



The Tutor in Tutorial (tu•to'ri•al)

INSIDE

A Case Study: Two Components In Parallel With Epistemic Uncertainty 2

Time-Series Calculations Via SAPHIRE 7

SAPHIRE Tip - Storing Cut Sets 11

The SAPHIRE TV&V 12

SAPHIRE Tutorials

tu•tor 1. A private instructor. 2. A teacher or teaching assistant.

Vital to the utilization of any complex technology is proper training on that technology. Consequently, one of the important focuses of the SAPHIRE Users Group has been the dissemination of information on both the use of SAPHIRE and the understanding of the probabilistic risk assessment that is behind the software. That tradition is carried on in this issue of the *Facets*, specifically

- ◇ The article by Emanuele and myself illustrates one of the more complex, yet fundamental, concepts in reliability assessment. Namely, the topics of “average,” “structure function,” and why SAPHIRE is not always correct are addressed.
- ◇ The article by Jim covers one of the newest calculational features in SAPHIRE, the “time-series” basic event. He illustrates use of this “plug-in” for a probability calculation that, up to now, was difficult to perform using traditional cut-set based risk analysis programs.
- ◇ The latest information on the SAPHIRE testing, verification, and validation exercise is provided.
- ◇ A short article is presented to

address how to use the “end state” save option for storing minimal cut sets. This technique will allow you to store multiple sets of minimal cut sets safely in the SAPHIRE database.

In addition to the tutorial-type of information you will find in the newsletter, the SAPHIRE web site (saphire.inel.gov) provides a growing list of beginner to advanced-level tutorials. These on-the-web tutorials provide a step-by-step approach to performing specific actions within SAPHIRE. So, if you want to see how to start a new project, build a fault tree, or save cut sets to an end state, go to the Users area of the web site.



YEAR 1900+100

And lastly, since this is the final newsletter that will have a “19xx” year on the cover and the subject matter deals with software (risk analysis software at that), I thought I would say a few brief words on Y2K.

First, SAPHIRE is Y2K compliant for two reasons; it stores dates as four-digit years and it does not use any dates in its calculations. Second, it appears that the likelihood of a Y2K catastrophe is small, so enjoy December 31st and January 1st. Third, have a productive and happy new year!

SAPHIRE FACETS
Idaho National Environmental and Engineering Laboratory

Editor: Curtis L. Smith
P.O. Box 1625
Idaho Falls, ID 83415-3850
(208) 526-9804
FAX (208) 526-2930
e-mail: CLS2@inel.gov

SAPHIRE FACETS is published twice a year. Submissions are welcome and can be sent to the editor.

Copyright © 1999 by Bechtel, B&W Idaho, Inc.

A Case Study: Two Components In Parallel With Epistemic Uncertainty

Emanuele Borgonovo and Curtis Smith

Introduction

In this article we will illustrate the general steps in the assessment of the reliability and availability of a system. The first part of the article will focus on the theoretical aspects, illustrating the fundamental concepts of a "structure function," the rare event approximation, aleatory uncertainty, and epistemic uncertainty (for now, think of these as two types of uncertainty, model and parameter). The second part will apply those concepts to the case of two components in parallel, illustrating how SAPHIRE can approximate an availability evaluation.

General Definitions

A system is designed and built to perform a specific function. When it is not able to perform its intended function, we generally speak of failure of the system. Let us indicate with "X" the variable which represents the *state* of the system and assign to X the value of 1, when the system is down, and 0 when the system is working. With this assumption, the behavior of the system is governed by Boolean logic.

It is well known that, in real life, it is not always easy to draw a sharp line between total failure, and partial failure modes. Therefore, at least in theory, we should assign to X a third value (1/2, for example) to indicate that it is not perfectly working, but it is not totally failed. This is *not* normally done in traditional probabilistic risk or reliability analysis, and so the state variable (also called indicator variable) X can assume only two values and is defined as a Boolean variable.

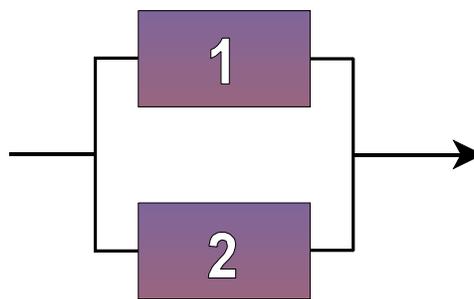
Again, these two values for our example are either 0 (working) or 1 (failed).

The generic system X will normally consist of a set of components. At a certain time, any one of these generic components will be failed or working. Depending on the configuration of the components, the system in turn will be up or down. This dependence of the state of the system X to the state of the various components is expressed by the following notation:

$$X_T = X(X_1, X_2, \dots, X_N), \quad (1)$$

where N is the total number of components. Equation (1) is what is called "**structure function**" of the system. It embodies the logical structure of the system.

Let us for simplicity refer to the case of a system made up of two component in parallel. The system will be working when *either* one of the two components is working. Consequently, both components must be failed to fail the system.



Since each component can be up or down, the structure function is in this case given by:

$$X_T = X_1 X_2 \quad (2)$$

The structure function of the system is related to its cut sets and the minimal cut sets. Let us consider a system constituted by N components. The system will certainly be down when all of the components will be failed. But it

may fail before *all* the components are failed (depending on the system design). Now, a sufficient condition for the system to be failed is that a certain set of components are failed. This condition brings us to the definition of a cut set as a set of indicator variables (i.e., indicating system is inoperable)

$$M_i = \{ X_1, X_5, \dots, X_h \}$$

for which, if X_1, X_5, \dots, X_h are all equal to 1 (failed) then this implies that X_T is equal to 1. That is, if the set of components 1, 5, ..., and h is failed, the system is also failed. Thus, by definition, $M_i = 1$ if and only if each of the indicator variables of the set is equal to 1. As you can guess, we are leading up to the concept of minimal cut sets.

Suppose now, that if component 5 is working, the system is still failed. This would mean that the set of component we have considered is a cut set, but the cut set is not *minimal*. A **minimal cut set** is a cut set which does not admit another cut set as a subset. That is, if we are considering a minimal cut set, all the components in it must be failed in order to fail the system.

