EMRALD, Dynamic PRA for the Traditional Modeler

Steven Prescott*, Curtis Smith, and Leng Vang
Idaho National Laboratory, Idaho Falls, USA

Abstract: Recently, dynamic probabilistic risk assessment (DPRA) has been used by risk researchers to analyze problems that are either difficult or impossible to solve using traditional fault tree and event tree methods. Several tools have been developed and are helping to advance system safety. However, multiple disciplines use traditional PRA modeling practices and have built advanced and complex models of their systems and DPRA modeling techniques would see faster and more wide-spread adoption if model creation and associated tools corresponded closer to traditional methods.

A new modeling approach was implemented in Event Modeling Risk Assessment using Linked Diagrams (EMRALD) which incorporates the following features:

- An intuitive web-based graphical user interface for modeling.
- Traditional modeling aspects including basic events, fault trees, and event trees, are all captured in a dynamic stat diagram model.
- An Open framework for simple coupling with physics codes.

After running an EMRALD model, the user is able to not only obtain probabilistic results, but can also analyze the timing and ordering of events. Additionally, a coupling framework for physics codes, allows the user to incorporate and then determine when complex phenomena simulation, such as flood or fire analysis, is important to model accuracy.

Keywords: Dynamic PRA, Simulation, Dynamic PSA.

1. INTRODUCTION

Although there are many different scenarios where DPRA would be of benefit, most would be too complex or cost prohibitive given the current tools. First, traditional modelers generally do not possess the skillset needed to use many existing DPRA tools and thus would require extensive training. Second, converting existing models into dynamic models could be very time intensive. Third, the DPRA modeling effort would require an in-depth knowledge of the existing models and how to translate them from a static model to a dynamic one.

Current DPRA tools often use complex scripts for the models, lack a familiar graphical user interface, and/or have a completely different modeling mindset from traditional approaches. We demonstrate a modeling method that captures DPRA capabilities while incorporating basic aspects of traditional PRA.

EMRALD development and testing over the previous few years, was performed as part of the Light Water Reactor Sustainability (LWRS) project, analyzing time dependent scenarios and coupling with physics based simulation tools. One example was a multi-hazard scenario of seismic induced internal flooding [1]. Feasibility work for a graphical user interface (UI) and coupling communication was also initiated at that time. Because the capabilities of EMRALD are applicable to many domains and has commercial potential, a Technology Commercialization Fund (TCF) level 1 [2] was awarded to the EMRALD development team in order to bring the UI and the communication protocol to a beta ready level. The following sections outline the modeling techniques, interface tools, computational capabilities, and coupling communication protocol implemented by EMRALD.

* Steven.Prescott@inl.gov

2. DYNAMIC VS. TRADITIONAL MODELING

Although traditional PRA methods are widely used, and of great benefit, DPRA methods have been developed because of the limitations of the traditional PRA [3] and the increased availability of computation resources. Since a large effort has been expended to build current PRA models, transitioning to DPRA may be economical if extensive modelling rework is required. In order to make DPRA tools more accepted by industry, the transition to these new tools must be efficient and/or auto convert existing models. The following sections outline the different pieces used for DPRA modeling color in EMRALD, and demonstrate how traditional modeling methods are mimicked or can be easily translated.

2.1 Dynamic Components

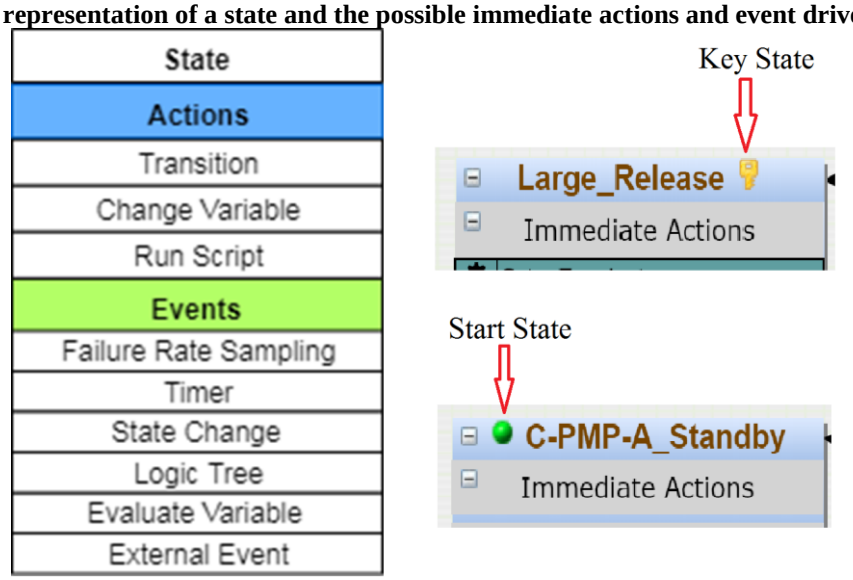
The following sections describe commonly used types for developing EMRALD models.

2.1.1 States

States are a logical representation for a current condition in a diagram as shown in Figure 1. Each state has or can have the following attributes:

- Type – Start, Standard, Key State, or Terminal, which establishes simulation behavior when setting up, running, or post-processing a simulation run.
- Immediate Actions – These actions are executed when entering the state.
- Event Actions – Items to monitor when in this state which can then trigger one or more actions.

Figure 1: Visual representation of a state and the possible immediate actions and event driven actions.



2.1.2 Actions

Actions change the properties or cause movement though a model during a simulation run. The following are the main types of actions, but with the expandable design of EMRALD, additional or customized actions can easily be added.

- Transition – Indicates a state or set of states that the simulation will process and be made current.
- Change Variable – Changes the value of a variable according to a user defined script.
- Execute Application – Runs user defined scripts to both execute an external piece of code and process the results to direct state changes.
- External Sim Message – Sends a message to an external code through coupled communication messaging protocol.

2.1.3 Events

Events monitor for specified criteria and can have one or more actions that are executed when that criteria is met. There are two categories of events; time based and conditional, with different types in each category.

