INL/EXT-14-31884

SAPHIRE 8 Web Based Application Specification Plan to Implement a Web Based Implementation of SAPHIRE

Steven Prescott
Curtis Smith
Ted Wood
James Knudsen

April 2014

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-14-31884

SAPHIRE 8 Web Based Application Specification Plan to Implement a Web Based Implementation of SAPHIRE

Steven Prescott
Curtis Smith
Ted Wood
James Knudsen

April 2014

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

U.S. Nuclear Regulatory Commission
Washington, DC 20555

Idaho National Laboratory

April 2014

SAPHIRE 8 Web Based Application

Specification plan to implement a web based implementation of SAPHIRE

By Steven Prescott, Curtis Smith, Ted Wood, and James Knudsen

Revision 2

Chapter 1

Introduction

Problem Statement

Capabilities available because of advancements in web based applications.

Advances in web based technologies such as HTML5 have made it possible to achieve things previously only available to desktop applications. Now a web based application can have the capabilities and feel of a desktop application while retaining the benefits of a web interface. With this technology a web based version of SAPHIRE could run online, taking advantage of cluster computing power, distributed collaboration, and cross platform compatibility; all while maintaining the current capabilities.

Design Overview

The SAPHIRE web version will allow users to run SAHPIRE through a compatible web browser. The initial web version will be simplified but allow users to edit events, model Fault Tree and Event Trees, and perform quantifications. Following versions will include existing features of SAPHIRE and incorporate new features such as revision tracking and transparency items.

When online, users will have access to models residing on the server and powerful solve engine capabilities. Ideally, the user interface for the web version when online would be similar to the offline user interface version. When offline, the models need to be stored and accessed through a local interface located on the user's local computer. Figure 1 illustrates the offline and online version of SAPHIRE will be set up.

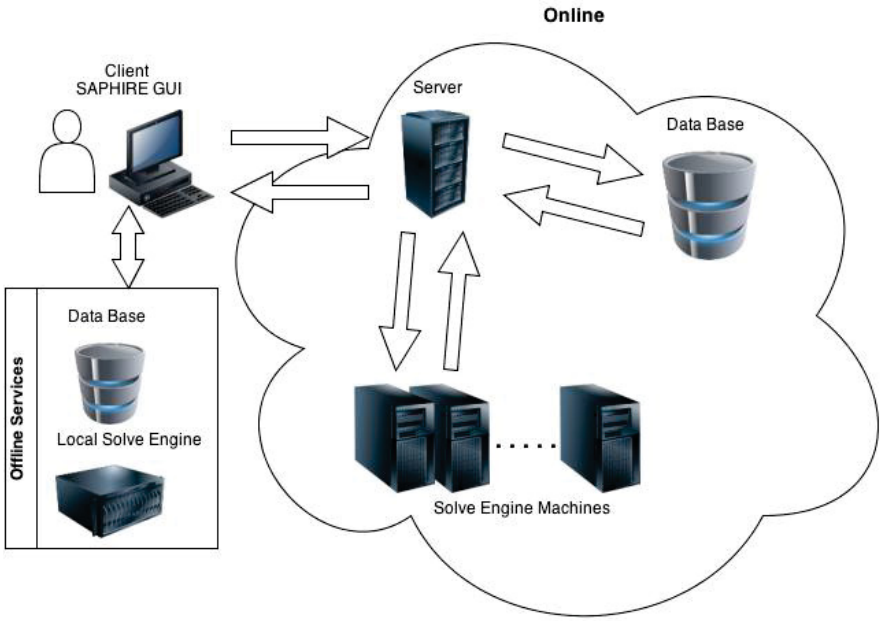

Figure 1 - Ideal design for a web version of SAPHIRE

However, the technology to allow HTML5 communication with data on the client PC is still evolving and there are limitations. These limitations will be detailed later in this report. While these key features are being developed, implementing the web version in stages and taking advantage of the "Cloud" based advantages is the best option.

First the "Cloud" design will be implemented to handle model storage (i.e., fault tree and event tree logic and data) and the different solve engine pieces. Next a browser based user interface will be developed to view, modify and solve models. Finally the installed version of SAPHIRE will be modified to interface with "Cloud" options for model uploading, downloading and solving. Figure 2 illustrates how the design and implementation of the web version of SAPHIRE will developed.

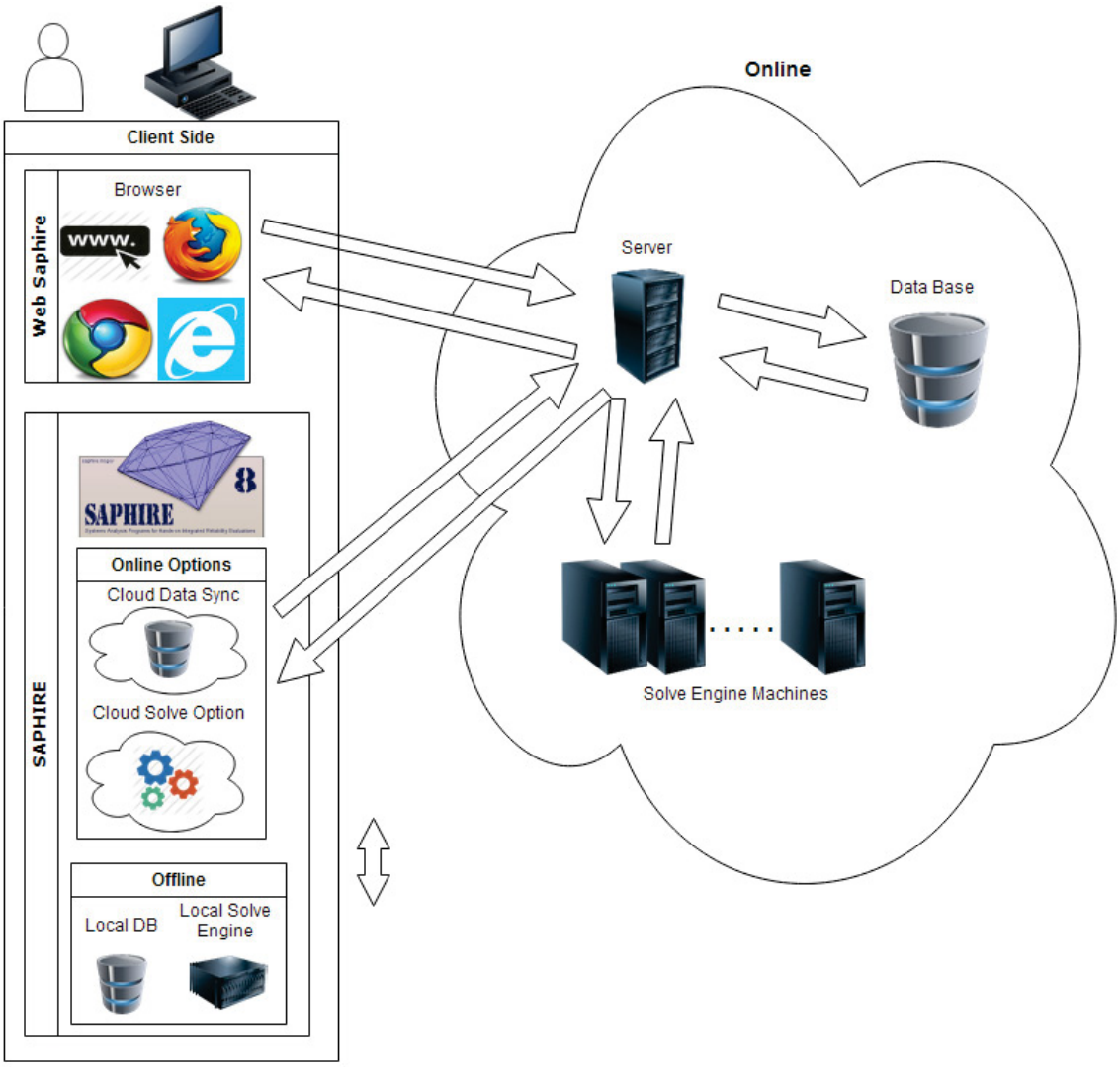

Figure 2 - Staged design approach

Users running SAPHIRE as a web version will be able to use any compatible browser and always have access to the latest version of code and the latest versions of the models. This will reduce compatibility problems and issues resulting from older versions of the software and/or models.

