Engineering of Computer Based Systems 98, Jerusalem, Israel, April 1998.

Integrated Analysis Environment for High Impact Systems

James Davis, Jason Scott, Janos Sztipanovits, Gabor Karsai
Measurement and Computing Systems Laboratory
Vanderbilt University
Nashville, TN 37235

Marcus Martinez
Sandia National Laboratories
Albuquerque, NM

Abstract

Modeling and analysis of high consequence, high assurance systems requires special modeling considerations. System safety and reliability information must be captured in the models. Previously, high consequence systems were modeled using separate, disjoint models for safety, reliability, and security. The MultiGraph Architecture facilitates the implementation of a model integrated system for modeling and analysis of high assurance systems. Model integrated computing allows an integrated modeling technique to be applied to high consequence systems. Among the tools used for analyzing safety and reliability are a behavioral simulator and an automatic fault tree generation and analysis tool. Symbolic model checking techniques are used to efficiently investigate the system models. A method for converting finite state machine models to ordered binary decision diagrams allows the application of symbolic model checking routines to the integrated system models. This integrated approach to modeling and analysis of high consequence systems ensures consistency between the models and the different analysis tools.

Introduction

Safe and reliable operation is a primary goal in the design of high consequence systems. This category of systems covers a wide range of systems, from nuclear weapons to chemical plants, where malfunctions may have catastrophic consequences. High consequence systems typically have the following characteristics:

- The systems are *complex and heterogeneous*. Safety and reliability are system-level properties.
- The systems are *reactive;* they are in continuous interaction with their environment.
- *Dynamics* is an essential characteristic of system behavior.

Safety analysis methods originated in the early ICBM-based weapon systems [1]. That was the first time when engineers have recognized that complex, system-level interactions, and cascading effects are the primary sources of safety hazards. The goal of safety analysis is to answer the following question*: Is there any normal or abnormal system input or fault conditions under which the system is able to produce behaviors that are considered to be safety hazards?* Safety analysis focuses on abnormal behaviors that must be avoided or prevented by safe design (inherently safe systems) or active safety control mechanisms. A frequently used technique in safety analysis is Failure Modes and Effects Analysis (FMEA). The FMEA analysis utilizes a fault propagation model with the objective of investigating component failures.

Reliability analysis is a different, but strongly related, aspect of system design. The primary question to be answered by reliability analysis is the following: *What is the probability of losing a required function as a result of component failures?* Reliability analysis typically uses physical component models characterized by failure rate statistics. The goal of the analysis is to calculate system-level failure rates for selected functionalities. One of the important methods in reliability analysis is based on fault tree models of

systems. Fault trees capture the relationship between a 'top event' (a critical functional failure in the system operation) and a combination of component faults.

Fault diagnosability analysis is a third strongly related element in the design of high impact systems. The question to be answered is the following: *Does the system's design include an adequate number of sensors built-in-tests to guarantee the required level of fault detectability, distinguishability and predictability?* Diagnosability analysis is based on causal network or fault propagation models that are augmented with observation models.

Our previous experience in the development and testing of real-time diagnostic and monitoring systems for aerospace [2][3], electric utility [4] and chemical process applications [5], and our current work in the safety, reliability and fault analysis of nuclear weapon system surety components have shown that the selection of suitable modeling paradigms plays a critical role in obtaining practical, usable answers during analysis. Traditionally, safety, reliability and fault analysis tools use different modeling approaches and different analysis methods. Since the questions they want to answer refer to the same underlying system, the models cannot be, and are not, independent. The consistency among the models and consequently the analysis results is not guaranteed due to the lack of any formal description of relationships among the different models. Naturally, this lack of consistency makes the objective evaluation of design tradeoffs impossible.

This paper discusses a new approach for integrated safety, reliability and diagnosability analysis using Model-Integrated Computing principles and tools. The essence of the approach is to perform system modeling using a domain-specific, multiple-aspect modeling environment which allows the integrated, consistent representation of system models. This integrated model is translated into the input languages of analysis tools, thereby maintaining the consistency among the tool-specific models. An important element of the method used is the introduction of relational models in system representation and the application of Ordered Binary Decision Diagrams (OBDD) [6] for representing, transforming and analyzing the models. The result of using OBDD representations and manipulations is a significant improvement in scalability and an opportunity for using discretized models for very large-scale problem domains.

