Using EMRALD to Simplify and Perform Dynamic Analysis with MAAP

Steven Prescott[a], Daniel Nevius[b], Zhegang Ma[d], and Svetlana Lawrence[e]
[a] Idaho National Laboratory, Idaho Falls, ID, USA, steven.prescott@inl.gov
[b] Idaho National Laboratory, Idaho Falls, ID, USA, daniel.nevius@inl.gov
[c] Idaho National Laboratory, Idaho Falls, ID, USA, zhegang.ma@inl.gov
[d] Idaho National Laboratory, Idaho Falls, ID, USA, svetlana.lawrence@inl.gov

Abstract: Event Modeling Risk Assessment using Linked Diagrams (EMRALD) is a software tool developed at Idaho National Laboratory (INL) for researching the capabilities of dynamic probabilistic risk assessment (PRA). It provides a simple interface to represent complex interactions often seen when developing dynamic models. EMRALD can also interface with other applications by modifying inputs, running, and using their results within EMRALD for dynamic and integrated assessment. Linking with external codes was formerly done by user-defined scripts, requiring users to be familiar with both the scripting and the details of the application they wanted to use. A recent, enhanced feature to EMRALD provides a library capability to add custom forms for developing simple interfaces to run specific applications, making it so the user does not need to write scripts or have extensive training in the tool from which they want to pull data. This feature is especially useful for thermal hydraulic applications, such as the Modular Accident Analysis Program (MAAP), in which after the plant model is developed, an in-house MAAP expert is still needed to make any modifications for an analysis. This report outlines the custom form process used to develop an EMRALD user interface for MAAP and demonstrates how it can be used to enable an analyst with minimal MAAP-specific expertise to easily perform a custom thermal hydraulic analysis with an existing MAAP general model.

1 INTRODUCTION

A critical part of dynamic PRA is the ability to integrate and use data from other tools. To assist in this, recent updates to the open-source Idaho National Laboratory (INL)-developed software, Event Modeling Risk Assessment using Linked Diagrams (EMRALD) [1] were made, include a process to simplify the one-way coupling of other simulation tools through a custom user-interface. This can allow a user to assign and open another application's model and directly assign how features are affected by the EMRALD simulation to then run the dynamically adjusted model from EMRALD.

Many simulation scenarios for the nuclear power industry would benefit from including thermal dynamics analysis. However, understanding and modifying these codes can take an expert in the field and could be prone to user errors. This research demonstrates both the benefit of coupling dynamic probabilistic risk assessment (PRA) with Modular Accident Analysis Program (MAAP) thermal dynamics code and how an EMRALD custom user interface makes it simple for beginning users to perform some analyses.

2 DYNAMIC PRA AND THERMAL HYDRAULICS

The reactor safety is the foundational basis for the power production using nuclear energy. Thermal hydraulics (TH) analyses help to ensure that the public health and environment remain protected during the operation of commercial nuclear power plants (NPPs).

2.1 Thermal Hydraulics and Reactor Risk Assessments

The TH analyses provide vital information that serves as the basis in many areas of NPP safety and risk assessments—from the identification of undesirable plant conditions that could disrupt normal plant operations (initiating events), planning strategies to prevent or mitigate initiating events, and ending

with post-accident consequence analyses. Each phase of a plant risk assessment requires inputs from the TH analyses, so their importance cannot be overstated.

However, TH analyses are complicated and require the involvement of subject matter experts (SMEs) with a significant level of knowledge and experience. An analyst embarking on TH computations should be knowledgeable in both physics of TH processes and in computer software performing the analyses. The complexity of the task and limited pool of SMEs often necessitates a choice of performing bounding TH analyses where simulation scenarios are grouped together, and most limiting physical conditions are postulated. This approach provides a bounding case for many scenarios, but it is usually also overly conservative since most limiting conditions are postulated for TH analyses, and then the most limiting outcomes are used for risk assessments.

An example of the TH analysis outcome is time to core damage, which is assumed if the limiting temperature or pressure conditions are reached. For example, core damage is assumed to occur if the instantaneous fuel temperature reaches 2,200°F or a 10-min. sustained fuel temperature reaches 1,800°F. The boundary conditions for a given TH analysis can be very different. For example, a system state initiates an automatic reactor trip, which is usually an entry condition to an emergency operating procedure (EOP). Multiple triggers can initiate a reactor trip (e.g., abnormally lowering or raising pressure, temperature, water level, etc.). The time between reactor trip and core damage is important since this is the governing parameter for assessment of operator actions associated with mitigation of the postulated event. In the case of a bounded TH analysis, usually the most conservative boundary condition is assumed; in this case, system state that would trigger a reactor trip at the latest time, which minimizes the time available for operator actions. Multiple other system state conditions are postulated, also typically bounding to ensure that the TH analysis outcome is applicable to a wide range of scenarios. While this practice of bounding TH analyses is acceptable since it helps to reduce the complexity of risk assessments, it often results in overly conservative parameters that are used as the success criteria and boundary conditions in the overall plant risk assessment.

2.2 Static vs Dynamic TH Analyses

Other deficiency with the current TH analyses is their static nature. The typical workflow is to: (1) identify boundary conditions for a given scenario (e.g., full power operational stage); (2) postulate certain system states to reflect the success or failure of subsystems (e.g., alternating current [AC] power is not available, feedwater sources are lost, etc.); and (3) perform the TH analyses and produce output results (e.g., time to core damage). Multiple opportunities exist to change the trajectory of a scenario (e.g., operators manually start equipment that failed to start automatically), yet these interventions are not accounted for in the actual TH analyses. The scenario alteration is typically represented by a PRA event tree, and subsequently, a fault tree.

