we manually selected the proper
combinations to bind with the fault tree leaf nodes. We
also assigned a multi-instance relation tag to each leaf
node, “AND” or “OR”, in cases more than one
implementation of the same component exists. The sequence is shown as a diagram in Figure 6.

**Figure 6.** Construction Sequence of PLFT with AADL-PLFT plug-in

### 4.5. Safety Analysis with PLFT

We can select and load the Incorrect_Wind_Speed PLFT into the project and start to analyze the failure that reports the incorrect wind speed. After confirming the modification by the PLFT plug-in, the system implementation model error annex has been updated with the desired PLFT in it. The PLFT analysis feature will prune and adapt the PLFT into the fault tree of the current product.

---Product Line Fault Tree Excerpt---
PLGuard_Out=>Incorrect_Wind_Speed_Reported when (((Message_Generator[Failed_To_Convert][CVAnalysisResults:va_speedUnitMPH_KPH][OR VA][0.00004] or Message_Generator[Failed_To_Convert][CVAnalysisResults:va_speedUnitMPH_KPH][OR VA][0.00004]) or (StackManagement[Stack_Initialization_Failed][CVAnalysisResults:va_stackDataAsInteger][OR VA][0.00004]) or ...

---Fault Tree Excerpt For Current Product Implementation---
Guard_Out=>Incorrect_Wind_Speed_Reported when (Transmitter_Driver[Stack_Initialization_Failed] or ...)

**Figure 7.** Product Line Fault Tree and Fault Tree in AADL Model

By comparing the PLFT and the fault tree for current implementation (Figure 7), we can see that the fault tree has been pruned according to the CV properties and that each binding component of the fault tree has been substituted by its implementation(s). Next, preliminary fault tree analysis results were obtained (Figure 8). The details of all minimum cut sets are not shown in Figure 8 but exist in the annex.

---Quantitative Analysis---
- Probability of Failure is: 1.1477376E-15
- Qualitative Analysis---
- Min Cuts as following:
  - Number of Min Cut Sets is: 264
  - Min Cut Set Cardinality Range: 4 .. 5
  - Ranking of Top10 Bottle Neck Failures in Min Cut Sets:
  - App: Appearance in all min cut sets
  - Aff: Min Cut Sets Affected by this failure

<table>
<thead>
<tr>
<th>Times of App</th>
<th>App %</th>
<th>Num of App</th>
<th>AFF %</th>
<th>Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>288</td>
<td>109</td>
<td>264</td>
<td>100</td>
<td>WFS_Controller[Stack Initialization Failed]</td>
</tr>
<tr>
<td>68</td>
<td>25%</td>
<td>64</td>
<td>24%</td>
<td>aStack[Stack Initialization Failed]</td>
</tr>
<tr>
<td>68</td>
<td>25%</td>
<td>64</td>
<td>24%</td>
<td>aDataBanker[Memory Failed]</td>
</tr>
<tr>
<td>66</td>
<td>25%</td>
<td>66</td>
<td>25%</td>
<td>aMessage_Generator[Format Failed To Convert]</td>
</tr>
<tr>
<td>66</td>
<td>25%</td>
<td>66</td>
<td>25%</td>
<td>Transmitter_Driver[Timer Failed]</td>
</tr>
<tr>
<td>66</td>
<td>25%</td>
<td>66</td>
<td>25%</td>
<td>aDataBanker[Format Failed To Convert]</td>
</tr>
<tr>
<td>66</td>
<td>25%</td>
<td>66</td>
<td>25%</td>
<td>Transmitter_Driver[Stack Initialization Failed]</td>
</tr>
<tr>
<td>66</td>
<td>25%</td>
<td>66</td>
<td>25%</td>
<td>aDataBanker[Formatter Failed To Convert]</td>
</tr>
<tr>
<td>44</td>
<td>16%</td>
<td>44</td>
<td>16%</td>
<td>aProcessor[Processor Failed]</td>
</tr>
<tr>
<td>44</td>
<td>16%</td>
<td>44</td>
<td>16%</td>
<td>aMessage_Generator[Algorithm Failed]</td>
</tr>
</tbody>
</table>

**Figure 8.** FTA Results for Current Implementation

To investigate how redundancy might affect the overall system failure rate in future FWS products, we introduced variations of the redundant units’ number into the FWS. Three types of FWS components were chosen for illustration: the sensor monitor, the message generator and the processor. Each of these has different multi-instance gates in the PLFT. The sensor monitor had only OR gates (i.e., sensor failure occurred if any one of the redundant monitors failed); the message generator had only AND gates (i.e., messages could be transmitted unless all message generators failed); and the process generator had both OR and AND gates. The FTA of the candidate members was then derived automatically from PLFT. The analysis results are shown in Figure 9. As expected, sensor monitor duplications decreased the system failure rate and the increment of message generator duplications increased the system failure rate. Interestingly, the figure shows the increment of sensor monitor duplications will not affect the system failure rate too much after the first duplication (the 2nd unit instance); this is because the conjunction of the probabilities for more than two
failure events is too small to have an impact on the overall fault tree failure. Moreover, the system failure rate will increase faster when duplication number is larger for message generators. This is because when there are more disjointed events, their overall failure rate is nearer to 1, thus have stronger impact on the fault tree logics. For the processor, it’s hard to predict how its duplications will affect the root node failure rate. In this case, duplication analysis is more necessary for taking design decisions.