Time Based:

- Timer – Monitors the current time against the user specified time.
- Failure Rate – Samples a given probability of failure to determine the time of this event.
- Distribution – Samples a distribution according to user specified parameters for the event time.

Conditional:

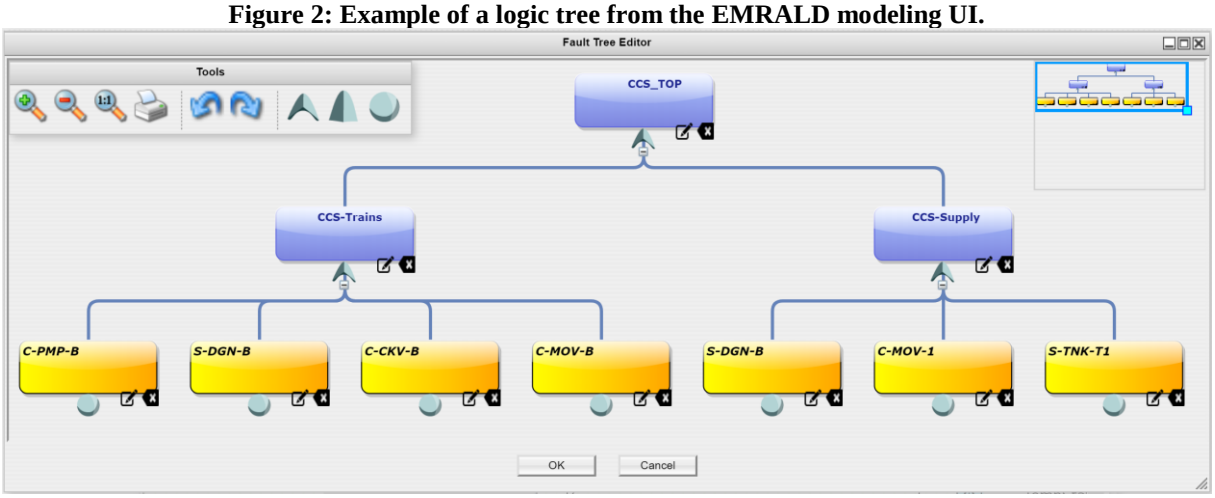
- State Change – Monitors when entering or exiting a specified state to determine event execution.
- Component Logic Tree – Evaluates a logic model of component diagrams to determine event execution.
- Variable Condition Event – Monitors variable values with user-defined code to determine event execution.
- External Sim Event – Links to an external simulation flag to determine event execution.

Actions executed by an event are state dependent, meaning that if the same event is in more than one state, the event can trigger different actions for each state. Multiple actions can also be assigned to a single event, and each of these actions is executed if the event occurs. Finally, each event in a state has a flag that indicates if this state is to be exited if the event occurs.

2.1.4 Logic Tree

Logic trees, as shown in Figure 2, use standard Boolean gates to map and evaluate the behavior of components. These trees use the assigned Boolean value for the state a component is currently in, to solve for the top value of the tree. Logic trees are linked to the event type "Component Logic Tree" listed in the previous section, and are evaluated whenever an associated component state changes. For more information on their use in a system model see section 2.2.2.

Figure 2: Example of a logic tree from the EMRALD modeling UI.



2.1.5 Variables

Variables define a value that can be evaluated or modified by user defied scripts in some events and actions. External variables are linked to an external code and can be modified or evaluated by either.

2.2 Diagram Modeling Equivalents

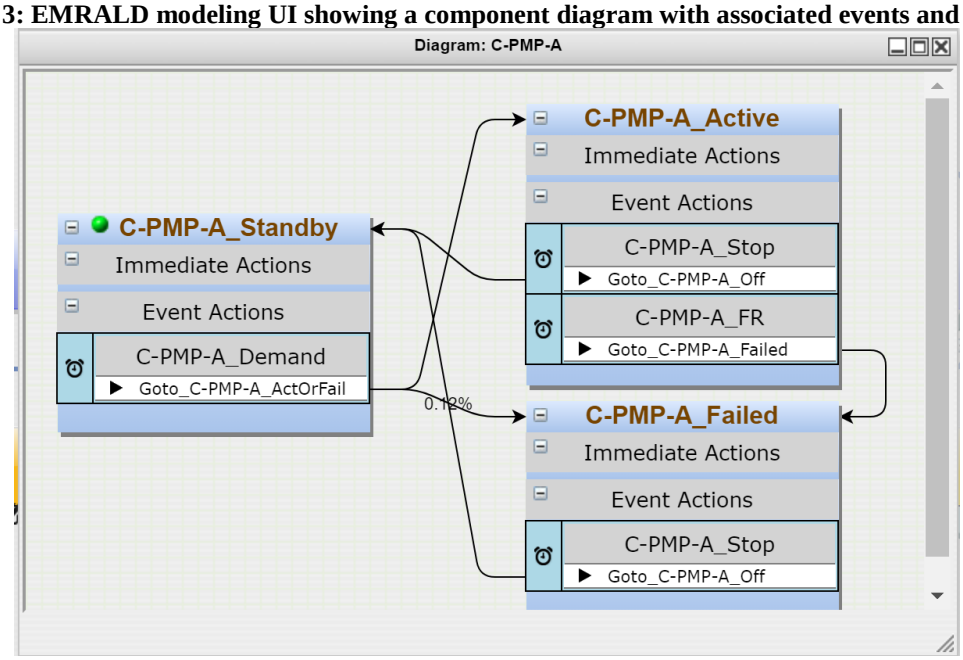
Like a traditional PRA, an EMRALD model consists of multiple diagrams. Each diagram represents a particular piece of the model and the various conditions or states that this piece of the model can be in. These pieces correlate to aspects of traditional PRA modeling, and range from small-scale components to a large scope of plant response and design. A diagram contains multiple states with the events that can occur and actions that may be executed. These all define how the current states of the simulation may shift over time.