Background

Model-Integrated Computing

In *Model-integrated computing*, integrated, multiple-view, domain-specific models capture information relevant to the system under design. Models explicitly represent the designer's understanding of an entire system, including the information-processing architecture, physical architecture, and operating environment. Integrated modeling explicitly represents dependencies and constraints among various modeling views.

The Multigraph Architecture (MGA) is an infrastructure for model-integrated computing and is described in detail elsewhere. [7] A typical model-integrated tool configuration is shown in Figure 1.

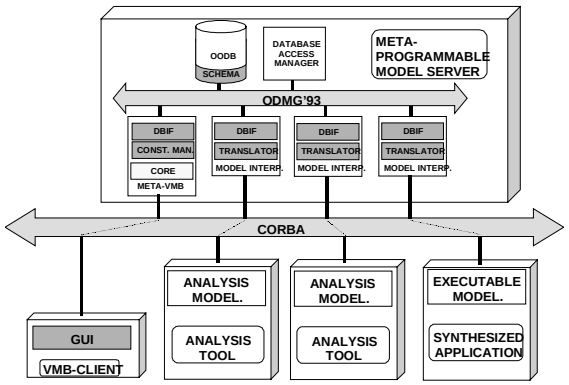

Figure 1: Model-Integrated Tool Environment in the MGA framework

The integrated environment includes Modeling Tools, Integrated Model Database, Analysis Tools, and Application Synthesis Tools. The Analysis Tools work with tool-specific analysis models; the applications are specified in terms of executable models. The modeling paradigm of the analysis tools and the executable models are domain independent. In a given domain, the relevant information about the design artifact is captured by a multiple-view, domain specific modeling paradigm. In MGA, the modeling paradigm is described by a meta-language. The meta-language representation of the modeling paradigm is used to generate components of a Metaprogrammable Model Server, and a Metaprogrammable Visual Model Builder (META-VMB). Key components of the model server are the "Model Interpreters". The role of the Model Interpreters is to translate the domain specific model into the analysis models for the tools and the executable models of applications to be synthesized.

This architecture allows that the analysis and synthesis tools to share design information that is common without requiring that the tools use the same modeling paradigm.

An integrated tool environment is built in the following steps using the MGA infrastructure:

1. Systems and domain experts conduct domain analysis and specify an integrated modeling paradigm, which can capture key aspects of the system. The modeling paradigm is comprised of the concepts, relationships, model composition principles and constraints which are specific to the domain.
2. Using the formal representation of modeling paradigms, systems and domain experts specify and create a domain specific model building, model analysis, and software/system synthesis (MIPS) environment. The environment includes reusable domain specific components, general building blocks, domain specific model analysis tools, and software synthesis tools. Completion of this step is supported by MGA meta-tools.
3. Domain and application engineers build integrated, multiple view models of systems to be designed and implemented. The multiple view models typically include requirement and design models, are based on formally specified semantics, and support performance, safety and reliability analysis processes.
4. Domain and application engineers analyze the models according to the nature and needs of the domain. The analysis is typically supported by generic tools (e.g. performance analysis, simulation, reliability analysis, safety analysis etc.). The domain specific models are translated into the input languages or input data structures of the connected analysis tools. The model translation is completed by MGA model interpreters that are part of the modeling and model analysis environments.
5. If necessary, the validated models are used for the automatic synthesis of software applications.

Ordered Binary Decision Diagrams (OBDD)

Safety, reliability, and fault analysis tasks use discrete models and operations over finite domains. The most general difficulty in all of the analysis techniques is the size of the state space in large-scale systems. Combinatorial explosion is the result of the exponential increase in the number of discrete elements (states, hypotheses, etc.) during operations, which sooner or later makes access to the individual elements unfeasible. By introducing a binary encoding for the elements, the individual elements, sets of elements, and relations among them can be expressed as Boolean functions. For example, the 2^{100} states of a finite state automaton can be encoded with binary variables *{s(1),…,s(100)}* forming a binary state vector ***s***. The Boolean functions

$f_1[s(1),\ldots,s(100)]=s(1)' \wedge s(23) \wedge s(99)$ and

$f_2[s(1),\ldots,s(100)]=s(1) \wedge s(22) \wedge s(89)$

represent two subsets, S1 and S2, of the 2^{100} states including 2^{97} elements each. The set S3=S1∪S2 can be derived symbolically as the disjunction of the two Boolean functions:

$f_3[s(1),\ldots,s(100)]= f_1[s(1),\ldots,s(100)] \vee$

$f_2[s(1),\ldots,s(100)]= s(1)' \wedge s(23) \wedge s(99) \vee s(1) \wedge s(22) \wedge s(89)$

without the need to enumerate and compare the individual elements - which would be a formidable task otherwise. In general, using Boolean function representations, we can express operations and algorithms in diagnosis and safety analysis in symbolic form, by means of symbolic Boolean function manipulations.