Imagine a small break loss of coolant accident (SBLOCA) scenario in a pressurized water reactor (PWR) as an example. An SBLOCA is an event where a small break occurs in the reactor cooling system (RCS) piping. The break size in this scenario is too small to depressurize the RCS to the point where the safety injection system can be actuated to replenish RCS inventory lost through the break. Operator actions are required to perform RCS depressurization, which is normally performed via the secondary system (i.e., RCS heat is removed via steam generators). In the event when RCS cooldown via a secondary system in not available, depressurization can be done via once-through-cooling where an RCS vent valve is opened to lower the pressure. The operator action to initiate once-through-cooling is a critical point in the scenario progression and its success or failure is included in the PRA model. The probability of operator action failure is calculated based on the time available to perform the action, which is derived from the TH analysis. For example, a TH analysis would show that if the operator action to initiate once-through-cooling is performed approximately 60 minutes after the reactor trip, core damage would not occur. The 60-minute parameter becomes the input into the human reliability analysis (HRA) to determine the probability of success of the operators to perform the action within the available time, given other conditions (e.g., stress, workload, adequacy of the procedures).

While the above approach has been successfully used to provide needed inputs into the plant PRA model, it is certainly not ideal. The scenario progression is evaluated piece-by-piece with assumptions required to connect all the pieces together. Instead, a dynamic simulation of an accident progression would provide a much more comprehensive picture. In the dynamic TH analysis, an operator could perform an action at various times during the scenario progression, which would alter the scenario trajectory and provide output in ready terms of success or failure with a clear indication of factors that contributed to the failure. Operator actions depend on the context in which they are performed (e.g., situational awareness, stress level, clarity of the instructions, etc.). These contributing factors ideally should be informed by the overall plant state at the moment when action is expected, the preceding plant states, and actions taken. However, this is not possible in the static scenario assessment and the context for operator actions postulated as a boundary condition in HRA.

The dynamic simulation can also demonstrate multiple 'what-if' possibilities to alter the scenario. For the SBLOCA example above, the situation of a secondary system not available for the RCS depressurization is typically caused by the loss of all sources of feedwater, which can be caused by the equipment failing to start. In this case, the operators usually have the means to manually start the system, but the PRA does not account for such a possibility. A dynamic simulation could allow this scenario trajectory.

Given the complexity of a typical plant PRA model, it is not feasible or practical to transition to a fully dynamic-based model. However, it will be beneficial for safety-significant scenarios to perform simulations dynamically. This would allow a much better understanding of the progression of critical scenarios, demonstrate opportunities for mitigation, and discover the trajectories that possibly were not accounted for in a typical static analysis.

3 EMRALD USER INTERFACE FOR DYNAMIC PRA

EMRALD is an open-source dynamic PRA tool based on three-phase discrete event analysis. The goals of EMRALD are to simplify the construction of dynamic modeling by using a graph-based interface with structuring and features that are similar to existing static PRA methods [2]. EMRALD consists of two main parts, the web-based modeling tool and a solve engine application, which is typically run on a desktop or via the command line on a server. This section provides details on the main features of the modeling interface. For more information on how to build EMRALD models, see reference [3]. For specifics, see the documentation pages in reference [4].

3.1 Graph-based Diagram Modeling

The dynamic simulation in EMRALD is achieved by starting off in a set of states, and then events associated with that state are sampled or monitored, which drive different actions. These actions cause changes, such as new values for variables or shifting to new states. This is similar to how things occur in a real environment: everything is in a particular state until something happens to change it. EMRALD uses a node base structure to represent these interactions and how things progress through time. EMRALD uses the web-based MXGraph [5] to display the visualization for these diagrams. Figure 1 provides an example of an EMRALD diagram.

This web interface also provides modern drag and drop options for modeling. Items such as Events, Actions, or States can be selected from the left side of the interface and dropped into the appropriate areas or fields in forms. The Logic Tree modeling follows a similar process, allowing the user to construct a boolean logic evaluation by drag and drop of component diagrams.

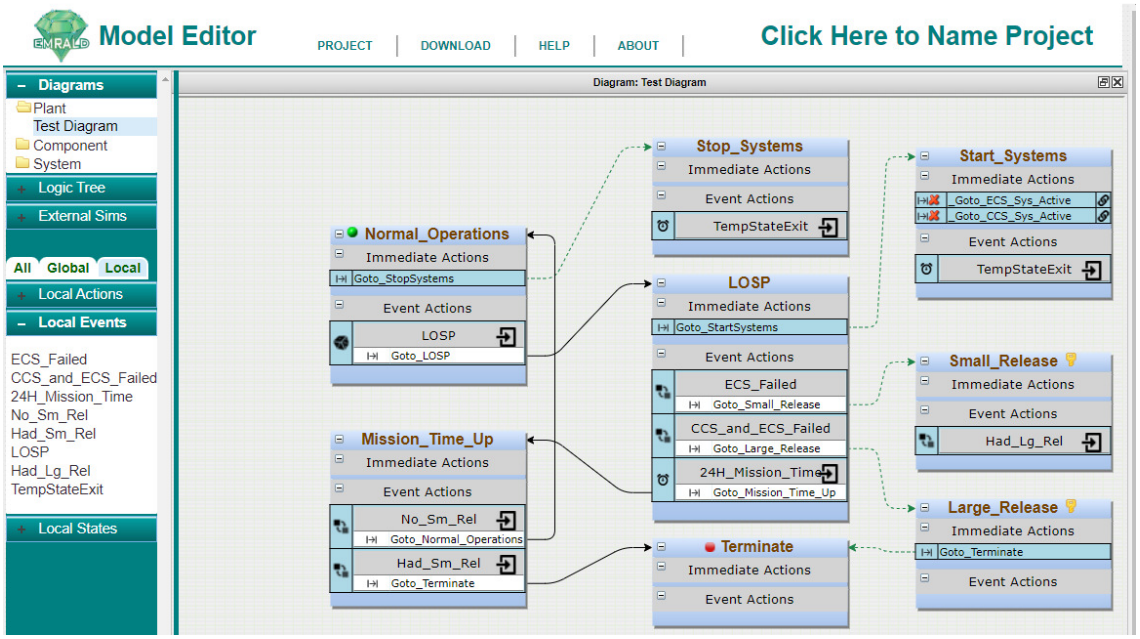

Figure 1. Example of an EMRALD diagram.