Additionally, some diagrams (Component and State Diagrams) can also be evaluated for a Boolean result depending on which state they are currently in. This is a key feature that, when combined with a Component Logic Event, can greatly simplify a model. Unlike the more general diagrams such as plant response, these diagrams are restricted to only be in one state at a time during the evaluation process. For example, a pump cannot be both active and failed at the same time.

2.2.1 Component Diagrams and Basic Events

All aspects of a component can be captured in a component diagram as shown in Figure 3. Basic events in traditional modeling are the events that move the component between its different states such as standby, running, and failed. Component diagrams are "single state diagrams," meaning that only one of the states in the diagram can be current. Each state of the diagram also has a Boolean value that is used if the component is evaluated in logic trees. Only component diagrams can be used as leaf nodes inside a logic tree.

Figure 3: EMRALD modeling UI showing a component diagram with associated events and actions.



2.2.2 System Diagrams and Fault Trees

A System diagram is typically a simple diagram consisting of two states; active and failed. A logic tree built of gates and leaf nodes reference components. This tree is evaluated whenever a child component diagram changes and determines any movement from "active" to "failed" as shown in Figure 4. This diagram is also a single state diagram.

2.2.3 Plant State Diagrams and Event Trees

Plant response diagrams are the main scenarios to be evaluated. They are similar to event trees and are typically the largest diagrams. These diagrams must have a starting state, terminating state, key states, and other plant states, mapping the conditions and events of the scenario. States that represent critical outcomes must be marked as key states. When each simulation ends, all key states that are in the current state list are logged with the timing and paths of all the states and events that lead to this state.

Figure 4: A system diagram shown on the top right evaluates a logic tree.

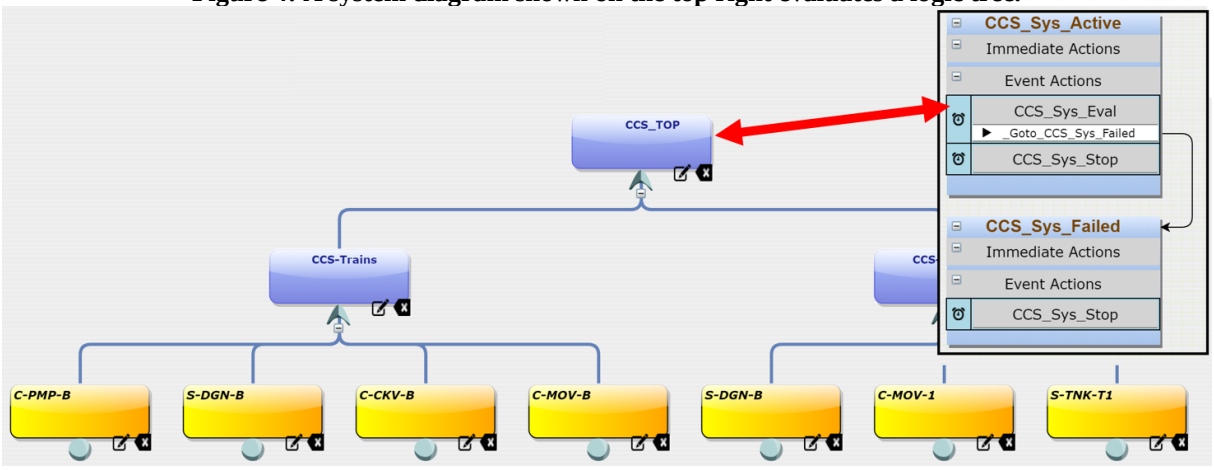
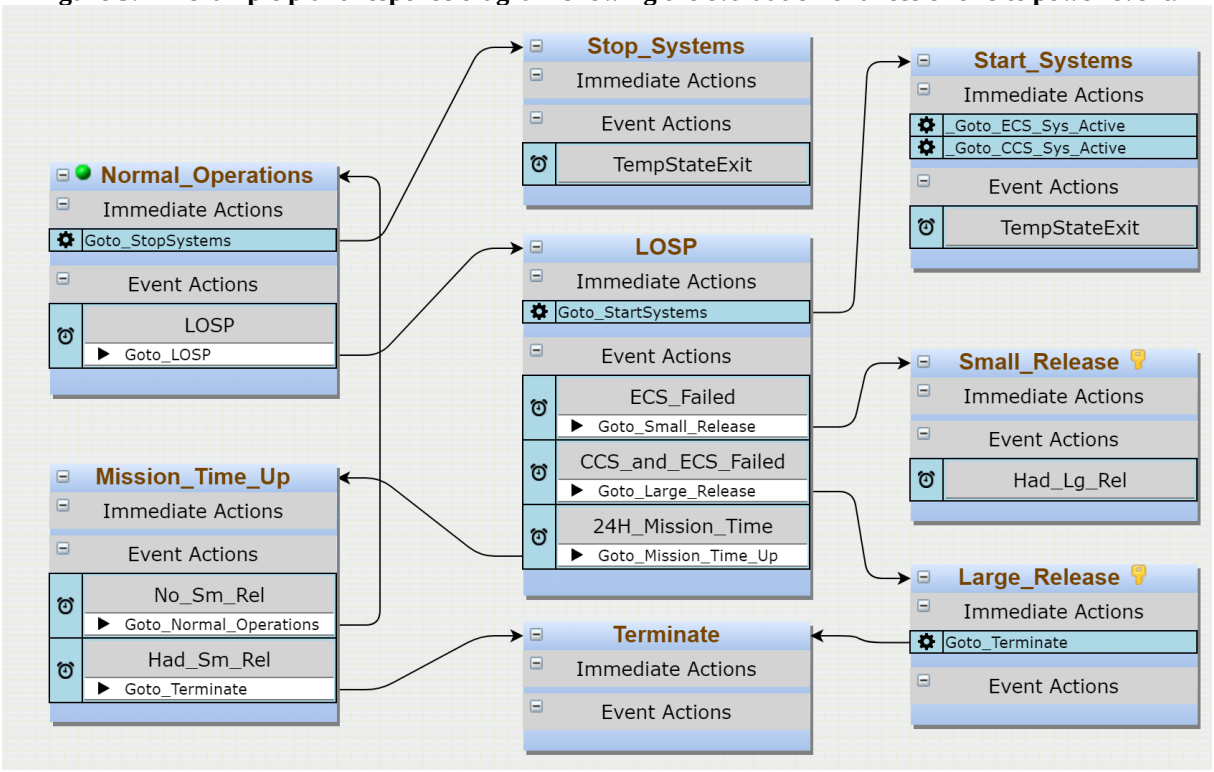


Figure 5: An example plant response diagram showing the evaluation of a loss of offsite power event.