The installed version of SAPHIRE will be modified for users to run offline or online in a highly secure manner. For those users who are online, model synchronization will be simplified to reduce issues and allow the user to take advantage of the online solve engines for computationally intensive models.

Chapter 2 Design Details

On/Off-Line Option

The need or focus on platform independent applications and anywhere access has helped shift development to web based technologies. To achieve these goals much development recently has been done in HTML5 and the use of JavaScript. Items such as Google Docs, Cloud 9, draw.io and similar applications show how advanced web applications can be running online. Other requirements have shown a need for these applications to run on or offline. A few simple methods have been developed for this purpose.

Some proprietary technologies such as XBAP, Silverlight, and Flash have made it possible to run applications in standalone or online versions, but limit cross platform capabilities. This makes them less desirable for many applications. Example – Netflix uses Silverlight to achieve many of its features including video streaming. Thus it has required extensive software packages on non-Windows platforms to run and is currently still not officially supported on a Linux OS.

In order for a highly interactive web application to run smoothly, much of the work must be done on the Client Side instead of the Server Side. If running the application offline, server side activities such as database access must be done locally. HTML5 does not do all of this directly, but enables application programming interfaces (APIs) along with JavaScript (2). This allows you to achieve some limited local persisted storage and offline application support. Figure 3 provides an overview of the on and offline architecture that can be used. Although HTML5 is widely used, it is currently not an official standardization and so not all abilities are supported by all browsers. World Wide Web Consortium (W3C) plans to make a formal recommendation by the end of 2014. (1)

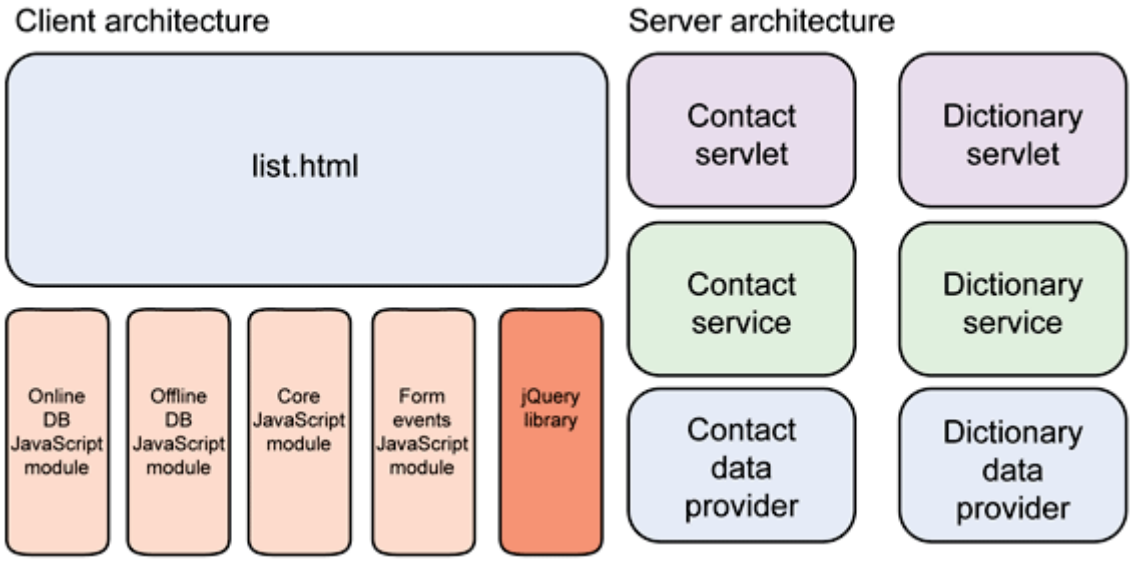

Figure 3 - On and Offline architecture example

One of the main challenges for making a web application able to run offline is access to local recourses. HTML5 has helped overcome some of these issues. However, for

security reasons, access is limited or restricted and the user must initiate all connections to local recourses. To alleviate some of the excessive user permission issues, HTML5 proposes the use of single authorization of a file for the duration of a session. It also proposes read-only access to file header and directory information so an application could scan for available files. (*5*)

In order for a browser based SAPHIRE version to run effectively offline, some parts of the software will have to be "Installed" locally. Figure 4 provides an illustration of how the web version may need to be parsed up between "Cloud" resources and local software. These would include the PRA solve engine, a data base, and currently a simple local server. This server would be needed to establish direct access from the browser to the local resources. The server would verify the latest versions of the required software parts are installed locally.

Installing these multiple required resources onto a user's machine via the web browser is becoming easier with tools such as "Click Once" (6*)* and BitNami. The use of a local server has been used by developers for a long time, in order to test web sites, but the idea of a simple server being part of the web app, is an emerging technology. These tools make it easier, but there is no adopted standard procedure for setting up a client thin server.

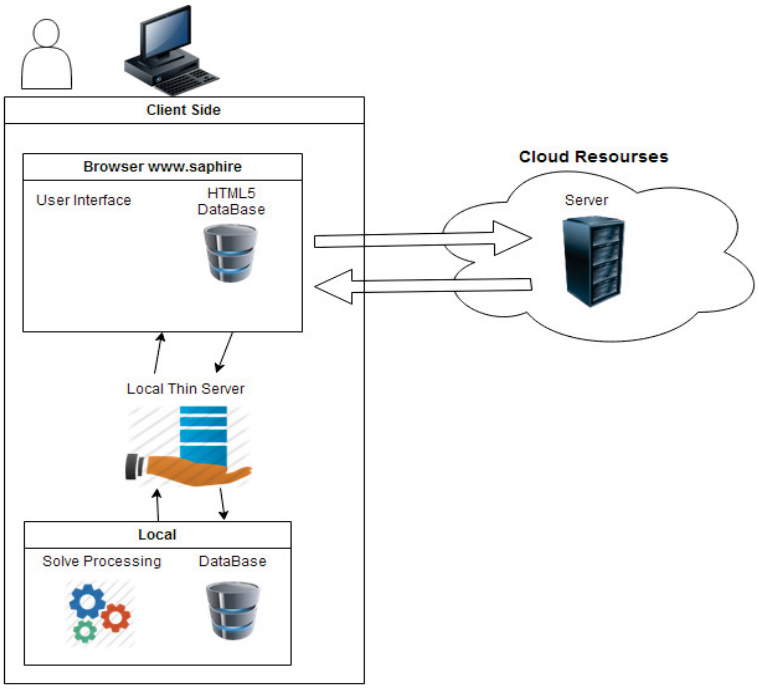

Figure 4 – Locally installed software for web version of SAPHIRE

Display GUI

Several packages for web development of graph based tools exist. These packages enable web based applications to construct complex graphs without delays or refresh

updates previously caused by many web pages. This is achieved by shifting the processing to the local machine instead of making calls to the server. SAPHIRE will use, MxGraph (3), to draw the graphical representation of Event Trees and Fault Trees. Unlike many of the other tools, MxGraph meets most of the needs for a web version of SAPHIRE. It is also easily customized to add other features.

