

Light Water Reactor Sustainability Program

Guidance Document for Using Dynamic Force-on-Force Tools



September 2021

U.S. Department of Energy

Office of Nuclear Energy

DISCLAIMER

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness, of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

Guidance Document for Using Dynamic Force-on-Force Tools

Robby Christian
Steven R. Prescott
Christopher P. Chwasz
Vaibhav Yadav
Shawn W. St. Germain

September 2021

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy

SUMMARY

The requirements for U.S. nuclear power plants to maintain a large onsite physical security force contribute to their operational costs. The cost of maintaining the current physical security posture is approximately 10% of the overall operation and maintenance budget for commercial nuclear power plants. The goal of the Light Water Reactor Sustainability (LWRS) Program Physical Security Pathway is to develop tools, methods, and technologies and provide the technical basis for an optimized physical security posture. The conservatism built into current security postures may be analyzed and minimized in order to reduce security costs while still ensuring adequate security and operational safety. The research performed at Idaho National Laboratory within LWRS Program Physical Security Pathway has successfully developed dynamic force-on-force (FOF) modeling framework using various computer simulation tools and integrates them with the dynamic assessment Event Modeling Risk Assessment using Linked Diagrams (EMRALD) tool.

This document provides a guidance for using the dynamic computational framework for optimizing physical security at nuclear power plants. The instructions in this document cover the FOF tools that are used by a majority of U.S. commercial plants for their physical security modeling and assessment. Following the guidance in this document will enable a user to integrate their plant specific FOF models with the dynamic simulation tool EMRALD, model operator actions, integrate with probabilistic risk assessment tools, such as Computer Aided Fault Tree Analysis System (CAFTA) or Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE), and with thermal-hydraulic tools, such as RELAP-5. The dynamic computational framework enables a plant's physical security to transition from the traditional binary assessment of success/failure to a risk-informed performance-based assessment. Such an assessment enables further analysis, such as what-if scenarios and staff-reduction evaluation thereby optimizing physical security at plants.

CONTENTS

SUMMARY	iii
ACRONYMS.....	ix
1. INTRODUCTION.....	1
2. LWRS DYNAMIC PHYSICAL SECURITY FRAMEWORK.....	1
2.1 Framework Outline	1
2.2 Physical Security Optimization Steps	2
2.2.1 Base Case Evaluation.....	2
2.2.2 Comparison Results	3
2.2.3 Potential Strategy Evaluation.....	4
2.2.4 Staff-Reduction Evaluation.....	5
3. KEY SOFTWARE TOOLS	7
3.1 Force-on-Force Tools.....	7
3.1.1 SCRIBE3D.....	7
3.1.2 AVERT	8
3.1.3 SIMAJIN.....	8
3.2 Plant Response Tools.....	8
3.2.1 Traditional PRA Tools.....	8
3.2.2 Thermal-Hydraulics Tools	9
4. EMERALD	9
4.1 EMERALD Setup.....	10
4.1.1 EMERALD Source Code Setup.....	10
4.1.2 EMERALD Web Service	10
4.2 EMERALD Modeling Pieces	11
4.2.1 States in EMERALD	11
4.2.2 Actions in EMERALD	12
4.2.3 Events in EMERALD.....	12
4.2.4 Variables in EMERALD (Including Dynamic XML and JSON Variables)	13
4.2.5 Diagrams in EMERALD.....	13
4.2.6 Logic Trees	13
4.3 Modeling Operator Actions.....	14
4.3.1 Example Procedure: FLEX Generator	14
4.3.2 Diagram and Operator States	14
4.3.3 Determining Event and Actions.....	15
4.3.4 Triggering the Task and Upon Completion or Failure.....	18
4.4 Integration of FOF Tool with EMERALD	19
4.4.1 EMERALD-SCRIBE3D.....	19
4.4.2 EMERALD-AVERT	28
4.4.3 EMERALD-Simajin.....	36
4.5 Integration of Thermal-hydraulic Tool with EMERALD	42
4.6 Running Analysis.....	47
4.6.1 User Interface.....	48

4.7	EMERALD Results	50
4.8	Helpful Hints.....	50
4.8.1	Modeling.....	50
4.8.2	Solving.....	52
5.	REGULATORY CONSIDERATIONS	52
5.1	Changes to the Physical Protection System	53
5.2	Security Plan Changes	53
6.	CONCLUSION	54
7.	REFERENCES	54
	Appendix A SCRIBE3D JSON File	57
	Appendix B XML Output File of Simajin	63

FIGURES

Figure 1.	Outline of the dynamic framework.....	1
Figure 2.	Flow for creating base case comparison results.	3
Figure 3.	Hypothetical base case results example.....	4
Figure 4.	Flow for option evaluation.	5
Figure 5.	Process to evaluate staff reduction for a strategy change.	6
Figure 6.	Hypothetical example of post importance per scenario.....	7
Figure 7.	Download menu option to obtain the solve engine for running locally.....	11
Figure 8.	Example states for a set of operator actions to turn on a pump.	12
Figure 9.	Example of a logic tree.....	14
Figure 10.	Adding a new diagram for the FLEX procedure.	15
Figure 11.	States for the FLEX procedure.	15
Figure 12.	Variable condition event to determine availability.....	16
Figure 13.	Normal distribution event for how long it takes to do a load shed.....	16
Figure 14.	Diagram for a FLEX diesel generator.	17
Figure 15.	Event property to indicate if exiting the current state after the event occurs.	17
Figure 16.	Events and Actions for the FLEX procedure of switching to EDG power.....	18
Figure 17.	Event and action triggering the transition to start the FLEX EDG procedure.....	18
Figure 18.	The "Plant_Damage" state tracks the times this condition occurs.....	19
Figure 19.	LonePine Facility Layout in SCRIBE3D [2].....	20
Figure 20.	Sample of SCRIBE3D JSON file.	21
Figure 21.	Main EMERALD diagram to integrate EMERALD and SCRIBE3D.	22

Figure 22. Sub-diagram of adversary tasks at PIDAS.	23
Figure 23. AlarmTrigger action to set a probabilistic PIDAS alarm.	23
Figure 24. GetPidatime event.....	24
Figure 25. C# codes for RunScribe1 action.	25
Figure 26. IfPidatimeDefeated event.....	26
Figure 27. IfPidatimeAlive event.	26
Figure 28. RunScribe2 action.	27
Figure 29. UpdateScribeDGATime action.....	28
Figure 30. Main EMERALD diagram.....	29
Figure 31. C# code to set the plant shutdown timing.....	30
Figure 32. C# code to determine the timing when all target components are sabotaged.....	31
Figure 33. C# code to determine FLEX usage.....	32
Figure 34. C# code to determine if all targets are sabotaged.....	32
Figure 35. C# code to count the number of intact EDGs.....	33
Figure 36. C# code to determine the number of intact TDPs.....	34
Figure 37. C# code to decide whether LOOP occurred or not.....	35
Figure 38. C# code to calculate when the sabotage attack ends.	35
Figure 39. Set_CD_Prob action.	36
Figure 40. Snapshots of Simajin [10].....	37
Figure 41. Simajin XML file on combat statistics of each player.....	38
Figure 42. Simajin XML file on overall FOF results.....	38
Figure 43. Main diagram for the example of EMERALD-Simajin integration.	39
Figure 44. DoSimajin action.	40
Figure 45. ReadResults action.	41
Figure 46. XML variable of Diesel Generator 1 sabotage timing.....	41
Figure 47. XML variable of Diesel Generator 2 sabotage timing.....	42
Figure 48. EMERALD diagram to feed data to and read output from MELCOR.	43
Figure 49. SetFilesCopied C# code.....	45
Figure 50. MC_BatteryTime variable.....	45
Figure 51. MELCOR_exe action.	46
Figure 52. DetermineRunSuccess action.	47
Figure 53. MELCORE_EndMsg variable.....	47
Figure 54. Model tab of the EMERALD solve engine.....	48
Figure 55. "Simulate" tab of the EMERALD solve engine.....	49
Figure 56. EMERALD results summary area.	49

Figure 57. Example of debut output file.	50
Figure 58. Example of Basic results output.	50
Figure 59. Testing using a transition action with multiple “To States.”	51
Figure 60. Project merge menu option.	51
Figure 61. Example of settings to debug a specified simulation run.	52

ACRONYMS

BRE	Bullet resistant enclosures
CAFTA	Computer Aided Fault Tree Analysis
CD	Core damage
DG	Diesel Generator
DID	Defense in Depth
DOE	Department of Energy
EDG	Emergency Diesel Generator
EMRALD	Event Modeling Risk Assessment using Linked Diagrams
EPRI	Electric Power Research Institute
FOF	Force-on-force
INL	Idaho National Laboratory
LWRS	Light Water Reactor Sustainability
MAAP	Modular Accident Analysis Program
MCC	Motor Control Center
NRC	Nuclear Regulatory Commission
PIDAS	Perimeter intrusion detection and assessment system
PK	Kill Probability
PPS	Physical protection system
PRA	Probabilistic Risk Assessment
RAVEN	Risk Analysis and Virtual Environment
ROWS	Remote operated weapons systems
SAPPHIRE	Systems Analysis Programs for Hands-on Integrated Reliability Evaluations

GUIDANCE DOCUMENT FOR COMBINING DYNAMIC FORCE-ON-FORCE TOOLS

1. INTRODUCTION

The requirements for U.S. nuclear power plants to maintain a large onsite physical security force contribute to their operational costs. The cost of maintaining the current physical security posture is approximately 10% of the overall operation and maintenance budget for commercial nuclear power plants [1]. The goal of the Light Water Reactor Sustainability (LWRS) Program Physical Security Pathway is to develop tools, methods, and technologies and provide the technical basis for an optimized physical security posture. The conservatisms built into current security postures may be analyzed and minimized in order to reduce security costs while still ensuring adequate security and operational safety.

This research performs dynamic force-on-force (FOF) modeling using various computer simulation tools and integrates them with the dynamic assessment Event Modeling Risk Assessment using Linked Diagrams (EMRALD) tool developed at Idaho National Laboratory (INL). These FOF models are powerful tools to perform a quantitative assessment of a plant's physical security performance effectiveness under simulated scenarios. These models enable the analysis of current postures, perform sensitivity analyses of variables and elements of physical security, identify strengths and weaknesses in the current strategy, explore different strategies by simulating variables and outcomes in a given sabotage scenario, and derive potential approaches to optimize a plant's physical security posture.

This report describes the overall LWRS Dynamic Physical Security Framework, the tools that may be used to employ this framework, how to set up the tools for performing dynamic physical security analysis, how the various tools may be integrated, and, finally, how to interpret and use the results to evaluate or optimize a physical security posture at a nuclear facility. The goal is to allow utilities to follow a formalized process for using FOF simulation to evaluate, identify, and implement optimized physical security postures. This framework may be useful when combined with previously published economic analysis tools to identify and evaluate cost effective security upgrade options.

This report utilizes generic values associated with the Sandia National Laboratory's Lone Pine Model of a nuclear power plant [2]. The analysis in this report is generic in nature and bears no resemblance to any existing nuclear power plant, the site protective strategy, nor the adversary characteristics (i.e., design basis threat [DBT]).

2. LWRS DYNAMIC PHYSICAL SECURITY FRAMEWORK

2.1 Framework Outline

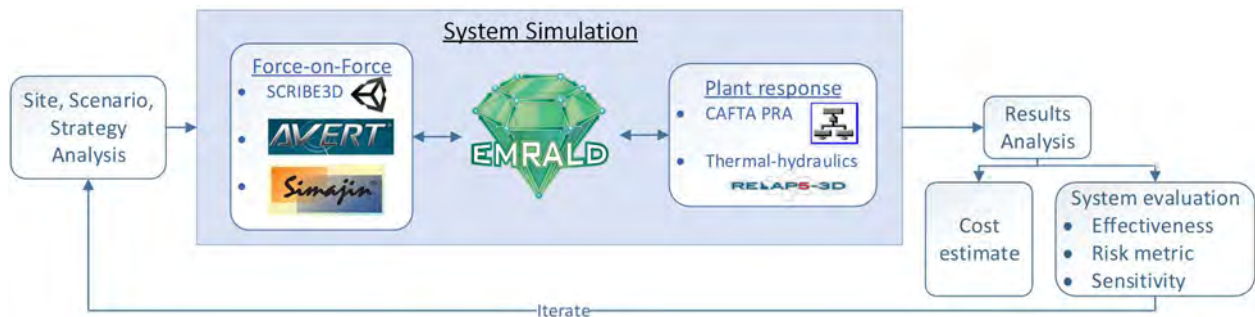


Figure 1. Outline of the dynamic framework.

In the LWRS Dynamic Physical Security Framework, EMRALD is used to manage the different FOF simulation tools and supplement the simulation capabilities with dynamic uncertainties, as shown in

Figure 1. This framework allows a security analyst to identify the excess conservatism in their security posture and gain further insight on optimizing the posture for protection effectiveness and associated costs. The following are examples of analyses that may be performed using this framework:

1. Demonstrating minimal impact to security effectiveness using randomized shift breaks.
2. Evaluating the potential cost savings or performance improvements for implementing new physical security features such as bullet resistant enclosures (BREs).
3. Optimizing the placement of BREs on a site.
4. Analyzing operator actions to mitigate sabotage attacks, such as:
 - a. The likelihood and possible pathway for control room operators to evacuate to a backup control room under certain circumstances.
5. Crediting diverse and flexible coping (FLEX) mitigation strategies.
6. Evaluating the effectiveness of potential new security technologies such as advanced non-lethal delay devices or remote operated weapons systems (ROWS).

2.2 Physical Security Optimization Steps

Justifications to optimize changes in a physical security posture, according to Section (p) of U.S. Nuclear Regulatory Commission's (NRC's) 10 CFR 50.54 [3], must show the changes do not reduce the effectiveness. To act as a standard, a process is outlined for evaluating potential strategy or equipment changes and determining the staff reduction made possible by using those changes while showing the maintained effectiveness. This process can be done through simulations and results comparisons. The following subsections outline the process, while Sections 3 and 4 go over the software and how to implement the framework.

2.2.1 Base Case Evaluation

Before developing optimization strategies, baseline results from a plant's current defensive posture needs to be modeled in a simulation tool capable of capturing the strategies and procedures established by the nuclear power plant. To test alterations vs. the base case, a cover-set of scenarios needs to be developed. Typically, expert judgement, past FOF exercises, and possibly software tools are used to identify only the top attack scenarios. While these scenarios will be included in the cover-set of scenarios, additional scenarios testing various attack directions and more protected areas that may normally be removed because of low adversary success are included. If a facility currently has FOF models and scenarios, start with these and reevaluate to make a complete cover-set.

While in traditional numerical analysis there is only a single set of base results to compare against, the FOF model will likely need two different sets of base case data. This is due to the layered defense approach and possible high value of probability of effectiveness that can cause some aspects of the security posture to not be tested thoroughly. This second base case model, called the "Defense in Depth" (DID) model, will have a modification that allows more success scenarios by the adversary and provides more cases where the inner layers of defense are tested. This modification could be an increased attack force, a reduction of hit-to-kill ratio for the guards, or a layered scenario approach where attacks start inside initial defenses. Do not remove any posts to develop the DID model as the removal of posts will be part of the optimization process. If only the unmodified base case were used, few variations in failure scenarios or cases would be available to evaluate against and would result in a high uncertainty. To get results with low uncertainty, a result set needs to have a significant number of cases with varied paths of failure.

When developing the DID model, choose methods that will not be detrimental to the potential strategy changes discussed in Section 2.2.3. For example, if you are considering adding a delay barrier to

give less guards more time to take down the adversaries, your DID model should probably not use a large increase in attack force numbers, as this would mask the effectiveness of the change.

Even with a slight reduction in effectiveness, the number of failed evaluation cases should significantly increase. For example, in 5,000 simulations, if the base case effectiveness is 98%, only 100 evaluation cases are available and are likely from only a couple paths. With a DID model of 91% effectiveness, 450 cases would be generated from the 5,000 simulations. The key is not only to capture more failure cases, but also to generate different failure avenues. The evaluations corresponding to failure cases will be used to evaluate the modified strategies and can clearly identify improvements or defense reductions where only using the original base case tests would show little to no change.

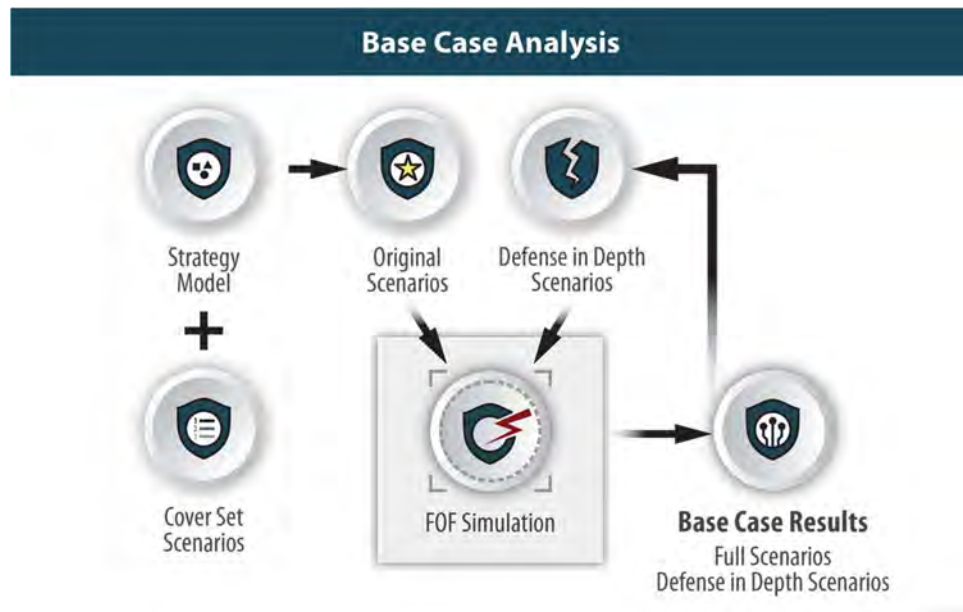


Figure 2. Flow for creating base case comparison results.

In summary, the process for developing the base case results are the following steps, as shown in Figure 2:

1. Model the plant protection strategy
2. Develop cover-set scenarios
3. Run FOF simulations with cover-set scenarios and save results
4. Develop a DID model
5. Run DID simulations with cover-set scenarios and save results.

2.2.2 Comparison Results

The conservative approach to verify the protection level for a facility has not decreased is to verify no scenario's adversary success probability has increased above the highest base case value and the overall risk has not increased.

The results from the base case evaluation should be laid out in a manner like Figure 3, listing each scenario, the adversary success probability, and calculating an importance measure. The importance measure is the probability of that scenario divided by the sum of all the scenario probabilities. For an overall comparison value, the min cut upper bound equation can be used so that no matter how many

scenarios there are, the risk cannot be greater than 1; this is more important for the DID comparisons. This layout simplifies the comparison process in Section 2.2.4.

Original Base Case			DID Base Case		
Scenario	Adversary Success Prob.	Importance Measure	Scenario	Adversary Success Prob.	Importance Measure
A	0.011	40.29%	A	0.51	40.48%
B	0.01	36.63%	B	0.53	42.06%
C	0.003	10.99%	C	0.1	7.94%
D	0.0015	5.49%	D	0.04	3.17%
E	0.0015	5.49%	E	0.05	3.97%
F	0.0003	1.10%	F	0.03	2.38%
	Min Cut			Min Cut	
	0.027045626			0.816640667	

Figure 3. Hypothetical base case results example.^a

2.2.3 Potential Strategy Evaluation

Each facility can have different options they consider for optimizing their defensive posture. Some options can be evaluated in a research setting for a variety of facilities meeting defined conditions. Others could be site-specific. A potential evaluation should be done to determine the probable and best improvement options before the full in-depth modeling process is done and evaluated, as described in Section 2.2.1.

The critical part for evaluating a potential change is having a tool that can correctly simulate the response or effect of the potential change and apply those effects to the FOF simulation. If the FOF simulation tool used for the base case evaluations has the capability to model the change correctly or conservatively, this evaluation can be a fairly simple process. Some protection strategies can require complex modeling of operator procedures and timing, such as using the FLEX equipment designed for beyond-design external events as additional safety equipment after an attack [4]. Other strategies could be including simple actions that also need plant system modeling or thermal dynamics to get more precise failure timing. These would require coupling the FOF simulation with other tools needed to correctly model the behavior.

For this guidance document, INL’s EMERALD tool is coupled with the FOF simulation tool [5]. EMERALD allows the user to model complex operator actions and couple that model with the FOF simulation by using data from the model to make a decision or adjust the FOF model according to events in the EMERALD model.

Modify the DID model to reflect the new strategy. Next, run the model using the cover-set scenarios. If the results show a significant improvement to the base case DID results, the model can be used for the staff-reduction evaluation process in Section 2.2.4.

^a Note: Values were manipulated to demonstrate a higher likelihood of adversary success for the purposes of the analysis in this report. These do not represent realistic values.

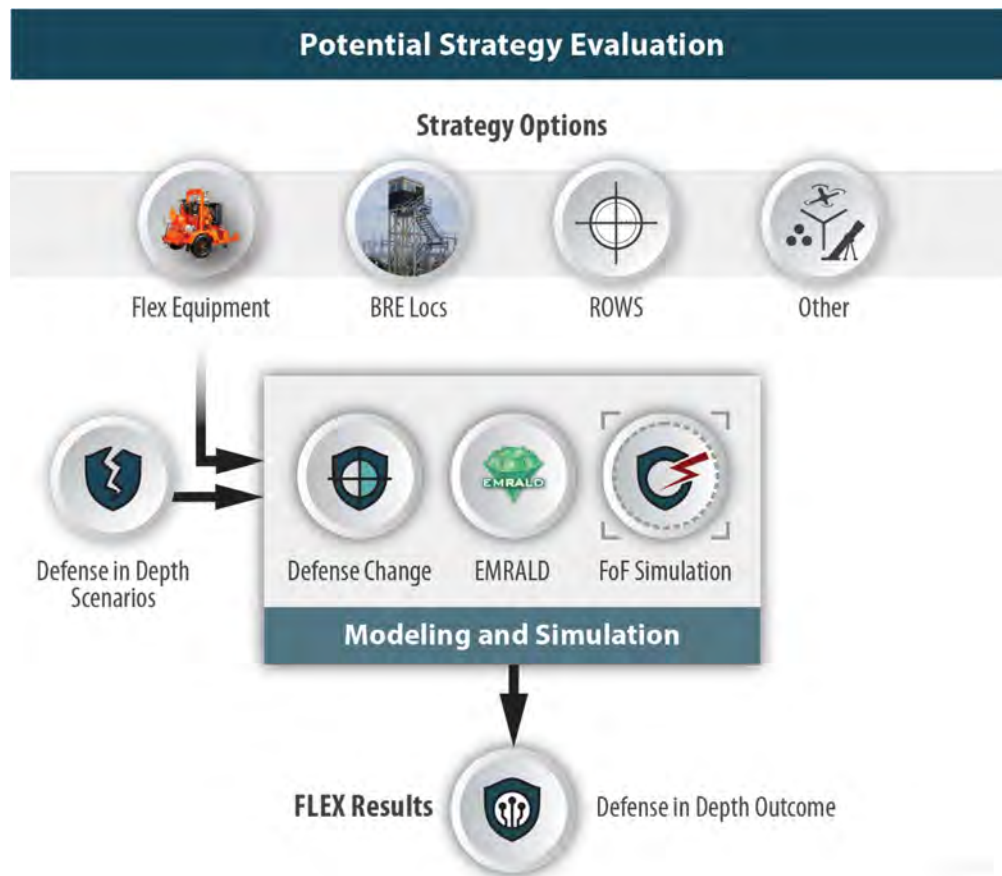


Figure 4. Flow for option evaluation.

In summary, the following steps are used to evaluate a potential strategy protection option, shown in Figure 4:

1. Determine likely improvement methods for strategy change
2. Build a model of those changes using an appropriate tool or tool combination
3. Apply the DID scenarios to the new model/s and run the simulations
4. Compare the results to the original DID results.

2.2.4 Staff-Reduction Evaluation

Once likely improvement methods have been identified and modeled, the process for determining a staffing reduction can begin using the new model and the base case models. This process will ensure, even after a potential staff reduction, an equivalent protective strategy is maintained to at least its current level. The four main steps to this process are outlined in Figure 5 and described below. Before the process begins, a copy of the original and DID base case simulation scenarios and results is made. This is an iterative process and stops once the criteria has been met.



Figure 5. Process to evaluate staff reduction for a strategy change.

The following steps outline the process shown in Figure 5 and are further described in the subsequent sections.

1. Use the initial new strategy model or previous iteration’s results to determine the least effective post.
2. Remove the identified “least effective post” from the new strategy model.
3. Run the FOF simulation using the new strategy model with removed posts.
4. Compare the results from Step 3 with the DID base case results. If new strategy is still better than DID base case, update the remove list and continue to Step 1; if not, go to Step 5.
5. Modify the original model with the new strategy and perform a final verification against the original base case results.

2.2.4.1 Evaluating Least Effective Post

The primary measurement to evaluate the least effective post, in Step 1, should be the number of kills, but other considerations could also be a factor, such as shots fired and detections. The FOF simulations can provide this information and they should be used to give an importance measure for each post in each scenario as shown in Figure 6. While the data can be gathered through an automated process, determining the least effective post is a manual process and will most effectively be done by someone familiar with the defensive strategy and the dynamics of the scenarios.

Scenario	Adversary Success Prob.	Importance Measure	P1	P2	P3	P4
A	0.497	45.81%	0.75	0.25	0	0
B	0.29	26.73%	1	0	0	0
C	0.11	10.14%	0.5	0.25	0.25	0
D	0.188	17.33%	0.25	0.75	0	0
	Min Cut Risk	Sum				
	0.741909292	1.0000				

Figure 6. Hypothetical example of post importance per scenario.

Some general guidance for picking the least effective guard:

- First, pick guards that are not very effective in any scenarios, as shown in yellow in Figure 6
- Do not pick guards that are key for a particular scenario, as shown in green in Figure 6
- If several are close in value, test removing each to determine if one has a significantly higher increase in overall “Min Cut” or particular scenario.

2.2.4.2 Comparing Results

When comparing results in Step 4 in Section 2.2.4, first look at the “Min Cut” value and verify the new strategy DID result is not greater than the DID base case. Then, make sure no individual scenario’s adversary success probability is higher than that of the highest DID base case. If neither case is met, update the remove list with the post chosen for that round and repeat Step 1. If either case is met, no more posts can be removed, and the process moves to Step 5.

2.2.4.3 Final Check Step

The final verification needs to be done using the original base case model vs. the new strategy with the remove list applied. Modify the original base case model with the new strategy and remove all the guards in the list. Run all the cover-set scenarios and verify the highest adversary success probability scenario is less than or equal to the original highest. Also, verify the “Min Cut” is equal to or lower than the original.

3. KEY SOFTWARE TOOLS

Developing an enhanced dynamic risk model for physical security involves the interaction of several software tools, as outlined in Section 2. This section goes over the basic capabilities of some key tool options for use in the framework, namely FOF simulation, probabilistic risk assessment (PRA), and thermal-dynamics tools. Other tools may be needed or useful for specific scenarios or plant needs, but are not covered in this report.