3.2 Coupling with Other Applications

One of the key use cases in the development of EMRALD was the ability to modify and run other calculations or simulation codes that then use those results to drive the EMRALD model. Two coupling methods were developed: (1) one-way coupling that could attach to any application with command line running options; and (2) two-way coupling through an Extensible Messaging and Presence Protocol (XMPP). The one-way coupling is more widely used and is what was expanded to develop the custom forms.

Originally, to do one-way coupling, an action of type 'Run Application' is used, as shown in Figure 2. This action has three main areas for the user to define. First, a pre-processing script is defined by the user, which has access to the different variables, current states, and simulation times. This data can be used to modify an input file or define execution parameters. Whatever is returned by this script passes as a parameter to the executable. The second input is the path to the executable to be run. Finally, the user also defines what to do when the process is done running in the post-processing code. In this mode, the user can read a results file or return value and then enter or exit the states according to the values.

While this works well, it also creates one of the more difficult learning curves for EMRALD, can also take the most time in model development, and is difficult to debug. The user must learn how to write C# scripts, and then test and debug the scripts outside of EMRALD.

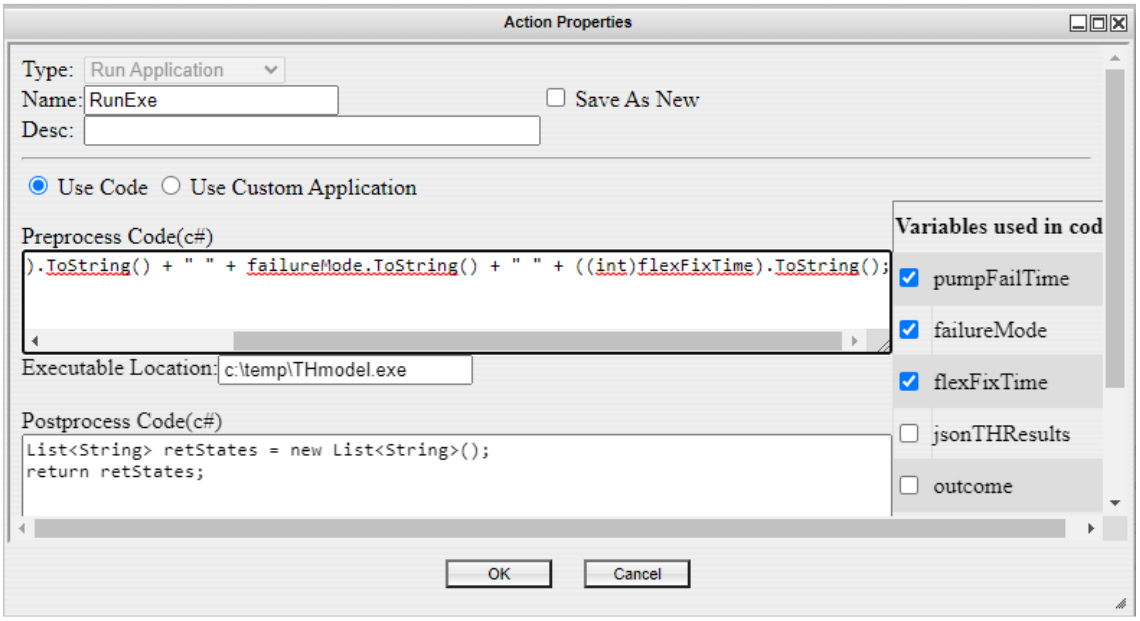

Figure 2. Run external application form.

4 EMRALD MAAP CUSTOM FORM

4.1 EMRALD Custom Forms

To reduce this learning curve, the ability to create custom forms for external applications in EMRALD was added. Users interacting with these forms need no programming experience and can simply interact with a graphical interface that generates the underlying code automatically. EMRALD provides a JavaScript (JS) library to simplify the creation of these forms by automatically integrating with EMRALD and providing methods for developers that make manipulating data within the EMRALD project much easier. This library was used to create a form within EMRALD for interacting with MAAP and the MAAP files.

Creating a form with the library begins with creating an HTML file for the interface and a JS file to interact with the library. EMRALD is built using AngularJS, which is expected to be used to build forms as well, but is not strictly required. The library exposes the 'CustomForm' class that each form must extend with their own respective class. The main output of a custom form is an object containing all the data necessary to run the 'Run Application' action, which is the return value of the form class's 'getDataObject' function. This object must specify the location of the executable file, the names of all variables used in the code, the pre-code to set up the external application, and the post-code to process the output of the application. The construction of this code is the responsibility of the form, but the library provides methods for extracting any required data out of the EMRALD project, as well as saving the form state within the project with a single call of the save function of the class.