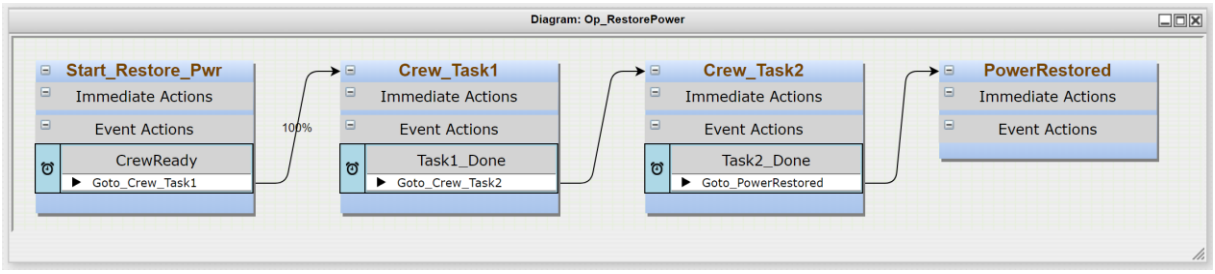
2.3 Dynamic Capabilities

Although DPRA can perform many of the same capabilities as traditional PRA, because of higher computation costs, it is not suited for this. However, it is ideal to evaluate several different dynamic conditions not available through traditional methods.

2.3.1 Operator Actions

Operator actions are a critical area for risk mitigation for a variety of scenarios. DPRA provides a better way to evaluate sequential procedures, by being able to credit excess time from one step of the procedure for use in a different step. Additionally, the probability for success can be adjusted dynamically depending on condition of the working area, without the need for binning.

Figure 6: Example of stepped procedures using events with distribution curves to sample times for each task.



2.3.2 Feedback Loops

Some scenarios can cause feedback loops where one condition affects a system, which then changes the condition it is monitoring. For example a pump may depend on a temperature, but activating the pump in turn reduces that temperature. The time of failure for either the sensor or pump in this condition could be critical to the outcome of the scenario. This conditional response and feedback is captured in an EMRALD model by using an event that monitors a temperature variable and an action that adjusts the same variable.

3. MODELING USER INTERFACE

The new user interface for constructing an EMRALD model uses web based technologies and is accessed using a browser. The interface is designed as a single page application, which makes the interaction with the user similar to desktop applications. The following sections describe the main types of windows used for editing the model.

3.1 Main Layout

In the main layout of the UI lists all of the modeling pieces on the left side. These items can be drag and dropped into other modeling windows where applicable. The right side is the modeling area where diagrams and edit windows are displayed, as shown in Figure 7.

3.2 Properties Editor

Each item in the left side list has a right click option to edit its properties. The properties editor allows the user to enter or modify standard items such as the name, description, and type, then custom information for each item and the type selected. For example, an event item of type “state change” lets the user edit the list of states it is monitoring, and the conditions for triggering it. This is shown in the event editor window in Figure 7.