3.1 Force-on-Force Tools

There are several FOF simulation tools available. Each has different advantages.

3.1.1 SCRIBE3D

The SCRIBE3D software was developed at Sandia National Laboratory and is available for free upon request [6]. This software visualizes the environment in a three-dimensional (3D) view. It offers a visual recording and playback of scenarios developed during a tabletop exercise. This visualization allows analysts to open discussions and capture their results, visualize the attack consequences, collect data, and record events. The terrain data in a SCRIBE3D model is developed from various sources such as maps, satellite imagery, GIS height, area data, and high-fidelity 3D laser scan models. Movement and actions of

entities or players in the model are developed by using a spreadsheet or dynamic placement. The entity attributes include their position, posture (crouching or standing), role, equipment or weapons, rules of engagement, and skill levels. The events during simulations are recorded based on the time and location they happen. These events are displayed on a Gantt timeline showing the entity's posture, the entity's state (active, suppressed, or killed), the entry and exit time, the state of engagement, and general notes made by the analyst.

3.1.2 AVERT

AVERT is a modeling and simulation tool that provides in-depth evaluations of a site's vulnerability to a terrorist attack. As an agent-based, Monte Carlo simulation, AVERT is designed to simulate large numbers of attacks under varying conditions with simulated security features—including sensors, barriers, and security forces—and adversaries responding to simulated events.

Security forces and adversaries are represented by software agents. Security forces are given normal operations and alert operations. Adversaries are assigned a set of objectives and strategies with which to accomplish them. Included path analysis tools are used to automatically determine the best routes for agents, based on criteria, such as the fastest route or the route providing the most cover.

The software is commercially available with an EAR99 classification; the library of data has additional restrictions. See <https://aressecuritycorp.com>.

3.1.3 SIMAJIN

Simajin/Vanguard provides users an ability to determine risk associated with physical security processes using modeling and simulation. The FOF simulation tool enables physical security analysis across a range of environments and with any number of threat vectors and variables. The features enable analysts to conduct risk assessments and quantify how well a protective measure will repel, or defeat, a suite of tailored threats. It is licensed and used in the U.S. Department of Energy (DOE) and NRC marketplace for probability of neutralization computations.

This software is commercially available with no export restrictions. See <https://www.rhinocorps.com/products/vanguard-vulnerability-assessment-tool>.

3.2 Plant Response Tools

Different types of plant modeling tools can be used in developing a dynamic FOF model. Some of these tools can be used for model data or coupled to the simulation runs. This section lists PRA and thermal-hydraulics tools, but other tools such as explosive analysis or radiation plume software could also be included.

3.2.1 Traditional PRA Tools

3.2.1.1 CAFTA

Computer Aided Fault Tree Analysis System (CAFTA) is a software tool developed by Electric Power Research Institute (EPRI) for quantifying fault tree models of nuclear power plants. It is used widely in industry for risk-informed decision-making. The demo version of the software is available for the public to download while the full version is available for purchase.

Existing plant models can be used for determining the likelihood of core damage, given specific component failures in a scenario; this is further described in Section 4.4.2.

3.2.1.2 SAPHIRE

Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) is a software tool developed at INL for the NRC to perform PRA of fault trees and event trees of nuclear power plants.

The software is available to SAPHIRE user group members and older versions are available for free with approval from the NRC.

In a similar manner as CAFTA, SAPHIRE models can be used for determining the likelihood of core damage, given specific component failures in a scenario. This is further described in Section 4.4.2.

3.2.2 Thermal-Hydraulics Tools

3.2.2.1 RELAP5-3D

RELAP5-3D performs analysis of transients and accidents in water-cooled nuclear power plants and related systems, as well as the analysis of advanced reactor designs. It can perform fully-integrated, multi-dimensional, thermal-hydraulic, and kinetic modeling. The software is freely available through the INL but is export controlled.

A thermal-hydraulic model can provide a more precise determination of core damage, given specific component failure times, instead of using a conservative number from the PRA model. One limitation of RELAP5-3D is that the simulation stops once any simulated core damage occurs; if the level of environmental release is desired as part of the analysis, a different thermal-hydraulic analysis needs to be used.

3.2.2.2 MELCOR

MELCOR provides the impacts of severe accident phenomena as they propagate through defined control volumes. These control volumes are flexible enough to model various nuclear system components, such as the reactor core, spent fuel pool, and reactor containment. Results from MELCOR can determine the amount of material that combusts, melts, and is ejected at high or low pressures and carries radioisotopes with it. MELCOR models the progression of severe accidents in light-water reactor nuclear-power plants using a control volume approach. MELCOR was developed specifically for the NRC to evaluate the phenomena associated with hypothetical severe accidents at reactors. It is restricted but is available through the NRC Cooperative Severe Accident Research Program.

As with RELAP5-3D, a MELCOR thermal-hydraulic model can provide a more precise determination of core damage, given specific component failure times, instead of using a conservative number from the PRA model, and results can be used for extended radioactive plume analysis. One limitation of MELCOR is the long quantification times, which can be days, dependent upon the stability of the model.

3.2.2.3 MAAP

Modular Accident Analysis Program (MAAP) is a thermal-hydraulics tool developed by EPRI to model severe accident phenomena. It uses predetermined values to reduce some of the calculations to algebraic expressions, which makes the calculation time 10 to 100 times faster than MELCOR. However, the predetermined values may limit the flexibility of the model to within given boundaries.

As with the other thermal-hydraulics, MAAP can be used to provide a more precise determination of core damage, given specific component failure times. The software is available to EPRI members. For a more detailed comparison on thermal-hydraulics tools, see [7].

4. EMERALD

EMERALD is a dynamic PRA tool designed for visually modeling interactions between components, systems, and people, along with integrating external codes. For this work, EMERALD is used to demonstrate incorporating plant behavior and procedures directly into the LWRS Dynamic Physical Security Framework. [5]

4.1 EMERALD Setup

The EMERALD software is open source and available on GitHub at <https://github.com/idaholab/EMERALD>. There are two main parts to EMERALD: the web user interface for developing the model and the solve engine to run the model and get results. Typically, the web user interface is hosted on a server, but the following goes through both the steps for using EMERALD hosted on a server and setting up and running everything on a local windows machine.

4.1.1 EMERALD Source Code Setup

The following steps outline how to setup and run EMERALD on a local machine.

1. Install Git - <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
2. From the Git command line, move to the location to place the source and run the command “git clone <https://github.com/inl-labtrack/EMERALD.git>”
3. Install Visual Studio (can use community edition) - <https://visualstudio.microsoft.com/vs/community/>
4. Open the project in Visual Studio, and in the “Emrald_Site” project, right-click on “RunFirst.bat” and run
5. Right-click on the EMERALD solution in the “Solution Explorer” and select “Build Solution.” It should say the build succeeded and have no errors in the “Error List.”

To view the documentation:

1. Download and install Node.js from <https://nodejs.org/en>. (This will also install npm.)
2. Navigate to the "emerald-docs" directory (i.e., \EMERALD\emerald-docs) in a terminal window, such as Command Prompt.
3. Run the command npm install to install the project's dependencies.
4. Start the local dev server by running the command “npm run dev”.
5. Once the server is running, this message should appear: VuePress dev server listening at <http://localhost:8080/>.
6. Navigate to <http://localhost:8080/> to view the EMERALD Documentation.

To view the web-based modeling tool:

1. Set the default browser for Visual Studio to Chrome
2. Right-click on Emrald_Site in the solution explorer and select “View in Browser.”

To run the EMERALD simulator:

1. Right-click on in the solution explorer and select “Set as Startup Project”
2. To run in debug, use the Visual Studio menu “Debug” -> “Start Debugging.”

4.1.2 EMERALD Web Service

A web service hosted by INL allows users to develop EMERALD models and download the solve engine to run the simulations on the local machine. Currently, the modeling site is down while security upgrades are implemented but the following outlines how to access the tools.

To view the documentation:

1. Go to <https://emerald.inl.gov/docs> and navigate to the desired documentation.

To view the web-based modeling tool:

1. Go to <https://emerald.inl.gov> and navigate to the online modeling tool.
2. Select from the menu “Project” -> “New”
3. Develop the model as desired.

To run the EMERALD simulator:

1. Open the user interface for the web-based modeling tool listed above.
2. Select “Download” -> “Solve Engine” from the menu (see Figure 7).

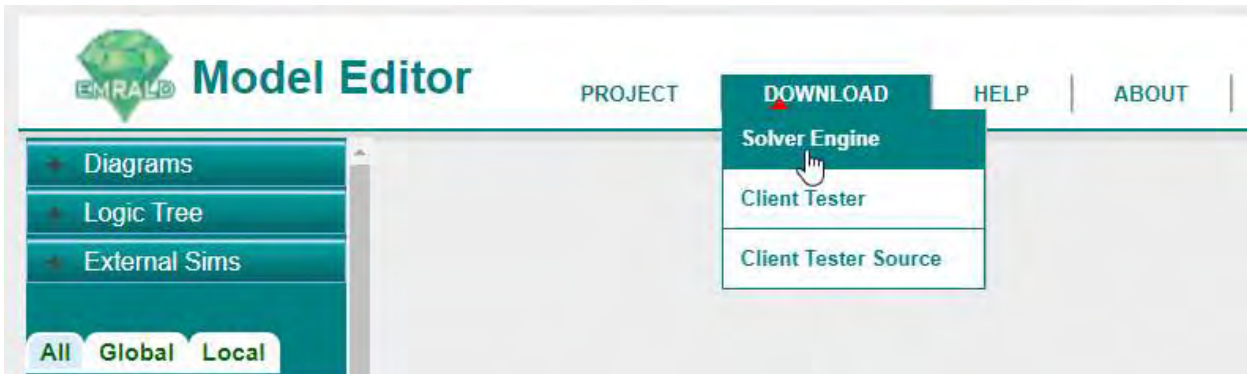


Figure 7. Download menu option to obtain the solve engine for running locally.

3. Save and unzip the file to the desired work folder
4. Open a Windows File Explorer to the unzipped location
5. Right-click and run “EMRALD_Sim.exe”

4.2 EMERALD Modeling Pieces

EMERALD modeling is based on a three-phase discrete event simulation where there is no fixed time stepping, but the next thing to happen is the next step in time [8]. An EMERALD model defines what can happen to components, humans, or systems, along their interactions or effects over time. The model is run repeatedly to explore the most probable outcomes and what caused them. This section outlines the different modeling pieces of EMERALD and relates where they may be used for physical security modeling. For general modeling information, see the EMERALD documentation described in Section 4.1 or the videos posted at

<https://www.youtube.com/playlist?list=PLX2nBoWRisnXWhC2LD9j4jV0iFzQbRcFX>.

4.2.1 States in EMERALD

A state is a logical representation for the condition of a component, person, system, process, etc. If you are modeling the process of an operator performing a task, then each significant step in that task would be a different state. For example, if there is a task for manually turning on a pump, the states of the operator could be “OpAvaliable,” “OpGoingToSwitch,” “OpTurnOnPump,” and “OpGoBack”. If you are modeling an adversary target such as a pump, then you may have the states “PumpOK” and “PumpDamaged.” Each of these states can have actions (see Section 4.2.2) that are performed when that state is entered and events (see Section 4.2.3) that monitor for their activation criteria can start other actions if that criteria is met. Depending on type of diagram used for modeling, states can also have a “True” or “False” value associated with it, which can be used to evaluate system combinations. For example, if three pump components are modeled as targets and all three have a “True” value for “PumpOK,” and a “False” value for “PumpDamaged,” adding the components’ “Logic Tree” (see section 4.2.6) could determine if all three pumps failed.

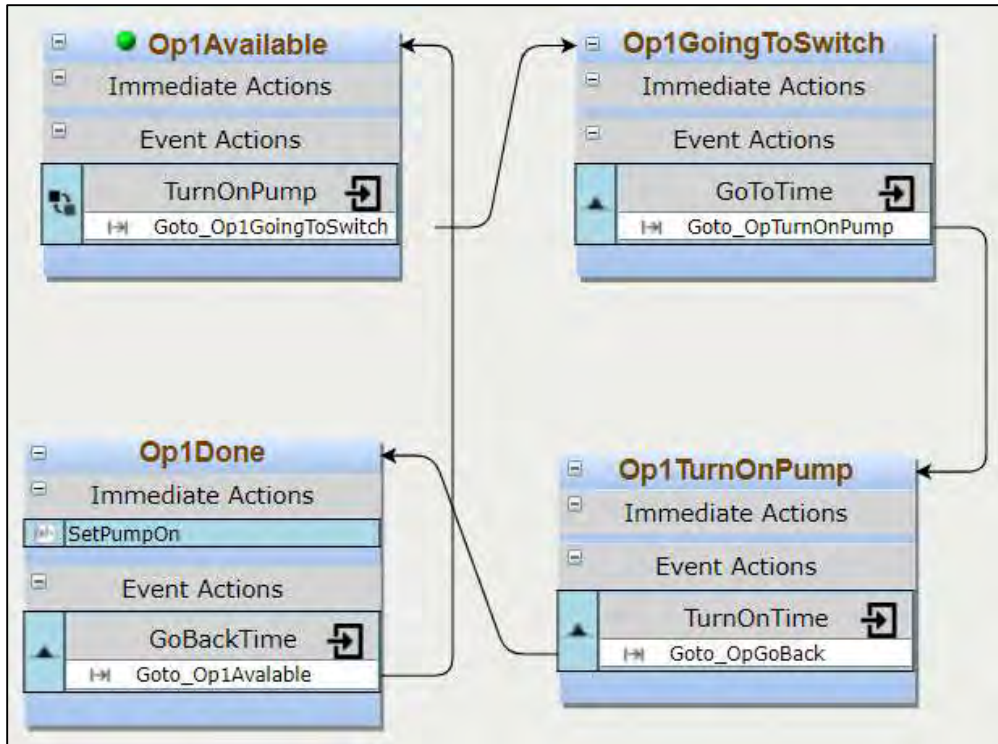


Figure 8. Example states for a set of operator actions to turn on a pump.

4.2.2 Actions in EMRALD

There are several types of actions available in an EMRALD model.

- Transition – moves from one state to another or adds a new state to the current conditions. This will be used in all aspects of modeling.
- Change Var Value – alters the value of a variable according to user defined parameters. This can be used to alter the value of variables, such as one used in a FOF simulation tool.
- Run Application – runs an external code to determine a change in the EMRALD model. This action will be used to run a FOF code such as Simajin or AVERT. A preprocess script is executed to make any modification or set parameters before executing. Then, a post-processing script can be used to evaluate results of the simulation and change states if desired.

4.2.3 Events in EMRALD

Events monitor for specified criteria and can have one or more actions that are executed when their criteria are met. When a state is entered, all the events for that state are then monitoring for their trigger; if a state is exited, then all those events are inactive and cannot be triggered. There are multiple types of events; the following are most relevant for physical security modeling:

- Var Condition – evaluates a user defined script which can use the model variables. If the script returns “true,” the event is triggered; if “false,” it is not. Whenever any variables used in the script are modified inside the simulation, then the event is re-evaluated.
- State Change – event is triggered if the condition of the assigned states is met. For example, the “TurnOnPump” event in the “Op1Available” state in Figure 8 can be set to occur if a state is entered indicating there is primary pump damage.

- **Component Logic** – evaluates a logic tree to determine if the event is triggered. This makes it possible to evaluate complex combinations of components to determine the event. For example, a logic tree could be made for all the combinations of target sets. Then, one event could determine if any full target sets have been hit.
- **Timer** – is a fixed time event. After entering the state, this event is triggered when the specified time is up.
- **Distributions** – There are several distribution types available. These use the parameters specified to sample an event time from the distribution type desired. Distribution events are key to model human times to perform steps in a task. For example, the “GoToTime” event in the “Op1GoingToSwitch” from Figure 8 determines how much time it takes for the operator to travel to the switch.

4.2.4 Variables in EMRALD (Including Dynamic XML and JSON Variables)

Variables store values available to be read or written in different areas of the EMRALD model. There are a few different scopes for variables in EMRALD applicable for current physical security modeling.

- **Global** – a general variable to keep track of data in the model.
- **Document Link** – a document link variable is associated with the value in a file. This file type can be JSON, XML, or a text file. The link is made by assigning the file and a “Var Link,” which is the XPath, JSONPath, or regular expression to the location of the variable value in the document. The value of the variable is retrieved from the document and, whenever it is changed in the simulation, the file is modified to reflect the value. These variables are key to modifying a FOF model before it is run (see “Run Application” in Section 4.2.1 and Section 4.5).
- **Accrual** – perform an equation or adjust a variable based on the time spent in the specified states.

4.2.5 Diagrams in EMRALD

Diagrams represent a particular piece of the model and the various conditions or states this piece of the model can be in. A diagram contains multiple states with the events that can occur, actions that can be executed, and variables used. These all define how the simulation may sift through the diagram over time in the simulations.

The scope of the diagram indicates the type of diagram to use: Plant, Component, System, or Other. Besides categorization, the main difference between the diagram types is Component and System diagrams can only be in one active state at a time. For example, a pump would be a component diagram because it cannot be running and failed at the same time. Also, each of the Component and System diagram states can have a Boolean value associated with it for evaluation in logic trees.

4.2.6 Logic Trees

A logic tree in EMRALD is a visualization Boolean logic expression. “AND” and “OR” gates are combined to evaluate a combination of component diagrams. The value of the current state for each diagram is propagated up the tree and evaluated by a “Component Logic” event as shown in Figure 9.

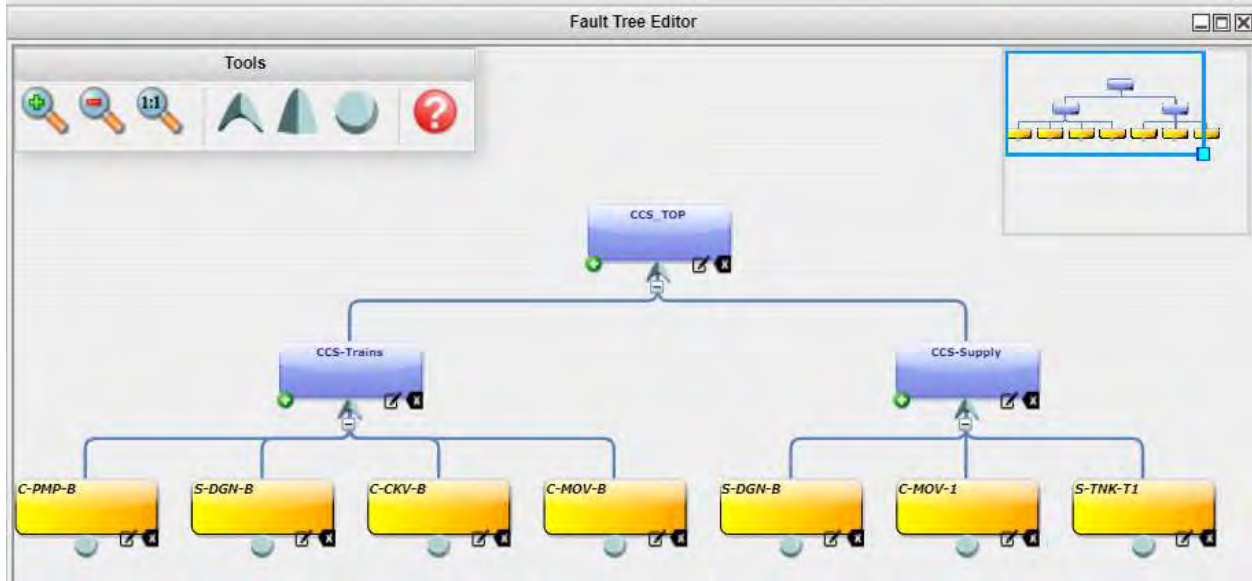


Figure 9. Example of a logic tree.

4.3 Modeling Operator Actions

One of the main aspects often missing from FOF modeling is including operator procedures. This section will outline the main process for developing an operator task and how it fits into the FOF model.

4.3.1 Example Procedure: FLEX Generator

A common operator procedure to model could be switching to the use of FLEX generators and will be used for the subsequent sections. For this example, it is assumed there are two flex generators pre-staged and two Emergency Diesel Generator (EDG) components have already been modeled (see Figure 14). The procedure steps are:

1. Verify the generators are available
2. Perform a “DC Load Shed;” non-essential equipment is turned off in the breaker panels
3. Switch the Motor Control Center (MCC) Breakers opened for FLEX use.

4.3.2 Diagram and Operator States

It is good practice to put each operator procedure in its own diagram. Create a new diagram by right clicking on the diagram list in the left panel and select “New Diagram”. Then use “Other” for the diagram type and fill in the name and description (“Desc”), as shown in Figure 10.

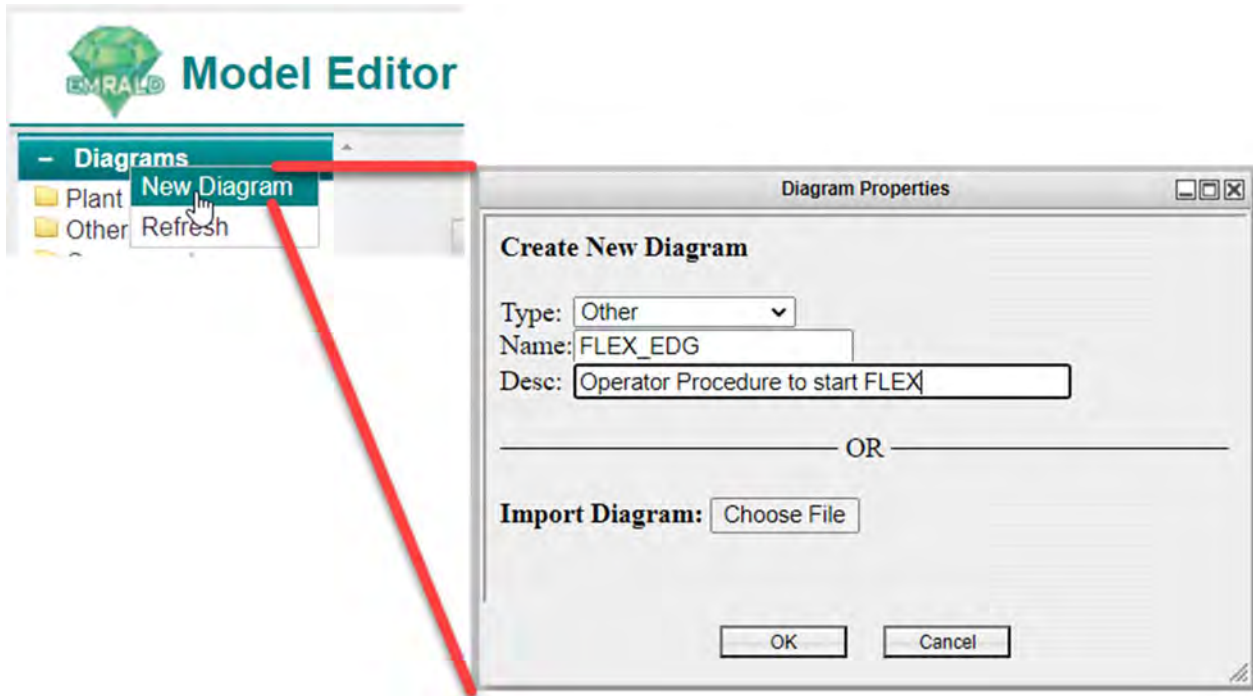


Figure 10. Adding a new diagram for the FLEX procedure.

A new empty diagramming area opens after clicking “OK.” The next step is to determine the likely states that will be needed for the diagram. To do this, start with the steps of the procedure and add them as states in the diagram. To add a state, right-click in the diagram and select “New State,” and enter the name. For this case, the names are “Check_FLEX_EDG,” “DC_Load_Shed1,” and “MMC_Breakers1. After the procedure is done, to know if the operator successfully got an EDG, an additional state called “FLEX_EDG_Running” is created. These states are shown in Figure 11.

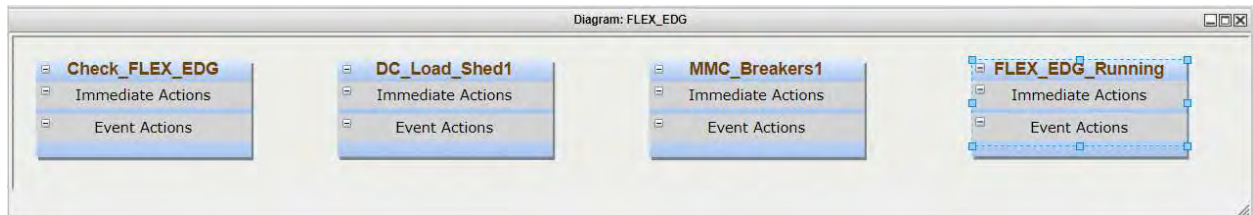


Figure 11. States for the FLEX procedure.

4.3.3 Determining Event and Actions

Once the states are added, events and actions need to be created. In the “Check_FLEX_EDG” state (see Figure 16), it is necessary to model the evaluation of the diesel generator (DG) availability. To do this, there is a variable set by the physical security simulation that indicates availability, and two “Var Condition” events are added: if available and if unavailable. One of the two events will occur and a transition action needs to move from the “Check_FLEX_EDG” state to either the “Plant_Damaged” state (defined in the main diagram) or to the next step, which is “DC_Load_Shed1.”

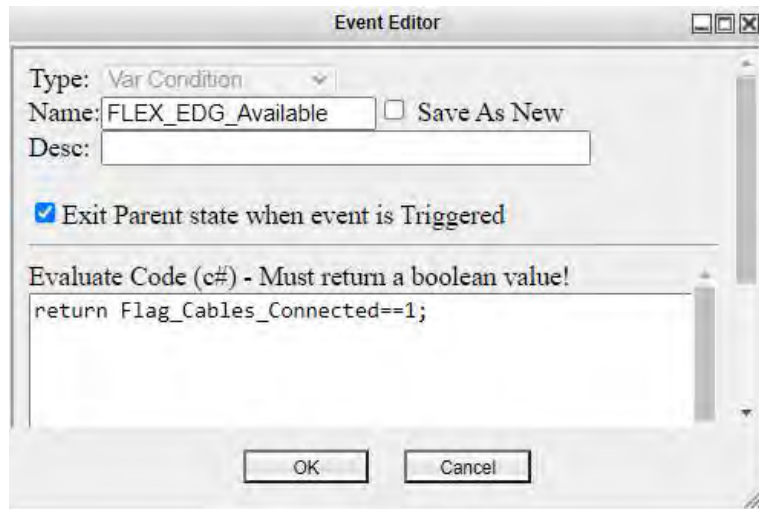


Figure 12. Variable condition event to determine availability.

The next task is for the operator to shed the load of non-essential equipment. This is a matter of time and can be done according to a distribution determined through testing and statistical data. For this case, a normal distribution with a mean of 10 minutes, standard deviation of 1 minute, minimum time of 1 minute, and maximum of 50 minutes is used, as shown in Figure 13. This event also has a transition action that moves to the next state “MCC_Breakers1” (see Figure 16). The event in “MCC_Breakers1” is also a normal distribution representing how long it takes the operator to switch the breakers to the FLEX EDG and has an action to move to “FLEX_EDG_Running” (see Figure 16).

