ANALYZING DYNAMIC FAULT TREES DERIVED FROM MODEL-BASED SYSTEM ARCHITECTURES

JOSH DEHLINGER* and JOANNE BECHTA DUGAN
Charles L. Brown Department of Electrical and Computer Engineering, University of Virginia
351 McCormick Road, PO Box 400743, Charlottesville, VA 22904-4743
*Corresponding author. E-mail : jdehlinger@virginia.edu

Received June 20, 2008

Dependability-critical systems, such as digital instrumentation and control systems in nuclear power plants, necessitate engineering techniques and tools to provide assurances of their safety and reliability. Determining system reliability at the architectural design phase is important since it may guide design decisions and provide crucial information for trade-off analysis and estimating system cost. Despite this, reliability and system engineering remain separate disciplines and engineering processes by which the dependability analysis results may not represent the designed system.

In this article we provide an overview and application of our approach to build architecture-based, dynamic system models for dependability-critical systems and then automatically generate dynamic fault trees (DFT) for comprehensive, tool-supported reliability analysis. Specifically, we use the Architectural Analysis and Design Language (AADL) to model the structural, behavioral and failure aspects of the system in a composite architecture model. From the AADL model, we seek to derive the DFT(s) and use Galileo's automated reliability analyses to estimate system reliability. This approach alleviates the dependability engineering – systems engineering knowledge expertise gap, integrates the dependability and system engineering design and development processes and enables a more formal, automated and consistent DFT construction. We illustrate this work using an example based on a dynamic digital feed-water control system for a nuclear reactor.

KEYWORDS : Dynamic Fault Trees, Architecture Description Languages, Fault Tree Synthesis, Reliability Analysis, Dynamic System Modeling.

1. INTRODUCTION

Dependability-critical systems require the engineering techniques and tools to provide high assurances of their safety and reliability. Similarly, software-intensive systems are increasingly becoming essential to dependability-critical infrastructure requiring dependability engineering techniques to sufficiently analyze the impact of the hardware and software (as well as their interactions and interfaces) on its overall reliability and safety. Despite the acknowledged need for high assurance in dependability-critical infrastructure software systems, such as the Digital Feed-Water Control System (DFWCS) for a nuclear reactor used in this work [1], the engineering for these systems focus design and development efforts on the functional behavior required of the system under normal operating conditions and given environmental assumptions such that consideration of failure scenarios may be delayed until after the design is completed.

Dependability engineering entails calculating module and system failure rates [2-5]; modeling fault and failure propagations [6-8]; determining failure modes and contingencies; developing diagnostic, prognostic and system health monitoring and management mechanisms [9-12]; and designing for recovery, reconfiguration and reconstitution strategies to handle operational faults and failures [13]. Such dependability engineering technologies need to be developed and incorporated early in design and development process to allow for ample consideration of possible failures, mitigation strategies, tradeoff analyses and economic considerations.

Dependability engineers typically perform the dependability analysis manually based on informal software engineering assets including design models, architecture diagrams and requirements documents. This manual process places the onus of creating and analyzing the dependability engineering assets and on the dependability engineers and thus is subject to the skill and expertise of the engineer [4] [14]. Fault tree analysis is a common dependability engineering technique that is subject to inconsistency since engineers can develop differing, but accurate, fault trees for the same system [4] [14-15]. To resolve differences among engineers, a system's final fault tree often must be resolved through review and consensus building among dependability and system engineers to ensure that the fault tree reflects the actual system [14]. Further, the burden of manually identifying and exposing the potential failure modes in the interactions among system components is a complex task that can result in missing/unaccounted failure scenarios in the final fault tree(s).

Dynamic fault trees are an extension to traditional fault trees that allow for dynamic system events (e.g., functional dependencies, sequence-dependent failures, spares, etc.) when modeling the possible failures of a critical system [2] [4] [13] [16]. Like fault tree analysis, dynamic fault trees provide the capability to qualitatively and quantitatively analyze the failure scenarios of a critical system. Galileo, used in this work, provides tool support for traditional, static fault trees and dynamic fault trees and provides numerous reliability analyses to the engineer [17].

In this article, we attempt to bridge the divide between dependability and system engineering. We describe our approach to build architectural-based, dynamic system models for dependability-critical systems, such as DFWCS, and then automatically derive the dynamic fault trees for further reliability analysis using Galileo. We first describe how the Architectural Analysis and Design Language [6] [18-20] can be utilized to model the structural, behavioral and failure aspect of a critical system in a composite architecture model. From this model, we describe how a dynamic fault tree can be automatically derived, and then, finally, we detail the use of Galileo's automated reliability analyses to estimate system reliability. The contribution of this is an approach that alleviates the dependability engineering – system engineering gap by integrating dependability engineering into the design and development process of a critical system and enabling a more formal, automated and consistent dynamic fault tree analysis.

2. RELATED WORK AND BACKGROUND

The work presented here builds upon recent work in the architectural modeling of a critical system using the Architectural Analysis and Design Language as well as our work in probabilistic risk assessment using dynamic fault trees and its tool support, Galileo. We discuss this previous work along with other related work to provide the context and background for the work presented in Section 3.

