System Dependability Modelling and Analysis Using AADL and HiP-HOPS

Zhibao Mian*, Leonardo Bottaci**, Yiannis Papadopoulos*** and Matthias Biehl****

*, **, ***: Computer Science Department, University of Hull, HU6 7RX, UK
(e-mail: {Z.Mian@2009, L.Bottaci@, Y.I.Papadopoulos@}.hull.ac.uk).

****: Embedded Control Systems, Royal Institute of Technology (KTH), Sweden
email: biehl@md.kth.se

Abstract: The Architecture Analysis and Design Language (AADL) is gaining widespread acceptance in aerospace, automobile and avionics industries for designing dependability-critical systems. The design process of dependable systems must address both cost and dependability (safety, reliability, availability, maintainability) concerns. This requires translating concepts of the design domain to the dependability analysis domain. We automate such a translation between AADL and the dependability analysis tool HiP-HOPS by using model transformation techniques. A generic primary-standby example system is used to show the mechanics of the transformation and the potential for highlighting problems and assisting design work using this technology.

Keywords: Dependability modeling, Dependability Analysis, AADL, HiP-HOPS, Model transformation.

1. Introduction

Dependable systems are those systems that have high dependability requirements. For example, safety critical systems such as transport and medical engineering systems, need to be dependable because their failure or malfunction may harm people or the environment. During the last two decades, a considerable body of work has been developed to ensure that dependability concerns are satisfied. A large body of this work, classed broadly as safety analysis (Vesely et al., 2002; Joshi et al., 2006), is concerned with understanding the relationship between system failures and their causes and then reducing the probability of failures by preventing those causes or modifying the system design to make them less likely.

The emerging paradigm of model-based development promises a more formal system development process which is consistent, automated, and most important, tightly integrated with the safety engineering processes. However, integrating safety analysis into model-based design requires the translation of concepts from the design domain to the safety analysis domain. Biehl et al. (2010) has translated automotive architecture description language (EAST-ADL2) concepts to the safety analysis tool HiP-HOPS (Adachi et al., 2011) using model transformation techniques. Recently, the Architecture Analysis and Design Language (AADL) (SAE-AS5506, 2006; Feiler et al., 2006) has emerged as a potential future standard for model-based development. A number of approaches have been proposed to enable dependability analysis based on AADL models. Joshi et al. (2007) produced a static fault tree generator prototype based on AADL models. This work has been extended by Dehlinger and Dugan (2008), so that dynamic fault trees are generated automatically from AADL models. In Rugina et al. (2008), an AADL dependability model is transformed into a Generalised Stochastic Petri Net.

In this paper we outline a new model transformation framework (AADL2HiP-HOPS) for the automatic generation of HiP-HOPS-oriented dependability analytical models from high level AADL architecture models. The benefit of this approach compared to earlier work is that it opens a path that will enable the AADL language to take advantage of some of the unique capabilities of HiP-HOPS which include the synthesis of multiple failure mode FMEAs, temporal fault tree analysis and evolutionary architecture optimisation with respect to dependability and cost.

In the proposed approach, AADL is used as the notation for capturing the system architecture model and the AADL Error Model annex is used to capture the component faults and failure modes. The system architecture model annotated with the AADL error model of components is called an AADL dependability model. AADL2HiP-HOPS is the tool that transforms this AADL dependability model to a HiP-HOPS representation which is then used for synthesis of fault trees, FMEAs and other analyses to automatically generate the fault tree and FMEA table of system for further dependability analysis.

2. Introduction to AADL and HiP-HOPS

2.1 AADL Overview

The Architecture Analysis and Design Language (AADL) is an SAE standard for the specification and analysis of the software and hardware architecture of real-time performance-critical systems. Performance-critical systems are systems whose operation strongly depends on meeting non-functional
system requirements such as reliability, availability, timing, safety and security. AADL uses a component-based paradigm and provides a number of modelling concepts, in a number of dimensions, that can be used for both analysis and design of embedded systems. The language is used to describe the structure of systems as an assembly of software components mapped onto an execution platform. It describes how components are combined into sub-systems and how they interact. The language supports early and repeated analysis of system architectures with respect to performance-critical properties through an extensible notation, a tool framework and precisely defined semantics (Feiler et al., 2006). The language can also describe adaptable systems through the use of operational modes and mode transitions.