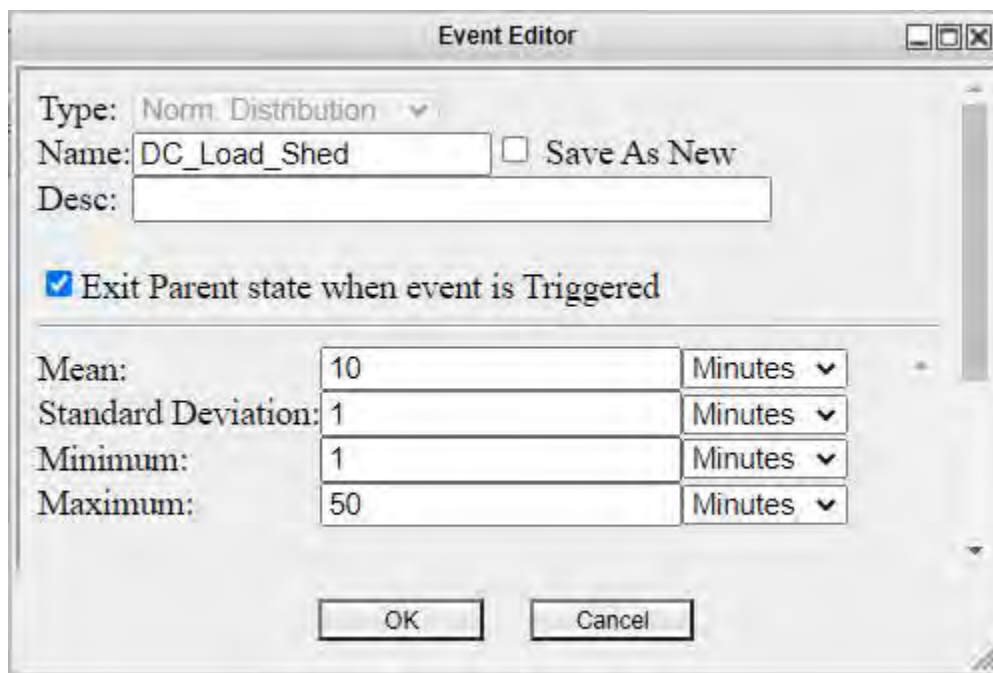


Figure 13. Normal distribution event for how long it takes to do a load shed.

The final step after the operator has performed his tasks is to evaluate if everything in the process went OK and make sure the generators run as long as needed. The “FLEX_EDG_Running” state captures this using an immediate action and three events. An immediate action is used to set a variable to save the

time the EDG are being started. Additionally, when the “FLEX_EDG_Running” state is entered, an event in the EDG diagrams is triggered which moves their states from standby to either active or failed, shown by the highlighted area in Figure 14.

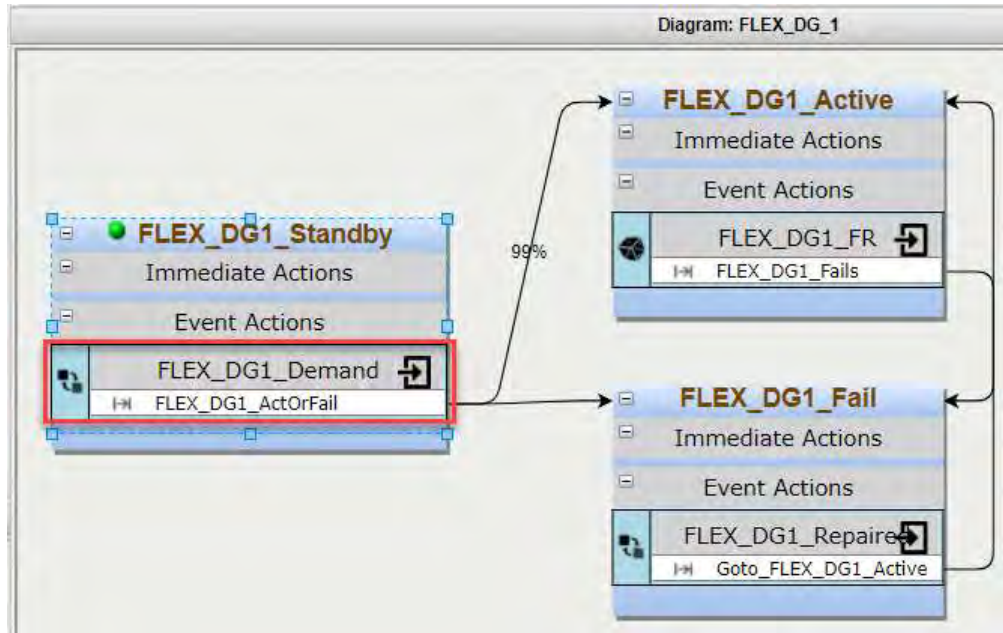


Figure 14. Diagram for a FLEX diesel generator.

Then, the first event in “If_FLEX_DG_Fail” (see Figure 16), a “Component Logic” event, evaluates the already modeled EDG components for the duration of the simulation and, if both fail, then an action moves to the “Plant_Damaged” state. The next event evaluates if the time it took to perform the FLEX procedure is greater than 60 minutes. Then, an action also moves to “Plant_Damaged” state. The final event “If_FLEX_Timely” is triggered if the FLEX procedures were done in time (i.e., < 60 minutes). This event action must not have the “Exit parent state when event is triggered” property checked, shown in Figure 15, so the simulation will stay in this state and continue to monitor whether the EDGs are running.

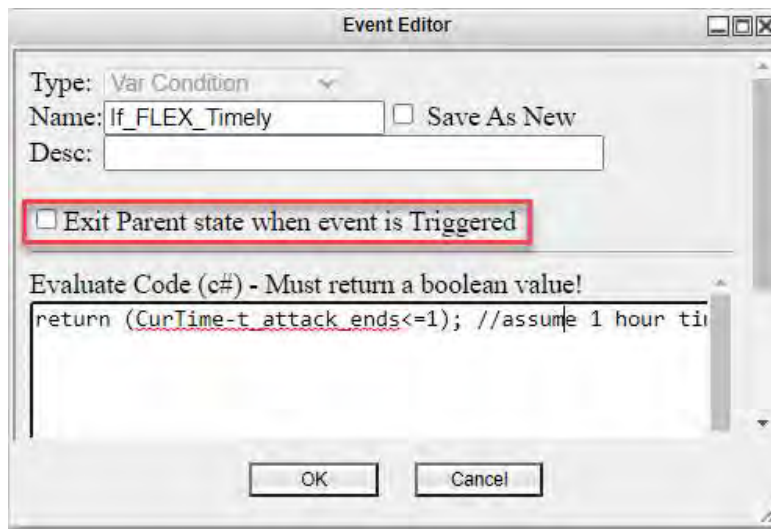


Figure 15. Event property to indicate if exiting the current state after the event occurs.

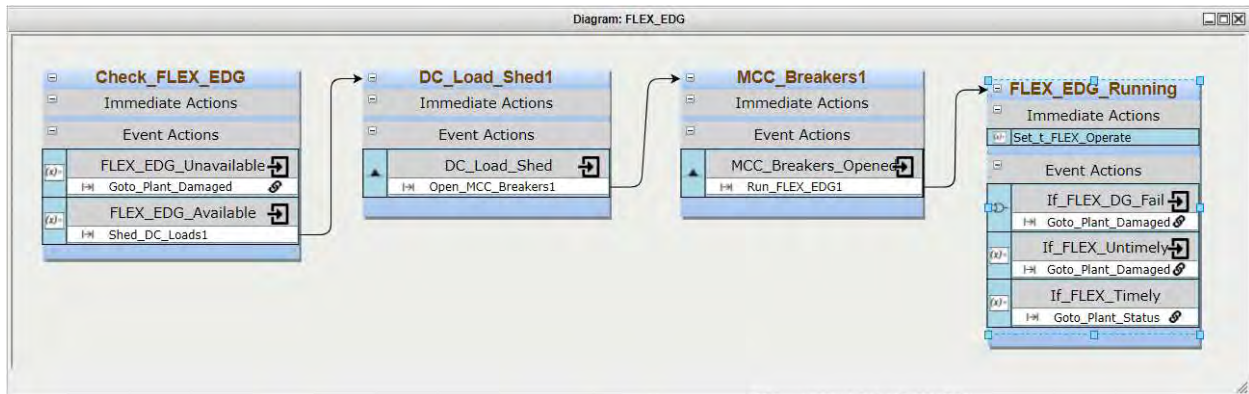


Figure 16. Events and Actions for the FLEX procedure of switching to EDG power.

4.3.4 Triggering the Task and Upon Completion or Failure

Simulating the modeled operator action procedure is only needed if a certain condition of the plant arises, namely a station blackout. This condition is determined by the main diagram; in this case, there is a state called “Assess_Plant_Condition.” Here, there is the event “Grid_EDG_Damaged;” if this event occurs, then a transition event specifies the start of the “Check_FLEX_EDG” state, as shown in Figure 17.

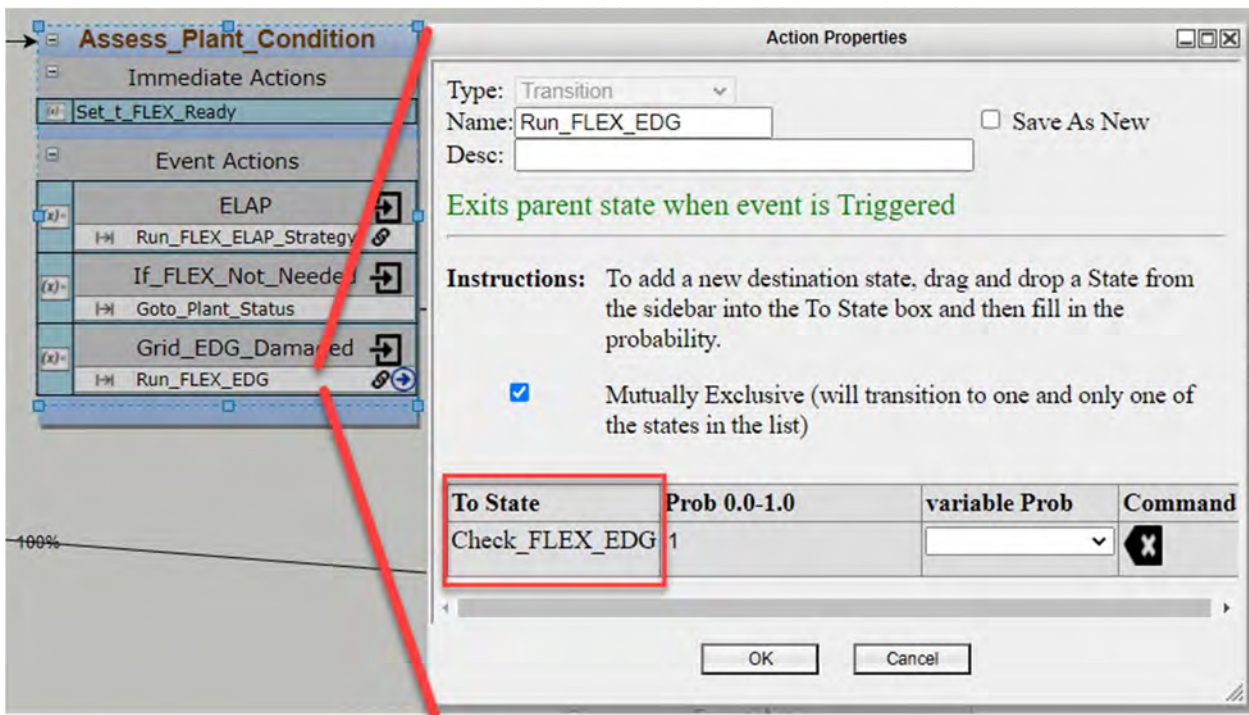


Figure 17. Event and action triggering the transition to start the FLEX EDG procedure.

As stated, if one of the events, “FLEX_EDG_Unavailable,” “If_FLEX_DG_Fail,” or “If_FLEX_Untimely” occur, then they cause the transition to the “Plant_Damaged” state which is in the main diagram, (see Figure 18). If the “If_FLEX_Timely” event occurs, then it transitions to the “Safe_Shutdown” state, also shown in Figure 18. These are “Key” state and is monitored for the results.

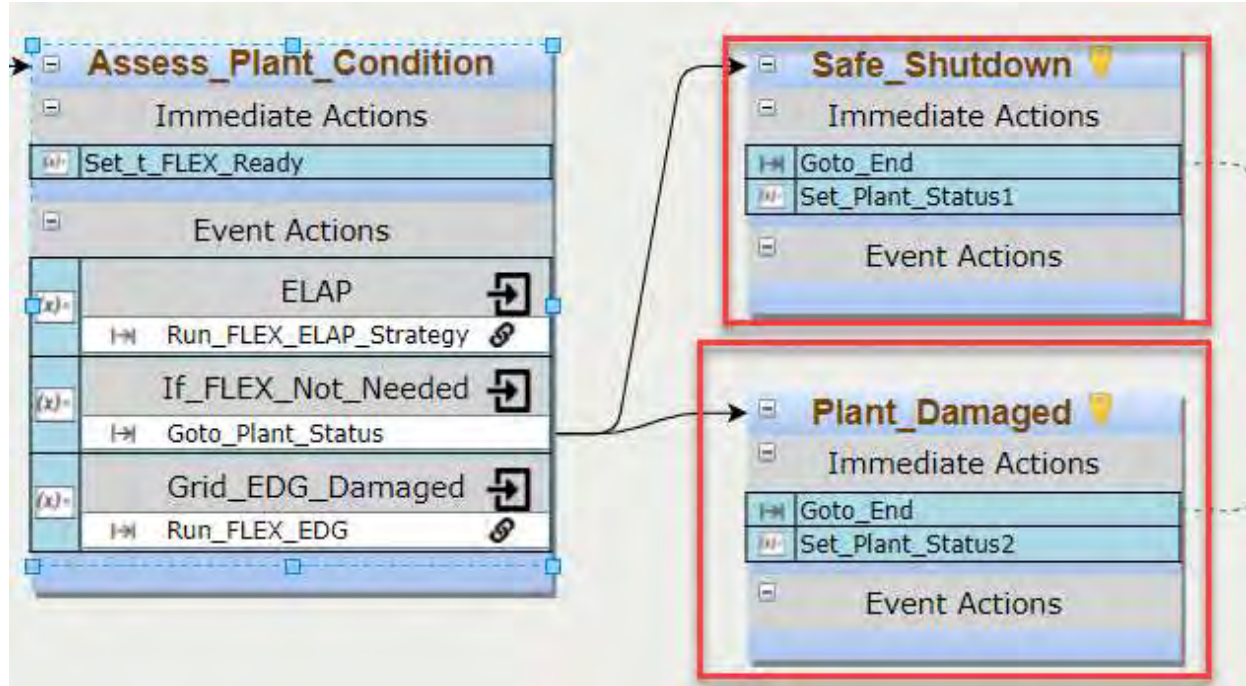


Figure 18. The "Plant_Damage" state tracks the times this condition occurs.

4.4 Integration of FOF Tool with EMRALD

The process and modeling requirements for integrating with a FOF tool depends on the built-in capabilities of the tool and what level of realism the user tries to achieve in the model. The following sections describe the integration that has been done for SCRIBE3D, AVERT, and Simajin.

4.4.1 EMRALD-SCRIBE3D

SCRIBE3D features a means of data communication with external software through the use of a text JavaScript Object Notation (JSON) file. The coupling between EMRALD and SCRIBE3D is done through this json file.

An example of the integration between SCRIBE3D and EMRALD is given in this section, which involves an attack scenario of sabotaging EDGs, the electrical system, and safety-related equipment. This attack scenario is conducted by three adversaries, and the attack path is shown in Figure 19 below. As AVERT was initially designed as a tabletop tool, the EMRALD model is complex and is used to adjust many aspects of the model such as attack times and objectives. The nuclear power plant model in this example is a hypothetical one and does not represent any actual operating facilities in existence. It is the same model used for in training exercises and workshops organized by the Sandia National Laboratories, for which a description is available online [2, 9].



Figure 19. LonePine Facility Layout in SCRIBE3D [2].

The SCRIBE3D simulation is controlled by an external JSON text file which contains simulation variables, such as the time to breach the fence, the time to sabotage the target, and visibility. This file is adjusted by EMERALD before each simulation run. SCRIBE3D returns a JSON output file in the same format as the input json file, which EMERALD reads to extract key output variables in the simulation. Contents of the JSON input file for this example are given in Appendix A. A partial snapshot of this JSON script is shown in Figure 20. The data includes the total simulation time (SimTime), the time when the simulation ends (StopAtTime), at which objectives did the simulation stop (StopAtObjectives), the reason for the simulation to stop (StopReason), and the parameters for individual players. These individual parameters include the objectives for each player (Objectives), the status for each objective whether it has been completed, the delay time required to finish that objective, and whether the objective is to be done or be skipped. It also returns the status of whether the player is alive and the time when the player is neutralized. These objectives are different for each player. In this hypothetical attack example, the player Adversary_1 has a mission to sabotage all the target sets meanwhile the other two adversaries provide support.


```
{
  "SimTime": 0.0,
  "StopAtTime": 0.0,
  "StopAtObjectives": [],
  "StopAfterEngagement": false,
  "LastEditedBy": "Scribe",
  "StopReason": "StoppedAtTime: 0.0",
  "Exception": "",
  "EntitiesList": [{
    "Name": "Adversary_1",
    "Objectives": [{
      "Name": "A_1 Breach Outer PIDAS",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_1 Breach Inner PIDAS",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }
  ]
}
```

Figure 20. Sample of SCRIBE3D JSON file.

The main EMERALD diagram for this example is shown in Figure 21 below. The model starts from the initialization of simulation parameters. In this simple example, only the adversary's visibility was initialized in the StartAttack state. The Breach state is activated after the StartAttack state. The intermediate action named CalcPidasTime in this state directs the simulator to a sub-diagram shown in Figure 22. This sub-diagram models the adversary actions to breach the perimeter intrusion detection and assessment system (PIDAS). It samples the time to do each action including breaching the outer fence, crossing the PIDAS area, and breaching the inner fence from a set of probability distributions. It also estimates the probability of adversaries triggering the PIDAS alarm as shown in Figure 23. If the alarm is triggered, it alters the probability distribution for adversaries to complete further tasks. This sub-diagram saves the timing data to breach the outer and inner fences in OuterFenceTime and InnerFenceTime variables, respectively. When the simulation exits the PidasBreached state, it activates the GetPidasTime event as shown in Figure 24, and the simulation control returns to the main diagram.

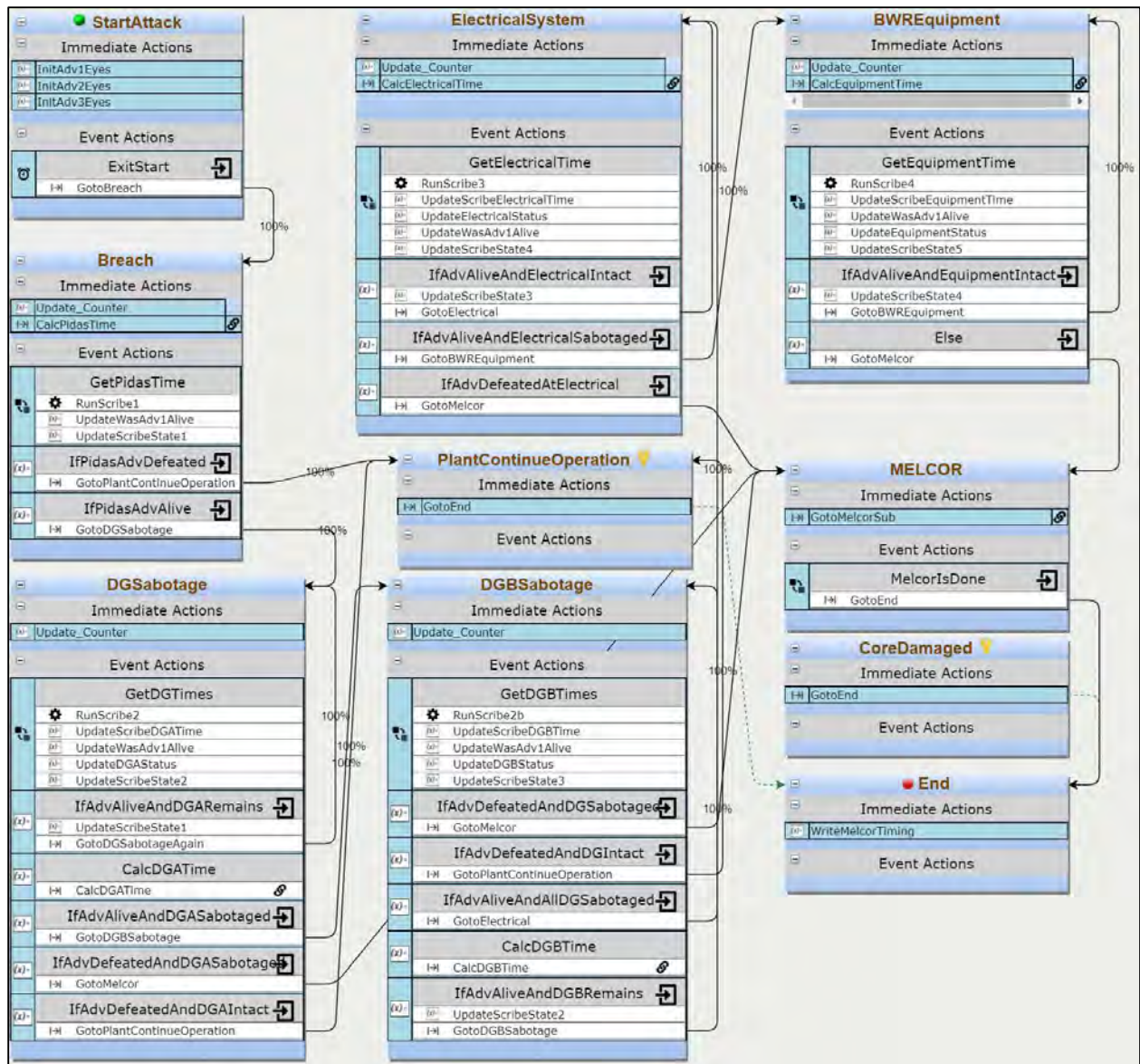


Figure 21. Main EMRALD diagram to integrate EMRALD and SCRIBE3D.

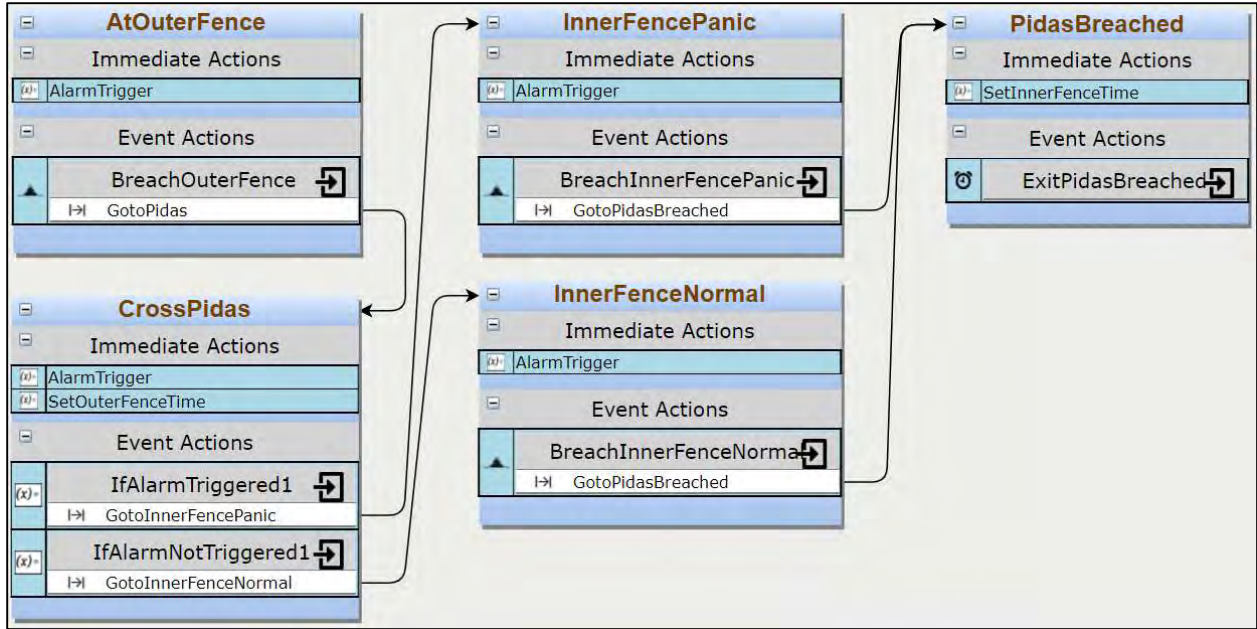


Figure 22. Sub-diagram of adversary tasks at PIDAS.

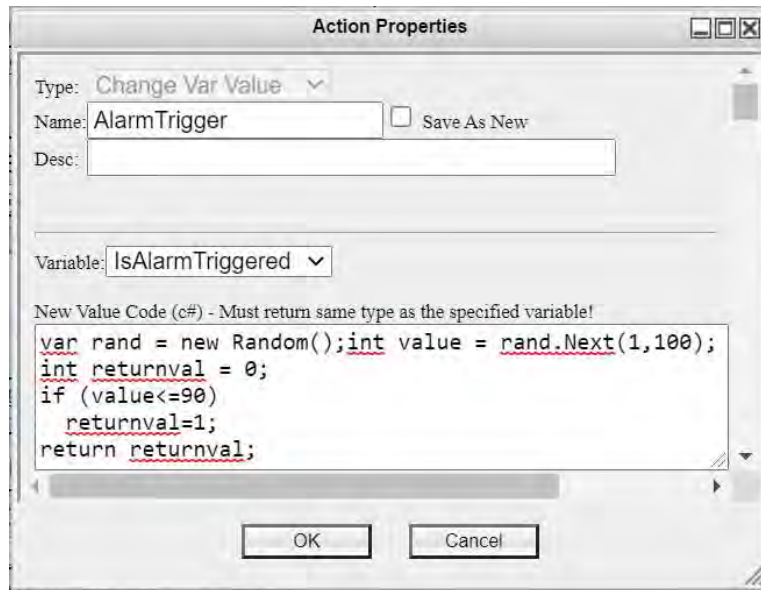


Figure 23. AlarmTrigger action to set a probabilistic PIDAS alarm.