OBDD-s provide a symbolic representation for Boolean functions in the form of directed acyclic graphs [6]. They are a restricted, canonical form version of Binary Decision Diagrams (BDD) [8]. Bryant [9] described a set of algorithms that implement operations on Boolean functions as graph algorithms on OBDDs. Taking advantage of the efficient symbolic manipulations, researchers have solved a wide range of problems in hardware verification, testing, real-time systems, and mathematical logic using OBDDs that would have been otherwise impossible due to combinatorial explosion. Symbolic model checking is extensively used in hardware design (see, e.g., [10]), and has shown to be efficient in state space sizes 10^{120} and beyond.

Selection of Domain-Specific Modeling Paradigm

Safety, reliability and diagnosability analysis algorithms work with a "model" (a suitable representation) of the system. The level of detail in the models is determined by the required depth of the analyses.

1. *Models for Safety and Diagnosability Analysis.* Safety and diagnosability analysis requires the development of models that represent the relationship between failure modes (or fault events) of physical components and discrepancies (or discrepancy events) in the high-level behavior of the system. Taking into consideration of the

characteristics of the high impact system category (complexity, dynamic behavior), we selected the following model organization:

- The structure of the system is captured in two independent hierarchies, forming the *Behavioral Model* and the *Physical Model*.
- The Behavioral Model represents the system behavior in the discrete state space in terms of hierarchical, parallel state machines. The Behavioral Model includes both functional and fault behaviors by representing functional and fault states, and transitions among these states triggered by input, local, and fault events. We have selected the StateChart notation [11] for behavior modeling because the StateChart models are expressive, scalable, and support incremental modeling.
- The Behavioral Models are augmented with *Observation Models.* The observation models define the set of observable (instrumented) events that can be used to perform system diagnosis.
- The Physical Model captures the component hierarchy of the system. The physical components are modeled as component assemblies and sub-assemblies. Each physical component has a fault model view. The fault model view lists the physically possible, and functionally meaningful fault modes of the components.
- The interdependencies between the Behavioral Model and Physical Model are represented in the form of references between these models. The *implemented-by* references show the relationship between system states and physical components. Their role is to specify the set of components that have impact on the system behavior in each state. The *caused-by* references link fault events in the state transitions of the Behavioral Models to fault modes of components. It is expected that all of the fault modes of the components which are referenced by the *implemented-by* references of a state are accounted for in the outgoing transients of the state machine.

Explicit representation of the interdependencies between behavioral models and physical models is a critical element of the integrated modeling paradigm. It guides the model builder to understand their relationship, and enforces the systematic analysis of the effects of the fault modes of components.

2. *Models for Reliability Analysis.* The analysis environment includes a reliability analysis tool, WinR [12], which requires modeling the systems with a fault tree. The fault tree represents the relationship between a top event and fault modes in the form of an AND-OR tree. Using the fault tree, and the failure rate information of the components, the tool calculates the expected rate for the occurrence of the selected critical system state.

The models for reliability analysis have strong overlap with the models for safety and fault analysis. The most important relationships are the following:

- The top event in reliability analysis corresponds to a transition into a selected critical system state, which is modeled as part of the behavioral models.
- The fault events correspond to fault modes of components that are part of the physical models.
- The fault tree can be derived from the set of all possible state trajectories that lead to the selected critical system state. These trajectories are fully defined by the behavioral models.
- The failure rate of the components used in reliability analysis correspond to the expected rate of the component fault modes introduced in the physical models.
- The conclusion is that the behavioral and physical models contain all the information required for reliability analysis except failure rate data for the component fault modes. Therefore, by extending the component fault models with probabilistic information, the modeling paradigm will allow safety, diagnosability and reliability analysis from the same model set. It is important to note that the relationship between the fault tree models required by the reliability analysis tool and the behavioral models is quite complicated. It is practically impossible to expect that independently built behavioral models and fault tree models will be consistent with each other even in trivially small size systems.