2.1 The Architectural Analysis and Design Language

We utilize the Architectural Analysis and Design Language (AADL) [6] [20], a Society of Automotive Engineers (SAE) standard [21], to model, specify and analyze the structural, behavioral and failure aspects of a real-time, critical system using graphical and/or textual notations. AADL was specifically designed to model "real-time embedded systems, complex systems of systems, and specialized performance capability systems" [6] and is thus applicable to nuclear domain systems, such as the Digital Feed-Water Control System (DFWCS) used in this article. In addition, to allow for the generation of dynamic fault trees (DFT) from the specified AADL models, we use AADL's Error Model Annex [22], a companion language to AADL that support the specification of fault and failure information and propagations to system components. The use of AADL's Error Model Annex facilitates the generation of system fault trees since it allows for the specification of fault/failure annotations directly integrated into the original architecture model [6] [22]. This ensures that the fault/failure scenarios accurately reflect the designed system at an early stage in the development lifecycle and enables the dependability analysis to consider the component fault/failure scenarios and their interactions in the context of the system architecture [14].

While this work proposes and focuses on deriving DFTs from the AADL model, the combination use of AADL and its Error Model Annex has been used for other dependability engineering activities. AADL has been previously used as a validation and verification model for dependability-critical systems using various approaches and case studies including scheduling analysis on complex avionics systems [18], a Generalized Stochastic Petri Net model derived directly from an AADL model [7-8], model-driven fault tolerance design [22], security level analysis and memory buffer analysis on a queuing system [24]. This work will build upon these approaches by providing further dependability engineering processes and methods for AADL models and will improve the utilization of AADL as an engineering tool in dependability-critical domains.

2.2 Dynamic Fault Tree Analysis and Galileo

Fault trees are a graphical representation of a system hazard depicting the underlying causal events using Boolean logic gates and are used to reason about and/or quantifiably estimate the potential cause(s) of a system failure [2] [15]. Fault tree analysis is a backward search technique that starts from a system failure and works towards the initiating events (i.e., the causing events that may lead to the system hazard/failure) [15]. Dynamic fault trees (DFT), developed by Dugan et al. [2] [4] [13], are an extension of traditional fault trees that utilizes Markov chains to provide for dynamic system events (e.g., functional dependencies, sequence-dependent events, spares, etc.) when modeling the possible causal events of a root node, system hazard.

Coppit, Sullivan, Dugan and others developed Galileo as the tool support for developing and analyzing both traditional, static fault trees and DFTs [16-17]. Further, Galileo enables for multiple reliability analyses (e.g., uncertainty analysis, common cause groups, cutset generation, phased-mission modeling and analysis, etc.) to quantifiably estimate the system probabilistic risk assessment (PRA).

More recently, Dugan, Pai and Xu have used dynamic event/fault trees (DEFT) to improve the expressiveness of DFTs. DEFT combines event trees, a decision tree that graphically represent how an initial perturbation or event can propagate forward to result in a mitigating or aggravating possible outcome(s) [15], with DFTs [2]. The DEFT represents a tree that can terminate in either success and/or failure scenarios and allows for the modeling and identification of dependencies affecting system components, dependencies between components and the relationship between system components and software in the context of a PRA [2].

This article builds upon earlier work by utilizing a DEFT and its associated DFTs and performing a PRA using Galileo and complements it by allowing for the automatic generation of a DFT directly from a critical system's architectural model to ensure the dependability analysis accurately reflects the design.

2.3 Linking Fault Trees and System Design Diagrams

This work uses the Architectural Analysis and Design Language (AADL), described in Section 2.1, to model a critical system and then derive the relevant dynamic fault trees (DFT) to perform probabilistic risk assessment using Galileo, described in Section 2.2. While our approach is novel, related comparable research exists that compliments the work presented in this article.

Pai and Dugan provided some initial work similar to that presented here by generating DFTs from specially annotated UML models [4]. Although the goal is similar, their work relies on specialized UML semantics and is dependent on a particular UML modeling tool that can not model the critical systems concerned in this work. Unlike [4], we advocate for the use of AADL to model critical systems because it: 1. allows for the specification of a system's structure, behavior and potential faults/failures within the system model; 2. enables and supports other dependability analyses, described in Section 2.1; and, 3. provides a formal, standardized model-based, architectural language specifically for real-time, critical systems.

Sun, Hauptman and Lutz in developed a tool to manually associate a user-built, product-line software fault tree analysis [25] with the related product-line AADL models with the intention of extending the scope of early safety analyses into the architecture stage of system development [26]. The work presented in this article differs from that of [26] in that we advocate for the automatic derivation of DFTs directly from the AADL architectural model to ensure the accurate reflection of the system in the dependability analysis and to enable comprehensive PRA by including dynamic events.

Most relevant to this work, Joshi, Vestal and Binns developed the process of automatically deriving static fault trees from ADDL models [14]. In this article, we extend their work to allow for the derivation of DFTs from an AADL model so that a probabilistic risk assessment, using Galileo, can be performed to estimate the reliability of a critical system. The generation of DFTs as outlined in this article, rather than static fault trees as in [14], is advantageous in that the DFT can better reflect the faults, spares and repair/recovery properties found in high assurance systems.

3. MODELING CRITICAL SYSTEMS USING AADL

In this article, we advocate for the use of the Architectural Analysis and Design Language (AADL) [6] [20] as the medium for specifying and modeling the structural, behavioral and failure characteristics of a dependability-critical system. The motivation is, from the architectural model specified in AADL, we propose (see Section 4) how dynamic fault trees can be automatically generated and then analyzed for the systems probabilistic risk assessment, see Section 5. In this section however, we give a brief overview of AADL and how it can be used to specify/model the critical components of a Digital Feed-Water Control System (DFWCS) [1] for a nuclear power plant; for a full description of AADL, interested readers should consult [6] [20].