4.2 User Interface – Loading Base Model

The initial MAAP custom form was designed to allow the user to easily see and ensure that the Parameters, Initiators, and modification Input Blocks are in the default input file. Secondly, the form allows the ability to search and add new initiators from the main model, which is done by parsing both the parameter and input file. The MAAP form also has a header section that prompts the user for basic information that will be required for displaying the rest of the form, which includes the path to the MAAP executable file, a file upload dialog for both the input and parameter files, and text inputs for the locations of the input and parameter files on the user's local hard drive. The uploaded files are parsed and used to display the other sections of the form. The absolute paths to the files allow the generated C# code to splice in the user's modifications to the original file without having to store the entire text

of the file within the EMRALD project, which also is required as security restrictions prevent web applications from determining the path of the uploaded files. The rest of the form is organized into four tabbed displays on the left side of the screen: 'Parameters,' 'Initiators,' 'Input Blocks,' and 'Outputs.'

When the parameter file is uploaded to the form, it is split into individual lines, where it then identifies the parameters according to the syntax. A custom text parser is used to parse these lines and create JS objects containing the parameter's index, description, and true/false value; each of these objects is stored in a dictionary. When the input file is uploaded to the form, the file scanner again iterates over each line in the file while keeping track of which section of the input file is currently processing. These sections are parsed to fill the data for the 'Parameters,' 'Initiators,' and 'Input Block' tabs.

4.3 User Interface – Parameters

The first tab shows the current parameters. The first column displays the name of the parameter being modified, the 'Value' column contains the value being assigned, and the 'Use Variable' column contains check boxes, which, when checked, allow the user to replace any of the values being assigned with the value of a variable from their EMRALD project, as shown in Figure 3. The initial parameters are pulled from the first section of the input file starting from the 'PARAMETER CHANGE' marker. Once it has found this line, each incoming line is parsed to create a JS object containing the name of the variable, the line it is assigning, as well as the value being assigned to it. This object is used to display and save values to construct a modified input file.

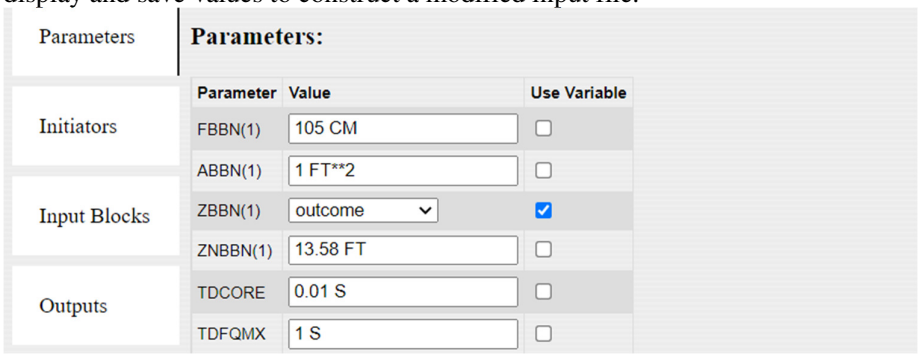

Figure 3. Example user interface to edit or assign variables to parameters read from the MAAP input file.

4.4 User Interface - Initiators

The initiators are the scenarios with which the MAAP analysis needs to start. The scenario file may contain a set of these and are read by looking for the "INITIATORS" section. Each line after the marker is parsed and can be an initiator and value. The initiators are displayed in the "Initiators" tab of the form, as shown in Figure 4. At the top of the tab is an input box where the user can search for any parameter specified in the parameter file and add it to the initiators section of the input file. Below the input box, the current list of initiators is displayed in a table, where the user can also delete any initiators currently in the file.

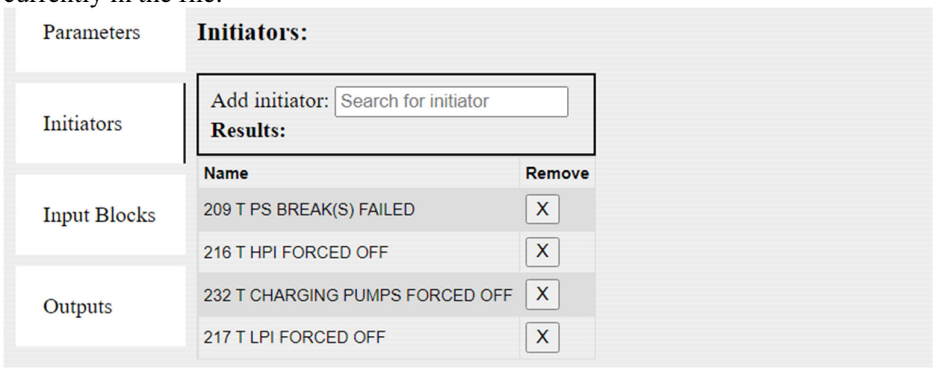

Figure 4. Example of the 'Initiators' tab showing the current initiators from the input file, as well as the option to search and add new ones found in the main model.

4.5 User Interface - Condition Blocks

The last section of items pulled from the input file are the condition blocks that make changes when the conditions are met. This last section is parsed to populate the 'Input Blocks' tab, as shown in Figure 5. The form currently supports editing 'IF' and 'WHEN' statement blocks in this section. 'IF' statements are identified by any line beginning with 'IF.' The line is split by each occurrence of 'AND' or 'OR,' and each set of expressions between these operators is parsed for display within the form. The sections in this tab contain a table of conditions and a table of outcomes. In the conditions table, the variable being tested is displayed in the first column, the operator used for comparison is displayed in the second column, and the value being compared is displayed in the third column. In the fourth column, a check box allows the user to replace the value being compared with the value of a variable from EMRALD. The outcomes table displays the variable being modified, the current value being assigned, and an option to replace the assigned value with an EMRALD variable. 'WHEN' block sections follow the same format, allowing the user to insert EMRALD variables into the code where appropriate. In both cases, if an outcome does not follow the format of assigning a value to a variable, it will be available to edit directly in a text box.