Formal Model for Integrated Analysis

The role of the formal model is to give a domain independent, mathematical specification for the models. We will focus on the Behavioral Model, since it plays a central role in the analysis methods. The Physical Model is primarily descriptive, and used in information management. The selected domain-specific form of the Behavioral Model is the StateChart

notation. While StateCharts are convenient for building large-scale, parallel state machine specifications, the analysis algorithms require a formal mathematical model, which captures the precise semantics of the hierarchical, parallel state machines. We use Discrete Event System (DES) models for this purpose.

The DES model of a dynamic system (system with memory) is shown on the left side of Figure 2.

The DES model is the $(X,F_Y,F_S,S,\Gamma,\mathbf{f},s_0,Y,\mathbf{g})$ finite state automata where:

X is the input event set,

F_S, F_Y are the sets of component faults and output faults, both considered to be input events,

S is the state set,

$\Gamma(s)$ is a set of feasible or enabled events, defined for all $s \in S$ with $\Gamma(s) \in X$,

$\mathbf{f}$ is a state transition function, $\mathbf{f}: X \times F_S \times S \rightarrow S'$, defined only for $x \in \Gamma(s)$ when the state is s,

s_0 is the initial state,

Y is the output set, and

$\mathbf{g}$ is an output function, $\mathbf{g}: X \times F_Y \times S \rightarrow Y$, defined only for $x \in \Gamma(s)$ when the state is s.

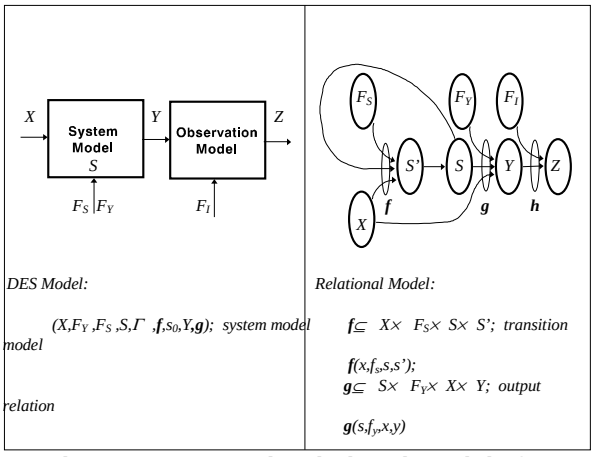

Figure 2: DES and Relational Models for Dynamic Systems

Since the models need to support diagnosability analysis, the model is divided into a *System model* and an *Observation model*. In this approach, the component faults are considered as additional inputs to the system. It is also possible to model the abnormal (out of range) inputs as elements of the X input set, creating a 'normal' and 'faulty' partition in X. In order to model partial observations of the state trajectory independently from the outputs of the dynamic system, we use the $\mathbf{h}: Y \times F_I \rightarrow Z$ observation model describing the mapping between the Y outputs, F_I instrumentation faults, and the Z observations. The DES formalism also allows the representation of non-deterministic state machines, which is an important requirement for modeling large-scale systems.

The right side of Figure 2 shows the equivalent *Relational Model* of a dynamic system. In the relational model, the $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{h}$ mappings are considered to be the $\mathbf{f} \subseteq X \times F_S \times S \times S'$, $\mathbf{g} \subseteq X \times F_Y \times S \times Y$ and $\mathbf{h} \subseteq Y \times F_I \times Z$ relations. The significance of the relational representation is that the models can be re-written as Boolean functions by introducing some binary encoding for the sets $X \times F_S \times S \times S'$, $X \times F_Y \times S \times Y$ and $Y \times F_I \times Z$. The Boolean functions $\mathbf{f}(x,f_S,s,s')$, $\mathbf{g}(x,f_Y,s,y)$ *and* $\mathbf{h}(y,f_I,z)$ evaluate to *true* for those elements of the sets $X \times F_S \times S \times S'$, $X \times F_Y \times S \times Y$ and $Y \times F_I \times Z$ (encoded by the Boolean vectors (x,f_S,s,s'), (x,f_Y,s,y) and (y,f_I,z)), which are related by the $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{h}$ relations. The Boolean representation of the DES model can be directly translated into an OBDD form, allowing the symbolic evaluation of the analysis algorithms.