3.1 Modeling a System Architecture in AADL Overview

In the Architectural Analysis and Design Language (AADL), system *components* serve as the main construct in specifying the architecture of a system. AADL system components are partitioned into two distinct types: *software components* and *execution platform components* [20]. Software components include architectural structures for *processes, threads, thread groups, data* and *subprograms*. Execution platform components include architectural structures for *processors, memory* and *buses*. Using these software and execution platform components, a system is then composed using the composite *system* AADL architectural structure. AADL defines required binding properties between AADL software components and execution platform components (e.g., all software threads must be associated with a processor) to ensure a viable system.

AADL further defines system architectural components through *type* and *implementation* declarations [20]. An AADL component's type declaration specifies its externally observable attributes and interface elements. A component's interface elements include AADL *features*, defined interaction points with other system components (e.g., data input/output ports, event input/output ports, subprogram calls, etc.), and *properties*, the inherit characteristics of a system component (e.g., periodic/aperiodic intervals, deadlines, etc.). A component's implementation declaration specifies its internal structure by declaring its

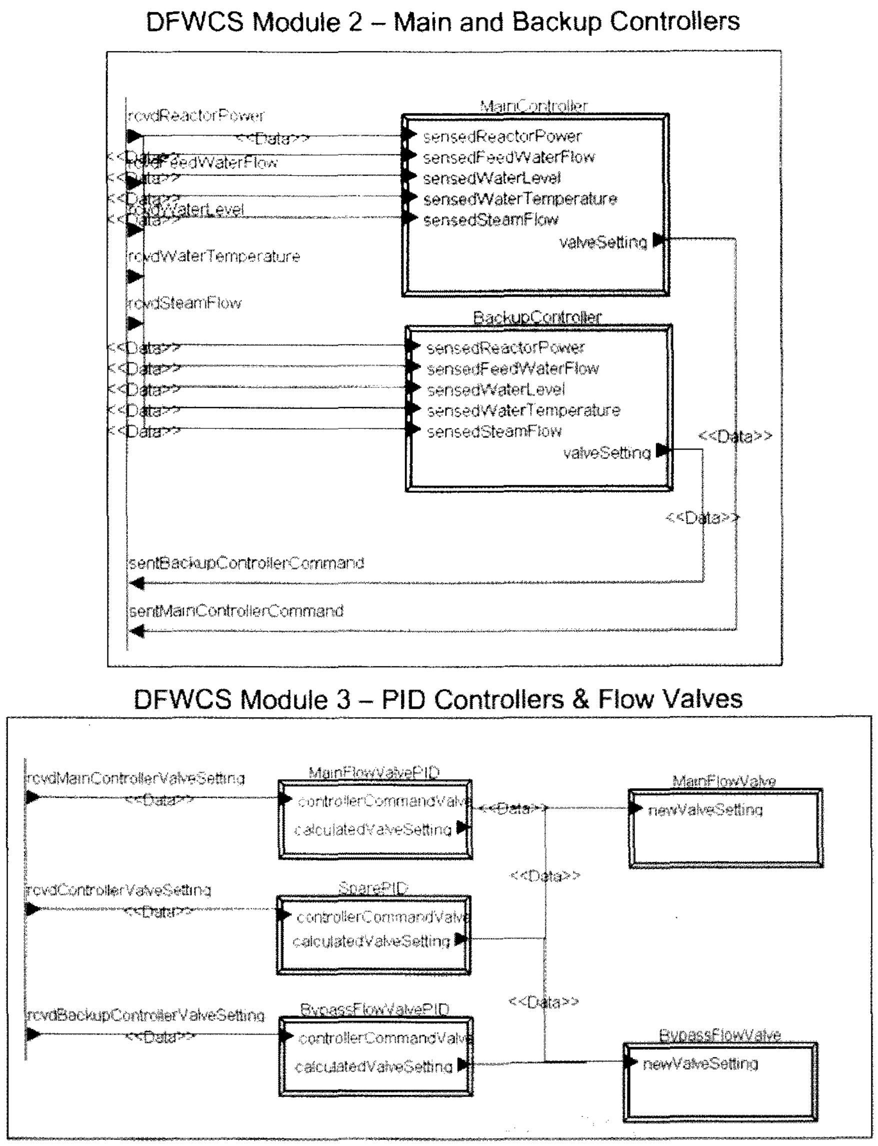

Fig. 1. Graphical AADL Model Excerpt of the DFWCS

subcomponents, connections, flows, modes and *properties*. The combination of defining the components types and implementation specification allows for the creation of a *system instance model* in AADL that is generated from the declarative models by the user indicating an AADL system implementation as the root of the system instance. AADL tooling then recursively instantiates the defined subcomponents comprising the system and the system instance is bound by identifying the component's defined binding properties (e.g., binding software threads to processors).

In this work, we utilize the OSATE [27] to specify, validate and analyze the AADL architecture models. OSATE was developed by the Software Engineering Institute (SEI) and is an open source AADL Tool Environment, built as a set of plug-ins for the Eclipse platform. OSATE provides the interface for specification of a system's architectural model (both graphical and textual interfaces), the parsing and semantic checking capabilities to ensure validate AADL specifications and the facility to generate a system's bound *System Instance Model*, an XML-based specification. OSATE was chosen for this work because it provides the necessary processing for AADL models while also enabling us to include the generation of dynamic fault trees, proposed in Section 4, into the OSATE tooling using its parsing, validation and System Instance Model generation facilities.