AADL is designed to be flexible and can be extended to accommodate analyses of the runtime architectures that the core language does not completely support. This is done through its extensible standard language that permits textual and graphical system representation and the addition of analysis-specific properties as well as approved sublanguage extensions. By using the extension capabilities of the language, additional models and properties can be included. In particular, an AADL Error Model Annex can be used to define error models and properties of components facilitating a Markov or fault tree analysis of the system dependability.

2.2 AADL Error Model Annex Overview

The AADL Error Annex (SAE-AS5506/1, 2006) defines a sub-language of AADL that supports specification of dependability-related information such as fault and repair assumptions, error propagations, fault tolerance policies, and stochastic parameters specifying the occurrence of fault events and propagations. The error models describe the behaviour of the components to which they are associated in presence of local (internal) failure and repair events, as well as in presence of output failure propagations from the component’s input deviations. The resulting annotated model can then be used as an input to a dependability analysis. In this way, the error models enable the qualitative and quantitative assessments of system properties such as safety, reliability, availability, and maintainability.

In the AADL Error Annex, error propagation rules are predefined to specify the potential error propagation paths between various types of components and connections. For example, a processor can propagate an error to the process hosted on that processor. These error propagations can only occur in the direction along the dataflow in the architecture and cannot occur where the components are not connected to each other either through direct connections (port or access) or through explicit bindings.


2.3 HiP-HOPS Overview

HiP-HOPS is a state-of-the-art system dependability (i.e. safety, reliability and availability) analysis technique. It offers a significant degree of automation and reuse, countering problems arising from the increasing complexity of systems. HiP-HOPS uses the topology of a system together with reusable local failure specifications at component level to automatically produce a network of interconnected fault trees and an FMEA (Failure Modes and Effects Analysis) for the system. The technique is supported by an automated tool.

HiP-HOPS defines a language for the description of failure behaviour at component level. In the basic version of this language, the failure behaviour of a component can be specified as a list of internal failure modes of the component (internal malfunctions) and a list of deviations of parameters as they can be observed at component outputs (output deviations). Each internal malfunction is optionally accompanied by quantitative data, for example a failure and a repair rate if these are known. Output deviations carry Boolean expressions which describe their causes as a logical combination of internal malfunctions of the component and similar deviations of parameters at component inputs (input deviations).

HiP-HOPS has recently been extended with multi-objective optimisation capabilities (Adachi et al., 2011). These allow the tool to search the design space, defined by the variability of a design model, for potential design solutions that are optimal, or near optimal, in terms of dependability and cost. In this approach, a variable design model for a system is one in which components and subsystems have alternative user defined implementations which can include standard fault tolerant configuration schemes.

3. Transformation of AADL dependability models to HiP-HOPS models

3.1 Model to Model transformation

Integrating dependability analysis into the design process requires the translation of concepts from the design domain to the dependability analysis domain. More specifically, the goal of the model transformation is to generate from the AADL model, a HiP-HOPS model that captures the relevant component structure, topology and local failure information required for the HiP-HOPS analysis.

The transformation design described in this paper combines the general dependability evaluation methods introduced in Rugina at el. (2007) and Rugina (2007) with the transformation design introduced in Biehl et al. (2010). The key reason that we use the concepts in Biehl et al. (2010) is because their work is related to ours. However the transformation is from a different model (AADL) and the scope is broader aiming to encompass not only the dependability analysis but also the optimisation and temporal analysis (Mahmud et al., 2010) capabilities of HiP-HOPS. Fig. 1 shows the two main steps of our transformation.
1. Model to Model transformation (M2M in Fig. 1): The first step is a semantic mapping. It transforms an AADL model into a corresponding HiP-HOPS model containing the relevant failure and reliability information. Some of the key semantic mappings are shown in Table 1.

2. Model to Text transformation (see M2T in Fig. 1): The second step transformation is a model to text transformation. It transforms the HiP-HOPS model into the concrete syntax of the HiP-HOPS input file format.

Fig. 1. The overview of transformation design.

<table>
<thead>
<tr>
<th>Source pattern (AADL + Error Annex)</th>
<th>Target Pattern (HiP-HOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemInstance</td>
<td>System</td>
</tr>
<tr>
<td>ComponentInstance</td>
<td>System.Component.Implementation</td>
</tr>
<tr>
<td>ComponentErrorModelProperty</td>
<td>System.component.Implementation.FailureData</td>
</tr>
<tr>
<td>ComponentInstance.featureInstance</td>
<td>System.Component.Ports</td>
</tr>
<tr>
<td>ConnectionInstance</td>
<td>System.Lines</td>
</tr>
</tbody>
</table>

Table 1. Semantic mapping between AADL and HiP-HOPS.