In general there will be more than one way to collect the components of the system and produce its minimal cut sets. If this is done (and let us suppose we have found k minimal cut sets), it can be proven that the structure function expressing the failure of the system can be written as:

$$X_T = 1 - \prod_k (1 - M_i) \quad (4)$$

If we look at Equation (4) in some detail, we can see that it is analogous to the structure function of the series of k components: the system is down, when a component is down. In our case, when a minimal cut set is verified ($M_i = 1$ by definition), the system is down.

From Structure Function To Probabilities

Let us now refer to a generic system. Suppose we have studied its failure modes and we have identified its minimal cut sets. We are now ready to answer the question "what is the failure probability of the system?" In asking such question, we are making a modeling assumption. That assumption is we are considering the failure of a system as a "random" event. From this way of thinking comes the link between the structure function and the subsequent failure probability for the system.

To illustrate such a link, we will refer to the case of two components in series.

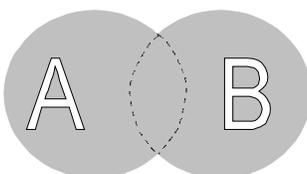


The system will be failed when:

- A fails or B fails
- A and B fail together

In terms of a Venn diagram and random events, we represent the system as follows: the system is failed when the random events A, B, or AB have occurred.

In view of Boolean logic, the events



are related by an **OR** gate . From the logic, the minimal cut sets of the system are (in SAPHIRE notation):

- A +
- B .

The structure function of the system is:

$$X_T = 1 - \prod_k (1 - M_i) = 1 - (1 - X_A)(1 - X_B) \tag{5}$$

Now we are ready to quantify the system probability. But, we must first expand the structure function before numerically evaluating the component probabilities. That is, we expand the structure function to:

$$X_T = 1 - \prod_k (1 - M_i) = 1 - (1 - X_A)(1 - X_B) = X_A + X_B - X_A X_B \tag{6}$$

Now we can (and must) write the corresponding probability of system failure (i.e., event T) as:

$$P(T) = P(A) + P(B) - P(A \cdot B) \tag{7}$$

To calculate P(T) we need the probabilities: P(A), P(B), and P(AB). Probability theory tells us that:

$$P(AB) = P(B|A) P(A) \tag{8}$$

where P(B|A) is the conditional probability of B happening given that A has happened. If we do not consider "common cause" failures, but assume only independent-type failures, we are allowed to write:

$$P(AB) = P(A) P(B) \tag{9}$$

Then, we can write:

$$P(T) = P(A) + P(B) - P(A) P(B) \tag{10}$$

The Rare-Event Approximation

In the case of a generic system with

N components, the structure function will be more complex. For example in case of a two-out-of-three logic we get:

$$X_T = 1 - \prod_k (1 - M_i) = 1 - \{(1 - X_A X_B)(1 - X_B X_C)(1 - X_C X_A)\} = X_A X_B + X_B X_C + X_C X_A - 2 X_A X_B X_C \tag{11}$$

After reducing this equation via standard Boolean algebra, we can proceed to the probabilities to get:

$$P(T) = P(AB) + P(BC) + P(AC) - 2P(ABC) \tag{12}$$

In many situations of modern reliability and risk assessment, we are dealing with events that have a very low probability of occurrence. In other words, they are "rare events." In this case of a "rare event," we know that the term P(ABC) is much smaller than the term given by P(AB) + P(BC) + P(AC). Therefore, it is a good approximation not to consider this probability [P(ABC)] in the calculation.

In general then, the rare-event approximation for a set of minimal cut sets tells us that we may simply sum each cut set to obtain the overall (approximate) probability.

Unreliability And Unavailability – Two Components In Parallel

Let us refer to a system of two components A and B in parallel (for example, see the previous page). The corresponding cut set is:

$$T = A \cdot B \tag{13}$$

The structure function can be written, in this case, as:

$$X_T = X_A \cdot X_B \tag{14}$$

Let us now consider the following question: "What is the reliability of the system?" To answer this question, we must go to event probabilities and write:

$$P(A B) = P(B | A) \cdot P(B) \tag{15}$$

Let us neglect, for the moment, dependent failures. Thus, we can use:

$$P(A B) = P(B) P(A) \tag{16}$$

Now, assume that in order to obtain P(B) or P(A), we need to consider an **aleatory** model. Aleatory models represent a random process; an example of this process would be the time until a light bulb burns out. For this example, we will assume that P(B) and P(A) can be represented by the aleatory model:

$$P = 1 - e^{-\lambda t} \tag{17}$$

where λ is the event failure rate and t is the duration of operation (i.e., the mission time). If the product " λt " is small (less than 0.1), then Equation 17 is approximately:

$$P \approx \lambda t \tag{18}$$

We can rewrite Equation 16 as:

$$P(A B) = \lambda_A \lambda_B t^2$$

Suppose now that we are interested in the average unavailability of this simple system. To find the average, we can return to the standard definition of an average from calculus class. This definition is given by:

$$X_{ave} = \frac{1}{T} \int_0^T X(t) dt = \bar{X} \tag{19}$$

For our system, the time of interest (i.e., time over which we measure the unavailability) is the **mission time**. If T is the mission time of the

system we obtain:

$$\begin{aligned} \overline{P(AB)} &= \frac{1}{T} \int_0^T dt \lambda_A \lambda_B t^2 \\ &= \frac{1}{3} \lambda_A \lambda_B T^2 \end{aligned} \tag{20}$$

Note that Equation 20 provides a different result than what we see out of SAPHIRE. The SAPHIRE software deals with minimal cut sets and does not treat system results in an exact, calculus-based framework. To illustrate this point, we need to look at how the individual components (i.e., A and B) are brought together in SAPHIRE.

