The HUNTER Dynamic Human Reliability Analysis Tool: Procedurally Driven Operator Simulation

Thomas Ulrich, Ronald Boring, Jooyoung Park, Yunyeong Heo, and Jeeyea Ahn
Human Factors and Reliability Department, Idaho National Laboratory, Idaho Falls, Idaho, USA
{Thomas.Ulrich, Ronald.Boring, Jooyoung.Park, Yunyeong.Heo, Jeeyea.Ahn}@inl.gov

Abstract: The Human Unimodel for Nuclear Technology to Enhance Reliability (HUNTER) software is a procedurally driven nuclear operator simulation developed to support dynamic human reliability analysis (HRA) research. The approach aims to reduce the complexity of dynamic HRA by leveraging existing plant procedures to aid analysts in developing simple proceduralized tasks to define the virtual operator. The simulation was developed to run in tandem with a plant model to enable dynamic performance context evaluations using GOMS-HRA primitives to define time distributions and base human error probabilities for individual procedure steps. The simulation supports HEP quantification with a dynamic approach intended for integration with dynamic probabilistic risk analysis. This paper describes the initial version of the software implementation and details the various modules used to perform the simulation.

1. INTRODUCTION

Nuclear power plants (NPP) are safety critical systems characterized by low probability, but significant and potentially catastrophic failure consequences. This low failure probability is a direct result of the substantial and rigorous nuclear, systems, and human factors engineering performed to ensure safe operation. Indeed, the nuclear industry has an impeccable safety track record evidenced by the existence of a mere handful of failure events. Without empirical evidence, which fortunately is scant, probabilistic risk analysis (PRA) methods are used to evaluate the design for potential failures before they occur. Until recently, the nuclear industry primarily used static risk analysis methods comprised of plant functionality models comprised of component and system logic with an assigned failure probability. Manufacturers of the equipment and control systems do have empirical failure data on individual elements, and therefore the models can integrate component and subsystem probabilities to evaluate larger critical system function failures to determine system risk. There have been recent advances in dynamic probabilistic risk analysis to perform Monte Carlo based simulations of these risk models in tandem with a plant model representing the critical processes. Through this approach, emergent failures are revealed, and a richer evaluation of system risk is achieved.

One crucial aspect that is currently quite limited in these dynamic PRA approaches, at least within the U.S., is integration with realistic human reliability analysis (HRA) estimates. Human activities are often represented generically by one of two human error probability (HEP) values, 0.01 or 0.001 for diagnosis or action activity types respectively, taken from SPAR-H [1]. Operator actions are much more varied and can have drastic consequences—both positive and negative on system performance. As such, dynamic HRA modeling techniques are needed to provide a more realistic HEP value based on the plant state and operational context suitable for integration with dynamic PRA. This paper is one of a series of related papers documenting the research and development performed to translate the Human Unimodel for Nuclear Technology To Enhance Reliability (HUNTER) framework to a dynamic human reliability software tool [2]. Specifically, this paper presents the software architecture developed to perform the simulation. The software modules and their functionality are described within the context of analysts workflow, which includes model construction and model execution.

2. HUNTER ARCHITECTURE

The HUNTER framework was translated into a simulation application written in the Python programming language. The simulation code supports the ability to execute analyst defined scenarios organized as tasks comprised of procedures or procedure sections. The procedures within the tasks are predefined as inputs to the application and contain all the necessary data elements to execute proceduralized tasks based on the simulated nuclear plant state. The state is used to evaluate the logic within each procedure step, and the virtual operator completion of each step is evaluated with a dynamic HRA module that uses the simulation context to calculate completion durations, HEPs, and success or failure.

A central aspect of this dynamic HRA software implementation is the proceduralization of activities to drive the virtual operator within the simulation. This model forms the roadmap that the scheduler references as it progresses through the simulation. When performing a task, operators execute a series of procedures or sections of procedures to arrive at the intended goal, i.e., plant state. Given its central importance to the simulation, the structure of procedures forms the basis for the overall software architecture and the modules were built to accommodate this structure. NPP operators use several different types of procedures, but from an HRA perspective, the most pertinent procedures are emergency operating procedures (EOPs) and abnormal operating procedures (AOPs).

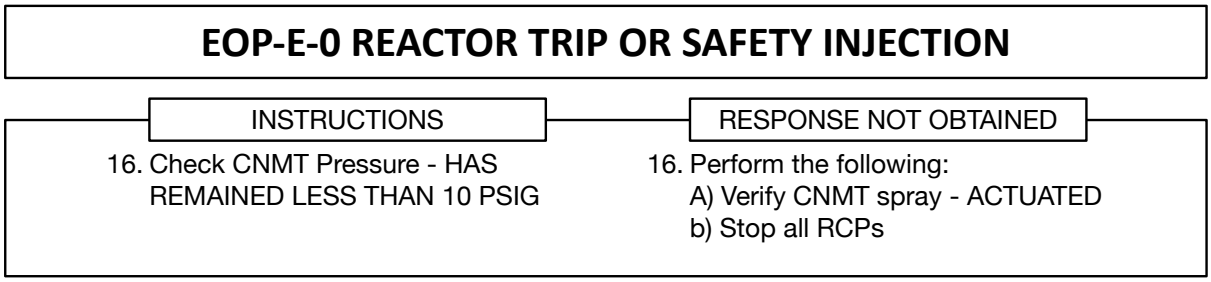


Figure 1: Example Emergency Response Procedure in the Two-Column Format

These types of procedures follow the two-column format in which the left Instruction column is the primary route towards a goal plant state and the right Response Not Obtained (RNO) column is used when criteria are not satisfied while completing a step in the left column. To complete a procedure, the operator executes each procedure step sequentially until all are completed, or the operator encounters a step with criteria that are unsatisfied and alternative activities are a required. Immediately after encountering failed criteria, the operator branches into the corresponding RNO column step, denoted with the same step number or substep identifier. The RNO step may entail performing other actions or require a transition to another step or another procedure entirely. This basic step execution forms the basis for the HUTNER software architecture and can be seen in Figure 2. These elements include the following basic structures:

- ***Step***—refers to a small set of activities that represent the base organization unit defined within the *element* class in the software. Procedures are commonly organized in two columns of steps as exemplified by those used at Westinghouse pressurized-water reactors. The primary tasking steps occur in the left column, denoted as instructions. If the criteria for the instructions are not met, then the operator moves to the right RNO column, which offers an alternative set of activities that may align with corresponding instruction column's goal. Not all instruction column steps have a corresponding RNO step, depending on the nature of the procedure. By design, all procedures attempt to be closed such that any step's success or failure branches to another step within the same procedure or transitions to a different procedure.
- ***Substep***—refers to the secondary tasking that is performed within a main procedure step. Often a procedure step requires multiple substeps by operators to complete the desired tasking. While

steps are numbered, substeps are usually treated alphabetically—Step 1a, 1b, etc.—meaning main Step 1 followed by substeps *a* and *b*.

- ***Logic***—each step or substep contains a statement that instructs the operator to act or determine a particular plant state. The logic contains the *object* which is a component or more specifically a parameter of a component and boolean *criteria* to evaluate the object's state. The result of the logic evaluation dictates whether the step *can* succeed or fail.

To support the dynamic human reliability aspects of the simulation, contextual information is integrated with the procedure steps. This is done because each step activity is assigned a GOMS-HRA primitive to provide a time duration associated with completion of the task and HEP for the task [2][3]. The simulation is able to use the plant state to evaluate the context to calculate nominal time durations and HEPs. Additional details on the modules evaluating the context and plant state is provided in a subsequent section.

- ***Primitive***—refers to the GOMS-HRA task level primitives ([2]HUTNER REPORT)[4], which the HUNTER code uses to determine time durations and nominal HEPs for tasks. More than one task level primitive may be associated with a step.
- ***PSFs***—refers to performance shaping factors that are dynamically or statically calculated during the simulation and applied to the *primitive*. The nominal HEP and time duration sampled during the simulation are multiplied by the combined PSFs to adjust the simulated time duration and the HEP value at each time point in the simulation.
- ***Result***—for a given step to be executed successfully, the logic must be upheld and the primitive evaluation must be evaluated as within the available time and exceed the success criteria for the HEP.

3. SIMULATION MODEL CONTRUCTION

The simulation requires an analyst to load a simulation model that defines all aspects of the simulation including the following:

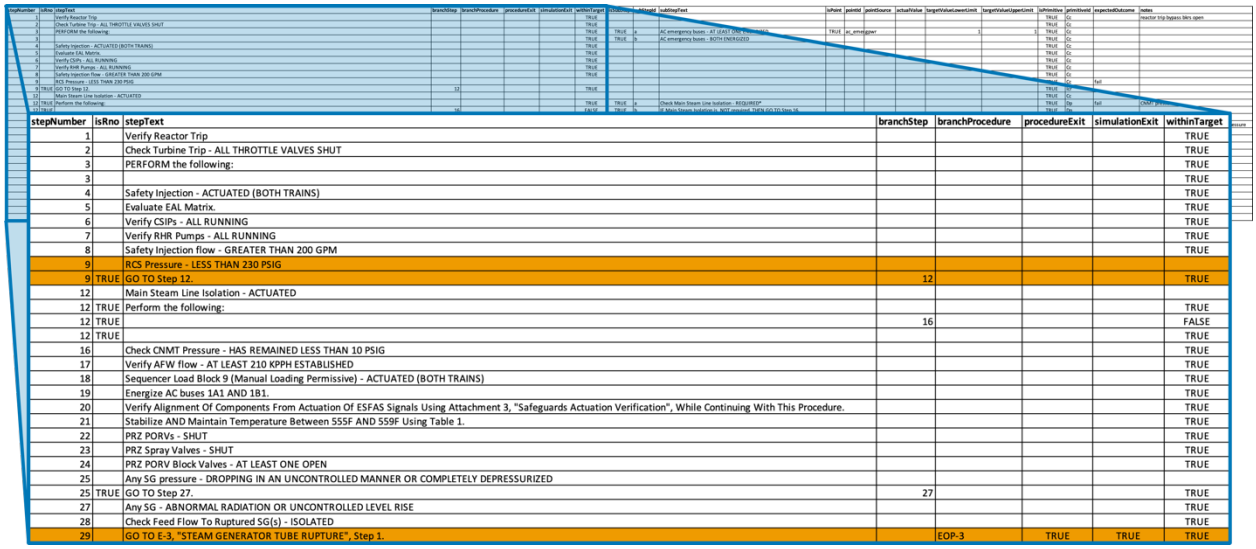
1. Tasks, procedures, and steps mapped with specific plant parameters
2. Performance shaping factors with levels mapped to particular steps or definitions of their general equations for calculation over the simulation
3. Meta simulation parameters including number of Monte Carlo runs and result output format of the simulation parameters and the task and procedure objects

To construct a model of these elements, an analyst enters these elements into a formatted file following a specific structure and coding scheme. The model file format based on comma-separated values (CSV) was selected as the primary means for an analyst to construct the model, since it provides a table-based format that lends itself to simple manual entry of procedure elements with widely available commercial software. This format shares some similarity to input deck files used for other codes such as RELAP5-3D [5]. In the current application version, the analyst must manually enter each model element; however, a module to automatically parse procedure documents and populate the table with procedure elements is under active development and will be included in a future release of the software. Furthermore, analysts can bypass the CSV manual entry by constructing models in a recently developed graphical user interface (GUI) for HUNTER that allows analysts to more easily create and edit the model file within the same application that runs the simulation and displays the results. A description of this GUI is outside this scope of the basic architecture described here, but additional details can be found in an accompanying paper in this same series [6].

The analyst must also populate details for the human reliability context and the virtual plant context for each procedure. Specifically, the analyst can prescriptively drive the procedure path by specifying plant parameters needed to support procedure logic at branch points. The code also supports defining multiple possible paths toward different outcomes. With the open-ended procedure construction, the analyst must

select a supporting simulator and define specific parameters associated with each step that are evaluated at run time to determine the success or failure of a given step. Where external simulation codes are not available, information may be dummy coded to support HUNTER's progression through the procedures. In both the open-ended and dummy mode of procedure execution, the analyst can also define the human reliability parameters for each step to add the HRA layer onto the procedure execution. In this simulation mode, the procedure uses the provided context information to calculate an HEP for each step, which yields a success or failure outcome and can change the course of the simulated procedure path.

Each row of the input deck represents a step level item and must have a step number (see Figure 2). This step number is used to group any additional related rows (e.g., all rows tagged as 12 are processed by the parser as a collection of sub-elements to be grouped within step 12). As the number of items contained within a step is variable, additional rows with the step number tag support the ability to add as many elements as needed without complex reformatting of the model CSV file.



stepNumber	isRno	stepText	branchStep	branchProcedure	procedureExit	simulationExit	withinTarget
1		Verify Reactor Trip					TRUE
2		Check Turbine Trip - ALL THROTTLE VALVES SHUT					TRUE
3		PERFORM the following:					TRUE
3							TRUE
4		Safety Injection - ACTUATED (BOTH TRAINS)					TRUE
5		Evaluate EAL Matrix.					TRUE
6		Verify CSIPs - ALL RUNNING					TRUE
7		Verify RHR Pumps - ALL RUNNING					TRUE
8		Safety Injection flow - GREATER THAN 200 GPM					TRUE
9		RCS Pressure - LESS THAN 230 PSIG					
9	TRUE	GO TO Step 12.	12				TRUE
12		Main Steam Line Isolation - ACTUATED					
12	TRUE	Perform the following:					TRUE
12	TRUE		16				FALSE
12	TRUE						TRUE
16		Check CNMT Pressure - HAS REMAINED LESS THAN 10 PSIG					TRUE
17		Verify AFW flow - AT LEAST 210 KPPH ESTABLISHED					TRUE
18		Sequencer Load Block 9 (Manual Loading Permissive) - ACTUATED (BOTH TRAINS)					TRUE
19		Energize AC buses 1A1 AND 1B1.					TRUE
20		Verify Alignment Of Components From Actuation Of ESFAS Signals Using Attachment 3, "Safeguards Actuation Verification", While Continuing With This Procedure.					TRUE
21		Stabilize AND Maintain Temperature Between 555F AND 559F Using Table 1.					TRUE
22		PRZ PORVs - SHUT					TRUE
23		PRZ Spray Valves - SHUT					TRUE
24		PRZ PORV Block Valves - AT LEAST ONE OPEN					TRUE
25		Any SG pressure - DROPPING IN AN UNCONTROLLED MANNER OR COMPLETELY DEPRESSURIZED					
25	TRUE	GO TO Step 27.	27				TRUE
27		Any SG - ABNORMAL RADIATION OR UNCONTROLLED LEVEL RISE					TRUE
28		Check Feed Flow To Ruptured SG(s) - ISOLATED					TRUE
29		GO TO E-3, "STEAM GENERATOR TUBE RUPTURE", Step 1.		EOP-3	TRUE	TRUE	TRUE

Figure 2: Region of the HUNTER Input CSV File Demonstrating Additional Elements Under the Same Procedure Step

An intermediary process converts the CSV file upon opening and initializing the HUNTER application with a parser class into a JavaScript Object Notation (JSON) format that is then directly consumed by the scheduler class. Analysts can construct their models in JSON, but the structure of these files is not as readily human readable as CSV files. This parser class allows the analyst to work with the more easily human read and edited CSV file, while the JSON format is much easier for the application to consume. The CSV file can be edited and viewed as a spreadsheet table, meaning it has consistent fields mapped across a matrix. In contrast, a JSON file does not populate cells without information, and only fields that are actively changed are conveyed in the file. Thus, the structure of a JSON files is much less linear in appearance and more less humanly readable.

The parser file relies on dummy coding to process each row, or subsequent rows, as a part of a single procedure step. The input file can be divided into the four nesting levels of step, substep, logic, and HRA as depicted in Figures 2 - 4, respectively.

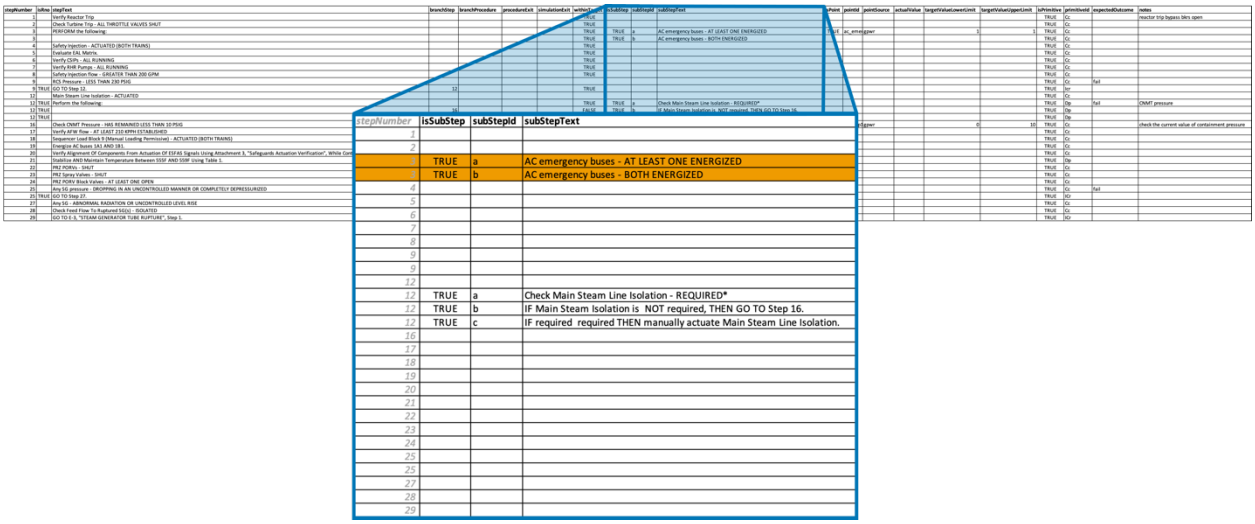


Figure 3: Procedure Substep Region of the HUNTER Input File

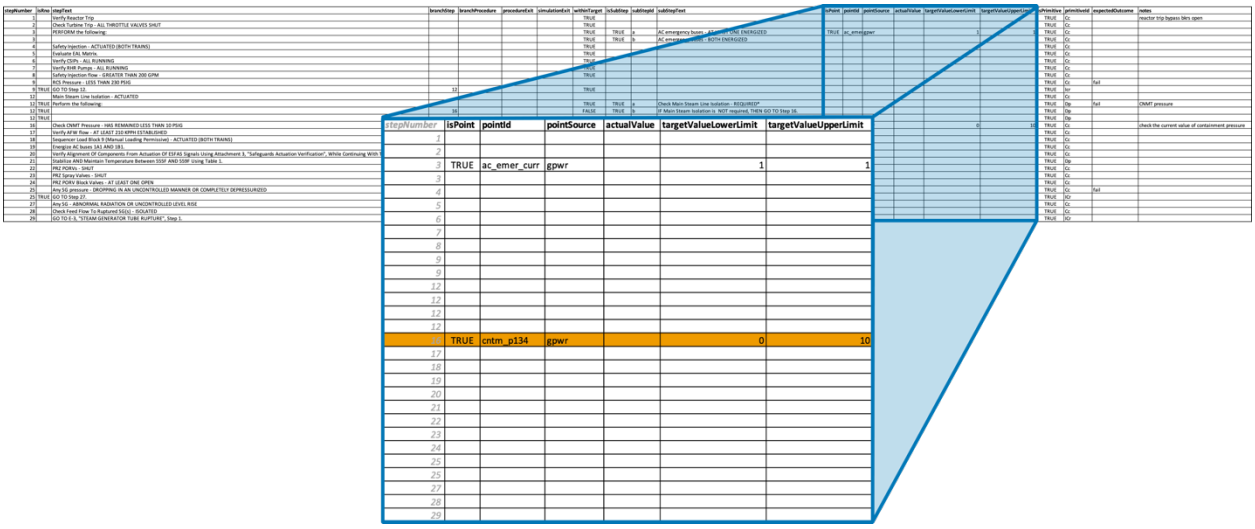
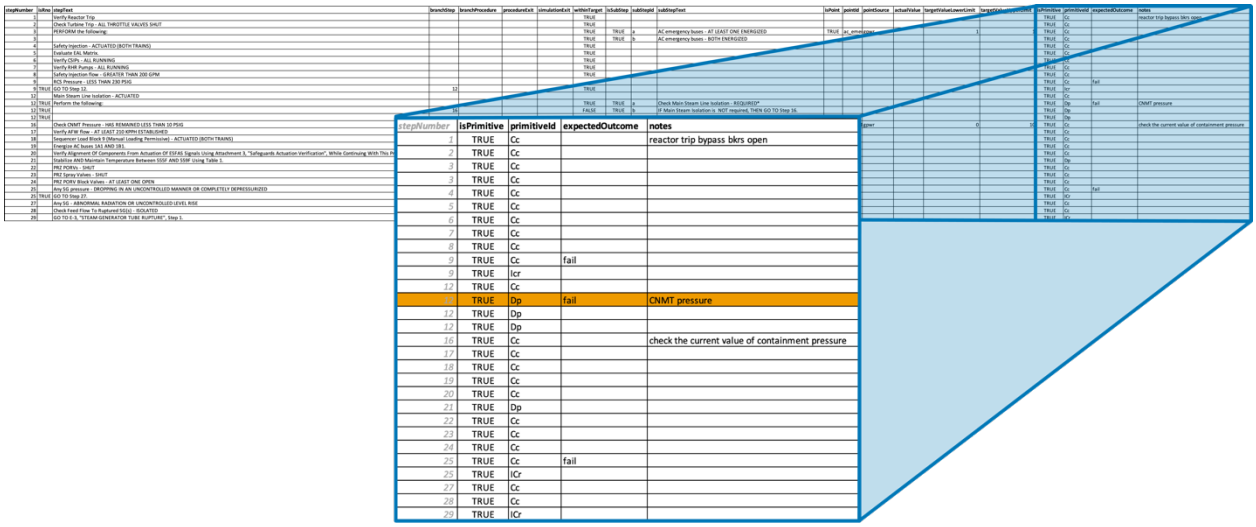


Figure 4: Logic Region of the HUNTER Input File Denoting Names of Plant Parameters

Populating any row of these four regions requires adding TRUE to the corresponding parser flag columns, which are labeled *isRno*, *isSubStep*, *isPoint*, and *isPrimitive*, based on their associated element type. For example, Figure 2 above shows the left region of the CSV input file for a procedure with the leftmost column indexing each step as a numerical number. The next column, *isRno*, is a parser flag column used to denote if the step is a response obtained or an RNO type of step. Step 9 in the figure, which is highlighted in orange, demonstrates the dummy coding for a step with a response obtained. The second row of Step 9 shows an RNO step. The second nesting region, the substep region shown in Figure 3, follows exactly the same organization but for substeps. Step 3 in the figure shows substeps *a* and *b* appearing as two sequential rows. Similar to adding an RNO element, adding substeps requires adding rows and tagging those rows with the step number. It is also possible to combine RNO and substeps following this same structure and coding scheme.



stepNumber	isPrimitive	primitiveId	expectedOutcome	notes
1	TRUE	Cc		reactor trip bypass bkrs open
2	TRUE	Cc		
3	TRUE	Cc		
3	TRUE	Cc		
4	TRUE	Cc		
5	TRUE	Cc		
6	TRUE	Cc		
7	TRUE	Cc		
8	TRUE	Cc		
9	TRUE	Cc	fail	
9	TRUE	Icr		
12	TRUE	Cc		
10	TRUE	Dp	fail	CNMT pressure
12	TRUE	Dp		
12	TRUE	Dp		
16	TRUE	Cc		check the current value of containment pressure
17	TRUE	Cc		
18	TRUE	Cc		
19	TRUE	Cc		
20	TRUE	Cc		
21	TRUE	Dp		
22	TRUE	Cc		
23	TRUE	Cc		
24	TRUE	Cc		
25	TRUE	Cc	fail	
25	TRUE	ICr		
27	TRUE	Cc		
28	TRUE	Cc		
29	TRUE	ICr		

Figure 5: GOMS-HRA Task-Level Primitives Found in HUNTER Input File

The third and fourth nesting regions, shown in Figures 4 and 5, depict the point and primitive fields, respectively. These elements can be nested at the step or substep level. Therefore, any element added to this row must ensure that the *isPoint* and *isPrimitive* parser flag columns are set in addition to the *isSubStep* column if the point or primitive should be nested within a substep item.

4. SIMULATION MODEL EXECUTION

This section describes the modules as they are used to run each of the Monte Carlo simulations. Before describing the individual modules, the plant model itself is worth describing. The plant model is not in itself a module, but it is a crucial aspect of the overall simulation and enables much of the dynamic capabilities provided by HUNTER. A plant model is run in tandem to track the plant state as the virtual operator manipulates components based on the procedure as well as the natural progression of the simulation without intervention, which is typically initiated with a fault and progresses towards a system failure state without operator intervention. Therefore, the virtual operator's success or failure is largely governed by the amount of time remaining to mitigate a given fault based on the amount of time required to complete the mitigating activities. The scheduler travels along the procedure path while polling the plant state through an API module to determine whether key parameters have exceeded threshold values indicating a system failure and the end of that simulation run. In its current version, the API module supports communication with RELAP5-3D [5] and Rancor Microworld [7] simulations.

4.1 Scheduler Module

The Scheduler module includes several classes that collectively perform Monte Carlo based simulations of the task defined through the CSV input files. The scheduler acts as the executive for what is being done by which module and when. Practically, it serves as the placekeeper for the other modules. The scheduler stores analyst-defined configurations for the overall simulation in a configuration class that is accessible throughout the simulation to serve as a central data repository for the application. The Scheduler module has access to all the other modules and contains several subclasses itself, most notably the log class that outputs data to CSV log files.

The Log class serves as the historian and performs input/output functions to record each simulation run in CSV output files. The log class also contains some debugging capabilities to assist analysts in testing the CSV input files and logging errors in procedure path execution, such as an unclosed procedure path with no possibility to advance. As the scheduler is executing simulation runs, it is monitoring the runs to cease any failed runs and move to the next run attempt.

4.2 Task Module

Within each simulation run, the Scheduler calls the Task module to execute the procedure steps. The Task module contains the classes that store and manipulate the activity executed during each simulation run. The Task module contains the procedures with their steps. The module can execute unlinked procedures in sequence, but in practice the task typically begins on a specific procedure at a specific step and then the steps themselves contain all the information to guide the simulation since each step contains the appropriate transition to other procedures in the task. Specifically, each step contains a transition object that contains an identifier for the next step based on the results of its own successful or failed execution. These transitions can be:

1. The next instruction column step within the procedure sequence
2. The adjacent RNO column step or substep, or a step in another procedure

Each step is executed, and the result of the execution contains the appropriate next step item. The steps within the procedures of the task are completed until no more instruction steps remain or an end simulation flag is encountered. While executing each step, the HRA module calculates the elapsed time required to perform and an HEP for each step.

4.3 HRA Module

The Task module relies on the HRA module to evaluate the plant context and performs two key functions. First, it calculates an elapsed time for each step or substep activity and increments the simulated time for the main simulation *and* the plant simulation. This is important to progress the plant state in step with the time durations required to perform each step activity. Second, the HRA module calculates an instantaneous HEP, which is used to alter the course of the procedure path or incur a time debt. The HEP aspects of the module are still quite limited, but the simulator can fail a step based on a calculated HEP exceeding an analyst defined threshold. Based on the analyst defined configuration, the step is repeated, and a time debt is incurred, which over several failed steps can lead key parameters within the plant simulation to exceed their thresholds such that the simulation run ends in a system failed state. The other approach that is supported currently is to treat the HEP threshold exceedance as a procedure logic failure and follow the failed procedure step transition to the next appropriate step. In this configuration, the failure leads the virtual operator down the wrong path. The critical element that binds the Task and HRA modules together is the Step module, which contains the parameters used by the HRA module.

4.4 Step Module

The Step module is responsible for executing each step, which entails evaluating the plant state against the logic defined within the step and evaluating the HRA context. The Step module contains a step class, which serves as the data structure to store parameter objects defining each step. One key parameter object is the logic which contains the plant model component parameter and the logic test used to evaluate the state of the component. The step class also contains several HRA parameter objects. GOMS-HRA task-level primitives are defined in the primitive subclass parameter object. Each step can contain multiple primitives to represent more complicated activities, but in practice there should never be more than two based on initial simulator validations. Ideally a single primitive should be assigned to each step or substep, but due to the variability within procedure steps, more complicated steps that do not contain any substeps may require two primitives to capture the intended tasks. Primitives are defined with the GOMS-HRA coding scheme, such as C_C to denote the "check in the control room" task-level primitive. Each task-level primitive type has a predefined execution time distribution and HEP associated with it. The various contexts surrounding the virtual operator are defined in the PSF class. Relevant PSFs from SPAR-H can be assigned with default levels corresponding to specific multiplier values that are applied on top of the base HEP linked to the GOMS-HRA primitive.

Figure 2 depicts the different data elements and their relationships in terms of how the Step module evaluates each procedure step. The evaluation of the step serves both as a datapoint recorded in the log of the simulation run and dictates the next procedure step selection for the simulation. Figure 3 below depicts how the Step module evaluates each step by using the API module and HRA module to gather plant state data to evaluate the step logic and evaluate the HRA context.

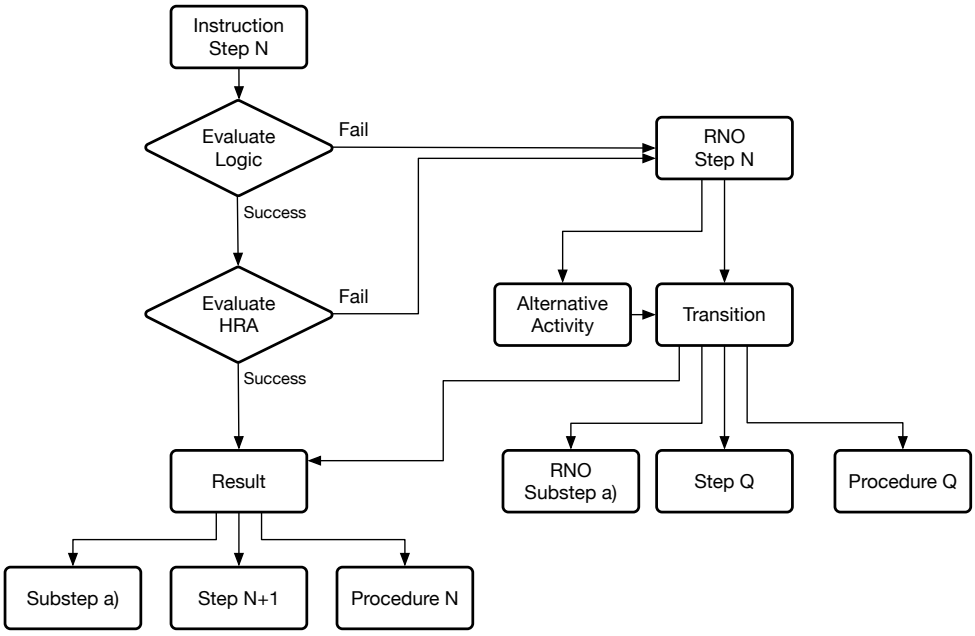


Figure 2: Basic Procedure Step Execution

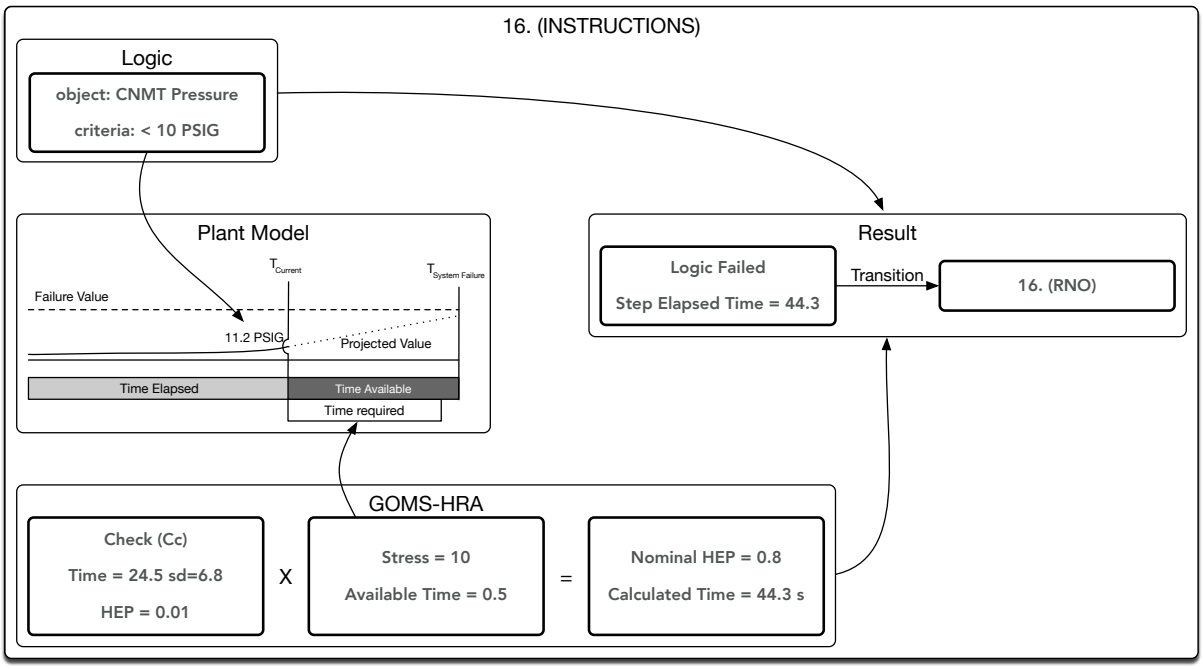


Figure 3: Step Module Execution Example Using the Instruction Step, Left Column, from the Sample Procedure in Figure 1

Not all of these elements must be defined within each step. In fact, HUNTER can be run without the plant model to perform time uncertainty quantification estimates that purely look at the variability in the execution of particular tasks. In this manner, the simulation model must define the outcome for each step, since there is no way to evaluate the logic in the absence of a plant model running in tandem. The intent of this dummy mode is to support examining the human reliability variables along a prescribed path to examine known scenarios or validate to an empirically observed scenario.

5. CONCLUSION

The HUNTER software was developed to bring together the disparate capabilities contained within its framework that shares the same name. Modelling simplicity served as the guiding philosophy for its development. Indeed, the simple proceduralized representation of a virtual operator aims to achieve this simplicity goal and alleviate analysts from constructing complicated models to drive virtual operator action. However, after this initial first round of development the need for some cognitive aspects has become apparent. The treatment of human error induced failures is quite simplistic in the software's current implementation. When a step fails due to a large calculated HEP, the simulation either repeats the step to represent a fairy insignificant human error or the procedure path diverges from the logically evaluated state calculated with values from the simulated plant model to represent a more substantial human error. When should either of these two failure outcomes be used over the other? The capability to perform the two was developed to test each option, but it raised the issue of how to treat human errors in general since there are many more failure modes possible and plausible. A cognitive model could potentially drive the operator response, but this would quickly complicate the modelling and render the simulation too cumbersome for wide use. Instead, a more simplistic approach could entail a coding scheme to map human error types to operator responses. For example, an operator might sample the wrong value and this value does uphold the logical evaluation. The HEP is no longer used only to calculate the outcome of the task, but instead it is used to also determine how the information is used to interpret the step logic or perform the control action. Certainly, this approach could provide a richer set of outcomes than the binary success or failure that is currently modelled. This is a promising area of research that will be pursued in future work.

Despite some of the shortcomings contained within this initial version, the HUTNER software provides a simplified means to create and run dynamic HRA models in tandem with a plant model. The overall goal of the HUNTER simulation is to enable better quantification of HEPs, such that the simulation can run in tandem with a larger dynamic PRA simulation or be used to build more accurate tables of HEP estimates that are suitable as inputs for dynamic PRA. The simulation is under active development to increase its functionality and achieve a technology maturity sufficient for widespread release in the hopes of more accurately capturing the nature of human error in complex nuclear process control.

Acknowledgements

This work of authorship was prepared as an account of work sponsored by Idaho National Laboratory (under Contract DE- AC07-05ID14517), an agency of the U.S. Government. Neither the U.S. Government, nor any agency thereof, nor any of their employees makes any warranty, express 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

[1] Gertman, D., Blackman, H., Marble, J., Byers, J., & Smith, C. (2005). The SPAR-H human reliability analysis method. *US Nuclear Regulatory Commission*, *230*(4), 35.

[2] Ulrich, T., Boring, R., L., Ewing, S., & Rasmussen, M. (2017a). Operator timing of task level primitives for use in computation-based human reliability analysis. *Advances in Intelligent Systems and Computing, 589,* 41-49.

[3] Boring, R., Ulrich, T., & Rasmussen, M. (2018). Task level errors for human error prediction in GOMS-HRA. In *Safety and Reliability–Safe Societies in a Changing World* (pp. 433-439): CRC Press.
[4] Boring, R.L., & Rasmussen, M. (2016). GOMS-HRA: A method for treating subtasks in dynamic human reliability analysis. *Risk, Reliability and Safety: Innovating Theory and Practice, Proceedings of the European Safety and Reliability Conference,* pp. 956-963.
[5] Fletcher, C. D., & Schultz, R. R. (1992). *RELAP5/MOD3 code manual* (No. NUREG/CR-5535-Vol. 5; EGG-2596-Vol. 5). Nuclear Regulatory Commission, Washington, DC (United States). Div. of Systems Research; EG and G Idaho, Inc., Idaho Falls, ID (United States).
[6] Ahn, J., Boring, R, Ulrich, T., Park, J., and Heo, Y. (2022). The HUNTER Dynamic Human Reliability Analysis Tool: Graphical User Interface. Proceedings of the Probabilistic Safety Assessment and Management (PSAM 16) Conference.
[7] Ulrich, T. A., Lew, R., Werner, S., & Boring, R. L. (2017, September). Rancor: a gamified microworld nuclear power plant simulation for engineering psychology research and process control applications. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Vol. 61, No. 1, pp. 398-402). Sage CA: Los Angeles, CA: SAGE Publications.