AADL and HiP-HOPS models differ in complex ways. For example, in AADL, type declarations are separated from implementation descriptions. The implementation description references the type declaration by name. In HiP-HOPS, however, the usage of a type is always coupled with its declaration. When transforming from AADL to HiP-HOPS, the declarations have to be “in-lined” into every point of usage. The Boolean failure expressions required by HiP-HOPS must be generated from the AADL error model state machine. The connectivity concept in HiP-HOPS is a line. There is no direct analogue of a line in AADL but information from which lines can be created can be obtained from AADL data connection elements in conjunction with a number of other AADL elements.

3.2 The Model transformation language

Model to model transformation languages are well suited for our semantic mapping transformation, since both input and output are models. Model transformation languages and engines have been classified in Czarnecki and Helsen (2006), and Bieth et al. (2010). Different model transformation languages have their advantages and disadvantages in solving specific types of tasks. We chose the ATLAS Transformation Language (ATL) (Jouault et al., 2008) which is a hybrid language containing a mixture of declarative and imperative constructs. We choose ATL not only because it hides the complexity of model transformation behind a simple syntax but also because:

1. It has been shown to provide effective means of achieving similar model transformations (Bieth et al., 2010).

2. The model transformation tool can be integrated into the growing toolset surrounding AADL. The Open-Source AADL Tool Environment (OSATE) developed by SEI (Feiler et al., 2006; SAE, 2006) is a set of plug-ins based on Eclipse and the Eclipse Modelling Framework (EMF) (Steinberg et al., 2009). We use OSATE to parse and semantically validate textual AADL specifications into models for input to ATL transformations. Readers are referred to Jouault et al., (2008) and ATLAS group (2005) for more detailed information on ATL. The HiP-HOPS model conforms to the ecore (Steinberg et al., 2009) modelling language.

4. Case study: Standby-Recovery System

The approach is being applied on a model of a brake-by-wire system for cars. A discussion of this case study would require much more space than is available in this paper. However, to illustrate the useful application of the method, we discuss an example based on a small and manageable fragment of the system. Fig. 2 illustrates a simple standby-recovery system in which function P processes the value generated by input function I.

Fig. 2. Standby-Recovery System.

When omission at the output of P is detected, a normally redundant function S is initiated to replace P. In this system, each component has a failure mode “Failed” which causes omission of output of that component. In addition omission of input will cause omission of P and S respectively when they are in operation.

4.1 AADL model of Standby-Recovery System

Fig. 3 shows the state machines that define the AADL error model for this system. In Fig. 3, the S component is initially in the Dormant state. When P fails (this is designated with transition to an error state of component P in the AADL error model), an omission of P output is detected as Omission-monitor by component S, in which case S moves to the Active state replacing the failed component P. While S is active, a failure of I detected as Omission-input by S or an internal failure of S will cause a transition to an error state where there is omission of output. Once in this error, S propagates output error Loss-data. The state machines for the
P and I components are similarly constructed and self-explanatory.

![Fig. 3. State machines for Standby-Recovery error behaviour.](image)

A partial AADL description for the standby recovery system is shown in Fig. 4, 5, and 6.

```
package My_ErrorModels
public
annex Error_Model {**
    error model S_Error
    features
    Dormant: initial error state;
    Omission-output: error state;
    Active: error state;
    Failed: error event;
    Omission-input: in error propagation [occurrence -> fixed 0.8];
    Loss-data: out error propagation [occurrence => fixed 0.8];
    Omission-monitor: in error propagation [occurrence => fixed 0.8];
end S_Error;
    error model implementation S_Error.impl
    transitions
    Dormant -> in Omission-monitor -> Active;
    Active -> Omission-output;
    Active -> Failed -> Omission-output;
    Omission-output -> out Loss-data -> Omission-output;
    properties
    occurrence => poisson 1.0e-6 applies to Failed;
end S_Error.impl;
**};
end My_ErrorModels;
```

Fig. 6. AADL error model type and implementation declaration for S component which encodes the state transitions shown in Fig. 3.

4.2 Translation of state machine to HiP-HOPS failure expression

A conversion algorithm is needed to translate the state machines to failure expressions. Essentially, the algorithm generates a fault tree for every final state of the state machine with that final state as the top event. A backwards traversal is performed which starts with each final state and ends at the start state. Every path between those states becomes a new branch in the fault tree.

