Integrating Product-Line Fault Tree Analysis into AADL Models

Hongyu Sun[1], Miriam Hauptman[1], Robyn Lutz[1,2]
[1] *Department of Computer Science, Iowa State University*
[2] *Jet Propulsion Lab/Caltech*
{sun, miriamh, rlutz}@cs.iastate.edu

Abstract

Fault Tree Analysis (FTA) is a safety-analysis technique that has been extended recently to accommodate product-line engineering. This paper describes a tool-supported approach for integrating product-line FTA with the AADL (Architecture Analysis and Design Language) models and associated AADL Error Models for a product line. The AADL plug-in we have developed provides some automatic pruning and adaptation of the fault tree for a specific product from the product-line FTA. This work supports consistent reuse of the FTA across the systems in the product line and reduces the effort of maintaining traceability between the safety analysis and the architectural models. Incorporating the product-line FTA into the AADL models also allows derivation of basic quantitative and cut set analyses for each product-line member to help identify and eliminate design weaknesses. The tool-supported capabilities enable comparisons among candidate new members to assist in design decisions regarding redundancy, safety features, and the evaluation of alternative designs. Results from a small case study illustrate the approach.

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 family of products designed to take advantage of the common aspects and predicted variabilities of a set of systems [14].

Product-line engineering (PLE) typically consists of two phases: domain engineering and application engineering [14]. 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 [4]. Here we are primarily interested in such critical product lines.

In 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 [5]. Fault Tree Analysis (FTA) was extended to Product Line Fault Tree Analysis (PLFTA) to accommodate product-line engineering for safety-critical domains. To date, work on PLFTA has mainly focused on safety analysis during the requirements phase.

An on-going problem in the product-line engineering of safety-critical software systems is how to represent and use the results of the safety analysis in the subsequent architectural design modeling. Integrating product-line safety analyses with Architecture Analysis and Design Language (AADL) models [13] allows us to apply the results of early safety analyses in the architectural phase of product-line development.

The work described here to incorporate product-line FTA into AADL models also builds on previous work in AADL. For example, 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]. Similarly, Feiler has provided an introduction to modeling families of systems in AADL [7].

This paper describes a tool-supported approach to integrate the product-line FTA into AADL models and the AADL Error Models. The contribution of the work is to provide 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 process consists of the following three steps:

1) **Domain Engineering: PLFT Construction**. This step partially automates the construction of the PLFT. It binds the original product-line fault tree with the 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 subsequent 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 the different types and versions of the PLFTs for the various members of the product line.
3) **Application Engineering: PLFT Analysis.** This feature of the AADL-PLFT assists in the safety analysis of the architectural design when a new member of the product line is built. It contains four functions: pruning a PLFT, adapting a PLFT, basic quantitative analysis and cut set analysis.

A fault tree for a specific product is thus automatically pruned and adapted from the product-line FTA, reducing analysis effort and enhancing consistent reuse of the FTA across the product line. The integration of PLFTA into the AADL models also 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.

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) [14], 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 [14] 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 a fault tree [11]. It includes the following aspects:

1) System Boundary Definition
2) Fault Tree Construction Qualitative Analysis. This typically involves Cut Set Analysis [11] and other analyses related to the quality of the system.
3) 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 [9]. Because the FTA's quality depends heavily on the analyst's expertise, 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 for the new members.

To enable the use of fault trees in product-line engineering, Dehlinger and Lutz introduced product-line fault tree analysis [5]. This reduces the effort and cost to construct the fault trees for each new member in the product line. Since the data in the PLFT have been reused repeatedly across the members of 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 requirements development phase. When new, unanticipated requirements occur, PLFaultCat can 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 usefulness. The AADL-PLFT plug-in described here takes advantage of the concrete system models available at the architectural design phase to automate fault tree generation and analyses. The integration of AADL and PLFT also enables more analyses of design decisions in the product-line members.