In Figures 1 and 2, we provide a portion of the

```
system Module3
  features
    rcvdMainControllerValveSetting: in data port;
    rcvdControllerValveSetting: in data port;
    rcvdBackupControllerValveSetting: in data port;
end Module3;

system implementation Module3.impl
  subcomponents
    MainFlowValvePID: device PID.impl;
    SparePID: device PID.impl;
    BypassFlowValvePID: device PID.impl;
    MainFlowValve: device FlowValve.impl;
    BypassFlowValve: device FlowValve.impl;
  connections
    DataConnection1: data port rcvdMainControllerValveSetting ->
MainFlowValvePID.controllerCommandValveSetting;
    DataConnection2: data port rcvdControllerValveSetting ->
SparePID.controllerCommandValveSetting;
    DataConnection4: data port rcvdBackupControllerValveSetting ->
BypassFlowValvePID.controllerCommandValveSetting;
    DataConnection5: data port
MainFlowValvePID.calculatedValveSetting ->
MainFlowValve.newValveSetting;
    DataConnection6: data port
BypassFlowValvePID.calculatedValveSetting ->
BypassFlowValve.newValveSetting;
end Module3.impl;

device PID
  features
    controllerCommandValveSetting: in data port;
    calculatedValveSetting: out data port;
end PID;

device implementation PID.impl
end PID.impl;
```

Fig. 2. Textual AADL Model Excerpt of the DFWCS

graphical and textual AADL models for the Digital Feed-Water Control System (DFWCS) [1] for a nuclear power plant example used here. Note that the graphical and textual AADL models are identical equivalent and can be generated from each other. The DFWCS system shown in Figures 1 and 2 consists of two redundant digital computer controllers, one main controller and one backup controller, that receives various sensor inputs (e.g., water temperature, reactor power, feed-water flow rate, etc.) and provides control outputs to set the operating positions of the main flow valve and bypass flow valve, and the speed of the feed-water pump. The controller outputs are first sent to a set of four Proportion, Integral and Derivative (PID) controllers of which three are dedicated to the main flow valve, the bypass flow valve and the feed-water pump and the fourth PID controller acts as a spare for either the main flow valve PID controller or the bypass flow valve PID controller. The PID controller associated with each of the controlled devices performs a comparison of the requested setting indicated by the main and backup controllers and then delivers the final outputs to the controlled device.

3.2 Modeling Faults and Failure Propagations in AADL Overview

The Architectural Analysis and Design Language's (AADL) Error Model Annex allows for the specification and analysis of potential failures and failure propagations within the architectural diagram of a critical system [6]. The Error Model Annex of AADL allows for the specification of fault models on individual components, on the interaction between fault models through fault propagations, rules determining when/how faults can propagate among different types of system components, fault filtering and/or masking failure modes and hierarchical composition of component and subcomponent fault models [14].

The AADL Error Model Annex allows for the specification of potential component faults in a generic error model, through the AADL *type* declarations, as well as for specific component *implementation* declarations. The specification of an error model for a component type declaration is appropriate for common cause failures of a component (e.g., common failure modes for a type of sensor) and typically defines the failure modes (e.g., loss of availability, loss of integrity, corrupt data, etc.), initial error states, potential error events and input/output error propagations [14]. Although not covered in this article, specified error events in the error model can additionally specify repair/reconfiguration events. The specification of an error model for a component implementation declaration is used to indicate the failure state transitions

```
system implementation Module3.impl
  annex error_model{**
      model => Module3_PID.error;
      occurrence = fixed 1E-4 applies to error fail_stop;
      occurrence = fixed 1E-4 applies to error loss_of_availability;
        guard_in =>
            mask when rcvdMainControllerValveSetting[error_free] or
rcvdBackupControllerValveSetting[error_free],
            corrupt_data when rcvdMainControllerValveSetting[loss_of_integrity] and
                  rcvdBackupControllerValveSetting[loss_of_integrity]
        applies to rcvdMainControllerValveSetting, rcvdBackupControllerValveSetting;
        derived_state_mapping =>
         error_free when MainFlowValvePID[error_free] and BypassFlowValvePID[error_free],
         error_free when MainFlowValvePID[error_free] and SparePID[error_free],
         error_free when BypassFlowValvePID[error_free] and SparePID[error_free],
         loss_of_availability when MainFlowValvePID[loss_of_availability] and
            BypassFlowValvePID[loss_of_availability],
         loss_of_integrity when MainFlowValvePID[loss_of_integritry] and
            BypassFlowValvePID[loss_of_integrity],
      report => loss_of_availability, loss_of_integrity;
   **};
  subcomponents
    MainFlowValvePID: device PID.impl;
    SparePID: device PID.impl;
    BypassFlowValvePID: device PID.impl;
    MainFlowValve: device FlowValve.impl;
    BypassFlowValve: device FlowValve.impl;
      ...
end Module3.impl;
```

Fig. 3. AADL Error Model Excerpt for the DFWCS

of a component (e.g., error free to loss of availability) and is based on the failure events and propagations defined both in the component's type and implementation declaration. Finally, failure occurrence probabilities (i.e., fixed, Poisson and/or user defined failure rates), important for our probabilistic risk assessment of the derived dynamic fault tree in Galileo, can be associated with failure events.

Figure 3 provides a small excerpt of the AADL error model specified for a portion Digital Feed-Water Control System (DFWCS) AADL model, shown in Figures 1 and 2. The **guard_in** keyword specifies the guard conditions for the input fault propagations and enables the particular component to mask the fault at the receiving interface. Following the **guard_in** property of the error model excerpt given in Figure 3, we specify the possible failure states of the component and the component's subcomponents. For example, Figure 3 specifies that, due to the presence of a spare Proportion, Integral and Derivative (PID) controller (i.e., **SparePID** in the AADL model), the **Module3** component will be error free unless two of the three PIDs fail. Finally, most relevant to this work, Figure 3 illustrates the use of the report keyword in AADL's Error Model Annex to indicate designated possible root nodes for a fault tree.