Although it is not the purpose of this discussion, it is worthwhile to note that DES (or relational) models preserve composability and can be constructed in a modular fashion using either component oriented modeling approach [13] or process-oriented modeling approach [14].

Integrated Safety, Reliability and Diagnosability Analysis using OBDD-s

The primary difficulty in safety, diagnosability, and reliability analysis is combinatorial explosion. For example, the generation of a fault tree from the behavioral model requires the exhaustive enumeration of all possible state trajectories that may lead from an initial state (or a set of possible initial states) to a critical state under all fault conditions. By representing the Behavioral Model symbolically as an OBDD, the required calculations can be completed symbolically without explicitly enumerating the exponential number of alternatives.

The application of OBDD-s for the analysis requires the following steps:

1. Mapping the Behavioral Models into OBDD-s: This step is completed automatically. In accordance to the general framework of the Multigraph Architecture (MGA) (Figure 1), the StateChart models in the Model Database are traversed by a Model Interpreter, which selects a binary encoding for the sates and incrementally builds up the OBDD representation for the relational model of the corresponding DES.
2. Safety analysis: The safety analysis tool receives the OBDD representation of the Behavioral Model and performs forward reachability analysis on the state

machine. Given a set of initial states S_0, reachability analysis calculates the set of reachable states $S^*(S_0)$ under all possible combination of $x \in X$ input event, $f_S \in F_S$ and $f_I \in F_I$ fault events. The goal of the safety analysis is to show that selected critical events are not elements of the reachability set. The reachability set is calculated symbolically, therefore the analysis is feasible even for very large state spaces.

3. Diagnosability analysis: The diagnosability analysis tool receives the OBDD representation of the Behavioral Model and Observation Model and performs *fault detectability* and *fault distinguishability* analysis on the state machine. An $f_S \in F_S$ or $f_I \in F_I$ fault event is detectable if the reachable set of observations $Z^*(f, S_0)$ under all possible combination of $x \in X$ input events is nonempty. A system and its observation model provide *single-fault distinguishability*, if all possible observations are unique to the single faults. These definitions can be extended to stronger detectability and distinguishability conditions, which take into consideration fault masking, instrument faults, and multiple faults. In all cases, the analysis algorithms are based on forward and backward reachability analyses that are executed symbolically.
4. Reliability analysis: As it was mentioned above, the reliability analysis tool, WinR, expects a fault tree that represents all possible combinations of fault events leading to a selected top event. Unfortunately, simple backward reachability analysis does not provide the logic function of fault events that must expressed in a fault tree. Instead, the analysis algorithm generates all of the state trajectories leading to the top event using backward propagation, and simultaneously builds up the logic relationship between the fault events and the top event.

Modeling and Analysis Tool Architecture

The Model-Integrated Safety, Reliability and Diagnosability Analysis tool architecture is an instance of the generic architecture of Model-Integrated Computing Environments discussed before. Components of the tool architecture are shown in Figure 3. The domain specific models are built by the Metaprogrammable Visual Model Builder, and are stored in the Model Server. The constraints defined in the meta-language representation of the modeling paradigm. The capture constraints are enforced by the Visual Model Builder and prevent the user to create invalid models.

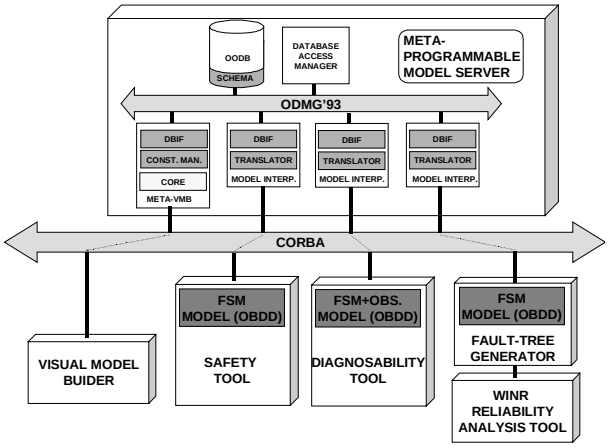

Figure 3: Analysis tool configuration

There is a separate model interpreter for each tool. The model interpreters traverse the domain specific models and collect/translate the information into the required input data structures of the tools. This solution enables the reuse of the tools even if the domain specific modeling paradigm is changing.