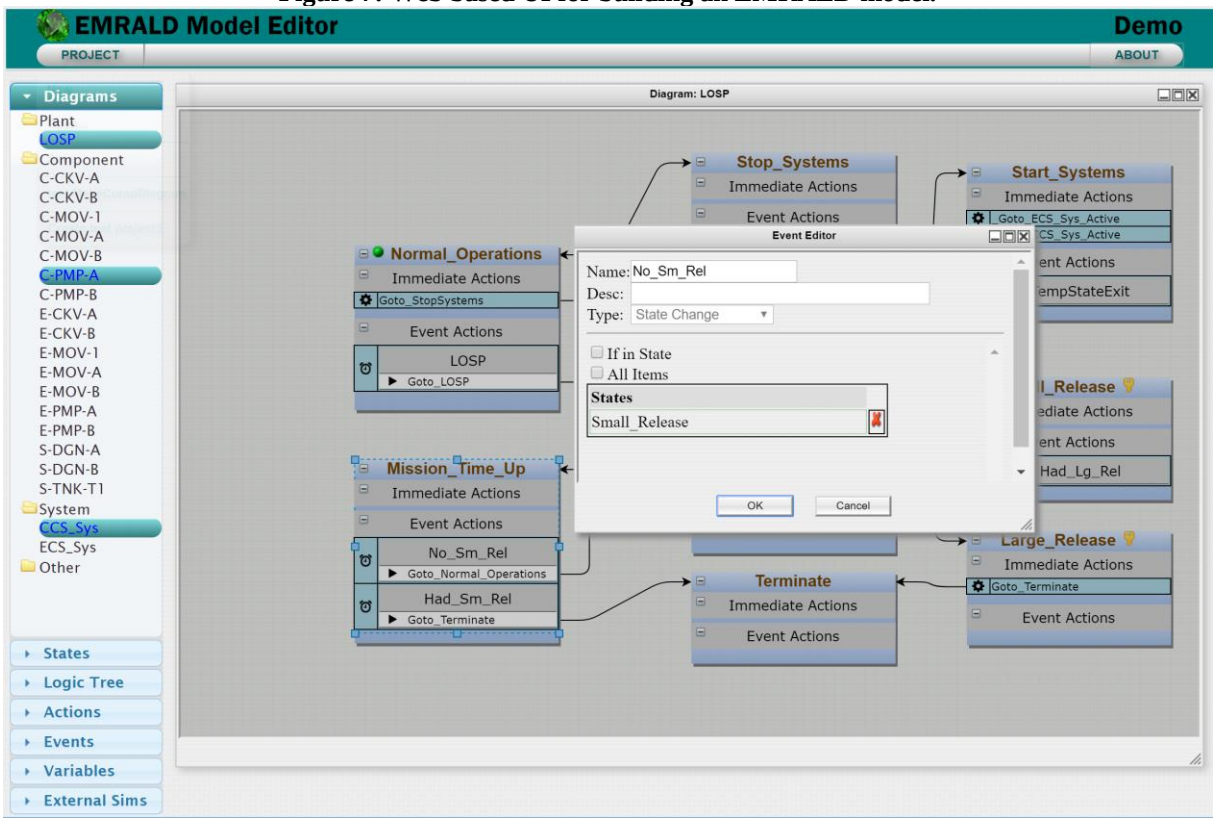
3.3 Diagram Editor

The diagram editor displays the states and transitions between those states, as shown in Figure 5, for different types of diagrams, “plant response,” “component,” and “system” diagrams, shown on the left side of Figure 7. Here the user can make new diagrams by adding new states, events, and actions or edit diagrams by selecting those pieces. Editing is through standard drag and drop capabilities, double clicking, or right click options.

3.4 Logic Editor

Users construct a logic tree in the logic editor by adding gates and nodes. Each leaf node is a reference to a component diagram and uses its current value when being evaluated which can be added by dragging and dropping diagram names from the left side list. The UI is shown in Figure 2.

Figure 7: Web based UI for building an EMRALD model.



3.5 Model Format

The data used during the modeling process is temporary and all long term storage of an EMRALD model is dependent upon the user to save the model to their local computer or a network storage device. The model is saved as a JSON text file. This format allows for readability and direct modification for advanced users or possible automated script modification.

3.6 Software Technology

The following commonly used web technologies and languages were used to develop EMRALD's modeling UI.

Languages:

- JavaScript – The main development language; all of diagramming user interface are generated in code.
- HTML5 – Hypertext Markup Language (HTML) is used primarily for basic form layout to be custom filled by JavaScript procedures.
- CSS3 – There are styling files for menu and navigation. All other visual user interfaces' styling are hard-coded using JavaScript code.

Libraries: These are libraries used and are open source.

- mxGraph – mxGraph is a fully client side JavaScript diagramming library that uses Scalable Vector Graphics (SVG) and HTML for rendering [4]. The software makes heavy use of this library.
- jQuery – A library to assist in querying an HTML document traversal and manipulation, event handling, animation, and Ajax, making it much simpler to develop an easy-to-use application program interface (API) that works across a multitude of browsers [5]. This software is minimally used, but it is required.
- jQuery-ui – jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library [6].

- AngularJS – This library handles data transport between the model view and the presentation views, strictly for the client user interface data storage [7]. All editor forms use this library for temporary data storage on the browser.

4. SOLVE ENGINE

In order to make EMRALD's solve engine more versatile, a decoupled approach was taken. This means it can be installed on a local computer, or customized and set up on a server for multi-user as a fully web based simulation tool, including support for high-performance computing.

4.1 Discrete Event Simulation

To get DPRA results for an EMRALD model, a large number of simulations can be run (thousands to millions of times), logging any results ending with key states. Running an EMRALD model uses a form of three phase discrete event simulation, as shown in Figure 8.

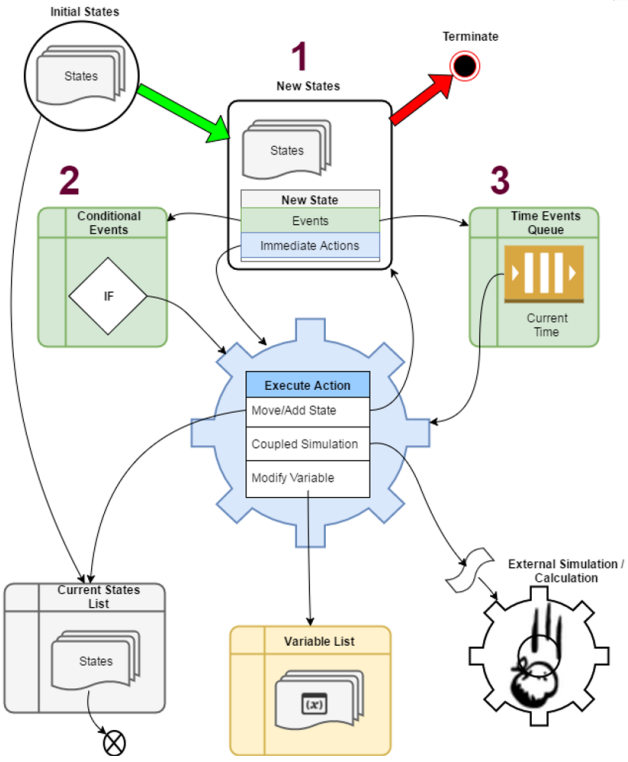
Figure 8: Flow process for EMRALD's three phase discrete event simulation

Upon loading, initial start states are added to the Current and New States list.

1. While there are States in the New Sates list, For each State :
   - Add the Events to the Time Queue or Conditional List.
   - Execute any Immediate Actions.

2. If any Conditional Events criteria is met.
   - Execute that events action/s.
   - (Go to Step 1).

3. Jump to the next chronological event.
   - Process that event's actions.
   - (Go to Step 1).



4.2 User Interface

The solve engine has a basic user interface that allows the user to setup and test a model, however, it can also be executed from the command line for batch or server processes. Opening a model shows the JSON text for the model in the first tab area, verifies the model, shows any errors, and even allows for modifications by advanced users. The second tab area in Figure 9 shows needed external references, variables to monitor, parameters for running the simulations, and viewing of the solving progress. The other tabs, as shown in Figure 10, are for debugging and allow the user to monitor and send messages to connected external codes.