4. GENERATING DYNAMIC FAULT TREES FROM AADL MODELS

In this section, we describe our high-level approach for automatically deriving dynamic fault trees (DFT) from an Architectural Analysis and Design Language (AADL) model annotated with the appropriate/desired error models. While the process described here is similar to the approach of Joshi et al. in [14], it differs in that we generate DFTs rather than static fault trees and thus must additionally account for functional dependencies, sequence-specific failures, spares, etc. In this article, we propose the approach and plan on implementing this approach as a plug-in into the existing OSATE [27] AADL tool, described in Section 3.1, to ease the development of DFTs to be analyzed using Galileo [16-17], discussed in Section 2.2. Note that, the implementation of the DFT generation described in this section is underway.

Our approach for automatically deriving DFTs from an AADL model will consists of three high-level steps: 1. system instance error model extraction; 2. intermediate fault tree generation; and, 3. dynamic fault tree augmentation and generation. We briefly discuss each of these steps before describing the use of a DFT using Galileo.

As described in Section 3.1, the specification of an AADL model and the declaration of a system results in a System Instance Model. However, because of a deficiency in the OSATE tooling, the associated error models are not also included into the System Instance Model and thus, some preprocessing is needed to extract the error model to generated a System Instance Error Model. Within the System Instance Model, there are component instances (e.g., processors, systems, devices, etc.) and connection instances (e.g., bus connections, data connections, etc.). In both cases, an error model can be directly linked to a component and/or connection instance the Error Model Annex's **model** construct (see Figure 3) and can be retrieved directly from the instances.

If, however, there is no associated error for a connection instance, the error model of the source applies to the connection instance. For a component instance, the **derived_state_mapping**, **model_hierarchy**, and **guard_in** AADL Error Annex constructs additionally need to be retrieved to derive the proper system instance error model. For a connection instance, the **occurrence** AADL Error Annex property additionally needs to be retrieved to derived the proper system instance error model. Finally, the error propagations need to be determined for the component and connection instances. *Direct propagations* are determined as those error propagations that occur via port connections (i.e., resulting from the error model of the connection of the connected component instance. *Indirect propagations* are determined from other component instances through access connections or from binding properties (e.g., a software process bound to a processor).

Following the extraction of the System Instance Error Model, the system and its associated error model is stored in an XML-like file such that the information is stored in the nodes of a directed graph. For component instances with a derived error model, a node's edge points to all the hierarchically contained subcomponents. For component instances with abstract error models (i.e., error models defined on the component type) and connection instances, the node's edge points to all the components that may be a source of their input error propagations. Consequently, the error propagation paths can lead to potential cycles in the directed graph that result in the use of a functional dependency (FDEP) dynamic gate in a DFT.

The final step in generating a DFT from the AADL model involves identifying the root node(s) of interest and deriving the DFT from the directed graph of the previous step. As mentioned in Section 3.2, the AADL Error Model Annex provides the **report** keyword to denote error events that should be reported. For example, in the Digital Feed-Water Control System (DFWCS) error model shown in Figure 3, a candidate DFT root node could be "Module3 loss of availability" to indicate that the PID controllers, components of Module3, failed to command the water flow valves.

These high-level steps will enable the generation of a DFT that, after being properly formatted to be compatible with Galileo, can be loaded and can be analyzed to estimate the critical system's probabilistic risk assessment. The following section describes the use of Galileo for analyzing DFTs.

5. ANALYZING DYNAMIC EVENT/FAULT TREES USING GALILEO

In this section, we provide an overview of the analysis capabilities of the dynamic fault tree (DFT) and Galileo that motivate us to develop the ability to automatically generate DFTs from a critical system's architecture model. First, however, we examine the dynamic event/fault tree (DEFT) as a guide in developing the error models for the architecture model specified using the Architectural Analysis and Design Language's (AADL) [6] [20] Error Model Annex and how the DEFT can guide the probabilistic risk assessment of a critical system. Note that readers interested in a full discussion of these dependability engineering techniques and tools should consult [2] [4] [13] for DFT analysis, [16-17] for further information on Galileo and [15] for DEFT analysis, respectively.

5.1 Modeling System Faults/Failures using Dynamic Event/Fault Trees

The dynamic event/fault tree, described in Section 2.2, combines the use of event trees [15] and dynamic fault trees (DFT) [2] and can be used as a part of a critical system's probabilistic risk assessment (PRA). Event trees and DFTs compliment each other and their combination, the DEFT, better captures dynamic system events [2]. Further, we found that the development of a DEFT aided in the identification of possible system component failure modes and the development of the error models for the system architecture.

Using the Digital Feed-Water Control System (DFWCS), described in Section 3 and specified in Figures 1-3, Figure 4 illustrates a DEFT for the initiating event (IE) "module2 sends new flow valve setting to module3" to represent the event of the DFWCS when the main and backup controllers send a flow valve setting to the flow valve Proportion, Integral and Derivative (PID) controllers (see Figure 1). This IE is of interest to the DFWCS and was developed as a DEFT to investigate the possible failures, and their potential results, for the critical function of setting the water flow valves as commanded by controllers and PID controllers.

