The Galileo Fault Tree Analysis Tool

Kevin J. Sullivan (Contact Author)	Joanne Bechta Dugan	David Coppit
University of Virginia	University of Virginia	University of Virginia
Dept. of Computer Science Thornton Hall, Charlottesville, VA 22903	Dept. of Electrical Engineering Thornton Hall, Charlottesville, VA 22903	Dept. of Computer Science Thornton Hall, Charlottesville, VA 22903
Phone: (804) 982-2206 Fax: (804) 982-2214	Phone: (804) 924-2078 Fax: (804) 924-8818	Phone: (804) 982-2291 Fax: (804) 982-2214
sullivan@cs.virginia.edu	jbd@virginia.edu	coppit@cs.virginia.edu

Abstract

We present Galileo, a dynamic fault tree modeling and analysis tool that combines the innovative DIFTree analysis methodology with a rich user interface built using package-oriented programming. DIFTree transparently integrates binary decision diagram and Markov methods under the common notation of dynamic fault trees, allowing the user to exploit the benefits of both techniques while avoiding the need to learn additional notations and methodologies. Package-oriented programming (POP) is a novel software architectural style, in which large-scale software packages are used as components, exploiting their rich functionality and familiarity to users. Galileo can be obtained for free under license for evaluation, and can be downloaded from the World-Wide Web.

Keywords: Galileo, dynamic fault tree, DIFtree, software tools, component-based, package-oriented

Approximate word count: 3000

This material has been cleared for publication by the University of Virginia.

The Galileo Fault Tree Analysis Tool

Kevin J. Sullivan and David Coppit
University of Virginia
Department of Computer Science
Thornton Hall, Charlotetsville, VA 22903
{sullivan|coppit}@cs.virginia.ed

Joanne Bechta Dugan
University of Virginia
Department of Electrical Engineering
Thornton Hall, Charlotetsville, VA 22903
jbd@Virginia.edu

Abstract

We present Galileo, a dynamic fault tree modeling and analysis tool that combines the innovative DIFTree analysis methodology with a rich user interface built using package-oriented programming. DIFTree transparently integrates binary decision diagram and Markov methods under the common notation of dynamic fault trees, allowing the user to exploit the benefits of both techniques while avoiding the need to learn additional notations and methodologies. Package-oriented programming (POP) is a novel software architectural style, in which large-scale software packages are used as components, exploiting their rich functionality and familiarity to users. Galileo can be obtained for free under license for evaluation, and can be downloaded from the World-Wide Web.

1. Introduction to Galileo

Software and electrical engineering researchers at the University of Virginia in are collaborating on the Galileo project, an experiment in the use of *package-oriented programming (POP)* to deliver innovative engineering analysis techniques supported by functionally rich, industrially viable software tools. *POP* is the use of off-the-shelf software packages as components [6][8][9]. It supports the *DIFTree* technique for efficient modular analysis of dynamic fault trees [1][2][3][4].

Galileo is an easy-to-use research prototype software tool that runs on personal computers machines running Microsoft's *Windows 95*, *98* or *NT* operating systems. Fault tree analysis is one of several important approaches to probabilistic risk assessment for engineered systems. We selected the name Galileo for our tool because Galileo was the first person to study the failure of engineered systems scientifically.

Galileo hosts *DIFtree* in a richly functional easy-to-use tool, the vast bulk of the user-level function of which is provided by a tightly integrated framework comprising Microsoft's *Word* and *Internet Explorer* programs and Visio Corporation's *Visio Technical* drawing program. The user interface is based on the familiar Microsoft Windows "MFC" user interface style. The tool enables engineers to edit and display fault trees in textual and graphical form through widely used, commercially-supported, volume-priced

