On Automating Failure Mode Analysis and Enhancing its Integrity*

Kam S. Tso Ann T. Tai
IA Tech, Inc.
Los Angeles, California

Savio N. Chau Leon Alkalai
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California

Abstract

This paper reports our experience on the development of a design-for-safety (DFS) workbench called Risk Assessment and Management Environment (RAME) for microelectronic avionics systems. RAME is built upon an information infrastructure that comprises a test-reporting/failure-tracking system, an off-the-shelf data mining tool, a knowledge base, and a fault model. This infrastructure permits systematic learning from prior projects and enables the automation of failure mode, effect and criticality analysis (FMECA). More importantly, RAME is able to directly accept source code in hardware description languages (HDLs) for automated design validation.

1 Introduction

Risk identification and mitigation are essential in design for safety, especially for microelectronic avionics systems. However, the widely used, traditional practice of failure mode, effect, and criticality analysis (FMECA) that relies on manual, textual-document manipulation is often ambiguous, inconsistent, and error-prone. Those shortcomings are primarily due to that those analyses rely heavily on the knowledge and experience of individual system/quality-assurance engineers who perform FMECA worksheet generation. Furthermore, the manual production of FMECA documents can absorb a significant amount of time during the system design stage, and is unable to provide system engineers with timely feedback for design modifications. For example, the FMECA worksheet for the System Input/Output (SIO) board that implements the fault-tolerant bus interface of an avionics system took 6 person-months to complete at JPL.

As the increased device complexity and reduced development cycle time collectively have made FMECA automation highly desirable, a number of commercial tools were developed facilitate the time-consuming process of performing FMECA (see [1, 2], for example). Nonetheless, the capabilities of those tools are generally limited to automating the process of organizing data, providing a graphical interface for users to report their analysis results, or formating the reports according to Military Standards. More specifically, the major inadequacy of those tools is the lack of an ability to automate the most crucial part of FMECA, namely, the time-consuming and error-prone process of identifying failure modes, causes, and effects.

With the motivation of transforming design for safety (DFS) practice from a traditional ad-hoc process that relies on error-prone, textual-document manipulation to a stringent engineering process, we develop a DFS workbench called the Risk Assessment and Management Environment (RAME). RAME comprises a design-source-code static analyzer, a FMECA automation engine, a fault model, a knowledge base, a test-reporting and failure-tracking system (TRFTS), and a data mining tool (DMT-WEKA). The last two components together enable the knowledge base and fault model to be built and kept up-to-date along a project's life cycle and across successive projects, constituting a closed loop that enables RAME to evolve.

As we view consistency, completeness, and accuracy collectively as failure mode analysis *integrity*, the specific objective of the RAME project is to improve both integrity and turnaround-time of design validation. In addition, RAME is intended to relieve system engineers from the tedious and error-prone process of manually generating FMECA worksheets and to enable them to gain time for design inadequacy mitigation.

The remainder of the paper is organized as follows. Section 2 presents the information infrastructure of RAME. Section 3 describes our approach to FMECA automation. Section 4 reports an initial experimental evaluation of the RAME prototype using the actual design source code of JPL's SIO board. Section 5 provides concluding remarks.

2 Information Infrastructure

2.1 RAME Architecture

As shown in Figure 1, the architecture of RAME comprises six components which together enable high-integrity fail-

*The work reported in this paper was supported in part by Small Business Innovation Research (SBIR) contract NAS3-02096 from the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

ure mode analysis. In particular, the VHDL[1] static analyzer and FMECA automation engine carry out failure analysis, with the crucial support from the information infrastructure consisting of a fault model, a knowledge base, TRFTS, and DMT-WEKA. More specifically, TRFTS allows engineers to report test results and track failures based on a unified taxonomy and terminology, serving as the information source for the knowledge base and fault model, while DMT-WEKA extracts information concerning failure modes, causes, and effects from TRFTS for building and maintaining the knowledge base and fault model.