4.3 Simulation Results

For each simulation run, if it finishes in any "key" state, then that occurrence along with the failed components, events, and times are logged. A simplified set of results shown on the bottom of Figure 9, lists standard failure rates and components for these key state, similar to traditional cut sets. These results can be further processed to look for patterns or clustering of contributors over time, as shown in Figure 11, allowing users to develop mitigations methods for time critical areas.

Figure 9: EMRALD's solve engine UI, showing the parameters for solving the loaded model and results.

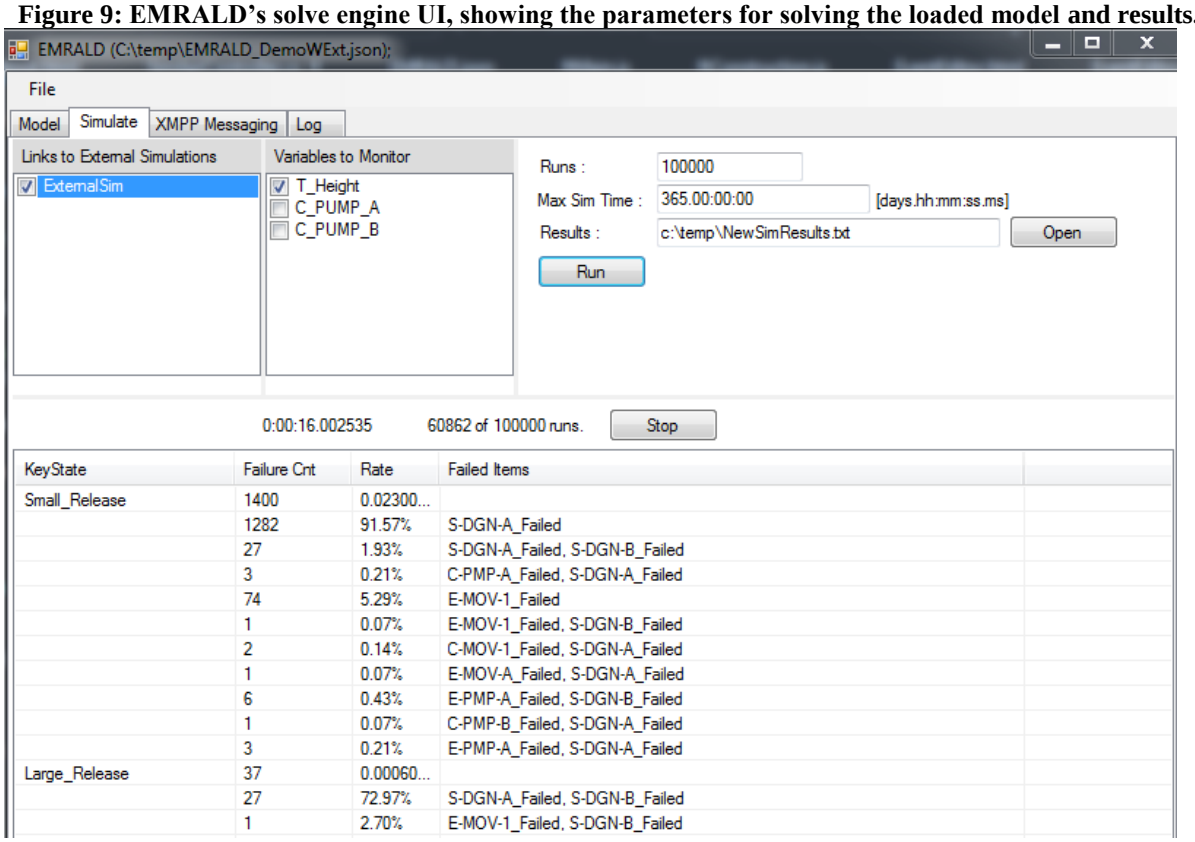


Figure 10: Tab showing connected codes with message monitoring and testing tools.

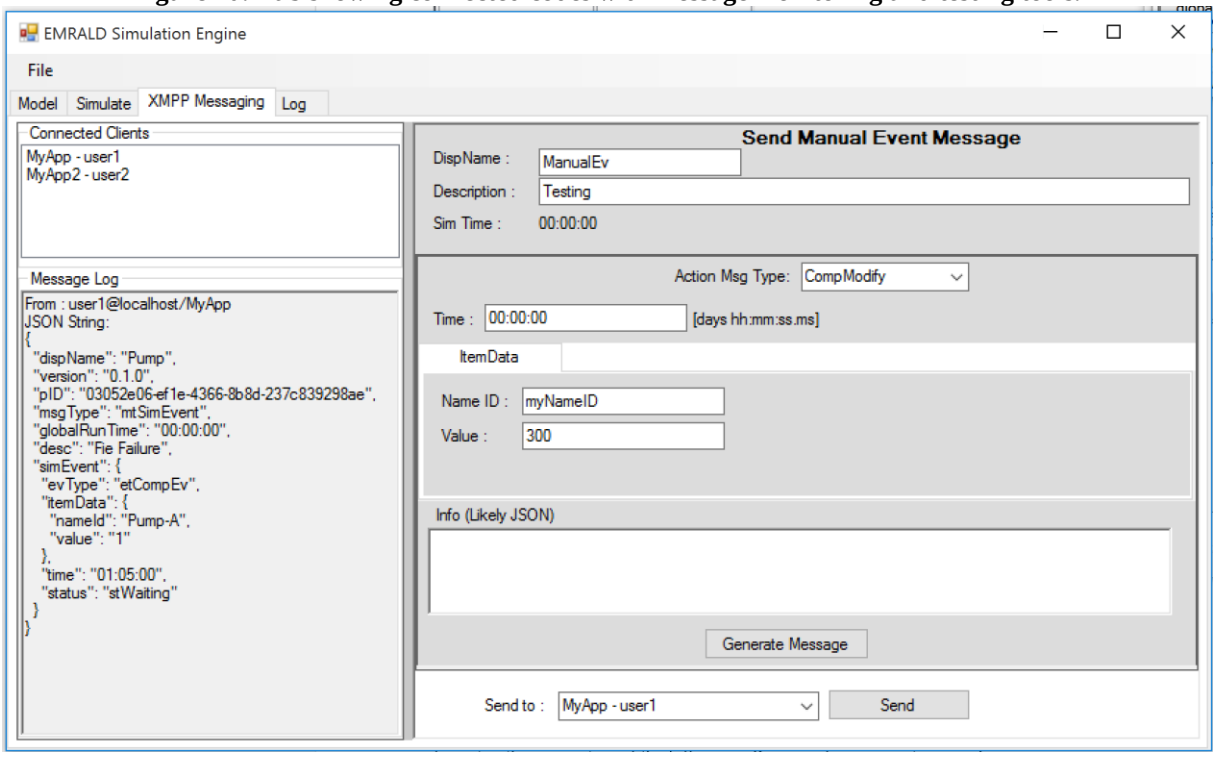
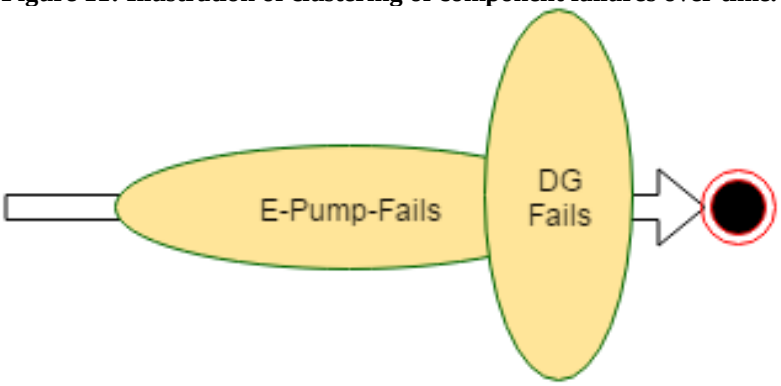