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 [13]. It has been used to predict and verify critical properties in several case studies, including Schedulability Analysis on Complex Avionics System (CAS) [1], Deadline, Priority Check based on Rate-Monotonic Analysis (RMA) [6], etc. Feiler describes the modeling of product lines or families in AADL in [7] and provides details on modeling component variations in AADL.

Joshi et al. [10] have used a related approach to automatically extract fault trees from the error models associated with the AADL system model. Their work differs from ours in that we try to reuse the fault trees from the requirements phase and across the members of the product line rather than generating them from the error models.

3. Approach

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 [13]. It describes the internal view of the system.

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.

This section introduces the features of the AADL-PLFT plug-in and describes how to use them.

3.1. AADL-PLFT Plug-in Overview

The AADL-PLFT Plug-in depends on the AADL System Model 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 [13]. 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) [13] in Eclipse. The plug-in contains three main features (see Figure 1), fault tree construction, management, and analysis, which correspond to the three process steps described in Section 1.

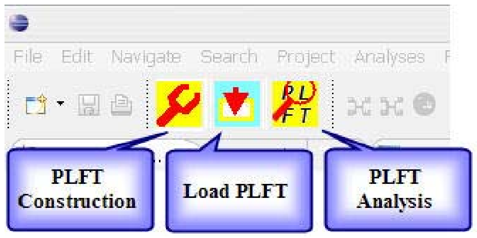

Figure 1. Three features of AADL PLFT Plug-in

3.2. Domain Engineering: PLFT Construction Feature

The following steps show how to construct a PLFT in the AADL model and refer to labels in Figure 2.

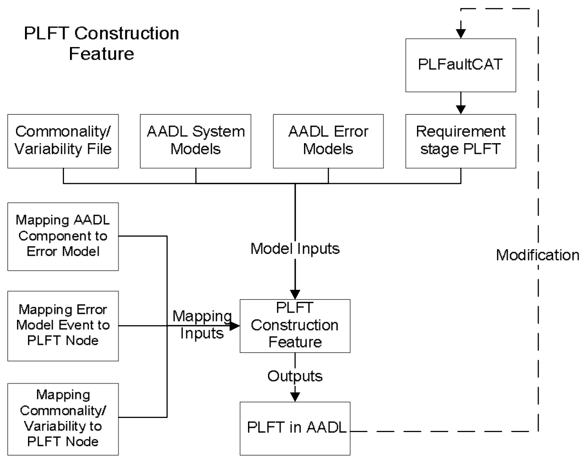
PLFT Construction Feature
PLFaultCAT
Commonality/ Variability File
AADL System Models
AADL Error Models
Requirement stage PLFT
Mapping AADL Component to Error Model
Model Inputs
Modification
Mapping Error Model Event to PLFT Node
Mapping Inputs
PLFT Construction Feature
Outputs
Mapping Commonality/ Variability to PLFT Node
PLFT in AADL

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

1) Model Inputs in Fig. 2: The requirements–phase PLFT is an output of PLFaultCAT. Its production is described in [5]. Besides this, the AADL system models, error models and specifications of the product-line requirements, i.e., the CVs (commonalities and variabilities), are used for the PLFT construction in AADL. We represent the CVs as a property set in AADL. Each commonality or variability requirement is 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., “a FWS has either a high-resolution sensor or a low-resolution sensor”), the default choice is set to true and the other choices are initially set to false.
2) Mapping Inputs in Fig. 2: The AADL-PLFT plug-in reads all the model inputs and enumerates all the components in the system, all failure events in the

error model and all the CVs. The domain engineer then manually creates the mappings between the components in the AADL system model and the PLFT leaf nodes, between the failure events in the error models and the PLFT leaf nodes, and between the CVs and the PLFT node. The mapping is only partially automated because it requires domain knowledge.

3) Outputs in Fig. 2: The PLFT plug-in uses the model inputs and mapping inputs to generate the PLFT in the AADL annex.
4) Modification in Fig. 2: Domain engineers can use PLFaultCAT to reuse this PLFT for subsequent corrections and updates.