With the increasing use of handheld and touch devices, user interface designs have also shifted. In developing a web based version of SAPHIRE, different design styles will need to be reviewed to determine what would most clearly display information to the user, and allow for easy model development.

Ease of transition is important to users, so a design close to the desktop application will be used for the web interfaces. Figure 5 is a mockup example of the new SAHPIRE browser interface. HTML5 has Drag-D-Drop, and right click popup capabilities, which will allow for a similar feel for interfacing with a model.

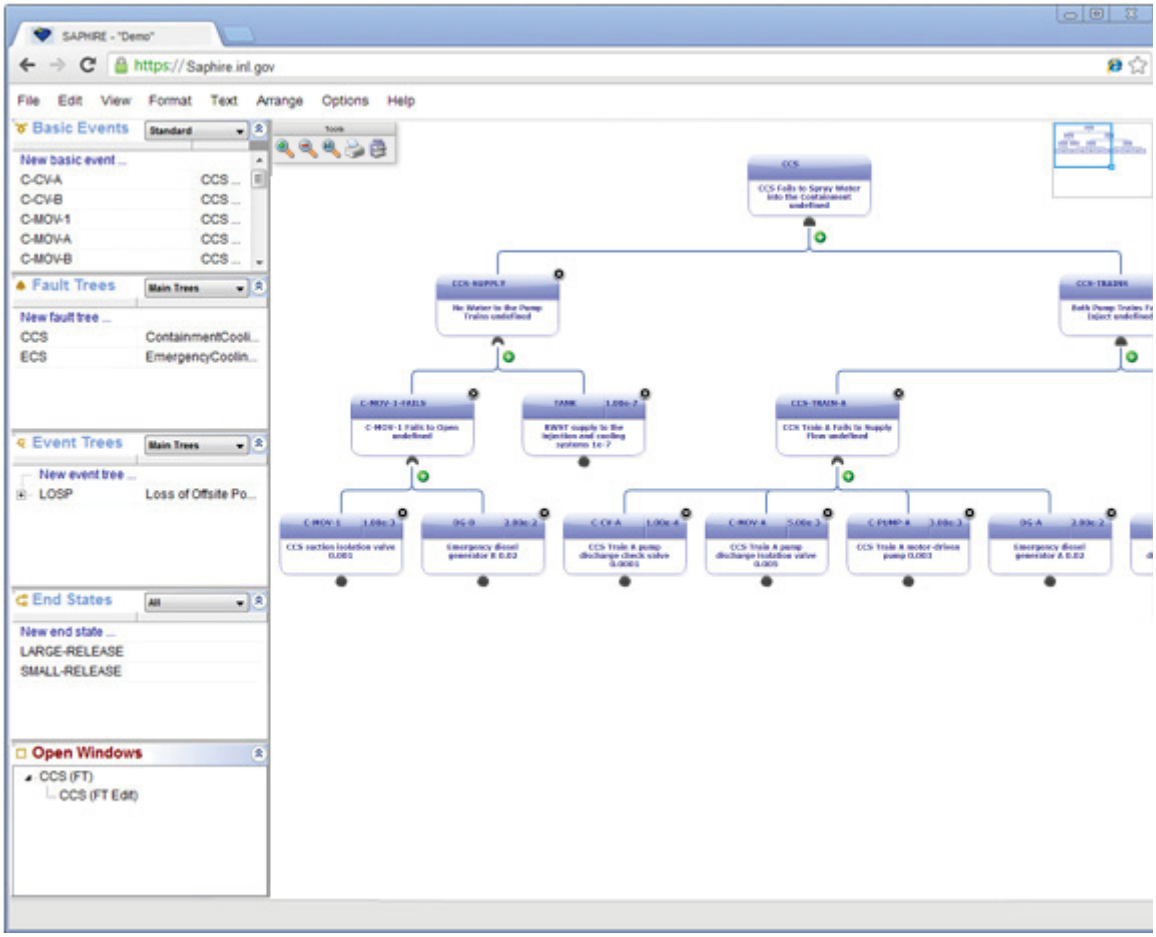

Figure 5 - Web interface similar to SAPHIRE desktop application

Solving Engine

Another advantage of having a web application is that the user interface is by default, detached from the solving engine. This would allow for multiple solve engine options, updates, or hardware upgrades without an adverse effect on the user.

When a web application needs lots of computation power, this is shifted off of the web server and sent to another machine, or a cluster of machines, shown in Figure 6. This allows for easy expansion when more users need the system. A starting web version of SAPHIRE could run with as little as one machine dedicated to performing solve calculations. Currently a 16 core machine costs about $6,000 and would sustain up to 8 users running small calculations simultaneously.

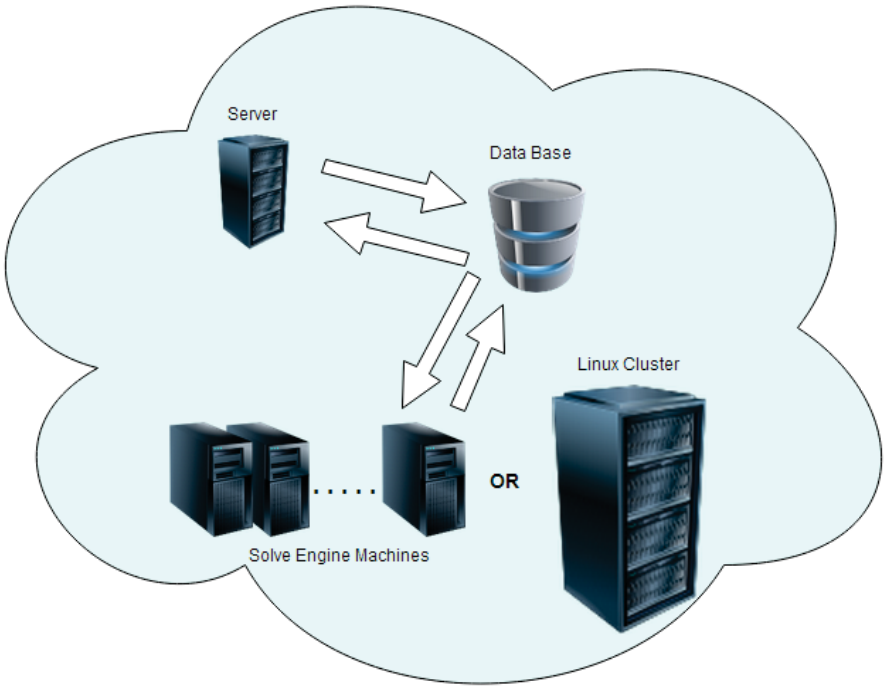

Figure 6 - Solve Engine Hardware Design

The solving engine needs to match the capability of the current SAPHIRE solving engine. The functionality needed includes the following:

1) Change set application
2) Logic translation
3) Cut set generation
4) Post-processing rule application
5) Cut set updating
6) Cut set quantification
7) Event tree linking
8) Nominal case updating
9) Partition rule application
10) End state gathering
11) Uncertainty analysis
12) Importance measure creation.