Figure 11: Illustration of clustering of component failures over time.



5. EXTERNAL COUPLING

A key feature of EMRALD is the ability to easily couple with other applications. Both one way and loose two way coupling is available. Two way coupling makes it possible for external codes to affect the EMRALD simulation and in turn, gives EMRALD the ability to adjust the external code during run time.

5.1 Setup

One way coupling is done using a specialized "action" which runs a user script and starts the specified code. When the execution is complete, a second user defied script can process the results and assign state transitions, probabilities and other parameters. Two way coupling requires that an API or direct source code be available, but implementation is relatively simple given the use of open source tools available and discussed in sections 5.2 and 5.3

5.2 XMPP Library

All messaging capabilities are handled using the open source XMPP messaging standards [8]. There are numerous packages available for XMPP, so it can be used by most programming languages. EMRALD has a XMPP server built into the solve engine, so no additional server setup is necessary. Once the XMPP client package has been added to the source code or in a wrapper, a connection is made by providing the EMRALD domain information, the client's user name, and resource type. An example wrapper application written in the C# programming language, shown in Figure 12, provides a template and an example of a package to use, along with basic debugging tools.

5.3 Message Protocol

Messaging tools such as XMPP do not restrict the configuration or data format contained in the messages. In order to standardize data between EMRALD and external codes, an opening messaging standard was developed and may be used by other event/action applications. This standard requires all data being sent to use a JSON format. JSON was chosen for multiple reasons. It is compatible with most programming languages, easily expandable, has a high level of human readability, and a schema can be used to validate messages. These JSON messages can typically be accessed or easily converted to language objects through available JSON software packages, making it easy to construct or evaluate messages and execute appropriate actions.

Each message has a standard set of data including name, description, version, id, global time, and type. Additional data depends on the type and subtype of the message. An example of a message is shown in Figure 13. Messages are of two main types; "actions," and "events." Actions are typically sent by master application, normally EMRALD, in order to adjust the external simulation through actions such as start, stop, call-back times, or modify parameters. Events are things that occur in the external simulation that affect the EMRALD model, such as pressure on a component reaching a set level.

Figure 12: Sample XMPP wrapper with UI for message generation and testing.