The AND and OR logical operator is added during the traversal process. AND operators are used between each error event or error propagation, so that all events in each path are represented as a conjunction. For each visited state, if there are two or more error events or error propagations between the same two error states, which is common to more than one path, the OR logic operator is added between each error event and error propagation.

Using the above AND and OR rules, the logical failure expression for the final state Omission-output of S component can be produced:

S: Omission-output = Omission-monitor AND (Omission-input OR Failed(S))

With some simplification for clarity, the failure logic of functions I, P and S is:
I: Omission-pvalue = Failed(I)
P: Omission-output = Omission-input or Failed(P)
S: Omission-output = Omission-monitor and (Omission-input or Failed(S))

A failure expression is a logical expression showing how the failure of a component depends on events such as omission of input, internal failure and so on. Each such expression is effectively a local fault tree for each component.

In HiP-HOPS, the propagation of failures between components is described by Line elements. There is no single equivalent element in the AADL model. Instead Line elements must be constructed from an analysis of the AADL data connection elements. In the Standby-Recovery system, there are three data connections, two of which are from the same port (Lpvalue).

DataConnection1: Lpvalue -> P.input
DataConnection2: Lpvalue -> S.input
DataConnection3: P.output -> S.monitor

These two data connections correspond to a single HiP-HOPS Line. The third data connection, which does not share a port with any other data connection should be transformed into a single Line. In HiP-HOPS, these two Lines are part of a Lines element. The result is shown below.

Fig. 7. Fault tree and FMEA generated by HiP-HOPS.

In general, the set of data connections must be partitioned according to common source ports. There is also the problem that in AADL, the source port and destination port of a data connection need not belong to components at the same hierarchical level (e.g. the source may be in a sub-component of the destination). In HiP-HOPS, the sub-component Line is distinct from the inter-component Line.

From the above expressions and the component structure of the model, HiP-HOPS is able to generate the fault trees and FMEA illustrated in Fig. 7. The FMEA shows that a failure of the input function I causes omission of both the normal and standby outputs of the system and is, therefore, a critical failure.

On the other hand, a single failure of P causes only omission of normal output and can, therefore, be seen as less critical. Finally, failure of S does not have any direct effect on the system. It becomes significant only in conjunction with failure of P which in this design provides the condition that precisely triggers the need to deploy S.

The FMEA indicates that input function I is the critical element in this design, representing a hazardous dependency between the two redundant processing functions P and S. Failure of I is, indeed, a direct cause of a critical system failure (omission of the standby output) and should, therefore, be made unlikely by design. On the other hand, the analysis shows that an independent failure of either P or S cannot cause a critical system failure. Emphasis in the design should, therefore, be placed on how to protect these two functions from common cause failures such as those caused by electromagnetic interference.
dependency is very close to the affected functions but also because the model and associated failure logic are very simple. However, in reality, hazardous dependencies are not always as simple to detect especially those originating from remote energy and data sources which are deeply hidden in the hierarchy of complex designs. The detection of such dependencies is, indeed, a hard task which justifies, we believe, the provision of useful automated support to AADL system designers and safety analysts.

4.3 Model transformation from AADL to HiP-HOPS

At the top level, the algorithm for model transformation iterates through the basic components of the AADL model. For each component, a HiP-HOPS failure expression is generated from the state machine error description. Finally, the component connection information is obtained by transforming AADL connection objects to HiP-HOPS line objects. Note that the implementation of the AADL2HiP-HOPS plugin including model to model (M2M) transformation and the model to text transformation is being extended to enable access to more advanced capabilities of HiP-HOPS.

5. Summary and future work

This paper has described a method and tool support for using HiP-HOPS to perform dependability analysis on systems modelled in AADL. Model transformation techniques have been used to construct a HiP-HOPS model. A simple example was used to show the mechanics of transformation and the potential for highlighting problems and assisting design work using this technology. Currently this work supports fault tree and FMEA analysis of AADL models. Future work will focus on providing access to temporal safety analysis, and multi-objective system optimisation in the context of AADL design.

Acknowledgement

This work was supported partly by the EU Project MAENAD (Grant 260057).

REFERENCES


SAE-AS5506, 2006, Architecture Analysis and Design Language (AADL), Society of Automotive Engineers (SAE).