The WinR reliability analysis tool is an 'external' component in the tool architecture. WinR is interfaced to the integrated modeling environment through the Fault Tree Generator module. The output of the Fault Tree Generator is the fault tree file required by the WinR. It is important to note that the WinR tool has a separate model building interface, therefore the tool can be used independently from the integrated environment. The advantage of using WinR in the configuration above is that the overlapping modeling views are kept consistent by the integrated modeling environment.

Example

The techniques described above will be demonstrated in a simple system (Figure 4) including a tank, a pump, and a valve. The pump is used to fill the tank, while the valve is used to empty the tank. Only the actual system behavior is modeled; it is assumed a controller exists for controlling the operation of the valve and the pump. The example assumes the pump and the valve have the same flow rates and that the tank is instrumented sufficiently to determine the level of material in the tank. The controller ensures the tank is never full or empty.

The finite state machine model for the system is shown in part A of Figure 4. Each piece of hardware has its behavior modeled as a separate state machine. When the system is analyzed, these state machines are analyzed as if they are parallel components of the same FSM. The pump is modeled as being in one of four states: PON, POFF, PstuckOn, and PstuckOff. These states correspond to the pump being on, off, stuck on (the pump cannot be turned off), and stuck off (the pump cannot be turned on). Similarly, the valve is modeled as being in the states Opened, Closed, StuckOpen, and StuckClosed. The tank is modeled as being in either nominal operating ranges (TOK) or in failure ranges (TError). Nominal system events are: ON, OFF, CLOSE, and OPEN. Failure events are FailOn, FailOff, FailClose, and FailOpen which occur when the pump fails to turn on, pump fails to turn off, the valve fails to close or the valve fails to open.

Tank failure ranges correspond to the tank becoming full or empty. Only when the pump is stuck on and the valve is stuck closed can the tank become full. Otherwise, the controller will either open the valve or turn off the pump to keep the tank in nominal operating ranges. Likewise, only if the pump is stuck off and the valve is stuck open can the tank completely empty its contents.

The event tree shown in Figure 4B is generated from the FSM models of Figure 4A. This tree shows all possible trajectories through the state space that could lead to the TError state from the default system state of {POff, VClosed, TOK}. The nodes in this tree correspond to Boolean AND (oval nodes), Boolean OR (diamond nodes), and the Boolean encodings for the system events (rectangular nodes). This tree represents the Boolean expression relating events to the selected failure state. Symbolic techniques are used to derive this expression automatically from the OBDD representation of the system models. This tree structure is not normally realized, due to its size and the fact that all information contained in the tree is also contained in the OBDD representing the trajectories. Realizing the tree requires enumerating the trajectories represented by the corresponding OBDD.

Figure 4C shows the simplified fault tree. This tree only contains Boolean AND, Boolean OR, and the Boolean encodings for the failure events. This tree is generated automatically through simplification of the event tree. Again, an OBDD is used to capture this failure expression. The fault tree is translated in a format analyzable by WinR.

For our example, the number of failure trajectories is quite large compared to the size of the system's Behavioral models. However, the Behavioral models actually represent a state space of 32 discrete states if parallelism is not used in modeling. Even for this limited example, the number of failure trajectories would be difficult to discover manually. When automatically simplified, the fault tree becomes manageable. By inspection, one can determine the only possible event sequences which can lead to the failure state are PStuckOn and StuckClosed or PStuckOff and StuckOpen. This is the expression captured by the fault tree.