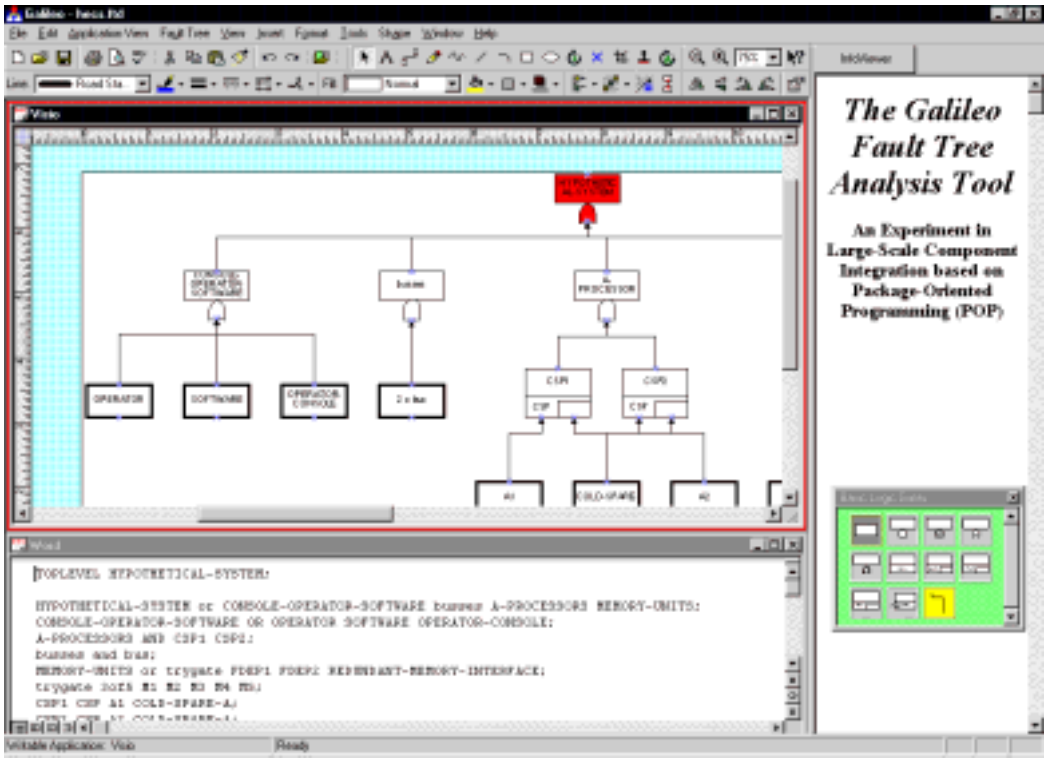

Figure 1. A screenshot of Galileo

components that are easy to integrate into real engineering practice. Our POP software development approach has allowed the construction of a tool that appears to be far richer, easier to use and more easily changed than would otherwise possible at a comparable level of development cost and code complexity. Some of the important features of Galileo are as follows:

- supports the *DIFTree* modular dynamic fault tree analysis approach
- allows the user to edit a fault tree in either a textual or graphical representation
- provides automatic rendering from the textual view to the graphical view, or vice-versa
- exploits the user interfaces of off-the-shelf packages, e.g., zoom, find-and-replace, print preview, etc.
- exploits the user's familiarity with common applications, significantly reducing training costs
- integrates the separate interfaces of the component applications into a single Galileo interface
- provides on-line documentation through an embedded internet browser and World-Wide Web pages

Figure 1 shows a screenshot of the current version of Galileo (2.1-Alpha). The upper left window shows the graphical view, hosted by Visio. The lower left window shows the same fault tree in textual format, as hosted by Word. The right side shows the on-line help, hosted by Internet Explorer. Because the graphical view provided by Visio is currently selected for viewing, Galileo presents to the user the

Visio menus and toolbars combined with the Galileo menus. When either of the other views is selected, the menus and toolbars are updated to reflect the interfaces of those components.

2. The *DIFTree* Approach to Modular Dynamic Fault Tree Analysis

Fault trees are mathematical models of how component-level failures in an engineered system combine to produce system level failures. Fault trees have a convenient graphical representations that makes it easy for reliability engineers to build, reason about, and validate such models [1]. Dynamic fault trees extend traditional (static) fault trees to enable the modeling of computer-intensive, fault-tolerant systems, in which important failure modes can depend on the ordering of component and subsystem failures, and can include cascading and common-cause failures (functional dependencies).