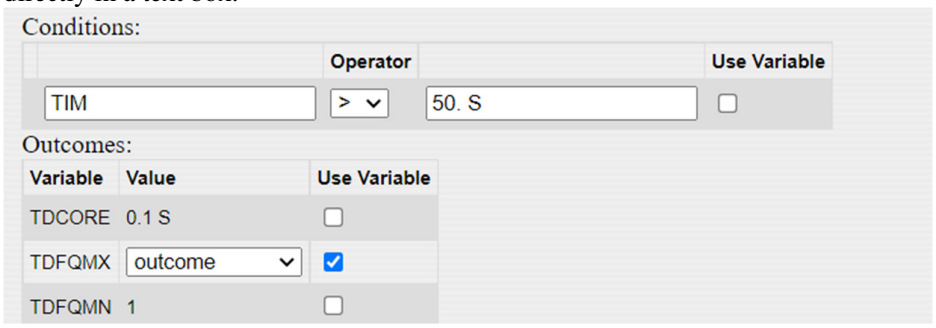

Figure 5. Example of the input blocks tab, showing the condition and variables being changed due to the condition and the option to link to an EMRALD variable.

4.6 User Interface - Output

The final tab of the form is the 'Outcomes' tab, as show in Figure 6, which allows the user to assign a MAAP output (e.g., 'CORE UNCOVERY,' 'VESSEL FAILURE,' 'CONTAINMENT FAILURE') to a document link variable from their project. The form auto-generates the script to construct a MAAP input file by replacing sections in the one specified by the user with new fields from the tabs described above. This input file is copied along with all the files described in Section 5.1 to a Windows temporary AppData directory. After the EMRALD simulation engine executes MAAP using the copied files and passes the new input file and parameter file as arguments to the executable, the results file is read into the specified document link variables. These variables can then be used to determine outcomes, such as core damage or start operator procedures, in the EMRALD model.

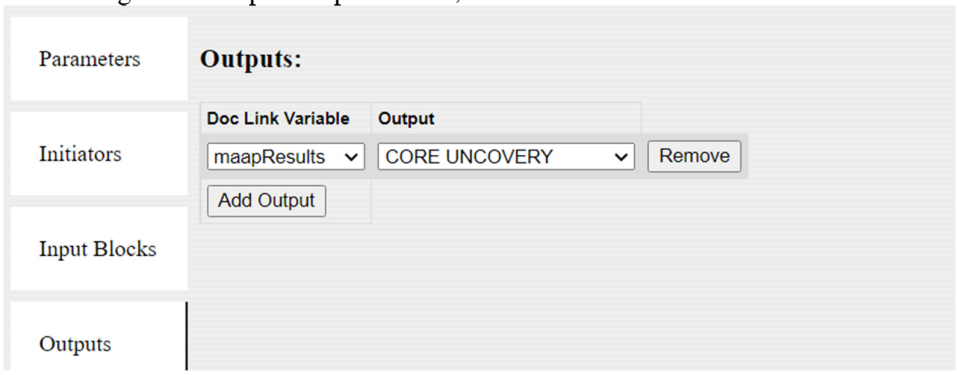

Figure 6. An example of the outputs tab where results from the MAAP analysis is assigned to EMRALD variables.

5 EXAMPLE MAAP & EMRALD MODEL

This section demonstrates how EMRALD can be used to: (1) run a MAAP analysis; and (2) revise (or propagate) the MAAP base case input file to run a series of MAAP cases and investigate the impact of parameters interested on the success criteria or operator action timing requirements. One of the example input decks, CRA1AI.INP, and the associated sample parameter file, WICE.PAR, from the MAAP 5.0.4 software package were used as a case study for the above demonstration. This sample parameter file represents a Westinghouse 4-loop PWR with an ice condenser containment, while the example input deck simulates a large loss of coolant accident (LOCA) scenario in which both high-pressure and low-pressure injections have failed to mitigate the accident. The original example input deck was revised for a break size of 2-in. in diameter and renamed as WIC-LOCA.INP. The 'END TIME' was also changed to 1 hour for quicker program execution.

5.1 MAAP Test Case

The following process was done by hand and then used as a test case to run MAAP in EMRALD. The executable MAAP files (e.g., PWR5DOS.dll, PWR5.dll), the sample parameter file (e.g., 'WICE.PAR' from the MAAP examples) and the input deck (e.g., 'WIC-LOCA.INP,' which was created by modifying the CRA1AI.INP MAAP example to last 1 hour) were placed into a user folder for the analysis (e.g., D:\MAAPusingEMRALD\WIC-LOCA). In the folder, the following disk operating system (DOS) command can be used to run an input deck:

PWR5DOS *Input_Deck_Name* (e.g., *WIC-LOCA.INP*) <*Parameter_File*> (e.g., WICE.PAR)

After the successful run of the base case input deck, a series of input decks can be built to investigate the plant responses and the timing of key event sequences for different scenarios (e.g., a different LOCA break size or break location, the availability of various mitigating systems, operator actions at different times) by revising the associated MAAP parameters. In this case study, ABBN(1) is the MAAP parameter that defines the LOCA break area. The value of ABBN(1) was revised to represent from 2-in. to 14-in. break size in the input decks, as observed in Table 1.

Table 1. Change LOCA Break Size in MAAP.

