SANDIA REPORT
SAND2008-4746
Unlimited Release
Printed May, 2008

Development and Application of the Dynamic System Doctor to Nuclear Reactor Probabilistic Risk Assessments

David M. Kunsman, Sean Dunagan, Tunc Aldemir, Richard Denning, Aram Hakobyan, Kyle Metzroth, Umit Catalyurek and Benjamin Rutt

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SANDIA REPORT
SAND 2008-4746
Unlimited Release
Printed May, 2008

Development and Application of the Dynamic System Doctor to Nuclear Reactor Probabilistic Risk Assessments

Principal Investigators:

David M. Kunsman, Space Systems Engineering Department
Sandia National Laboratories
and
Sean Dunagan, Performance Assessment and Decision Analysis Department
Sandia National Laboratories

Tunc Aldemir, Richard Denning, Aram Hakobyan, and Kyle Metzroth
Nuclear Engineering Program, Ohio State University

Umit Catalyurek and Benjamin Rutt
Dept. of Biomedical Informatics, Ohio State University

Abstract

This LDRD project has produced a tool that makes probabilistic risk assessments (PRAs) of nuclear reactors—analyses which are very resource intensive—more efficient. PRAs of nuclear reactors are being increasingly relied on by the United States Nuclear Regulatory Commission (U.S.N.R.C.) for licensing decisions for current and advanced reactors. Yet, PRAs are produced much as they were 20 years ago. The work here applied a modern systems analysis technique to the accident progression analysis portion of the PRA; the technique was a system-independent multi-task computer driver routine.

Initially, the objective of the work was to fuse the accident progression event tree (APET) portion of a PRA to the dynamic system doctor (DSD) created by Ohio State University. Instead, during the initial efforts, it was found that the DSD could be linked directly to a detailed accident progression phenomenological simulation code—the type on which APET construction and analysis relies, albeit indirectly—and thereby directly *create and analyze the* APET. The expanded DSD computational architecture and infrastructure that was created during this effort is called ADAPT (Analysis of Dynamic Accident Progression Trees). ADAPT is a system software infrastructure that supports execution and analysis of multiple dynamic event-tree simulations on distributed environments. A simulator abstraction layer was developed, and a generic driver was implemented for executing simulators on a distributed environment.

As a demonstration of the use of the methodological tool, ADAPT was applied to quantify the likelihood of competing accident progression pathways occurring for a particular accident scenario in a particular reactor type using MELCOR, an integrated severe accident analysis code developed at Sandia. (ADAPT was intentionally created with flexibility, however, and is not limited to interacting with only one code. With minor coding changes to input files, ADAPT can be linked to other such codes.) The results of this demonstration indicate that the approach can significantly reduce the resources required for Level 2 PRAs. From the phenomenological viewpoint, ADAPT can also treat the associated epistemic and aleatory uncertainties.

This methodology can also be used for analyses of other complex systems. Any complex system can be analyzed using ADAPT if the workings of that system can be displayed as an event tree, there is a computer code that simulates how those events could progress, and that simulator code has switches to turn on and off system events, phenomena, etc.

Using and applying ADAPT to particular problems is not human independent. While the human resources for the creation and analysis of the accident progression are significantly decreased, knowledgeable analysts are still necessary for a given project to apply ADAPT successfully.

This research and development effort has met its original goals and then exceeded them.

Acknowledgements

The authors wish to thank Randy Cole, Michael Young, and Randy Gauntt for freely sharing their MELCOR expertise. In addition, the authors thank Mark Allen and Marianne Walck for their managerial oversight. Most of all, the authors thank Dana Powers for his original inspiration that all of the authors should talk because pieces of a problem solution were residing in separate places and urgently needed to be brought together. The whole became much larger than the sum of its parts.

(All of the people cited work at Sandia National Laboratories.)

TABLE OF CONTENTS

Figures

Tables

Summary

This LDRD project has produced a tool that can make probabilistic risk assessments (PRAs) of nuclear reactors—analyses which are very resource intensive—more efficient. PRAs of nuclear reactors are being increasingly relied on by the United States Nuclear Regulatory Commission (U.S.N.R.C.) in making licensing decisions for current and advanced reactors. Yet, PRAs are produced much as they were 20 years ago. They require significant resources to create and analyze. This work applied a modern systems analysis technique to the accident progression analysis portion of the PRA; the technique was a system-independent multi-task computer driver routine.

Initially, the objective of the work was to fuse the APET portion of a PRA to the dynamic system doctor (DSD) created by Ohio State University. Instead, during the initial efforts, it was found that the DSD could be linked directly to a detailed accident progression phenomenological simulation code—the type on which APET construction and analysis relies, albeit indirectly—and thereby directly *create* the APET. The expanded DSD computational architecture and infrastructure that was created during this effort is called ADAPT (Analysis of Dynamic Accident Progression Trees). ADAPT is a system software infrastructure that supports execution and analysis of multiple dynamic event-tree simulations on distributed environments. A simulator abstraction layer was developed, and a generic driver was implemented for executing simulators on a distributed environment.

As demonstrations of the use of the methodological tool in the probabilistic modeling of severe accident phenomena in Level 2 PRA, ADAPT was applied to quantify the likelihood of creep rupture of pressurizer surge line, hot leg, and SG tubes in a PWR with a large dry containment using MELCOR, an integrated severe accident analysis code developed at Sandia. (ADAPT was intentionally created with flexibility, however, and is not limited to interacting with only one code. With minor coding changes to input files, ADAPT can be linked to other such codes.) A station blackout initiating event with a failure of the AFWS was considered as in this test case.

The results of this demonstration indicate that the developed approach can significantly reduce the manual and computational effort in Level 2 PRA analysis. By implementing the model mechanistically, it also eliminates the potential of introducing errors while making changes in the input decks manually for running new accident scenarios. From the phenomenological viewpoint, it can also treat the epistemic and aleatory uncertainties associated with complex physical phenomena taking place during severe accident progression. Several different parallel processing configurations were investigated. It was found that more computational stations did not necessarily result in shorter analysis time. This was because some stations could be idle while waiting for a previous calculation to finish.

The ADAPT methodology can also be used for analyses of other complex systems. In PRAs, it could be applied to the Level 1 analysis, during which the frequency of

challenges to the core integrity are examined. Any complex system can be analyzed using ADAPT if the workings of that system can be displayed as an event tree, there is a computer code that simulates how those events could progress, and that simulator code has switches to turn on and off system events, phenomena, etc.

There is interest in ADAPT nationally and internationally. Future development work could include explicitly using another plant simulator, improving the metadata management system, making the creation of branching rules more user friendly, further optimizing the scheduling techniques developed, and developing a complier that will take high-level branching rules and generate application specific edit-rules.

Using and applying ADAPT to particular problems is not human independent. While the human resources for the creation and analysis of the accident progression are significantly decreased, knowledgeable analysts are still necessary for a given project to apply ADAPT successfully. It can be made more user friendly than it already is, but it will never be "user independent."

This research and development effort has met its original goals and then exceeded them.

1. Introduction

1.1 Purpose

This Laboratory Directed Research and Development (LDRD) project developed a methodological tool to make the creation and use of probabilistic risk assessments (PRAs) more efficient. Specifically PRAs for nuclear reactors were investigated, but the work has broader application than that. Hence, while this report shall almost totally concentrate on PRAs for nuclear reactors, the potential for broader application is also noted. This work was a joint effort of Sandia National Laboratories and Ohio State University.

PRAs of nuclear reactors are the most comprehensive tools in quantifying reactor safety, but they are notoriously resource intensive. Yet, PRAs are produced much as they were 20 years ago. The idea of this work idea applied one modern systems analysis technique to a specific part of the PRA—the accident progression event tree (APET). The Dynamic System Doctor (DSD), developed at Ohio State University, is system-independent, interactive software for model-based state/parameter estimation in dynamic systems. The DSD was initially linked to an APET so that it could directly and semi-automatically help construct additional event tree logic. The success in doing so led us to an additional step, the bypassing of the APET "seed" to begin with and linking an enhanced DSD directly to a detailed severe accident systems analysis computer code, MELCOR, that simulates potential nuclear reactor accidents, and thereby creating the APET. The enhanced DSD has been named "Analysis of Dynamic Accident Progression Trees "(ADAPT). (It must be emphasized here, as it is in the report as well, that ADAPT is independent of the severe accident simulation code chosen except for a small portion of computer code that performed the communication between them.) Before discussing the work, however, background material on PRAs and the DSD will be presented so that the reader will then be able to better understand why what was done was done and the significance of the accomplishment.

1.2 Background

PRAs are the method of choice for assessing and quantifying the risks of low probability, high consequence accidents, such as those related to nuclear reactors. There are three levels of probabilistic risk assessment (PRA) performed for nuclear power plants:

- Level 1 PRA quantifies the frequency of core damage.
- Level 2 PRA examines the mode and timing of containment failure and the release of radioactivity material to the environment.

• Level 3 PRA quantifies the risk of off-site adverse health effects.

Event trees are used in both the analysis of core damage frequency and in the analysis of containment failure modes [1]. The accident progression event trees (APETs) used in Level 2 PRA identify, sequentially order, and probabilistically quantify the important events in the progression of a severe accident. The development of an APET consists of

- identifying potentially important parameters to the accident progression and associated containment building structural response,
- determining possible values of each parameter (including dependencies on outcomes of previous parameters in the event tree),
- ordering the events chronologically, and,
- quantify the frequency and consequences of the ordered scenarios.

The quantification of an APET is primarily based on sensitivity studies performed with accident simulation computer codes that are validated against experimental data. An APET is conceptually similar to the system event trees in Level 1 PRAs. While the quantification of the branch probabilities in Level 1 PRAs relies on fault tree analysis, however, a number of calculations are performed with the accident simulation code prior to quantification of the APET which include a range of code parameter variations that provide insights to the analyst on the impact of uncertainties on the probability of alternative branches on the tree. (See References 2 and 3 for examples of detailed PRAs for nuclear reactors using APETs.)

For each general type of postulated accident, the APET analysis considers the important characteristics of the core damage process, the challenges to the barriers and structures designed to mitigate an accident, and the response of those barriers and structures to those challenges. APETs are used to identify, to order sequentially, and to quantify probabilistically the important events associated with the progression of a severe accident. The development of an APET consists of identifying potentially important issues, determining possible values of each parameter (including dependencies on previous parameters in the event tree), ordering the events chronologically, and defining the information needed to determine each parameter. In addition, an APET is static, no dynamic, so that parameters that could change during the accident need to be re-queried in the APET. Trying to account for the timings of events is made more complex because, a priori, the analyst does not know whether Event A precedes or follows Event B but must determine the ordering of events based on sensitivity calculations, or sometimes simply by making assumptions. Often because of uncertainties in accident progression, it is possible that Event A might precede Event B under some circumstances and follow Event B under other circumstances.

Describing the possible and credible accident pathways leads to the construction of a complex event tree, potentially involving hundreds of event questions with the potential of several branches at each question. An APET generates hundreds of millions of

different possible accident progression pathways that must each be analyzed in some sense before an estimate of risk can be made. Many hours and computing resources are needed to produce results for even a single pathway. The increase of computational power over the last two decades has made the calculational part of the effort more manageable, but the burden on the analyst has not been similarly assuaged. The construction and analysis of such a tree is still resource intensive as the pathway direction at each branch can potentially depend on all the results of all the preceding branch answers. That is, where the pathway goes from point A can depend on all the specifics of how the accident progressed to point A. Keeping the logic straight for such dependencies in the APET construction and analysis of results is fraught with error potential. An additional error potential is introduced by manually changing a portion of the input data for each new scenario in order to simulate different accident scenarios. These types of errors are very hard to identify in the post-analysis of enormous output database, thus making the overall time of the analysis even longer

In addition, as mentioned above, the current PRA methodology is the static and can only account for the time element in the accident progression through sequencing and re-querying of events. A review of the literature, however, indicates that the exact timing of failure events and exact magnitude of system variables at the time of a failure event can be critical in determining the risk associated with system operation. In fact, the standard approach to event trees also requires the analyst to establish a specific order of events, when in fact variability of accident conditions and uncertainty in the ability to model severe accidents could change the order of events.

The timing and magnitude of events can be understood via tests or simulations. The former is expensive for even examining one possible progression pathway, prohibitively so to examine several, let alone many. Integrated reactor systems computer codes have been written to simulate the progression of possible accidents. In the past, the APET analyst would incorporate the results of some such computational analyses (as well as any test data available) in the construction of the tree. But, the actual scenarios computed likely do not perfectly align with pathway logic under consideration in the APET construction so estimates of progression timing and events were necessary. This added unnecessary uncertainty in the results.

Therefore, automating the APET construction and subsequent analysis, incorporating timing explicitly in that construction and analysis, and linking the APET directly to an integrated accident analysis computer code are highly desirable objectives. These are the objectives of the work this project set out to achieve. This project achieved them.

1.3 Approach

The starting point for this work was the recognition that the Dynamic System Doctor (DSD) computer code developed at Ohio State University could be used to address the objectives of the effort. The DSD is a system-independent multi-task driver (MTD) for model-based state/parameter estimation in dynamic systems. It can provide input to the

real-time analysis for evolving systems conditions and has been successfully applied to real-time xenon estimation and stability analysis for nuclear reactors and fault detection in automobile engines. The DSD has also been linked with a neural net approach.

The DSD is a system independent, state/parameter estimation software [4] based on the cell-to-cell mapping technique (CCMT). The CCMT models the system evolution in terms of probability of transitions in time between sets of user defined parameter/state variable magnitude intervals (cells) within a user specified time interval (e.g. data sampling interval). The most important feature of DSD is that it is both an interval and a point estimator. Subsequently, it yields the lower and upper bounds on the estimated values of state variables/parameters, as well as their expected values. Knowledge of such bounds is particularly important in the determination of the operational safety margins. More importantly, the consequence probability functions will automatically incorporate the aleatory uncertainties[1] associated with the estimation process, which is a feature desirable by NRC but not achievable by other techniques.

The DSD also yields the probability of finding the system in a given cell in the state/parameter space that provides a probabilistic measure for model-based diagnosis to rank the likelihood of faults in view of modeling uncertainties and/or signal noise. Such a ranking is useful for risk-informed regulation and risk monitoring of nuclear power plants. Another important feature of this methodology is that its discrete-time nature is directly compatible with a look-up table implementation, which is very convenient for the use of data that may be available from tests or actual incidents; this is commonly the data used during APETs.

The research and development described in this report initially sought to address one part of the APET with the DSD: finding what the conditions are in the reactor coolant system during all of the potential severe accidents. Specifically, we shall examine the conditions in the steam generators of a pressurized water reactor (PWR). These conditions can substantially vary depending on the nature of the accident, and what happens to the tubes of the generators can alter the nature and subsequent progression of the accident itself; it is exactly this rigorous complexity that makes using the DSD during APET creation such a natural choice. As presented above, the four stages of development of an APET are identifying potentially important issues, determining possible values of each parameter (including the dependencies on previous parameters in the event tree), ordering the events chronologically, and determining the frequency of the resultant pathways, and the DSD could help in developing each of these for the creation of an APET .[2] Furthermore, the DSD could be used in quantifying the APET.

---

[1] That is, those uncertainties that are stochastic as opposed to epistemic uncertainties, those uncertainties that are due to lack of knowledge.

[2] Since the DSD/APET combination is a real-time model, it was recognized at the beginning of the project that it might be able to relay accident progression information to reactor operators quickly enough to give them the capability to mitigate severe accidents if the model combination was installed at a plant can running at the time the reactor systems were challenged by an initiating event.

The initial confidence that this effort could be brought to fruition came from the success the DSD had with other related technical challenges, such as estimating the amount of xenon in a reactor or analyzing the stability of a BWR [5]; the DSD has also been linked successfully to a simplified pressurizer model for APET generation [6]. Furthermore, the DSD already had multi-threading capability [7], and the multi-tasking extension of DSD was in progress when this work was begun, and off-line versions of the relevant modules had been successfully implemented in a distributed computing environment [8].

The methodology development could not be done simply on a theoretical basis. An application was necessary to develop and test the methods as the work progressed. A pressurized water reactor (PWR) with a large dry containment was used as a reference system, with station blackout as the initiating event compounded by the failure of the AFWS. Additional possible system events included a stuck open safety relief valve on the secondary side of the plant, a stuck open pressure operated relief valve on the primary side of the plant, and loss of reactor coolant through developing leaks in the seals of the reactor coolant pumps. In various pathways, the integrity of the reactor coolant system could be further challenged by induced leaks in the steam generator tubes, pressurizer surge line, or reactor coolant system hot leg piping.

In succeeding sections of this report, the methodology will be discussed, the developed code system architecture and infrastructure will be described, results of a demonstration of applying the developed code suite will be presented, and conclusions will be drawn.

2. Methodology

This section of the report presents the methodology implemented in the work. First, a brief history of dynamic event trees is presented. Then, the computer code suite developed in this effort is discussed, although the detailed architecture and infrastructure of it is described in Section 3. Finally, the severe accident phenomenological code linked to in the work is briefly described.

2.1 Dynamic Event Tree (DET)

There are different interpretations to the word "dynamic" when used along with PRA. One use of the term dynamic PRA or "living PRA" is to describe periodic updates of the PRA to reflect any changes in the plant configuration [9]. Another use is when the PRA model is updated to account for equipment aging [10]. The third use is to describe an approach that includes explicit modeling of deterministic dynamic processes that take place during plant system evolution along with stochastic modeling [11, 12, 13, 14, 15, 16, and 17]. In this third use, plant parameters are represented as time-dependent variables in event tree-construction with branching times often determined from the severe accident systems analysis code being used to examine the plant. It is this last definition of dynamic PRA that is used within the context of this effort reported here.

In dynamic PRA analysis, event tree scenarios are run simultaneously starting from a single initiating event. The branchings occur at user specified times and/or when an action is required by the system and/or the operator, thus creating a sequence of events based on the time of their occurrence. For example, every time a system parameter exceeds a threshold/setpoint, branching takes place based on the possible outcomes of the system/component response. These outcomes then decide how the dynamic system variables will evolve in time for each branch. Since two different outcomes at a branching may lead to completely different paths for system evolution, the next branching for these paths may occur not only in different times, but also based on different branching criteria. The main advantage of DET methodology over the conventional event tree method is that it simulates probabilistic system evolution more closely.

Software development for DET generation began in mid 1980's. A variety of tools and techniques have been proposed. The research work has modeled the response of both the plant systems and plant operators to an initiating event that propagates into an accident. Several institutions have been involved in developing DET generation methodologies both in the United States [11, 12] and Europe [13, 14, 15, 16, and 17].

In the mid 1980's, researchers at the Joint European Center at Ispra, Italy, developed a methodology for dynamic reliability analysis called Dynamic Logical Analytical

Methodology (DYLAM) [13, 14, and 15]. The basic idea of the DYLAM methodology is to provide a tool for coupling the probabilistic and physical behavior of a system for more detailed reliability analysis. All the knowledge about the physical system under study is contained in the system simulator. The active components of the system are allowed to have different states such as nominal, failed on, failed off and stuck. Once the simulator is linked to the DYLAM code, DYLAM drives the simulation by assigning initial states to each branch and triggering stochastic transitions in the component states, taking into account the time history of the logical states of components if necessary (e.g. for operator modeling). For each path (or branch), the (possibly time-dependent) probability of the system achieving that branch is evaluated from the user-provided branching probabilities. The probability of occurrence of a given consequence (or Top Event) is the sum of the probabilities of all the branches leading to that Top Event [15]. Each system component/operator is characterized by discrete states with different options to model transitions between these states, such as stochastic transitions with constant probabilities, functionally dependent transitions, stochastic and functionally dependent transitions, conditional probabilities, and stochastic transitions with variable transition rates. The time points at which the transitions (either on demand or stochastic) take place correspond to the branching points. The DYLAM approach has been used to perform dynamic reliability analysis not only in nuclear, but also in chemical, aeronautical, and other industries.

In 1992, Acosta and Siu [11] proposed a variant of DYLAM for Level-1 PRA[3] called DETAM (Dynamic Event Tree Analysis Method), to analyze the risk associated with nuclear power plant accident sequences. DETAM provided a framework for treating stochastic variations in operating crew states, as well as in hardware states. The plant process variables used to determine the likelihood of stochastic branchings were calculated from a system simulator. The branchings were allowed to occur at user-specified fixed points in time. In case of hardware-related branchings, the system unavailabilities were modeled as demand failure frequencies. In the cases of diagnosis state and planning state transitions, mainly expert judgment was used to assign probabilities/frequencies.

In 1993, Hsueh and Mosleh [12] developed the Accident Dynamic Simulation Methodology (ADS). It was an integrated dynamic simulation approach for Level-1 PRA developed for large scale dynamic accident sequence analysis. The modeling strategy of ADS was based on breaking down the accident analysis model into different parts according to the nature of the processes involved, simplifying each part while retaining its essential features, and developing integration rules for full scale application. Whenever a hardware system state transition point or an operator interaction point is reached, the accident scheduler chooses one path to follow. After the simulation process reaches an end point, the scheduler directs the simulation back to the previous branch point, reinitializes every simulation module back to this time point, and follows the other branch point path.

[3] As mentioned in the introduction, Level-1 PRA only analyzes a reactor to determine the frequency of core damage and does not analyze the subsequent progression of the accident to determine what consequences, if any, that damage might cause.

Another tool for DET generation developed in 1999 is DENDROS (Dynamic Event Network Distributed Risk Oriented Scheduler) [18]. The DENDROS was developed mainly to model response of safety features to a transient for Level-1 PRA and is a discrete event processor, managing messages coming from different calculation modules including the physical system simulator and decision processes. It is designed for a distributed computing environment using a network of processors exchanging information through an independent channel. During a simulation, the scheduler makes a decision about the need to create new processes if a setpoint is crossed (branching point), to change the already running processes to stand-by state for later reuse, or even to force some non-active ones to terminate based on the end conditions, such as probability falling below a user-specified cutoff value. The DENDROS was linked to the pressurized water reactor simulator TRETA (Transient Response and Test Analyzer)].