DIFtree (Dynamic Innovative Fault Tree) is a modeling methodology and analysis algorithm providing a unique approach for reliability analysis of complex computer-based systems. *DIFtre*e combines static and dynamic fault tree analysis techniques using a modular approach. It is unique in several aspects.

First, the decomposition of dynamic fault trees into modules uses an efficient algorithm to identify independent sub-trees. These sub-models can be solved separately using the most appropriate underlying methods, thereby allowing an exact solution to be obtained efficiently.

Second, the sub-models are classified as either static or dynamic, depending on the temporal relationships between the input events. Static gates express failure criteria in terms of combinations of events. Dynamic gates express the failure criteria in terms of combinations and order of occurrence for input events. A static sub-tree is one containing no dynamic gates. Other sub-trees are said to be dynamic. Static and dynamic sub-tree types are analyzed using different techniques. Static sub-trees are solved using combinatorial methods based on binary decision diagrams (BDDs). Dynamic sub-trees are solved with Markov methods. DIFtree combines the sub-tree results into an exact solution for the given tree.

Third, modeling imperfect fault coverage is critical to correct evaluation of computer-based systems, so we include this capability in both the static and dynamic solution techniques.

Fourth, the solution of static fault trees using methods based on BDDs, and including imperfect coverage in this solution, is unique to our approach. The BDD approach to static fault tree analysis is an innovation that has significant potential for the analysis of large models.

Fifth, the dynamic fault tree model allows the analysis of systems with complex redundancy management in a relatively simple manner. Several systems that defied analysis using standard techniques have been successfully analyzed using the dynamic fault tree model. Factors that often complicate analysis

include cold, warm and hot spares, spares which are shared among several different components, and functional dependencies. These are handled by the DIFtree dynamic gates.

3. Package-Oriented Programming

Package-Oriented Programming is a research project investigating the reuse and integration of very large-scale components. By large-scale we mean components whose implementations are on the order of a million or more lines of code. We are interested in large-scale reuse because without it, it appears that it will be impossible to build sophisticated engineering modeling tools quickly and inexpensively.

In the particular domain of tools, we see the following structure. Core analysis algorithms can often be implemented in thousands of lines of code. That is not the problem. The problem is building usability-engineered, interoperable, easy-to-learn, and well-documented tool "superstructures" that run on commodity Windows-based computers and that provide such features as rich graphical interfaces, cut-and-paste drawings, constraint-based layout, and the many other amenities typical in shrink-wrapped software.

3.1 Why Traditional Approaches to Component Reuse are Inadequate

Unfortunately, traditional approaches to software reuse based on object-oriented class libraries and design patterns, and on small-scale components such as typical “ActiveX controls,” does not attack the essence of the problem of developing such systems quickly and at low cost. Building million-line tools from small classes or components on the order of hundreds or thousands of lines of code each doesn't radically simplify the design problem. In particular, traditional approaches do not address the twin difficulties of designing sophisticated, usability-engineered interfaces, and of composing thousands of parts into a working system. The result of shortcomings in traditional software construction methods is that tools that support promising modeling techniques are not developed; are developed but are deficient in usability and interoperability properties; or are developed but at an extraordinary cost.

3.2 Package-Oriented Programming As an Attack on Essential Difficulties

Our work suggests that an approach based on the reuse and integration of just a few massive, commercially available, volume-priced components provides an effective attack on the tool superstructure and tool development problems. We call our approach Package-Oriented Programming (POP), and we hypothesize that it provides a workable basis for large-scale component reuse. In order to explore and evaluate the idea, we are developing Galileo. Galileo hosts the DIFtree codes in a usability superstructure based on the tight integration of multiple shrink-wrapped software packages. Galileo uses Microsoft Word 97 to present a textual view of a fault tree; Visio Technical 4.1 to 5.0 to present a graphical view; and (in ver-

sions prior to Galileo 2.1) Microsoft Access 97 to maintain a relational view. Galileo keeps multiple views consistent in a lazy, compiler-like manner that the user invokes by menu selection. The underlying analysis code is invoked in the same, easy way. The multiple application windows are composed into a single overall tool window using Microsoft's Active Document Architecture. This mechanism manages the integration of package and Galileo menu items, among other things. Galileo also uses Microsoft's Internet Explorer for on-line help and to support user interaction with each other and with the Galileo developers (e.g., for purposes of bug reporting, troubleshooting and for requesting enhancements).