Looking at an individual component (say for A), its average unavailability is:

$$\begin{aligned} \overline{P(A)} &= \frac{1}{T} \int_0^T dt \lambda_A t \\ &= \frac{1}{2} \lambda_A T \end{aligned} \tag{21}$$

a familiar result. Now, since SAPHIRE deals with cut sets, it would produce the system average result as:

$$\begin{aligned} \overline{P(A)} \overline{P(B)} &= (\frac{1}{2} \lambda_A T)(\frac{1}{2} \lambda_B T) \\ &= \frac{1}{4} \lambda_A \lambda_B T^2 \end{aligned} \tag{22}$$

Comparing Equation 22 to Equation 20, we notice that SAPHIRE slightly *underestimates* the correct average unavailability (since it uses a factor of $\frac{1}{4}$ rather than the correct $\frac{1}{3}$).

While we have shown that SAPHIRE underestimates the system unavailability for our simple system, we also know that the SAPHIRE calculation is a point estimate. Notice that we have not introduced the notion of uncertainty

sampling (in order to obtain a *average* value) yet in the discussion. It is possible that propagating uncertainty through our aleatory



model [e.g., Equation (22)] would bring us closer to the exact average system unavailability results. But, to investigate this avenue, we need to introduce the notion of **epistemic** (a.k.a. parameter) uncertainty.

Epistemic Uncertainty

When dealing with random events, we do not know *when* the event will happen (aleatory uncertainty), and therefore we use a distribution for the failure time of the component (the exponential, for instance, if aging is not considered).

In real problems, we are never given the exact failure rate of components, but we know their failure rates only up to a certain degree, based on the data available. This state of knowledge uncertainty is referred to as epistemic uncertainty.

Coming back to our example, this uncertainty will force us to utilize two distributions that display our state of knowledge about the real values of the component failure rates (λ_A, λ_B). These distribution are called epistemic distributions.

The effect of introducing the epistemic uncertainty is to render λ_A and λ_B as two random variables. Therefore, $P(A B)$ will become a function of random variables. The epistemic uncertainty on the individual parameters will propagate through the overall expression (20) in the case of an exact calculation or expression (22) in the case of the SAPHIRE approximation.

To summarize these analysis steps for determination of overall system average unavailability, one must:

1. Determine the structure function for the system. Alternatively, one could use SAPHIRE to generate minimal cut sets.
2. Expand the structure function to its complete form. Alternatively, one could approximate the structure function expansion by utilizing the SAPHIRE minimal cut set upper-bound quantification.
3. Pass the applicable probabilities into the structure function equation to obtain a point estimate to the unavailability. It may be possible to utilize the rare-event approximation to speed-up this step.
4. Obtain the average unavailability by integrating the structure function over the mission time.

At this point, we have not included the epistemic uncertainty in the average unavailability calculation for either the exact (i.e., calculus) equation or the SAPHIRE approximation. The fact that the parameters of the calculation have uncertainty implies that the calculation result will have uncertainty associated with it. Thus, we will have a distribution for the average unavailability calculation. While at first glance this might be confusing since we could obtain the average of our "average unavailability." We do not frequently speak of "averages of averages," but they do certainly exist.

To think of this "average" situation in another way, imagine that you took 12 shoelaces and measured

the length of each. From these measurements, you could determine an average length. But, the value of the average length is uncertain since we have a finite sample size (e.g., 12). In other words, measuring another shoelace (the 13th) and recalculating the average length would result in a new value. Consequently, it may be useful to estimate a lower bound and upper bound on the average length. Alternatively, an average value for the average length could be estimated.

$$\overline{P(A)} \overline{P(B)} = (\frac{1}{2} \lambda_A T)(\frac{1}{2} \lambda_B T)$$

$$= \frac{1}{4} \lambda_A \lambda_B T^2 \quad (23)$$

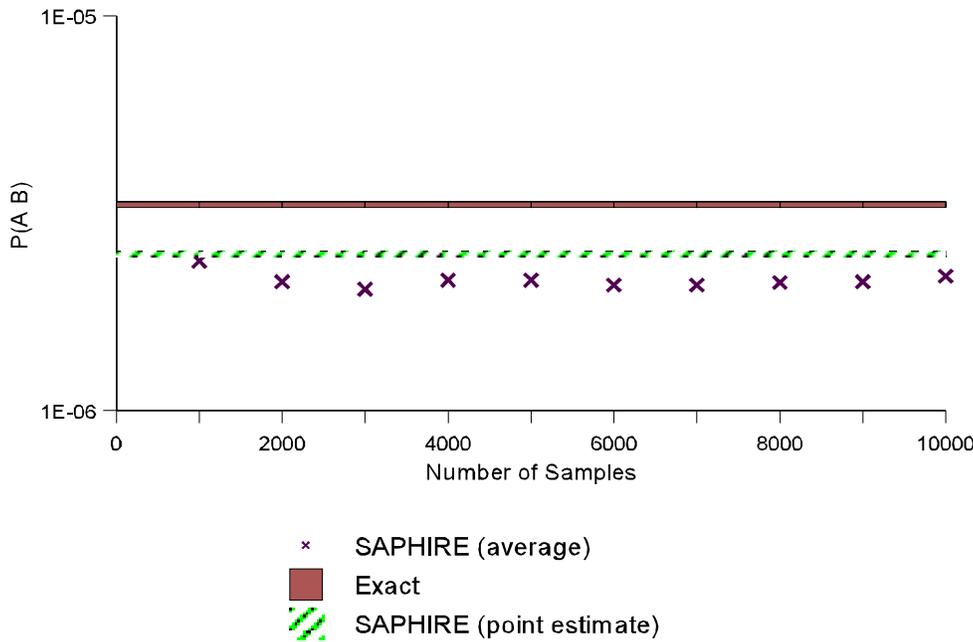
We can reproduce this equation in SAPHIRE by **ANDing** event A with event B. Then, for each event, we will assign a "calculation type" of 7 to the event. This calculation type represents an average unavailability over a mission time of T. The mean failure rate and

In the next issue of the *SAPHIRE Facets* we will demonstrate the calculation for evaluating the exact unavailability when considering uncertainty on the inputs (i.e., epistemic uncertainty). For now, we will demonstrate an epistemic uncertainty calculation using SAPHIRE.