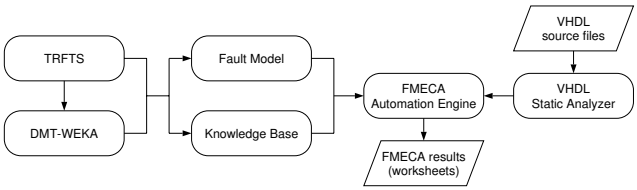

Figure 1: RAME Architecture

The goal of the information infrastructure is to 1) enable high-integrity FMECA automation by avoiding inconsistency, omission, and miscommunication, and 2) facilitate design, test, and reliability engineers to collaborate on risk assessment and management throughout a system's life-cycle.

Within the information infrastructure, the fault model defines the possible failure modes associated with each device type and the probable causes of a failure mode with which a device fails.

The knowledge base supplies information necessary for interpreting the terms used in the VHDL design source code under analysis and thus to enable FMECA automation. Knowledges encompass device types, signal types, and their characteristics that must consider for failure mode analysis. For example, in helping identify signal types during analyzing an interface board, the knowledge base will make the FMECA automation engine be aware that certain signals are bus signals so that each of them must be analyzed as an array.

2.2 Development of TRFTS

TRFTS is developed for recording, updating, and managing the reports concerning anomalies that are revealed during testing or simulation of avionics systems prior to launch. TRFTS is also the front end that enforces the use of the unified taxonomy and terminology for reporting failures. In turn, this ensures correct data exchanges among all the existing components in RAME and permits additional constituent tools to be integrated into RAME without violating analysis integrity.

Specifically, test reports are be stored in XML format to enable knowledge discovery from test and simulation data. TRFTS consists of several server-side components implemented on the Java 2 Enterprise Edition platform. As XML schemas are designed to express shared vocabularies and to allow machines to interpret rules defined by the designer, they are heavily exploited in TRFTS for the specifications of failure report contents, structures, constraints, and semantics.

2.3 Data Mining: Building and Enriching Knowledge Base and Fault Model

The data mining component in RAME's information infrastructure employs the open-source tool WEKA[2] to process and visualize data, and to extract knowledges from TRFTS. WEKA contains constituent tools for data pre-processing, summarization, classification, regression, clustering, association, and visualization.

In particular, we employ WEKA for data summarization, which is the abstraction and generalization of data, to aggregate the information in TRFTS concerning failures, causes and effects. Moreover, we let WEKA carry out data association that discovers the relationships between data items. For example, to relate failure modes to device types and to associate probable causes to failure modes. Clearly, association analysis is the most essential data mining mechanism for automating FMECA.

Further, we regard certain data items available in TRFTS (including device types, failure modes, probable causes) collectively as the universe and let each item type have a Boolean variable indicating its presence or absence in individual reports. Thus each test report can be marked by a vector of Boolean values that identifies items that frequently appear together. These patterns can then be translated into the form of *association rules*. As a simple example, the pattern in which a resistor may fail in an "open" failure mode which is usually caused by "cracked solder joint" can be translated into the following rule:

```
device=RESISTOR failure=OPEN ==>
                cause=CrackedSolderJoint
[support = 20%, confidence = 90%]
```

Levels of *support* and *confidence* are two metrics of rule validity. They respectively reflect the usefulness and certainty of discovered rules. A support of 20% for the above association rule means that 20% of all the test reports undergoing data mining show that `CrackedSolderJoint` is the probable cause of the `OPEN` failure mode of `RESISTOR`.

[1] I.e., VHSIC hardware description language, see Section 3.1 for a brief introduction.

[2] WEKA stands for Waikato Environment for Knowledge Analysis, and was developed at the University of Waikato in New Zealand.

A confidence of 90% means that 90% of the test reports that state that `RESISTOR` has the `Open` failure mode indicate that the failure mode is caused by `CrackedSolderJoint`.

3 FMECA Automation

3.1 General Approach

Over the past few years, hardware description languages (HDLs) have been widely used in electronic design automation (EDA). Two HDLs, namely, VHDL and Verilog have become the dominant *de facto* industry standard HDLs. VHDL offers a broad set of constructs for describing even the most complicated logic in a compact fashion. It can be used to model hardware systems and components, from gate level to system level.

Similar to VHDL, Verilog also describes hardware systems using structure and behavior models. Nowadays, most ECAD tools support both VHDL and Verilog [3]. Although our development is based on VHDL, all the techniques and tools implemented in RAME, including the FMECA automation engine, are compatible to both VHDL and Verilog.