These functions are described in detail below.

Change set application is the process where the user can apply a temporary basic event value (including logical true, false, ignore) or process flag to any number of basic events. These temporary values are stored in the change sets and are applied to the existing events.

Logic translation is the process where the current basic event data that was updated by the change set application is combined with the logical value changes (true, false, ignore) and process flags described in the flag sets that are associated with the logic. The logic is pruned or adjusted based upon the logical state of each of the basic events. This has been called house event pruning. This logic translation also streamlines the fault tree logic to reduce the number of gates by coalescing like gates, determining independent sub-trees and creating gate modules. This process also optimizes the fault tree logic so it can be solved in the most efficient way possible.

Cut set generation is the process which follows the failure paths of the logic to create cut sets that list the combination of events sufficient and necessary to cause failure. There are several methods of generating cut sets including traditional cut set generation, the use of truncated binary decision diagrams or satisfiability (abbreviated SAT) solvers.

The current version of SAPHIRE uses the traditional method or process of expanding the gates, applying boolean absorption, and cut set truncation. Gate expansion replaces each gate with their inputs. Boolean absorption applies the following identities to the cut sets:

1) A*A=A
2) A + A*B=A
3) A*B*/A=0
4) //A = A

The first identity (idempotent relationship) prevents two identical events from appearing in the same cut set. The second one (absorption relationship) is the most computationally difficult to apply. In terms of set theory it consists of eliminating subsets, because A*B is a subset of A. It can also be thought of as eliminating supersets; A*B is regarded as a larger entity than A because it has more tokens to manipulate. Both the subset and superset terminology can be found in the literature, but this document will use only the term "absorption." The absorption identity is used to eliminate cut sets that are non-minimal. The third identity (exclusion relationship) implies that no cut set will contain both the failure and the success of an event. The fourth identity (double negation relationship) states that the complement of a complemented event is the event itself.

The final step, cut set truncation, involves the elimination of cut sets that fall below the truncation limits defined by the user.

Recently, SAPHIRE has been changed to allow it to use a truncated binary decision diagram engine to produce cut sets. This is an area that could provide speed improvements in the web-based version of SAPHIRE. (7)

Satisfiability or SAT solvers might provide other advantages, but there currently no SAT solvers available for use with SAPHIRE. This is an area of possible research. (8)

Post-processing rule application is the process of modifying generated cut sets based upon rules. The SAPHIRE "post-processing rules" are textual logic rules that allow for the alteration or deletion of fault tree or sequence cut sets. Although previously called "recovery rules," the post-processing rules have evolved from the simple inclusion of recovery events into a powerful rule-based system for cut set manipulation. The post-processing rules can be used for probabilistic risk assessment techniques including:

1) The automated inclusion of sequence recovery events
2) The inclusion of common-cause failure cut sets
3) The elimination of mutually-exclusive events (e.g., impossible combinations of events).

Cut set updating is performed on existing cut sets. It is the process of applying boolean absorption and cut set truncation as described in the cut set generation section. This process is done after post-process rules have been applied to remove the non-minimal cut sets and cut sets that are below truncation levels that the rule processing might have created.

Cut set quantification is the calculation of individual cut set probabilities and the combining of those cut set probabilities. The individual cut set probabilities are determined by multiplying the probabilities of the applicable basic events.

$C_i = q_1 q_2 ... q_n$
where
C_i = probability of cut set *i*, and
q_k = probability of the *k-th* basic event in the *i-th* cut set

The exact probability of the union of the cut sets can be found, in principle, using the following inclusion-exclusion rule or min-max option. Basically, it is the sum of the probability of the individual sets, minus the sum of the probability of all possible pairs, plus the sum of the probabilities of all possible combinations of three, minus the probabilities of all possible combinations of four, plus the probability of intersection of all five minimal cut sets, etc. However, since all combinations of cut sets are evaluated, the min-max approach becomes intractable for calculations involving a many cut sets. Therefore, two approximations are often used by PRA software, the rare event approximation and the minimal cut set upper bound.

The rare event approximation is a common approach to calculate the probability for a top event. It adds together the probabilities for the cut sets. Thus, the rare event approximation is:
$S = \sum_{i=0}^{n} C_i$

Where

S = minimal cut set upper bound for the system unavailability,
C_i = probability of the ith cut set, and
n = number of minimal cut sets.

The default quantification of SAPHIRE is the minimal cut set upper bound calculation. The minimal cut set upper bound calculation is an approximation to the probability of the union of the minimal cut sets for the fault tree. The equation for the minimal cut set upper bound is:
$S = \prod_{1}^{n}(1 - Ci)$

Where

S = minimal cut set upper bound for the system unavailability,
Ci = probability of the i-th cut set, and
n = number of minimal cut sets.

The current version of SAPHIRE also has a cut set BDD algorithm that can be used to quickly calculate an exact answer on most groups of cut sets. This algorithm is one that could be enhanced in a web version of SAPHIRE.
Event tree linking is the process of generating sequence logic using the event tree graphical files and linkage rules. Once the logic is created each sequence can then be solved using the steps above to create minimal cut sets.

Nominal case updating is the process of copying the current case cut sets and uncertainty data to the nominal case. This nominal case can then be compared when changes are made to the current case.

Partition rule application is the process of using partition rules to assign each cut set an end state. Those cut sets can then be grouped together in the end state gathering process defined below.

End state gathering is the process of grouping event tree sequences' cut sets into end states. This grouping can be done by sequence defined end states or individual cut set defined end states. The individual cut set end states are assigned using the partition rule application defined above.

Uncertainty analysis is the process of calculating the uncertainty in the top event probability. SAPHIRE already has the top event expressed in terms of minimal cut sets which cut sets depend on many basic events, each of which has a probability described in terms of some parameter(s). For all the basic events, SAPHIRE randomly samples the parameters from their uncertainty distributions, and uses these parameter values to calculate the probability of the top event. This sampling and calculation are repeated many times, and the uncertainty distribution for the probability of the top event is thus found empirically. The mean of the distribution is the best estimate of the probability of the top event, and the dispersion quantifies the uncertainty in this probability. For an accident sequence the process is the same, except the sequence fault tree is preceded by an initiating event, whose frequency is also quantified by an uncertainty distribution. The term Monte Carlo is used to describe this analysis by repeated random sampling. Two kinds of Monte Carlo sampling are simple Monte Carlo sampling and Latin Hypercube sampling.

Importance measure calculation is the process of finding importance measures. These measures are found empirically by evaluating each basic event used in the cut sets of the respective fault tree, event tree accident sequence, or end state. SAPHIRE calculates seven different basic event importance measures. These are: Fussell-Vesely ratio, Risk reduction ratio, Risk increase ratio (Risk achievement worth), Birnbaum (the so-called first derivative importance), Uncertainty Importance, Risk reduction difference, and Risk increase difference. The ratio importance measures are dimensionless and consider only relative changes. The difference definitions account for the actual risk levels that exist and are more appropriate when actual risk levels are of concern, such as comparisons or prioritizations across different plants. For purely relative evaluations, such as prioritizations within a plant, the ratios sometime give more graphic results.