Evidence to date suggests that Galileo is close to being an industrially viable tool. We performed an industrial case study on the use of Galileo at the Lockheed Martin Corporation. An engineer there concluded that Galileo has tremendous potential to aid reliability engineers in that corporation (as reported in an internal study of the tool). Galileo has been acquired by over 150 enterprises in 32 countries. Moreover, Galileo has achieved very tight integration of multiple large-scale components produced by different commercial vendors. Galileo is distinguished in several ways with respect to component integration.

Tight Integration of Multiple Packages: Galileo is based on the tight integration of multiple shrink-wrapped commercial-off-the-shelf (COTS) packages. It is not built on a single package; instead it integrates multiple packages as co-equal components. Brooks claims that the ability to do this could lead to "radical improvement" in software development productivity. His thoughts on the matter are echoed in the comments of Feigenbaum, the Chief Scientist of the United States Air Force:

> We are now living in a software-first world. I think the revolution will be in software building that is now done painstakingly in a craftlike way by the major companies producing packaged software. They create a suite–a cooperating set of applications–that takes the coordinated effort of a large team. What we need to do now in computer science and engineering is to invent a way for everyone to do this at his or her desktop; we need to enable people to "glue" packaged software together so that the packages work as an integrated system. This will be a very significant revolution. [5]

- **Package-Based Views of Engineering Models:** Galileo uses packages as user-friendly and richly functional views of an engineering model (i.e., of a fault tree). Microsoft Word presents the tree being edited in a textual form based on the *Galileo fault tree language.* Visio presents the tree as a structured technical drawing using Galileo shapes and behaviors programmed for Visio. A potentially significant advantage of this approach is that it reduces user learning costs by presenting sophisticated modeling and analysis codes through packages that users already understand how to use. Learning costs are a major disincentive for users to acquire new tools. The POP approach to using familiar, low-cost, sophisticated shrink-wrapped packages as major user interface components promises to help overcome this problem.

- **Compiler-Based Consistency Model:** Galileo keeps views consistent in a manner based on an analogy with a compiler. In a compiler-model, a program is first edited using one tool, an editor, and is then submitted to a compiler to be translated into executable code that can run on computer hardware. The source code form is convenient for people; and the object code, for the computer. In Galileo, you edit a fault tree in either textual or graphical form, and, when you are ready, you ask Galileo to "compile" it into the other form. Unlike a traditional compiler, though, Galileo translates in both directions. You can thus use a form most appropriate for your needs, e.g., text for editing and graphics for presenting to engineering management. This two-way translation scheme appears to be unique among fault tree tools. One benefit of the approach is that, we have found, different people like to edit fault trees in different forms—some graphical, some textual. Our decision to use a compiler-style approach rather than, say, eager incremental of views in the style of the Model-View-Controller (MVC), was driven by several issues related to the package technologies. First, the unstructured nature of the representations manipulated by the packages (e.g., text in Word) makes it hard to know when an incremental update of other views is needed. Second, the packages did not provide all of the event notifications that we would need to know when to perform updates. Third, behavioral integration of packages relies on an inefficient procedure invocation mechanism (OLE Automation), so the performance of an eager incremental update scheme would almost certainly be unacceptable.
- **Tight Integration of Component User Interfaces:** Galileo achieves tight integration of component packages within a single, overall Galileo tool window. We use Microsoft's *active document architecture* to achieve this integration. In particular, there is only one set of menus visible at a time. As you select one or another view, the application menus appropriate for that view are integrated with the Galileo menus. What you see at the top of the Galileo window is thus a set of menus which is the sum of the Galileo-specific menus and the menus for the package that you have selected.