Epistemic Uncertainty In SAPHIRE

We return to our parallel system. As we have discussed, SAPHIRE would produce the system average result as:

uncertainty for each event is shown in Table 1. This information is also entered into the SAPHIRE database.



As can be seen in Figure 1, the results of the epistemic uncertainty analysis in SAPHIRE for the simple parallel system appears to converge around approximately 4,000 samples. Not shown on this figure are percentiles like the 5th or 95th. In the next issue of the *Facets*, when the uncertainty on the exact calculation is explored, the percentiles from the SAPHIRE calculation will be useful to compare the SAPHIRE results with the exact results. For now, we will close by reiterating a few points from the discussion.

Figure 1. Convergence results of SAPHIRE uncertainty sampling.

Table 1. The epistemic distributions.

Failure rate	mean	EF
λ_A	10 ⁻⁵	10
λ_B	10 ⁻⁶	10

Note: The rates are in per hour units. The "EF" represents the error factor for a lognormal distribution.

We set the mission time **T** to be a value of 1000 hours (note that for the calculation type 7, the mission time is entered into the "tau" field of the basic event data screen). At this point, we now have enough information to perform the calculation. From SAPHIRE, the cut sets of the system are:

A B.

The point estimate unavailability for these cut sets from SAPHIRE is compared to the exact calculation and is shown below.

Calculation	Value
SAPHIRE	2.5 × 10 ⁻⁶
Exact	3.3 × 10 ⁻⁶

Note that the correct calculation is slightly higher than the SAPHIRE results. Now, we can propagate the uncertainty on our failure rates through the calculation by telling SAPHIRE to perform an uncertainty analysis on the cut sets shown above. To perform the uncertainty calculation, select the fault tree from the fault tree screen, click the right mouse button, and select the **Uncertainty** option. Now, we must choose the number of samples; the seed; and either Monte Carlo or Latin Hypercube sampling.

The choices you make for the seed value and sampling type are generally not important. What is important is that your uncertainty analysis results are converging to an answer. Thus, it is prudent to start the uncertainty analysis with a low number of samples (e.g., 1,000) and then increase the number of samples while checking for convergence in your answer. This process is illustrated in Figure 1 where we plot the average (i.e., mean) value versus the number of samples.

- Underlying a logic model is an associated structure function.
- To determine an exact failure probability, one could utilize the structure function.
- SAPHIRE analysis results for some cases (e.g., the average unavailability for redundant components) is an approximation to an exact calculation.
- Both an exact calculation and an approximation may have uncertainty associated with the analysis if its respective inputs are uncertain.
- Aleatory uncertainty represents randomness in the outcome of events while epistemic uncertainty represents potential variation in the model input parameters.

"The
 true
 logic
 of
 this
 world
 is
 in
 the
 calculus
 of
 probabilities"

James Clerk Maxwell (1831-1879)

Time-Series Calculations Via SAPHIRE

James Wilson

As nuclear reactor probabilistic risk assessment (PRA) matures or PRA is applied to more diverse fields,

$$P_{NRAC}(t_{long} | t_{short}) = P[(L > t_{long} | L > t_{short}) \cap (G > t_{long} | G > t_{short})]$$

problems are encountered that cannot be easily solved with traditional codes. Two illustrations of this problem are:

- Calculating the probability of recovering power (at a power plant). During the time power is lost at a power plant, several competing events are occurring such as depletion of the plant batteries, restoration of the emergency diesel generators, and attempts to recover off-site power. All of these events have a time element associated with them that must be accounted for in the probability determination.
- Some systems are best modeled as a sequential wear-out failure with units of time. For example, if a storage container is buried in the ground, we do not need to worry about water corrosion until water is present around the container. Consequently, the probability that the container fails over a period of time is influenced by both the time to water intrusion and the probability of failure given water intrusion. These two elements may both be a function of time.

We will look at these two problems in greater detail.

Loss Of Power

At a nuclear power plant, the loss of off-site power (LOOP) nonrecovery probability (that is, the probability that off-site power *is not* recovered in a time t) may be modeled as

where

- NRAC = non-recovery of AC power
- t_{long} = plant battery depletion time (akin to a mission time)
- t_{short} = time interval to uncover the reactor core if no safety system were to function.
- L = duration of LOOP
- G = duration of diesel generator unavailability.

One traditional approach to this problem is an offline calculation using "convoluted" or nested integrals where the integration is over the time intervals that are important to the loss of power. With the recent upgrade to SAPHIRE, this processing may be done online using standard input parameters, by modeling the event P_{NRAC} as a "compound" event. We will demonstrate use of compound events and the new time-series calculation with an example in the next section.

Sequential Wear-out Failures

Several of the scenarios in a nuclear waste repository event tree (e.g., for Yucca Mountain) involve all the components failing within a given time-span. These time spans were very long compared to traditional PRA calculations. For example, one of the principle initiators was a new

ice age occurring over the next 100,000 years, shifting the jet stream southward over the repository, thereby increasing precipitation in the desert area. This increased precipitation could corrode through a waste package, potentially allowing fissile material to assemble into a critical configuration.

This string of events presented problems for traditional SAPHIRE calculations because:

1. The events were expressed in time to failure rather than probability of failure.
2. The events may have a period with no failures, then a period when all components failed. For example, it is possible that all wetted packages failed between 1,000 and 3,000

years. Prior to the 1,000 years, no packages failed (with probability of one).

3. Because of the mission times involved (e.g., 10,000 to 1 million years), prediction of component and geologic behavior relied heavily upon Delphi predictions (e.g., expert elicitation) involving probabilities and time. For example, output of the Delphi process may yield a probability of 0.5 that the component failed in 5,000 to 10,000 years and a 0.5 probability that the component did not fail at all.
4. These events were sequential in time. That is, the "clock" did not start ticking for Event B until Event A happened. For our case,

we can not have a critical configuration until precipitation increases in the desert area. And, we could not have precipitation until an ice age occurred (for this scenario).