3.3. Application Engineering: PLFT Management Feature

This feature allows management of the PLFTs for different failures and version control of PLFTs as the product line evolves. The AADL-PLFT plug-in can load any specific PLFT into the AADL annex in order to perform safety analysis on specific failure events. This feature also allows exploration and comparison of effects among different failures and different versions of PLFTs on the product-line members' design. Comparison among different failures' fault trees may also help reveal unanticipated coupling in the system.

3.4. Application Engineering: PLFT Analyses

Figure 3 shows the four analysis features, labeled F1 through F4. Feature F1 is described in section 3.4.1, feature F2 in 3.4.2, and features F3 and F4 in 3.4.3. R1 through R4 represent the results from the four features.

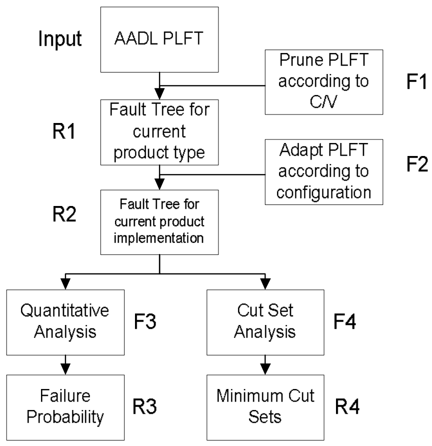
Input
AADL PLFT
Prune PLFT according to C/V
F1
R1
Fault Tree for current product type
Adapt PLFT according to configuration
F2
R2
Fault Tree for current product implementation
Quantitative Analysis
F3
Cut Set Analysis
F4
Failure Probability
R3
Minimum Cut Sets
R4

Figure 3. PLFT Analysis Feature Overview

3.4.1. Pruning the PLFT. The plug-in prunes the PLFT by reading the CVs for the current AADL product model. For example, if a feature is only available 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.

A leaf node in the AADL PLFT can be expressed as Com (F, V, M), where Com is a reference to an AADL component-type model, F refers to a Failure Event generated by the error model, V is a Boolean value indicating whether a variability associated with this component exists in the current product, and M is a multi-instance tag. M captures the relation among the redundant units in the product. For example, the product line in Section 4 allows multiple, redundant sensors. The value of M 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").

3.4.2. Adapting the PLFT. The PLFT plug-in reads in the implementations for all the components and adapts the PLFT according to their 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 replaces the component types in the PLFT with the implementations of the current product-line member. If one component type has more than one implementation, this leaf node will be expanded into a subtree of implementations according to its Multi-Instance Tag and the value of M.

3.4.3. Analyzing the PLFT. The AADL-PLFT Plug-in provides analysis capabilities to investigate the impact of the design decisions on the safety properties of the system. For example, a quantitative analysis can calculate the probability of the fault tree's root node failure event. By comparing failure probabilities among several configurations, the designers can improve their understanding of how the choices (e.g., of redundant units) affect the occurrence of the top events in the fault trees.

The plug-in also provides a cut set analysis capability to show all 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.

The minimum cut sets may be too numerous to analyze manually. For example, in our case study the number of minimum cut sets in one PLFT was 1029. The AADL PLFT plug-in can identify those failure events that exist in many minimum cut sets. If such events can be eliminated, e.g., by introducing a new algorithm, changing the error model's internal logic or using alternative component implementations, the number of minimum cut sets may be significantly reduced (depending on the gate logic).

This section has described an approach to construct, manage and analyze the PLFT in an AADL model using the AADL-PLFT plug-in. Analysis results can provide insights into how the architectural configuration of a product-line member affects the occurrence of root-node failures and into design alternatives for achieving a safer configuration with respect to a particular failure event.

4. Case Study

This section describes an application of PLFT in AADL to a small case study. The Floating Weather Station (FWS) is a product line described by Weiss and Lai in [14]. 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 different FWS products 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 [5].