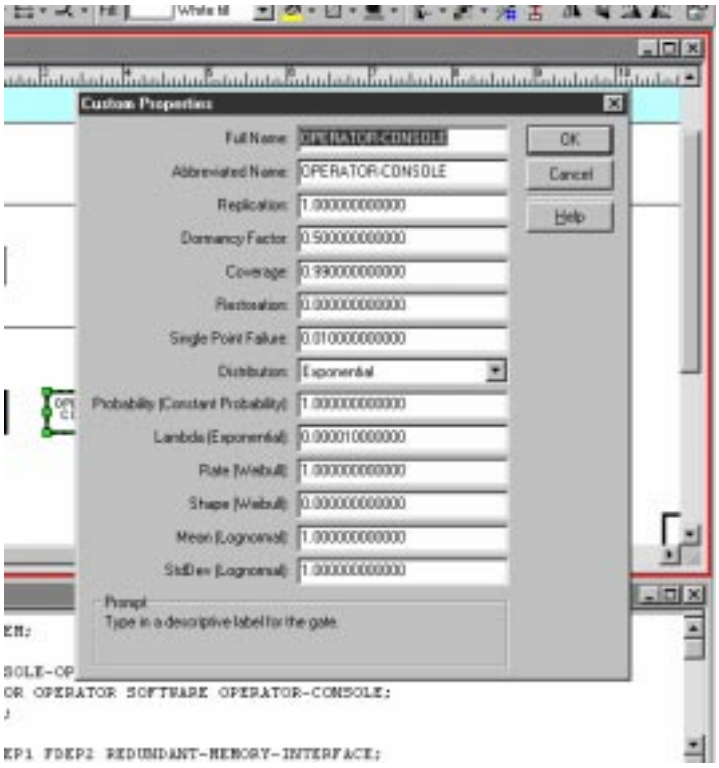

Figure 2. Editing custom properties in Visio

4. An example

Figure 1 shows a screenshot of Galileo and the "Hypothetical Example Computer System" (HECS) fault tree, described elsewhere [1]. Both the fault tree's graphical and textual representations are shown. Because Galileo uses off-the-shelf software as massive components, the user benefits from their rich functionality. In the figure, the graphical view is active, which makes Visio's toolbars and menus active. Through this interface the engineer can use Visio's support for aligning shapes, zooming the view, etc.

Basic event properties can be set from within the graphical view, as shown in Figure 2. When the user has finished editing the graphical view, the textual view can be automatically updated by Galileo, using the menu option as shown in Figure 3. At any time the user can invoke the analysis engine, DIFTree.