In 2002, researchers from GRS[4], Germany developed a DET method combined with Monte Carlo simulation called MCDET (Monte Carlo Dynamic Event Tree) [17]. The MCDET considers all combinations of two characteristics of a transition: "when" and "where to". Discrete and random "when" and/or "where to' are taken into account by DET analysis, while continuous and random ones were handled by Monte Carlo simulation. The MCDET was implemented as a stochastic module that could be operated in tandem with any deterministic dynamics code. For each element of Monte Carlo sample, MCDET generates a discrete DET using the system code and computes the time histories of all system variables along each path together with the path probability. The mean conditional probability distribution (conditional on the initiating event and the values of randomly sampled aleatory uncertainties) over all trees in the sample is the final result. To keep the computational effort practicable, a probabilistic "cutoff" criterion was introduced that would allow to terminate any branches with a probability below that cutoff value. For practical application, the MCDET was linked with severe accident analysis code MELCOR [19][5]. The focus was on the modeling of the response of the safety features of the plant and the reaction of the operating crew during severe accident progression.

[4] Gesellschaft fur Anlagenund Reacktorsicherheit.

[5] The MELCOR code will be discussed in more detail in a later section as it is the integrated accident progression phenomenology code that ADAPT was linked to in this work, although ADAPT could be linked to other such codes. It is not hard-wired to MELCOR.

2.2 Introduction to ADAPT

Originally, this project was conceived as creating a computer architecture that would link the DSD to an APET so that the DSD could directly and semi-automatically help an analyst construct additional tree logic. The authors quickly realized, however, that it was just as straight-forward to create an overall architecture that would have the DSD approach drive a phenomenological computer program as to which sets of input to run abd when to stop and make adjustments to the input parameters (to simulate changing plant events such as operator actions) and then restart so as to create an APET through the linked analyses. This work also created the computational infrastructure to support this automated process. These computational innovations are discussed in greater detail in the next section. Here, however, is an overview. The mechanized procedure that has been developed for the generation of APETs which can substantially reduce the manual and computation effort, reduce , the likelihood of input errors, develop the order of events dynamically, and treat accident phenomenology consistently is called ADAPT (Analysis of Dynamic Accident Progression Trees). ADAPT is based on the concept of dynamic event trees (DETs) which use explicit modeling of deterministic dynamic processes that take place during plant system evolution along with stochastic modeling [11, 12, 13, 20, 21]. In PRA using DETs, all scenarios starting from the initiating event are considered simultaneously.

The branchings occur at user specified times and/or when an action is required by the system and/or the operator. For example, every time a system parameter exceeds a threshold or setpoint (the thresholds and setpoints are specified by the analyst as input), branching takes place based on the possible outcomes of the system/component response. These outcomes then decide how the dynamic system variables will evolve in time for each branch. Since two different outcomes at a branching may lead to completely different paths for system evolution, the next branching for these paths may occur not only in different times, but also based on different branching criteria.

Like all the other DET generation techniques presented in the DET overview above, the philosophy of the ADAPT approach is to let a system code (simulator) determine the pathway of the scenario within a probabilistic context. When conditions are achieved that would lead to alternative accident pathways, a driver generates new scenario threads (branches) for parallel processing. The branch probabilities are tracked through the tree using Boolean algebra. To avoid numerical catastrophe due to enormous number of branch executions, it is necessary to terminate branches based on user defined truncation rules, such as truncating an execution when a branch probability falls below a given limit or when the user specified simulation time is exceeded.

Regarding its contribution to the state-of-the art, ADAPT combines the active component modeling approach and parallel processing capability of DENDROS [18 with passive component handling capability of MCDET [17]. It differs from MCDET, however, in the way uncertainties are handled. As indicated in Section 2.1, MCDET first divides the set of stochastic variables (which it regards as aleatory uncertainties) into two subsets of

discrete (V_d) and continuous (V_s) variables. Then it selects an element $v_s \in Vs$ using Monte Carlo sampling from V_s and runs the simulator with v_s for all elements of V_d (considered as paths of an event tree). ADAPT also regards the variables associated with the stochasticity in the active (e.g. valves, pumps) and passive (e.g. pipes, steam generator tubes, containment) component behavior and other severe accident phenomena (e.g. hydrogen combustion) as aleatory uncertainties. Uncertainties associated with simulator inputs (e.g. heat transfer coefficients, friction coefficients, nodalization) are regarded as epistemic. For active components, the ADAPT approach is similar to that used by DENDROS in that the timing of the branch initiation is determined by the simulator based on the computed magnitude of the process variables (e.g. pressure, temperature, level) and the control laws, as well as possible failure modes of the component. For example, the time at which a demand will be placed on a safety relief valve to open and close will be determined by the simulator based upon the computed pressure and valve setpoint. The valve may open and close in response to the setpoint pressure but may also fail to close on demand. At this point in time, ADAPT generates a branching point with two (or more) possible scenarios to be followed by the simulator. In the case of passive component behavior and other stochastic phenomena, ADAPT uses an approach similar to Latin Hypercube Sampling from the cumulative distribution function (CDF) of the dynamic variables relevant to the components and phenomena under consideration.

(The ADAPT approach to the stochastic modeling of passive components and severe accident phenomena allows reusable scenario information so that if the CDFs used to initiate the branches are changed, the simulations do not have to be repeated.)

ADAPT will be described in much more detail in Section 3.

2.3 MELCOR

As mentioned above, the integrated accident progression phenomenology computer code chosen to be used for this project was MELCOR [19], although there is nothing in the developed ADAPT that intrinsically is dependent on using MELCOR and only MELCOR. The authors strove to make ADAPT as flexible and phenomenological code-independent as possible. That is, as will be discussed below, the architecture is independent of the phenomenological code, but some small pieces of the infrastructure need to be written anew for each code which ADAPT is to drive.

MELCOR is a fully integrated, relatively fast-running code used to simulate the progression of accidents in light water reactor nuclear power plants. A wide range of accident phenomena can be modeled with MELCOR including thermal-hydraulic response of the reactor coolant system, reactor cavity, containment and confinement buildings; core heat-up, degradation, and relocation; ex-vessel debris behavior; core-concrete attack; hydrogen production, transport, and combustion; fission product release and transport; impact of engineered safety features on thermal-hydraulic and radionuclide behavior. MELCOR has been validated against experimental and plant data [22, 23]. It

uses the "control volume" approach to describe the plant systems. No specific nodalization of a system is forced on the user, which allows a choice of the degree of detail appropriate to the task at hand. Reactor-specific geometry is imposed only in modeling of the reactor core.

A MELCOR calculation is executed in two parts. First, an executable called MELGEN is used to specify, process, and check input data, as well as to generate the initial restart information, written to a restart file. Then, the second executable called MELCOR uses that restart file and specific MELCOR input data (general information including the problem duration, time steps, edit information, etc. written to a separate file called MELCOR Input File) to advance the problem through time.

MELCOR consists of a number of modules called packages. The packages that are of particular interest from the viewpoint of this research work include the Control Functions (CF) package, Flow Paths (FP) package, Burn (BYR) package, and Executive (EXEC) package. The CF package is used by the user to define functions of variables in the MELCOR database. The values of these functions are made available to other packages in MELCOR. ADAPT utilizes the CF package to implement the branching rules for simulations. For example, pressures in appropriate control volumes may be used to control the opening of a valve or initiate the failure of containment, the temperature in a volume may define the enthalpy associated with a mass source/sink, or the particle loading on a filter may modify the flow resistance in the corresponding flow path. The user can also simulate the complicated control logic, involving the values of a number of variables in the system. The FP package, together with Control Volume Hydrodynamics (CVH) package, is used to model thermal-hydraulic behavior of liquid water, water vapor, and gases in MELCOR. The main application of the FP package is to connect the control volumes from the CVH package. The BUR package allows the user to model gas combustion in control volumes. The EXEC package is used to control the overall execution of MELGEN and MELCOR calculations. It coordinates different processing tasks for other MELCOR packages, including file handling, input and output processing, modification of sensitivity coefficients, selection of system time-step, time advancement, and calculation termination.

3. ADAPT Architecture and Infrastructure

This section presents the computational architecture and infrastructure of ADAPT which allows flexibility to link ADAPT with different system simulation codes, parallel processing of the scenarios under consideration, on-line scenario management (initiation as well as termination) and user friendly graphic capabilities. First presented is a system overview. Then, the distributed execution support leveraged by ADAPT will be discussed. Finally, the implantation of a prototype which will implement the system described will be presented.

3.1. ADAPT Overview

A schematic overview of the ADAPT architecture is shown in Figure 1. The ADAPT system is composed of a *Server,* a set of compute nodes that will be used to follow transient in each branch via a *Plant Simulator* (*SIM*), a *Distributed Database Management System* that will enable the access to data generated by the *Plant Simulator*, and a set of *Client* programs and tools that will allow end-user to interact with the ADAPT system.

Following an initiating event, *Client* starts a new experiment by submitting the request to the *Server*. The request includes a reference to the *Plant Simulator*, and necessary input files that contain initial parameters. This request is recorded to the *Metadata Database* in the *Server,* and the *Scheduler* initiates an execution using the ADAPT's simulator-agnostic *Driver* on an available compute node. Upon termination of the *Plant Simulator,* the *Driver* parses its output to determine the cause of termination. If a setpoint crossing occurred on a branching condition, the *Driver* submits one ore more branch execution requests to the *Server*. It is the *Server*'s responsibility to compute the branch probability and check with the *Probability Module* to decide if the branch should be executed or not. The *PRA Database* (DB) contains data to quantify the likelihood of branches generated upon crossing setpoints or following operator intervention. The database can consist of the minimum cut sets for the Top Events relevant to the branch in the form of binary decision diagrams for fast pre-processing [9] or simply contain probabilities based on operational failure data. The branching probabilities (possibly obtained through preprocessing) are passed on to the *Probability Module*. If branching is initiated, the Scheduler then spawns a process to follow the branch. The *Scheduler* can spawn as many processes as needed to follow the subsequent branches. The resulting tree structure, branch probabilities, and some basic statistics are all recorded in Metadata Database. The actual simulation results are left intact in the compute nodes they have been executed. Access to those files is provided by *Distributed Database Management Systems* by leveraging the STORM middleware [10, 11].

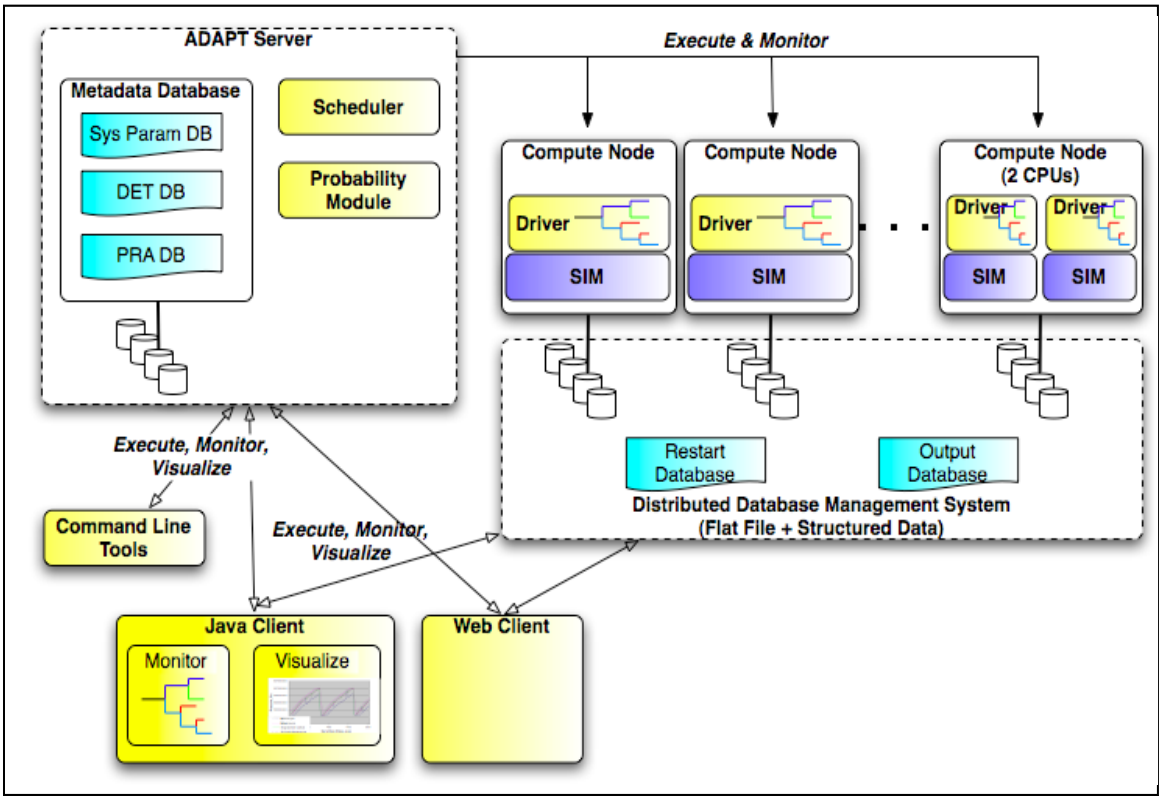


Figure 1. ADAPT System Architecture.

The interface to the *Plant Simulator* is abstracted to allow use of different plant simulators with possibly different computational models. A *Plant Simulator* needs to interface with the runtime system in two places: 1) during execution for task branching and migration, and, 2) before and after execution, to load and store its state and results. A plant simulator-agnostic driver has been developed that communicates with the distributed database system to retrieve and store the necessary input and output files needed by the plant simulator. In other words, the driver stages the necessary input files prior to execution of the plant simulator, and after completion of the execution it ``stores'' the output files generated by the plant simulator on the distributed database system. Thus, the plant simulator can be run without any modifications.

In summary, the significant features of the ADAPT system are:

1. The ADAPT system is designed for a distributed computing environment; the scheduler can track multiple concurrent branches simultaneously.

2. The scheduler is modularized so that the branching strategy can be modified (e.g. biasing towards the worse event).

3. Independent database systems store data from the simulation tasks and the DET structure so that the event tree can be constructed and analyzed later.

4. ADAPT is provided with a user-friendly client which can easily sort through and display the results of an experiment, precluding the need for the user to manually inspect individual simulator runs.

The ADAPT system has been designed targeting Enterprise-Grid environments. An Enterprise-Grid is a small Grid environment that is composed of a heterogeneous collection of computer and network resources within a single administrative domain and/or institution. Specifics and challenges of the runtime system will be discussed in the next sections.

3.2. System Software Infrastructure

In this section, the requirements and challenges of a system software framework that will support dynamic reliability and risk assessment techniques are further discussed. The focus is on three major components: distributed execution support, distributed database support, and the scheduling component that orchestrates the distributed execution.

3.2.1 Distributed Execution Support

In ADAPT distributed execution support is needed in order to run *Plant Simulators* on a set of heterogeneous compute and network resources. The framework should have an open architecture that will allow easy replacement of the components and the algorithms used in those components. The runtime environment should support execution of stand-alone or parallel plant simulators, staging of the necessary input and output files for the execution, and a mechanism to let the Plant Simulator communicate with the Scheduler to instantiate new branches in the DET by running new simulations.

For branching and task migration, as a first step application-level migration techniques are relied upon. That is, application specific control mechanisms and checkpoint code are used. For example, MELCOR [24] allows users to define their own *Control Functions* and those functions are provided in the simulation input files. With the use of MELCOR's control function syntax and language it is possible to monitor and modify simulation variables, create checkpoints, or even stop the execution of the application. If an application does not support user defined control functions, it is possible to insert those monitoring functions and the control logic to the application either at compile time or at runtime [25] with minimal intrusion. Since the focus of the framework is neither to deliver a new computation steering tool nor a new checkpointing system but provide an efficient distributed execution for restartable applications we plan to leverage the existing work on computation steering [26] and checkpointing [27, 28] when needed.

3.2.2 Distributed Database Support

ADAPT necessitates mainly two types of distributed database support. The first one is access to well-structured data including the DET structure itself and metadata about the simulations. The second one is access to the input and output files of the simulations. Although it is possible to define strongly-typed structures for the input and output files of the simulations, those data will be accessed by the simulations in application specific ways. In other words, unless the applications are modified directly, they will use their own access mechanism to retrieve and store those files. Below, the requirements of these two types of database support are examined in more detail.

For handling simulation metadata, various options exist. One possibility is to design a relational schema and implement it via existing relational databases. Such an approach may prove too restrictive, however, and would necessitate the development of application specific user interfaces to access and process the data. Consider an example case with multiple plant simulators. The metadata required for each plant simulator does not need to be exactly the same, and most likely it will not be the same. Although one might attempt to find a common schema that will cover all the existing simulators' metadata, this approach still has the problem of extensibility. An alternative would be to use XML schemas to describe metadata schemas and a generic framework such as Mobius [29, 30] that will allow the analyst to design and deploy schemas for the existing plant simulators as well as give him flexibility to extend those schemas for new simulators.

Direct and efficient access to the data stored in application specific format is the second type of database support needed. ADAPT uses a simple execution model, in order to avoid modification of simulators, that necessitates staging the data files in and out. To make this possible, a distributed database system is needed that is capable of retrieving and storing user-defined format data in an efficient manner.

Other than staging input files, another major responsibility of the distributed database system is efficiently processing of analysis queries. Even some of simple queries, such as plotting a system variable over time (e.g. pressure, temperature etc.) for a complete scenario might require accessing multiple files stored on multiple nodes; since a complete scenario could be composed of multiple branches executed on different nodes. Another type of the query may involve comparison of two or more scenarios' data.

There are multiple use cases for such a query. To start with a very basic case, a comparison of the results of two or more scenarios may be wanted, presented either visually or mathematically, in order to a gain better understanding of the dynamics of the plant. The same motivation inspires a second use case scenario, where the analyst may want to group/cluster multiple scenarios that are ``close'' to each other. Another use case involves dynamic execution. If it were possible to identify a scenario that had already been executed in either another study or even in the same study but happen to occur after

a different event sequence, we could eliminate the execution of redundant copies of it. One can extend this idea by searching not only exactly identical scenarios but ``very similar'' ones. Combining this with the risk factors might allow us to prune event trees at a much faster rate.

3.2.3 Scheduling

Many forms of the scheduling problems have been well studied over the last couple of decades, such as, independent task scheduling [31], DAG scheduling [32] scheduling of multiple parallel jobs on space-shared systems [33], and, recently, batch-shared I/O scheduling [34, 35]. Some of the recent Grid scheduling work [36, 37] addresses independent task execution on the Grid environment, and the others focus on workflow scheduling [38, 39, 40]. Kondo et al. [41] proposed resource prioritization heuristics for scheduling short-lived applications onto enterprise desktop grids. Raadt et al. [42] presents a framework for scheduling divisible loads [43], which has been implemented as an extension to A Parameter Sweep Tool [36, 44].

Dynamic Event Tree generation poses a new scheduling problem which is called here Dynamic Tree Scheduling. It has some unique properties that make existing scheduling techniques not directly applicable. First of all, the workload is dynamically generated while executing the portions of the workload. Standard DAG scheduling techniques and workflow scheduling techniques necessitates that the task graph and workflow--which is usually represented with a DAG too--is given as an input, and they compute mapping of the tasks to compute nodes (with possible duplication in workflow scheduling) that will minimize a cost function, e.g. execution time. The end of the DET generation is the creation of a tree, which is a special instance of DAG. Even if the exact shape of the tree and the execution time of the tasks in that tree could be guessed, that knowledge could not be used to find an optimum scheduling, because, to the best of the writers' knowledge, there is no optimum tree scheduling algorithm for non-uniform vertex weights (task execution times) and edge weights (communication costs).

Another approach for scheduling could be looking at a snapshot of this problem, and model the scheduling problem as an online independent job scheduling problem, such that at completion of each task, zero or more tasks are submitted to the system. The non-deterministic nature of the branching, however, makes it impossible to predict the execution time of every single branch accurately, if it is possible at all. Hence, neither the standard scheduling techniques, such as MinMin, MaxMin [31], nor their enhanced versions that would take the I/O into account [36, 44, 45, 46] nor more advanced hypergraph partitioning-based scheduling approaches [34, 35, 47] can be directly applied.

In the framework, a pluggable scheduling interface has been designed and three basic scheduling techniques have been implemented: 1) random scheduling, 2) first-come first-served scheduling, and 3) greedy staging minimization. As their name implies, when a compute node becomes idle, random and first-come first-served scheduling techniques

either picks a random task from the task queue, or picks the very first one in the queue. When a compute node becomes idle, greedy staging minimization algorithm first scans the task queue for a task whose parent had been executed on the same compute node; if such task exists it is picked and executed on that node. Otherwise, the first task in the queue is executed in that node.

3.3 ADAPT Prototype Implementation

ADAPT attempts to materialize many of the ideas presented in the Section 3.2. MELCOR [19, 24] has been used as an example of a plant simulator. MELCOR consists of four main components: the driver, the user tools, the scheduler process, and the database.

3.3.1 Driver

The developed driver interfaces with existing plant simulators, such as MELCOR [19, 24], in order to assimilate dynamic data inputs. In the current version of ADAPT, the driver requires that a plant simulator SIM provides following four features

• SIM reads its input from command-line and/or text file

• SIM has check-pointing feature

• SIM allows user-defined control-functions (e.g. stopping if a certain condition is true)

• SIM output can be utilized to detect stopping condition

If a simulator provides these features it can be used in ADAPT without any modifications. Luckily, many available plant simulators provide these four basic features.

