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 DYNAMIC 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

RAVEN is a software tool under development at the Idaho National Laboratory (INL) that acts as the control logic driver and post-processing tool for the newly developed Thermo-Hydraylic code RELAP-7. The scope of this paper is to show the software structure of RAVEN and its utilization in connection with RELAP-7. A short overview of the mathematical framework behind the code is presented along with its main capabilities such as on-line controlling/monitoring and Monte-Carlo sampling. A demo of a Station Black Out PRA analysis of a simplified Pressurized Water Reactor (PWR) model is shown in order to demonstrate the Monte-Carlo and clustering capabilities.

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) [1,2] is a software tool that acts as the control logic driver for the newly developed Thermo-Hydraylic code RELAP-7. The goal of this paper is to highlight the software structure of the code and its utilization in conjunction with RELAP-7. RAVEN is a multi-purpose Probabilistic Risk Assement (PRA) software framework that allows dispatching different functionalities. It is designed to derive and actuate the control logic required to simulate the plant control system and operator actions (guided procedures) and to perform both Monte-Carlo sampling of random distributed events and event tree based analysis. In order to facilitate the input/output handling, a Graphical User Interface (GUI) and a post-processing data mining module, based on dimensionality and cardinality reduction, are available. This paper wants to provide an overview of the software, highlighting the mathematical framework from which its structure is derived and showing a demo of a Station Black Out (SBO) analysis of a simplified Pressurized Water Reactor (PWR) model.

2. MATHEMATICAL FRAMEWORK

In this section the mathematical framework is briefly described by 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. In this respect, let be $\bar{\theta}(t)$ a vector describing the plant status in the phase space; the dynamic of both plant and control system can be summarized by the following 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 in Eq. 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 it 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 introduced . 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 [3] 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 based 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 by 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) [4] 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 Section 2, RELAP-7 solves $\frac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}_{t_i - 1}, t)$.

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 including the ones based on MOOSE. The code consists of four modules:

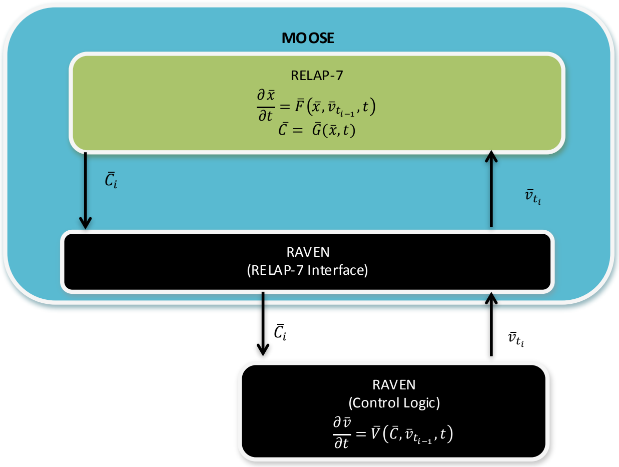

MOOSE
RELAP-7
$\frac{\partial \bar{x}}{\partial t} = \bar{F}(\bar{x}, \bar{v}_{t_{i-1}}, t)$
$\bar{C} = \bar{G}(\bar{x}, t)$
$\bar{C}_i$
$\bar{v}_{t_i}$
RAVEN
(RELAP-7 Interface)
$\bar{C}_i$
$\bar{v}_{t_i}$
RAVEN
(Control Logic)
$\frac{\partial \bar{v}}{\partial t} = \bar{V}(\bar{C}, \bar{v}_{t_{i-1}}, t)$

Figure 1: Control System Software Layout.

- 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 different solvers simultaneously 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 addition, it contains the tools to communicate to the MOOSE input parser whose information, i.e. input syntax, must be received as input in order to run a RAVEN calculation. So far, the input file includes four main sections.:

- *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 (e.g., average fuel temperature, average fluid pressure in a component). 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
   [../]
   ...
[]
```

Within the block identified by the keyword **Monitored**, the user can specify the monitored quantities that need to 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 the following 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 operation 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, boolean).

RAVEN can use all the post-processor operators that are available in MOOSE (e.g., ElementAverageValue, NodalMaxValue, etc.). Depending on which component it's acting on, some operations may be disabled (for example, 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 (e.g., fuel thermal conductivity, pump mass flow). The typical input structure for a controlled parameter in RAVEN is as follows:

```
[ 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
   [../]
   ...
[]
```

Within 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** (e.g., 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, boolean).

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 environment. The typical input structure for a auxiliary parameter in RAVEN is as follows:

```
[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** (e.g., 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 predefined probability distributions. It is only available in the `Python` control logic, since it is not needed a direct interaction with RAVEN/RELAP-7/MOOSE environment. The user can actually choose among nine different types of distribution (e.g., Normal, Triangular, Uniform, Exponential); each of them, in order to be initialized, requires a different set of parameters depending on the type of distribution. An an example, the following input create 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 is possible and inexpensive. The form through which the RAVEN variables can be called is the following:

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

Regarding the RavenDistributions mentioned in Section 4.1, 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
```

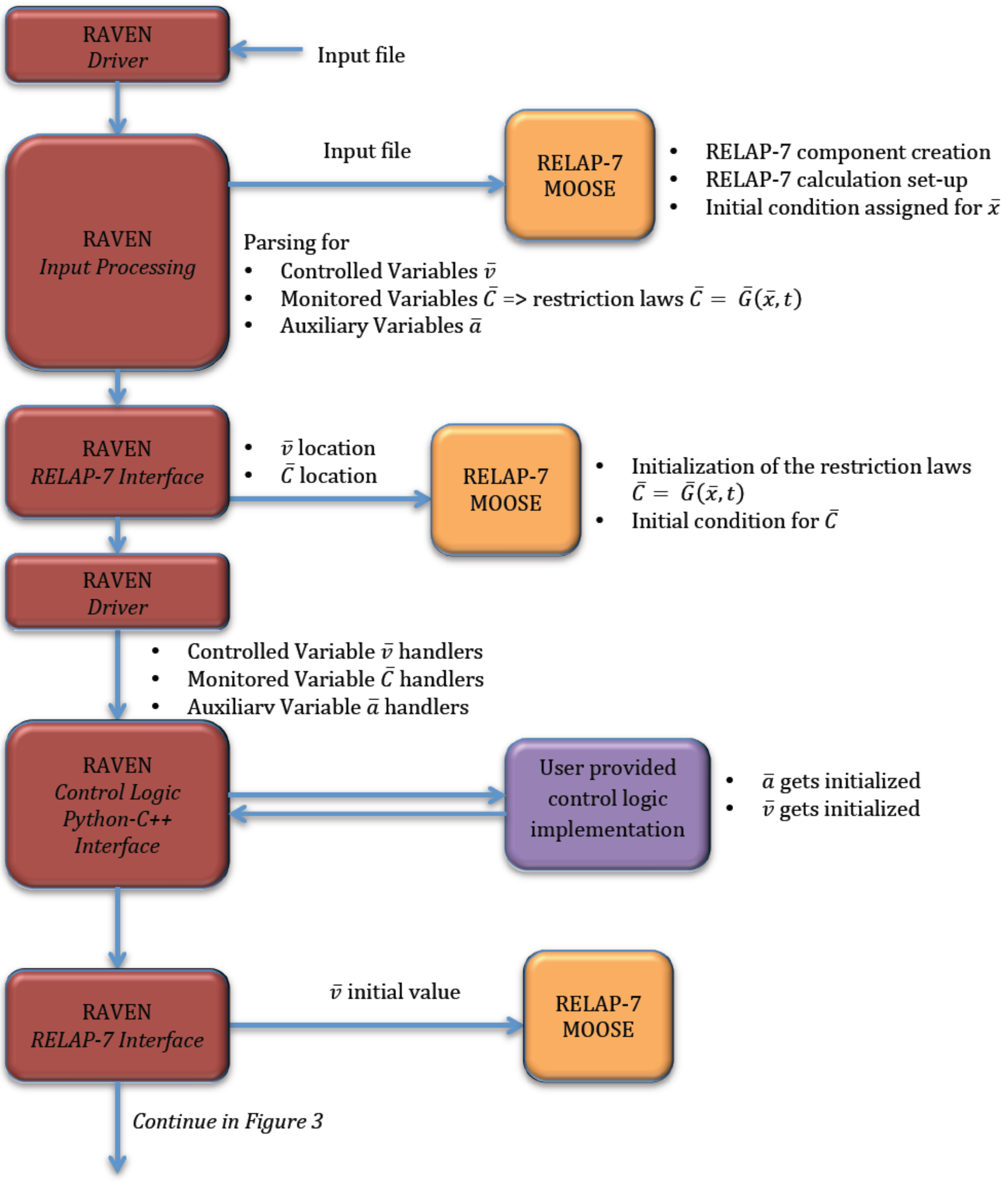

RAVEN Driver
Input file
RAVEN Input Processing
Input file
RELAP-7 MOOSE
• RELAP-7 component creation
• RELAP-7 calculation set-up
• Initial condition assigned for $\bar{x}$
Parsing for
• Controlled Variables $\bar{v}$
• Monitored Variables $\bar{C}$ => restriction laws $\bar{C} = \bar{G}(\bar{x}, t)$
• Auxiliary Variables $\bar{a}$
RAVEN RELAP-7 Interface
• $\bar{v}$ location
• $\bar{C}$ location
RELAP-7 MOOSE
• Initialization of the restriction laws $\bar{C} = \bar{G}(\bar{x}, t)$
• Initial condition for $\bar{C}$
RAVEN Driver