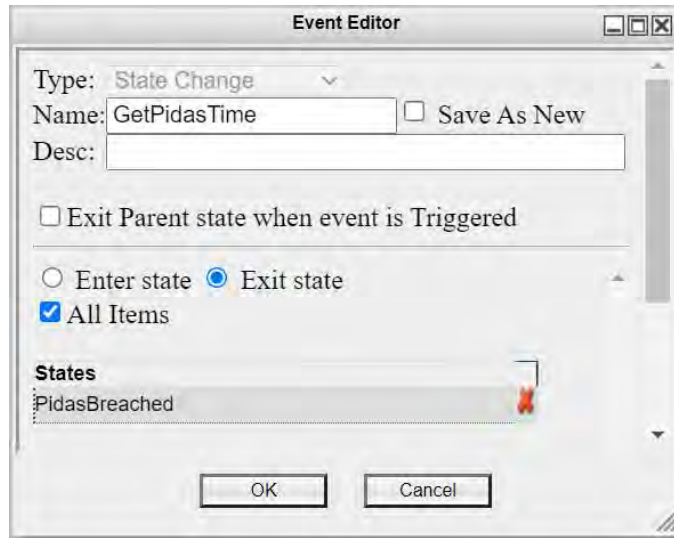


Figure 24. GetPidasTime event.

When the GetPidasTime event is triggered, it runs the RunScribe1 action. This action writes the OuterFenceTime and InnerFenceTime variables obtained from the PIDAS Breach sub-diagram. The C# preprocess and postprocess codes of this action are shown in Figure 25. The preprocess code works by deleting the output file from a previous simulation, if any. It then reads the JSON template format from the InitConfig.json file and replaces the WaitTime for adversaries to breach the outer and inner fences with the OuterFenceTime and InnerFenceTime variables. Because the WaitTime is a delay period, the WaitTime for the inner fence is the time difference between InnerFenceTime and OuterFenceTime. The code also sets the StopAtObjectives field as A_1 Breach Inner PIDAS, so that the SCRIBE3D simulation will end when the first adversary completes the PIDAS breach. Furthermore, it also sets the StopAtTime at 5000 seconds. This value was taken arbitrarily to give SCRIBE3D a sufficient time to complete the PIDAS breach simulation. The modified JSON data is then saved to a text file named config-1.json. The preprocess code returns the command line parameter to execute SCRIBE3D containing the location and name of the config-1.json file. In this example, the input JSON file is named InitConfig.json, and the output file from SCRIBE3D is named apiOutput.json.

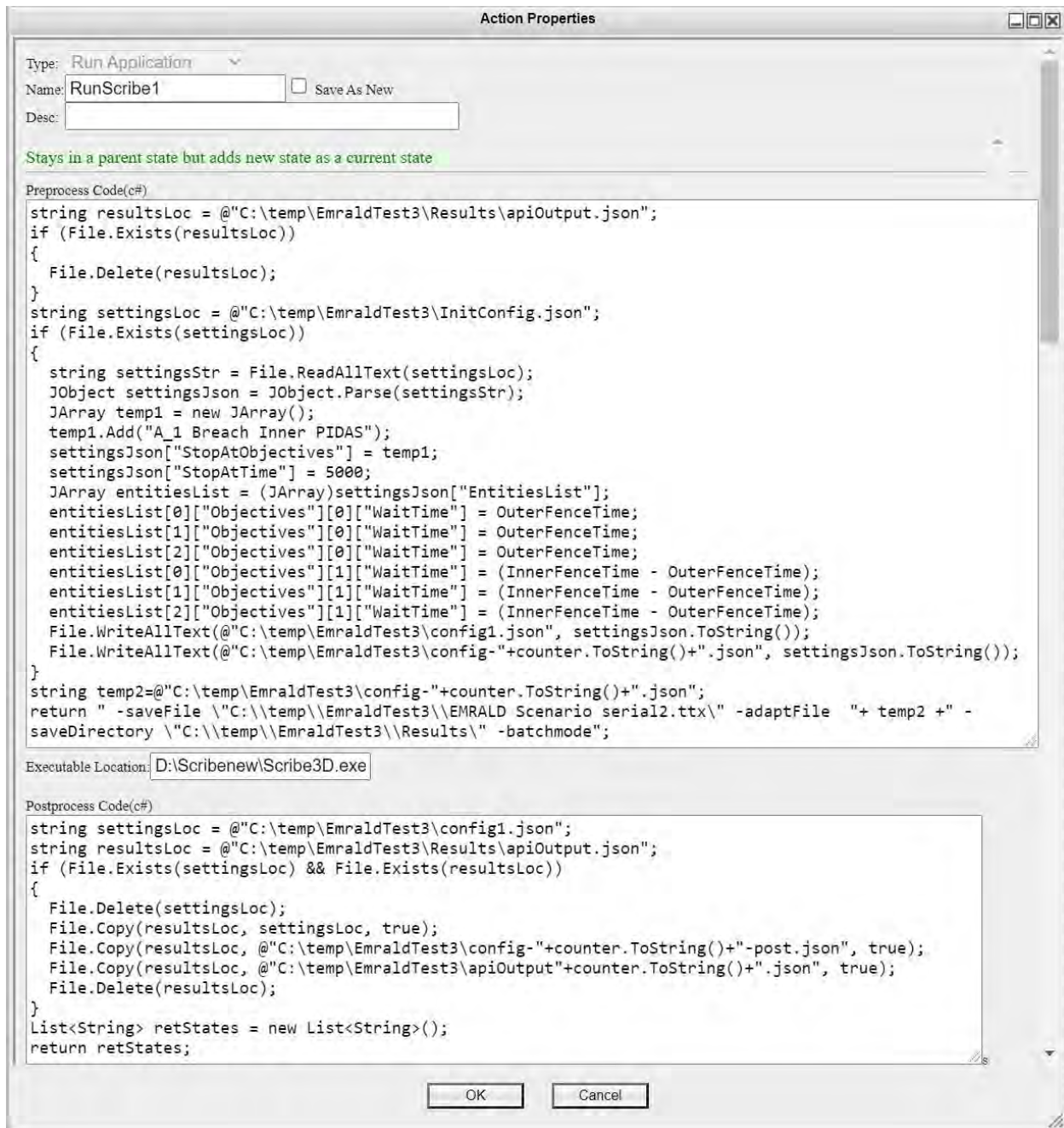


Figure 25. C# codes for RunScribe1 action.

The preprocess code executes the command line argument for SCRIBE3D, whose location is defined in the Executable Location box. The postprocess code is executed when SCRIBE3D has finished running. It fetches the output file, apiOutput.json, and copies it to a new location with a new name. It returns an empty set of states simply as a means to continue the simulation flow. The output file updates EMERALD variables—namely IsAdv1Alive, IsAdv2Alive, and IsAdv3Alive. These variables are then used as conditions to determine the activation of next events in the state, namely IfPidasAdvDefeated and IfPidasAdvAlive, as shown in Figure 26 and Figure 27. If adversaries are neutralized at the PIDAS, then the plant may resume operation as normal. However, if the adversaries made it through the PIDAS alive, the simulation continues to the DGSabotage state.

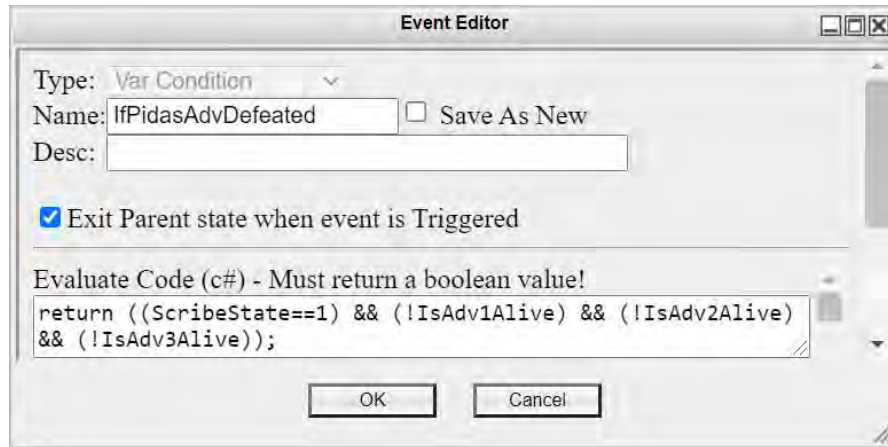


Figure 26. IfPidasAdvDefeated event.

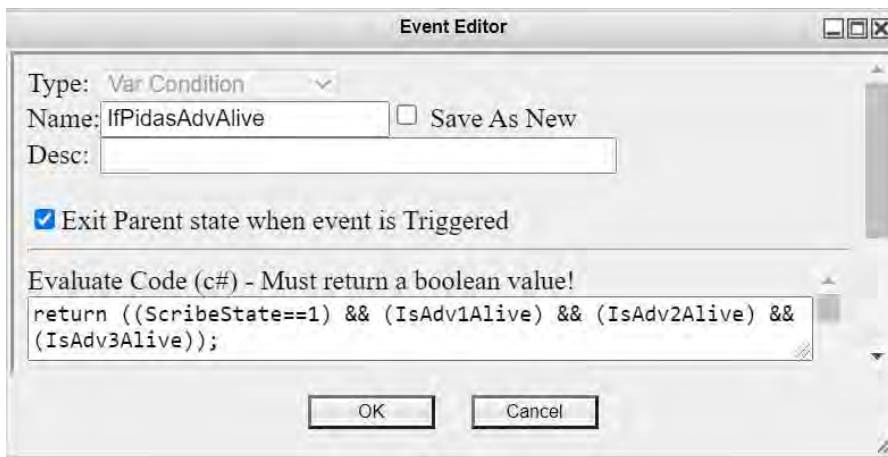


Figure 27. IfPidasAdvAlive event.

The DGSabotage state transfers the simulation to another sub-diagram. It works in a similar manner as the Breach state. The sub-diagram models the adversary actions in further detail when sabotaging the first diesel generator and returns the timing information. The time required to sabotage the generator is then fed to the SCRIBE3D input file as shown in Figure 28. The figure shows the preprocess and postprocess C# codes of the RunScribe2 action. EMERALD runs SCRIBE3D, fetches its output file, and reads the end state of the SCRIBE3D simulation. After the RunScribe2 action is completed, EMERALD reads the SCRIBE3D output file to extract the timing information on when Diesel Generator A is sabotaged, as shown in Figure 29. If the adversaries sabotage the equipment successfully and they are still alive, the simulation continues to the DGBSabotage event to run the next step of SCRIBE3D. If the equipment is sabotaged, but the adversaries are defeated, then the simulation continues to the MELCOR state to run MELCOR and estimate whether the reactor will be damaged or not.

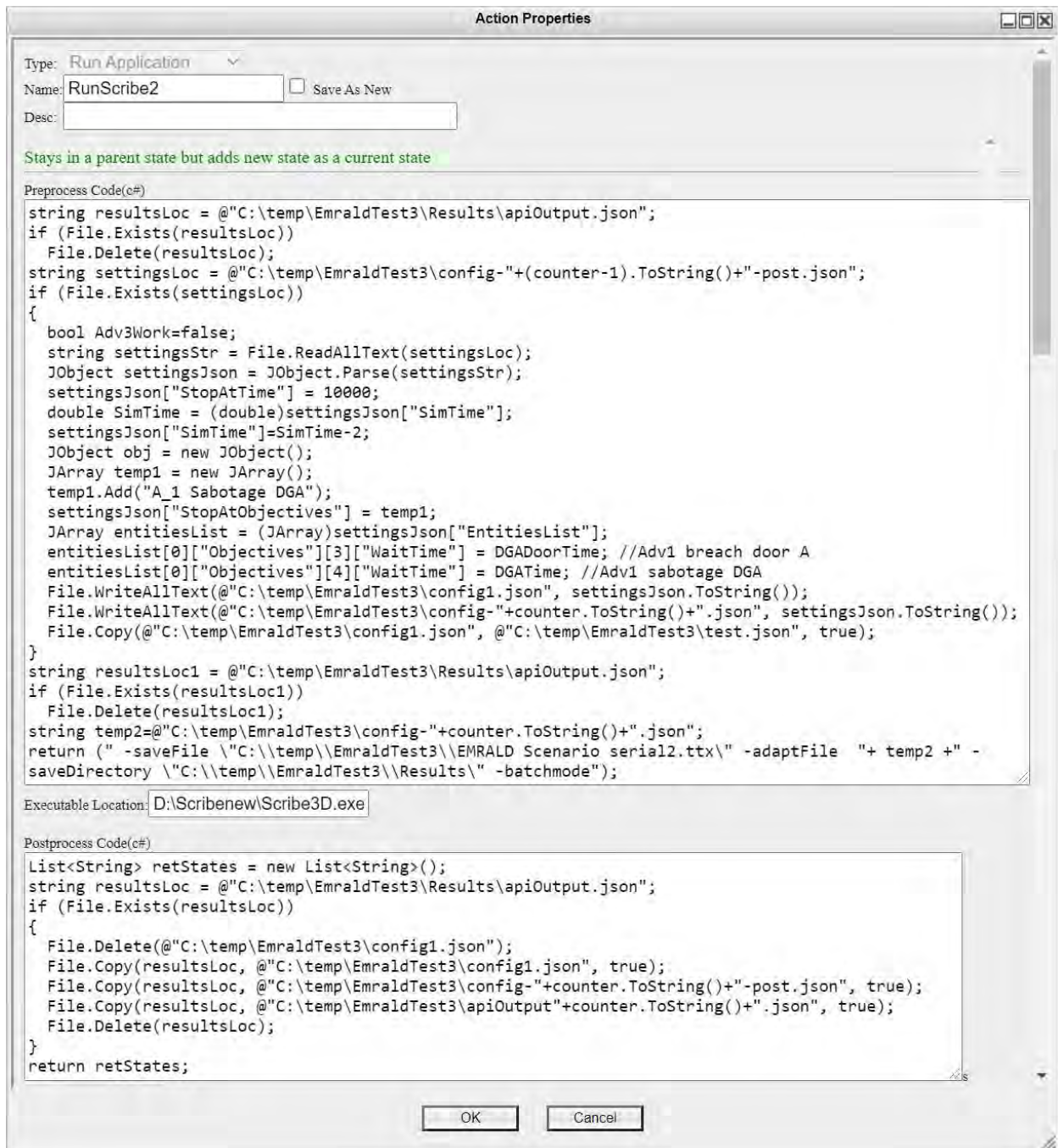


Figure 28. RunScribe2 action.

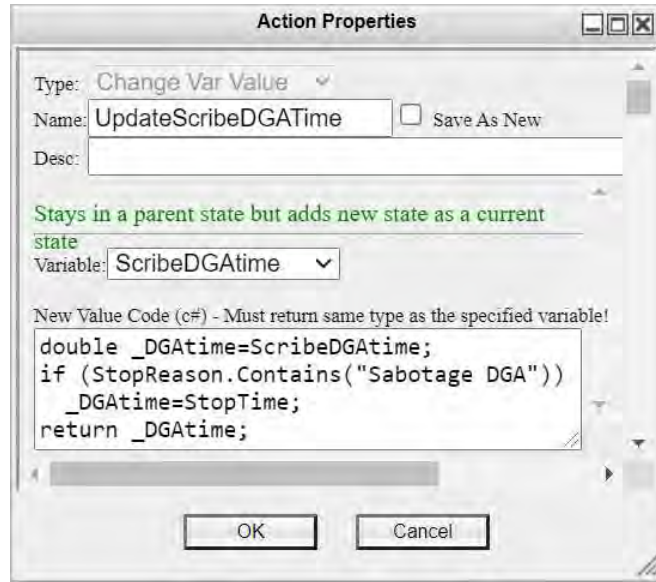


Figure 29. UpdateScribeDGATime action.

The SCRIBE3D simulation is run in a piece-wise manner to extract the timing information of when each target is sabotaged. This objective cannot be accomplished if SCRIBE3D is executed in a once-through fashion. The WaitTime in each adversary's Objective does not take into account the travel time for adversaries to move from one objective to the other. Therefore, the time when an equipment piece is sabotaged needs to be taken from the StopTime output field. Since there is only one StopTime field in the output file, the simulation needs to be run in several stages by stopping it when a target is sabotaged, read the StopTime data, repopulate the JSON input file, and re-run SCRIBE3D starting from that stage. The collected timing data may be forwarded to a thermal-hydraulic analysis input file to determine whether the reactor is damaged, given the result of the attack scenario.

4.4.2 EMERALD-AVERT

An example of EMERALD-AVERT integration is given in this section. The main EMERALD diagram for its integration with AVERT is shown in Figure 30. In this model, the AVERT simulations were run externally using a batch file. After all of the FOF simulations are completed, EMERALD analyzes the output files and models the corresponding operator actions. The process starts from the Start State in which all the variables are initialized. It then proceeds to the Read_Avert state to extract selected variables from the AVERT output file. The C# codes to perform these actions are shown in Figure 31 to Figure 38. AVERT returns the FOF simulation output in a text file, listing all the objectives both adversaries and protective players achieve. Therefore, the EMERALD variables are queried by searching for specific objective keywords in AVERT's output file.

The output variables determine which event in the Read_Avert state takes place. The If_Plant_Sabotaged event is executed if the Flag_Prepare_FLEX Boolean variable is true. This event transfers the simulation flow to the FLEX_Preparation state. Meanwhile, if the Flag_Prepare_FLEX returns false, the If_Plant_Intact event is executed instead, and the simulation continues to the Plant_Continue_Operation key state. The integration between EMERALD and AVERT ends at this state. The next states model the FLEX actions and the time it takes to complete them based on a predetermined plant operational procedure.

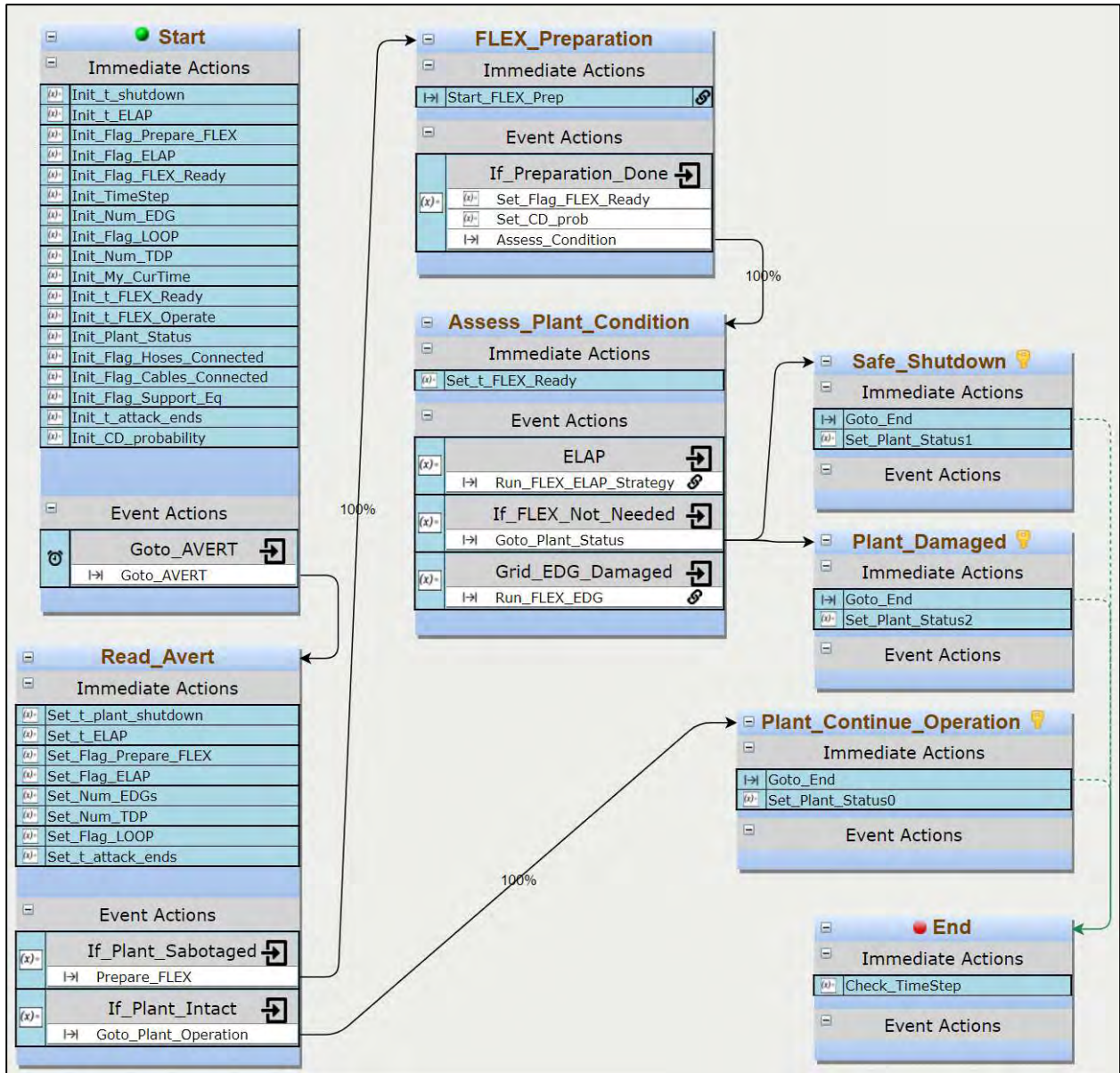


Figure 30. Main EMRALD diagram.

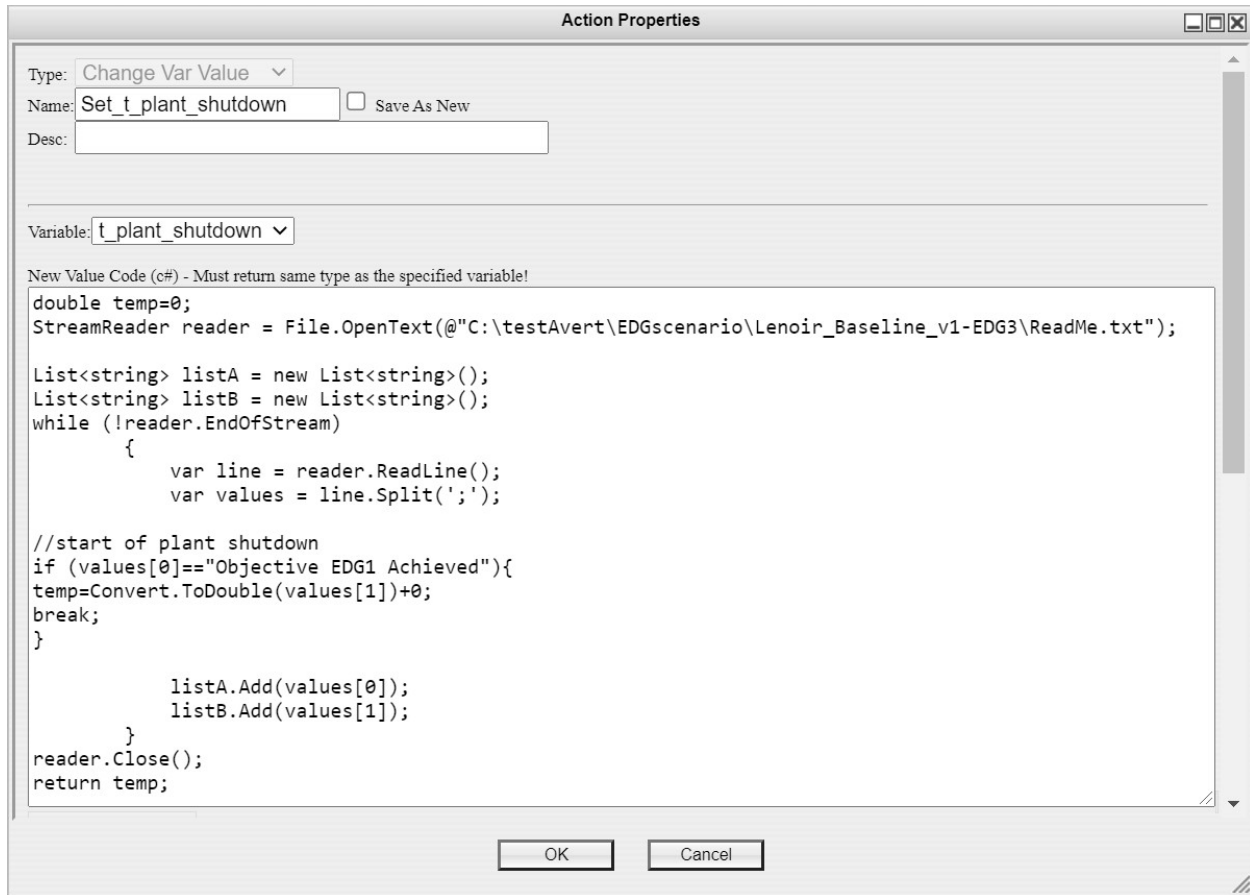


Figure 31. C# code to set the plant shutdown timing.

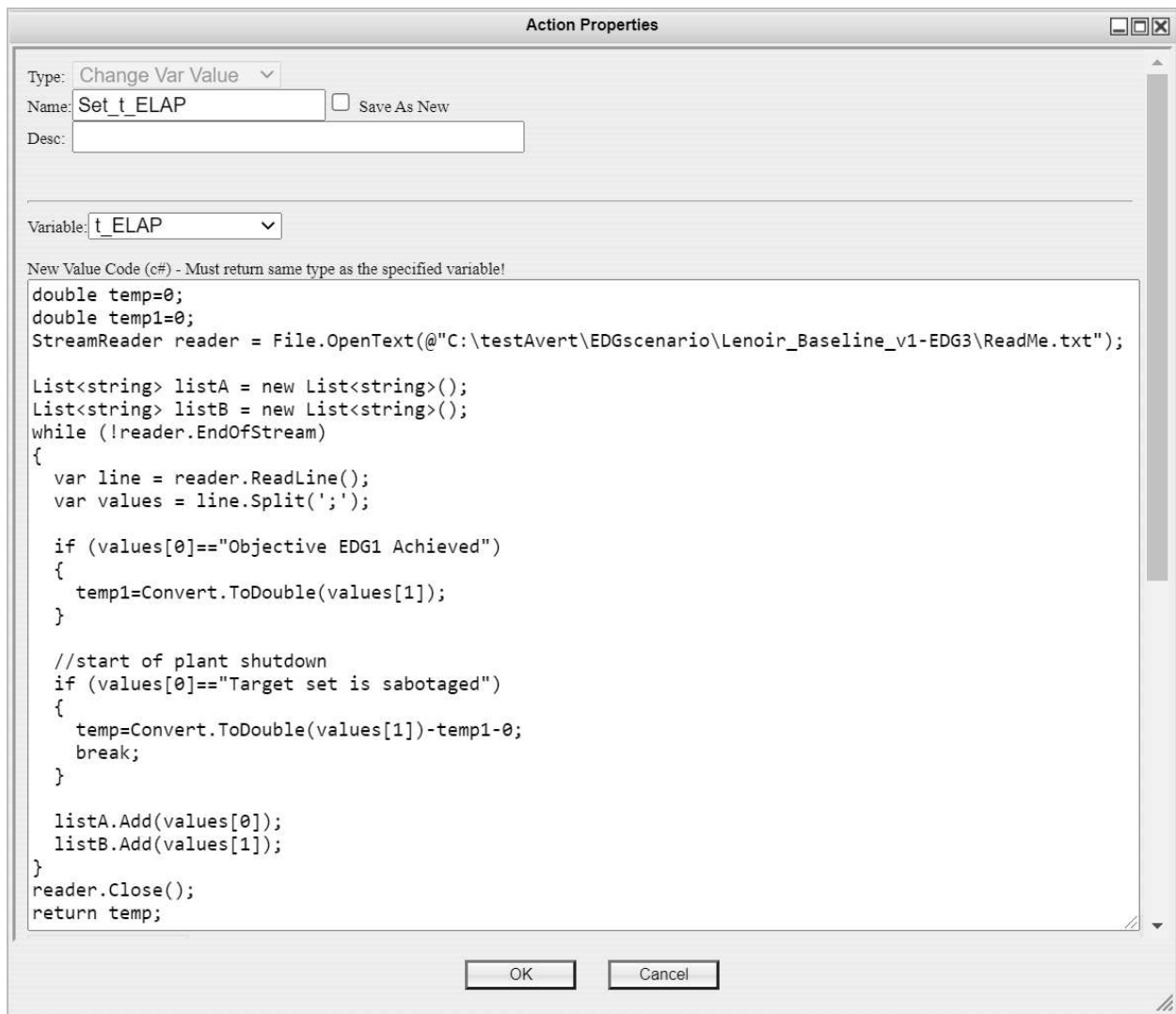


Figure 32. C# code to determine the timing when all target components are sabotaged.