Figure 2 illustrates the driver's workflow and its interaction with the plant simulator. The driver takes a templated version of the simulator's input file(s). Upon hitting a branch condition, it is driver's responsibility to prepare the input file, for branches, using an *edit-rules* file. In the current prototype the edit-rules file and templated input file(s) are provided by the application user. The long-term goal here is to develop a compiler that will read user-friendly branching rules and generates simulator specific edit-rules.

3.3.2 Client Tools

A stand-alone Java based GUI user interface (see Figure 3) has been developed as well as a Web Portal (see Figure 4) that will allow user to submit new initiating events, monitor the generation of dynamic event trees, checkpoint (pause) a running experiment, re-start a check-pointed experiment, and provides some analysis functionality of simulation results. The Java Client can operate on any system which can support the Java Runtime Environment (JRE) version 1.5. It supports the same functionality as the ADAPT web portal (experiment submission, experiment checkpointing, etc.) , but also allows for real-time monitoring of the experiment progress and has built in analysis capabilities which include but are not limited to: analyzing event tree structure and determining the probability of all scenarios, downloading and displaying plot data from simulator output files utilizing the STORM middleware, and the ability to monitor multiple experiments and make cross-experiment comparisons of the results.

Moreover a set of command-line tools is provided for more experienced users. Some of these tools are:

- An experiment submission tool, which gives the user the ability to launch a new experiment using driver.

- monitoring tools, which inform the user what progress has been made in executing any of the current experiments.

- a command to halt the execution of all current branches, and terminate the scheduler process, such that no branches are being executed. This effectively acts as a checkpoint, such that when resumed, the scheduler process can re-execute the terminated branches continuing where they left off. Please note that no branches that already completed successfully will need to be re-executed.

- a command to aggregate all file based outputs for a given experiment into a common location.

- a command to remove an experiment from the system, which involves cleanup of any metadata about the experiment, as well as cleaning up any file based outputs created by any branch executions.

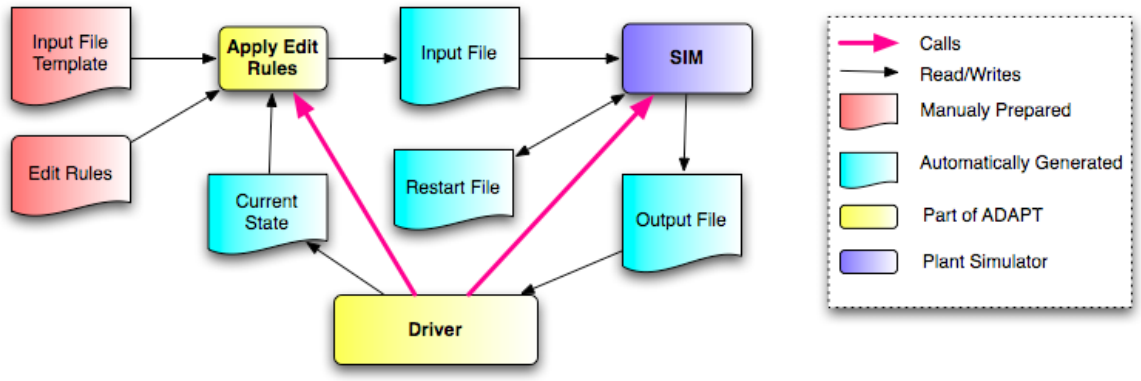


Figure 2. Driver.

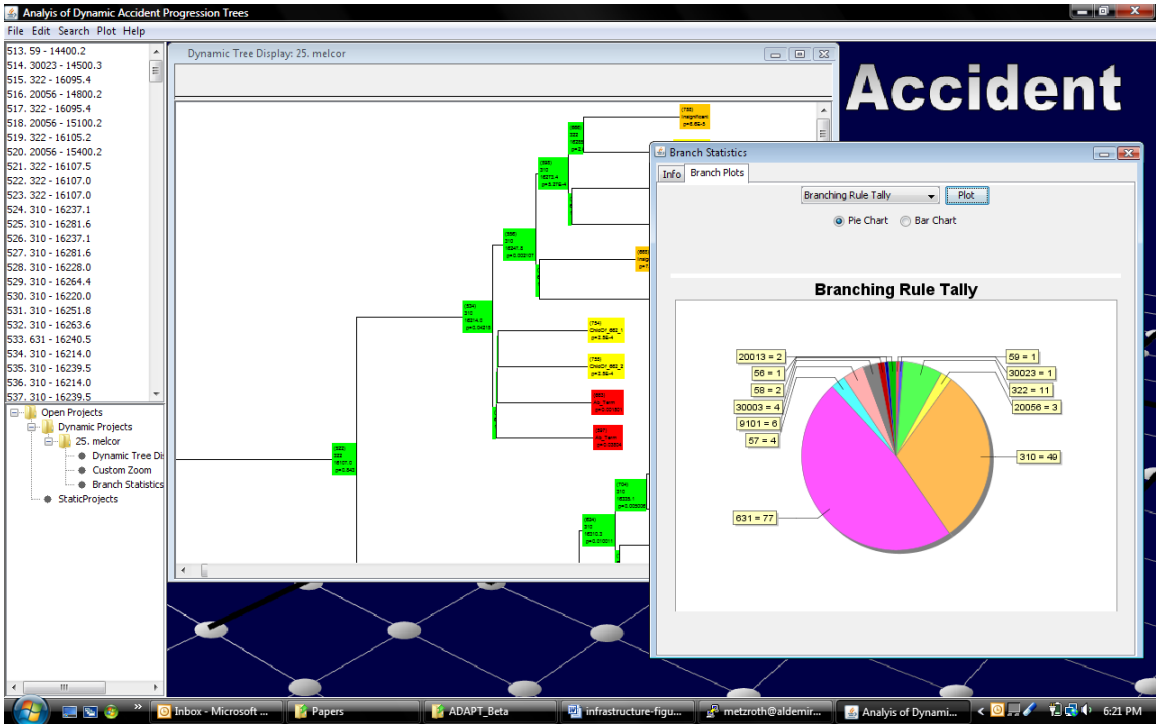


Figure 3. Stand-alone Java Client.

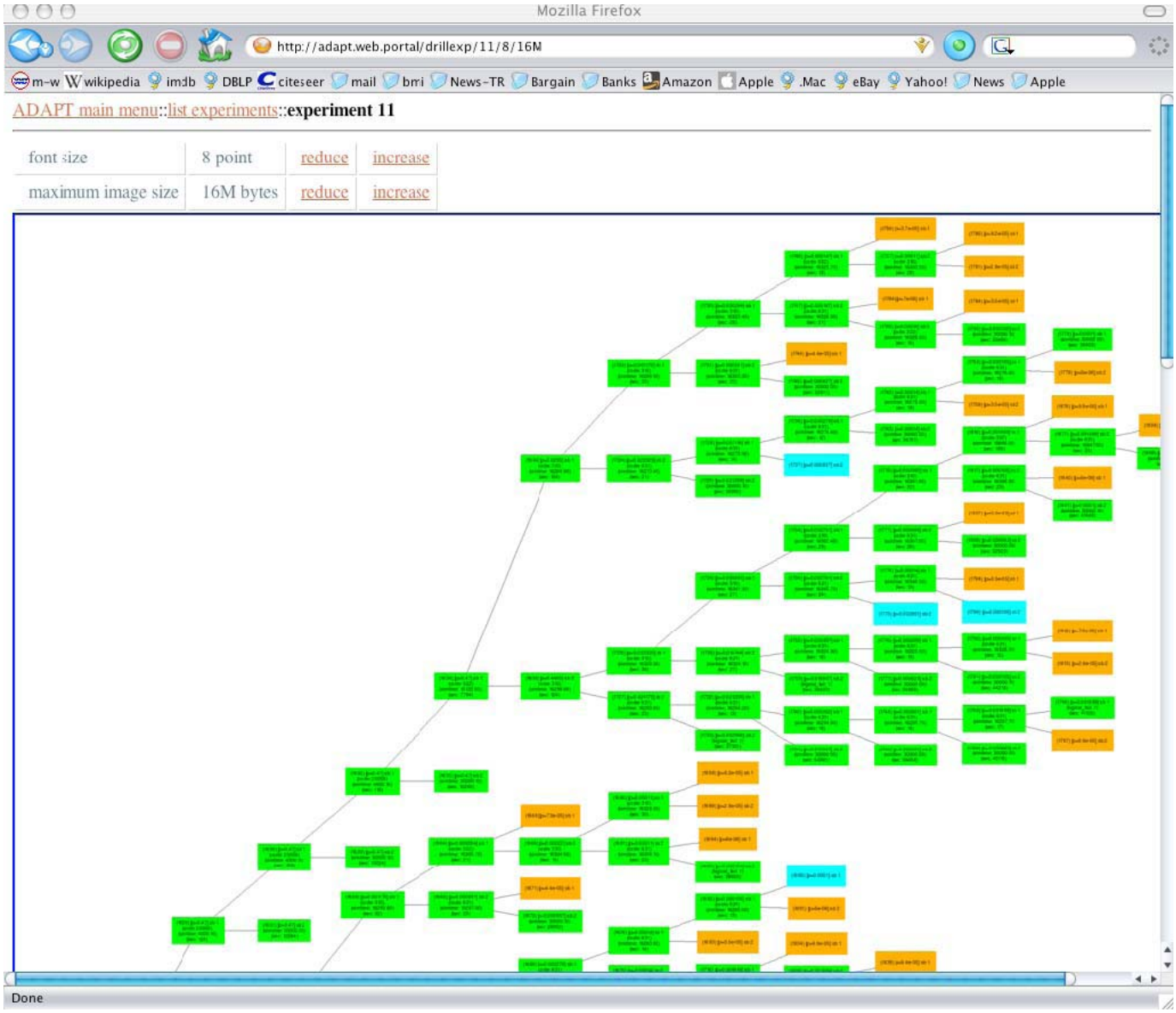


Figure 4. Web Portal.

3.3.3 Scheduler

The scheduler process is responsible for determining when and where a new branch of a dynamic event tree should be executed. As input, the scheduler process takes a list of compute nodes, output directory locations, a pointer to the database for metadata storage, and the desired remote shell (e.g. rsh, ssh) commands. The scheduler process will execute any branches that have not been yet executed, by managing the pool of available compute nodes. The order and the compute node that a branch will be executed depend on the scheduling algorithm chosen when the scheduler is initiated. Current scheduler runs

continuously as a background process and hence executes branches on compute nodes as long as there are new branches to be executed or until it is halted. When it is halted, it will checkpoint any branches that were currently running but had not yet completed; so that when the next time ADAPT server is re-started these branches can be resumed from the point they stopped.

As mentioned in Section 3.2.3, the current prototype has a pluggable scheduling interface, and out-of-box it provides three basic scheduling techniques: 1) random scheduling, 2) first-come first-served scheduling, and 3) greedy staging minimization. The default technique is greedy staging minimization but user can choose to use a different technique.

3.3.4 D Database

For distributed access to user created files, the STORM middleware has been leveraged. STORM is a middleware [48, 49] that is designed to provide basic database support for: 1) Selection of the data of interest, and 2) Transfer of data from storage nodes to compute nodes for processing. STORM's default binary and text file extractor object have been leveraged to read the data directly from simulation outputs. A customized STORM Java Client Object has been developed that interacts with ADAPT's Metadata Server and builds the required STORM database initialization files on-the-fly for dynamic even trees that are concurrently being generated. Hence, using the Java Client, the end-user can query the simulation output files both while dynamic event trees are being generated and after the trees have been generated.

In the current version of ADAPT, a relational database is used, in particular MySQL v5 [50] to store the metadata[6]. A MySQL database, with transaction supported tables, stores information about which experiments are in the system, in either a completed or incomplete state. An experiment consists of one or more branches. For each branch, it is in the state of being queued, running, or completed. Both the user tools and the scheduler process interact with the database. When a given branch issues sub-branches at the end of its execution, the update of the completion state and the submission of the sub-branches are collected together into one transaction, to maintain consistency.

[6] Storing the metadata in a static structure imposed by a relational database is definitely not the most desirable approach. In order to provide a more flexible solution our long-term goal is to use an XML database.

4. Demonstration

Since ADAPT must be utilized in consort with a pre-existing accident simulator, demonstration cases were performed with the MELCOR severe accident analysis code, as briefly described in Section 2.3. The MELCOR code has progressed from a probabilistic risk assessment tool, as it was originally developed and intended for, to a best-estimate severe accident analysis code. It has moderately complex system nodalization capability and physical models for simulation of plant system thermo-hydraulic behavior. It has a simulating capability for the containment behavior and evaluation of source term to the environment. Also, it has a good modeling flexibility through the use of control volumes for plant nodalization, control function approach to model an accident scenario of interest, and built-in sensitivity coefficients that allow the user to change a large number of modeling parameters via input, thus significantly facilitating the process of sensitivity.

Although an existing input deck for the Zion Nuclear Power Plant was used in the demonstration, the results are not intended to be representative of the behavior of any specific plant. Our objective is *not* to perform a PRA for a particular plant but to develop an advanced methodology for doing so and demonstrate the utility of that methodology.

Details of the plant model used in this demonstration are described in this section. The branching rules and their associated probabilities required for ADAPT input are detailed, as well as their phenomenological justification.

4.1 Reference Plant Nodalization

The input deck utilized in these studies is the model of the Zion Nuclear Power Plant, a Westinghouse-type PWR with a large dry containment. This plant has four loops each with a U-tube Steam Generator and a Reactor Coolant Pump. One of the loops also contains a pressurizer connected to the hot leg. Figure 5 [51] shows the MELCOR nodalization for this input deck. Figure 6 [52] gives the nodalization of the reactor itself as well as a modified nodalization of the pressurizer-leg steam generator. The steam generator nodalization shown in the one which is used in all experiments presented here.

The reactor pressure vessel contains the core with fuel assemblies, control rods, support structures, the upper plenum with inlet and outlet nozzles, the downcomer region and the upper head. The core itself is represented by a 5-ring, 12 level model with three core control volumes per thermal-hydraulic level and 10 heated levels. The thermal-hydraulic nodalization of the vessel core region is divided into a 5-ring, 4-level control volume (CV) geometry (CV 341-5, 351-5, 361-5, 371-5, 381-5). The upper plenum is divided into a 5-ring, 2 level nodalization with a control rod housing volume for each ring (not shown). The rest of the reactor vessel area is represented by three nodes: Reactor

Pressure Vessel downcomer (CV 310), lower plenum (CV 320), and upper head (CV 399) [51].

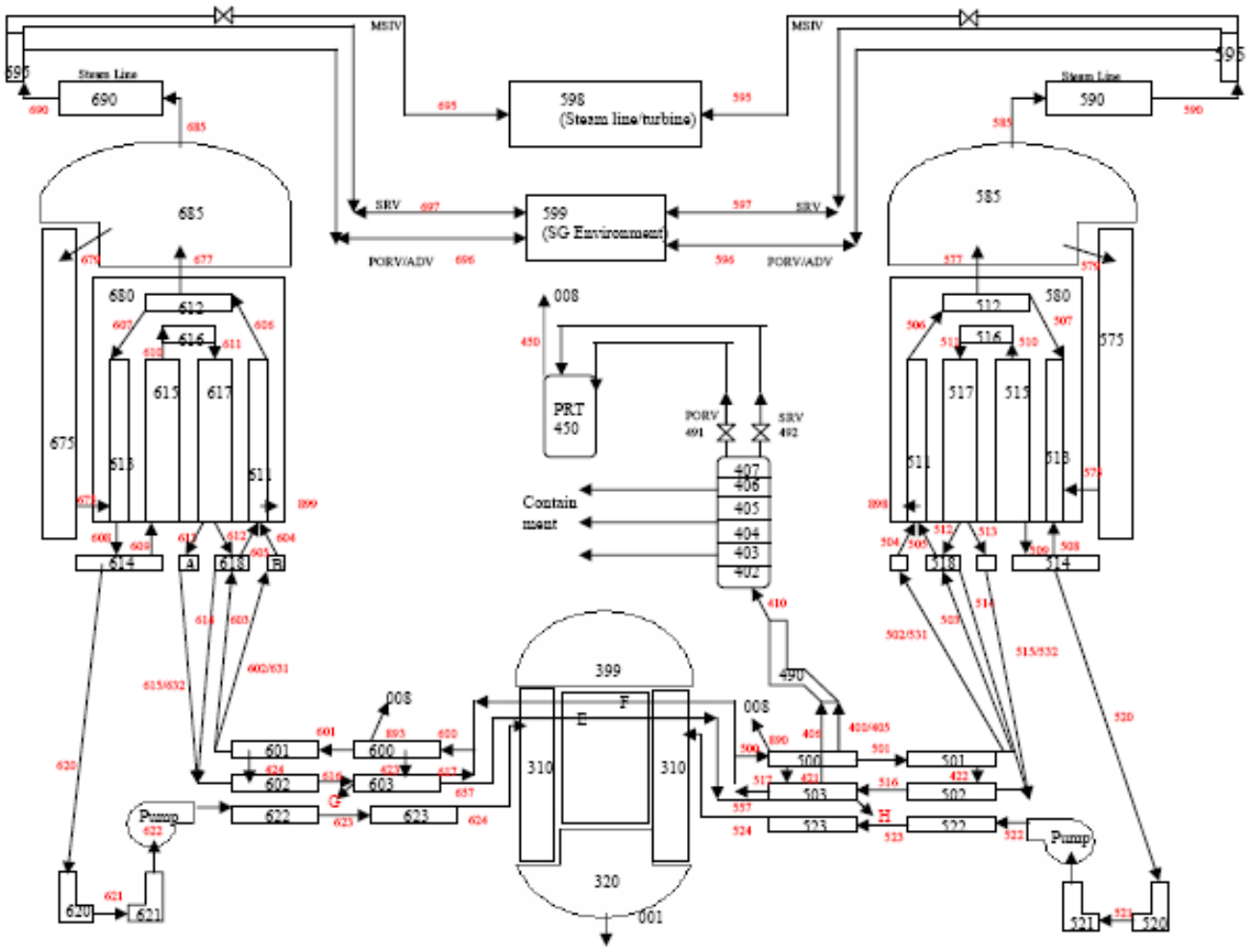


Figure 5. Reference Plant Nodalization.

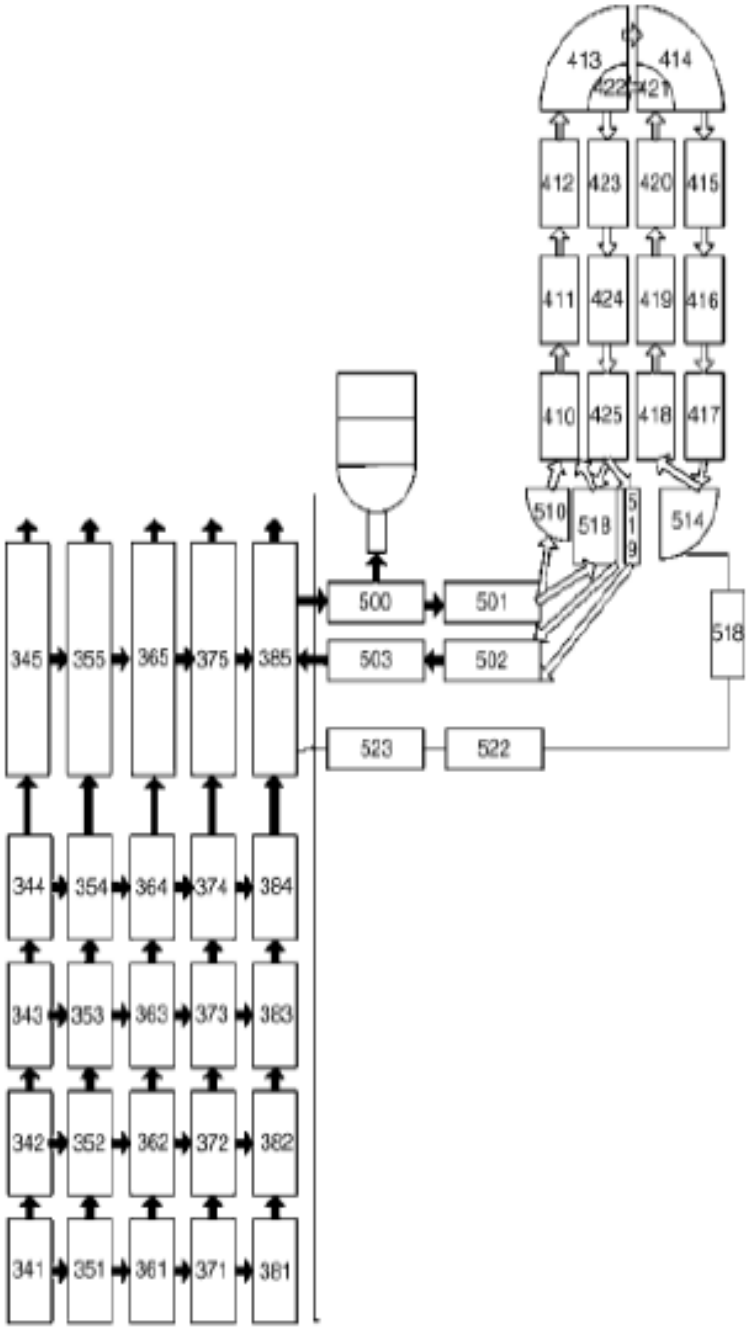


Figure 6. MELCOR Nodalization of Natural Circulation.

The steam generator consists of both the primary and secondary sides of the plant. The primary side contains thousands of steam generator tubes as well as inlet and outlet nozzles. The secondary side of the steam generator (SG) contains the feedwater inlet nozzle and associated lower plenum as well as the steam outlet nozzle connected to the upper plenum.