MAAP Input Deck	LOCA Diameter (inch)	LOCA Area (ft^2)	MAAP Code
WIC-LOCA	2	0.0218	ABBN(1) = 0.0218 FT**2
WIC-LOCA-1	3	0.0491	ABBN(1) = 0.0491 FT**2
WIC-LOCA-2	4	0.0873	ABBN(1) = 0.0873 FT**2
WIC-LOCA-3	5	0.1364	ABBN(1) = 0.1364 FT**2
WIC-LOCA-4	6	0.1963	ABBN(1) = 0.1963 FT**3
WIC-LOCA-5	7	0.2673	ABBN(1) = 0.2673 FT**4
WIC-LOCA-6	8	0.3491	ABBN(1) = 0.3491 FT**5
WIC-LOCA-7	10	0.5454	ABBN(1) = 0.5454 FT**6
WIC-LOCA-8	12	0.7854	ABBN(1) = 0.7854 FT**7
WIC-LOCA-9	14	1.0690	ABBN(1) = 1.0690 FT**8

MAAP ran the new input decks using the DOS command. The key event sequence timing can be obtained from the corresponding MAAP output summary files (e.g., WIC-LOCA-1.SUM). Table 2 shows the key events timing for different LOCA sizes (during the first hour after LOCA occurs). The column titles such as 'Reactor Scram,' 'PZR Empty,' and 'Core has Uncovered' are the MAAP phrases that the user can use to search for the events in which they are interested. Note that MAAP has other types of output files, such as '.LOG,' and '.TAB,' which include plentiful information on plant status (e.g., containment temperature, hydrogen generated, RCS waster mass). EMRALD can develop a user

interface and link to specific parameters in any of those output files that users may be interested in. In the next section, EMRALD shows the results extracted from the '.LOG' file for core 'uncovery' time.

Table 2. MAAP Key Events Timing (Seconds) with Different LOCA Size.

MAAP Input Deck	LOCA Diameter (in.)	REACTOR SCRAM	PZR EMPTY	MAIN COOLANT PUMPS OFF	CORE HAS UNCOVERED	CORE EXCEEDED 2499 K
WIC-LOCA	2	55	90	1706	2513	NA
WIC-LOCA-1	3	23	45	769	1610	NA
WIC-LOCA-2	4	11	28	431	948	NA
WIC-LOCA-3	5	7.3	20	279	713	NA
WIC-LOCA-4	6	5.1	16	197	260	NA
WIC-LOCA-5	7	3.8	13	144	197	NA
WIC-LOCA-6	8	2.9	12	110	158	NA
WIC-LOCA-7	10	1.9	10	70	112	2890
WIC-LOCA-8	12	1.2	9.6	48	90	2230
WIC-LOCA-9	14	0.9	9	35	71	1793

5.2 EMRALD Model

A very simple EMRALD model, as shown inFigure 7, adjusts the break size, as described above, using the custom MAAP form outlined in Section 4. First, a new plant diagram is created. In the diagram, three states are needed: (1) the start state; (2) a key result state; and (3) a done state. A 'Run Application' event with the custom MAAP form is used to setup the MAAP run. By being in the start state, MAAP will be executed immediately on startup of the EMRALD simulation. A document variable called 'CoreUncoveryHours' was created and assigned in the MAAP form to get the result of interest. Once the MAAP simulation is completed, this variable is updated and the EMRALD model moves to the 'MAAPResults' state before ending the simulation run.

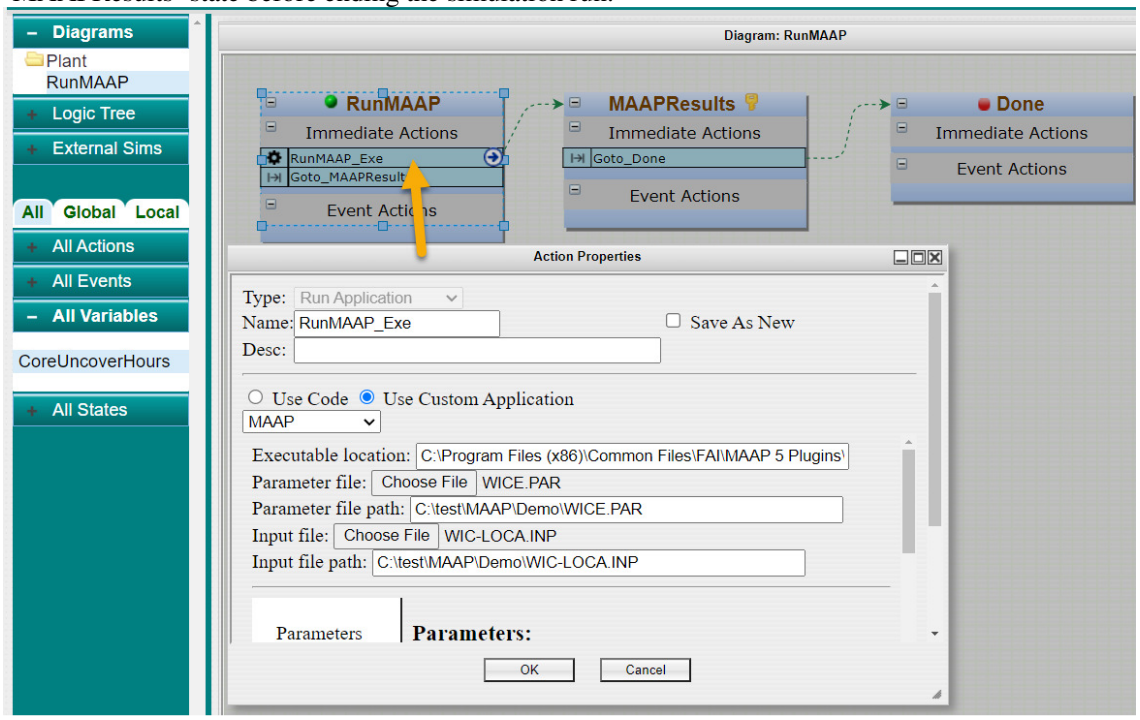

Figure 7. Simple EMRALD model to run MAAP.