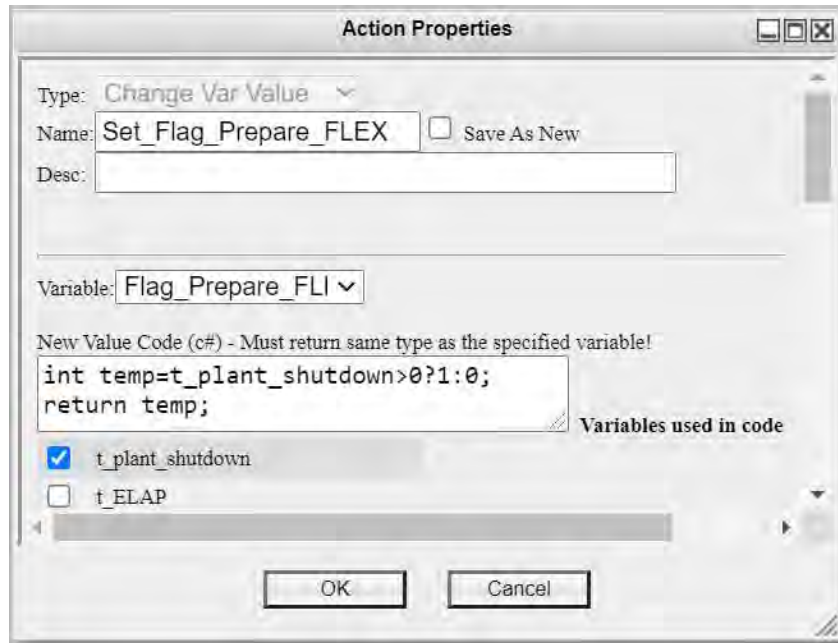


Figure 33. C# code to determine FLEX usage.

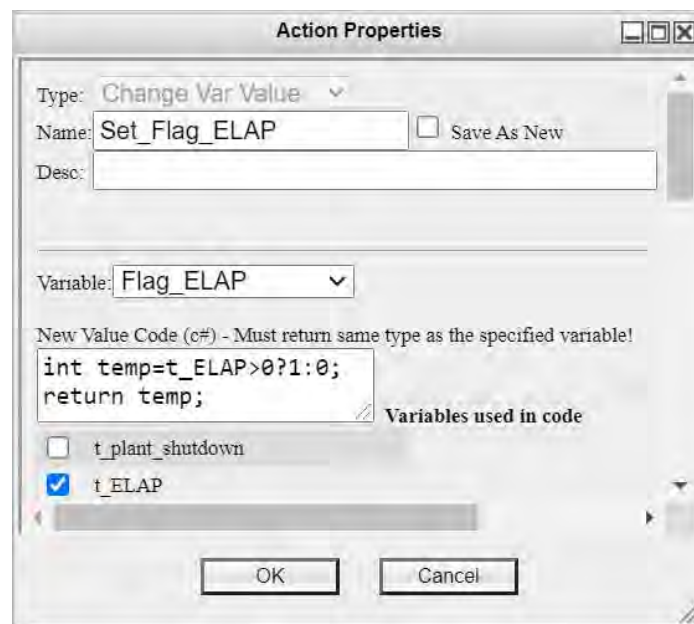


Figure 34. C# code to determine if all targets are sabotaged.

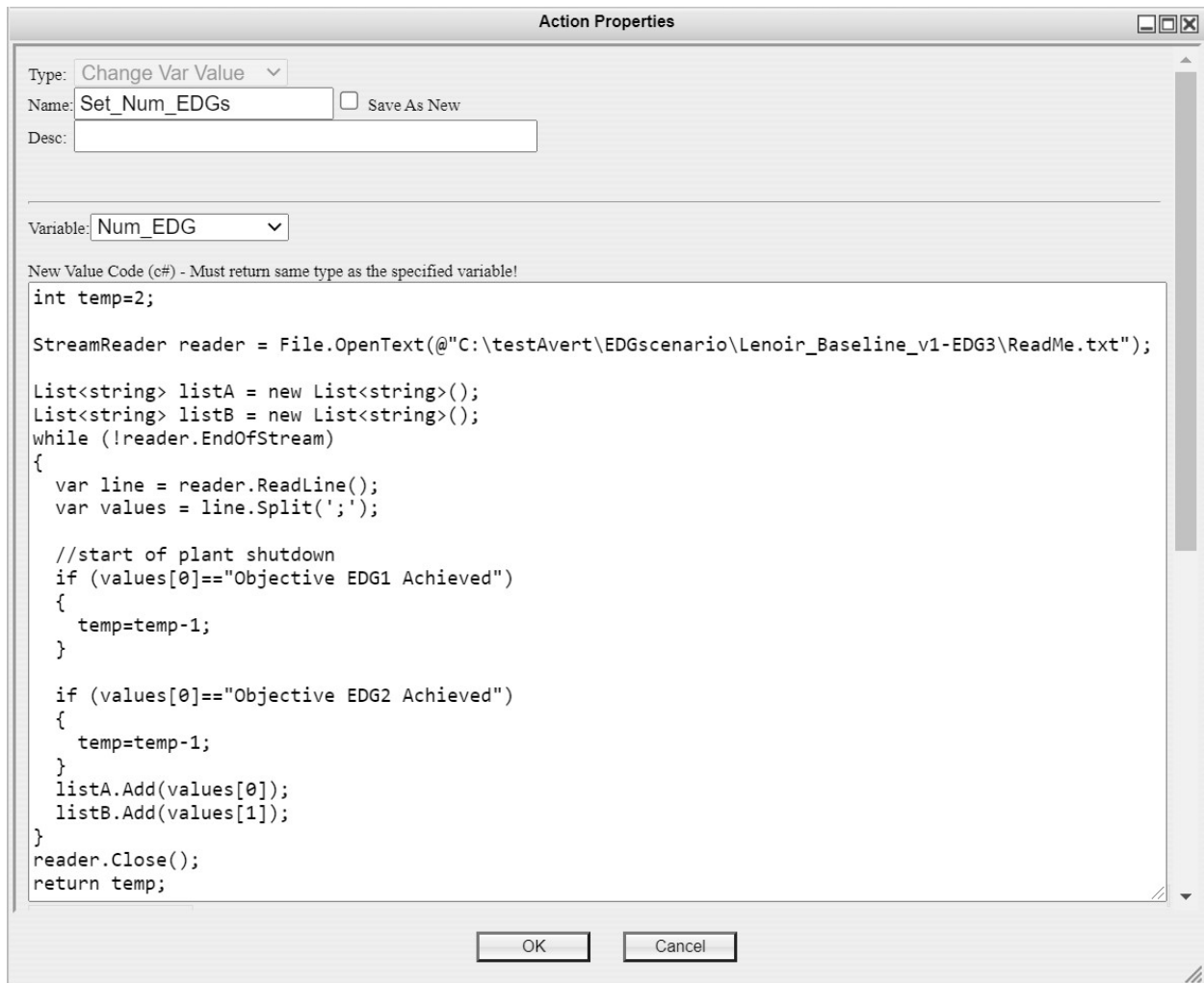


Figure 35. C# code to count the number of intact EDGs.

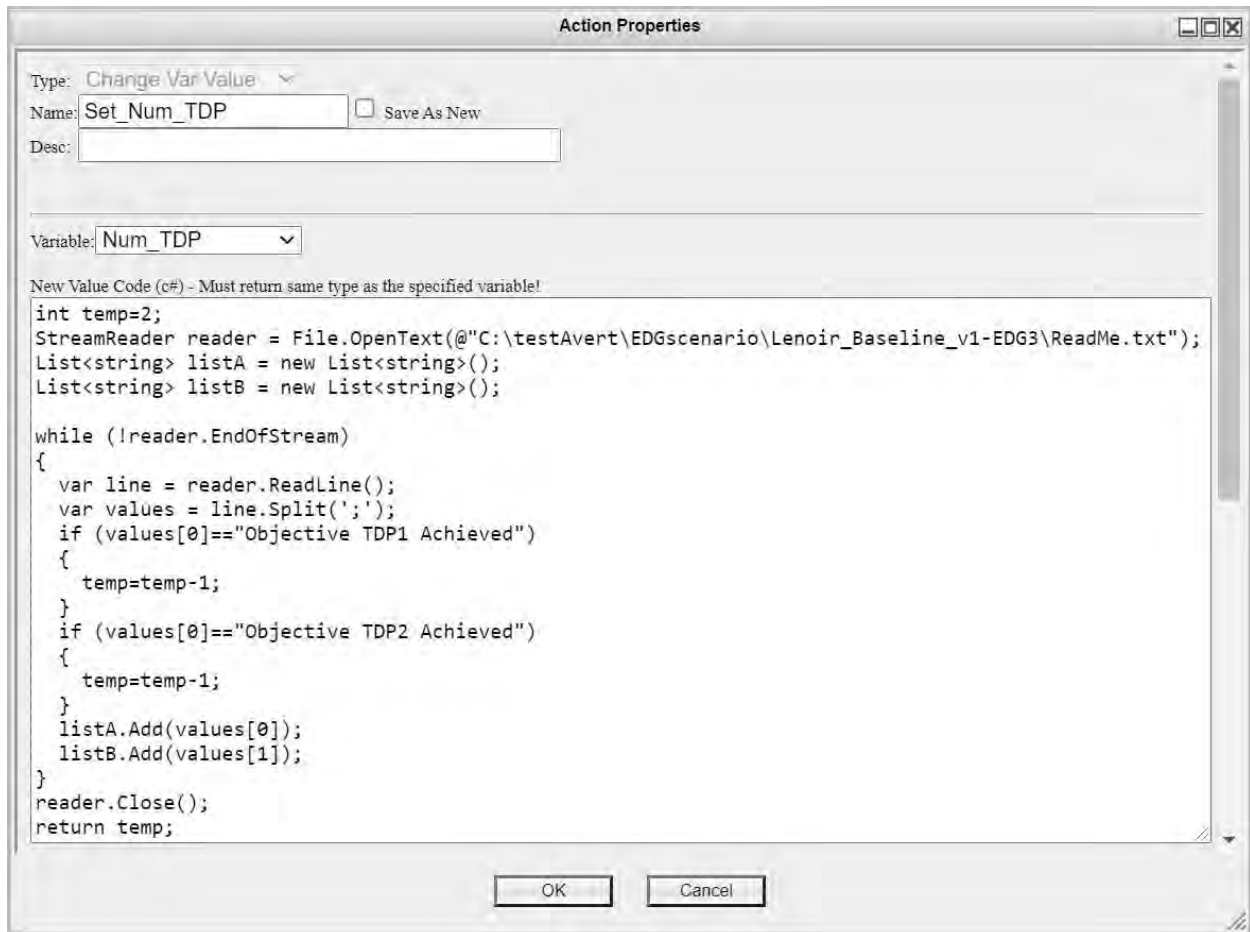


Figure 36. C# code to determine the number of intact TDPs.

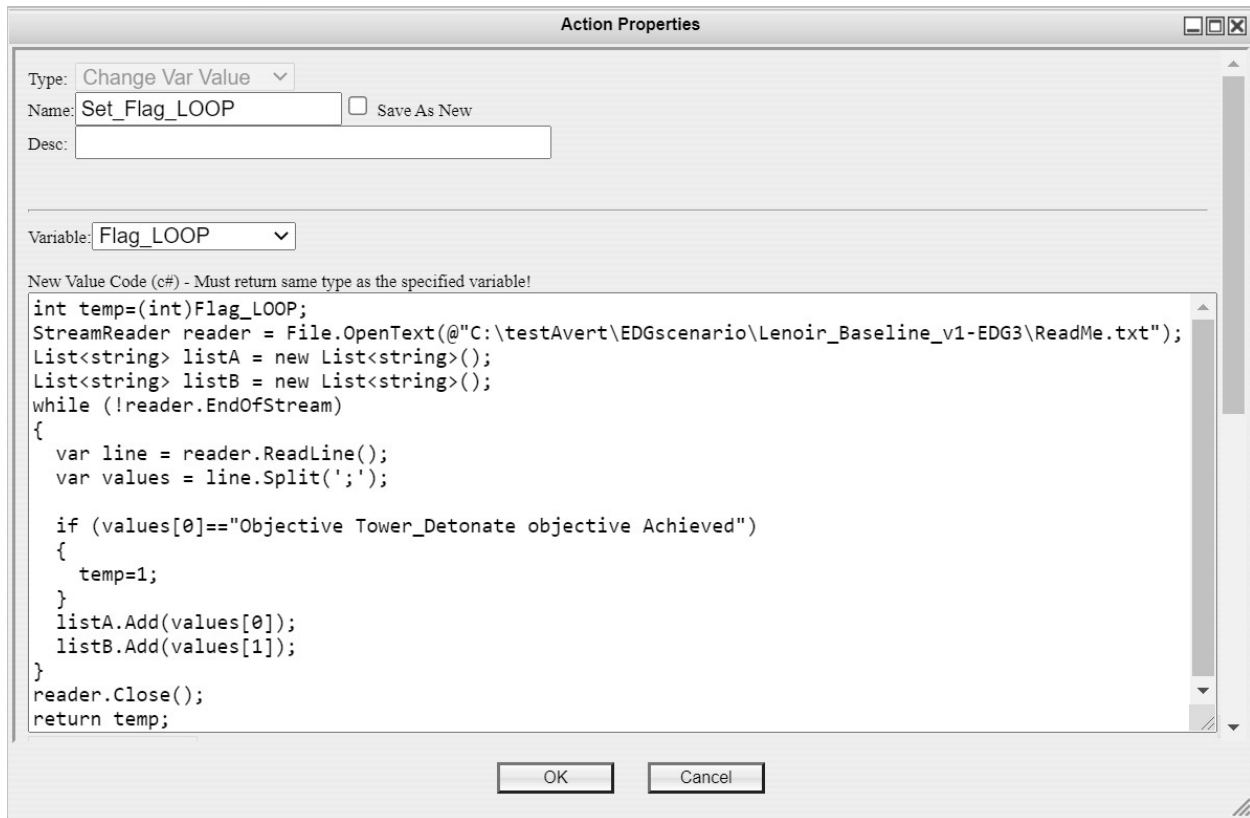


Figure 37. C# code to decide whether LOOP occurred or not.

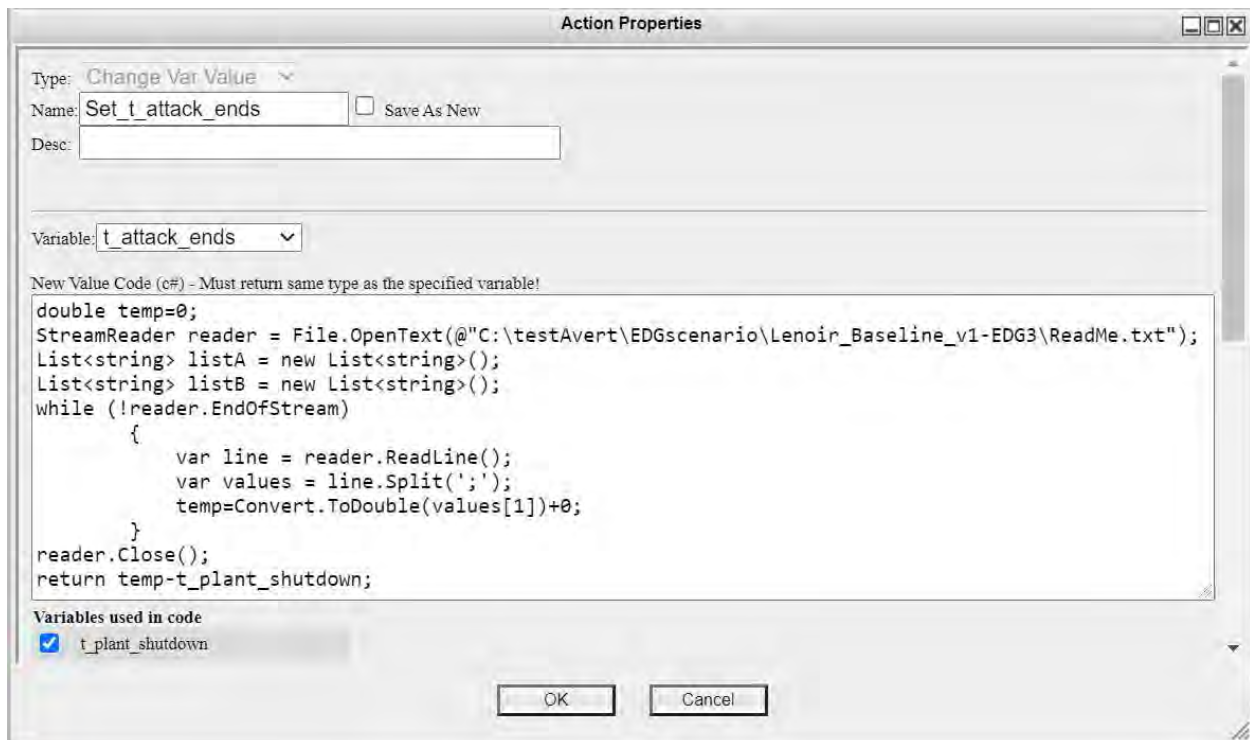


Figure 38. C# code to calculate when the sabotage attack ends.

In the FLEX_Preparation state, there is an action called Set_CD_Prob, as shown in Figure 39. This action is meant to set the probabilistic value of core damage (CD), given a specific set of conditions, such as the available number of safety components after the sabotage attack. This probabilistic CD value is used in the Goto_Plant_Status action in the diagram, such that, even when FLEX equipment is not needed, there is a certain probability that CD will happen due to random failures from the design-basis safety systems. These values can be obtained from static PRA models.

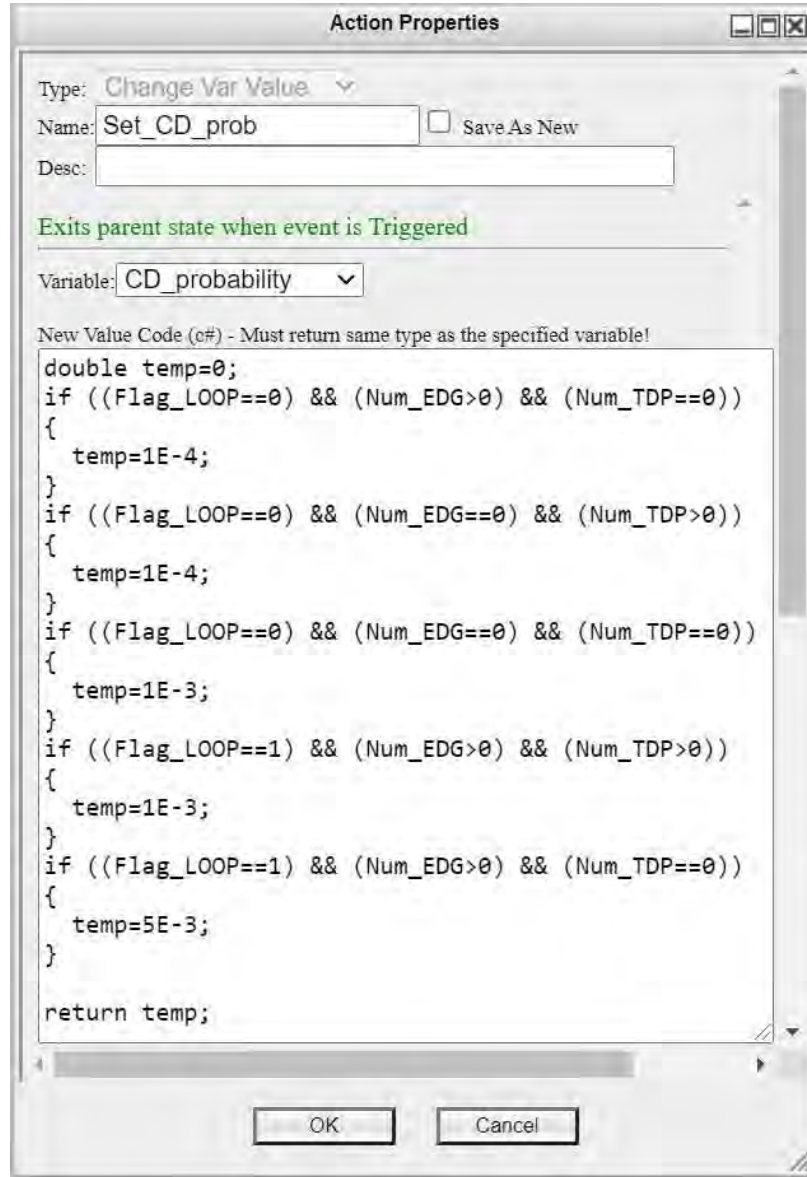


Figure 39. Set_CD_Prob action.

4.4.3 EMERALD-Simajin

Simajin is one of the commercial FOF tools utilized in this research. It is comprised of the Simajin Simulation Engine, Simanij simulation-management graphical user interface and analysis tool, and the Jasmin 3D visualization tool [10]. A snapshot of this FOF tool is shown in Figure 40.

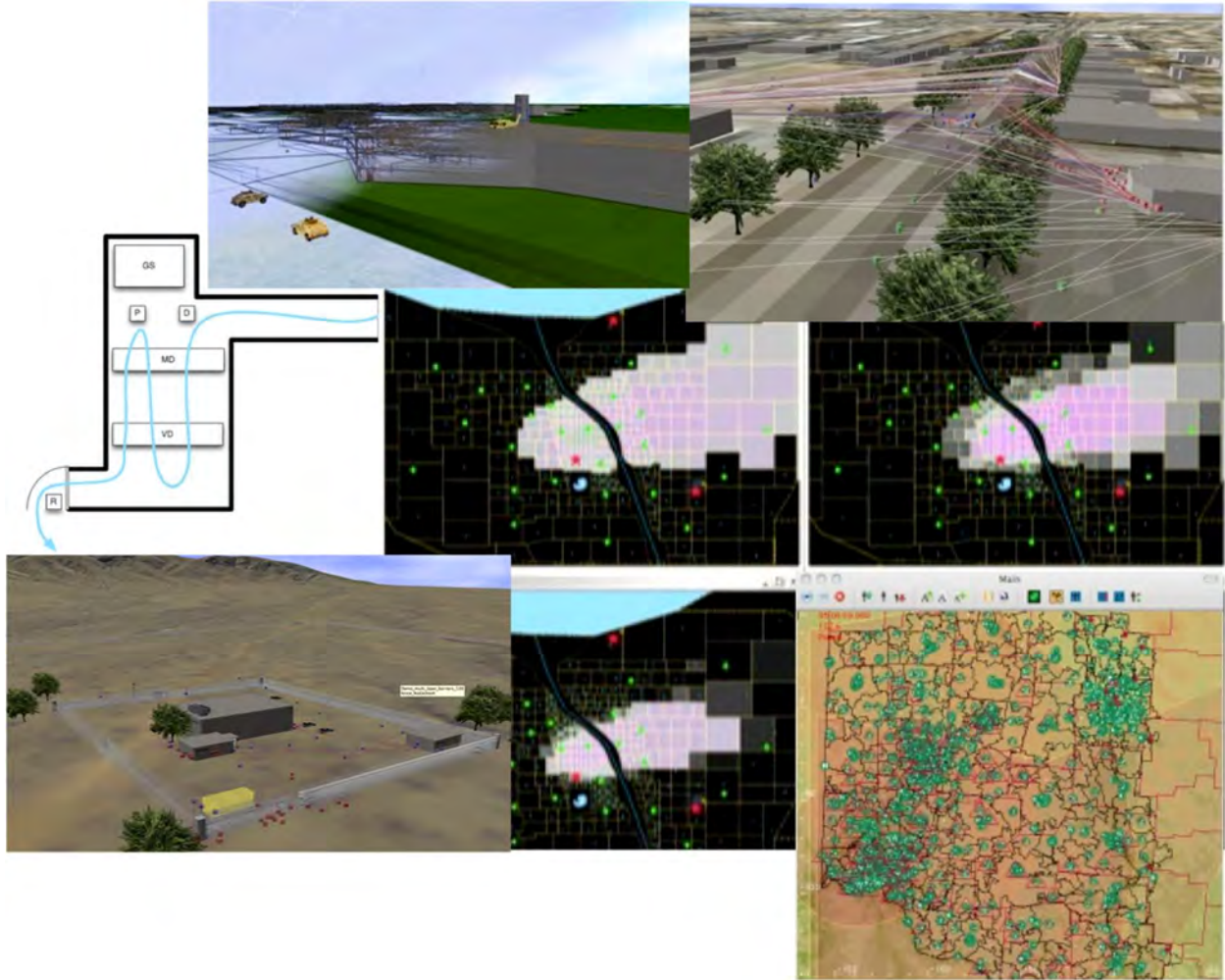


Figure 40. Snapshots of Simajin [10].

Simajin generates output files in the XML format. EMRALD reads selected XML variables from Simajin output files. Samples of these XML files are shown in Figure 41 and Figure 42. The combat statistics of each player can be extracted from the XML file in Figure 41, while other simulation results can be extracted from the file shown in Figure 42. The XML format in Figure 42 is shown in more detail in Appendix B. Examples of EMRALD XML variables are given in the following subsection.

```

*Untitled - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="ISO-8859-1"?>
<collection>
  <cell position="1">
    <row group-key="proforce, Pro_1" kills="0.0" times-
killed="0.0" shots-fired="0.0" shots-at="0.0" time-of-death="0.0"/>
    <row group-key="proforce, Pro_2" kills="0.0" times-
killed="0.0" shots-fired="0.0" shots-at="0.0"/>
    <row group-key="adversary, Adv_1" kills="0.0" times-
killed="0.0" shots-fired="0.0" shots-at="0.0" time-of-death="0.0"/>
  </cell>
</collection>
Ln 9, Col 1      100%  Windows (CRLF)  UTF-8

```

Figure 41. Simajin XML file on combat statistics of each player.

```

*Untitled - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="ISO-8859-1"?>
<study runs-per-cell="0" prefix="Attack1">
  <resultset date-executed="00/00/0000 00:00:00" result-id="[some number]">
    <study-cell>
      <run run-id="0">
        <data name="ADV_deaths" value="0"/>
        <data name="ADV_detained" value="0"/>
        <data name="ADV_fratricide" value="0"/>
        <data name="ADV_gives_up" value="0"/>
        <data name="ADV_in_site" value="0"/>
        <data name="ADV_kills" value="0"/>
        <data name="ADV_shots_taken" value="0"/>
        <data name="battle_time" value="0"/>
        <data name="civilian_deaths" value="0"/>
        <data name="civilian_deaths_by_opfor" value="0"/>
        <data name="civilian_deaths_by_proforce" value="0"/>
        <data name="DG_Room_1_breach_time" value="0"/>
        <data name="DG_Room_4_breach_time" value="0"/>
        <data name="elapsed-time" value="0"/>
      </run>
    </study-cell>
  </resultset>
</study>
Ln 86, Col 22   100%  Windows (CRLF)  UTF-8

```

Figure 42. Simajin XML file on overall FOF results.