Compatibility / Versions

An initial simplified web version will be first developed as a proof of concept and successful design validation from users. This simplified version would allow for basic

modeling, quantification, and validation. It would also provide a jump point for a successful full feature version.

In order to achieve successful transition from a SAPHIRE desktop application to the Web version, users would have to have the ability to transfer between the two. An initial project convert operation would be developed to move a desktop model to the Simple web version. From there the Web version would always contain a way to move data to the desktop version. Once the web version is available, the next release of the desktop version would allow for porting to the web version. It may even be possible to provide a Sync option which would allow updating between a web model and a desktop model, illustrated in Figure 7.

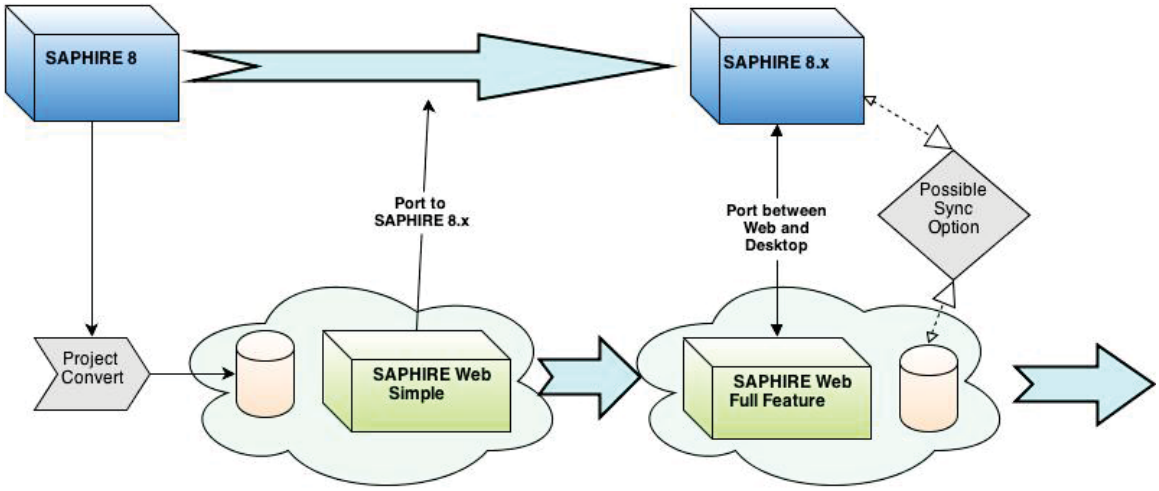

Figure 7 - SAPHIRE Desktop and Web version timeline

Data Base Storage

Currently SAPHIRE stores models on a custom database system. In order for SAPHIRE to run as a web application, data would need to be stored on a compatible Database Management System (DBS). A larger DBS would store and manage the models on a central server, shown in Figure 8 (Examples - PostgreSQL, MySQL, MS_SQL).

An option in database storage is “NoSQL” (Not Only SQL). NoSQL is the concept that there are other methods besides relational databases that may fit an application better. Some of these designs include Column, Key/Value, Document, and Graph based. Each has its advantages and disadvantages. For example, a delivery system would better fit a graph based DB in order to take advantage of built in capabilities for shortest paths. There are some advantages that could be beneficial to SAPHIRE such as trending. However most of the NoSQL designs don’t maintain referential integrity and are much slower when scanning, adding, and removing large amounts of data. Speed in this area is critical for SAPHIRE functions, and so a NoSQL DB will not be used for the main design of SAPHIRE.

An Ideal design would be to transfer the database for a model directly to a database maintained by the web browser and have our web application interface with to it with periodic synchronization to the server version. However options for this are currently limited. One promising option was SQLite – a small (~350 KB) relational a database management system library. It was capable of being integrated directly into the Client web application by some browsers. SQLite is a popular choice as an embedded database for local storage in many. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems. (4) However the web interface for SQLite, called WebDB was deprecated and will not be a standard for HTML5. Instead IndexedDB, a NoSQL design, is the new standard. The disadvantage of IndexedDB is its size limitation and lack of SQL interface. Currently JavaScript packages such as "SequelSphere" are being created to make interfacing with IndexedDB easier. These packages are also in their infancy and we expect them to have more capabilities as time progresses.

To begin our web based version will download sections of the model that the user is manipulating and retain that in memory. After modification, these sections will be uploaded back to the server. As compatible and easier to use technologies arise they will be incorporated to enable the entire project to be on the client browser with periodic synchronization to the server.

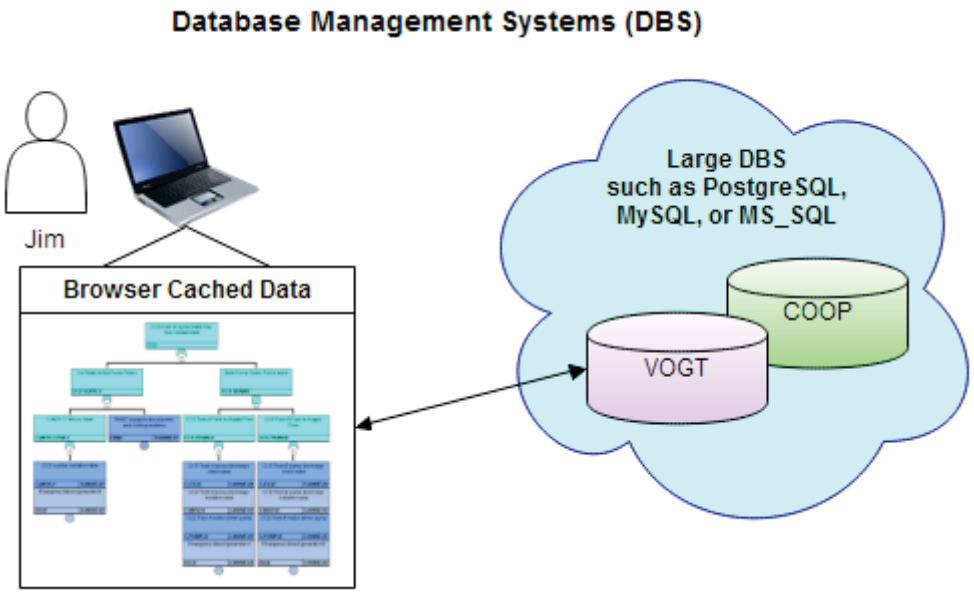

Figure 8 - Database Managements Systems

Multi User

One of the main benefits of a web based SAPHIRE is the ability for multiple users to access and modify the same information. This will allow users to have models on a

central repository and always have access the most current model. They could also collaborate on the development of new models.

A valid design would be to have a “Read Only” section of Baseline projects. The user would not be allowed to modify a Baseline project, only perform their local analysis. Figure 9 provides how the multi user option will work.