The four Reactor Coolant System (RCS) loops are modeled as two loops, one representing the loop containing the pressurizer and a triple loop representing the other three loops. The nodalization for these two loops is identical with the exception of the pressurizer in the single loop. The hot leg for each loop is divided into two directions (CV 501-4, CV 601-4), each direction containing two nodes. This is to account for steam counter flow from the steam generators during accident situations. The primary side of the steam generator has a finer nodalization scheme. The SG inlet plenum is divided into four nodes (CV 510,514, 518,519), the rising tube region into six nodes (CV 410-2, CV 418-420), the cross tubes into four nodes (CV 413-4, 421-2), the downside tube region into six nodes (CV415-7, 423-5), and finally the SG outlet plenum is represented with a single node (CV 518). Finally, the cold leg has four nodes: two before (CV520-1, 620-1) and two after the Reactor Coolant Pump (CV 522-3, 622-3).

Figure 6 [52] illustrates modeling of natural circulation in the system during the accident progression. As shown, the hot leg is modeled in two sections, an upper half and a lower half. The halves are connected by flow paths (FL 421-2), which allow mixing between them. Such a division of the hot leg is necessary to model the flow of steam to and from the steam generators because countercurrent flow of a single fluid cannot be calculated in a single control volume.

The overall pressurizer volume including the surge line is divided into seven nodes: six in the pressurizer itself (CV 402-7), and one for the surge line (CV 490). The pressurizer relief tank is represented by a single node. No control volumes are allocated to pressurizer PORVs (Pressure Operated Relief Valve) and SRVs (Safety Relief Valve). Instead, fluid removal from the pressurizer through these valves is simulated using ‘flow paths’ (FL 491 for PORV, FL 492 for SRV).

The nodalization of the secondary loop is rather simplistic (see Figure 5). The SG secondary side consists of just three nodes: the downcomer, the boiler, and the dome (not shown). The rest of the secondary side includes two control volumes for the main steam line for the single (CV 590) and triple loops (CV 690), two control volumes to represent main and auxiliary systems (CV 595, 695), and another two control volumes, one for the main turbine (CV 598), and the other for the SG environment (CV 599). The containment volume is divided into four nodes: cavity, lower compartment, annular compartment, and upper compartment. The SG steam line consists of two nodes, with the corresponding relief valves modeled through ‘Flows’. One node is allocated to the turbine, and one to the SG environment. Finally, the general environment is represented by two nodes (not shown).

The secondary side of the plant begins with the main steam line directed to the main turbine. After the turbines, the steam condenses back to water in the condenser using service water (e.g., from a nearby lake or pond) as coolant, and is pumped back to the steam generator (not shown).

4.2 Initiating Event and Accident Progression

Because Station Blackout with loss of Auxiliary Feedwater System has the potential for high consequences, it was chosen as the reference initiating event for the severe accident simulation. In the scenario analyzed, one of the relief valves on the SG Main Steam Line (CV 590) is assumed to stick open. An initial small leakage is assumed from the RCP seals following loss of offsite power. Due to the loss of pressure in the Steam Generators, the water on the secondary side begins to boil quickly and the water level decreases. This results in a rapidly degraded heat removal capacity from the primary coolant. After all the water in Steam Generators boils off and no more heat is removed by SGs, the added decay heat from the reactor core to the primary coolant makes the RCS pressure rise steadily. Due to the loss of power to the pressurizer PORVs (Valve 491), they do not respond to the RCS pressure increase above the setpoint for PORV opening. Instead, upon reaching a higher pressure setpoint, the passive Safety Relief Valves (Valve 492) open dumping the primary system inventory to the Pressurizer Relief Tank (CV 450). The system pressure then cycles in response to SRV openings and closing.

After a period where the SRV cycles, the Pressurizer Relief Tank becomes full and its rupture disk fails resulting in flashing of the hot steam to the containment. With each SRV opening, the primary coolant system loses a part of its inventory, which in time leads to the uncovering of the reactor core. The uncovered part of the reactor core starts to melt damaging both the fuel and the cladding. The hot radioactive gases are released to the in-vessel space and travel through the hot leg (CV 500-503) to the Pressurizer (CV402-407) and Steam Generators. The superheated steam and hydrogen flow results in hot leg, Pressurizer surge line (CV 490), and steam generator tube heating with a potential of a failure by creep rupture. Eventually the core becomes fully uncovered, with extensive damage to the fuel and cladding. With the growing creep rupture challenge to the hot leg, surge line, and steam generator tubes (CV 410-425), eventually one of those components fails due to creep rupture, thus rapidly depressurizing the primary coolant system. If the steam generator tubes fail first, the radiological consequences are most severe because of the release of radioactive material to the environment by containment bypass. Otherwise, if the surge line or hot leg fails first, rapid depressurization of the primary system relieves the thermal load on steam generator tubes and prevents their failure [53, 54, 55]. The entire lost RCS inventory ends up in the containment (not shown), adding a large amount of heat to its volume and increasing the containment pressure. Since there is no power in this scenario to operate a heat rejection system to relieve the increase in pressure, if power is not restored, with time the pressure will rise to a point at which the containment fails.

Another contributor to the pressure growth in the containment is Reactor Coolant Pump seal leakage events. Different leak sizes are possible based on different models for seal degradation. The size of such a leak can also significantly affect the accident progression path. For example, a large enough leakage can lead to an early primary system depressurization considerably relieving the stress on the hot leg, surge line, and SG tubes, which may prevent creep rupture of those components.

Two other critical events can have a substantial impact on the accident path and consequence: a hydrogen deflagration or detonation in the containment and the possibility of power recovery. Hydrogen is produced in the reactor core as a result of the steam-zirconium reaction. Hydrogen can be released to the containment through SRVs, failure of either the hot leg piping or the surge line, or due to failure of the lower head of the reactor vessel. Depending on the concentration of hydrogen along with other diluents, a hydrogen deflagration or detonation can occur having an immediate threat to the containment.

Recovery of AC power can have a variety of impacts on accident progression depending on the time of power recovery. If it occurs before core damage has begun, the accident can be arrested without any core damage or significant radiological consequences. Core damage may also be arrested as in the Three Mile Island Unit 2 accident, if power recovery occurs early enough in the severe accident progression process. Similarly, if power is recovered following failure of the lower head of the vessel, but before containment failure, active containment cooling systems will arrest the continued rise in containment pressure and containment failure can be averted.

4.3 Branching Rules

For the test case, the authors formulated and utilized the following branching conditions for this demonstration:

- Power Recovery
- Creep Rupture of RCS Components
- Hydrogen Burn in the Containment
- Containment Failure Due to Overpressure

These phenomena represent four distinct areas of accident progression analysis—reactor system performance, reactor coolant response, containment challenge, and containment response—so that the breadth of situations encountered in a complete APET analysis is considered in the demonstration. All of these phenomena are calculated deterministically within MELCOR with no consideration of what the uncertainties may be and how they might affect the calculations. In order to consider these uncertainties in the analysis, probability distributions were assigned for each of the items listed above. These

distributions are discretized with each discrete point representing a branching condition. Each of the modeled phenomena for this demonstration is described in further detail in the following subsections.

4.3.1 Creep Rupture of RCS Components

During this Station Blackout scenario, as the core begins to overheat, hot gases travel through the RCS increasing the temperature of structures and decreasing their strength. The three components of concern are the steam generator tubes (SG Tubes), the hot leg, and the surge line to the pressurizer. If SG tubes fail before the hot leg or the surge line, a pathway will exist for direct radionuclide release to the environment. However, failure of the surge line or hot leg first will cause depressurization of the RCS which will preclude rupture of the SG tubes. In MELCOR, the criterion for rate-dependent creep rupture is based on the time-fraction damage integral [56]:

$$\int_0^{t_f} \frac{dt}{t_R\left(T, m_p\sigma\right)} = 1$$

where

t_f = creep rupture failure time

t_R = time to rupture as a function of T (t),

σ = stress in the pipe wall

m_p = intensity factor associated with a flaw in the wall (assumed to be unity)

The value of t_R in the denominator is given by the Larsen-Miller correlation [57] and is calculated by MELCOR. The form of t_R differs with different materials. For the hot leg and surge line (SS316), it is

$$t_R = 10^{\frac{p}{T}-20}$$

$$p = -13320\log\sigma + 54870$$

for SG tubes (Inconel 600), it is

$$t_R = 10^{\frac{p}{T}-15}$$

$$p = -13320 \log \sigma + 54780$$

where

p = pressure inside the pipe (kPa),
T = temperature of the structure (K),
σ = mechanical stress in the structure (log σ given in kPa).

Like any correlation, the Larsen-Miller correlation has an associated uncertainty. To represent this uncertainty, the following value is calculated

$$R = \int_0^{t_f} \frac{dt}{t_R\left(T, m_p \sigma\right)}$$

with R known as the *creep rupture parameter*. Normally, a wall is assumed to fail when R reaches unity. For the case studied, if the uncertainty in the creep rupture parameter is not taken into account, surge line rupture will always precede and preclude rupture of the steam generator tubes. In order to quantify the uncertainty in R, a cumulative distribution function (CDF) was constructed from experimental data in the form of a lognormal distribution [56]. The distribution is given as

$$\Phi(R) = \int_0^R \frac{\exp(-\frac{1}{2}\left(\frac{\ln(R')}{0.4}\right)^2}{0.4R'\sqrt{2\pi}}$$

where Φ(R) is called the *fragility curve*. For the purpose of analysis in the ADAPT framework, this CDF is discretized into 5 points where the points correspond to probabilities of 5%, 25%, 50%, 75%, and 95%. The corresponding R values are 0.518, 0.764, 1.00, 1.31, and 1.931. These points are now used as branching criteria. When MELCOR calculates an R value of 0.518, the execution will stop and two new branches will be created. One branch will continue executing with a rupture while the other will have no rupture and the threshold of R will be increased to the second point, 0.764 and so

it goes until there are no points left. Figure 7 shows a graphical representation of the fragility curve as well as the branch points at which the analysis is interrupted.

4.3.2 Power Recovery

The station blackout scenario is typically found to be an important contributor to nuclear power plant risk. In the absence of all AC power, key emergency systems with the purpose of heat removal from the core will fail to actuate potentially leading to core damage. There is always a chance, however, that power will be recovered at some point during the scenario. Power recovery at a critical moment in the severe accident evolution can lead to the arresting of core damage or preventing containment failure. It is also possible that power recovery could lead to worse consequences. For example, a high concentration of steam in the containment atmosphere can prevent a hydrogen combustion event. Resumption of AC power could not only result in a decrease in steam concentration but also produce sparks igniting the hydrogen and carbon monoxide in the containment and producing an explosion. Both the potential positive and negative effects of power recovery must be considered.

There are several categories of loss of offsite power (LOOP) events which can include plant centered, switchyard centered, grid related, and weather related. With the use of composite data (considering all types of LOOPs) from NUREG/CR-6890 [58] (see Table 1) the probability of power recovery was assessed as a function of time as a log-normal probability density function.

$$f(t) = \frac{1}{t\sigma\sqrt{2\pi}} e^{-\frac{(\ln(t)-\mu)}{2\sigma}}$$

The non-recovery probability is thus the complementary cumulative distribution function

$$F(t) = erf\left(\frac{\ln(t)-\mu}{\sigma}\right)$$

where

t = offsite power recovery time
μ = mean of natural logarithm of the data
σ = standard deviation of natural logarithm of the data

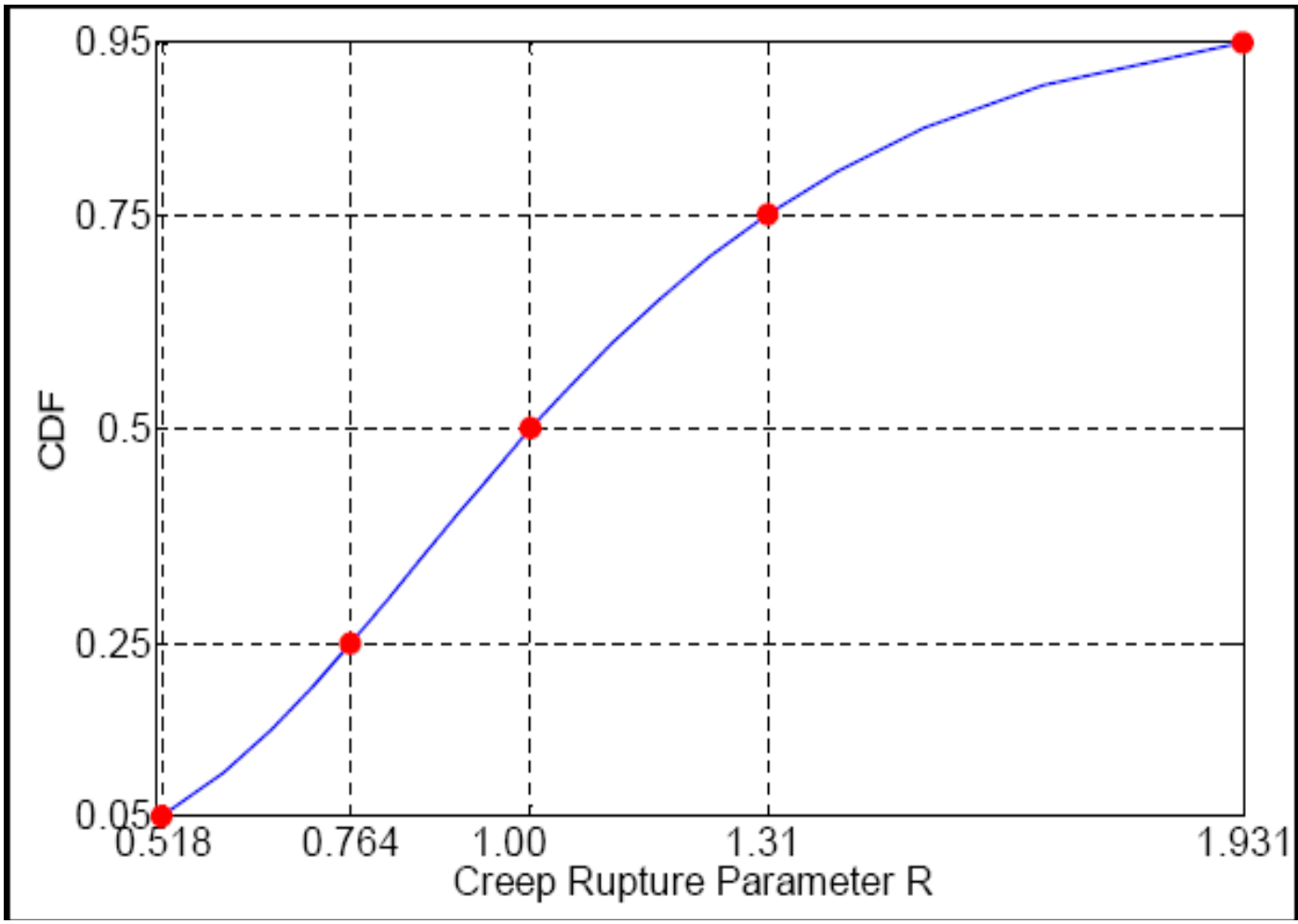


Figure 7. Graphical representation of the discretization of the creep rupture curve.

Table 1. Probability of exceedance versus duration curve fits and summary statistics [58]

	LOOP Category (Critical or Shutdown Operation)					Critical Operation		Shutdown Operation	
Duration (h)	Plant Centered	Switchyard Centered	Grid Related	Weather Related	Combined Plant and Switchyard Centered[a]	Composite[b]	Actual Data	Composite[b]	Actual Data
0.00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00
0.25	6.87E−01	7.86E−01	9.43E−01	8.64E−01	7.53E−01	8.72E−01	8.52E−01	7.82E−01	7.31E−01
0.50	4.79E−01	5.95E−01	8.25E−01	7.73E−01	5.56E−01	7.31E−01	6.48E−01	6.08E−01	4.63E−01
1.00	2.77E−01	3.78E−01	6.11E−01	6.56E−01	3.44E−01	5.30E−01	4.63E−01	4.13E−01	2.99E−01
1.50	1.83E−01	2.63E−01	4.61E−01	5.78E−01	2.36E−01	4.03E−01	3.89E−01	3.08E−01	2.09E−01
2.00	1.29E−01	1.94E−01	3.56E−01	5.20E−01	1.73E−01	3.18E−01	2.22E−01	2.44E−01	1.79E−01
2.50	9.64E−02	1.49E−01	2.81E−01	4.75E−01	1.32E−01	2.58E−01	1.85E−01	2.00E−01	1.64E−01
3.00	7.44E−02	1.18E−01	2.27E−01	4.39E−01	1.04E−01	2.15E−01	1.48E−01	1.69E−01	1.49E−01
4.00	4.77E−02	7.86E−02	1.54E−01	3.82E−01	6.87E−02	1.57E−01	1.30E−01	1.29E−01	1.34E−01
5.00	3.28E−02	5.57E−02	1.09E−01	3.40E−01	4.85E−02	1.20E−01	9.30E−02	1.04E−01	9.00E−02
6.00	2.37E−02	4.11E−02	8.05E−02	3.07E−01	3.57E−02	9.63E−02	5.60E−02	8.64E−02	9.00E−02
7.00	1.78E−02	3.14E−02	6.10E−02	2.80E−01	2.72E−02	7.95E−02	5.60E−02	7.42E−02	9.00E−02
8.00	1.37E−02	2.46E−02	4.73E−02	2.58E−01	2.13E−02	6.72E−02	3.70E−02	6.49E−02	9.00E−02
9.00	1.08E−02	1.97E−02	3.73E−02	2.39E−01	1.70E−02	5.79E−02	3.70E−02	5.78E−02	7.50E−02
10.00	8.67E−03	1.60E−02	3.00E−02	2.23E−01	1.38E−02	5.07E−02	3.70E−02	5.21E−02	7.50E−02
11.00	7.07E−03	1.32E−02	2.44E−02	2.09E−01	1.14E−02	4.50E−02	3.70E−02	4.75E−02	6.00E−02
12.00	5.85E−03	1.10E−02	2.00E−02	1.97E−01	9.51E−03	4.04E−02	3.70E−02	4.36E−02	4.50E−02
13.00	4.89E−03	9.31E−03	1.67E−02	1.86E−01	8.03E−03	3.66E−02	3.70E−02	4.03E−02	4.50E−02
14.00	4.13E−03	7.93E−03	1.40E−02	1.76E−01	6.84E−03	3.34E−02	3.70E−02	3.75E−02	4.50E−02
15.00	3.52E−03	6.81E−03	1.18E−02	1.67E−01	5.87E−03	3.08E−02	3.70E−02	3.51E−02	4.50E−02
16.00	3.03E−03	5.89E−03	1.01E−02	1.59E−01	5.08E−03	2.85E−02	3.70E−02	3.30E−02	4.50E−02
17.00	2.62E−03	5.13E−03	8.66E−03	1.52E−01	4.43E−03	2.65E−02	3.70E−02	3.11E−02	3.00E−02
18.00	2.28E−03	4.50E−03	7.47E−03	1.45E−01	3.88E−03	2.48E−02	3.70E−02	2.94E−02	3.00E−02
19.00	2.00E−03	3.96E−03	6.49E−03	1.39E−01	3.42E−03	2.33E−02	3.70E−02	2.79E−02	3.00E−02
20.00	1.76E−03	3.51E−03	5.66E−03	1.33E−01	3.03E−03	2.20E−02	3.70E−02	2.66E−02	1.50E−02
21.00	1.56E−03	3.12E−03	4.96E−03	1.28E−01	2.69E−03	2.08E−02	3.70E−02	2.53E−02	1.50E−02
22.00	1.38E−03	2.79E−03	4.37E−03	1.23E−01	2.41E−03	1.97E−02	3.70E−02	2.42E−02	1.50E−02
23.00	1.24E−03	2.50E−03	3.86E−03	1.19E−01	2.16E−03	1.88E−02	3.70E−02	2.32E−02	1.50E−02
24.00	1.11E−03	2.25E−03	3.42E−03	1.14E−01	1.94E−03	1.79E−02	1.90E−02	2.22E−02	1.50E−02

	Lognormal Fits[c]				
	Plant Centered	Switchyard Centered	Grid Related	Weather Related	Combined Plant and Switchyard Centered[a]
p-value	>0.25	>0.25	>0.25	>0.25	>0.25
Mu (μ)	−0.760	−0.391	0.300	0.793	−0.512
Sigma (σ)	1.287	1.256	1.064	1.982	1.278
Curve Fit 95% (h)	3.88	5.34	7.77	57.60	4.90
Curve Fit Mean (h)	1.07	1.49	2.38	15.77	1.36
Actual Data Mean (h)	1.74	1.41	2.43	14.21	1.52
Curve Fit Median (h)	0.47	0.68	1.35	2.21	0.60
Actual Data Median (h)	0.30	0.67	1.56	1.28	0.50
Curve Fit 5% (h)	0.06	0.09	0.23	0.08	0.07
Error Factor (95%/median)	8.31	7.89	5.76	26.07	8.19

a. For plant risk models that combine the plant-centered and switchyard-centered LOOPs, this column should be used.

b. The composite curve is a frequency-weighted average of the four individual category curves. Frequencies are presented in Table 3-1.

c. The LaCrosse and two Pilgrim events were excluded from these analyses. See Appendix A, Table A-1 for more information.

Five time points from the power recovery were selected as branching points. The values chosen were 1 hour, 2 hours, 4 hours, 6 hours, and 8 hours with corresponding probabilities of non-exceedance at 47%, 68.2%, 84.3%, 90.37%, and 93.28%, respectively. During MELCOR execution, when the first branching point is reached (1 hour), the code execution stops and two child branches are create. One branch runs with power restored at a 47% probability and those runs with no power recovery at a 53% probability. For the latter, the value of the power recovery time is increased to 2 hours (the next discrete point) and this process repeats until all discrete points are exhausted.