The weather information from the FWS is required to be highly available and accurate. The FWS is safety-critical for navigation and weather purposes.

4.1. AADL-PLFT construction

The development process for the FWS in AADL follows the workflow shown in Fig. 2.

Commonality/Variability file. The CV file contains the product-line requirements [5, 14], represented as a property set in AADL (as described above in Sect. 3.1). 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
  -- if VariationParameters::speedUnits is knot then
co_speedUnitKnots => true;
  co_speedUnitKnots: inherit aadlboolean  => true applies to (all);
  --------Variabilities
  --v1 the fws may represent wind speed measurements in KPH and
MPH besides knots
  -- if VariationParameters::speedUnits contains (MPH,KPH) then
va_speedUnitMPH_KPH => true
  va_speedUnitMPH_KPH: inherit aadlboolean  => false applies to
(all);
end CVAnalysisResults;
```

AADL system models. The AADL system models contain the type models and the implementation models [13]. The commonalities for the FWS product line are modeled as a system implementation model with all commonalities inside, called a basic configuration. All the other system implementations for members of the FWS product line implement the commonalities by inheriting the basic configuration model. The FWS variabilities are 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. Some systems in the FWS product line may also have wind direction and temperature sensors (optional variabilities). Different kinds of sensors thus inherit a general sensor type model. The following excerpt shows how to model the sensors.

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

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

Each sensor may 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 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 updated.

AADL Error models. Next, we constructed an error model and its implementations for each component in the AADL system model.

In safety analysis of product lines, environmental and human effects are often considered. 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.

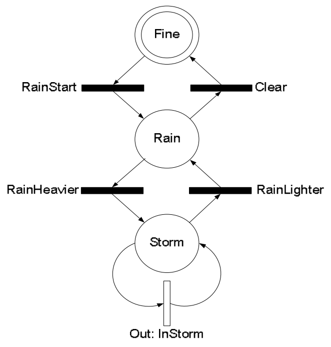

Figure 4. Markov Chain for storm error model

A simple example of a stormy weather environment model was constructed for the FWS use. Its corresponding Markov chain for the error model is illustrated in Figure 4. When the model is in the Storm state, it will output InStorm event to the upper level error model.

AADL-PLFT construction. There are two ways to build product-line fault trees in AADL: to use PLFaultCat as a front-end or to construct the PLFT in AADL manually. 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 [5].

The PLFT plug-in took the output of PLFaultCat, the AADL models for FWS (the system type, implementation models and the error models) and the CVs as input. It enumerated 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 where more than one implementation of the same component existed.

4.2 Safety Analysis with PLFT

After constructing the PLFT, we loaded the PLFT into the AADL model. Due to space limitations, we do not discuss further here the second step in Fig. 1, i.e., the management of the PLFT types and versions for the FWS. Instead, we proceed to the third step, "Fault Tree Analysis", to describe the results of loading the PLFT to analyze the root-node failure: "FWS reports incorrect wind speed." This failure is safety-related since navigation and weather-response decisions often depend on the accuracy of the reported wind speed.

The PLFT analysis feature pruned and adapted the PLFT (top of Fig. 5) into the fault tree of the current product-line member (bottom of Fig. 5).