This string of time dependent events initially required modeling all time-related events as a SAPHIRE logic model with all probability events represented separately on the fault tree. We then modeled this same scenario using the time-series compound event in SAPHIRE. Figure 2 shows the error introduced when we attempt to change the

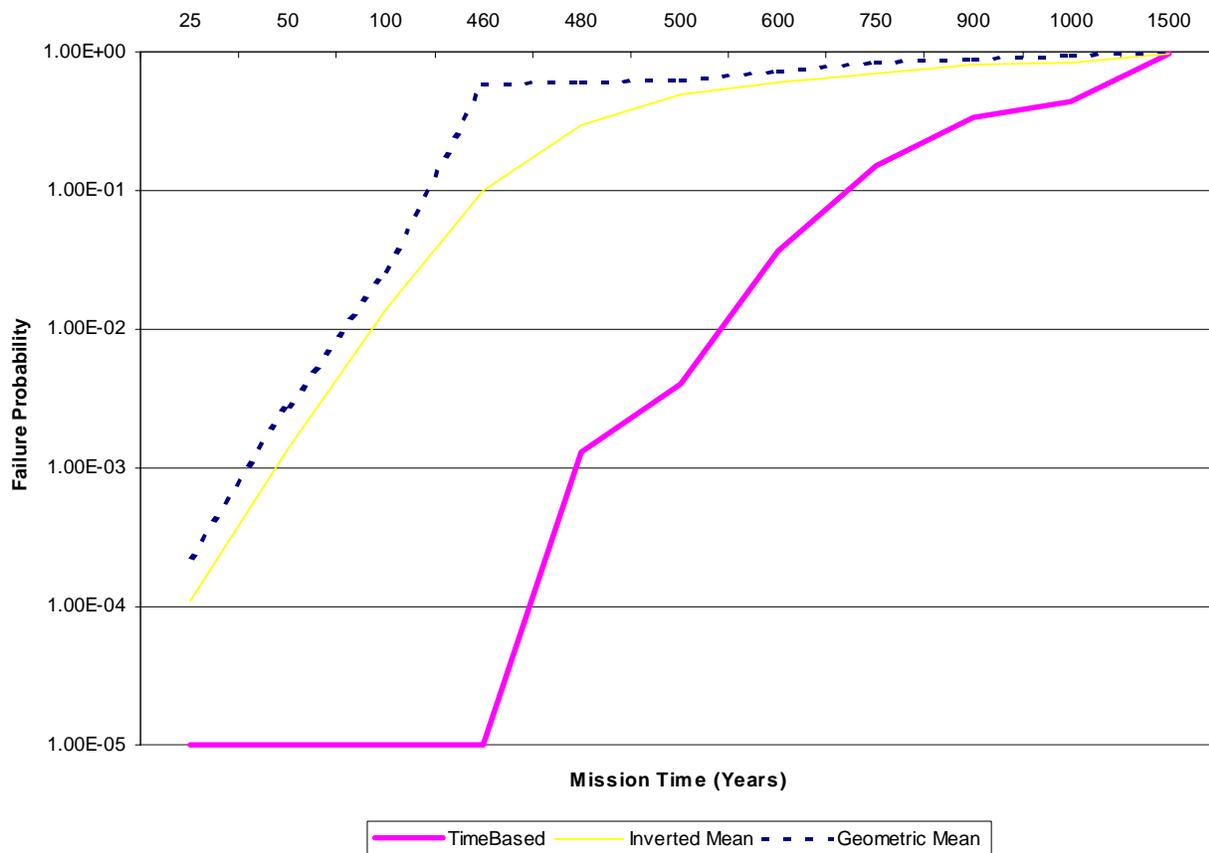


Figure 2. Illustration of calculational error between time-based and probability-based event analysis.

time-based data into probabilities (so that we can put them into the fault tree). The curve labeled "Time Based" represents the new SAPHIRE compound event for time-series calculations. The two curves labeled "Inverted Mean" and "Geometric Mean" represent the traditional fault tree modeling. Note that this discrepancy is not unique to SAPHIRE; any fault tree program that utilized tradition techniques would have the same calculation limitation.