4.3.3 Hydrogen Burn

Because of the magnitude of the deflagration event that occurred in the containment building in the TMI-2 accident, there has been considerable interest in hydrogen combustion in the severe accident research community [59]. A major deflagration event in the containment could potentially cause a pressure spike great enough to challenge the integrity of the containment walls. Hydrogen is produced during a severe accident in the reactor core through the steam-zirconium reaction. Hydrogen travels through RCS components including the SG and the pressurizer. In the case that is analyzed, hydrogen is released to the containment through the SRV due to high pressure in the primary system. Following a failure of the lower head of the reactor vessel, the hot leg piping, or the surge line, hydrogen can also be released to the containment volume.

In order for hydrogen combustion (or the reaction $H_2 + \frac{1}{2}O_2 \rightarrow H_2O$) to take place, the concentration of both hydrogen and oxygen must be sufficiently high as well as temperature of the reactants. Also, some diluents such as steam (H_2O gas) and carbon dioxide (CO_2) can act as suppressers to the hydrogen burn. There is a maximum limit of diluents combined concentration (approximately <55%) above which a hydrogen burn is prevented. Containment structures can act as heat sources needed to initiate hydrogen ignition. Carbon monoxide (CO) is produced in the later stages of the severe accident when the molten core attacks concrete. The presence of CO increases the potential for a propagating combustible gas explosion [60].

MELCOR utilizes LeChatelier's formula [61] to determine the threshold of combustible gas ignition provided that oxygen and hydrogen are within combustible limits and the combined concentration of inert gases is less than the maximum required levels, i.e.

$$\frac{n(H_2)}{N(H_2)} + \frac{n(CO)}{N(CO)} \geq 1$$

where *n* is the actual volumetric concentration (mole fraction) of gases in the containment atmosphere calculated by MELCOR and *N* is the flammability limit (mole fraction) of individual gases. While MELCOR-calculated quantities still have modeling uncertainties, the major source of uncertainty in the above is the input values of the flammability limits of the individual gases. These experiments show that these values can be significantly different for different sizes and geometries of test vessels with different direction of flame propagation (upward or downward) [62].

In order to represent the uncertainty on the flammability limits it was first assumed that $x = 1 / N(H_2)$ and $y = 1 / N(CO)$ are normally distributed about mean values $\mu_x = 1 / 0.01 = 10$ and $\mu_y = 1 / 0.16 = 6.25$ (where 0.1 and 0.16 are the MELCOR default values for the flammability limits of H_2 and CO respectively). Their corresponding standard deviations were chosen as 10% of the respective mean values, $\sigma_x = 1$ and $\sigma_y = 0.625$ based on experimental data [62, 63, 64]. By denoting the parameters $a = n(H_2)$ and $b = n(CO)$, and assuming they are fixed values for a given time point, the parameter $z = ax+by$ is formed. It is well known from statistics that *z* is also normally distributed with a mean value of $\mu_z = a\mu_x + b\mu_y$ and a variance of $\sigma_z^2 = a^2 \sigma_x^2 + b^2 \sigma_y^2$, i.e.

$$f(z) = \frac{1}{\sqrt{2\pi}\sigma_z} \exp\left(-\frac{(z-\mu_z)^2}{2\sigma_z^2} \right).$$

The corresponding cumulative distribution function is

$$F(z) = \int_0^\infty f(z)dz = \frac{1}{\sqrt{2\pi}\sigma_z} \int_0^\infty \exp\left(\frac{(z-\mu_z)^2}{2\sigma_z^2} \right) dz$$

Since the hydrogen ignition criterion is $1 \geq z$, however, the probability region of interest is

$$P(z \geq 1) = \frac{1}{\sqrt{2\pi}\sigma_z} \int_1^\infty \exp\left(\frac{(z-\mu_z)^2}{2\sigma_z^2} \right) dz = \frac{1}{2}\left(1 - erf\left(\frac{1-\mu_z}{\sqrt{2}\sigma_z} \right) \right)$$

At each time step, MELCOR calculates $N(H_2)$ and $N(CO)$. When the value of *P* reaches the first discrete point specified by the user, MELCOR checks the concentrations of oxygen and diluents. It also checks the temperatures of the structures to determine if the gas mixture will auto-ignite. If these values are in a range adequate for combustion, the execution stop and two branches are created. The first branch continues with a hydrogen burn occurring with probability *P* and the second branch executes with no hydrogen burn with probability *1-P*. For the branch without hydrogen burn, the probability threshold is raised to the next user-specified point. The process again continues until all user-specified branching points are exhausted.

4.3.4 Containment Overpressure Failure

The containment is the final barrier for the release of radioactive material to the environment. Containment failure modes are often categorized into *early* and *late* containment failures, both having differing consequences with regards to source term size (magnitude of release of radioactive material) and offsite effect. Early containment failures are characterized by larger source terms and insufficient time for an effective evacuation while late containment failure is characterized by smaller source terms and more time for an effective evacuation [65]. Since early failure tends to have more severe consequences it is usually emphasized in APET analyses. Many mechanisms are present that can lead to early containment failure such as rapid overpressurization from severe accident phenomena such as direct containment heating that can result from reactor pressure vessel lower head failure with the primary system at high pressure [66, 67] or explosive increase in pressure due to a steam explosion [68, 69] or from hydrogen deflagration or detonation [70, 71, 72]. The main mechanism of late containment failure is through slow pressure buildup because of failure of AC-powered containment heat removal systems.

While MELCOR cannot model the response of the containment to pressure loads, it can simulate the effect of containment failure on the accident evolution. The current MELCOR model considers containment pressure to be the dominant factor in containment failure, although other factors such as structure temperature can play a role [24]. Containment failure is modeled deterministically in MELCOR by causing a containment rupture when a user-defined pressure has been reached. Although the ultimate strength of the shell of the containment is known fairly accurately, it is difficult to assess the actual location and pressure level at which failure will occur. The most likely failure locations are at points of discontinuity, such as penetrations or the interface between the wall and base mat. Thus, containment failure pressure must be treated as a distribution.

Fragility curves for containment pressure are used to treat this uncertainty [73] in the form of CDFs for containment failure versus containment pressure. Since every plant has a unique design for its containment, different fragility curves are used not only for different types of containments but different plants with similar containment concepts. Figure 8 illustrates containment fragility curves for 5 different plants with a steel containment. The test case is not intended to be representative of a specific plant. The fragility curve for the Davis-Besse plant was used since it exhibits the highest failure probabilities for the lowest pressures.

For this study, a CDF $\Phi(P)$ was developed as a normal distribution

$$\Phi(P) = \frac{1}{\sqrt{2\pi}\sigma}\int_0^P e^{-\frac{(P'-\mu_p)^2}{2\sigma^2}}\,dP'$$

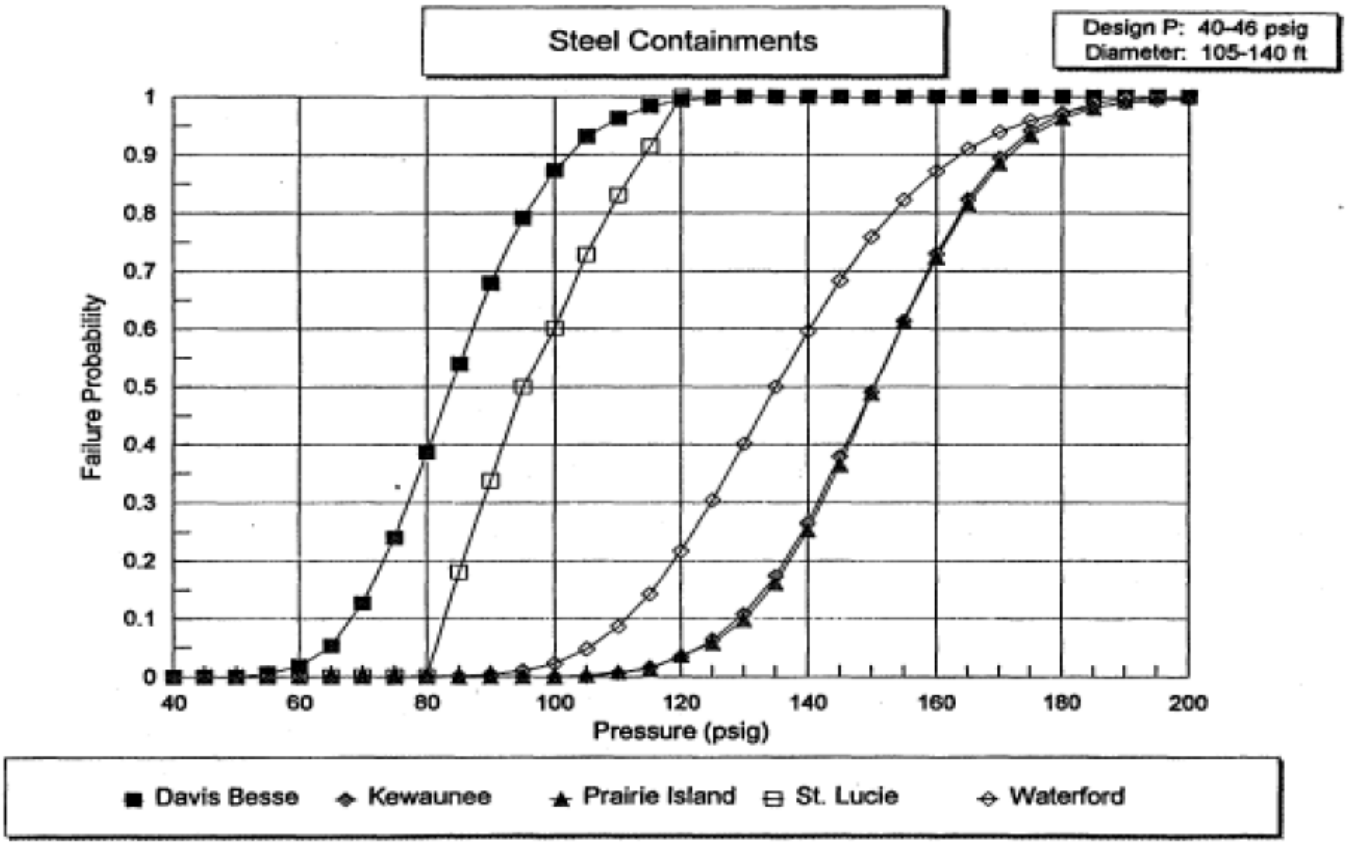


Figure 8. Containment fragility curves for five steel containments [73].

where μ_p = 85 psig is the mean value of this distribution and σ = 12 psig is the standard deviation obtained from curves in Figure 8. Once again this CDF was discretized into 5 points with probabilities 5%, 25%, 50%, 75%, and 95%. The corresponding pressure values are 65 psig, 77 psig, 85 psig, 93 psig, and 105 psig. When the MELCOR-calculated pressure reaches the first discrete point (65 psig), a branching occurs. Two new branches are created, one branch with a 5% probability and with containment failure, the second with a probability of 95% and no containment failure. For the second branch, the containment failure threshold is raised to the value of 77 psig and this process continues until all discrete points are exhausted.

4.4 Demonstration Results

The scenario described in Section 4.2 was executed in two different experiments on two different Linux clusters. In the first experiment (Experiment 1) the scenario was executed on a Linux cluster consisting of 8 nodes connected with a gigabit Ethernet switch. Each computer contained a Pentium 4 3.1 GHz processor, 3 GB of memory, and 2x250GB of

local serial ATA disks in a software RAID1 configuration. In the second experiment (Experiment 2) four simultaneous scenarios were carried out on a Linux compute cluster consisting of up to 40 compute nodes connected with a gigabit switched network. Each one of the compute nodes is equipped with dual AMD Opteron 250 processors running at 2.4 gigahertz, 2x250 GB of local serial ATA disks in a RAID0 configuration with 256KB block size, and 8 GB of memory. Experiment 1 served to test the speedup of a single experiment resulting from executing the scenario within the ADAPT framework as compared to running the scenario in a serial manner. Experiment 1 also served as a means of demonstrating some of the graphical features of the ADAPT Client.

Experiment 2 was performed to test the robustness of the ADAPT scheduling system with different scheduling schemes utilized. Each of these experiments will now be discussed in turn.

Experiment 1 had a total runtime of 9.3 days. This compares to a serial runtime of 72.1 days, resulting in a speedup factor of 7.8. The experiment resulted in a total of 197 different branch points, 54 of which were deemed "insignificant" (they fell below the experiment's probability threshold of 10^{-4}). There were also 74 "significant" scenarios pathways (a sequence of branches) on the tree (A significant scenario is one which ended with a probability higher than the experiment's probability threshold).
In Figure 9 a visualization of a portion of the event tree generated in Experiment 1 is shown. The branches in the tree are color coded based off of their status in the queue. In the figure, green branches signify branches whose execution in complete, orange branches are insignificant and red branches signify situations in which the simulator stopped via some non-user specified reason (an abnormal termination). The software can also show when branches are running, waiting in the queue, or have been paused by the user. The branches on the event tree are spaced based off of the simulation time at which the branch execution completed; hence the horizontal axis on the tree represents time. Since these trees may become very large, features are in place to navigate through the tree as well as select a desired branch from a list which will focus the event tree display on that branch.

In addition to event tree visualization, some analysis features have been implemented into the ADAPT client software. The user can view plots of certain simulator-output plot variables. While ADAPT does not currently have the ability to directly read simulator binary plot files, several plot variables were output to an external data file to demonstrate the ability of the client to retrieve the plot data. With plot data spread across several compute nodes, the STORM middleware was utilized to gather the data and return it to the client. Figure 10 shows an example of a plot that was made from ADAPT/MELCOR generated data. Also some other analysis tools are present to assist in obtaining some overarching results for the event tree. Figure 11 shows one such example of a chart which gives a tally of all different branches which appear on the tree.

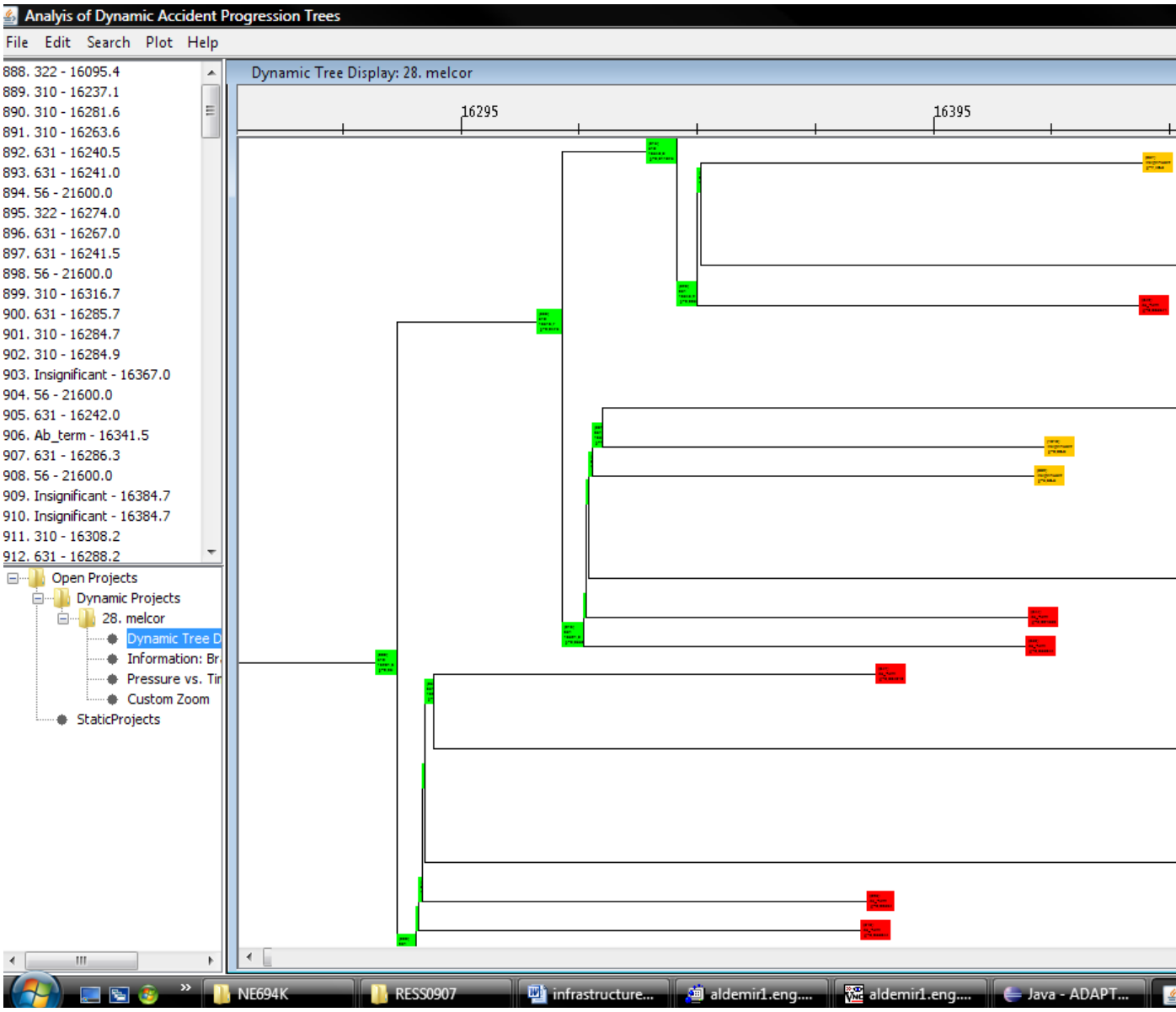


Figure 9. Picture of portion of event tree generated by Experiment 1. Green represent completed branches, orange represents insignificant branches, and red represents abnormally terminated branches.

Aside from event tree analysis capabilities the client software also has the capability to interact with the ADAPT server software and the cluster file system. From the client the user can launch experiments, pause them, or delete them from the database. Also, all files which are output from the simulator can be downloaded from the compute cluster to the user's hard drive.

Experiment 2 was been carried out using three different configurations: a 20-, 40- and 80-CPU configuration, using both CPUs per compute node, i.e. 10, 20 and 40 compute nodes, respectively. In the early stages of DET generation the number of branches (runnable jobs) will be less than available number of CPUs. Hence, system will not be able to utilize all of the available CPUs. Similarly, as the experiment is nearing completion, there are again fewer branches to be executed than CPUs. In other words, we can only obtain significant gains by increasing number of compute resources in the ``middle'' of dynamic even tree execution.

Figure 12.a shows that, as the number of CPUs increases, the average time that a branch spends waiting in the queue decreases significantly. Figure 12.b shows that the overall runtime decreases significantly as the number of CPUs are increased, although the speed up is not linear. Figure 13 plots the size of the queue and the number of active jobs over the course of the running experiments. With the 20-CPU case, the plateau where all CPUs are utilized is reached fairly quickly, and lasts almost the entire time. For the 80-CPU case, however, the period of full utilization of the cluster is only reached for about 50% of the time. The problem does not grow explosively enough in the early phase of the accident to realize a linear speedup with increased CPUs.

In Figure 14, the average time for each branch is presented in two parts: staging and execution. Staging time measures the time required to prepare the input files for the branch, and execution time is the time to execute a branch when all of its inputs are ready. Note that the execution time is identical regardless of the number of CPUs, since the workload is the same. For a greater number of CPUs (thus more compute nodes), the scheduler is less likely, however, to assign a job to the same compute node as its parent, which contributes to an increase in data staging time. Some data must be transferred from parent to child at each branch point. This handoff can be performed efficiently by creating symbolic links (a feature of the Linux operating system) between the parent and child, if they are on the same computer node.

The system developed in this work can exclude certain files from being handled in this way, in case symbolic links are not semantically valid and each child does need its own copy of every input. For the Station Blackout experiment, using symbolic links can reduce the staging time per branch to less than a second, compared to up to 45 seconds in the case that files must be copied between compute nodes. Thus, much of the time difference in Figure 12 is due to the increased chance that the 20-CPU job could use this optimization, versus the 80-CPU job. Although it didn't make a dramatic difference in average runtime for these experimental configurations, for other shorter-running jobs, the difference in staging time could be more significant.