An “Active Projects” area would contain editable models. Each user would have their own models that they can work on and modify. For security, models in a user’s area would only be available to that user. Models could be moved by the owner between a users area to a shared area to allow collaboration between specified users. (See Shared area between Sue and John in Figure 9) When sharing a model, pieces of the model they are working on would automatically get a temporary lock. This would allow editing by the locked user and place all other users in a read only mode for that piece. After changes are made and the piece is unlocked, the modifications with a notification would also be sent to any users of that model. This would provide a simple way of allowing multiple users, and maintaining model integrity.

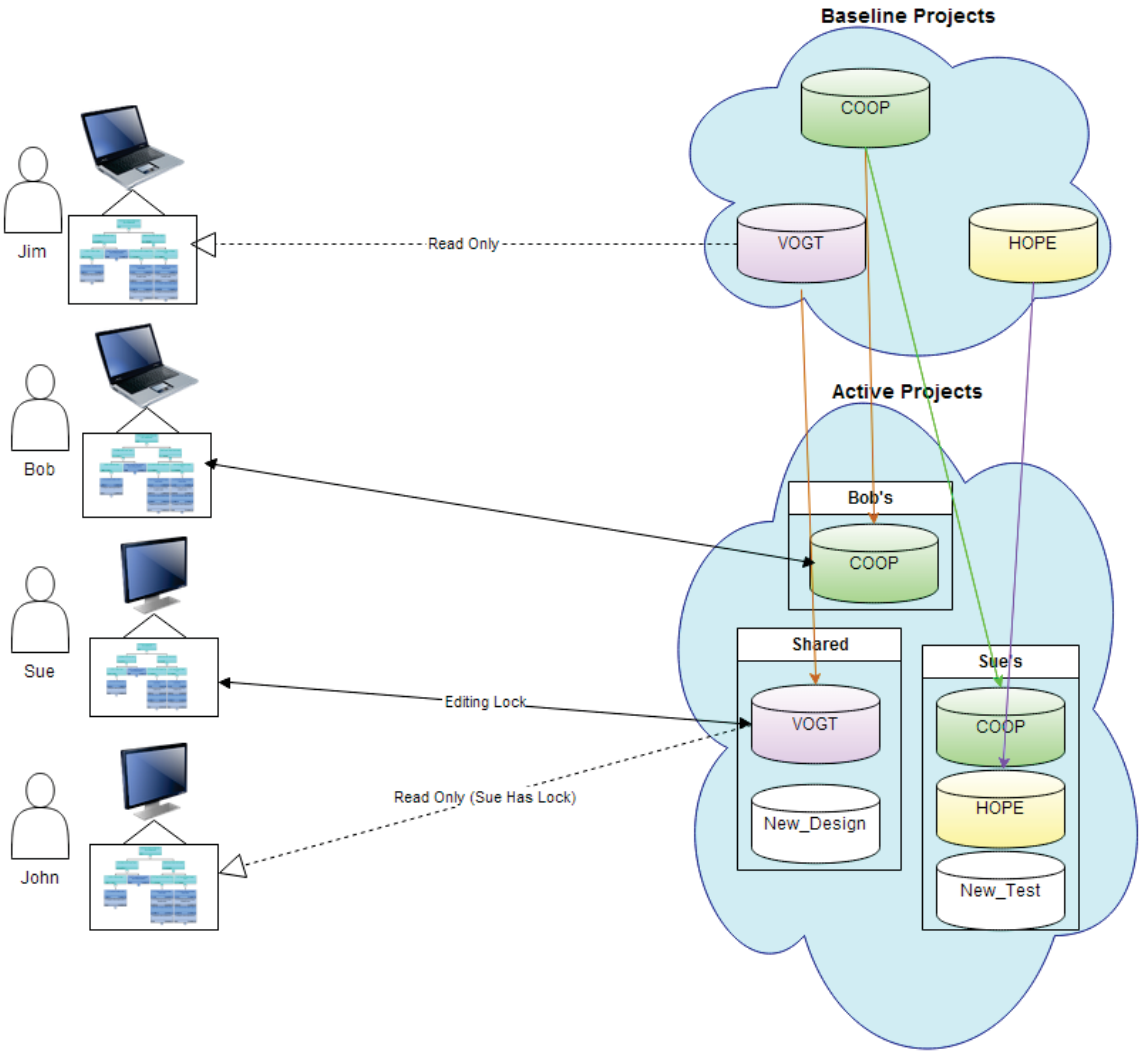

Figure 9 - Multi user model access

If users have proper permissions, they would be allowed to replace a Baseline Model with one from their or a shared Active Project area. This allows for easy updates to Baseline models while preventing unauthorized modifications to them.

Revision Tracking and Change Comparison

When giving multiple users the ability to modify the same model, many issues arise. One of those issues is knowing what has changed in a model, by who, and when. This issue will be resolved with the web application. The database system knows who and when changes are made and can track the changes. If a user is editing a model and makes changes to a few items, those changes can be compared to the original project and the difference saved in a change log along with a date/time stamp and the user who made the changes. Figure 10 shows the tracking system that will be used when projects are edited and stored. If a baseline project is replaced, a comparison between the two projects can be made, and a log of the difference added.

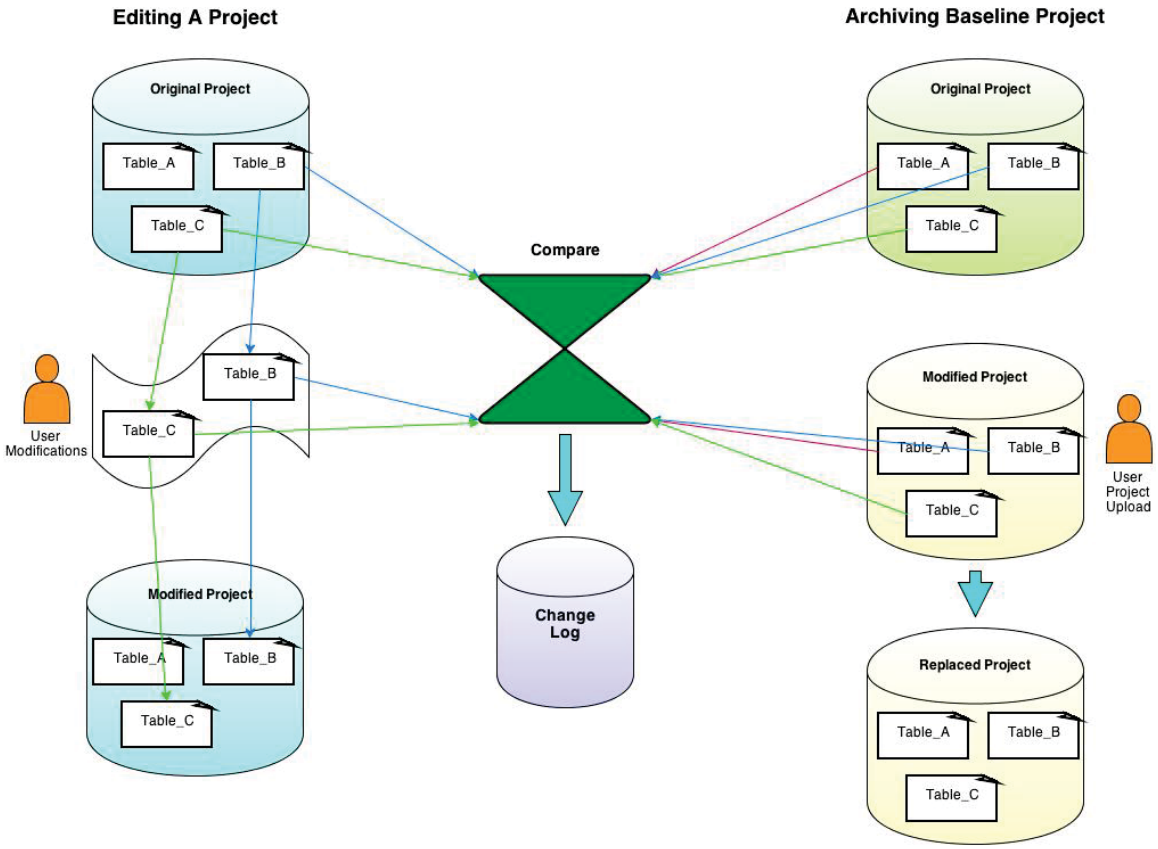

Figure 10 - Revision and Modification Tracking

This feature will allow for easy identification of changes, and the possibility to restore back to a previous revision.