```
----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_s
peedUnitMPH_KPH][OR_VA][0.00004]) or
(StackManagement[Stack_Initialization_Failed][CVAnalysisResults
::co_dataAsInteger][OR_VA][0.00004] ......
----Fault Tree Excerpt For Current Product Implementation----
Guard_Out=>Incorrect_Wind_Speed_Reported when
((Transmitter_Driver[Stack_Initialization_Failed] or
Transmitter_Driver[Timer_Failed] or (aDataBanker[Timer_Failed]
and aDataBanker[Formatter_Failed_To_Convert])) or ......
```

Figure 5. Product Line Fault Tree and Fault Tree in AADL Model

By comparing the PLFT and the fault tree for the current implementation, we see that the fault tree has been pruned according to the CV properties and each binding component in the fault tree has been substituted by its implementation(s). The preliminary fault tree analysis results are then obtained and are shown in Figure 6. The details of all minimum cut sets were also given in the annex.

To investigate how adding redundant units might affect the overall system failure rate in future FWS products, we introduced variations in the number of redundant units into the FWS. Three types of FWS components are shown for illustration: the sensor monitor, the message generator and the processor. Each of these had different multi-instance gates in the PLFT. The sensor monitor had only AND gates (i.e., sensor failure occurred only if all the redundant monitors failed); the message generator had only OR gates (i.e., incorrect messages could be transmitted if any message generator failed to generate a message correctly); and the process generator had both OR and AND gates. The FTA of the candidate members was then derived automatically from PLFT.