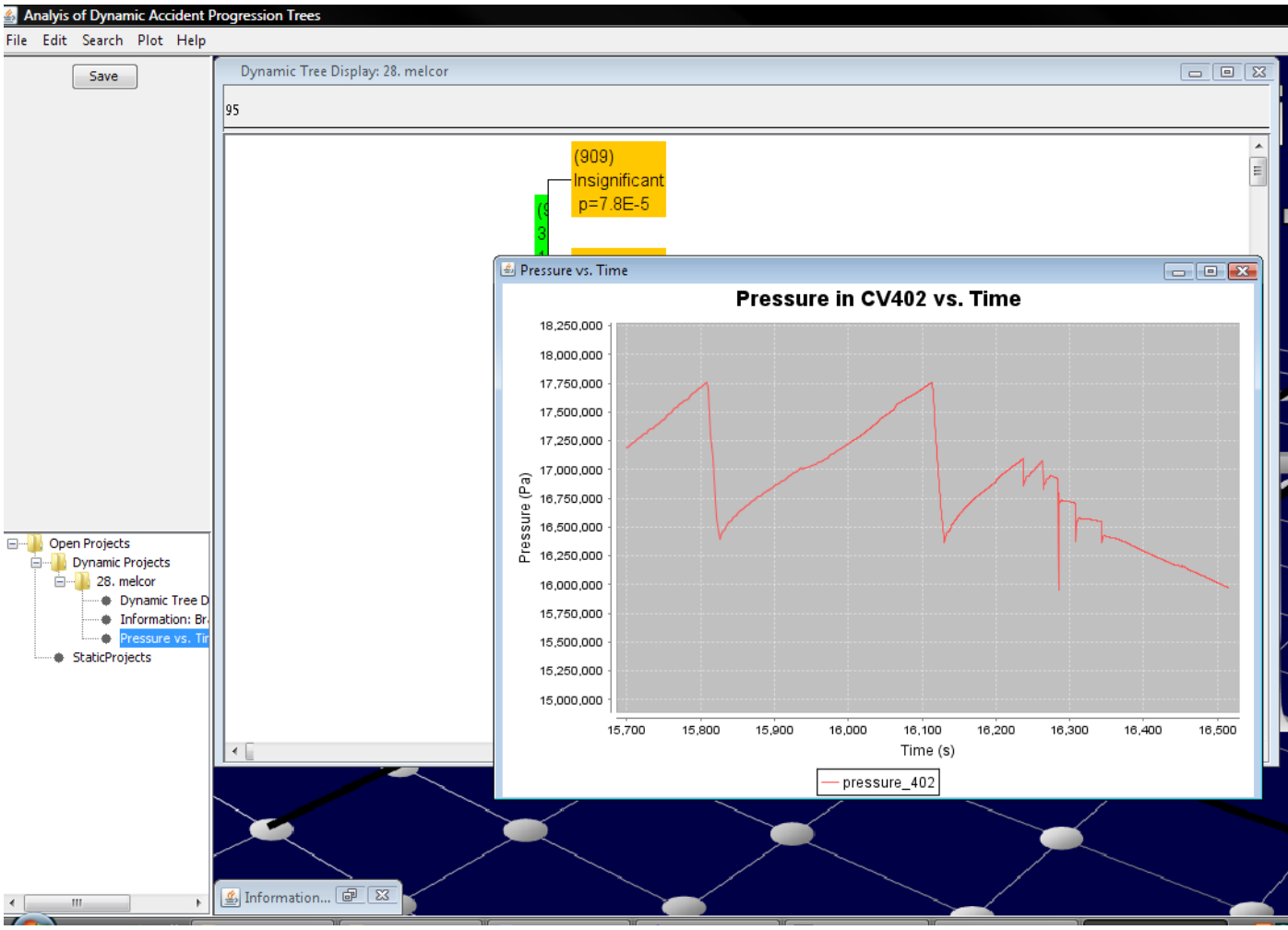


Figure 10. Snapshot of plot generated by ADAPT using MELCOR data. This plot shows the pressure in control volume 402, a volume of the pressurizer.

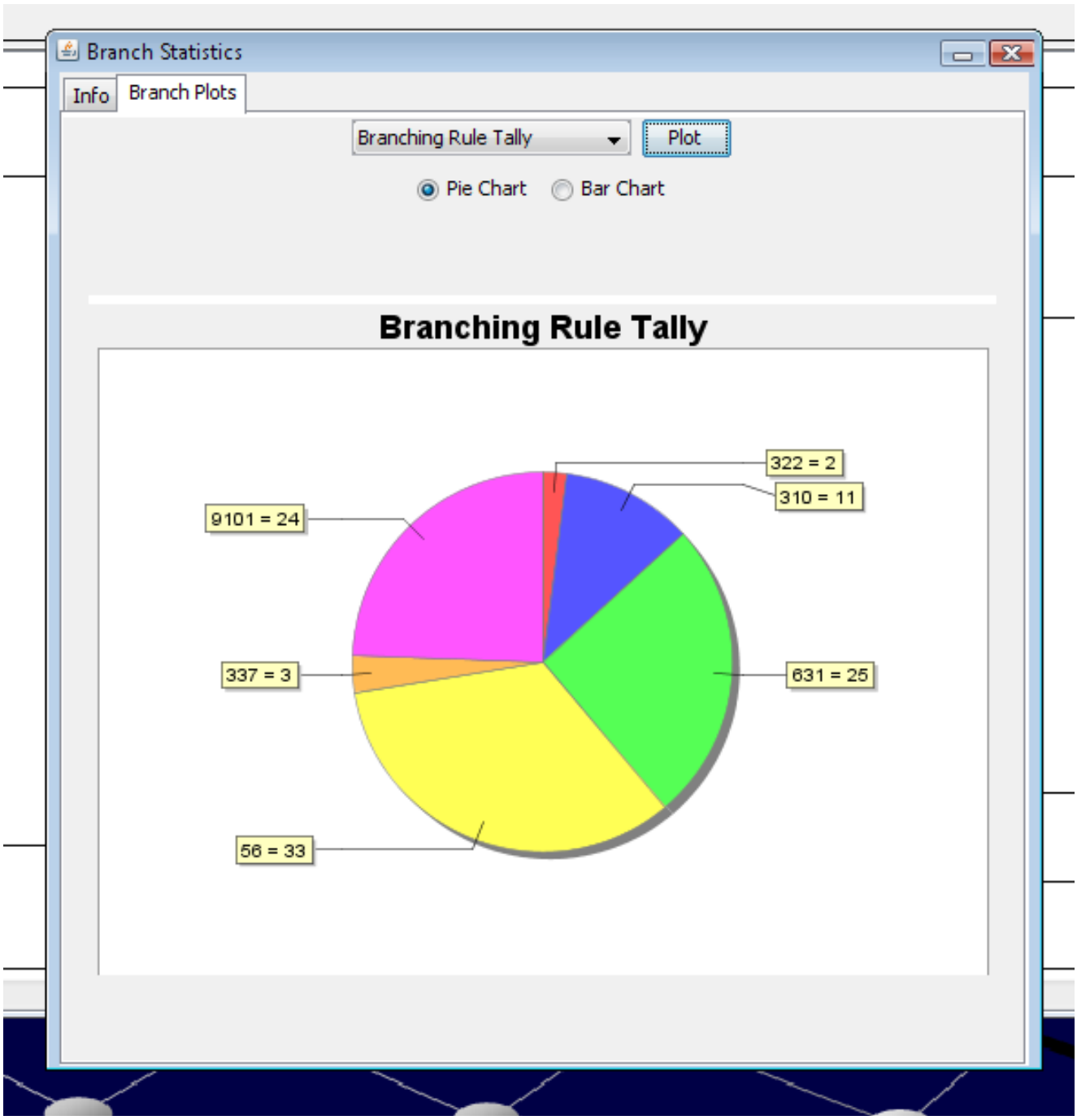
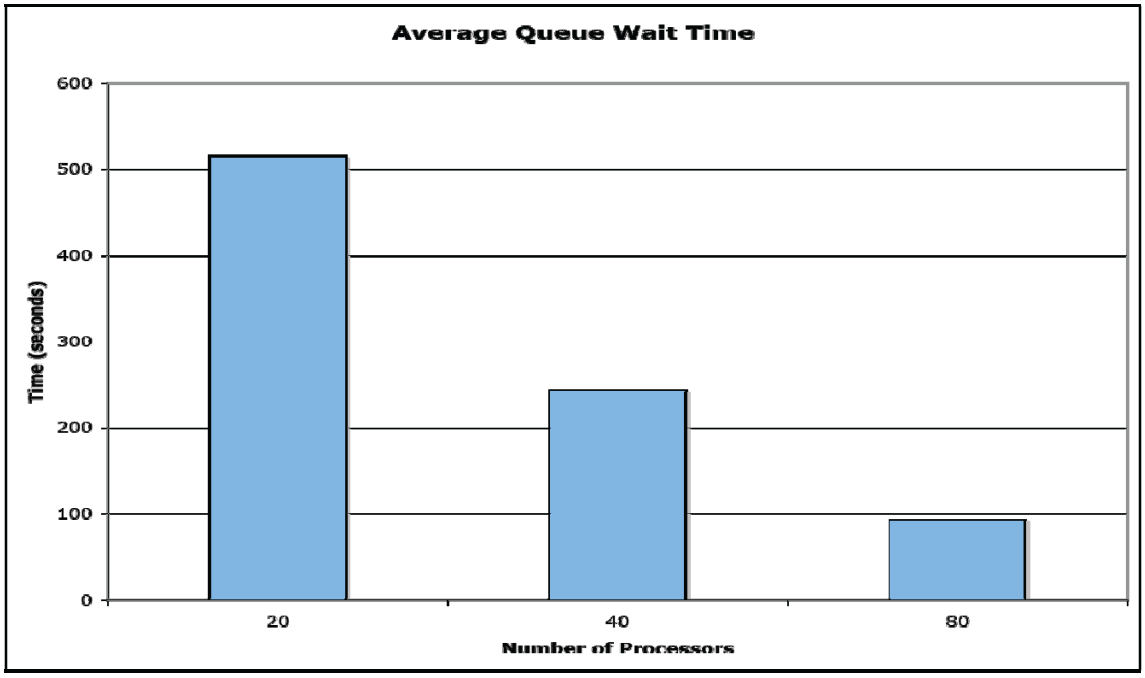
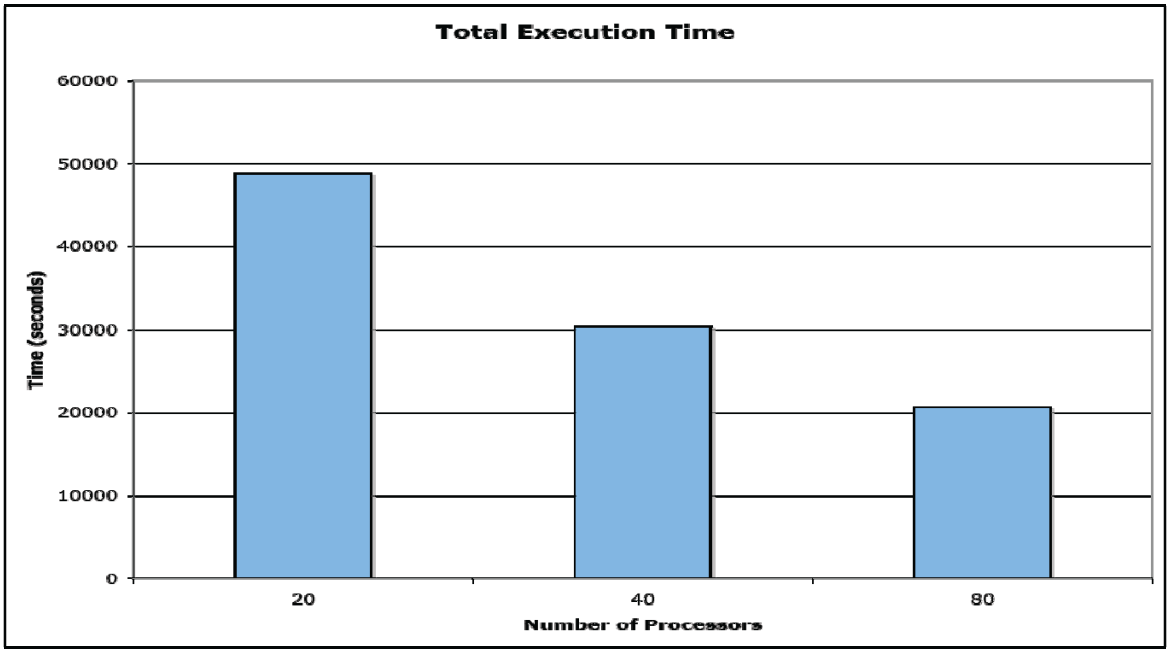


Figure 11. A plot of the distribution of branching types in Experiment 1. Note that branching types refer to the MELCOR control function which stopped the execution.

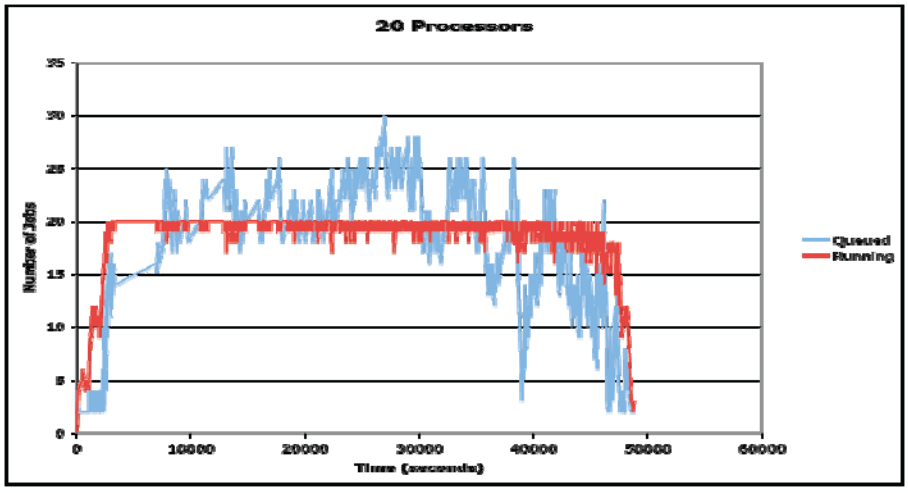


(a)

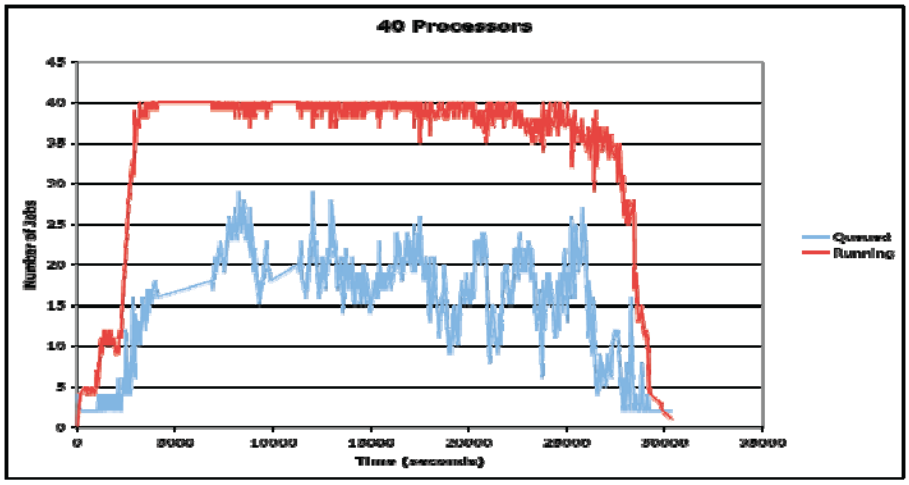


(b)

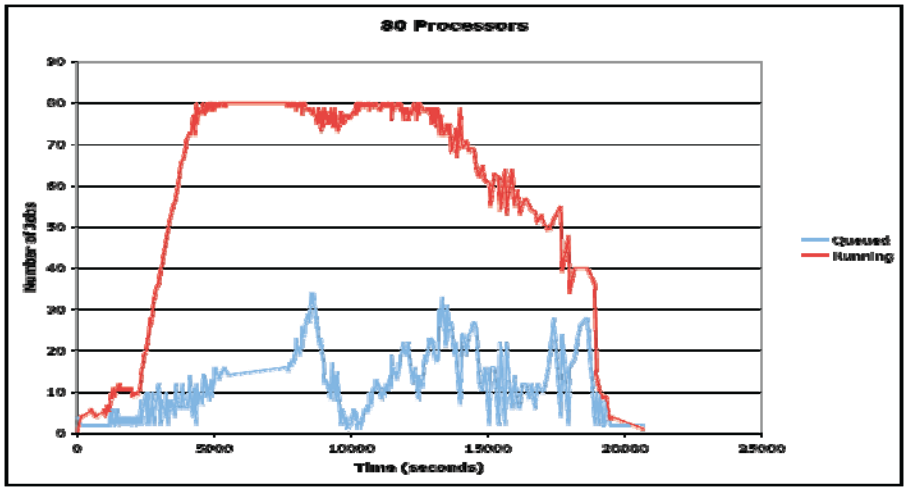
Figure 12. For Station Blackout experiment, average queue wait time and total execution time while varying the number of CPUs.


20 Processors
Number of Jobs
Time (seconds)
Queued
Running

(a)


40 Processors
Number of Jobs
Time (seconds)
Queued
Running

(b)



(c)

Figure 13. The Number of Queued and Running Jobs for Each Configuration During the Execution,

Figure 14. Breakdown of the average execution time for different configurations.

Figure 15 compares the three basic scheduling techniques: *Random*, *First-Come First-Served (FCFS)*, and *Greedy Staging Minimization (MinStaging)*, presented in Section III.C. Although random scheduling usually generated the worst scheduling, and MinStaging generated the best scheduling, unfortunately the improvement is not significant. On the average, improvement of MinStaging over Random is 1.6%, and maximum it is 2.7%. Although each same-host staging can save up to 45 seconds per branch on a single CPU, this savings can be absorbed in the case that the cluster is underutilized (as is true for much of 80 processors configuration, see Figure 13). Also, scheduling one branch on the same host may preclude scheduling another hypothetical branch on the same host immediately after, since that host may now be completely busy; depending on the duration of each of the two branches, it may have been better to schedule the second one on the same host rather than the first one. Also, different branches have different amounts of data that needs to be staged, which could vary the savings realized between different candidate branches.

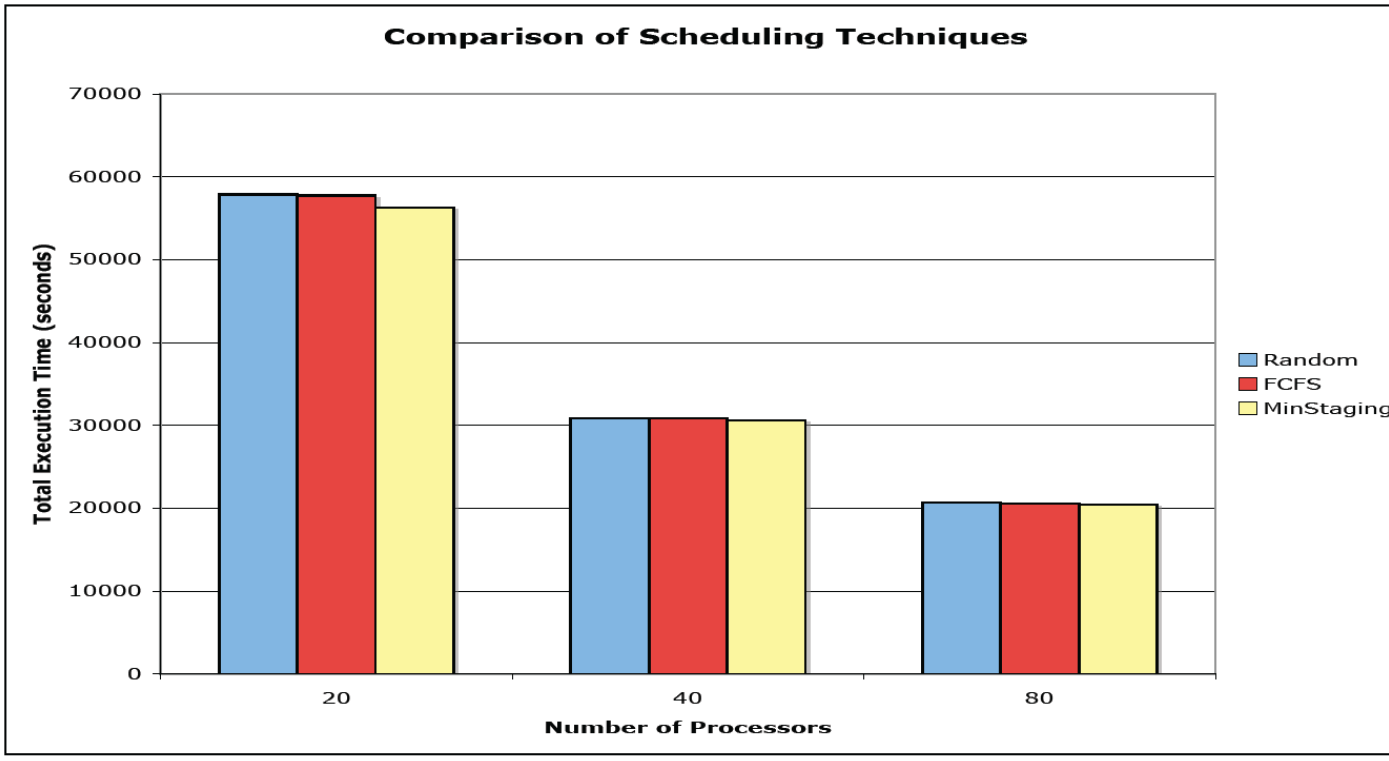


Figure 15. Comparison of basic scheduling techniques.

5. Conclusions

This project has produced a tool that can make PRAs of nuclear reactors—analyses which are very resource intensive—more efficient. The tool has wider applications as well. Initially, the work was to fuse the APET portion of a PRA to the DSD created by Ohio State University. During the course of the effort, however, it was found that the DSD could be linked directly to a detailed accident progression phenomenological simulation code—the type on which APET construction and analysis relies, albeit indirectly—and thereby directly create the APET. That is, instead of augmenting and simplifying APET construction and analysis to assist an analyst in creating and analyzing an APET, the product of this project is a tool that creates an APET and analyzes it with minimal analyst interaction. The expanded DSD computational architecture and infrastructure that was created during this effort is ADAPT.

PRAs of nuclear reactors being increasingly relied on by the U.S.N.R.C. in making licensing decisions for current and advanced reactors. Yet, PRAs are produced much as they were 20 years ago. They require significant resources to create and analyze. This work applied a modern systems analysis technique to the accident progression analysis portion of the PRA; the technique was a system-independent multi-task computer driver routine. Here, ADAPT has been presented, a system software infrastructure that supports execution and analysis of multiple dynamic event-tree simulations on distributed environments. A simulator abstraction layer was developed, and a generic driver was implemented for executing simulators on a distributed environment.

The ADAPT methodology has been described with implementation to severe accident phenomenological uncertainty treatment. As demonstrations of the use of the methodological tool in the probabilistic modeling of severe accident phenomena in Level 2 PRA, ADAPT was applied to quantify the likelihood of creep rupture of pressurizer surge line, hot leg, and SG tubes in a PWR with a large dry containment using MELCOR. A station blackout initiating event with a failure of the AFWS was considered as in this test case.