4.4.3.1 EMERALD Model

An example of the EMERALD-Simajin integration is given in this subsection. The FOF model is based on a hypothetical facility, and the operator's actions following a sabotage attack are taken from a publicly available reference [11]. The main diagram of the EMERALD model, combining the execution of the FOF simulation tool and the model of FLEX mitigation strategies is shown in Figure 43. The Start State randomizes selected parameters in the FOF simulation such as the weapons' kill probability (PK), the time delay to assess an alarm, and the penalty on adversaries' movement speed due to their unfamiliarity with the indoor areas. The RunSimanij State exports these parameters to the Simajin FOF model and executes the FOF simulation. It reads the FOF results and exports selected variables to a text file. Based on the results, the SimanijComplete event determines the number of intact diesel generators and turbine-driven pumps. The Asses_Plant_Condition State evaluates FLEX mitigation strategies to implement and

their results. For example, the Run_FLEX_EDG Event is initiated if all the design basis EDGs are sabotaged; it transfers the simulation flow to the FLEX_DG sub-diagram. The Check_FLEX_EDG Event is initiated if the FLEX_DG sub-diagram returns a value which indicates the success of the FLEX generator's operation. The FLEX_Unavailable_Or_Delayed Event is initiated if the FLEX equipment is sabotaged or brought into operation later than a conservative time limit of 1 hour. This state leads to the decision of whether the plant is safely shut down or damaged. The End State writes the timing data from the EMERALD simulation into a text file for further statistical analysis.

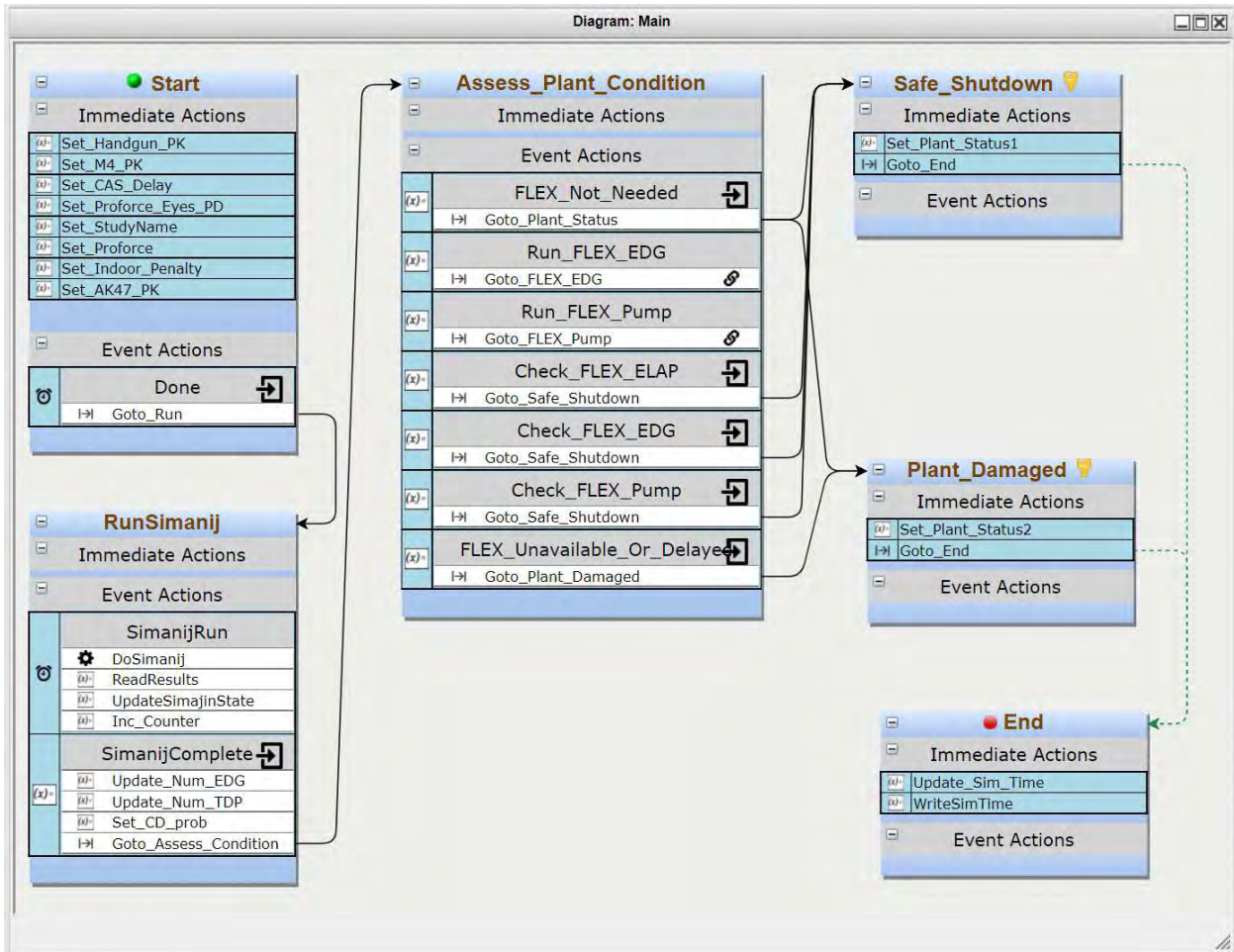


Figure 43. Main diagram for the example of EMERALD-Simajin integration.

Figure 44 shows the C# scripts used to preprocess inputs to the Simajin software, and the postprocess code used to save the data to a text file. These scripts are embedded in the DoSimanj action within the RunSimanj State. The preprocess code sets the directory for Simajin to save its output files and deletes it if it already exists from the previous run. It returns the simulation parameters, which includes the path to the job file and the path to the input file. It executes the Simajin batch file whose path is listed in the “Executable Location” field, with the parameters returned from the Preprocess Code. The Postprocess Code block is executed after the Simajin simulation is completed. It opens the output files and copies them to new files for further analysis. The Simajin output file that contains the variables of interest is saved and renamed to “outputfile.xml” because the EMERALD model always reads from this specific file. It also saves the simulation parameters previously set in the Start State into a text file named “results-all.txt.”

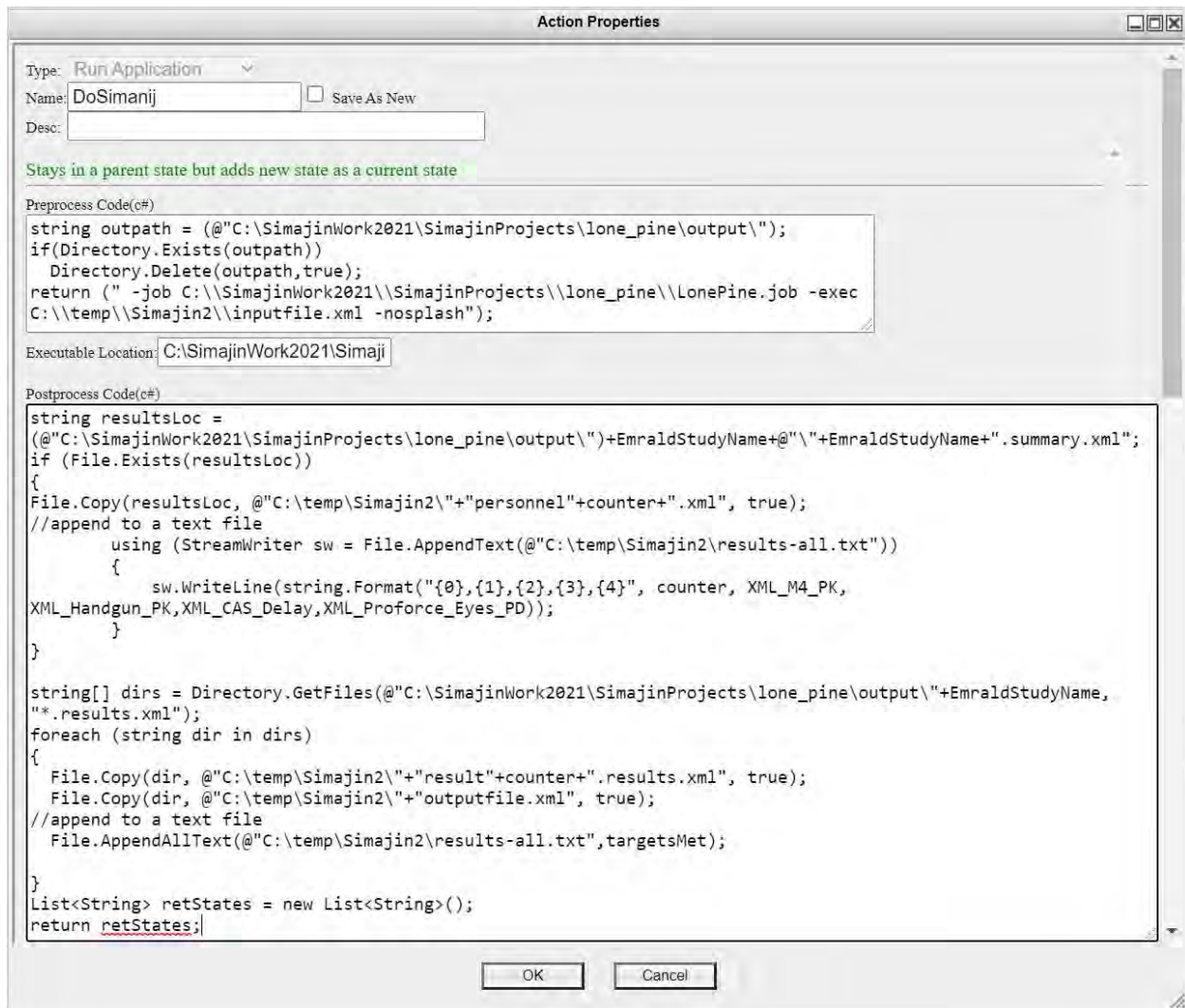


Figure 44. DoSimanij action.

Figure 45 shows the C# scripts used to read selected results from Simajin FOF simulations and save them into a text file for further statistical analysis. This script is embedded in the ReadResults action within the RunSimanij State. The script extracts and saves selected results from the Simajin simulation to a comma-separated text file for further processing. These variables are the timings when the target components are sabotaged, including two diesel generators (EDG), two turbine-driven pumps (TDP), FLEX generators, and FLEX pumps. These variables are defined in EMRALD as XML-link variables, which are automatically extracted from the output file, named outputfile.xml, when the variable is used in a block of C# code. For that reason, the Simajin output file is always copied and renamed to outputfile.xml after the Simajin simulation is complete.

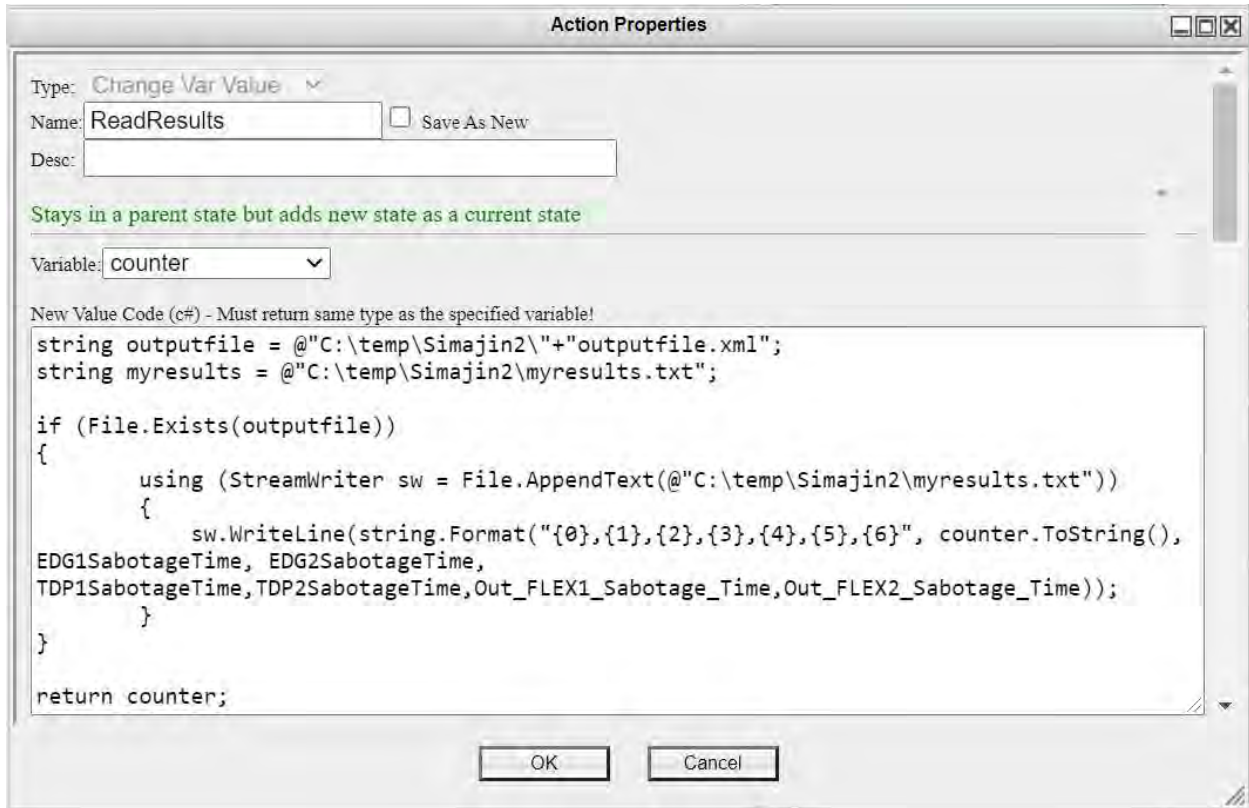


Figure 45. ReadResults action.

Output variables of interest are defined in the EMRALD model as dynamic variables. These variables are extracted and updated from a text file named “outputfile.xml.” Figure 46 and Figure 47 below show how these variables are initialized. The “Var Link” field is the XML Path designation, which queries the requested data from the XML file produced by Simajin. The names in the XML file can be specified differently by the modeler from the ones shown in the Figure 46 and Figure 47. For this simple illustration, the names Generator_4_breach_time and TDP_1_breach_time represent the sabotage time of a diesel generator and of a pump component, respectively.

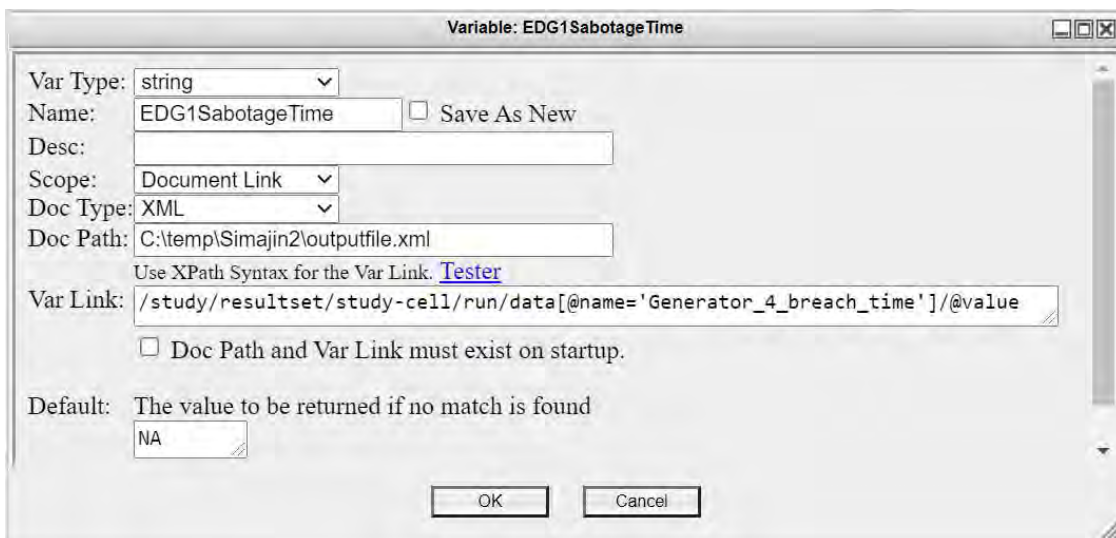


Figure 46. XML variable of Diesel Generator 1 sabotage timing.

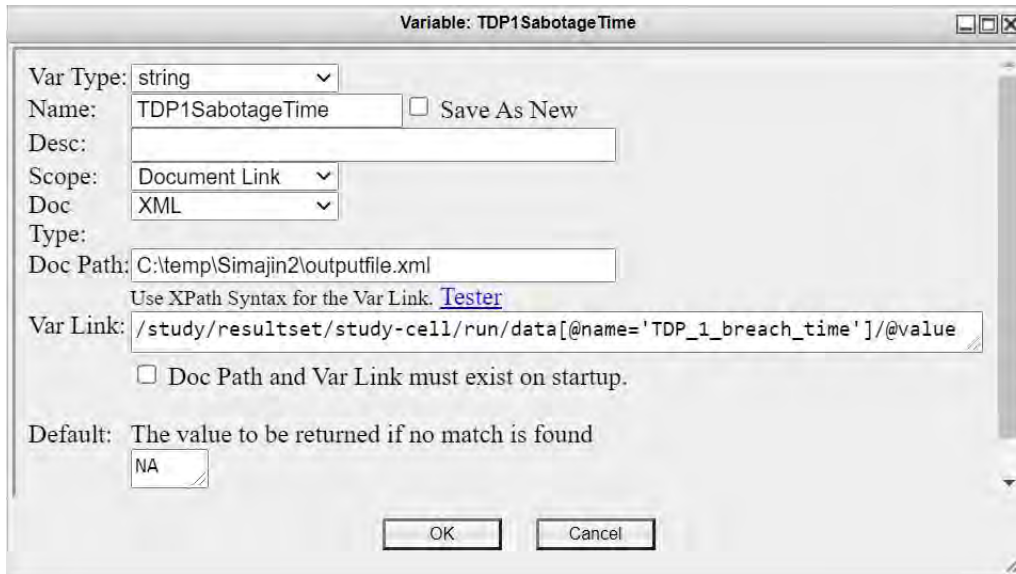


Figure 47. XML variable of Diesel Generator 2 sabotage timing.

4.5 Integration of Thermal-hydraulic Tool with EMRALD

The dynamic modeling of a sabotage attack and operator's response may also need to analyze the plant's response. The plant's response can be estimated by using an existing static PRA model of the plant. However, these estimates are typically conservative since PRA models are developed mostly under the assumption of static scenarios. For example, a PRA model may require an operator to perform a certain action within a certain time limit if Component A and Component B are unavailable. This time limit might be estimated under the assumption that Components A and B are unavailable at the same time. However, in a sabotage attack, there are delays between the time when adversaries sabotage Component A and when adversaries sabotage Component B. Therefore, operators may perform mitigation actions using Component B for some time and reduce the severity of the situation before Component B is eventually unavailable due to the sabotage attack. When this dynamic scenario is considered, the time limit for operators to perform the mitigation action may be longer than the conservative time limit given by the PRA model. The modeler can extract the timing data of when individual plant components or systems are sabotaged and feed this timing data to thermal-hydraulic analysis tools, such as RELAP5-3D or MELCOR, to estimate a less-conservative time limit for operators to perform a certain action.

A simplified example of such integration with MELCOR is shown in Figure 48. It initializes MELCOR in the Start State by running the Set_FilesCopied action shown in Figure 49. This action sets the directory and files for MELCOR analysis and sets a Boolean value to the Set_FilesCopied variable. If the action is completed successfully, the FileMoveSuccess event is triggered, but if it fails, the FileMoveFailed is activated.

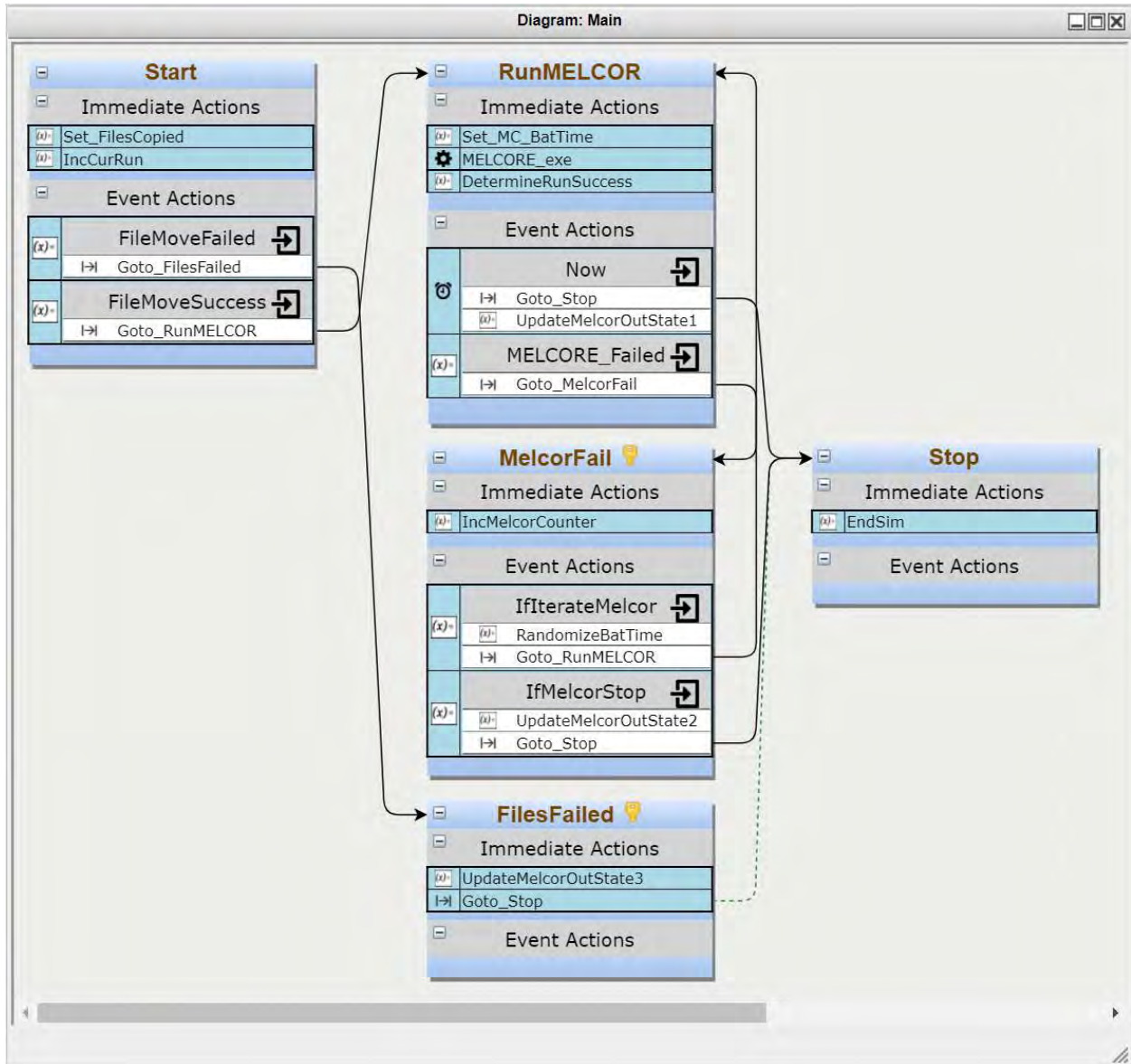


Figure 48. EMRALD diagram to feed data to and read output from MELCOR.

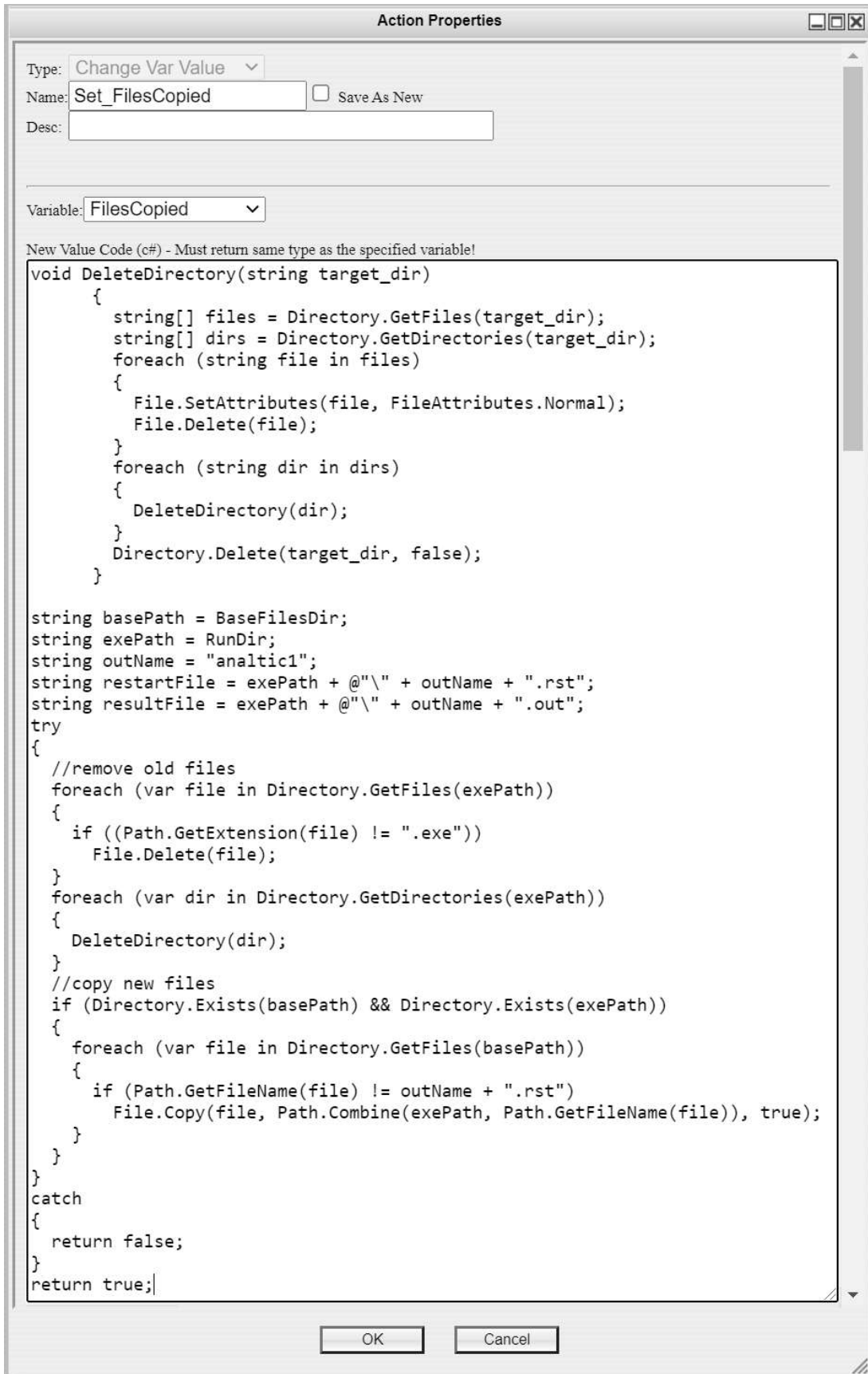


Figure 49. SetFilesCopied C# code.

The RunMELCOR state is active when the FileMoveSuccess event is triggered. The Set_MC_BatTime action in this state is given as an example to save a variable from the FOF simulation to a MELCOR input file. The action updates the value of the MC_BatteryTime variable, which is a document link type of variable, as shown in Figure 50. The variable is linked to a MELCOR input file at a specific line and position. Whenever this variable is updated in EMERALD, it automatically updates the value in that MELCOR input file. After that, EMERALD runs MELCOR and waits for its output through the MELCOR_exe action shown in Figure 51. If MELCOR crashes, the PostProcess code returns the state variable to execute the MelcorFail state. If MELCOR runs successfully, the PostProcess code reads the output file to find specific keywords which indicate whether the core is damaged. If the core is damaged, it returns the CoreDamage state.

Variable: MC_BatteryTime

Var Type: double

Name: MC_BatteryTime Save As New

Desc:

Scope: Document Link

Doc Type: Text RegEx

Doc Path: .\RunDir\neup_bwr.inp

Use Regular Expression Syntax for the Var Link. Tester

Var Link: cf_formula 1 bat

Doc Path and Var Link must exist on startup.

Line #: 1 (Use if a line after the RegExp match is desired.)

Beg Pos: 17 (Line start position, use 0 is the first character.)

Num Chars: 0 (Use for a specific # of characters, set to 0 for until next space, uncheck for end of line.)

Default: The value to be returned if no match is found

0

OK Cancel

Figure 50. MC_BatteryTime variable.

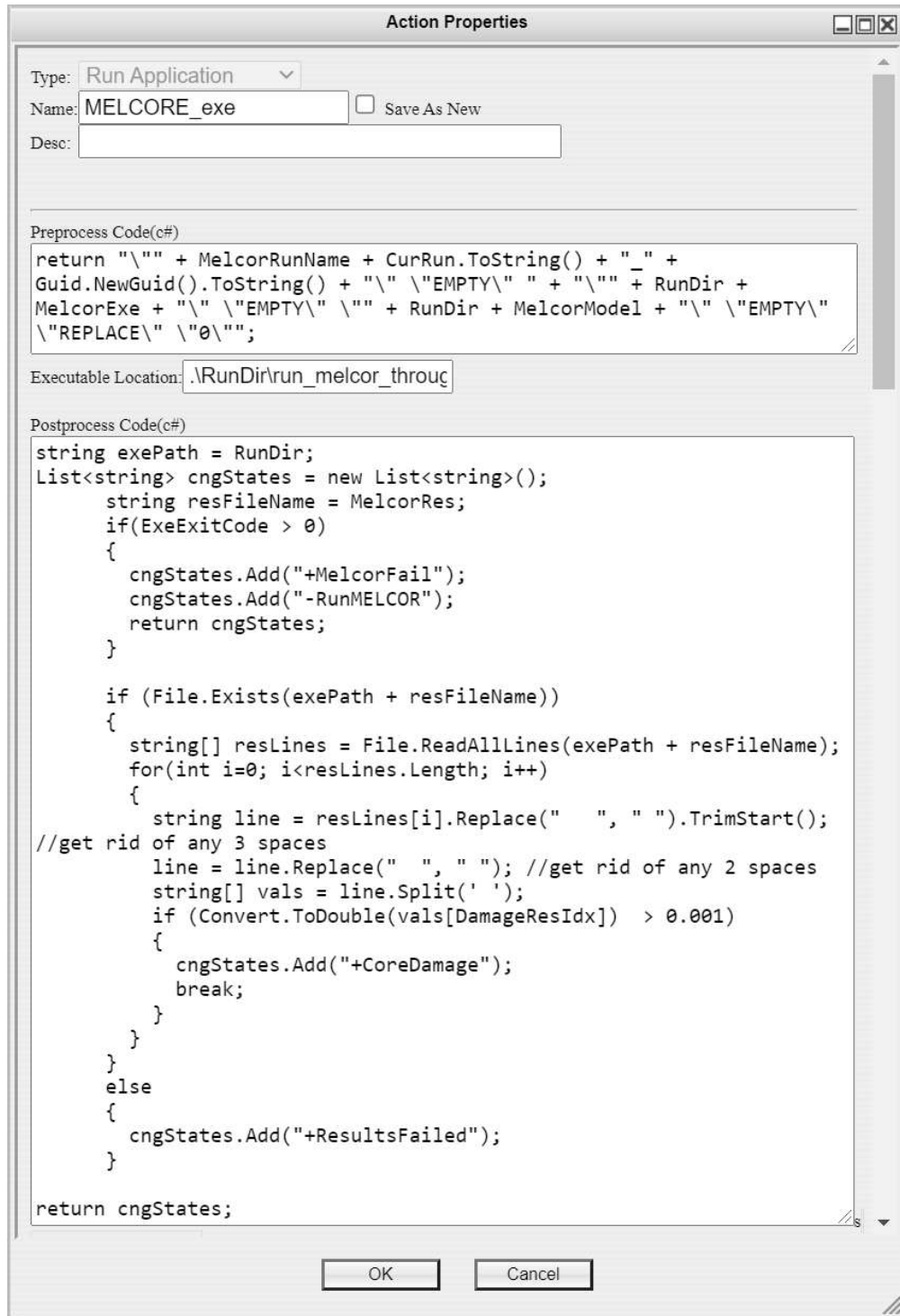


Figure 51. MELCOR_exe action.

After MELCOR_exe is completed, EMERALD runs the DetermineRunSuccess action shown in Figure 52. This action reads the MELCORE_Success variable and returns true if the variable contains the “Normal termination” keyword. This variable itself is a document link variable, as shown in Figure 53. It uses a regular expression (regex) to search for the keyword “Normal termination TIME” in the MELCOR output file. This keyword confirms MELCOR ran successfully and terminated normally. If the

keyword is found, EMRALD goes to the Stop State and ends the simulation. The user may add additional actions in this state to extract certain data from the MELCOR output file as needed.

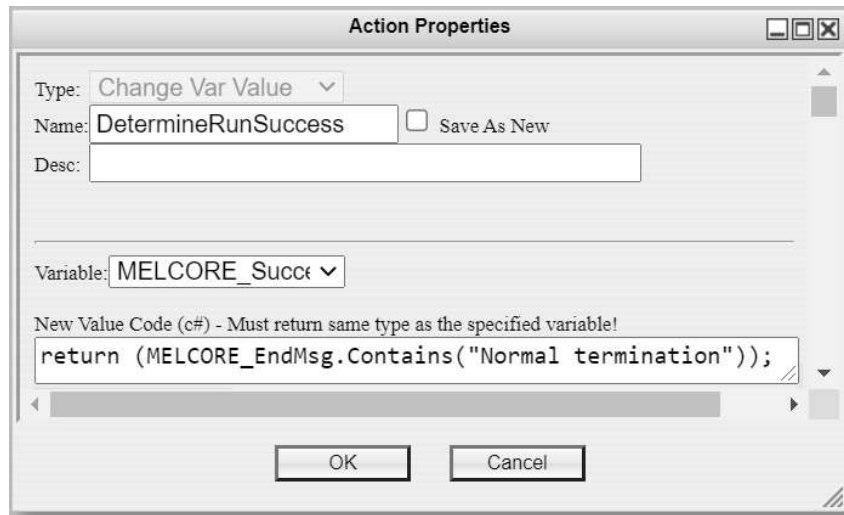


Figure 52. DetermineRunSuccess action.

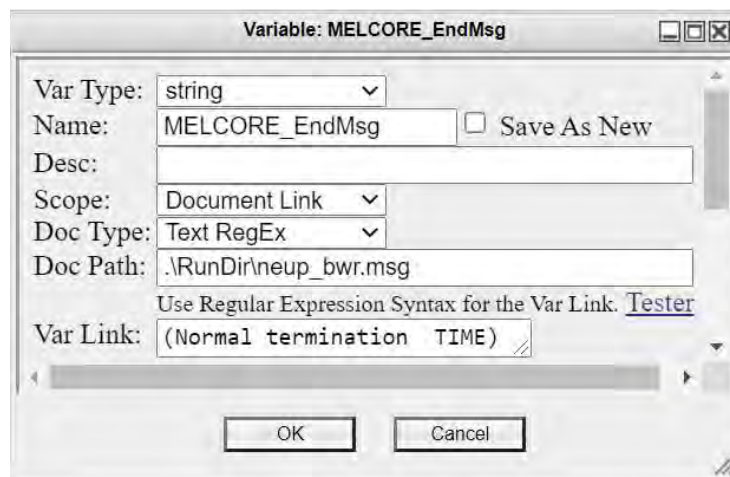


Figure 53. MELCORE_EndMsg variable.

A similar process using “Document Variables” can be used for the other thermal-hydraulics codes, such as RELAP5-3D. If many scenarios will need to run a thermal-hydraulics analysis, a surrogate model, such as a limit surface, can be made using a software tool such as Risk Analysis and Virtual Environment (RAVEN) [12]. A limit surface is constructed by sampling the parameters that were modified before, by EMRALD, along a specified range, building a multi-dimensional separation between CD and OK. Once this model is built, EMRALD can provide the parameters and immediately obtain the result for the core status.

4.6 Running Analysis

Running an EMRALD model steps through the provided model, performing Monte Carlo (random sampling) of data to determine outcomes. The simulation needs to be run numerous times (specified by the user) to obtain good statistical results. The number of runs needed depends on the model and the needs of the use case.

4.6.1 User Interface

To run an analysis of an EMRALD model, start EMRALD_Sim.exe. You may encounter a Windows warning about it being an unverified application; just continue. The application consists of four tabs with the first two (Model and Simulate) being applicable for the FOF modeling.

4.6.1.1 Main Tab

Use the file menu to select the model to open. The text for the model is loaded into the “Model Text Area,” and a logical/compile check of the model is performed. If all is okay, then the data and options in the other tabs are available. If there are any errors, they will show in the “Error List Area,” shown in Figure 54.

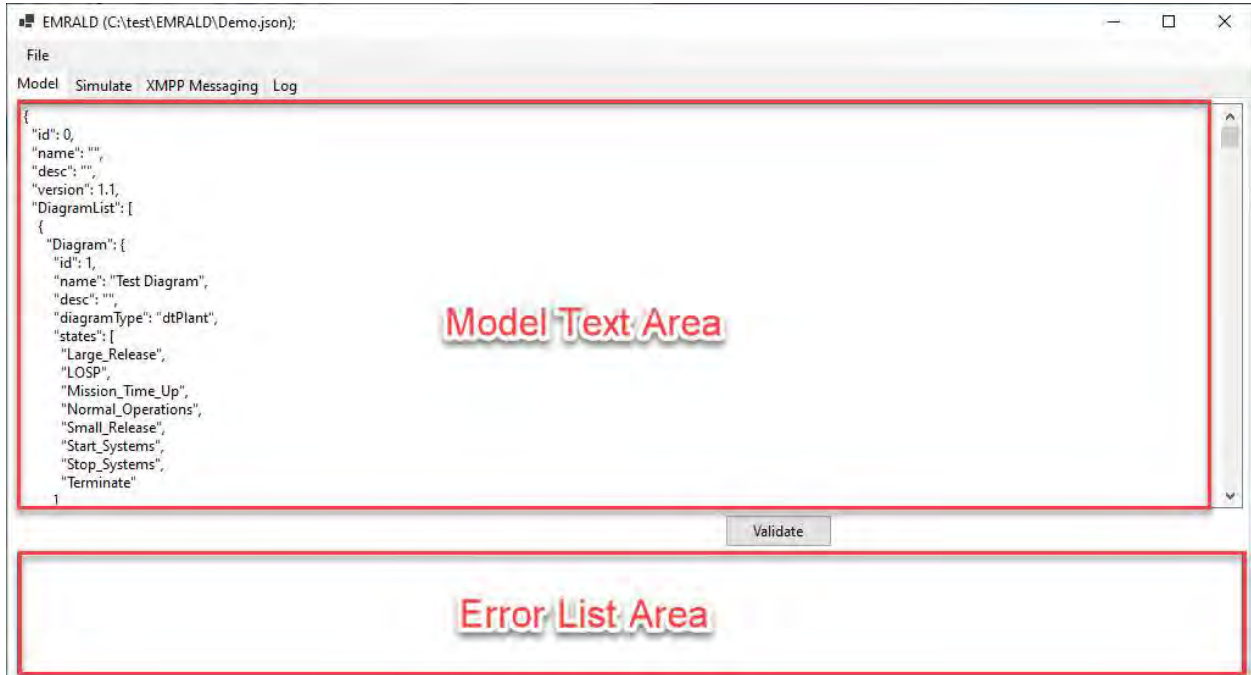


Figure 54. Model tab of the EMRALD solve engine.

4.6.1.2 Simulate Tab

After a model is loaded, click on the simulate tab to run the model. Before running, a few parameters need to be specified. If any variables need to be shown in the results, check the box shown in #1 of Figure 55. Then, enter then number of runs needed, as shown in #2 of Figure 55. Next, set the results locations, as shown in #3 of Figure 55. (Note, if the paths for the results locations do not exist, an error will occur.) Finally, hit the “Run” button to start the simulation. The simulation can run and all the parameters can be set using a command line interface. To view the command line options, type “EMALD_Sim.exe -help” in the command line.

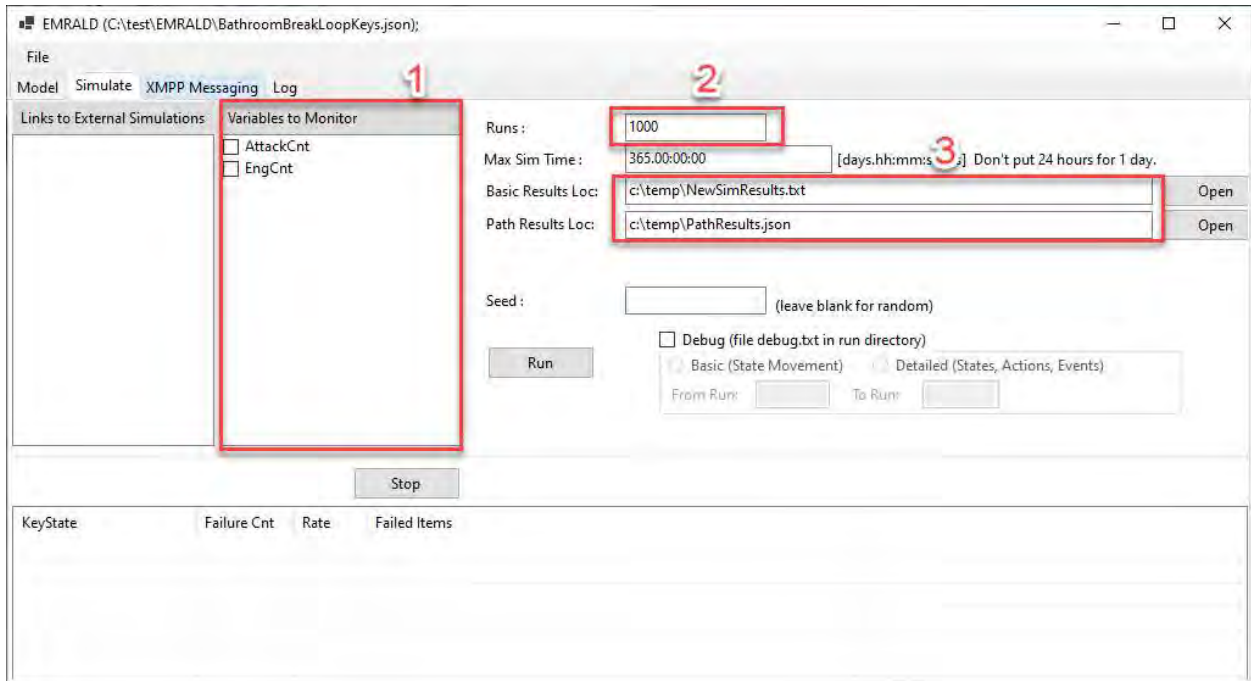


Figure 55. "Simulate" tab of the EMRALD solve engine.

A summary of current results for completed runs will be displayed in the bottom section, showing the key states and how many times the simulation finished with those key states. These results, as shown in Figure 56, are meant to be a quick visual check to verify the model is running as expected.

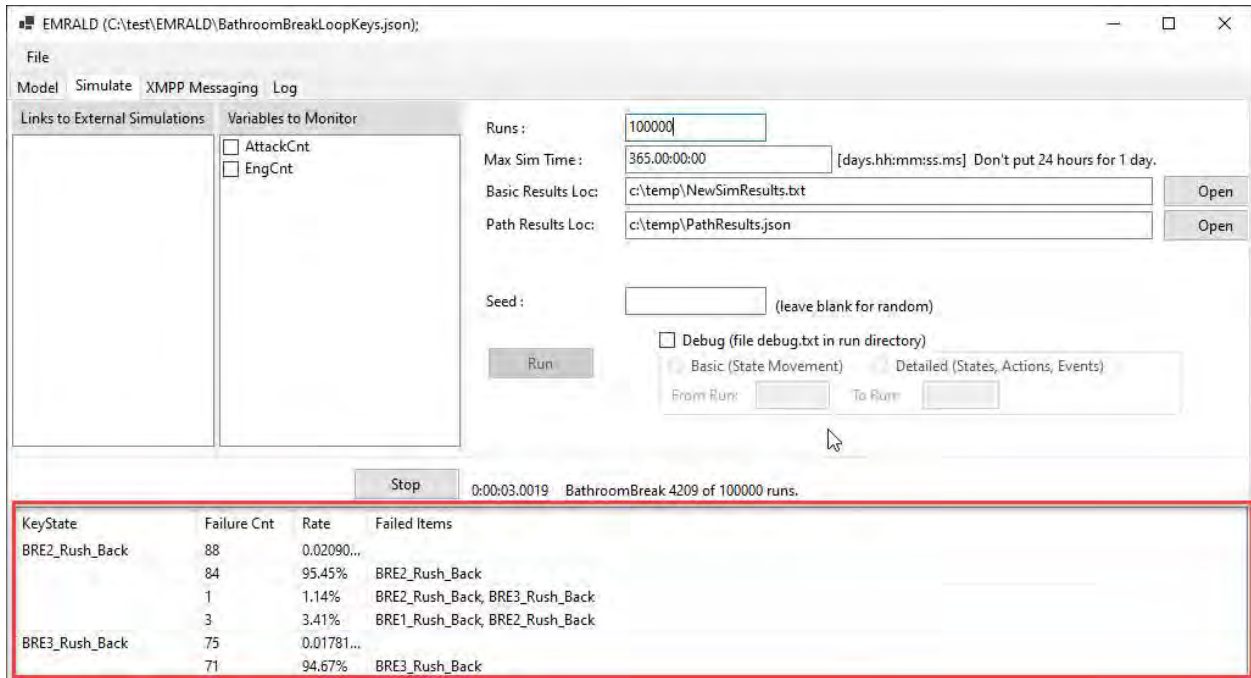


Figure 56. EMRALD results summary area.

4.6.1.3 Debug Options

When simulating, there is the option to output a debug log. This log can show the time and movement in and out of states, events triggered, and actions taken, as it steps through the simulation. This is done for each run in the specified “From Run” and “To Run” range. The log is named “debug.txt” and is located in the same directory as the “EMERALD_Sim.exe.” It is recommended that only a few runs are logged; otherwise, it slows down the simulation and creates a very large file.

```
12 EnterState: Isolate_SGISO_D, time: 00:18:37.8953533, Cause Event: Goto_Isolate_SGISO_D, fromState-Action: TimeToIdent
13 DoEvent: TimeToDepressRCS
14 DoTransitionAction: Goto_RCSDepressDone
15 ExitState: OpTryDepressRCS
16 EnterState: RCSDepressDone, time: 00:25:51.7339696, Cause Event: Goto_RCSDepressDone, fromState-Action: TimeToDepressRCS
17 DoEvent: TimeToIsolate
18 DoTransitionAction: Goto_TimeFail
19 EnterState: EvalTime, time: 00:39:43.5099518, Cause Event: Goto_TimeFail, fromState-Action: TimeToIsolate
20 DoEvent: EvalTimeGood
21 DoTransitionAction: Goto_IdentInTime
```

Figure 57. Example of debug output file.

4.7 EMERALD Results

There are two sets of result data: Basic and Path. The Basic results are similar to what is shown on the interface when solving. Additionally, the value of the selected variables for each run ending in that state are shown in a comma delimited list.

```
1 Simulation = BathroomBreak
2 Runtime = 00:00:02.6048596
3 Runs = 1000 of 1000
4 BRE3_Rush_Back Occurred 24 times, Rate =0.024
5 (23) [95.83333333333334]BRE3_Rush_Back
6 (1) [4.1666666666666666]BRE1_Rush_Back, BRE3_Rush_Back
7 - Variable Values -
8 AttackCnt = 24,141,176,192,224,239,325,369,465,510,605,627,635,682,709,721,732,774,812,819,826,828,858,973
```

Figure 58. Example of Basic results output.

The Path results are a JSON tree structure detailing the path, probability and variable values from start to finish for every key state result of the simulation runs. These results can be processed to determine significant avenues of failure or timing data. They can also be used for developing mitigation methods or optimizing procedures.

4.8 Helpful Hints

Developing an integrated model can be challenging; the following suggestions will help a user to avoid or overcome some of the main obstacles.

4.8.1 Modeling

- Save often and using a version #. There is no undo and being able to go back to a previous version is helpful.
- Sometimes the user interface does not refresh properly; if you performed an action and do not see the change made, close the diagram and reopen it.
- Develop small pieces and test; then, add them together to make the model. There are a couple ways to do this. One option is to create a testing diagram with a Start State and add a timer event with a transition action. As you develop a component or other small part of the model to test, add the beginning state of that model piece to the transitions “To State” list. Then, to test a particular piece of the model, set the probability for the “To State” to 1.0 and the rest of them to 0, as shown in Figure 59.

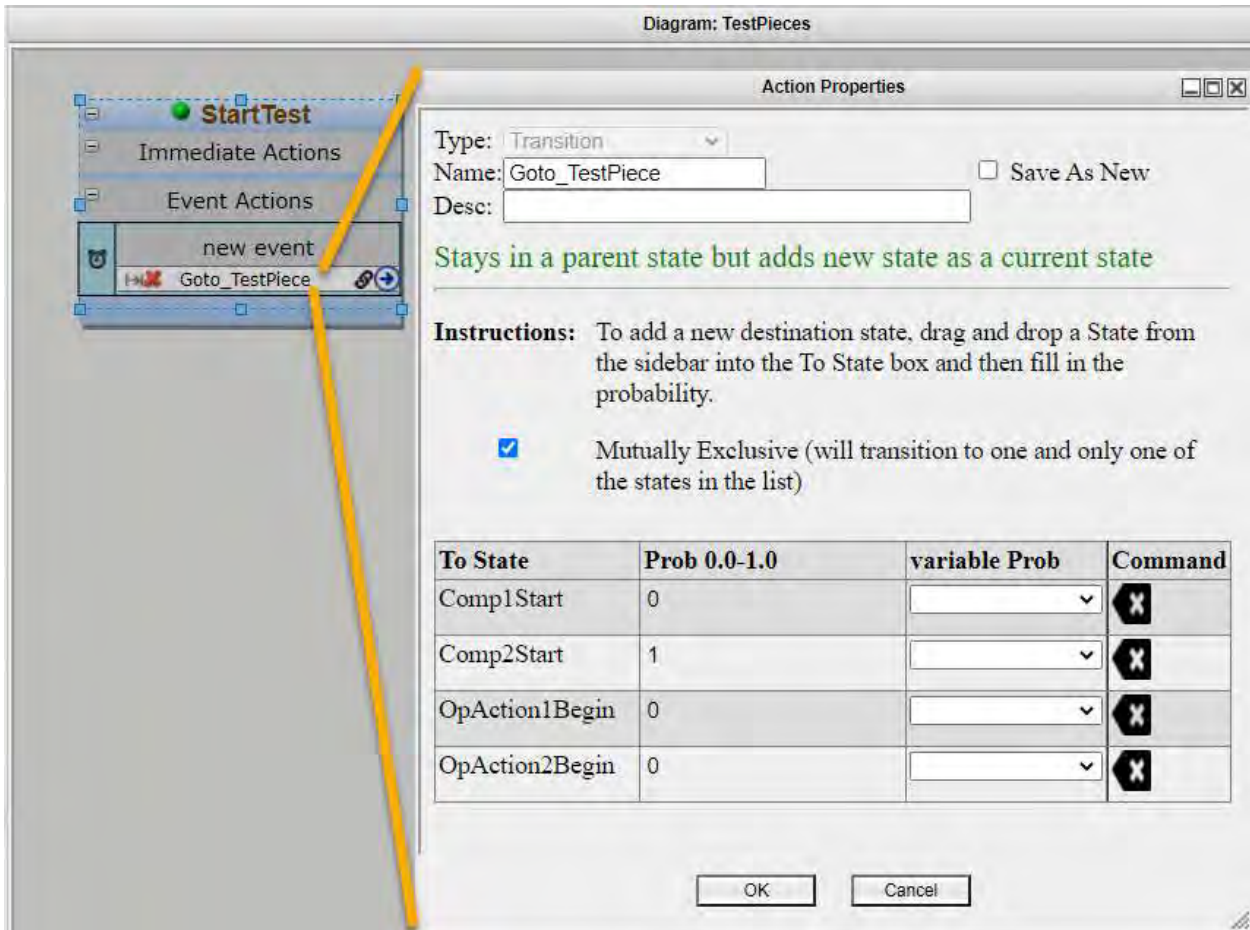


Figure 59. Testing using a transition action with multiple “To States.”

Another option is to develop separate EMRALD models for the pieces. Save the model pieces and use the solve engine to test. Once the pieces have been verified, create a new model, and use the menu option “Project”->”Merge...” to import the model pieces into one as shown in Figure 60.

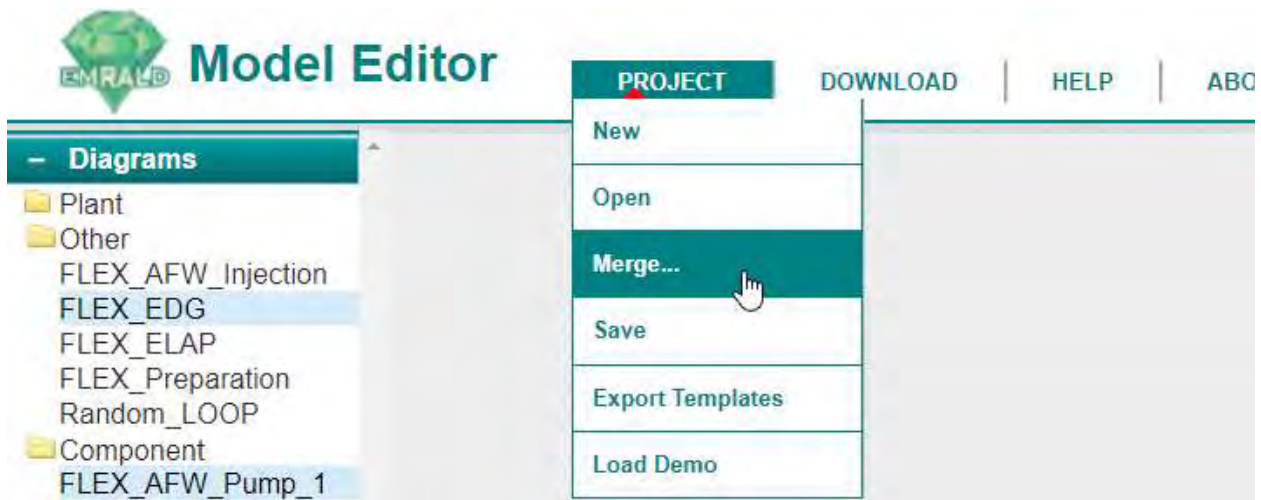
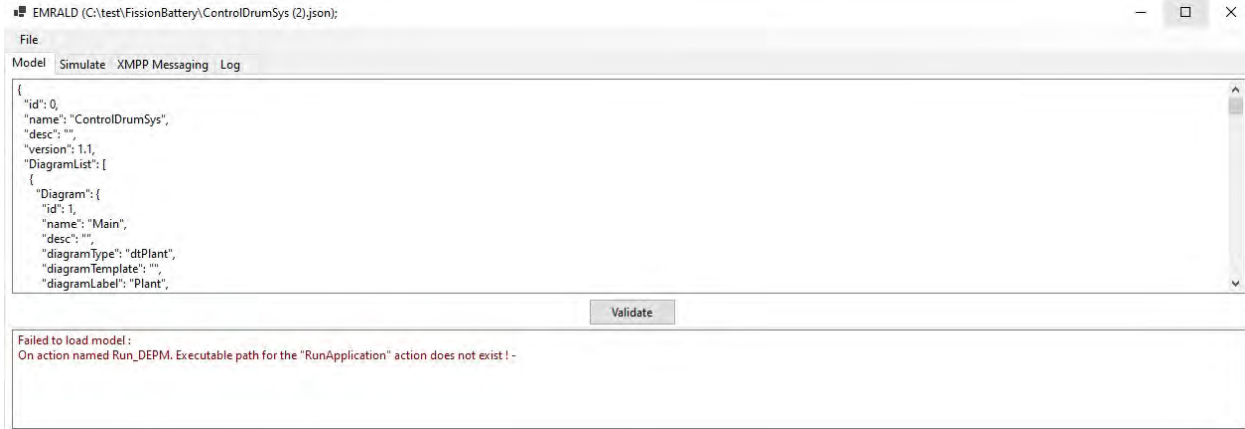


Figure 60. Project merge menu option.

4.8.2 Solving

- When opening a model with the solve engine, it automatically does a check on the model for logical and compile errors. If there are any errors, a message will show at the bottom of the screen in red to help diagnose the problem.



- The debug log is a great help in finding problems in the model; however, if the problem occurs seldom, then the debug log can get large and it can be hard to find the issue. To reduce the data, run the model with the “Seed” assigned and run. Determine which scenario run had the problem, check the debug box, and put the bad run number in the “From Run” and “To Run” field. The seed will make the runs reproducible and assigning the “From Run” and “To Run” causes the debug output to be limited to those runs.

Runs :

Max Sim Time : [days.hh:mm:ss.ms] Don't put 24 hours for 1 day.

Basic Results Loc:

Path Results Loc:

Seed : (leave blank for random)

Debug (file debug.txt in run directory)

Basic (State Movement) Detailed (States, Actions, Events)

From Run: To Run:

Figure 61. Example of settings to debug a specified simulation run.

5. REGULATORY CONSIDERATIONS

The purpose of the framework described within this report is to inform the protective strategy at an existing commercial nuclear power reactor site with new strategies that would result in lower operational costs while maintaining the capability to accomplish the security program general performance objective of 10 CFR 73.55(b) to “provide high assurance that activities involving special nuclear material are not inimical to the common defense and security and do not constitute an unreasonable risk to the public health and safety” [13]. Changes to the physical protection system (PPS) of a nuclear power reactor must

be done under the auspices of the applicable federal, state, and local authorities and regulatory agencies. The PPS of a commercial nuclear power reactor is primarily regulated by the NRC, under licenses authorizing operation by 10 CFR Part 50 and 10 CFR Part 52 [13, 14]. The NRC approves Physical Security Plans, Security Training and Qualification Plans, and Safeguards Contingency Plans (hereafter referred to as “security plans”) through the process of issuing an operating license under 10 CFR Part 50, a combined operating license under 10 CFR Part 52, or through the applicable license changes processes. The NRC also verifies the capability of the PPS through the implementation of a security baseline inspection program and FOF inspections. The PPS is evaluated against the DBT, a set of adversary characteristics that outline the challenges against which a nuclear power reactor must meet the performance objective.

Security plans are placed on the docket during initial plant licensing and continue as a part of the licensing basis of the commercial nuclear power reactor. The Physical Security Plan outlines how a licensee meets the requirements outlined in 10 CFR Part 73[16]. The Security Training and Qualification plan summarizes how security program personnel will effectively perform their assigned duties in accordance with 10 CFR Part 73 Appendix B [17]. The Safeguards Contingency Plan documents the licensee protective strategy and the roles and responsibilities of the staff required to act for security events.

5.1 Changes to the Physical Protection System

The framework described in this report is designed to analyze changes to a PPS and provide a basis to justify changes within the licensee site change process, and to a regulator. The framework can be used to analyze a variety of changes to a PPS. Changes should be analyzed using this framework and documented in a report. The changes proposed as part of the framework analysis should be evaluated for changes in protective strategy, engineered security and safety features, and personnel needed to execute the protective strategy.

Proposed changes to the PPS will illustrate different aspects of the protective strategy. Changes to armed responder locations and numbers, BRE locations and numbers, and inclusion of advanced weapons or ROWS may necessitate different procedures for security staff. The inclusion of FLEX equipment would require that the operators who are to act in specific security events, security managers, and armed responders participate in the development of procedures to include those equipment and actions within the protective strategy. Licensees should develop and document these new proposed procedures.

The addition of new security equipment within the protective strategy needs to be evaluated to comply with regulations in 10 CFR 73.55(n), “Maintenance, testing, and calibration” [18]. The inclusion of safety or non-safety related equipment within a target set should be documented and consistent with RG 5.81, “Target Set Identification and Development for Nuclear Power Reactors.” The new equipment included in target sets should be protected in the protective strategy consistent with 10 CFR 73.55 and RG 5.81 [13, 19].

The proposed changes may result in changes to the training and qualification requirements for site security personnel (e.g., protecting site operators during movement, field of fire and distance changes from BREs, changes in weapons, and ROWS). The inclusion of operators in the protective strategy should be consistent with the “credible operator action criteria” listed in RG 5.81 [19].

5.2 Security Plan Changes

The framework described in this report would necessitate changes to security plans to include changes to the protective strategy, engineered security and safety features, and the personnel that would need to be trained and qualified. Commercial nuclear power reactors enact changes to security plans according to the regulations under 10 CFR 50.54(p) and 10 CFR 50.90 [21, 23].

The regulations in 10 CFR 50.54(p) require that changes to security plans must be approved by the NRC, unless the changes do not decrease the safeguards effectiveness of the plan and:

- (i) “All safeguards capabilities specified in the safeguards contingency plan available and functional;
- (ii) Detailed procedures developed according to appendix C to part 73 of this chapter available at the licensee's site; and
- (iii) All appropriate personnel trained to respond to safeguards incidents as outlined in the plan and specified in the detailed procedures.” [21]

NRC approval of changes to security plans follow the requirements outlined in 10 CFR 50.90 for amendments of license, construction permit, or early site permit.

Though the purpose of the analysis in this framework it to demonstrate no reduction in the effectiveness of the PPS as the result of the proposed changes, it is encouraged that licensees develop a limited scope security assessment, consistent with NUREG/CR-7145, “Nuclear Power Plant Security Assessment Guide” to support the change notice required under 10 CFR 50.54(p) [22, 21]. Additionally, it is encouraged that licensees submit the security assessment before enacting changes to the protective strategy to avoid issues of non-compliance.

Changes to the PPS of a larger nature (reduction in armed responders, inclusion of ROWS, etc.) should seek a license amendment under 10 CFR 50.90 [23] and conduct a wider-scope security assessment to support the amendment application.

Licensees seeking to change their PPS beyond regulatory requirements delineated in 10 CFR Part 50, Part 52, and Part 7314–16 will need to apply for an exemption, consistent with 10 CFR 50.12 [24].

It is noted that after security plan changes are approved by the NRC, the licensee is still responsible for the successful execution of the protective strategy. Approval of the security plans does not guarantee compliance; the NRC will likely conduct follow-up inspections via the baseline inspection program and FOF exercise inspections to evaluate the effectiveness of the PPS changes.

6. CONCLUSION

This document provides a guidance for using a dynamic computational framework for optimizing physical security at nuclear power plants. The framework is developed at INL as part of the research within DOE’s LWRS program and is intended to be used by physical security experts at U.S. commercial nuclear power plants. The instructions in this document cover the FOF tools that are used by a majority of U.S. commercial plants for their physical security modeling and assessment. Following the guidance in this document will enable a user to integrate their plant specific FOF models with the dynamic simulation tool EMERALD, model operator actions, integrate with PRA tools such as CAFTA or SAPHIRE, and integrate with thermal-hydraulic tools such as RELAP5-3D. The dynamic computational framework enables a plant’s physical security to transition from the traditional binary assessment of success/failure to a risk-informed performance-based assessment. Such an assessment enables further analysis, such as what-if scenarios and staff-reduction evaluation, thereby optimizing physical security at plants.

7. REFERENCES

1. Pacific Gas & Electric Company. 2018. “PG&E Company 2018 Nuclear Decommissioning Costs Triennial Proceeding Prepared Testimony – Volume 1,” December 13, 2018. Accessed on August 24, 2021. <https://analysis.nuclearenergyinsider.com/pg-e-seeks-decommissioning-head-start-cost-estimates-rise>.
2. D. Osborn, et al. 2019. “Modeling for Existing Nuclear Power Plant Security Regime.” SAND2019-12015, Sandia National Laboratories.

3. Shah, A, et al. "Coping Time Analysis for Chromium coated Zircaloy for Station Blackout Scenario based on Dynamic Risk Assessment," Proceedings of The 15th Probabilistic Safety Assessment and Management Conference (PSAM 15), Venice, Italy, November 2020.
4. Nuclear Energy Institute. 2016. "NEI 12-06 Rev. 4: Diverse and Flexible Coping Strategies (FLEX) Implementation Guide." NEI 12-06 Rev. 4, Nuclear Energy Institute.
5. Idaho National Laboratory. n.d. "Emerald Overview." Accessed on August 24, 2021. <https://emerald.inl.gov>.
6. Le, T., J. Parks, and T. Noel. 2018. "Mixed Reality 3D Tabletop Tool with Radioactive Source Model Visualization." IAEA-CN-269, International Atomic Energy Agency.
7. "Severe accident analysis of a PWR station blackout with the MELCOR, MAAP4 and SCDAP/RELAP5 codes." 2004. Nuclear Engineering and Design 234 (1–3): 129–145. <https://doi.org/10.1016/j.nucengdes.2004.09.001>.
8. Robinson, S. 2018. "Introduction to Discrete-Event Simulation: How it Works." Proceedings of the 9th Simulation Workshop (SW18) edited by A. Anagnostou, M. Fakhimi, R. Meskarian, D. Robertson, 13–28. Birmingham: Operational Research Society.
9. Sandia National Laboratories. n.d. "The Hypothetical Facility—Lone Pine Nuclear Power Plant: The Twenty-Sixth International Training Course." Accessed on August 24, 2021. https://share-ng.sandia.gov/itc/assets/hypo_fac_lpnpp.pdf
10. Rhino Corps. "Simajin: Simulation Application Suite." Products. Accessed on August 24, 2021. <https://www.rhinocorps.com/products/simulation-application-suite>.
11. Christian, R., et al. 2020. "Integration of FLEX Equipment and Operator Actions in Plant Force-On-Force Models with Dynamic Risk Assessment." INL/EXT-20-59510, Idaho National Laboratory.
12. Idaho National Laboratory. n.d. Raven Overview. Accessed on August 24, 2021. <https://raven.inl.gov/SitePages/Overview.aspx>
13. "Requirements for physical protection of licensed activities in nuclear power reactors against radiological sabotage." 2012. U.S. Nuclear Regulatory Commission, title 10, part 73, section 55. <https://www.nrc.gov/reading-rm/doc-collections/cfr/part073/full-text.html#part073-0055>.
14. "Domestic Licensing Of Production And Utilization Facilities." 2019. U.S. Nuclear Regulatory Commission, title 10, part 50,. <https://www.nrc.gov/reading-rm/doc-collections/cfr/part050/index.html>.
15. "Part 52—Licenses, Certifications, And Approvals For Nuclear Power Plants." 2019. title 10, part 52, U.S. Nuclear Regulatory Commission. <https://www.nrc.gov/reading-rm/doc-collections/cfr/part052/index.html>.
16. "Physical Protection Of Plants And Materials." 2021. U.S. Nuclear Regulatory Commission, title 10, part 73. <https://www.nrc.gov/reading-rm/doc-collections/cfr/part073/full-text.html>
17. "Appendix B to Part 73—General Criteria for Security Personnel." 2019. U.S. Nuclear Regulatory Commission, title 10, part 73, appendix b. <https://www.nrc.gov/reading-rm/doc-collections/cfr/part073/full-text.html#part073-appb>.
18. "Requirements for physical protection of licensed activities in nuclear power reactors against radiological sabotage." 2012. U.S. Nuclear Regulatory Commission, title 10, part 73, section 55(n). <https://www.nrc.gov/reading-rm/doc-collections/cfr/part073/full-text.html#part073-0055>.
19. U.S. Nuclear Regulatory Commission. Rev. 1 – 2019. "Target Set Identification and Development for Nuclear Power Reactors (OUO-SRI)." RG 5.81, U.S. Nuclear Regulatory Commission.
20. "Specific exemptions." n.d. U.S. Nuclear Regulatory Commission, title 10, part 73, section 5. <https://www.nrc.gov/reading-rm/doc-collections/cfr/part073/full-text.html#part073-0005>.
21. "Conditions of licenses." 2019. U.S. Nuclear Regulatory Commission, title 10, part 50, section 54(p). <https://www.nrc.gov/reading-rm/doc-collections/cfr/part050/part050-0054.html>.

22. Zamanali, J. and C. Chwasz. 2013. "Nuclear Power Plant Security Assessment Guide." NUREG/CR-7145, U.S. Nuclear Regulatory Commission.
23. "Application for amendment of license, construction permit, or early site permit." 2007. U.S. Nuclear Regulatory Commission, title 10, part 50, section 90.
24. "Specific exemptions." 1985. U.S. Nuclear Regulatory Commission, title 10, part 50, section 12.

Appendix A

SCRIBE3D JSON File

This section gives an example of the JSON input file used to control SCRIBE3D simulation. The facility, attack scenario, and values used in this example are hypothetical in nature and do not correlate to any existing facility.

```
{
  "SimTime": 0.0,
  "StopAtTime": 0.0,
  "StopAtObjectives": [],
  "StopAfterEngagement": false,
  "LastEditedBy": "Scribe",
  "StopReason": "StoppedAtTime: 0.0",
  "Exception": "",
  "EntitiesList": [{
    "Name": "Adversary_1",
    "Objectives": [{
      "Name": "A_1 Breach Outer PIDAS",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_1 Breach Inner PIDAS",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_1 Breach Room Door A",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_1 Sabotage Target A",
      "HasCompleted": false,
```

```
"IsSkipped": false,
"WaitTime": 0.0
}, {
  "Name": "A_1 Breach Room Door B",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_1 Sabotage Target B",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "Breach Room Door C",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_1 Destroy Target C",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_1 Breach Room Door D",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_1 Sabotage Target D",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}],
"EyesOn": true,
"IsAlive": true,
```

```

    "DeathTime": 0.0
  }, {
    "Name": "Adversary_2",
    "Objectives": [{
      "Name": "A_2 Wait at Outer PIDAS",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_2 Wait at Inner PIDAS",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_2 Guard Room A",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_2 Guard Room C",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }, {
      "Name": "A_2 Guard Room D",
      "HasCompleted": false,
      "IsSkipped": false,
      "WaitTime": 0.0
    }
  ]},
  "EyesOn": true,
  "IsAlive": true,
  "DeathTime": 0.0
}, {
  "Name": "Adversary_3",
  "Objectives": [{

```

```
"Name": "A_3 Wait at Outer PIDAS",
"HasCompleted": false,
"IsSkipped": false,
"WaitTime": 0.0
}, {
  "Name": "A_3 Wait at Inner PIDAS",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Begin Waiting",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Guard Room B",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Breach Room Door A",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Sabotage Target A",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Breach Room Door B",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
```



```
"Name": "A_3 Sabotage Target B",
"HasCompleted": false,
"IsSkipped": false,
"WaitTime": 0.0
}, {
  "Name": "A_3 Guard Room C",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Breach Room Door C",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Destroy Target C",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Guard Room D",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Breach Room Door D",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}, {
  "Name": "A_3 Sabotage Target D",
  "HasCompleted": false,
  "IsSkipped": false,
  "WaitTime": 0.0
}],
```

```
    "EyesOn": true,
    "IsAlive": true,
    "DeathTime": 0.0
  }, {
    "Name": "B_1",
    "Objectives": [],
    "EyesOn": true,
    "IsAlive": true,
    "DeathTime": 0.0
  }, {
    "Name": "Response_1",
    "Objectives": [],
    "EyesOn": true,
    "IsAlive": true,
    "DeathTime": 0.0
  }, {
    "Name": "Response_2",
    "Objectives": [],
    "EyesOn": true,
    "IsAlive": true,
    "DeathTime": 0.0
  }
],
"EntitiesInEngagement": []
}
```

Appendix B

XML Output File of Simajin

The following script is a sample template of the Simajin output file. This file is only an example and may differ, based on the Simajin FOF model. It does not contain any facility-specific data; however, it shows the template or format of the text output file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<study runs-per-cell="0" prefix="Attack1">
  <resultset date-executed="00/00/0000 00:00:00" result-id="[some number]">
    <study-cell>
      <run run-id="0">
        <data name="ADV_deaths" value="0"/>
        <data name="ADV_detained" value="0"/>
        <data name="ADV_fratricide" value="0"/>
        <data name="ADV_gives_up" value="0"/>
        <data name="ADV_in_site" value="0"/>
        <data name="ADV_kills" value="0"/>
        <data name="ADV_shots_taken" value="0"/>
        <data name="battle_time" value="0"/>
        <data name="civilian_deaths" value="0"/>
        <data name="civilian_deaths_by_opfor" value="0"/>
        <data name="civilian_deaths_by_proforce" value="0"/>
        <data name="DG_Room_1_breach_time" value="0"/>
        <data name="DG_Room_4_breach_time" value="0"/>
        <data name="elapsed-time" value="0"/>
        <data name="Equipment5_repair_time" value="0"/>
        <data name="events" value="0"/>
        <data name="first_detection" value="0"/>
        <data name="first_response" value="0"/>
        <data name="game-time" value="0"/>
        <data name="Generator_1_breach_time" value="0"/>
        <data name="Generator_4_breach_time" value="0"/>
        <data name="KEPS" value="0"/>
        <data name="LAA_ADV_count" value="0"/>
        <data name="LAA_ADV_dead" value="0"/>
      </run>
    </study-cell>
  </resultset>
</study>
```

<data name="LAA_ADV_enter_t" value="0"/>
<data name="LAA_ADV_escapes" value="0"/>
<data name="LAA_ADV_exits" value="0"/>
<data name="LAA_PRO_count" value="0"/>
<data name="LAA_PRO_dead_in" value="0"/>
<data name="LAA_targets_stolen" value="0"/>
<data name="memory-used" value="0"/>
<data name="PA_ADV_count" value="0"/>
<data name="PA_ADV_dead" value="0"/>
<data name="PA_ADV_enter_t" value="0"/>
<data name="PA_ADV_escapes" value="0"/>
<data name="PA_ADV_exits" value="0"/>
<data name="PA_Inner_ADV_count" value="0"/>
<data name="PA_Inner_ADV_dead" value="0"/>
<data name="PA_Inner_ADV_enter_t" value="0"/>
<data name="PA_Inner_ADV_escapes" value="0"/>
<data name="PA_Inner_ADV_exits" value="0"/>
<data name="PA_Inner_breach_time" value="0"/>
<data name="PA_Inner_PRO_count" value="0"/>
<data name="PA_Inner_PRO_dead_in" value="0"/>
<data name="PA_Inner_targets_stolen" value="0"/>
<data name="PA_Outer_breach_time" value="0"/>
<data name="PA_PRO_count" value="0"/>
<data name="PA_PRO_dead_in" value="0"/>
<data name="PA_targets_stolen" value="0"/>
<data name="Powerplant_Door_breach_time" value="0"/>
<data name="PPA_breach_time" value="0"/>
<data name="proforce_deaths" value="0"/>
<data name="proforce_fratricide" value="0"/>
<data name="proforce_in_fight" value="0"/>
<data name="proforce_kills" value="0"/>
<data name="proforce_shots_taken" value="0"/>
<data name="proforce_wins" value="0"/>
<data name="route_failure_count" value="0"/>
<data name="sabotage_time_Turbine" value="0"/>

```
<data name="target_goal_count" value="0"/>
<data name="target_goal_met" value="0"/>
<data name="target_stolen" value="0"/>
<data name="target_work_pct" value="0"/>
<data name="target_work_pct_Turbine" value="0"/>
<data name="target_work_time" value="0"/>
<data name="target_work_time_Turbine" value="0"/>
<data name="TDP_1_breach_time" value="0"/>
<data name="TDP_2_breach_time" value="0"/>
<data name="VA_ADV_count" value="0"/>
<data name="VA_ADV_dead" value="0"/>
<data name="VA_ADV_escapes" value="0"/>
<data name="VA_ADV_exits" value="0"/>
<data name="VA_PRO_count" value="0"/>
<data name="VA_PRO_dead_in" value="0"/>
<data name="VA_targets_stolen" value="0"/>
<data name="warning_tactics" value="0"/>
<data name="warnings" value="0"/>
<data name="version" value="0"/>
<data name="build" value="0"/>
</run>
<summary>
  <data name="ADV_deaths" value="0"/>
  <data name="ADV_detained" value="0"/>
  <data name="ADV_fratricide" value="0"/>
  <data name="ADV_gives_up" value="0"/>
  <data name="ADV_in_site" value="0"/>
  <data name="ADV_kills" value="0"/>
  <data name="ADV_shots_taken" value="0"/>
  <data name="battle_time" value="0"/>
  <data name="civilian_deaths" value="0"/>
  <data name="civilian_deaths_by_opfor" value="0"/>
  <data name="civilian_deaths_by_proforce" value="0"/>
  <data name="DG_Room_1_breach_time" value="0"/>
  <data name="DG_Room_4_breach_time" value="0"/>

```

<data name="elapsed-time" value="0"/>
<data name="Equipment5_repair_time" value="0"/>
<data name="events" value="0"/>
<data name="first_detection" value="0"/>
<data name="first_response" value="0"/>
<data name="game-time" value="0"/>
<data name="Generator_1_breach_time" value="0"/>
<data name="Generator_4_breach_time" value="0"/>
<data name="KEPS" value="0"/>
<data name="LAA_ADV_count" value="0"/>
<data name="LAA_ADV_dead" value="0"/>
<data name="LAA_ADV_enter_t" value="0"/>
<data name="LAA_ADV_escapes" value="0"/>
<data name="LAA_ADV_exits" value="0"/>
<data name="LAA_PRO_count" value="0"/>
<data name="LAA_PRO_dead_in" value="0"/>
<data name="LAA_targets_stolen" value="0"/>
<data name="memory-used" value="0"/>
<data name="PA_ADV_count" value="0"/>
<data name="PA_ADV_dead" value="0"/>
<data name="PA_ADV_enter_t" value="0"/>
<data name="PA_ADV_escapes" value="0"/>
<data name="PA_ADV_exits" value="0"/>
<data name="PA_Inner_ADV_count" value="0"/>
<data name="PA_Inner_ADV_dead" value="0"/>
<data name="PA_Inner_ADV_enter_t" value="0"/>
<data name="PA_Inner_ADV_escapes" value="0"/>
<data name="PA_Inner_ADV_exits" value="0"/>
<data name="PA_Inner_breach_time" value="0"/>
<data name="PA_Inner_PRO_count" value="0"/>
<data name="PA_Inner_PRO_dead_in" value="0"/>
<data name="PA_Inner_targets_stolen" value="0"/>
<data name="PA_Outer_breach_time" value="0"/>
<data name="PA_PRO_count" value="0"/>
<data name="PA_PRO_dead_in" value="0"/>

<data name="PA_targets_stolen" value="0"/>
<data name="Powerplant_Door_breach_time" value="0"/>
<data name="PPA_breach_time" value="0"/>
<data name="proforce_deaths" value="0"/>
<data name="proforce_fratricide" value="0"/>
<data name="proforce_in_fight" value="0"/>
<data name="proforce_kills" value="0"/>
<data name="proforce_shots_taken" value="0"/>
<data name="proforce_wins" value="0"/>
<data name="route_failure_count" value="0"/>
<data name="sabotage_time_Turbine" value="0"/>
<data name="target_goal_count" value="0"/>
<data name="target_goal_met" value="0"/>
<data name="target_stolen" value="0"/>
<data name="target_work_pct" value="0"/>
<data name="target_work_pct_Turbine" value="0"/>
<data name="target_work_time" value="0"/>
<data name="target_work_time_Turbine" value="0"/>
<data name="TDP_1_breach_time" value="0"/>
<data name="TDP_2_breach_time" value="0"/>
<data name="VA_ADV_count" value="0"/>
<data name="VA_ADV_dead" value="0"/>
<data name="VA_ADV_escapes" value="0"/>
<data name="VA_ADV_exits" value="0"/>
<data name="VA_PRO_count" value="0"/>
<data name="VA_PRO_dead_in" value="0"/>
<data name="VA_targets_stolen" value="0"/>
<data name="warning_tactics" value="0"/>
<data name="warnings" value="0"/>
<data name="version" value="0"/>
<data name="build" value="0"/>
</summary>