The resulting DEFT illustrates the pivot events (PE) and the possible outcomes of the IE. A PE of the DEFT is an event that is the occurrence or non-occurrence of the event and indicates the root node of a possible fault tree (denoted as DFT1-DFT5 in Figure 4). From the IE, the DEFT branches into two (or more) outcomes, denoted as PE1: "received controller data comparison" to mark the event that the flow valve PIDs received the commands and data from the controllers and will compare them to ensure integrity and accuracy. The resulting paths are then repeatedly evaluated against other PEs that can occur within the system, marked as PE2-PE5 in Figure 4, to guide the analysis of a systems critical functionality. This process results in a set of possible outcomes that may be failure or non-failure system states.

The development of the DEFT for the DFWCS utilized the Architectural Analysis and Design Language (AADL) model, shown in Figure 1, to identify the system components, the data/event connections and the data/event

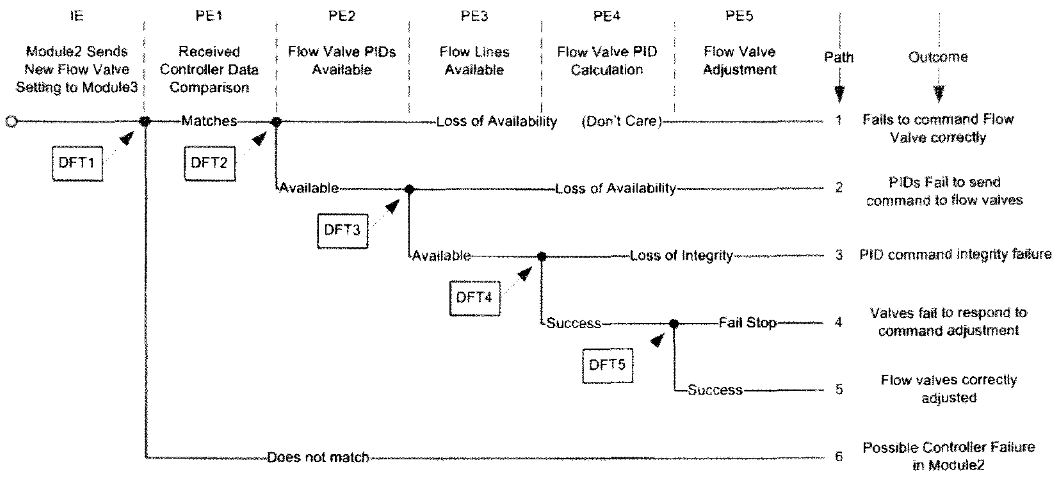
IE
Module2 Sends New Flow Valve Setting to Module3
PE1
Received Controller Data Comparison
PE2
Flow Valve PIDs Available
PE3
Flow Lines Available
PE4
Flow Valve PID Calculation
PE5
Flow Valve Adjustment
Path
Outcome
Matches
Loss of Availability
(Don't Care)
1
Fails to command Flow Valve correctly
DFT1
DFT2
Available
Loss of Availability
2
PIDs Fail to send command to flow valves
DFT3
Available
Loss of Integrity
3
PID command integrity failure
DFT4
Success
Fail Stop
4
Valves fail to respond to command adjustment
DFT5
Success
5
Flow valves correctly adjusted
Does not match
6
Possible Controller Failure in Module2

Fig. 4. DEFT Excerpt for the DFWCS

flows (not discussed in this paper due to space constraints) involved in the IE and PEs. The development of the DEFT using the AADL model can help guide the development of the AADL component's error model specifications in that the PEs identify possible error states (e.g., loss of availability, loss of integrity, fail stop, etc.) for the components of the system and provides a high-level view of the failure propagations that can then be specified in the AADL error model. Further, the PEs of the DEFT provide possible events that could be developed as a fault tree and may indicate which failure states for a system component warrants the use of the **report** keyword of AADL, discussed in Sections 3.3 and 4, and the possible generation of a DFT.

Figure 5 provides an excerpt of the DFT developed from the DEFT of the DFWCS shown in Figure 4. Specifically, the DFT models the events PE2 (right subtree of Figure 5) and PE3 (left subtree of Figure 5) and denoted at DFT2 and DFT3 in Figure 4, respectively. The resulting DFT describes the causal failure events that can lead to the "loss of availability" failure mode of Module3, as indicated as the desired DFT root node in the AADL error model shown in Figure 3. Similarly, the DFT is the result of the analysis of the system's AADL architecture, shown in Figures 1 and 2, using the process described in Section 4. Thus, the leaf nodes in the DFT for the DFWCS coincide with the failure modes described in the AADL error model. For example, the DFT leaf node "MainFlowValve" of Figure 5 results from a "fail stop" error state described the **MainFlowValve**'s AADL component error model.

5.2 Estimating and Improving the Probabilistic Risk Assessment of a Critical System

The capability to indentify the critical events, data and components of a dependability-critical system, such as the Digital Feed-Water Control System (DFWCS) used in this article, is essential to: 1. discover unanticipated or unknown critical system components and data/events that may lead to undesirable and possible unsafe system outcomes; 2. enable the greater understanding of the behavior, interaction, data and functionality of system components specified in the Architectural Analysis and Design Language (AADL) model; and, 3. determine and/or verify failure masking/mitigation strategies to quantifiably improve the critical system's reliability.

The dynamic event/fault tree (DEFT) shown in Figure 5 presents five pivot events (PE) that can be associated with DFTs to model and analyze the potential failures contributing to the DEFT outcomes (see Figure 4). For quantitative analysis, the DFTs modeled for the PEs in the DEFT provide the probability of an outcome at a given PE (ignoring the common cause failures, dependences between the DFTs and imperfect DFT failure coverage). For example, the probability of the loss of availability of the Flow Valve Proportion, Integral and Derivative (PID) controllers, shown in Figures 1 and 2, can be calculated from the associated DFT. The development of the DEFT along with the specification of a critical system's architecture and failure model, as described in Section 3, and buttressed by the capability of deriving the DFT directly from the AADL model, described in Section 4, will better enable the use of Galileo [16] [17] to calculate the probabilities of each PE, if desired, in the DEFT. By doing so, we can estimate the probability of each outcome in the DEFT as the products of each PE outcome along its path [2].