The results of this demonstration indicate that the developed approach can significantly reduce the manual and computational effort in Level 2 PRA analysis. ADAPT does not require any human intervention throughout the analysis. By implementing the model mechanistically, it also eliminates the potential of introducing errors while making changes in the input decks manually for running new accident scenarios. From the phenomenological viewpoint, it can also treat the epistemic and aleatory uncertainties associated with complex physical phenomena taking place during severe accident progression. Many potential accident scenarios that are ignored in current conventional PRA Level-2 analyses because of their static nature are accounted for in the proposed methodology, resulting in the consideration of a much wider variety of accident scenarios. The ADAPT methodology can be also potentially used for Level 1 PRA, as well as Level 2 analysis, of future plants with passively safe accident mitigation features.

While this specific project is completed, further development of ADAPT will occur, depending on future projects. Improving the usability of the system would add features to the client tools. These include

- test and improve simulator abstraction layer to work with other plant simulators, such as SCDAP/RELAP5 [74].
- design and develop more generic metadata management system to accommodate different plant simulators.
- develop a user-friendly method for setting up branching rules.
- investigate scheduling techniques especially under multiple, concurrent tree generation scenario.
- • develop a compiler that will take high-level branching rules and generate application specific edit-rules.

This additional development will occur with the applications of ADAPT. There is interest in it both nationally and internationally.

It is important to note that using and applying ADAPT to particular problems is not human independent. While the human resources for the creation and analysis of the accident progression are significantly decreased, knowledgeable analysts are still necessary for a given project to apply ADAPT successfully. ADAPT is not an "off the shelf" "plug in and walk away" tool.

This research and development effort has exceeded its original goals and can be applied to many systems analysis problems. The problem need not be nuclear reactor safety. More broadly,

- If there is a complex systems problem amenable to portrayal as an event tree, and
- If there is a computer code that simulates how the events could progress, and
- If this code has event switches or is amenable to adding them,

Then, ADAPT can be applied to analyze the problem.

6. References

1. U. S. N. R. C., "Reactor Safety Study - An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants," US Nuclear Regulatory Commission, Washington, D.C. 1975.

2. U. S. N. R. C., "Severe Accident Risks: An Assessment for Five U.S. Nuclear Power Plants," U.S. Nuclear Regulatory Commission, Washington, D.C. NUREG-1150, 1990.

3. A. Camp, D. Kunsman, et al., "Level III Probabilistic Risk Assessment for N Reactor," WHC-MR-0045, SAND89-2102, Volume 1, April 1990.

4. X. M.Chen, P. Wang, and T. Aldemir. "DSD; A Parameter/State Estimation Tool For Model-based Fault Diagnosis in Non-linear Dynamic Systems." Dynamic Reliability: Future Directions, International Workshop Series on Advanced Topics in Reliability and Risk Analysis, Center for Reliability Engineering, University of Maryland, 1999.

5. P. Wang and T. Aldemir. "Real-Time Xenon Estimation in Nuclear Power Plants." *Tras.Am.Nuc.Soc* 81 (1999): 154-156.

6. I. Munteanu and T. Aldemir. "A Methodology for Probabilistic Accident Management." *Nucl. Techno,* 144.10 (1993): 49-62.5.

7. P. Wang, X. M. Chen and T. Aldemir. "DSD: A Generic Software Package For Model-based Fault Diagnosis in Dynamic Systems." *Reliab. Engng & System Safety* 75 (2002): 31-39.

8. R. Munoz, et al. "DENDROS: A second generation scheduler for dynamic event trees." M&C '99 – Madrid. Eds. J. M. Aragones, C. Ahnert, O. Cabellos. Madrid, Spain: Senda Editorial, S.A., 1999. 1358-1367.

9. S. Sancaktar and D. R. Sharp, 1985. "Living PRA concept for plant risk, reliability, and availability tracking", Proceedings of International Conference on Nuclear Power Plant Aging, Availability Factor and Reliability Analysis, San Diego, CA

10. W. E. Vesely, 1991, "Incorporating aging effects into probabilistic risk analysis using a Taylor expansion approach", Reliability Engineering and System Safety, v. 32, n 3, pp. 315-337

11. C Acosta, and N. Siu, 1993. “Dynamic Event Trees in Accident Sequence Analysis: Application to Steam Generator Tube Rupture”, Reliability Engineering and System Safety 41, pp. 135-154

12. K-S Hsueh, and A Mosleh, 1996. “The Development and Application of the Accident Dynamic Simulator for Dynamic Probabilistic Risk Assessment of Nuclear Power Plants”, Reliability Engineering & System Safety 52**,** pp. 297-314

13. P. C. Cacciabue, et al, 1986. “Dynamic logical analytical methodology versus fault tree: The case of AFWS of a nuclear power plant”, Nuclear Technology 74, pp. 195-208

14. G. Cojazzi, R. Sardella, T. Trombetti, P. Vestrucci, 1994 “Assessing DYLAM methodology in the frame of Monte Carlo simulation”, Proc. Probabilistic Safety Assessment and Management, pp. 011/13-18, International Association for Probabilistic Safety and Managemen

15. G. Cojazzi, et al, 1996. “The DYLAM approach to the dynamic reliability analysis of systems”, Reliability Engineering & System Safety 52, pp. 279-296

16. R. Munoz, et al., 1999. A Second Generation Scheduler for Dynamic Event Trees, Mathematics and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications, pp. 1358-1367, Madrid, Spain

17. E. Hofer, M. Kloos, et al., 2004. Dynamic Event Trees for Probabilistic Safety Analysis, GRS, Garsching, Germany

18. R. Munoz, E. Minguez, E. Melendez, Izquierdo, J. M., Sanchez-Perea, M., 1999, “DENDROS: A Second Generation Scheduler for Dynamic Event Trees”, Mathematics and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications, Conference Proceedings, Madrid, Spain

19. R. M. Summers, R.K.Cole, et al., 1981. MELCOR 1.8.0: A Computer Code for Nuclear Reactor Severe Accident Source Term and Risk Assessment Analyses

20. J. M. Izquierdo, J. Hortal, J. Sanches-Perea, and E. Melendez, "Automatic Generation of Dynamic Event Trees: A Tool for Integrated Safety Assessment," in *Reliability And Safety Assessment of Dynamic Process Systems*, vol. 120, T. Aldemir, N. Siu, A. Mosleh, P. C. Cacciabue, and B. G. Goktepe, Eds., NATO ASI Series F ed. Heidelberg: Springer-Verlag, 1994, pp. 135-150.

21. M. Kloos, E. Hofer, and et al., "Dynamic Event Trees for Probabilistic Safety Analysis," GRS, Garsching, Germany 2004.

22. C. J. Burns, Liao, Y. , Vierow, K., 2005, “MELCOR Code Assessment by Simulation of TMI-2 Phases 1 and 2”, Proceedings of the 11th International Topical Meeting on Nuclear Reactor Thermo-hydraulics (NURETH-11), Avignon, France, October 2-6

23. J. Birchley, 2004, “Assessment of the MELCOR code against Phebus experiment FPT-1 performed in the frame of ISP-46”, Proceedings of the International Conference on Nuclear Engineering (ICONE-12), v.3, pp. 551-560

24. R. Gauntt, R. Cole, S. A. Hodge, S. B. Ridriguez, R. l. Sanders, R. C. Smith, D. Stuart, R. M. Summers, and M. F. Young, "MELCOR Computer Code Manuals, U.S. Nuclear Regulatory Commission, Washington, D.C. NUREG/CR-6119, Vol.1, Rev. I (SAND97-2398), 1997.

25. B. R. Buck and J. K. Hollingsworth, "An API for Runtime Code Patching," *The Journal of High Performance Computing Applications*, vol. 14, pp. 317-329, 2000.

26. W. Gu, G. Eisenhauer, K. Schwan, and J. S. Vetter, "Falcon: On-line monitoring for steering parallel programs," *Concurrency - Practice and Experience*, vol. 10, pp. 699-736, 1998.

27. S. S. Vadhiyar and J. Dongarra, "SRS: A Framework for Developing Malleable and Migratable Parallel Applications for Distributed Systems," *Parallel Processing Letters*, vol. 13, pp. 291-312, 2003.

28. J. S. Plank, Y. Chen, K. Li, M. Beck, and G. Kingsley, "Memory Exclusion: Optimizing the Performance of Checkpointing Systems," *Softw., Pract. Exper.*, vol. 29, pp. 125-142, 1999.

29. S. Hastings, S. Langella, S. Oster, and J. Saltz, "Distributed Data Management and Integration Framework: The Mobius Project," in *Proceedings of the Global Grid Forum 11 (GGF11) Semantic Grid Applications Workshop*, 2004, pp. 20-38.

30. S. Langella, S. Hastings, S. Oster, T. Kurc, U. Catalyurek, and J. Saltz, "A Distributed Data Management Middleware for Data-Driven Application Systems," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing (Cluster 2004)*, 2004, pp. 267-276.

31. O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonindentical Processors," *Journal of the ACM*, vol. 24, pp. 280-289, 1977.

32. Y. K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, vol. 31, pp. 406-471, 1999.

33. S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," 2002.

34. K. Kaya and C. Aykanat, "Iterative-Improvement-Based Heuristics for Adaptive Scheduling of Tasks Sharing Files on Heterogeneous Master-Slave Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, 2006.

35. N. Vydyanathan, G. Khanna, U. V. Catalyurek, T. M. Kurc, J. H. Saltz, and P. Sadayappan, "Scheduling of Tasks with Batch-Shared I/O on Heterogeneous Systems," in *Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS)*. 20th International Parallel and Distributed Processing Symposium (IPDPS) The 15th Heterogeneous Computing Workshop (HCW 2006), Rhodes, Greece, 2006.

36. H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," 2000.

37. K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. M. Mellor-Crummey, F. Berman, H. Casanova, A. A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, and J. Dongarra, "New Grid Scheduling and Rescheduling Methods in the GrADS Project," 2004.

38. J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task Scheduling Strategies Workflow-based Applications in Grids," 2005.

39. J. Yu and R. Buyya, "A Novel Architecture for Realizing Grid Workflow using Tuple Spaces," 2004.

40. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping Scientific Workflows onto the Grid," 2004.

41. D. Kondo, A. A. Chien, and H. Casanova, "Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids," 2004.

42. K. v. d. Raadt, Y. Yang, and H. Casanova, "Practical Divisible Load Scheduling on Grid Platforms with APST-DV," 2005.

43. B. Veeravalli, D. Ghose, and T. G. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing*, vol. 6, pp. 7-17, 2003.

44. H. Casanova, O. Graziano, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2000)*: ACM Press/IEEE Computer Society Press, 2000.

45. A. Giersch, Y. Robert, and F. Vivien, "Scheduling Tasks Sharing Files from Distributed Repositories," 2004.

46. A. Giersch, Y. Robert, and F. Vivien, "Scheduling tasks sharing files on heterogeneous master-slave platforms," 2004.

47. G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan, "A Hypergraph Partitioning Based Approach for Scheduling of Tasks with Batch-shared I/O," 2005.

48. S. Narayanan, U. Catalyurek, T. Kurc, X. Zhang, and J. Saltz, "Applying Database Support for Large Scale Data Driven Science in Distributed Environments," presented at Proceedings of the 4th International Workshop on Grid Computing (Grid 2003), Phoenix, Arizona, 2003.

49. S. Narayanan, T. M. Kurc, U. V. Catalyurek, and J. H. Saltz, "Database Support for Data-Driven Scientific Applications in the Grid," *Parallel Processing Letters*, vol. 13, pp. 245-271, 2003.

50. "MySQL Database," http://www.mysql.com/.

51. A. Hakobyan, "Severe Accident Analysis Using Dynamic Accident Progression Event Trees," vol. Ph.D. Dissertation: The Ohio State University, 2006.

52. K. Vierow, Y. Liao, J. Johnson, M. Kenton, and R. Gauntt, "Severe Accident Analysis of a PWR Station Blackout with the MELCOR, MAAP4, and SCDAP/RELAP5 Codes," *Nuclear Engineering and Design*, vol. 234, pp. 129-145, 2004.

53. A. Hakobyan, R. Denning, T. Aldemir, S. Dunagan, and D. Kunsman, "A Methodology for Generating Dynamic Accident Progression Event Trees for Level 2 PRA," presented at PHYSOR 2006.

54. A. Hakobyan, R. Denning, T. Aldemir, S. Dunagan, and D. Kunsman, "Treatment of Uncertainties in Modeling the Failure of Major RCS Components in Severe Accident Analysis," *Trans .Am. Nucl. Soc*, vol. 94, pp. 177-179, 2006.

55. Y. Liao and K. Vierow, "MELCOR Analysis of Steam Generator Tube Creep Rupture in Station Blackout Severe Accident," *Nuclear Technology*, vol. 152, pp. 302-313, 2005.

56. S. Majumdar, "Prediction of Structural Integrity of Steam Generator Tubes Under Severe Accident Conditions," *Nuclear Engineering and Design*, vol. 194, pp. 31-55, 1999.

57. F. R. Larson and J. Miller, "A Time Temperature Relationship for Rupture and Creep Stress," *Transactions of the ASME*, pp. 765-775, 1952.

58. S. A. Eide, C. D. Gentillon, T. E. Wierman, and D. M. Rasmuson, "Reevaluation of Station Blackout Risks at Nuclear Power Plants, Analysis of Loss of Offsite Power Events: 1986-2004," U.S. Nuclear Regulatory Commission NUREG/CR-6890.

59. J. O. Henrie and A. K. Postma, "Lessons learned from hydrogen generation and burning during TMI-2 event," Rockwell International, Rockwell Hanford Operations RHO-RE-EV-95P, March 1987.

60. R. K. Kumar, G. W. Koroll, M. Heitsh, and E. Studer, "Carbon Monoxide – Hydrogen Combustion Characteristics in Severe Accident Containment Conditions," Nuclear Energy Agency Report NEA/CSNI/R(2000)10, March 2000.

61. H. Le Chatelier and O. Boudouard, "On the Flammable Limits of Gas Mixtures," *Bull Soc Chim*, 1898.

62. H. F. Coward and G. W. Jones, "Limits of Flammability of Gases and Vapors," Bulletin 503, Bureau of Mines 1952.

63. J. O. Bradfute and M. P. Paulsen, "Hydrogen Flammability Model Based on Present Data," in *Proceedings of the Workshop on the Impact of Hydrogen on Water Reactor Safety, Volume III*, 1981.

64. M. P. Sherman, M. Berman, and R. F. Beyer, "Experimental Investigation of Pressure and Blockage Effects on Combustion Limits in H2-Air-Steam Mixtures," Sandia National Laboratories Sandia Report SAND91-0252, 1993.

65. W. T. Pratt, V. Mubayi, T. L. Chu, G. Martinez-Guridi, and J. Lehner, "An Approach for Estimating the Frequencies of Various Containment Failure Modes and Bypass Events," U.S. NRC Report NUREG/CR-6595, Rev.1, 2004.

66. V. Koundy, M. Durin, L. Nicolas, and A. Combescure, "Simplified modeling of a PWR reactor pressure vessel head lower head failure in the case of a severe accident," *Nuclear Engineering and Design*, pp. 835-843, 2005.

67. M. M. Pilch, H. Yan, and T. G. Theofanous, "The probability of containment failure by direct containment heating in Zion," *Nuclear Engineering and Design*, pp. 1-36, 1996.

68. J. H. Song, I. K. Park, Y. J. Chang, Y. S. Shin, J. H. Kim, B. T. Min, S. W. Hong, and H. D. Kim, "Experiments on the interactions of molten ZrO2 with water using TROI facility," *Nuclear Engineering and Design*, vol. 213, pp. 97-110, 2002.

69. D. E. Mitchell, M. L. Corradini, and W. W. Tarbell, "Intermidiate scale steam explosion phenomena: Experiments and analysis," Sandia National Laboratories Report SAND81-0124.

70. R. Klein, W. Breitung, I. Coe, L. He, H. Olivier, W. Rehm, and E. Studer, "Models and Criteria for Prediction of Deflagration-to-Detonation Transition (DDT) in Hydrogen-Air-Steam Systems under Severe Accident Conditions," Research Center Jülich, Germany 2000.

71. E. Studer and P. Galon, "Hydrogen combustion loads – PLEXUS calculations," *Nuclear Engineering and Design*, pp. 119-134, 1997.

72. W. Breitung and R. REdlinger, "Containment pressure loads from hydrogen combustion in unmitigated severe accidents," *Nuclear Technology*, vol. 111, pp. 395-419, 1995.

73. M. M. Pilch, M. D. Allen, and E. W. Klamerus, "Resolution of the Direct Containment Heating Issue for All Westinghouse Plants With Large Dry Containments or 211 Subatmospheric Containments," Nuclear Regulatory Commission NUREG/CR-6338, SAND95-2381, 1996.

74. "SCDAP/RELAP5-3D Code Manual," Idaho National Engineering and Environmental Laboratory, INEEL/EXT-02-00589, 2003.

Appendix

Presented here are the wrapper script and web interface that were created in this work for using ADAPT with MELCOR in the demonstration. The ADAPT code itself is not presented here as it is currently in the licensing process at Sandia.

A. Wrapper

The following document provides a line-by-line description of the wrapper script used to link ADAPT to the MELCOR severe accident analysis code. First, there is a picture of the actual wrapper script with line numbers on the left-hand side. Following this are descriptions of each line and their place in the wrapper script algorithm. Items in the description section highlighted in red are files or scripts specific to ADAPT and those items highlighted in green in the description section are those which are specific to MELCOR. Following each of these sections is a flow chart which provides a general schematic of how a wrapper script algorithm must proceed in order to work with ADAPT. Please note that this document is still under construction.

```
65 if [ -f adapt.cp ]; then
66   echo "we were checkpointed at `date` on `hostname`, removing sbo.stp"
67   rm -f adapt.cp
68   adapt-checkpoint-note-taken # notify system that we're taking a checkpoint
69   exit 0
70 fi
71
72
73 ## stop here if we don't care about this branch
74 if [ -f term-early ]; then
75   adapt-job-record logical_fail 1
76   exit 0
77 fi
78
79 echo "truncating at `date`"
80 set -e
81 mv sbo.mes sbo.mes.tmp
82 sed 's/^.rfmod.*newrestart.rst.*-1.*$/rfmod    """newrestart.rst"""  -1 /' < sbo.cor > sbo.cor2
83 echo E | $MELCOR_ROOT/$EXE sbo.cor2
84 mv sbo.rst sbo-nontruncated.rst
85 mv newrestart.rst sbo.rst
86 mv sbo.mes.tmp sbo.mes
87 set +e
88 echo "done truncating at `date`"
89
90 ## run acgrace
91 # set -e
92 # echo "running acgrace..."
93 # gunzip -qc $MELCOR_ROOT/$ACGRACE | tar xf -
94 # export GRACE_HOME=`pwd`/acgrace
95 # export PATH=`pwd`/acgrace/bin:$PATH
96 # mel2dmx -p MELPTF -d melcor.dmx
97 # echo "(not) running acgrace...done"
98 # rm -fr acgrace $ACGRACE
99 # set +e
100
101 ## record some attributes about the completed branch
102 mystopping_code=`grep ARAM sbo.mes 2>/dev/null | tail -1 | awk '{print $2}'`
103 sim_elapsed=`tail -5 sbo.mes | grep "^ TIME" | tail -1 | awk '{print $2}'`
104 sim_elapsed=`printf %.2f $sim_elapsed`
105 normal_term=`tail -1 sbo.mes | grep "^ Normal termination  TIME" | awk '{print $4}'`
106 if [ "$mystopping_code" = "" ]; then
107   if [ "$normal_term" != "" ]; then
108     adapt-job-record simtime `printf %.2f $normal_term`
109   else
110     adapt-job-record logical_fail 1
111   fi
112   exit 0
113 else
114   adapt-job-record code $mystopping_code simtime $sim_elapsed
115 fi
116 set -e
117
118
119 ## run the "edit rules" program to determine which child branches to submit
```

```
120 editrule-apply $EDITRULES $BRANCHESF
121 cat $BRANCHESF | while read config newstate branchnum stopcode branchhit probability terminate_early; do
122   adapt-submit                                          \
123       --terminate_early $terminate_early                \
124     --handoff    $EDITRULES $EDITRULES                      \
125     --handoff    $newstate $EDITRULES.state                 \
126     --handoffref $RST $RST                                  \
127     `test -f MELPTF && echo --handoffref MELPTF MELPTF`        \
128     `test -f sbo.ptf && echo --handoffref sbo.ptf sbo.ptf`     \
129     --handoff    $TEMPLATE $TEMPLATE                        \
130     --handoff    $config sbo.cor                    \
131     --probability $probability                      \
132     melcor                                          \
133     "sb $branchnum"                                         \
134     $THISSCRIPT
135    rm -f $config $newstate
136 done
137 rm -f $BRANCHESF
138 echo "goodbye, melcor on `hostname` on `date`"
```

1-5

6: set: set environment variables, options used:

```
Changing a "-" to a "+" will negate that option

-e Exit immediately if a simple command exits with a non-zero
  status, unless the command that fails is part of an until or
  while loop, part of an if statement, part of a && or || list,
  or if the command's return status is being inverted using !.

-x Print a trace of simple commands and their arguments
  after they are expanded and before they are executed.
```

7: Commented-out code
8: print out hostname and date
9: <blank>
10:<Comment>
11: Define melcor root directory variable. This variable is the location of melcor, sbo.rst, ,etc. and is defined as an input argument
12: Define RST variable, the restart-file
13: Define TEMPLATE variable, the templated input file
14: Define EDITRULES variable, the ADAPT editrules file
15: Define EXE variable, the melcor executable
16: Define BRANCHESF, which is the file which contains the branch-specific changes to the input file
17: Define PLOTF, the melcor-external plot file
18: <Commented> Define ACGRACE, the acgrace tarball
19-29: Actions for the root branch only
    19: Test if this is indeed the root branch and that we are NOT resuming a checkpoint

VARIABLES: NCENGINE_ROOT – Defined in ADAPT-server code, this variable is 1 if we are in the root branch and zero otherwise

NCENGINE_RESUMING_CHECKPOINT – Defined in ADAPT-server code, this variable is 1 if we are resuming a checkpoint and is 0 if we are not

20: Print out that we are initializing the root branch
21: Copy all files from the MELCOR_ROOT location to the current directory where MELCOR is going to be run
22: Check for the existence of the plot file in the MELCOR_ROOT directory, if it does exist, copy it to the current location
23: use the editrule-apply script with the –init flag to apply initial settings to the root branch (apply branching rules to the root branch)
24-27: Need to check on this one
28: Remove the branches.tmp file
29: Finish the "if" statement for the root branch

30: <Blank>
31: <Blank>
32: <Comment>
33: Remove the term-early file
34: Test to see if the NCENGINE_TERMINATE_EARLY variable is greater than 0

VARIBLES:
NCENGINE_TERMINATE_EARLY: This variable is a number of seconds that the simulator will be allowed to run before it is stopped, designed to prevent extremely long-running branches from taking up nodes when this is not desired.