To setup the MAAP parameters for testing, the 'WICE.PAR' file was selected for the parameter file and the 'WIC-LOCA.INP' for the input file, as shown in the Action Properties window in Figure 7. It is assumed that the input file is a base case from which changes will be made. After loading the two files, all the parameters, initiators, input blocks, and result options are loaded for the user to make the desired changes. In this case, the ABBN(1) parameter is simply changed to a value in Table 1, as shown on the left side of Figure 8.

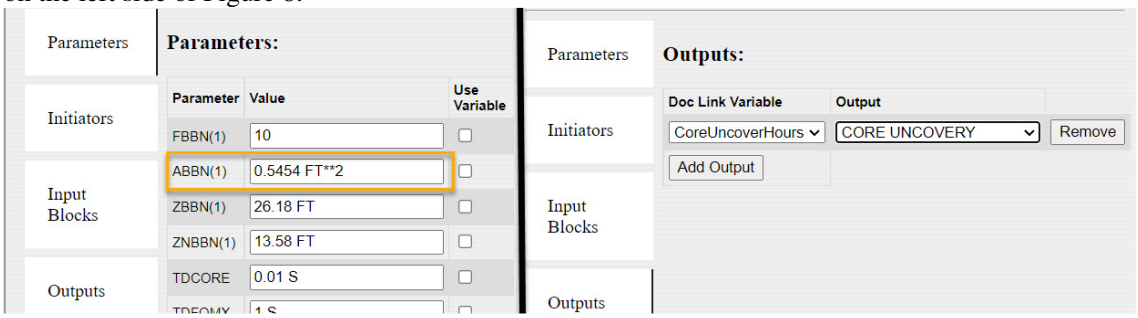

Figure 8. Modification and result options set for the test case.

In this initial proof-of-concept, the result gives the user the option to get data from the .LOG result file. The CORE UNCOVERY time was set to the CoreUncoveryHours variable for a comparison with the manual method, as shown on the right side of Figure 8.

5.3 EMRALD Simulation Results

To run MAAP, the web interface constructs a script to copy the MAAP dos.exe, input, and PAR files to a temporary data folder and assigns the properties needed for the results variable to get the data from the results. This model was saved and run using the EMRALD solver. Since the values are constant, only one run is needed to get a result, as shown in Figure 9. Each test case generated a MAAP input model, as expected, that came back with equivalent results as the manual execution except .LOG results are in hours instead of seconds.

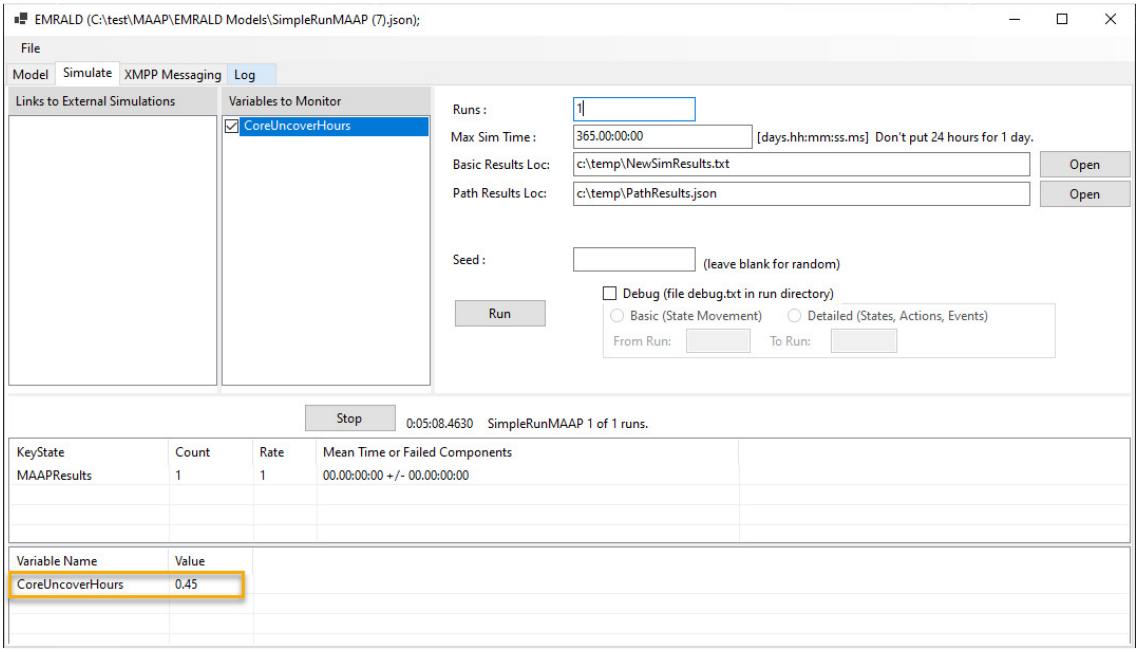

Figure 9. Example of EMRALD solve.

6 DYNAMIC EMRALD AND MAAP MODEL

With the coupling to MAAP, many useful statistical or dynamic analyses can be performed. The case demonstrates a simple example of an operational procedure tied to the MAAP model.