Transparency

The ability to easily understand the workings or how results are derived in a model is always of great importance to users. In designing and developing a web version of SAPHIRE we will retain current features that help achieve transparency. We will also implement other ideas that will improve transparency. Some of these items may include Advanced Cut Set Compare, Document Integration, and Model Tracing.

Advanced Cut Set Compare – Allow the user to compare cut sets from different stages in the solving steps. For example, allow the user to compare cut sets before and after Post Processing rules are applied.

Document Integration – Enable more advanced options to attach information to items. This information would be used to generate reports and develop dependency matrix information.

Model Tracing – a hierarchical-type of diagram could be created in order to see the relationships of key parts of the model, whether they are basic events, fault trees, or event trees. For example, to investigate the support system initiating event (SSIE) fault tree modeling, the following diagram was manually created (Figure 11).

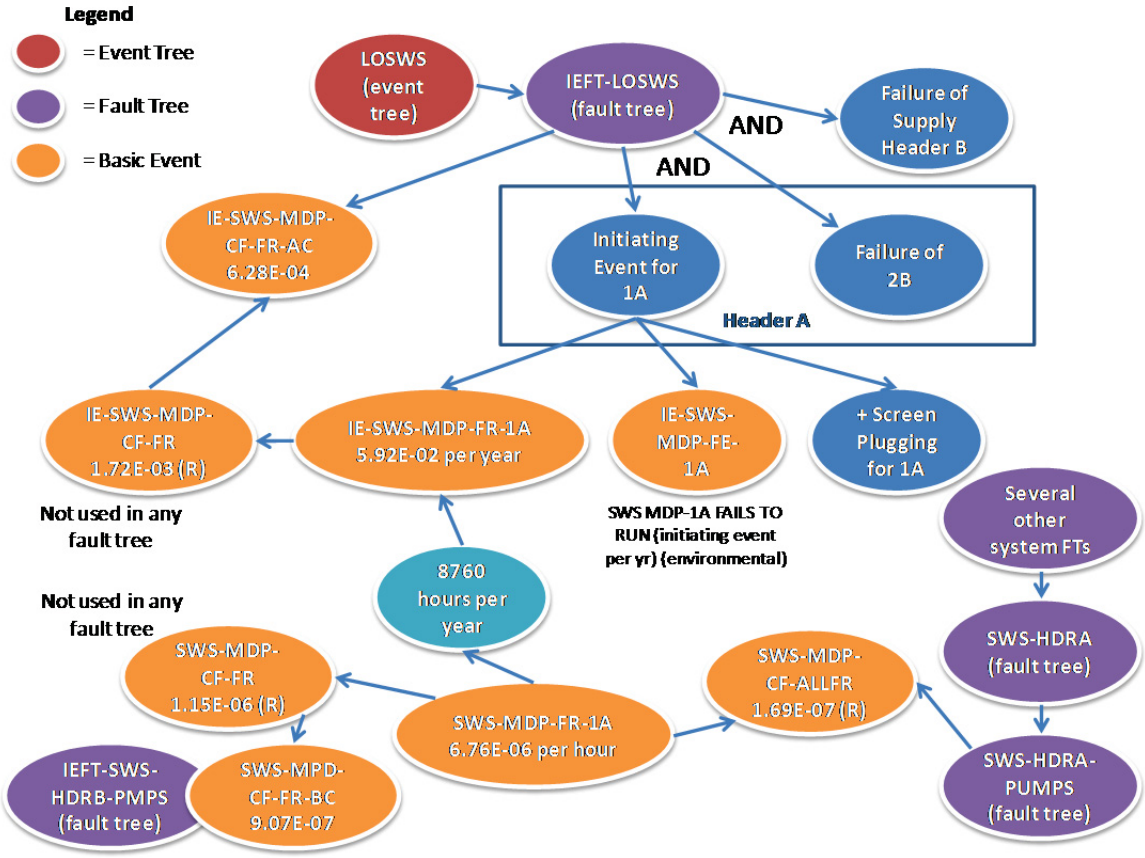

Figure 11 - Model Tracing Example

A module could be developed that would enable SAPHIRE to create this type of simplified diagram as needed to support end-user understanding of model details. These diagrams would be able to supplement the existing model documentation.

V&V

Verification and validation (V&V) of web-based software will entail all of the V&V procedures and documentation required through NRC, INL, and ASME NQA-1 standards. In addition to this effort, the SAPHIRE web version would require further V&V for the web hosting such as domain testing, throughput, stability and security. It would involve grey box testing, which is a combination of black box testing of its functionality and white box testing of its internal engine, accessed through a server. It also requires penetration testing of the program for security against hacking and exploitation. Penetration testing should be performed periodically even beyond V&V, even without a new release of SAPHIRE, since hacking and security against hacking is constantly evolving.

The capabilities of V&V at INL are not as comprehensive for web-based software as they were for the IV&V of the current desktop SAPHIRE 8 program. An approach of hiring an outside contractor to V&V portions the web-based SAPHIRE with INL oversight could be more cost-effective and comprehensive, especially in the realm of security and domain capabilities. V&V of other section such as the solving engine and interface features would still be done most effectively in-house.

V&V testing could use the following options similar to current desktop testing. Each module can have a set of before/after criteria executed from scripts it must run through before it can replace an existing module.

1. Functionality Testing – Link/Form/DataBase testing much can be automated.
2. Usability Testing – Ease of use, help hints, spelling
3. Interface Testing – Interrupts between transactions or double sent messages.
4. Compatibility Testing - Different Browsers, OS or Mobile users.
5. Performance Testing - Load Test, Stress Test.
6. Security Testing - Follow OWASP security testing standard.

Hardware

Hardware needed to implement a SAPHIRE web application.

Web Server - The web server is used to process client requests when online. It acts as a coordinator for model information in the database and sending solve requests to the Solve Engine computers. This would reside on INL's DMZ Server to allow access from any outside organization. One benefit of most stuff running on the client side is that there is much less of a demand on the web server, which makes it easy to handle a large number of users.

Database Server - Stores the models and user information. It keeps models up to date, authenticating permissions and preventing modification of locked models. This server would reside on a secure INL DMZ zone. Models stored on the Data base would be encrypted to prevent data access by any outside source.