```
--------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
```

Times of App	App %	Num of App	AFF %	Failure
288	109 %	264	100 %	WFS_Controller[Stack _Initialization_Failed]
68	25%	64	24%	aStack[Stack_ Initialization_Failed]
68	25%	64	24%	aDataBanker[Memory_Failed]
66	25%	66	25%	aMessage_Generator[For matter_Failed_To_Convert]
66	25%	66	25%	Transmitter_Driver [Timer_Failed]
66	25%	66	25%	aDataBanker[Timer_Failed]
66	25%	66	25%	Transmitter_Driver[Stack _Initialization_Failed]
66	25%	66	25%	aDataBanker[Formatter _Failed_To_Convert]
44	16%	44	16%	aProcessor[Processor_Failed]
44	16%	44	16%	aMessage_Generator [Algorithm_Failed]

Figure 6. FTA Results for Current Implementation

A graphical summary, based on the AADL analysis results, is shown in Figure 7. The units of measures on the axes were only speculative and have been removed here to make the figure more legible. As expected, sensor monitor duplications were seen in the AADL implementation to reduce the system failure rate, and message generator duplications were seen to raise the system failure rate. Interestingly, the figure shows that adding more than one redundant sensor monitor did not affect the system failure rate much. This is because the conjunction of the small probabilities was too small to affect the overall fault tree failure.

Cut set analysis can also provide useful information about ways to modify the systems to prevent an undesired root-node hazard from occurring. For the FWS, the cut set analysis was used to find the 264 minimum cut sets in this PLFT. We then took the top ten AFF (leaf nodes ranked according to the number of minimum cut sets affected by them) (Fig. 6) in the basic configuration. We asked what would happen if we could reduce a failure event's probability by half.

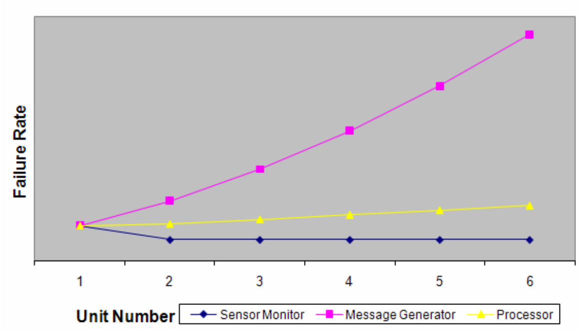

Figure 7. Analogy of failure rates on different numbers of unit duplications

From Table 1, we can see that the failure event WFS_Controller[Stack_Initialization_Failed]'s AFF percentage is 100%, which means that by eliminating it, we would remove all 264 minimum cut sets in the PLFT. For some basic events (leaf nodes), it was impractical to eliminate them, so decreasing their failure rate was explored as an option. In this speculative case, reducing its chance of occurrence by 50% could lessen the occurrence of the root node by as much as 52.1% (Table 1). Both the exploratory, quantitative analysis results and the cut-set analysis results provided some guidance on how the occurrence of the PLFT root node might be mitigated in alternate implementations.

Table 1. How Single Failures Affect System Failure Rate

Failure Name	AFF %	% Red uced
WFS_Controller[Stack_Initialization_Failed]	100	52.1
aMessage_Generator[Formatter_Failed_To_Conv ert]	25	17.1
Transmitter_Driver[Timer_Failed]	25	17.1
Transmitter_Driver[Stack_Initialization_Failed]	25	17.1
aDataBanker[Timer_Failed]	25	10.9
aDataBanker[Formatter_Failed_To_Convert]	25	10.9
aStack[Stack_Initialization_Failed]	24	14.7
aDataBanker[Memory_Failed]	24	10.9
aProcessor[Processor_Failed]	16	7
aMessage_Generator[Algorithm_Failed]	16	7

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 introduced 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 analysis capabilities can help developers pinpoint weak points both in the baseline product-line design and in the proposed architectural choices for new systems. Results from a small case study showed how this approach allows exploration of the consequences of alternative design choices (e.g., redundant units, possible failure mitigations). 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 in the development of critical product lines.

6. Acknowledgments

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

7. References

[1] R. Allen, S. Vestal and B. Lewis, “Using an Architecture Description Language for Quantitative Analysis of Real-Time Systems”, *Proc. 3rd Int’l Workshop on Software and Performance*, Rome, Italy, 2002, pp. 203-210.

[2] J. Bergey, S. Cohen, L. Jones and D. Smith, *Software Product Lines: Experiences from the Sixth DoD Software Product Line Workshop, Technical Note*, Section 2.5, CMU/SEI-2004-TN-011.

[3] B. Boehm, A.W. Brown, R. Madachy, and Y. Yang, “A software product line life cycle cost estimation model”, *Int’l Symposium on Empirical Software Engineering*, IEEE Computer Society, 2004, pp. 156–164.

[4] J. Dehlinger and R. Lutz, “A Product-Line Approach to Promote Asset Reuse in Multi-Agent Systems”, SELMAS IV, 2006 LNCS Volume 3914, pp. 161-178.

[5] J. Dehlinger and R. Lutz, “PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool”, *Automated Software Engineering*, 13(1), Jan 2006, pp. 169-193.

[6] P. Feiler, D. Gluch, J. Hudak and B. Lewis, “Pattern-Based Analysis of an Embedded Real-time System Architecture”, *Int’l Workshop on Architecture Description Languages (WADL04)*, Aug 2004.

[7] P. Feiler, Modeling of System Families, AADL-WIKI, Available at http://la.sei.cmu.edu/aadl-wiki/index.php/Modeling_of_System_Families (Current Aug 2007).

[8] L. Geyer and M. Becker, “On the Influence of Variabilities on the Application-Engineering Process of a Product Family”, *Proc. Software Product Line, Second Int’l Conference*, San Diego, CA, USA, August, 2002.

[9] K. Goseva-Popstojanova, A.E. Hassan, A. Guedem, W. Abdelmoez, D.E.M. Nassar, H.H. Ammar and A. Mili, “Architectural-Level Risk Analysis Using UML”. *IEEE Trans. Software Eng*, 2003, 29(10): 946-960.

[10] A. Joshi, P. Binns and S. Vestal, Automatic Generation of Fault Trees from AADL Models, *Proc. Aerospace Software Engineering Workshop*, ICSE, 2008.

[11] Raheja, D. and M. Allocco, *Assurance Technologies Principles and Practices: A Product, Process and System Safety Perspective* (2nd Edition), John Wiley & Sons, Inc.

[12] A. Rugina, K. Kanoun and M. Kaâniche, “An Architecture-based Dependability Modeling Framework using AADL”, *Proc. 10th IASTED Int’l Conference on Software Engineering and Applications (SEA'2006)*, Dallas, USA, November, 2006.

[13] SAE AADL Standard Info, www.aadl.info (Current Aug 2007).

[14] Weiss, D.M., and C.T.R. Lai, *Software Product Line Engineering: A Family-Based Software Development Process*. Boston: Addison-Wesley, 1999.