Cut set analysis can provide useful information on the ways to effectively modify the system in order to prevent the undesired root-node hazard from occurring. For the FWS, the cut set analysis was used to find the 264 minimum cut sets in the PLFT. For some basic events (leaf nodes), it was impractical to try to eliminate them; thus decreasing their failure rate was the only option. In the FWS case study, we took the top ten AFF (the number of minimum cut sets affected by each failure) failure events (in Figure 8) in the basic configuration. We assigned each event the same probability of .0004 of occurrence (the number has no basis in reality). We then asked what would happen if we could reduce that event probability by half.

From Table 1, we can see that the failure event WFS_Controller[Stack_Init_Failed]’s AFF percentage is 100%, which means that it was a basic event in every minimum cut set. Thus eliminating it would remove all 264 minimum cut sets in the PLFT. Since elimination was impossible, we looked at the possible effect of reducing its probability. In this speculative case, reducing its chance of occurrence by 50% could result in lessening the occurrence of the root node by as much as 52.1%. Both the quantitative analysis results and the cut set analysis results provided some guidance on how the occurrence of the PLFT root node (the hazard of concern) could be mitigated.

Table 1

<table>
<thead>
<tr>
<th>Failure Name</th>
<th>AFF %</th>
<th>% Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFS_Controller[Stack_Init_Failed]</td>
<td>100</td>
<td>52.1</td>
</tr>
<tr>
<td>aMessage_Generator[Formatter_Failed_To_Convert]</td>
<td>25</td>
<td>17.1</td>
</tr>
<tr>
<td>Transmitter_Driver[Timer_Failed]</td>
<td>25</td>
<td>17.1</td>
</tr>
<tr>
<td>aDataBanker[Stack_Init_Failed]</td>
<td>25</td>
<td>10.9</td>
</tr>
<tr>
<td>aDataBanker[Formatter_Failed_To_Convert]</td>
<td>25</td>
<td>10.9</td>
</tr>
<tr>
<td>aStack[Stack_Init_Failed]</td>
<td>24</td>
<td>14.7</td>
</tr>
<tr>
<td>aDataBanker[Memory_Failed]</td>
<td>24</td>
<td>10.9</td>
</tr>
<tr>
<td>aProcessor[Processor_Failed]</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>aMessage_Generator[Algorithm_Failed]</td>
<td>16</td>
<td>7</td>
</tr>
</tbody>
</table>

The AADL-PLFT plug-in supported a safety-centered architecture development of the FWS product line. With the automatic FTA feature, each architectural model in AADL could be integrated with basic safety attributes in the model annex. In addition, the AADL-PLFT helped to manage the traceability of safety-related requirements through the various members of the product line. The Cut Set and quantitative analysis results from the plug-in assisted in making design choices (redundant units, tradeoffs among alternatives, high-yield mitigations) in the different candidate products (different CV configurations) in the product line.

5. Conclusion

This paper presents a tool-supported mechanism for integrating a widely used safety-analysis technique, fault tree analysis (FTA), with the AADL models for a product line. The purpose is to make it easier to maintain traceability between safety properties and the architectural design models and to support consistent reuse of the FTA across the systems that compose the product line. We thus developed an AADL plug-in that automatically prunes and adapts the fault tree for a specific product from a previously constructed product-line fault tree. The automated derivation of safety properties for candidate products helps developers pinpoint weak points both in the baseline product-line design and in the proposed architectural choices for new systems to be built in the product line. The fault
tree that is produced by the plug-in uses information from the AADL error models to automatically perform basic quantitative and cut set analyses for each candidate product-line member. Results from application to a small, safety-critical product line, the Floating Weather Station, showed how this tool support allows exploration of the consequences of alternative design choices (e.g., redundant units, possible failure mitigations) on the failure profile as each new member is built in the product line. The introduction of PLFT into AADL extends the scope of early safety analyses (FTA) into the architecture stage of development and also enhances the usability of AADL on the development of the critical product lines. With semi-formal models as basis and automation assisted, more fault tree based safety analyses are ready to use and explore in AADL.

6. Acknowledgment

This research was supported by the National Science Foundation under grants 0204139, 0205588 and 0541163. We thank Josh Dehlinger for his help with PLFaultCAT. We thank David Gluch for introducing us to AADL and providing us with AADL-related information.

7. References