Comparison Of Probability Versus Time-Based Methods

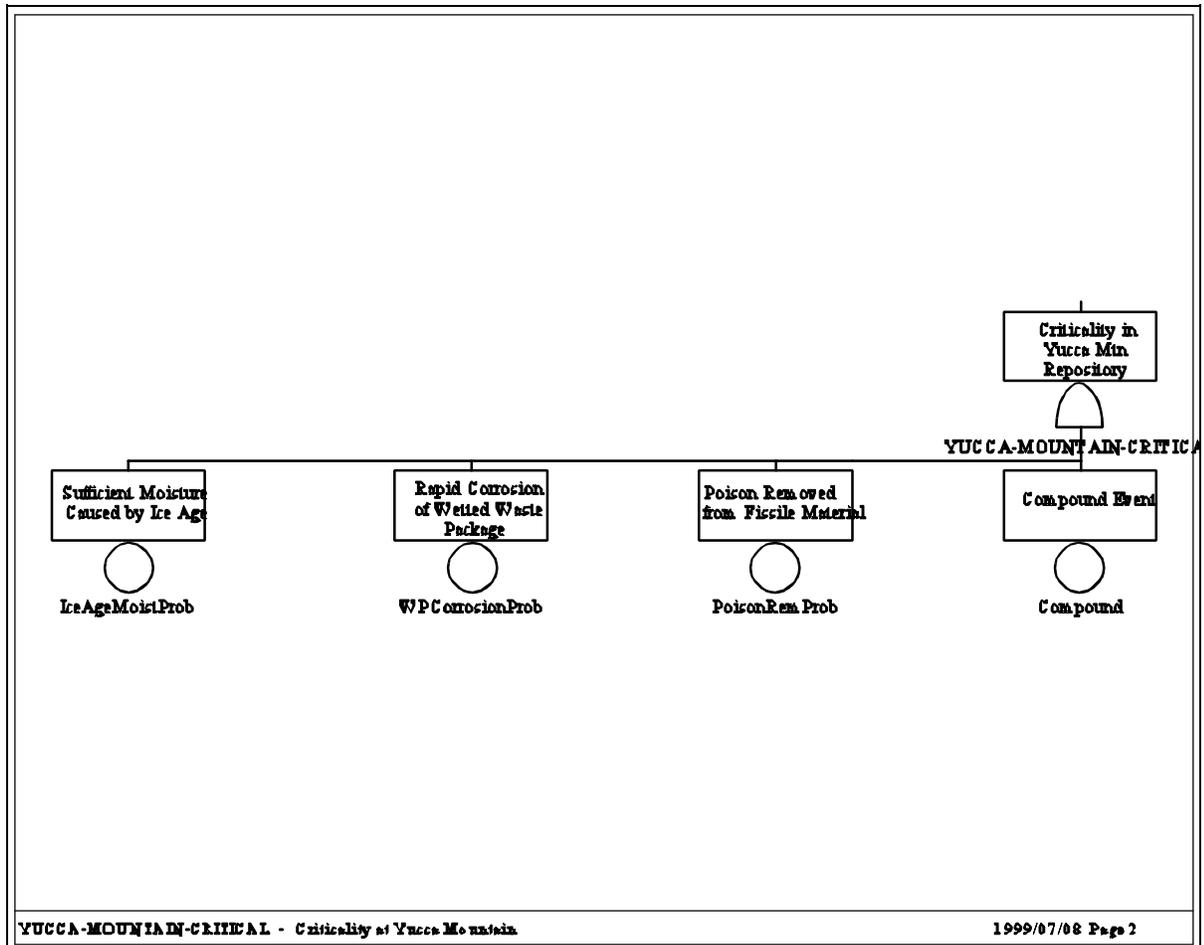


Figure 3. Fault tree with time-based events mixed with probability events. Note that the time-based events are evaluated using the SAPHIRE "time-series" plug-in calculation.

First, we will present a hypothetical scenario events for our criticality- accident model. Note that the events and data in this article do not represent any real-world repository calculations.

New Ice Age Arrives

An ice age is expected sometime between 1 to 900 years into the future, with a mean of 450 years (again, for illustration purposes only). Since we need a probability in the SAPHIRE fault tree technique, we could just invert the mean time to come up with an frequency, or $1/450 \text{ yr} = 2.2\text{E-}3/\text{yr}$. This is the "Inverted Mean" approach. A second approach would be to take a geometric average of the inverse upper bound (e.g., 900 years) and the inverse lower bound (e.g., 1 year) or $\text{sqr}t[(1/1\text{yr})(1/900 \text{ yr})] = 3.3\text{E-}2/\text{yr}$

Precipitation Infiltrates Repository

It takes 100 to 300 years for surface water to infiltrate to the repository, with mean of 200.

Waste Package Corrodes Through

If a particular waste package is wetted, it will corrode through in 200 to 400 years, with mean of 300.

Fissile Material Separated from Nuclear Poison and Assembled into Critical Configuration

The time for a corroded waste package to have poison removed, and resulting fissile material critically configured is 50 to 150 years, with mean of 100.

Other Factors

100 casks have sufficient fissile material to go critical if reconfigured. Thus, we need to

multiply resulting calculation by a factor of 100.

Figure 2 shows the analysis for a mission time of up to 1500 years. Note that the error resulting from a probability-space calculation (i.e., multiplying independent probability events together via a fault tree) is as high as four orders of magnitude from the more appropriate time-based calculation.

To perform the time-based calculation, we used the latest version of SAPHIRE with the new "time-series" plug-in. We explain these steps in the next section.

Overview of New SAPHIRE Time-Series Plug-In

SAPHIRE can now perform Monte Carlo calculations for the time domain via a new time-series basic event calculation. For this calculation, the inter-related time-based events (for a particular scenario) are represented in the fault tree under a single compound event. This compound event is just a normal basic event as far as the fault tree is concerned. In Figure 3, we represent this event as **COMPOUND** (where any name, up to 24 characters, will do). Note that you *do not* represent any of the individual time-based events in the fault tree; instead, they will be "hidden" within the probability calculation for the basic event named **COMPOUND**.

Next, even though the time-based events are not in the fault tree, we need to add them to the database (so the time-series calculation can use them). Thus, enter the applicable time-based events using the usual **Modify, Basic Events** option. These events would be items like **ICE_AGE** were it would be a uniform distribution from 1 year to 900 years. Note that these events must represent *time* (specifically, time to a particular event such as an initiator, failure, or recovery from a failure). Continue adding the remaining time-based events (**PRECIPITATION**, **CORRODES**, etc.)