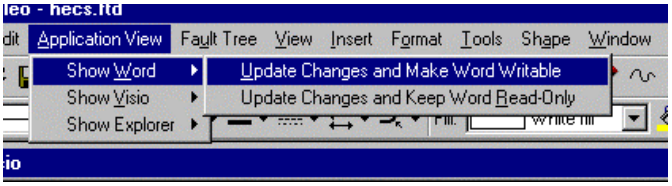

Figure 3. Rendering from Visio to Word

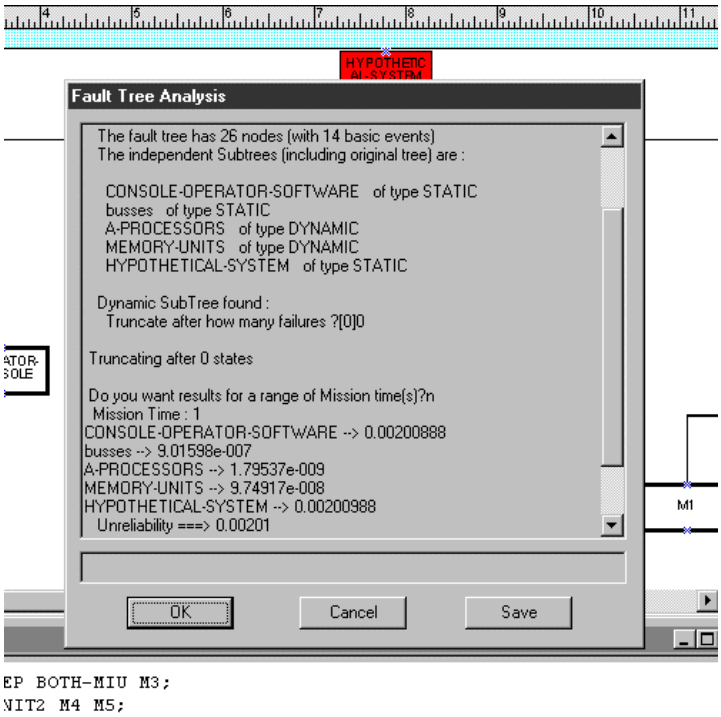

Figure 4. Analysis of the HECS fault tree

Figure 4 shows part of the output from DIFTree. 26 nodes were found, with 14 basic events. 5 sub-trees were found, two of which were dynamic. The solutions for each sub-tree are given, as well as the overall system unreliability of 0.00201.

Galileo also uses Microsoft's Internet explorer as an on-line help system, as shown in **Error! Reference source not found.**. In addition to providing a local copy of the user's manual and tutorial, the help system acts as a means of contacting the Galileo team and Galileo web page via the internet.

5. How to Obtain Galileo for Evaluation Purposes

Galileo is available on the World-Wide Web, free of charge, under license for evaluation purposes only (URL: http://www.cs.virginia.edu/~ftree/). Galileo 2.1 Alpha runs on Microsoft Windows 95, 98, and NT. It requires either or both of Microsoft Word (95 or 97) and Visio Corporation's Visio Technical (4.1 to 5.0). If *Internet Explorer* is available, Galileo will use it. If it is not available, it is not possible to browse the user documentation from within the tool. A 166 MHz "Pentium-class" computer is recommended, with at least 32 MB of main memory, and 50 MB of disk space. Installation involves the unzipping of the archive, and the execution of a standard setup program.

References

[1] Dugan, Venkataraman, and Gulati, "DIFtree: A software package for the analysis of dynamic fault tree models," *Proceedings of the 1997 Reliability and Maintainability Symposium*, January 1997.

[2] Mark A. Boyd, *Dynamic Fault tree models: Techniques for Analysis of Advanced Fault Tolerant Computer Systems*, Ph.D. thesis, Department of Computer Science, Duke University, 1990.

[3] Rohit Gulati and Joanne Bechta Dugan, "A modular approach for analyzing static and dynamic fault trees," in *Proceedings of the Reliability and Maintainability Symposium*, January 1997.

[4] Joanne Bechta Dugan, Salvatore J. Bavuso and Mark A. Boyd, "Dynamic fault tree models for fault tolerant computer systems," *IEEE Transactions on Reliability*, Volume 41, Number 3, pages 363-377, September 1992.

[5] E. Feigenbaum, Chief Scientist, U.S. Air Force, IEEE Computer, January 1998.

[6] Kevin J. Sullivan, Jake Cockrell, Shengtong Zhang, and David Coppit, "Package-oriented programming of engineering tools," In *Proceedings of the 19th International Conference on Software Engineering*, pages 616-617, Boston, Massachusetts, 17-23 May 1997, IEEE.

[7] Kevin J. Sullivan, "Galileo: An advanced fault tree analysis tool," URL:http://www.cs.virginia.edu/~ftree/index.html

[8] K.J. Sullivan and J.C. Knight, "Building Programs from Massive Components," in *Proceedings of the 21st Annual Software Engineering Workshop*, Greenbelt, MD, Dec. 4—5, 1996.

[9] K.J. Sullivan and J.C. Knight, "Experience Assessing an Architectural Approach to Large-Scale, Systematic Reuse," *Proceedings of the 18th International Conference on Software Engineering, Berlin, March 1996, pp. 220—229.*