6.1 Scenario

The same MAAP model will be used for this scenario. If there is a LOCA, an operator has the time from core 'uncovery' to identify and isolate the break. The hypothetical operator task time is a normal distribution of 5 min. with a standard deviation of 3 min. Ideally, there would be a parameter in the MAAP model for when the break is isolated to determine core damage. However, to use the example models in MAAP and just demonstrate the dynamic behavior, it is assumed that if the operator performs his tasks before the core is uncovered, then the plant would be fine.

6.2 EMRALD Model

To model this in EMRALD, a variable needs to be used to set the size of the break, which has a uniform distribution of size between 2-14 in. This model, which is shown in Figure 10, assumes there has been a break and will immediately do the following:

(1) Sample the size of the break.

(2) Run MAAP.

(3) Start the operator procedure.

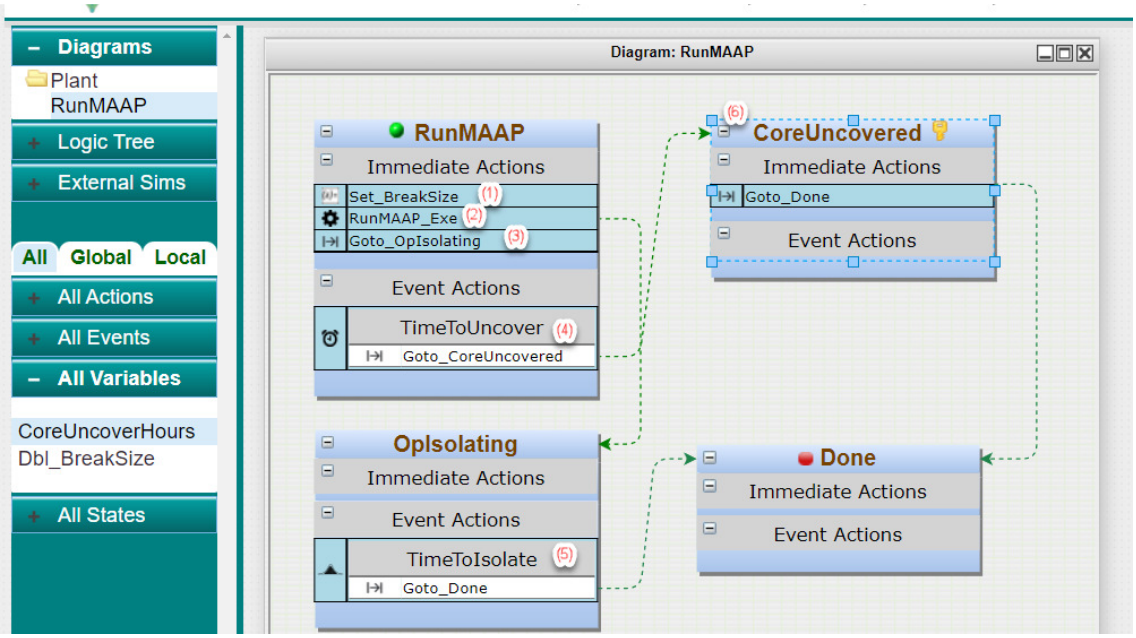

Figure 10. Dynamic EMRALD model running MAAP.

Event (4), 'TimeToUncover,' is a timer event linked to the uncover time variable set by the results of the MAAP run. If this event occurs before event (5), 'TimeToIsolate,' then the siumulation moves to event (6), the 'CoreUncovered' state, and quits. If event (5), 'TimeToIsolate,' occurs first, then the simulation quits without going into the 'CoreUncovered' State. The user specifies the number of times for the EMRALD model to run, depending on the desired uncertainty level, and the probability of the operator procedure failing to save the plant is determined by the times the simulation ends in "CoreUncovered" divided by the total number of runs.

6.3 Dynamic Example Results

The EMRALD model ran 25 simulations with 15 ending in 'CoreUncovered' (60% of the time). Other data or outcome states could also be tracked, depending on what the analyst needs. EMRALD also tracks all the paths to the final states so probability of specific avenues can also be analyzed. While this is a simple example, it shows the potential to model and run the dynamic behavior in a simple manner.

7 CONCLUSION

This work shows that EMRALD provides a simple method to enable both MAAP experts and non-MAAP experts to conduct analyses that are necessary for dynamic plant scenarios. The EMRALD custom application form makes it possible to add complex tools to EMRALD, thereby eliminating the need for the user to write complex scripts or fully understand the workings of the other tool. They just need to know what items to adjust, as well as the data they want out of it, to link with their EMRALD model.

Acknowledgements
The researchers express their appreciation to EPRI and Southern Nuclear Company for their assistance.

References

[1] Battelle Energy Alliance (BEA), "EMRALD," [Online] (n.d.). Available at: https://github.com/idaholab/EMRALD (last accessed 19 April 2022).

[2] S. Prescott, C. Smith, and L. Vang. "EMRALD: Dynamic PRA for the Traditional Modeler," Probabilistic Safety Assessment and Management (PSAM) 14, September 2018, Los Angeles, CA, USA [Online] (2018). Available at: http://www.iapsam.org/psam14/proceedings/paper/paper_76_1.pdf (last accessed 19 April 2022).

[3] Idaho National Laboratory (INL), "EMRALD," [Online] (n.d.). Available at: https://emrald.inl.gov (last accessed 19 April 2022).

[4] Idaho National Laboratory (INL), "EMRALD Docs," [Online] (n.d.). Available at: https://emraldapp.inl.gov/docs/ (last accessed 19 April 2022).

[5] JGraph Ltd., "mxGraph," [Online] (n.d.). Available at: https://github.com/jgraph/mxgraph (last accessed 19 April 2022).