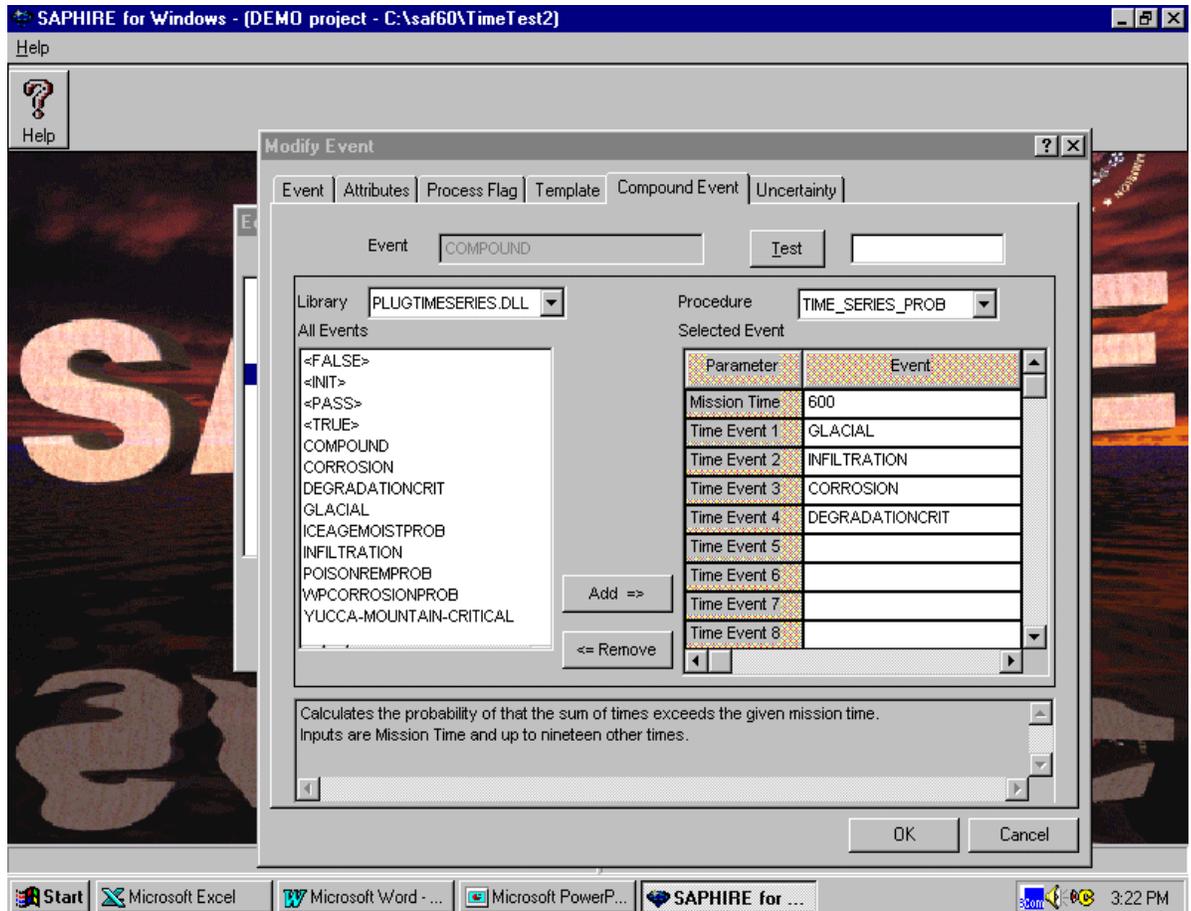


Figure 4. The SAPHIRE screen representing the compound-event information.

After all of the time-based events are added to the database, go to the time-series calculation event (still in the **Modify** screen), which, in our case, is the event named **COMPOUND**. We need to modify this event to make it use the new time-series calculation.

When we edit an event (under **Modify, Basic Event**), we will see a tab for "Attributes." Click this tab and select the option "Compound Event" in the "Special Use Flags" box. Then, click the "Compound Event" tab. At this point, we have told SAPHIRE that the event is a "compound-type" event, but we have not told it what plug-in calculation to use (a plug-in is simply a user-defined external calculation). So, we need to identify what calculation to use, which, in our case, is the "time series" calculation.

To identify the plug-in calculation to use, click the the "Library" option (still on the "Compound Event" tab). Select the:

PLUGTIMESERIES.DLL

option. Now that we have told SAPHIRE to use the time-series calculation, we need to tell it which parameters to use for the calculation. First, we can represent the mission time as a basic event or by typing a value directly in the "mission-time" location (i.e., the top of the event list). Now, we need to identify each of the time-based events in our scenario. To do this, highlight the basic event (from the left column) that represents the first time event and then highlight the "time event 1" field (from the right column). Click the **Add ->** button. Repeat

this process for each of the time-based events.

The output for the compound event will be a probability, representing the fraction of time that the *cumulative time* for the sequence of events (identified in the right column of Figure 4) is less than mission time. Pressing the "Test" button will calculate this probability so that you can see if the calculation is working correctly or not.

Conclusions

Since the fault tree *does not* treat the **Compound** event as multiple events, only one importance measure is generated for the whole event. To see the effect of changing subevents (i.e., those time-based events we identified above) within the compound event, one would have to use SAPHIRE change sets to perform a sensitivity analysis.

Time-based scenarios occur in both inside and outside the nuclear reactor industry. Analysts need to be aware that such problems require special analysis treatment. With this capability now in SAPHIRE, the analyst can be assured of proper treatment more realistic calculational results.

SAPHIRE Tip - Storing Cut Sets

If you have used SAPHIRE awhile and have read the on-line help, you know that SAPHIRE has two places to store the minimal cut sets that you generate. First, the analysis cut sets are automatically stored in the "current case" part of the SAPHIRE relational database. Second, you can move the current-case cut sets into a "permanent" storage location in the database. This second storage place is known as the "base case."

Now, lets assume that you have 15 sensitivity analyses to perform on a fault tree and you would like to store the cut sets results for each case in SAPHIRE. How can you do this?

Well, new to version 6.x is the ability to *save* cut sets directly to an end-state.

Best of all, this end-state is user-defined (at the time you save the cut sets) and can be named anything up to 24 characters long (do not forget to provide an appropriate description). To access this "save" option, generate your cut sets as normal (via the **Solve** option). Then, for the fault tree, select the **Display, cut sets** option. At this point, you should notice a "save" button. Click this button to bring up a dialog box prompting you for an end-state name and description. Type in an name (say "my_end_state") and description and then click the "save" button. You have just saved the cut sets for this fault tree to an end state.



Note that the cut sets you just generated (i.e., via the **solve** option) used the current-case data for the quantification portion of the analysis. When you go to the end state (e.g., my_end_state) to view the cut sets, the cut sets will reflect the data that is in the current case at the time you view the cut sets (since these cut sets for the end state are stored in the end-state, current-case part of the relational database). Consequently, if you move to the next sensitivity analysis (which entails changing the current-case data) and then go look at the previous cut sets (stored in my_end_state), the *order* of the cut sets may change since the data has changed. Thus, it is good practice

to note which change set(s) was used to generate its associated saved end state. Ways to note the change set information include:

- Type the change set name into the end-state description.
- Type the change set name into the end-state text. The "text" option is available under the **Modify, End State, Text** option.
- Record the change set name to end state relationship out side of SAPHIRE (e.g., in a WordPerfect file).

There is no limit to the number of end-states you may have in a database (other than available hard-drive space).