As illustrated in Figure 2, our approach to automating FMECA is based on 1) the structural and behavioral information embedded in the VHDL source files, and 2) the knowledge base and fault model in the RAME information infrastructure. In particular, the VHDL source files are first processed by the VHDL Static Analyzer, which parses the design source code according to the formal grammar of VHDL. If no syntax or type-check error is found, the analyzer generates an *abstract syntax graph* which enables FMECA automation.

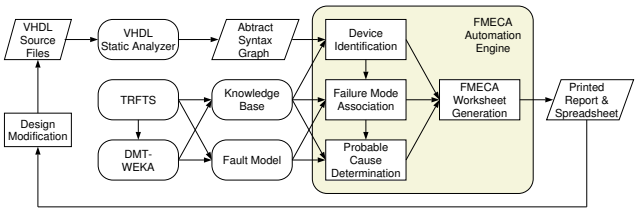

Figure 2: FMECA Automation

From the structural perspective, the FMECA automation engine traces the signals from the outer layer of the system under analysis (e.g., a connector of the SIO board) to the inner layer (e.g., an ASIC inside the SIO board), and from the higher level (e.g., the board level) to the lower level (e.g., the chip level). In particular, a recursive trace is carried out based on the architectural information encoded in the VHDL files. The trace stops when it reaches "terminals" such as power, ground, connector, or an entity that has no further elaboration (e.g., an IP-based ASIC which does not have a VHDL source file revealing its architecture specification).

Through the trace, the FMECA automation engine collects all the devices along the paths. It subsequently exploits the information in the knowledge base and fault model to generate the FMECA worksheet.

3.2 VHDL Static Analyzer

RAME's VHDL static analyzer (VSA) is able to analyze IEEE VHDL-93 compliant designs, such as JPL's SIO board design (mentioned earlier and described in Section 4). VSA consists of two components, namely, a *lexical scanner* and a *syntax interpreter*.

The output of the lexical scanner is a stream of tokens which is subsequently passed to the syntax interpreter. According to the VHDL grammar, the syntactic interpreter organizes the tokens to build syntactic structures. For example, the three tokens "`TDI`", "`=>`", and "`MSIO_A_TDI`" are grouped into a syntactic structure called an *expression*. Expressions are in turn combined to form statements, etc. Eventually, those syntax structures constitute an *abstract syntax tree* whose leaves are the tokens produced by the lexical scanner. More specifically, the abstract syntax tree is represented by a collection of C++ objects tightly connected by pointers. With the tree structure, the objects can then be accessed by a set of C++ methods for FMECA automation.

3.3 FMECA Automation Engine

The FMECA automation engine (FAE) automates the otherwise labor-intensive process of performing failure mode, effect and criticality analysis of hardware designs, by exploiting the results from VSA and the information infrastructure.

Device Identification

The first step of the FMECA automation is to identify all the devices (i.e., the structure primitives at a given abstraction level) whose failures may affect system operation. In VHDL, a module (which could be an ASIC, a board, a subsystem, or a system) is described using an *entity declaration* which specifies the interface in terms of *ports*, the connecting points for input and output signals. FAE traces the signal one by one from the outside of a module to its inside until it cannot be traced any further, that is, the trace reaches the power, ground, connector, or other types of termination entity. But when FAE encounters a device which does have a VHDL design file, FAE will trace down one level further. Through the trace, FAE collects all the devices along the path and constructs a list of devices relating to that signal.

Failure Mode Identification

After a list of devices is established, FAE relates particular potential failure modes to each of the devices, based on the association rules for devices and failure modes that are maintained in the fault model.

Another piece of information FAE exploits is the operational mode of each signal. The VHDL declaration of each signal includes its operational mode, such as `in`, `out`, and `inout`. This information is necessary for FAE to determine the possible failure modes of a particular device.

Probable Cause Identification

The probable causes of failure modes are extracted from TRFTS and stored in the fault model. From TRFTS, DMT-WEKA also produces the association rules for the fault model which relate the probable causes to a failure mode of a particular device type. Accordingly, based on the fault model, FAE is able to determine the probable causes for the failure modes of every identified device.