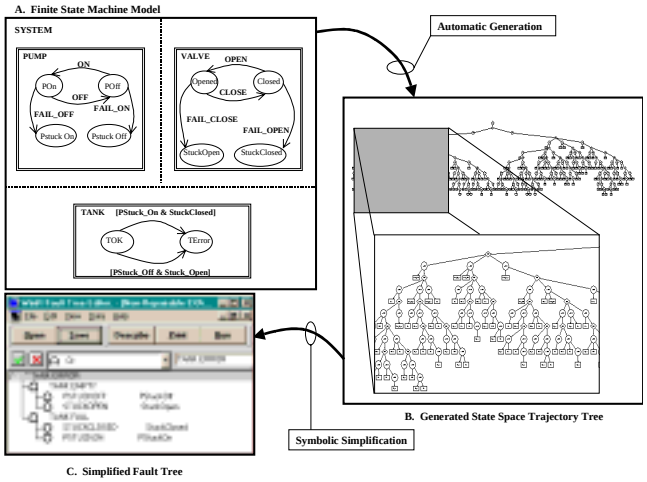

Figure 4: Example System Model and Analysis

Summary

Safety, diagnosability, and reliability analysis is a difficult problem due to two primary reasons. First, the models to be used in these analyses are not independent form each other, therefore guaranteeing the consistency of the analysis results is a major concern. Second, the generally used discrete, finite state modeling techniques require analysis methods that are plagued by combinatorial explosion of the state and event sets derived during the analysis. The proposed model-integrated modeling and analysis environment and the proposed analysis methods address both problems. The introduction of an integrated modeling paradigm allows the construction of models that are domain specific, and consistent for each analysis task. The problem of combinatorial explosion is mitigated by the use of relational models and OBDD representations. Although symbolic manipulations offer tremendous advantages in the analysis of large-scale systems, scalability remains an important issue in analyzing these systems. The size of OBDD data structures is sensitive to the ordering of the Boolean variables, which indicates the need for the development of good heuristics while mapping the models into OBDD representations. Our experience

with the analysis of a variety of systems has shown the feasibility of the approach.

Acknowledgements

Support for this project comes from Sandia National Laboratories and DARPA. Sandia has supplied all of the example systems used for tool evaluation. DARPA's support has been through the Evolutionary Design of Complex Systems (EDCS) project.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract De-AC04-94AL000.

References

[1] Leveson, N. : Safeware: System Safety and Computers, Addison Wesley, New York, 1995.

[2] Misra, A., Sztipanovits, J., Underbrink, A., Carnes, R., Purves, B., "Diagnosability of Dynamical Systems," *Proc. of the Third International Workshop on Principles of Diagnosis*, pp. 239-244, Rosario, WA 1992.

[3] Misra, A., Sztipanovits, J., Carnes, R., "Robust Diagnostic System: Structural Redundancy Approach," *Proc. Of the SPIE International Symposium on Knowledge-Based Artificiaal Intelligent Systems in Aerospace Systems*, Orlando Florida, April 5-6, 1994.

[4] Padalkar, S., Karsai, G., Biegl, C., Sztipanovits, J., Okuda, K., Miyasaka, N., "Real-Time Fault Diagnostics," *IEEE Expert*, pp. 75-85, June, 1991.

[5] Karsai, G., Padalkar, S., Franke, H., Sztipanovits, J., "A Practical Method for Creating Plant Diagnostic Applications," *Integrated Computer-Aided Engineering* to be published in 1996.

[6] Bryant, R. E., "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, C-35(8), 1986.

[7] Sztipanovits, et al., "MULTIGRAPH: An Architecture for Model-Integrated Computing" *Proc. of the IEEE ICECCS'95* Ft. Lauderdale, Florida, Nov. 6-10. 1995.

[8] Lee, C. Y., "Representation of Switching Circuits by Binary Decision Programs," *Bell System technical Journal* pp. 985-999, 1959.

[9] Bryant, R. E., "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams", *Technical Report CMU-CS-92-160*, School of Computer Science, Carnegie Mellon University, June 1992.

[10] Burch, J. R., Clarke, E.M., Long, D. E., "Symbolic Model Checking for Sequential Circuit Verification," Technical Report, CMU-CS-93-211, Carnegie Mellon University, July 15, 1993.

[11] Harel, D., "StateCharts: A Visual Formalism For Complex Systems", *Science of Computer Programming 8*, pp. 231-278, 1987.

[12] Sandia National Laboratories, *WinR Reliability Analysis Software*, Systems Reliability Departement.

[13] Sampath, M. et al., "Failure Diagnosis Using Discrete-Event Models," *IEEE Transactions on Control Systems Technology*. Vol. 4, No. 2, pp. 105-124, March, 1996.

[14] Sztipanovits, J., Carnes, R., Misra, A., "Finite-State Temporal Automata Modeling for Fault Diagnosis," *Proc. Of the 9th AIAA Conference on Computing in Aerospace*, San Diego, CA, October, 1993.