The cut sets that you store in an end-state will remain as part of the relational database. If you would like to remove these cut sets, you will have to utilize the **Modify, End State** option. Here, you can delete an end-state from the database. If you attempt this, you will be prompted to make sure that you really want to delete the data. Once deleted, you can not recover the data, so make sure that you really want to remove the data from the database.

The SAPHIRE TV&V

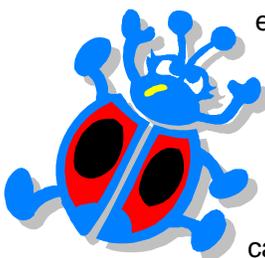
Older versions of SAPHIRE have been tested in formal verification and validation (V&V) processes. For example, both versions 4.0 and 5.0 were subjected to a V&V that consisted of the following steps:

- Preparation of a V&V plan
- Evaluation of the code development control procedures
- Test case development
- V&V testing
- Documentation of the test results and recommendations.

These steps are consistent with the IEEE's "Standard for Software Verification and Validation Plans" (IEEE 1012-1986).

For later versions of SAPHIRE, specifically versions 6.0 and 7.0, a new process was used for testing the software. We call this process testing, verification, and validation (TV&V), with an emphasis on the testing portion.

The benefits of the TV&V process over that of the older formal V&V are many. In general, the majority of the effort



expended in the testing is spent on developing rigorous tests that focus on potential bugs in the calculational areas of

SAPHIRE, namely the generation and manipulation of minimal cut sets. More specific benefits of the TV&V over the V&V process include:

- The TV&V process is less

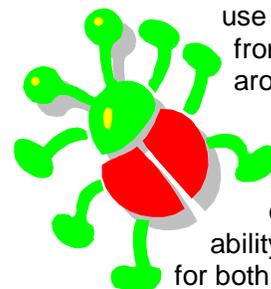
prone to human error since the test script performs each test, is repeatable, and has a single (known) set of results from which to compare the analysis results against.

- The TV&V process is consistent and thorough since the test scripts can be constructed to test single or multiple portions of the software once or numerous times. Tests that would be burdensome to analysts (e.g., many repetitive tests, very long test runs) can easily be performed by the testing software.
- The TV&V process is applied to every release of SAPHIRE rather than specific versions or releases at some selected point in time. Since the old V&V was analyst time-intensive, this option of testing each version of the software simply was not available.
- The TV&V process actually encourages difficult tests to be performed. The old V&V process was limited in the number of tests that could be performed since the tests were analyst time-intensive. The new process utilizes automated testing software which, in practice, is fairly insensitive to the complexity of a particular test.
- The TV&V process readily builds upon prior knowledge gained from the testing process. Since every test is re-run for each release of SAPHIRE, the initial testing conditions for each release is at least that of the previous release.

Like most software development projects, time and budget constraints prohibited exhaustive testing. So, the TV&V effort focused entirely on quantitative PRA results produced by SAPHIRE. Very little time was put into consideration and testing of non-quantitative aspects of the software.

Even though the current focus is on the quantitative portion of SAPHIRE, it would be incorrect to assume that the other areas of the software (e.g., the user interface) were not tested. During the operation of the automated test calculations, the testing software mimics the actions taken by an analyst. These actions include moving the cursor, selecting objects, clicking on-screen buttons, and typing information into SAPHIRE. Since a wide variety of PRA tests are included in the suite of tests, much of the user interface portion of the software is tested along with the calculations.

In addition to these automated tests, the SAPHIRE software is tested in the day-to-day use it receives from users around the world. Hundreds of users rely on the calculational ability of SAPHIRE for both risk and reliability calculations.



Included in these users are U.S. National Laboratory personnel; U.S. and foreign government regulators, university students, and nuclear power plant PRA analysts. While the TV&V does not credit this ad hoc testing, it should be pointed out that this testing mechanism is still an important addition to the automated testing.

Developing the Tests

To develop the automate tests, we first identified a list of important SAPHIRE features; features related to typical PRA calculations that analysts perform. By determining this set of important features, selecting various models which utilize these PRA features, and by writing test scripts to exercise and verify results of these features on these models, it can be reasonably concluded that SAPHIRE produces valid results. A summary of the PRA functions tested by the test suite is as follows:

1. Fault tree analysis.
2. Event tree and sequence analysis.
3. End state analysis.
4. Importance Measures.
5. Uncertainty analysis.
6. Change set feature.
7. GEM initiating event and condition assessments.
8. Data utility functions.

Running the Tests

To perform the tests, all test scripts and test databases are first stored

on a network drive, accessible via version control software. The version control software tracks *all* changes by author and time. The version control software stores and marks the changed copy as the newest version, but retains the old versions as well for historical purposes.

Prior to running the test suite, the latest scripts are "checked out" of the control library and compiled. The compiled tests, along with the database files and SAPHIRE, are transferred to the test machine on which the tests are to be run. This delivery mechanism allows us to quickly test SAPHIRE on a variety of computer platforms and operating systems. Currently, SAPHIRE is supported for the Microsoft Windows™ operating systems of Windows 95, Windows 98, and Windows NT. The SAPHIRE software should function properly under derivatives of these operating systems (e.g., Windows 2000™), but the TV&V has not evaluated these other operating systems.

The test suite is then started. For each test, the test database is started from a new "fresh" database that is in a known state. The compiled test script then runs a series of test scenarios on that database, recording expected results and any deviations into summary and detail files. These results files are named according to the run date and particular test. If

SAPHIRE fails a test, the cause is investigated and fixed, and the entire process is repeated.

Testing Results

As versions of SAPHIRE are released, new results of the testing are generated. The SAPHIRE Users Group provides an Internet location listing recent changes to the SAPHIRE software and downloadable results of TV&V tests. These files may be accessed at:

<http://saphire.inel.gov/recent.htm>

The two basic conclusions that came out of the TV&V process are:

1. SAPHIRE performs accurate PRA analysis calculations since areas required for these calculations have been tested with a reasonable degree of confidence.
2. Automated testing allows each new version of SAPHIRE to be tested for accuracy at least as well as the previous version. Adding additional tests to the test suite over time will increase the overall confidence in the software performance.