Multi-Level Trace: Failure-Mode Effect Identification

An avionics system is usually designed as a hierarchical collection of modules. Hence ECAD tools that have the ability to generate hierarchical models are increasingly used for design specification, which results in a trend that VHDL source files become available at not only the chip or board levels but also the subsystem and system levels. For example, the ECAD tool I-Logix Statemate Magnum which is able to generate hierarchical system models in VHDL and Verilog is increasingly used for avionics architecture design at JPL [4]. Accordingly, it is important to let FAE have the ability to trace the effects of potential failures across different levels of a design.

Specifically, FAE uses a recursive algorithm to perform top-down trace of failure effects. When a signal is traced to an entity which is a VHDL structure, FAE loads the VHDL file of that entity and traces down the signal through that structure. Figure 3 shows an example in which the trace starts from the connector `CONN1394` at the SIO board level and ends at the point when the `MSIO_A` ASIC is reached.

Note that when the trace reaches the terminator `MGC1394_A_TERM` (in Segment 2 in the path of the trace), FAE recognizes that it is a structure specified at one abstraction level below (i.e., the chip level next to the board level) and thus loads the `MGC1394_A_TERM.vhd` file to trace the internal structure of the entity `MGC1394_A_TERM`. FAE then reaches the resistor `RM1005_55ohm` and the capacitor `CDR33_0_0461uF_50V` at the chip level. As shown in Figure 3, the trace of Segment 2 branches into Segments 3 and 4 which are in the path of the trace for the a chip-level structure in the VHDL hierarchical design model. After the chip-level trace, FAE comes back to Segment 2 at the board level and reaches the `MSIO_A` ASIC. To this end, FAE stops that portion of the trace by treating `MSIO_A` as a termination entity, because it is defined as a `black_box` in the VHDL design source code for the SIO board. That means, there is no further architectural information about the entity available in the VHDL design source code (MSIO_A is an IP-based ASIC for which no design details are supplied).

```
Segment 0: SIO_x_036X2000->A0_TPA
Segment 1: CONN1394->A0_TPA
Segment 2: MGC1394_A_TERM->A0_TPA
Segment 3: RM1005_55ohm->MGC0_TPA
Segment 3: RM1005_55ohm->local_TPBias0
Segment 4: CDR33_0_0461uF_50V->local_TPBias0
Segment 4: CDR33_0_0461uF_50V->local_1394_3_0463VGND
Segment 2: MGC1394_A_TERM->local_1394_3_0463VGND
Segment 5: MSIO_A->A0_TPA
```

Figure 3: Tracing of the `A0_TPA` Signal

This recursive algorithm will exhaustively collect the information for FAE to analyze the effects of a failure mode of a particular device or an entity of another type. The trace is carried out in a top-down manner, starting from an entity e at the top level and going down level by level successively until reaching a termination entity. Along the path of the trace, all the entities that e has dependency with are entered into a tree-type data structure. In addition, potential failure modes of each entity and their probable cause are also identified per the fault model and saved/organized in that data structure.

Upon the completion of the trace for every top-level entity, FAE searches through the information organized in a collection of the tree-type data structures (each of which corresponds to a particular top-level entity) in a bottom-up manner, summarizes the failure modes (a collection denoted as F) of the top-level entities that are resulting from a failure mode f of a bottom-level device d, and concludes that F is the top-level effects of the failure mode f of device d. Finally, the criticality of f is determined by F's ranking.

From a view point similar to that suggested by [5], the usage of the above algorithm can be generalized. More succinctly, a failure mode f_n of an entity e at design abstraction level n will be 1) an effect of a failure mode f_k of an entity e' at level k ($k \leq n-1$), and 2) a cause of a failure mode f_m of an entity e'' at level m ($m \geq n+1$), if the trace results combined with the information in the knowledge base and fault model imply such dependencies.

4 Experimental Evaluation of RAME Prototype

In order to validate the RAME prototype, we carry out an initial experimental evaluation by applying the prototype to an actual VHDL design, namely, the SIO board which implements a fault-tolerant bus interface. Among other objectives, this initial experimental evaluation aims to validate the following:

Data consistency: To validate the effectiveness of the enforcement for the use of a unified taxonomy and terminology.