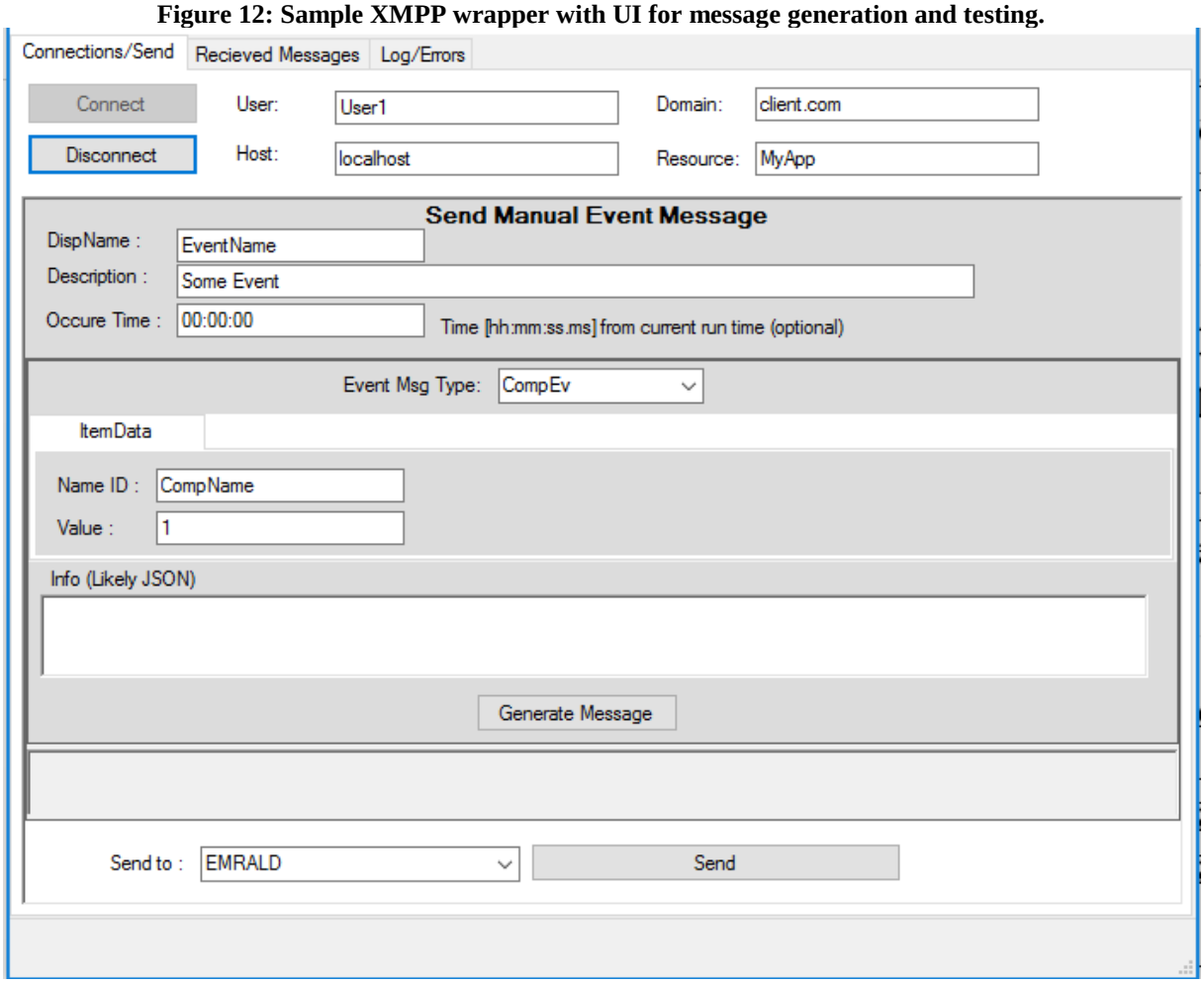


Figure 13: Example JSON message from an external code to indicate that a pump has failed.


```
{
  "dispName": "PumpFail",
  "version": "0.1.0",
  "pID": "73842e5d-132e-4d40-a8b5-be777b3e463d",
  "msgType": "mtSimEvent",
  "globalRunTime": "00:00:00",
  "desc": "Fire Failed Pump",
  "simEvent": {
    "evType": "etCompEv",
    "itemData": {
      "nameId": "PUMP-A",
      "value": "1"
    },
    "time": "00:42:00",
    "status": "stWaiting"
  }
}
```


6. EXAMPLE RESULTS

EMRALD was used to perform DPRA for an initiating event of seismically induced flooding of a nuclear power plant switch gear room. In this example seismic hazard curves and dynamic failure rates

were used in combination with flooding simulation and thermal hydraulic results using EMRALD. Details on this example were given at BEPU 2018 [9].

Several insights were observed from these results. First a ranking of the components most likely to fail from internal flooding was easily determined. Second, Figure 14 results show some large conservatisms between traditional modeling and DPRA with physics simulation, in some sequences.

Figure 14: RISMIC IA2 results showing the increase in core damage frequency between traditional SAPHIRE modeling and EMRALD with physics simulation.

Sequence Case (Bin2+ Bin3)	CDF No SWGR Pipe Failure [SAPHIRE]	CDF SWGR Pipe Failure [SAPHIRE] (S1)	CDF SWGR Pipe Failure [EMRALD+NEUTRINO+RELAP5-3D/RAVEN] (E1)	CDF Reduction EMRALD vs. SAPHIRE (1-E1/S1)*100
LOOP 2-02-05	5.65E-07	6.64E-07	5.74E-07	-14%
LOOP 2-15	1.94E-06	1.03E-05	1.94E-06	-81%
SBO 2-16-03-10	2.09E-06	3.85E-06	2.09E-06	-46%
SBO 2-16-45	9.68E-06	2.25E-05	5.74E-07	-97%

6. CONCLUSION

EMRALD has been designed to make the DPRA approach accessible to a large community of risk researchers and practitioners by:

- Following traditional modelling methods aligned with traditional PRA practices
- Providing an intuitive graphical UI that is modeled upon an existing PRA tool (SAPHIRE)
- Simplifying the approach to coupling simulation and physics calculations by using commonly accepted communication methods.

It is believed that by following these design principles, we will be able to reduce modeling and analysis efforts for advanced risk analysis methods, making it more attractive to the next-generation of risk and reliability analysts found in a variety of industries. Current documentation and info for the future beta release is provided at www.EMRALD.inl.gov. We anticipate a beta release of EMRALD's modeling UI and solve engine for review by the end of Sept 2018.

References

[1] C. Parisi, S. Prescott, J. Coleman, Z. Ma, B. Spears, B Kosbab, R Szilard, "Risk-Informed External Hazards Analysis for Seismic and Flooding Phenomena for a Generic PWR," INL/EXT1742666, (2017)

[2] Office of Technology Transitions, "Technology Commercialization Fund Fiscal Year 2018 Solicitation," https://proposals.inl.gov/Home/PublicUploads/TCF%20FY18%20Solicitation%20Webinar%20Slides.pdf, (2017)

[3] A. Mosleh. "*PRA: A Perspective on Strengths Current Limitations, and Possible Improvements,*" Nuclear Engineering and Technology, pp. 1-10, (2014).

[4] MxGraph, https://github.com/jgraph/mxgraph, (2018).

[5] jQuery, https://jquery.com, (2018).

[6] jQuery-ui, https://jqueryui.com, (2018).

[7] angularJS, https://angularjs.org, (2018).

[8] XMPP, https://xmpp.org/about/, (2018).

[9] C. Parisi, S. Prescott, Z. Ma, J. Coleman, R. Szilard, C. Smith, "RISK-INFORMED EXTERNAL HAZARDS ANALYSIS BY EEVE TOOLKIT," BEPU 2018, Real Collegio, Lucca, Italy, May 13-19, 2018