35: If the NCENGINE_TERMINATE_EARLY variable is greater than zero, we will sleep for the number of seconds defined in this variable and then touch the term-early file and the sbo.stp file which will stop MELCOR. The "&" launches this set of commands on a separate thread, so that it will run alongside the execution of the simulator and create the sbo.stp file after the simulator has run for the desired number of seconds.
36: end of if statement testing for early-termination
37: <Empty>
38: <Empty>
39: <Comment>
40: Set to not exit immediately if something returns with a non-zero status
41: print that we are launching the simulator and the date
42-43: print the contents of the current directory we are running MELCOR in
44: runs MELCOR and passes the letter "E" to that execution, since MELCOR asks the question to (O) overwrite or (E) extend the restart information. We wish to extend. The "nice" command is used to launch a command with a specified priority. Without any arguments as given here it simply prints its current scheduling priority.
45: print that MELCOR is done executing
46: Define the rc variable, the MELCOR return code
47-48: Once again print out the content of the current directory

49: print out the MELCOR return code
50-52: If the MELCOR return code was anything but 0, quit
53: <Blank>
54: <Blank>
55: <Comment>
56: Test if the adapt.cp file exists
57: If we were indeed checkpointed (the adapt.cp file exists) note such and note the date and hostname. Also remove sbo.stp.
58: remove the adapt.cp file
59: run adapt-checkpoint-taken to notify the ADAPT-server that the checkpoint as been acknowledged.
60: Exit with rc=0
61: Finish if statement for checkpoint test
62: <Blank>
63: <Blank>
64: <Comment>
65: Check if the term-early file exists
66: Notify the system of the job status, in this case namely that we terminated early using the adapt-job-record script
67: Exit with rc=0;
68: End if statement testing for early termination
69: <Blank>
70: print out that we are truncating the restart file and the date
71: set environment settings to quit on a non-zero return code
72: copy sbo.mes to sbo.mes.tmp
73: sed is a stream editor which can modify in the content of text files. This line is setting the MELCOR input file such that it will create a restart file with only the most recent restart dump. This is done to save space and to reduce the size of the file that is being transferred from one node to the next.
74: run MELCOR again in order to create the new restart file
75: move sbo.rst to sbo-nontruncated.rst (the no truncated restart file)
76: recopy sbo.mes.tmp back to sbo.mes
77: change the name of newrestart.rst (the truncated restart file) to sbo.rst
78: Set environment settings such that it will not quit with a non-zero return code
79: print that we are done truncating
80: <Blank>
81-90: Commented section which was initially used to unzip acgrace and run mel2dmx. This was originally planned as a way of manipulating plot files but has not been used for some time.
91: <Blank>
92: <Comment>
93: retrieve the stopping code from the sbo.mes file by looking for the word “ARAM” in the output (this was programmed into the MELCOR stopping control functions)
94: retrieve the simulation elapsed time from the sbo.mes file
95: convert the elapsed time to a number with precision to 2 decimal places
96: retrieve the normal termination time from the sbo.mes file(full experiment time)

97: check to see if the mystopping_code variable is the empty string (there was no branching rule which stopped the code but the simulation came to its simulation end time or the simulation was aborted by some unknown means)
98: check to see if the normal_term variable is NOT the empty string
99: if normal_term is NOT the empty string then record the job status with adapt-job-record
100: else
101: if the mystopping_code is the empty string record the job as a logical fail with adapt-job-record
102: end if statement checking normal_term variable
103: exit with return code 0 if mystopping_code is the empty string
104: else – if mystopping_code is NOT the empty string
105: record the status of the job if mystopping_code is NOT the empty string with adapt-job-record
106: end if statement testing if mystopping_code is the empty string
107: set environment settings to quit if a nonzero return code is received
108: <empty>
109: <empty>
110: <Comment>
111: Run the editrule-apply script to determine which child branches to submit
112:

B. The ADAPT Web-Interface

These are the instructions for the web interface that was used in the demonstration.

Open your desired internet browser and go to http://adaptURL:adapt_port , where "adaptURL" is the host name of the computer where the web-service is being hosted and the "adapt_port" is the port on the web-host where the service is being provided.

Provide login information

ADAPT main menu:

1. List experiments
2. Launch a new experiment
3. add a new type of simulator
4. modify an existing simulator
5. delete an existing simulator

PLEASE NOTE that the first time a particular user logs on to the web server, it may take several seconds to bring up the main page. The web-server is setting up web session data for this user and this delay is normal.

List Experiments:

Click this link to list all experiments that have been run for this database user. Since there is one database user account this is the list of experiments for this installation of ADAPT. The list of experiments will give the experiment number, description, experiment name (this is actually the simulator used), the state of the experiment (running, checkpointed, finished), the total number of branches, branches completed, branches running as well as some control links. The "stats" link will give the total runtime of this experiment and compare it to the serial runtime for this experiment. This allows the user to calculate the speed-up from using ADAPT versus running each scenario serially. Also, the second to last column in the list of experiments gives the user the option to checkpoint or restart (depending on the current experiment state) the experiment. The final column allows the user to delete this experiment. This will delete all database entries corresponding to this experiment as well as all simulator data created on the cluster.

Clicking on the experiment number of a particular experiment will pull up a graphical visualization of the tree. Note that this graphic shows a time-*independent* event tree. Each branch on the tree is color coded (Green: completed, Yellow: Queued, Orange: Insignificant, Blue: Checkpointed, Cyan: Running, Red: Many successive execution failure, usually 5) according to its current state and lists on it the unique branch number and the probability as well as the simtime of completion, stopping code, and elapsed real time (for completed branches).

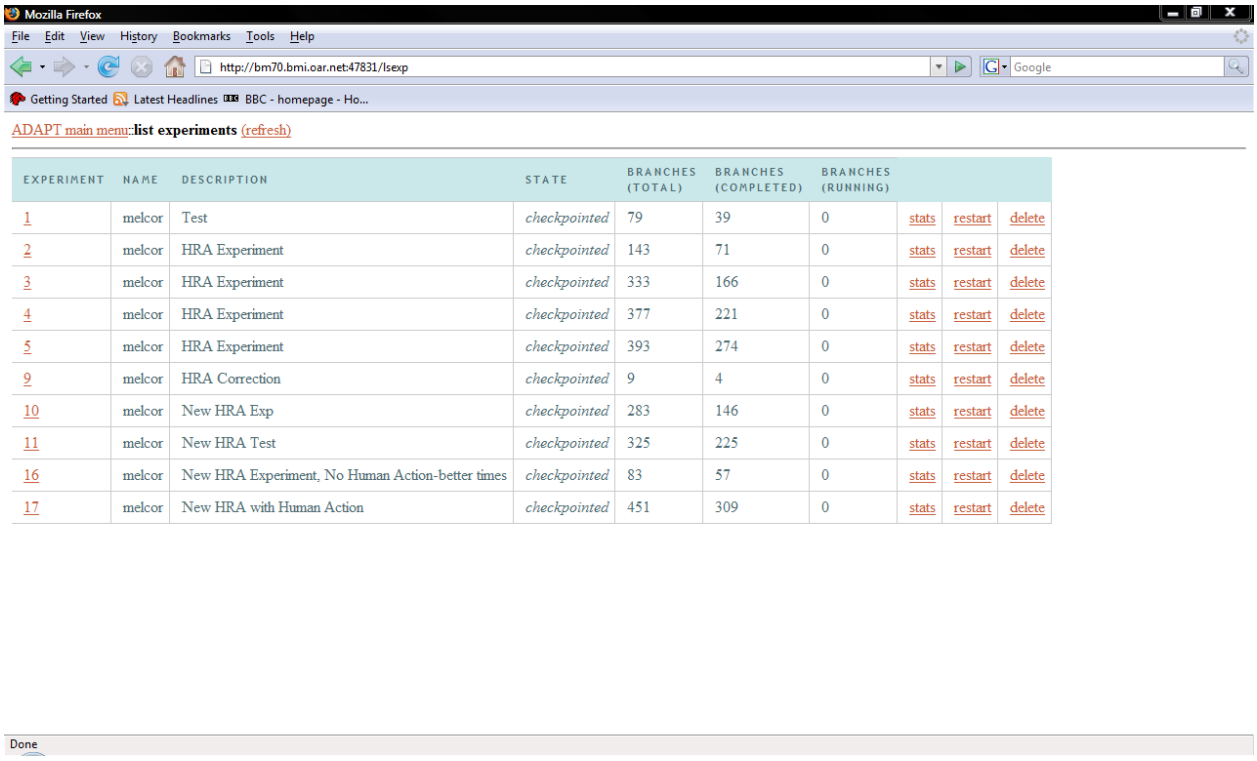


Figure 1: Snapshot of the ADAPT web-portal's list of experiments page.

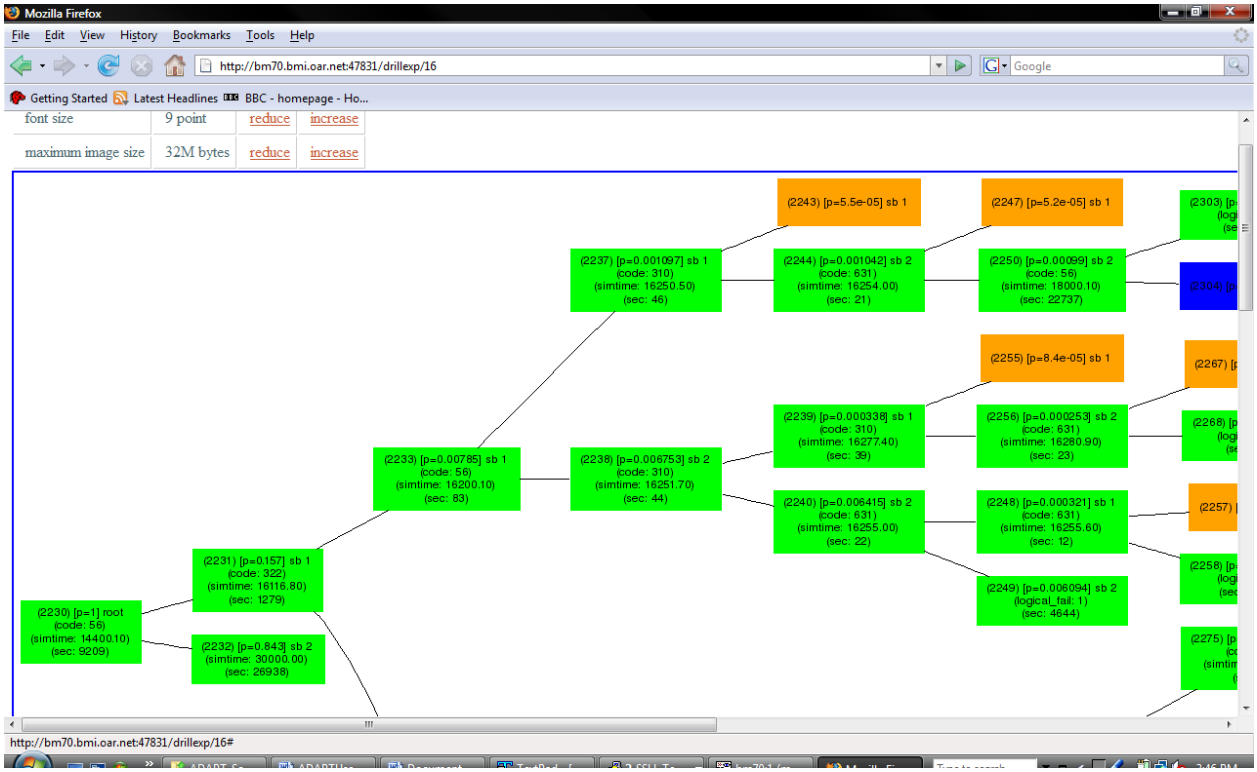


Figure 2: Snapshot of the ADAPT web-portal's graphical tree display.

Launch a new experiment:

Click this link to launch a completely new experiment. When you click on this link a new screen opens with several options. First, the option is given to select which simulator is desired. Select the desired simulator and provide a description in the description field. A default description is given in the form of "Web-Initiated Experiment" followed by the date and time of submission. Click the submit button.

Next, a new page is opened which requires the user to input the needed input files for the chosen simulator. These files are based off of the input provided in the "add a new type of simulator" section which is usually provided by the administrator. Upload the necessary files by using the browse buttons or by inputting the path on your hard drive where the files are located manually. Click the submit button to submit your experiment. Depending on the file size and network connection speed it could take several minutes to transfer the necessary files, please be patient.

Add a new type of simulator:

ADAPT is designed to accept as its input any simulator given that it meets the following requirements:

1) The code execution can be internally stopped
2) The code has "restart" capability
3) The code has a text input file

If your simulator meets these three criterion, then the next step is to register this simulator with ADAPT using the ADAPT web-interface. The purpose of this process is to minimize the number of files needed each time the user wishes to launch an experiment and give the user a consistent outline for the files required for each execution. ADAPT requires the following as input for an experiment:

1) The simulator's executable file
2) A wrapper script for the simulator
3) A checkpoint script for the simulator
4) The simulator's template text input file
5) The experiment edit rules file
6) Any other files required by the simulator
7) The web wrapper script

We will look at what each of these in this section. Some of the above will never change from experiment to experiment: the simulator executable, the wrapper script, and the checkpoint script. These can be input when a new simulator is registered and the user never needs to supply them again.

1. Simulator Executable

This is the simulator's executable file and should be rather self-explanatory.

2. Wrapper Script

No two simulators are alike and for each simulator registered with ADAPT a wrapper script must be supplied which tells ADAPT how to execute the simulator. Inside this wrapper script the input file is altered given what branching rules are applied, the simulator is executed (with the proper options and command line arguments), when the simulator quits, the output file is then parsed for reason the code was stopped (by a user-specified branching rule, abnormal termination, or the experiment ran to completion, etc.) and using several ADAPT-provided scripts, the ADAPT database is updated. After this, the stopping condition is used to determine what branching rules to apply to the children of this branch (if any) and the child branches are submitted. The wrapper script should be recursive, meaning that the script submits itself to the child branches and the child branches are executed in the same way. More detail about the structure and flow of the wrapper script is given in another section. This input needs to be given only once when the simulator is registered.

3. Checkpoint Script

Checkpointing has been built in to ADAPT as a convenience for the user. Many simulators are constructed with ability to halt its execution through some external means (in the case of MELCOR, if STOP file is created in the running directory, the simulation is halted) and then restart the simulation from that same point later on. If the simulator you are using has this feature, you can utilize the checkpoint feature built into ADAPT. This feature allows the user to pause an entire event tree execution and then restart it at a later date. The checkpoint script is responsible for containing the commands necessary for stopping the code externally (in the case of MELCOR, a stop file is created in the run directory). This input is required only once when the simulator is registered.

4. Template Input File

ADAPT takes a template of the simulator's input file for a particular experiment as input. What is meant by "template" can be illustrated by the following example. We are going to consider MELCOR input in this example. Let us say in the MELCOR input file there is a control function value we wish to use as a branching condition.

MELGEN input:

```
CF10000 'TimeStop'  equals 1 1.0 0.0
CF10010  0.0   200.0  time

CF10100 'TimeCmpr' l-gt 2 1.0 0.0
CF10110 1.0  0.0  time
```

CF10111 1.0 0.0 cfvalu.100

*this will be true when the simulation time becomes greater than the value of CF100, in
*this case, 200.0

CF899900 'StopCF' L-EQUALS 1 1.0 0.0
CF899910 1.0 0.0 cfvalu.101

*This will stop the code if CF101 becomes true

MELCOR Input:

*This is desired we wish to change the value of CF100 from execution to execution

CF10001 2000.0
CF10002 3 0.0 2000.0

*This will raise the value of CF100 to 2000.0, which will increase the amount of time the
*code will run (from 200.0 to 2000.0)

With ADAPT, the MELGEN input stays the same (the logic does not change), what we change are the values at which the logic becomes true or false. If we wish to use the above as an example and translate what it might be in ADAPT input, we would look only at the MELCOR input. For ADAPT input, it would look similar to the following:

CF10001 {V100}
CF10002 3 0.0 {V100}

Where {V100} is an ADAPT variable whose values are defined in the edit rules file. Before MELCOR is executed in each branch, ADAPT will replace these variables with values from the edit rules files depending on the branching conditions. In this way, ADAPT has dynamic control of the variables in the simulator's input and branching rules can be applied as the tree grows. This input is required each time a new experiment is submitted.

5. Edit Rules File

One of the most important files in the ADAPT input is the branching rules file. This file is simulator independent and the input structure is defined by ADAPT. This file contains a list of all of the variables that the user has defined in the template input file with their initial conditions. This file also contains a list of the branching conditions with the variables and values that need to be changed when that particular branching condition is met. Finally, this file also contains tables of branching probabilities that are to be used for each different branching rule. The structure of this file is given in another section. This file needs to be input for each new experiment that is executed.

6. Other Simulator Files

Sometimes simulators require more than just a single text input file for execution. For example, in the case of MELCOR, a restart file is required for execution. ADAPT never attempts to alter or manipulate this file,
but it is required for all executions of ADAPT with MELCOR. Depending on the setup of your simulator, these other files may be required each time an experiment is executed or they may be provided only at the time of simulator registration.

7. Web-Wrapper Script

The web-wrapper script specifies the "adapt-submit" command and specifies the local storage directory of the files which are registered to the particular simulator.

When registering a new simulator, some or all of the above file will be required and the web-interface required that they be named and the user needs to specify whether the file will be provided only at registration or provided by the user each time an experiment is launched. The naming of the files depends on their nomenclature in the wrapper-script. For the case of the example provided with ADAPT, the naming convention is as follows:

Executable:	melcor
Wrapper-Script:	melcor-wrapper
Checkpoint Script:	melcor-checkpoint
Template Input File:	sbo.cor.tpl
Editrules File:	sbo_editrules.cor
Other File:	sbo.rst (restart file)
Wrapper Script:	melcor-wrapper-web

Update an Existing Simulator:

This menu option is given to allow the user to update or modify an existing registered simulator and the registered simulator files. In the "add a simulator" option, the user is given the chance to specify files which are "provided by the installer" and the user only has to input once and not for each experiment run. Here the user can upload new "provided by the installer" files. Note that if not ALL installer-provided files are being altered; the user still has to provide a file for the entries that are not being altered. In these cases, the user must simply provide the original files for these entries.

Delete an existing simulator:

This menu option allows the user to delete a registered simulator. Simply select the desired simulator and click "Submit".

Distribution

10 Dr. Tunc Aldemir
The Ohio State University
Nuclear Engineering Program
201 West 19th Avenue
Columbus, OH 43210

1 Dr. Richard Denning
The Ohio State University
Nuclear Engineering Program
201 West 19th Avenue
Columbus, OH 43210

1 Dr. Umit Catalyurek
The Ohio State University
Department of Biomedical Informatics
333 West 10th Avenue
Columbus, OH 43210

1 Aram Hakobyan
The Ohio State University
Nuclear Engineering Program
201 West 19th Avenue
Columbus, OH 43210

1 Kyle Metzroth
The Ohio State University
Nuclear Engineering Program
201 West 19th Avenue
Columbus, OH 43210

1 Benjamin Rutt
The Ohio State University
Department of Biomedical Informatics
333 West 10th Avenue
Columbus, OH 43210

5	MS 0972	David M. Kunsman	05578
5	MS 6711	Sean Dunagan	06711
1	MS 1156	M. S. Allen	05434
1	MS 0972	S. E. Camp	05572
1	MS 0757	F. A. Duran	06414
1	MS 0757	G. D. Wyss	06414

1	MS 0719	J. J. Danneels	06760
1	MS 0748	J. D. Brewer	06761
1	MS 0748	J. Dionne	06761
1	MS 0748	J. L. LaChance	06761
1	MS 0748	R. O. Gauntt	06762
1	MS 0748	L. L. Humphries	06762
1	MS 0748	J. Jun	06762
1	MS 0748	D. Osborn	06762
1	MS 0748	V. D. Cleary	06763
1	MS 0748	G. E. Rochau	06763
1	MS 0744	M. F. Hessheimer	06764
1	MS 0744	T. A. Wheeler	06764
1	MS 0736	D. A. Powers	06770
1	MS 0736	M. C. Walck	06800
1	MS 0405	T. D. Brown	12347
1	MS 0405	L. J. Shyr	12347
1	MS 0405	R. D. Waters	12347
1	MS 0899	Technical Library	9536 (Electronic)
1	MS 0123	D. Chavez, LDRD Office	1011