Completeness: To examine whether FAE is able to exhaustively identify the failure modes of a VHDL design.

Accuracy: To examine whether FAE is able to produce correct entries for the failure mode of each device and for the probable causes of those failure modes.

Turnaround time: To examine whether FAE can significantly reduce the time required to perform a design validation.

4.1 Overview of SIO Board

The SIO board selected for the RAME prototype evaluation implements the IEEE 1394 and I2C bus interfaces for connecting the flight computers, microcontrollers in the X2000 avionics system, and their internal PCI bus [6]. The architectural structure of the board is illustrated in Figure 4.

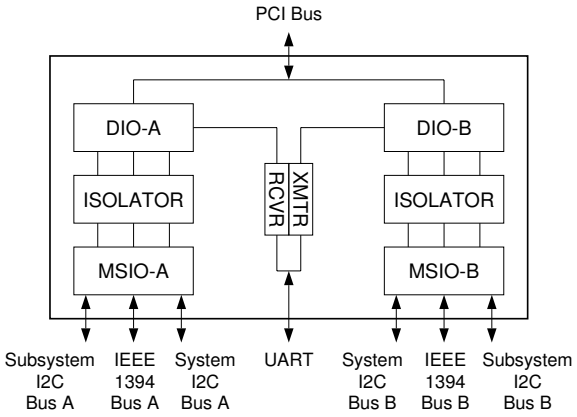

Figure 4: Architectural Structure of the SIO Board

As shown in the figure, the SIO board consists of two redundant bus interface units, each of which has one 3-port IEEE 1394 bus interface and two I2C bus interfaces. Each bus interface unit is composed of two ASICs, namely, the Digital I/O (DIO) ASIC and the Mixed Signal I/O (MSIO) ASIC. The DIO ASIC implements the link layer of the IEEE 1394 bus, the two I2C bus controllers, and the logic for fault tolerance mechanisms of the buses; the MSIO ASIC serves as the physical layer in the overall data-communication architecture and implements an analog interface to the IEEE 1394 and I2C bus cables.

4.2 Experimental Evaluation

To evaluate the effectiveness of RAME's enforcement of data consistency, we enter into TRFTS the data from testing and simulation (for the SIO board). The reports are then processed by TRFTS for DMT-WEKA to create the knowledge base and fault model (which enable FAE to analyze the SIO VHDL files and to generate the FMECA worksheet). FAE provides two options for the types of output documents, namely, 1) *signal-list worksheet* which is generated based on a list of signals, and 2) *port-list worksheet* for which FAE analyzes a VHDL modules by tracing the signals based on a list of ports.

We conduct the experimental evaluation on a PC/Linux platform in which the PC is a Dell Dimension 8200 desktop computer with a 2.8 MHz Pentium 4 processor and 1 GB of memory running Red Hat 9 Linux. The SIO design source code consists of 14 VHDL files with a total of 17,854 lines of code.

4.3 Discussion of Results

We exercise both the output options explained earlier for the experimental validation. The analyses using signal-list and port-list options both yield a FMECA worksheet of 872 lines in 1.27 seconds.

In order to evaluate consistency, completeness, and accuracy of FMECA automation, we compare the worksheets produced by FAE to that generated manually (using the signal list) at JPL. The comparison reveals that while the manually generated worksheet has 689 items, both the signal-list and port-list worksheets generated automatically for the same VHDL design source code have 872 items. By carefully inspecting all the entries of the worksheets and the VHDL design source code, we identify the underlying causes of the discrepancy as follows:

1. The manually generated worksheet missed many internal signals. For example, in analyzing the `UARTA_SIN_P` signal, the manually generated worksheet only has three items for the "open", "shortVCC", and "shortGND" failure modes. But the `UARTA_SIN_P` signal connects to the `SIN` pin of the DIO ASIC chip after it passes through the driver `26CLV32RH`. As a result, there should also be "open", "shortVCC", and "shortGND" failure modes for the `SIN` signal at `DIO_ASIC_A`, as identified by FAE. Since there is a

total of 28 such signals, the automatically generated worksheet has 84 more entries.