Solve Engine - A machine or group of machines used to do the solving calculations of a SAPHIRE model. A simple expandable machine would consist of 16 cores with 32-36 GB of ram and cost around $6000. Each machine would support up to 8 users solving items simultaneously and would be easily expandable if the number of users expands. If more users try to solve than machines are optimally set up for, then all jobs will be slowed. A priority queue could be established to give some users precedence.

If security requires models to reside in-house, it would be possible establish a local Database Server for these users to connect to.

Progression Strategy

The development strategy will consist of a staged process where useable sections will be developed and functioning prior to moving to the next useable section. The starting development will focus on implementation of a cloud based solve engine (all of the pieces, i.e., cut set generation) and a web user interface for viewing Fault Trees and Event Trees.

Phase 2 will focus on a database design compatible with migration of data to and from a local PC to web server and also meet the requirements laid out in this design document. Phase 3 will work on the web interface to allow for modifying the model; solving the logic models using the cloud solve engine; and displaying and publishing the results. Lastly, Phase 4 will add a connection to the installed version of SAPHIRE so it can access the cloud based resources and download those resources to be used offline. Figure 12 shows an illustration of how the web version SAPHIRE will be developed.

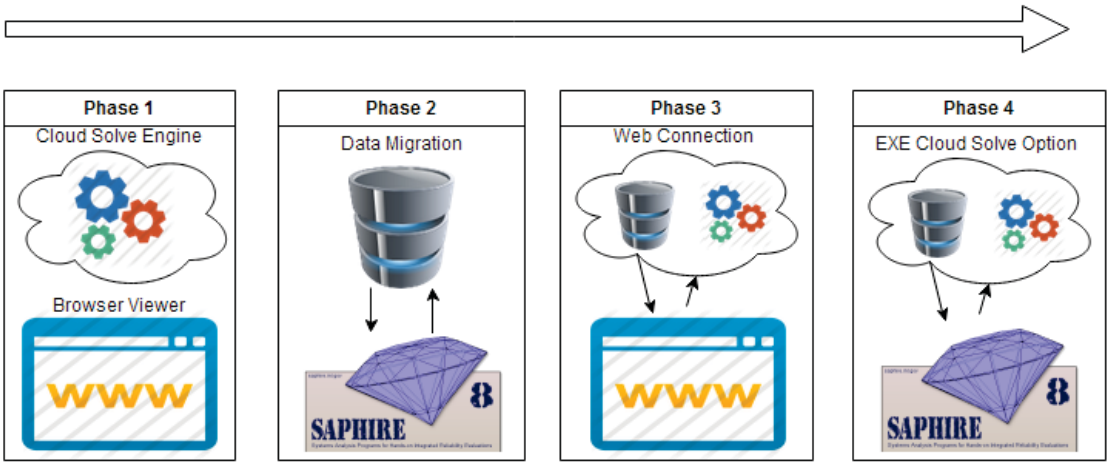

Figure 12 - Phase approach to development of web-based SAPHIRE

Current Progress

A beta standalone solve engine, which has the capability to solve fault tree(s) for Cut Sets has been developed. This is the first step in moving all of the solve engine capabilities to a web based version of SAPHIRE. This is only one piece of the solve engine discussed above, but it proved that this operation is doable. This solve engine worked on a simple test fault tree, but will require extensive testing. Another piece of the web version of SAPHIRE has the ability to display and modify fault tree(s) in a web browser. An example fault tree developed using this technology is shown in Figure 13.

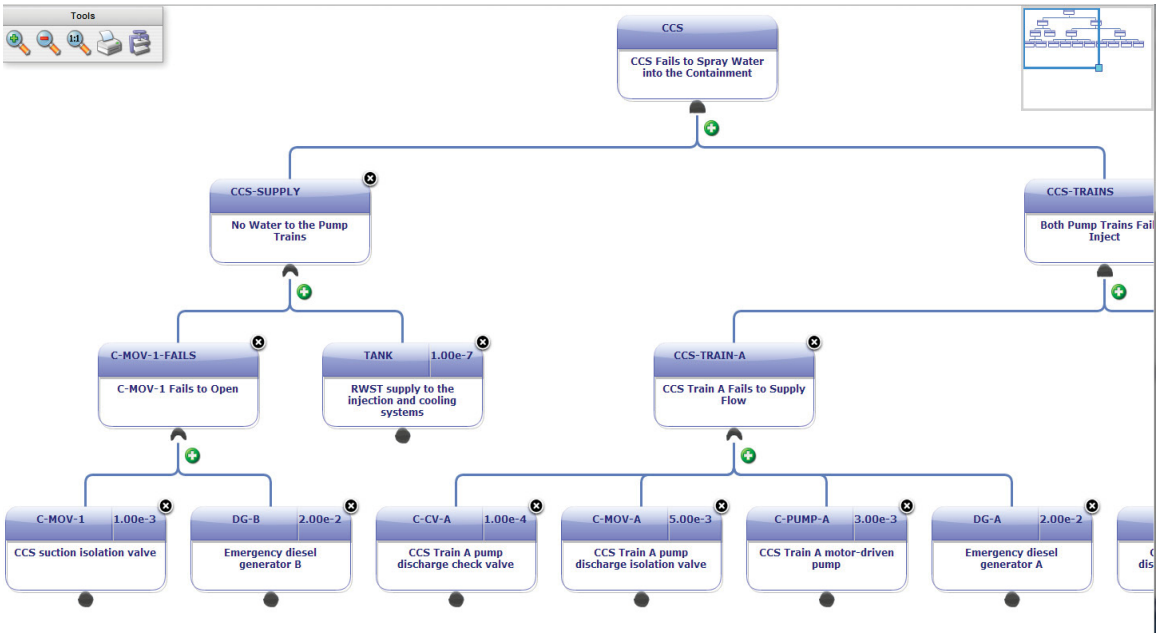

Figure 13 - Graphic model using graphic package for the web-based version of SAPHIRE

Chapter 3 References & Info

1 - http://en.wikipedia.org/wiki/HTML5
2 - http://www.ibm.com/developerworks/web/library/wa-html5db/index.html
3 - http://www.jgraph.com/index.html
4 - http://en.wikipedia.org/wiki/SQLite
5 - *http://www.w3.org/TR/FileAPI/*
6 - *http://en.wikipedia.org/wiki/ClickOnce*
7 - *Jung, W. S., Han, S. H., and Ha, J. A fast BDD algorithm for large coherent fault trees analysis. Reliability Engineering System Safety., 2004, 83, 369–374.*
8 - *http://en.wikipedia.org/wiki/Boolean_satisfiability_problem*

http://www.w3.org/TR/FileAPI/

MxGraph – is a Javascript library that uses built-in browser capabilities to provide an interactive drawing and diagramming solution. MxGraph does not depend on any third-party plug-ins and proprietary software. It works straight out of the box, no client configuration, no plug-in installation, no platform dependencies. (Free – Non-commercial or 5 developer licenses for ~$10,000)
http://www.jgraph.com/purchase.html

.

Chapter 4 Revision Log

Initial draft completed 1/23/2013.

Revision 2 draft completed 4/30/2014.

Chapter 5 Design Review

Reviewer Name(s):

Date:

Location:

Time:

Design Review Appraisal

Accepted: As is ( )

With minor modifications ( )

Note modifications:

Not Accepted: Requires major revision ( )

Requires minor revision ( )

Note revisions:

Additional Notes: