International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)
Sun Valley, Idaho, USA, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013)

RAVEN AS A TOOL FOR PROBABILISTIC RISK ASSESSMENT: SOFTWARE OVERVIEW

A. Alfonsi, C. Rabiti, D. Mandelli, J.J. Cogliati, R.A. Kinoshita
Idaho National Laboratory
2525 Fremont Avenue, Idaho Falls, ID 83415
{andrea.alfonsi, cristian.rabiti, diego.mandelli, joshua.cogliati, robert.kinoshita}@inl.gov

ABSTRACT

[To be added]

Key Words: Reactor Simulation, Probabilistic Risk Assessment, Dynamic PRA, Monte-Carlo, Relap-7

1. INTRODUCTION

RAVEN (**R**eactor **A**nalysis and **V**irtual control **EN**viroment) is a software tool that acts as the control logic driver for RELAP-7. The goal of this paper is to highlight the software structure of the code and its utilization in conjunction with the newly developed Thermo-Hydraylic code RELAP-7. RAVEN is a software framework that allows dispatching the following functionalities:

- Derive and actuate the control logic required to:
  - Simulate the plant control system;
  - Simulate the operator actions (guided procedures);
  - Perform Monte-Carlo sampling of random distributed events;
  - Perform event tree based analysis.
- Provide a Graphical User Interface (GUI) to:
  - Input a plant description to RELAP-7(components, controlled variables, controlled parameters);
  - Concurrent monitoring of Controlled Parameters;
  - Concurrent alteration of Controlled Parameters.
- Provide a post-processing data mining capability based on:

The paper is divided in three main sections:

- RAVEN mathematical framework;
- RAVEN software structure;
- Demonstration of a Station Black Out (SBO) analysis of a Pressurized Water Reactor (PWR).

2. MATHEMATICAL FRAMEWORK

In this section the mathematical framework is briefly described, analyzing the set of the equations needed to model the control system in a nuclear power plant.

2.1. Plant and Control System Model

The first step is the derivation of the mathematical model representing, at a high level of abstraction, both the plant and the control system models. Let be $\bar{\theta}(t)$ a vector describing the plant status in the phase space, and the governing equation:

$$\frac{\partial \bar{\theta}}{\partial t} = \bar{H}(\theta(t), t) \tag{1}$$

In the above equation we have assumed the time differentiability in the phase space. This is generally not required and it is used here for compactness of notation. Now an arbitrary decomposition of the phase space is performed:

$$\bar{\theta} = \begin{pmatrix} \bar{x} \\ \bar{v} \end{pmatrix} \tag{2}$$

The decomposition is made in such a way that $\bar{x}$ represents the unknowns solved by RELAP-7, while $\bar{v}$ are the variables directly controlled by the control system (i.e., RAVEN). Equation 1 can now be rewritten as follows:

$$\begin{cases} \dfrac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}, t) \\ \dfrac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{x}, \bar{v}, t) \end{cases} \tag{3}$$

As a next step, it is possible to note that the function $\bar{V}(\bar{x}, \bar{v}, t)$ representing the control system, does not depend on the knowledge of the complete status of the system but on a restricted subset that we call control variables $\bar{C}$:

$$\begin{cases} \dfrac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}, t) \\ \bar{C} = \bar{G}(\bar{x}, t) \\ \dfrac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{x}, \bar{v}, t) \end{cases} \tag{4}$$

2.2. Operator Splitting Approach

The system of equations 4 is fully coupled and in the past it has always been solved with an operator splitting approach. The reasons for this choice are several:

- Control system reacts with an intrinsic delay
- The reaction of the control system might move the system between two different discrete states and therefore numerical errors will be always of first order unless the discontinuity is treated explicitly.

RAVEN as well is using this approach to solve Eq. 4 which becomes:

$$\begin{cases} \dfrac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}_{t_i-1}, t) \\ \bar{C} = \bar{G}(\bar{x}, t) \\ \dfrac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{x}, \bar{v}_{t_i-1}, t) \end{cases} \tag{5}$$

2.3. The auxiliary plant and component status variables

So far it has been assumed that all information needed is contained in $\bar{x}$ and $\bar{v}$. Even if this information is sufficient for the calculation of the system status in every point in time, it is not a practical and efficient way to implement the control system. In order to facilitate the implementation of the control logic, a system of auxiliary variables has been implemented . The auxiliary variables are those that in statistical analysis are artificially added to non-Markovian systems into the space phase to obtain back a Markovian behavior, so that only the information contained in the previous time step is needed to determine the future status of the system. These variables can be classified into two types:

- Global status auxiliary control variables (e.g., SCRAM status, time at which scram event begins, time at which hot shut down event begins)
- Component status auxiliary variables (e.g., correct operating status, time from abnormal event)

Thus, the introduction of the auxiliary system into the mathematical framework leads to the following formulation of the Eq. 5:

$$\begin{cases} \dfrac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}_{t_i-1}, t) \\ \bar{C} = \bar{G}(\bar{x}, t) \\ \dfrac{\partial \bar{a}}{\partial t} = \bar{A}(\bar{x}, \bar{C}, \bar{a}, \bar{v}_{t_i-1}, t) \\ \dfrac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{x}, \bar{v}_{t_i-1}, t) \end{cases} \tag{6}$$

3. RELAP-7 and MOOSE

MOOSE is a computer simulation framework, developed at Idaho National Laboratory (INL), that simplifies the process for predicting the behavior of complex systems and developing non-linear, multi-physics simulation tools. As opposed to traditional data-flow oriented computational frameworks, MOOSE is founded on the mathematical principle of Jacobian-Free Newton-Krylov (JFNK) solution methods. Utilizing the mathematical structure present in JFNK, physics are modularized into Kernels allowing for rapid production of new simulation tools. In addition, systems are solved fully coupled and fully implicit employing physics based preconditioning which allows for great flexibility even with large variance in time scales. Other than providing the algorithms for the solution of the differential equation, MOOSE also provides all the manipulation tools for the C++ classes containing the solution vector. This framework has been used to construct and develop the Thermo-Hydraulic code RELAP-7, giving an enormous flexibility in the coupling procedure with RAVEN.

RELAP-7 is the next generation nuclear reactor system safety analysis. It will become the main reactor systems simulation toolkit for RISMC (**R**isk **I**nformed **S**afety **M**argin **C**haracterization) project and the next generation tool in the RELAP reactor safety/systems analysis application series (the replacement for RELAP5). The key to the success of RELAP-7 is the simultaneous advancement of physical models, numerical methods, and software design while maintaining a solid user perspective. Physical models include both PDEs (Partial Differential Equations), ODEs (Ordinary Differential Equations) and experimental based closure models. RELAP-7 will eventually utilize well posed governing equations for multiphase flow, which can be strictly verified. RELAP-7 uses modern numerical methods which allow implicit time integration, higher order schemes in both time and space and strongly coupled multi-physics simulations. RELAP-7 is the solver for the plant system except for the control system. From the mathematical formulation presented so far, RELAP-7 solves $\frac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}_{t_i-1}, t)$. The nuclear power plant

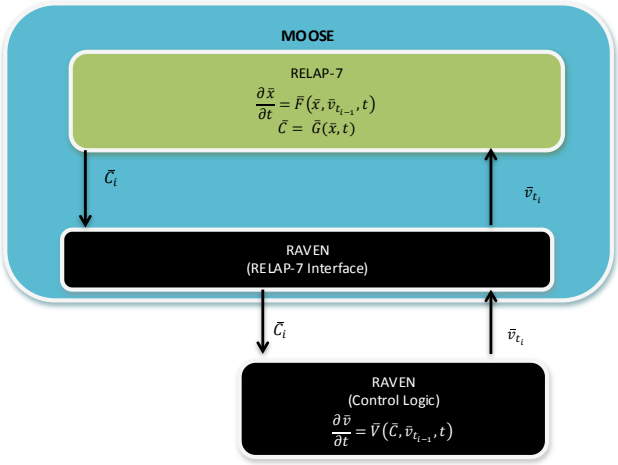

Figure 1. Control System Software Layout.

is represented and modeled by a set of components (Pipes, Valves, Branches, etc.) and each component type corresponds to a C++ class.

4. RAVEN

RAVEN has been developed in a high modular and pluggable way in order to enable easy integration of different programming languages (i.e., C++, Python) and coupling with other applications based on MOOSE and not. The code consists of four modules:

- RAVEN/RELAP-7 interface (see Section 4.1)
- Python Control Logic (see Section 4.2)
- Python Calculation Driver (see Section 4.3)
- Graphical User Interface (see Section 4.4)

4.1. RAVEN/RELAP-7 interface

The RAVEN/RELAP-7 interface, coded in C++, is the container of all the tools needed to interact with RELAP-7/MOOSE. It has been designed in order to be general and pluggable with differen solvers simultaneosly in order to allow an easier and faster development of the control logic/PRA capabilities for multi-physics applications.The interface provides all the capabilities to control, monitor, and process the parameters/quantities in order to drive the RELAP-7/MOOSE calculation. In addiction, it contains the tools to comunicate to the general MOOSE input parser which information, i.e. input syntax, must be inputted in order to run a RAVEN driven calculation. It includes four main sections so far:

- RavenMonitored class;
- RavenControlled class;
- RavenAuxiliary class;
- RavenDistributions class.

The RavenMonitored class provides the connection with the calculation framework in order to retrieve the post-processed quantities from the simulation (i.e. average fuel temperature, average fluid pressure in a component, etc.). The typical input structure for a Monitored parameter in RAVEN is as following:

```
[Monitored]
  [./MaxTempCladCH1]
    component_name = CH1
    operator = NodalMaxValue
    path = CLAD:TEMPERATURE
    data_type = double
  [../]
  [./MaxTempCladCH2]
    component_name = CH2
    operator = NodalMaxValue
    path = CLAD:TEMPERATURE
    data_type = double
  [../]
  ...
[]
```

Whithin the block identified by the keyword **Monitored**, the user can specify the monitored quantities must be processed during the calculation. Each monitored variable is identified through a **Raven Alias** (i.e. MaxTempCladCH1), the name that is used in the control logic python input in order to refer to the parameter contained in the simulation. The user has to provide different information in order to build a monitored variable:

- **component_name**, the name of the RELAP-7 component that contains the variable the code must act on;
- **operator**, the post-processor oparation that must be performed on the variable;

- **path**, the variable name and its location within the calculation framework (RELAP-7/MOOSE variable name);
- **data_type**, data type (i.e. double, float, int, bool).

RAVEN can use all the post-processor operators that are available in MOOSE [reference] (i.e. ElementAverageValue, NodalMaxValue, etc.). Depending on which component it's acting on, some operations may be disabled (i.e. ElementAverageValue is not available in 0-D components).
The RavenControlled class provides the link between RAVEN and RELAP-7/MOOSE in order to retrieve and/or change properties within the simulation (i.e. fuel thermal conductivity, pump mass flow, etc.). The typical input structure for a Controlled parameter in RAVEN is as following:

```
[Controlled]
  control_logic_input = control_logic_input_file_name
  [./power_fraction_CH1]
    property_name = FUEL:power_fraction
    data_type = double
    component_name = CH1
  [../]
  [./power_fraction_CH2]
    property_name = FUEL:power_fraction
    data_type = double
    component_name = CH2
  [../]
  ...
[]
```

Inwardly the block identified by the keyword **Controlled**, the user can specify the properties that, during the calculation, will be controlled through the Python control logic. The name and path of the control logic input file are provided by the parameter **control_logic_input** (not specifying the ".py" extension). Each controlled variable is identified through a **Raven Alias** (i.e. power_fraction_CH1), the name that is used in the control logic python input in order to refer to the parameter contained in the simulation. The user has to provide different information in order to build a controlled variable:

- **component_name**, the name of the RELAP-7 component that contains the variable the code must act on;
- **property_name**, the variable name and its location within the calculation framework (RELAP-7/MOOSE variable name);
- **data_type**, data type (i.e. double, float, int, bool).

Through this class, RAVEN is able to retrieve property values and, in case of changes, push the new values back into the simulation.
The RavenAuxiliary class is the container of auxiliary variables. The Raven Auxiliary system is not connected with RELAP-7/MOOSE enviroment. The typical input structure for a auxiliary parameter in RAVEN is as following:

```
[RavenAuxiliary]
  [./scram_start_time]
    data_type      = double
    initial_value = 61.0
  [../]
  [./CladDamaged]
    data_type      = bool
    initial_value = False
  [../]
  ...
[]
```

Each auxiliary variable is identified through a **Raven Alias** (i.e. CladDamaged), the name that is used in the control logic python input in order to refer to the parameter contained in the RAVEN interface. The user has to provide different information in order to build a auxiliary variable:

- **initial_value**, initialization value;
- **data_type**, data type (i.e. double, float, int, bool).

As previously mentioned, these variables are needed to ensure that system remains Markovian, so that only the previous time step information are necessary to determine the future status of the plant. // The RavenDistributions class contains the algorithms, structures and interfaces for several predifined probability distributions. It is only available in the python control logic, since it is not needed a direct interaction with RAVEN/RELAP-7/MOOSE enviroment. The user can actually choose among nine different types of distribution (i.e. Normal, Triangular, Uniform, Exponential, etc.), each of them, in order to be intialized, requires a different set of parameters. The following input, for example, builds a Normal and a Triangular distribution:

```
[Distributions]
  [./ExampleNormalDis]
    type =  NormalDistribution
    mu = 1
    sigma = 0.01
    xMax = 0.8
    xMin = 0
  [../]
  [./ExampleTriangularDis]
    type   = TriangularDistribution
    xMin  = 1255.3722
    xPeak = 1477.59
    xMax  = 1699.8167
  [../]
  ...
[]
```

The class RavenDistributions is the base of the Monte-Carlo and Dynamic Event Tree capabilities present in RAVEN.

4.2. Python Control Logic

The control logic module is used to drive a RAVEN/RELAP-7 calculation. Up to now it is implemented by the user via Python scripting. The reason of this choice is to try to preserve generality of the approach in the initial phases of the project so that further specialization would be possible and less expensive. The form through which the RAVEN variables can be called is the following:

- Auxiliary.RavenAlias;
- Controlled.RavenAlias;
- Monitored.RavenAlias.

Regarding the RavenDistributions mentioned in the previous section, they are also available for the control logic in a similar form to the other variable (distributions.RavenAlias(allowable list of arguments)). The implementation of the control logic via python is rater convenient and flexible. The user only needs to know few python syntax rules in order to build an input. Although this extreme simplicity, it will be part of the GUI task to automatize the construction of the control logic scripting in order to minimize user effort. A small example of a control logic input is reported below: the thermal conductivity of the gap (thermalConductGap) is set equal to the thermal conductivity of the fuel when the fuel temperature (averageFuelTemperature) is greater than 910 K.

```
import sys
def control_function(monitored, controlled, auxiliary):
    if monitored.averageFuelTemperature > 910:
        controlled.thermalConductGap = controlled.thermalConductFuel
    return
```

4.3. Python Calculation Driver

Analysis of stochastic systems can be extremely challenging due to the complexity and high dimensionality of the system solving equations. An analytical solution is only available for rather simple cases. When an analytical solution is not available, numerical methods are often employed. In order to solve the system governing equations, two main approaches can be followed:

- Determine approximate solutions of the exact problems;
- Determine the exact solution for the approximate models.

Due to the very large complexity and the high dimensionality of the system considered, RAVEN uses the first approach by employing a Monte-Carlo based algorithm. The main idea is to run a set of simulations having different dynamic and static uncertainty of physical parameters, present of noise and initial conditions and terminate them when one of the following stopping conditions are reached:

- Mission time (i.e., an user specified end time);

- Main event (i.e., maximum temperature of the clad or core damage).

These algorithms have been implemented in the python module called Raven Runner. It consists in a python driver which calls RAVEN multiple times, changes initial conditions and seeds the random generator for the distributions. The multiple calculations, required by the employment of these algorithms, can be run in parallel, using queues/sub-process/PBS systems.

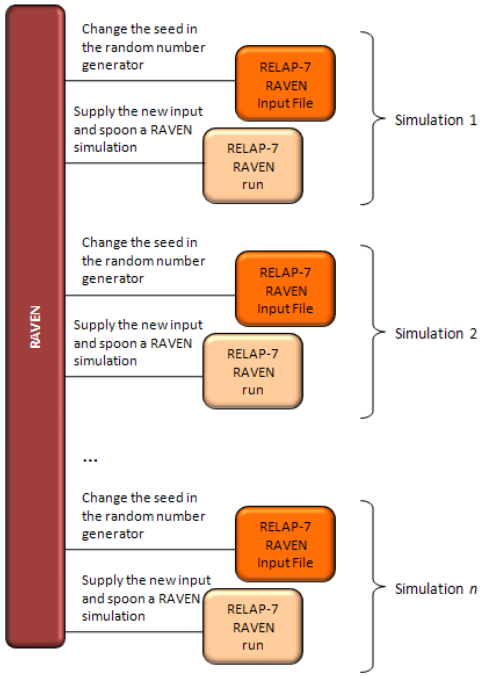

Figure 2. Monte-Carlo sampling scheme.

The analysis of dynamic stochastic systems through Monte-Carlo algorithm can schematized (Figure 2) as following:

1. Initial Sampling of:
   (a) Static and dynamic uncertainty values of physical parameters;
   (b) Initial conditions;
   (c) Transition conditions, i.e. time in which transition events occur (time in which a reactor scram occurs, time delta to recover power grid, etc.).
2. Run the system simulator using the values previously sampled and eventually applying a random noise to some parameters at each time step.;
3. Stop the simulation when a transition condition occurs, and move from the actual status of the system to the new one;
4. Run the simulation as performed in step 3 starting from the new coordinates and stop when a new transition condition occurs;

5. Repeat steps 3 and 4 until a stopping condition is reached;
6. Repeat 1 through 4 for a large number of calculations (user input);

In the Figure 2 is reported a scheme of the interaction between the code and the RAVEN runner in case of Monte-Carlo calculations. The runner basically perform a different seeding of the random number generator and interact, through RAVEN, with the python control logic input in order to sample the variables specified by the user.

4.4. Graphical User Interface

As previously mentioned, a **G**raphical **U**ser **I**nterface is not required to run RAVEN, but it represents an added value to the whole code. The GUI is compatible with all the capabilities actually present in RAVEN (control logic, Monte-Carlo, etc.). Its development is performed using QtPy, which is a Python interface for a C++ based library (Qt) for GUI implementation. The GUI is based on a software named Peacock, which is a GUI interface for MOOSE based application and, in its base implementation, is only able to assist the user in the creation of the input. In order to make it fit all the RAVEN needs, the GUI has been specialized and it is in continuous evolution.

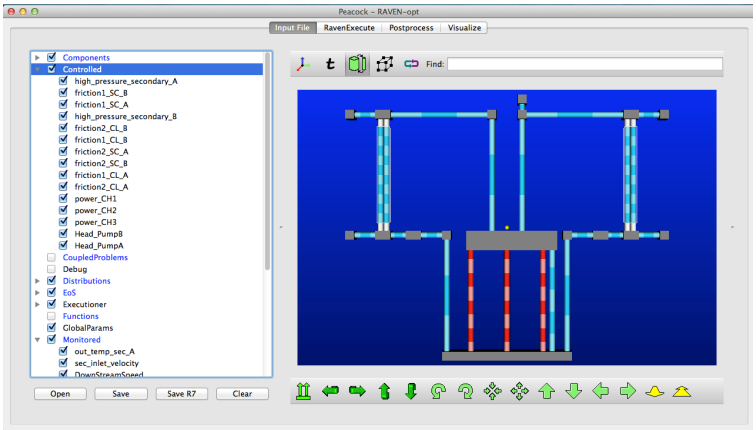

Figure 3. Input/plan Visualization GUI Window.

5. Software layout and calculation flow

At each time step RELAP-7/MOOSE updates the information within the classes with the current solution $\bar{x}$, then RAVEN will ask MOOSE to perform the needed manipulation to construct the monitored quantities $\bar{C}$. Once $\bar{C}$ is constructed, the information is reduced to a vector of numbers understandable by the control system. The equation $\frac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{x}, \bar{v}_{t_i-1}, t)$ is solved and the set of control parameters for the next time step v_{t_i} is obtained. Up to now no situations required a numerical solution of the equation $\frac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{x}, \bar{v}_{t_i-1}, t)$, therefore for the moment RAVEN remains numerical integration free. Once the information is transferred to C, the way through which the plant solution x is computed or stored is irrelevant. The last statement highlights the capability of RAVEN to represent an easily generalizable tool. This functional scheme is represented in Figure 1. To be more specific, in reality MOOSE is made aware of the need to compute at the end of each time step the C as a consequence this is immediately available at the end of each time step.

As a consequence scheme in Figure ?? is more accurate in terms of software implementation. In the following of the discussion depending of which aspect will be more relevant either scheme in Figure 1, Figure 4 or Figure 5 will be referred.

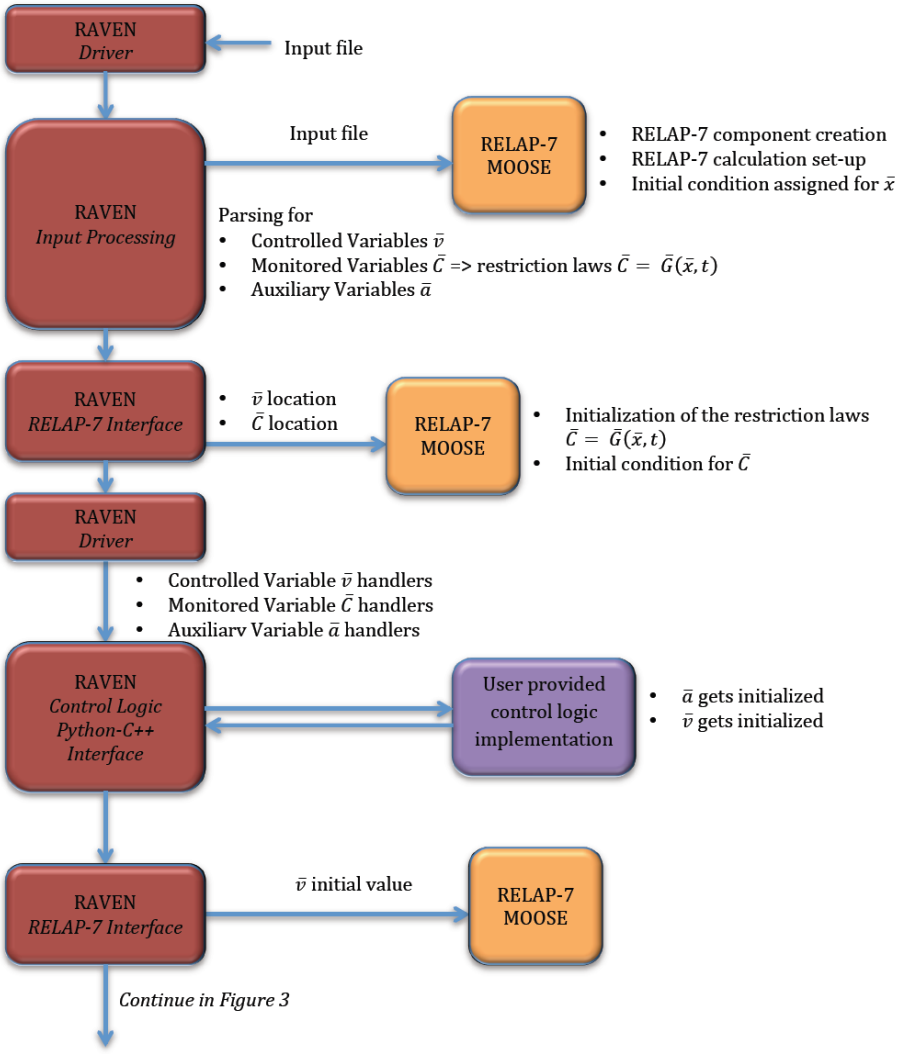

Figure 4. Control System Software Layout.

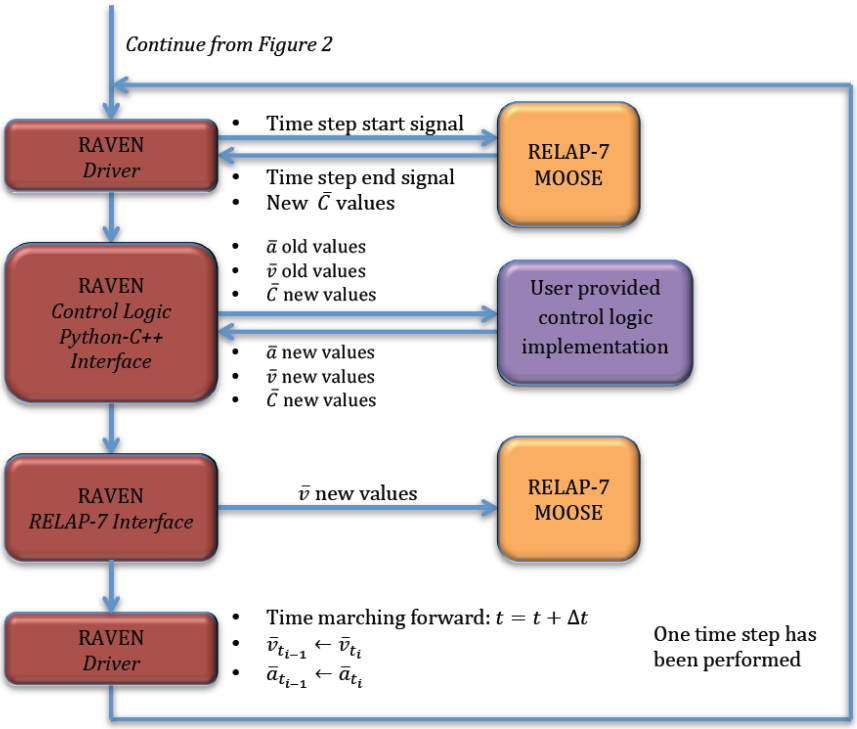

Figure 5. Control System Software Layout.

6. PROBABILISTIC RISCK ASSESMENT DEMO

6.1. Three miles station black out

6.2. Results

7. CONCLUSIONS

Fin!

REFERENCES

[1] B. Author(s), "Title," *Journal Name in Italic*, **Volume in Bold**, pp. 34-89 (19xx).

[2] C. D. Author(s), "Article Title," *Proceedings of Meeting in Italic*, Location, Dates of Meeting, Vol. n, pp. 134-156 (19xx).

[3] E. F. Author, *Book Title in Italic*, Publisher, City & Country (19xx).

[4] "Spallation Neutron Source: The next-generation neutron-scattering facility in the United States," http://www.sns.gov/documentation/sns_brochure.pdf (2002).