The capability of estimating the DEFT's outcome probabilities, assisted by Galileo, allows for dependability engineers to: 1. better partition resources into which DEFT system outcomes should be further investigated/analyzed using alternate dependability engineering techniques and tools; 2. a quantifiable estimate of the possible outcomes

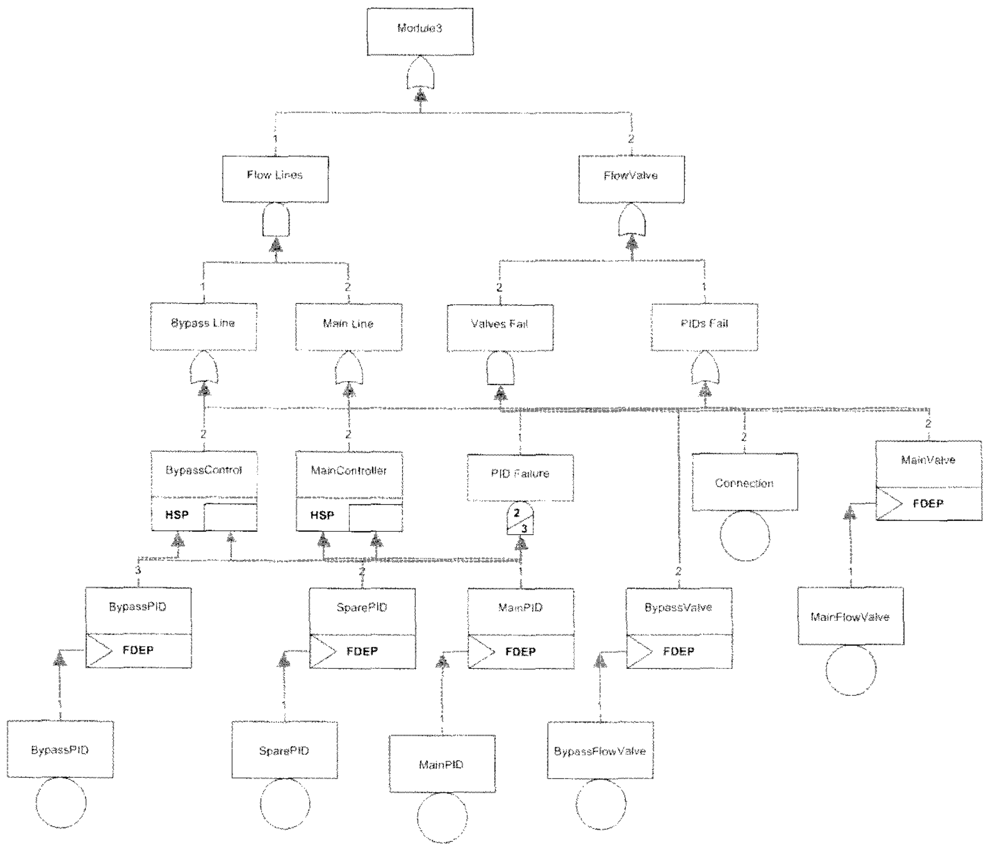

Fig. 5. DFT Excerpt for the DFWCS from the AADL Model

of an initiating event (IE); and, 3. document the evidence of failure outcomes and analysis given initial faults when certification and/or a dependability audit is required.

6. CONCLUDING REMARKS AND FUTURE WORK

This article advocated and demonstrated for the use of the Architectural Analysis and Design Language (AADL) as a medium to model the architecture and faults and failure propagations of a dependability-critical system and as a mechanism to more accurately reflect the actual system design and architecture in the dependability analysis. To support this, we outlined our approach to enable the generation of dynamic fault trees (DFT) from AADL models annotated with associated error models to enable probabilistic risk assessment of the DFTs using the Galileo tool. In doing so, the main insights of this article include:

· An illustration of how AADL can be utilized to specify and model the structural, behavioral and failure characteristics of a dependability-critical system composed of software and hardware components
· A description of our approach to enable the automatic derivation of DFTs from the specified AADL model
· An overview of the dependability analysis facilities of using Galileo for a probabilistic risk assessment of a critical system using a dynamic event/fault tree and DFTs

The intention of this work is to assist in closing the dependability engineering – system engineering gap by integrating dependability engineering into the design and development process of a critical system and enabling a more formal, automated and consistent DFT analysis.

While this is an important step in bridging the gap between system engineering and dependability engineering to produce better dependability assessments of the actual designed system, additional work following this article remains to be addressed and explored. In particular:

· Implementation and demonstration of the tooling described in Section 4 to enable the automatic generation of DFTs from AADL models and their use with Galileo
· Development of an intuitive interface, such as the use of a Failure Modes, Effects and Criticality Analysis [15], for dependability engineers to ease the specification of the AADL's error model to blunt the needed AADL domain expertise
· Investigation on how additional dependability engineering

techniques and tools can be used in conjunction with AADL; in particular, those techniques based on Markov Chains as they are compatible with AADL and could be generated from the derived System Instance Error Model, described in Section 4

ACKNOWLEDGEMENTS

This research was supported by the Charles L. Brown Department of Electrical and Computer Engineering at the University of Virginia.

REFERENCES

[1] Y. Yu. The Quantitative Safety Assessment for Safety-Critical Computer Systems. *PhD Thesis*, University of Virginia, 2006.

[2] J. B. Dugan, G. Pai and H. Xu. Combining Software Quality Analysis with Dynamic Event/Fault Trees for High Assurance Systems Engineering. In *Proceedings 10th IEEE High Assurance System Engineering Symposium*, pp. 245-255, Dallas, TX, 2007.

[3] M.C. Kim and P.H. Seong, "Reliability Graph with General Gates: An Intuitive and Practical Method for System Reliability Analysis", *Reliability Engineering and System Safety*, vol. 78 pp. 239-246, 2002.

[4] G. J. Pai and J. B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML System Models. In *Proceedings of 13th International Symposium on Software Reliability Engineering*, Annapolis, MD, pp. 243-256, 2002.

[5] Z. Tang and J. B. Dugan. An Integrated Method for Incorporating Common Cause Failures in System Analysis. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pp. 610-614, 2004.

[6] P. Feiler and A. Rugina. Dependability Modeling with the Architecture Analysis & Design Language (AADL). Available from http://www.sei.cmu.edu/pub/documents/07.reports /07tn043.pdf (Accessed June 2008).

[7] A. E. Rugina, K. Kanoun and M. Kaâniche. An Architecture-based Dependability Modeling Framework Using AADL In *Proceedings International Conference on Software Engineering and Applications*, Dallas, TX, 2006.

[8] A. E. Rugina, K. Kanoun and M. Kaâniche. A System Dependabiliy Modeling Framework using AADL and GSPNs. In *Architecting Dependable Systems* 4th Volume, R. de Lemos, C. Gacek and A. Romanovsky, eds., Springer, 2007.

[9] T. Assaf and J. B. Dugan. A Probabilistic Expert System for Failure Diagnosis. In *Proceedings Reliability and Maintainability Symposium*, 2004.

[10] T. Assaf and J. B. Dugan. Automatic Diagnosis via Sensors Modeled by Dynamic Fault Trees. In *Society of Automotive Engineers Transactions*, 2005.

[11] T. Assaf and J. B. Dugan. Diagnostic Decision Trees based on Estimating Diagnostic Importance Factors from Markov Models. In *IEEE Instrumentation and Measurement Magazine*, 2005.

[12] T. Assaf and J. B. Dugan. Diagnostic Expert Systems from Dynamic Fault Trees. In *Proceedings of the Annual Reliability and Maintainability Symposium*, LA, pp. 444-450, January 2004.

[13] J. B. Dugan and T. S. Assaf. Dynamic Fault Tree Analysis of a Reconfigurable Software System. In *Proceedings of the 19th International System Safety Conference*, Huntsville, AL, pp. 480-487, 2001.

[14] A. Joshi, S. Vestal, and P. Binns. Automatic Generation of Static Fault Trees from AADL Models. In *DSN Workshop on Architecting Dependable Systems*, Edinburgh, Scotland-UK, 2007.

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

[16] D. Coppit, K. J. Sullivan and J. B. Dugan. Formal Semantics of Models for Computational Engineering: A Case Study on Dynamic Fault Trees. In *Proceedings of the International Symposium on Software Reliability Engineering*, San Jose, CA, pp. 270-282, 2000.

[17] J. B. Dugan, K. J. Sullivan, and D. Coppit. Developing a Low-Cost, High-Quality Software Tool for Dynamic Fault Tree Analysis. In *IEEE Transactions on Reliability*, 49(1): 49-59, 1999.

[18] R. Allen, S. Vestal and B. Lewis. Using an Architecture Description Language for Quantitative Analysis of Real-Time Systems. In *Proceedings of the 3rd International Workshop on Software and* Performance, Rome Italy, pp. 203-210, 2002.

[19] P. Dissaux. Using the AADL for Mission-Critical Software Development. In *Proceedings of 2nd European Congress on Embedded Real-Time Software*, Toulouse, France, 2004.

[20] P. Feiler, D. P. Gluch and J. J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Available from http://www.sei.cmu.edu/pub/documents /06.reports/pdf/06tn011.pdf (Accessed June 2008).

[21] SAE-AS5506. Architecture Analysis and Design Language Annex Volume 1. SAE, June 2006.

[22] SAE-AS5506/1. Architecture Analysis and Design Language Annex Volume 1. SAE, June 2006.

[23] D. Srivastava and P. Narasimhan. Architectural Support for Mode-Driven Fault Tolderace in Distributed Applications. In *Proceedings of the 2005 Workshop on Architecting Dependable Systems*, St. Louis, MO, pp.1-7, 2005.

[24] F. Singhoff, J. Legrand, L. Nana and L. Marce. Scheduling and Memory Requirements Analysis with AADL. In *Proceedings of the 2005 Annual ACM SigAda International Conference on Ada: The Engineering of Correct and Reliable Technologies*, Atlanta, GA, pp. 1-10, 2005.

[25] J. Dehlinger and R. R. Lutz. PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool. In *The Automated Software Engineering Journal*, 13(1):169-193, 2006.

[26] H. Sun, M. Hauptman and R. Lutz. Integrating Product-Line Fault Tree Analysis into AADL Models. In *Proceedings of the 10th IEEE International Symposium on High Assurance System Engineering*, Dallas, TX, pp. 15-22, 2007.

[27] SEI AADL Team. An Extensible Open Source AADL Tool Environment (OSATE). [Online]. Available from http://la.sei.cmu.edu/aadl/downloads/osate13/AADLTool UserGuide1.3.0%202006-06-02.pdf (Accessed June 2008).