2. The manually generated worksheet omitted the trace for both the paths in the IEEE 1394 terminators. Specifically, in analyzing the `A0_TPB` signal, that worksheet only considered the path of the 55ohm resistor and 270pF capacitor. It was neglected that in an IEEE 1394 terminator there is another path in which the 55ohm resistor connects to a 5K resistor (i.e., the part `RM1005_5K`). If this 5K resistor fails and becomes open, the effectiveness of the terminator will degrade. Hence, the FAE-generated worksheet has one more item to take into account the failure of the 5K resistor. Indeed, besides `A0_TPB`, there are many such signals passing through the terminators in the design. Consequently, the negligence amplifies the discrepancy between the manually and automatically generated worksheets.

3. For analyzing the power and ground signals, the manually generated worksheet ambiguously uses one item (i.e., "Any single capacitor between `1394A_3.3V` and `1394A_3.3VGND`") for all the capacitors connecting the 1394A power and ground, while the automatically generated worksheet enumerates exhaustively all the capacitors whose failures will affect the signal.

4. For analyzing the `MSIO_A_TMS` signal, the automatically generated worksheet identifies a 7K resistor (i.e., the device `RM1005_7K`) while the manually generated worksheet considers it as a 10K resistor. By inspecting the VHDL design source code, we confirm that the result from FAE is correct.

5. There are two signals (i.e., `P3.3V` and `P3.3VGND`) that appear in the manually generated worksheet are absent in the worksheet from FAE. By carefully examining both the port list of the VHDL design source code and the schematic diagram, we confirm that those two signals are not designated as the ports of the SIO board. The mistake in the manually generated worksheet was indeed a result of failing to make corresponding update in the worksheet when a design modification eliminated those two signals.

Aside from those differences, the worksheet generated by FAE is the same as the manually generated worksheet. Accordingly, as described above, automated FMECA performs significantly better than the manual FMECA, with respect to consistency, completeness, and accuracy. In addition, in terms of turnaround time, automatic generation for each of the two worksheets (using signal list and port list) takes less than 1.5 seconds, while the manual generation of the worksheet for the same VHDL design took multiple person-months. Clearly, FMECA automation will lead to a significant improvement of design validation efficiency.

5 Concluding Remarks

In summary, RAME is distinctive from those previously developed failure mode analysis tools in several respects as follows. First, none of those previously developed tools directly accepts HDL design source code for analysis, which seriously prevents those tools from being compatible to other ECAD tools and from becoming widely accepted tools in industry. Hence RAME's ability to accept HDL source code as FMECA input is an important advantage.

Second, the extent of automation implemented in the previously developed tools is generally very limited when they are compared with RAME. In particular, most of them are intended to escort a manual FMECA process by supplying online menu and input checking or to facilitate the post-analysis documentation process by providing automatic plotting/graphing capabilities. In contrast, RAME emphasizes the automation of the analysis aspect *per se* and ensures FMECA integrity by applying model- and knowledge-based techniques.

Finally, while none of the existing tools addresses the issue of automatic learning from previous projects, RAME's self-enriching information infrastructure continuously enhances the coverage and integrity of failure mode analysis and provides RAME with the most unique advantage.

References

[1] D. Palumbo, "Automating failure modes and effects analysis," in *The Proceedings of Annual Reliability and Maintainability Symposium*, (Anaheim, CA), pp. 304–309, 1994.

[2] C. J. Price, "Effortless incremental design FMEA," in *The Proceedings of Annual Reliability and Maintainability Symposium*, (Las Vegas, NV), pp. 43–47, 1996.

[3] V. Berman, "Standard Verilog-VHDL interoperability," in *Proceedings of the VHDL International Users Forum (VIUF)*, (Oakland, CA), pp. 142–149, May 1994.

[4] S. Chau, R. Hall, M. Traylor, and A. Whitfield, "Using COTS tools in the development of a model based avionics architecture tool," in *Proceedings of the International Council on Systems Engineering*, (Melbourne, Australia), July 2001.

[5] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, Jan. 2004.

[6] S. Chau *et al.*, "The implementation of a COTS based fault tolerant avionics bus architecture," in *Proceedings of IEEE Aerospace Conference*, vol. 7, (Big Sky, MT), pp. 297–305, Mar. 2000.