• Controlled Variable $\bar{v}$ handlers
• Monitored Variable $\bar{C}$ handlers
• Auxiliary Variable $\bar{a}$ handlers
RAVEN Control Logic Python-C++ Interface
User provided control logic implementation
• $\bar{a}$ gets initialized
• $\bar{v}$ gets initialized
RAVEN RELAP-7 Interface
$\bar{v}$ initial value
RELAP-7 MOOSE
Continue in Figure 3

Figure 2: RAVEN Calculation Flow - Initialization.

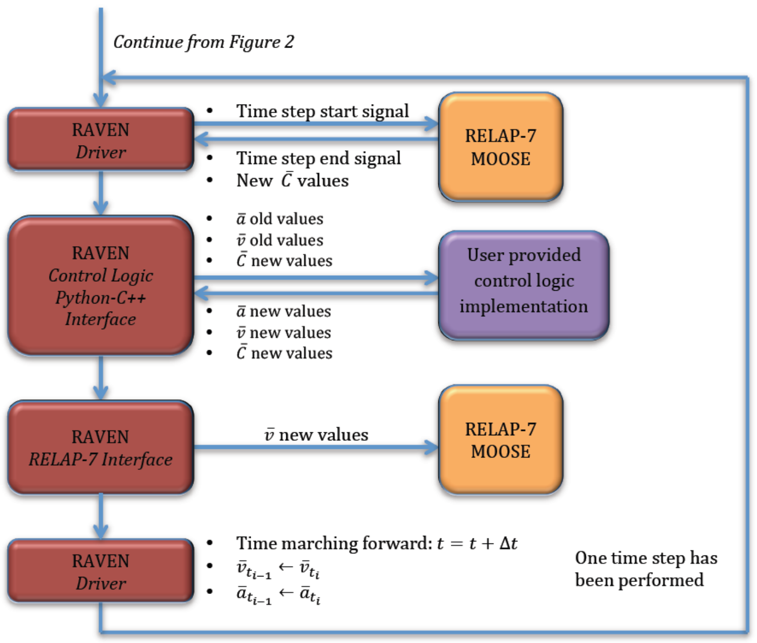


Figure 3: RAVEN Calculation Flow - Run.

4.3. Python Calculation Driver

Analysis of dynamic 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

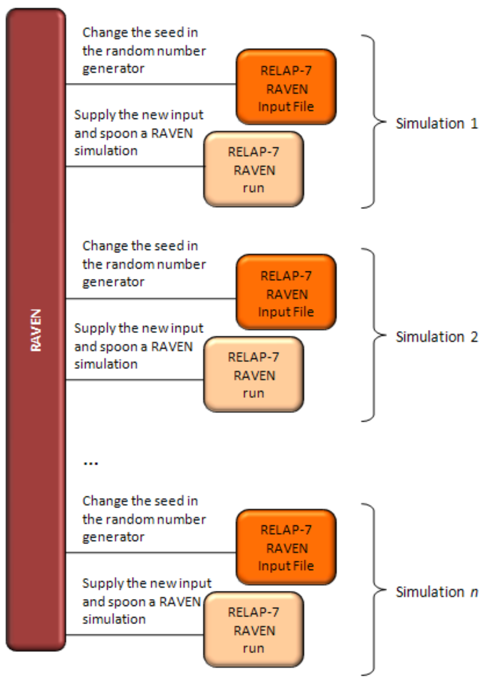
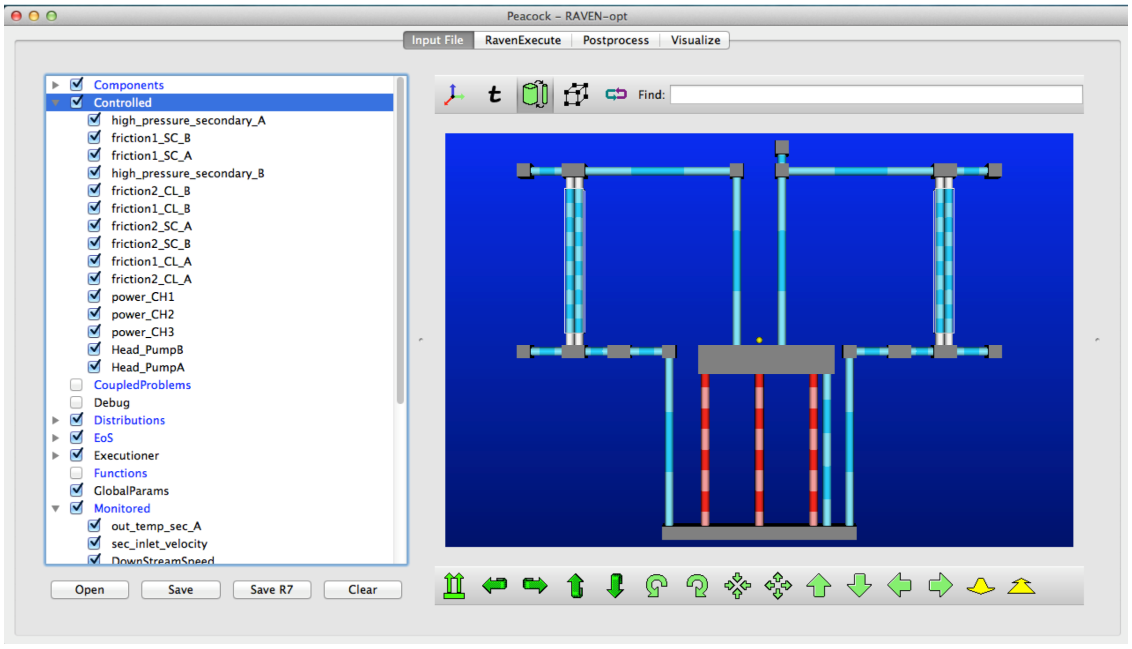

RAVEN
Change the seed in the random number generator
RELAP-7 RAVEN Input File
Supply the new input and spoon a RAVEN simulation
RELAP-7 RAVEN run
Simulation 1
Change the seed in the random number generator
RELAP-7 RAVEN Input File
Supply the new input and spoon a RAVEN simulation
RELAP-7 RAVEN run
Simulation 2
...
Change the seed in the random number generator
RELAP-7 RAVEN Input File
Supply the new input and spoon a RAVEN simulation
RELAP-7 RAVEN run
Simulation *n*

Figure 4: Monte-Carlo sampling scheme.

generator for the distributions. The multiple calculations, required by the employment of these algorithms, can be run in parallel, using queues/sub-process/`Python` systems.

The analysis of dynamic stochastic systems through Monte-Carlo algorithm can be summarized (Figure 4) as follows:

1. Initial Sampling of:
   (a) Static and dynamic uncertainty values of physical parameters
   (b) Initial conditions
   (c) Transition conditions, i.e. time instant in which transition events occur (e.g., time in which a reactor scram occurs, time delta to recover power grid)
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)



- ✓ Components
- ✓ Controlled
  - ✓ high_pressure_secondary_A
  - ✓ friction1_SC_B
  - ✓ friction1_SC_A
  - ✓ high_pressure_secondary_B
  - ✓ friction2_CL_B
  - ✓ friction1_CL_B
  - ✓ friction2_SC_A
  - ✓ friction2_SC_B
  - ✓ friction1_CL_A
  - ✓ friction2_CL_A
  - ✓ power_CH1
  - ✓ power_CH2
  - ✓ power_CH3
  - ✓ Head_PumpB
  - ✓ Head_PumpA
- ☐ CoupledProblems
- ☐ Debug
- ✓ Distributions
- ✓ EoS
- ✓ Executioner
- ☐ Functions
- ✓ GlobalParams
- ✓ Monitored
  - ✓ out_temp_sec_A
  - ✓ sec_inlet_velocity
  - ✓ DownStreamSpeed

Figure 5: Input/plan Visualization GUI Window.

Figure 4 shows 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 Graphical User Interface (GUI) 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 (`C++`) 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.

5. SOFTWARE LAYOUT AND CALCULATION FLOW

Figures 2 and 3 show the calculation flow employed by RAVEN/RELAP-7/MOOSE software. A typical RAVEN calculation can be summarized in the following logic steps:

1. Perform Initialization
2. RELAP-7/MOOSE updates the information contained in each component class with the actual solution $\bar{x}$
3. RAVEN requests MOOSE to perform the post-processing manipulation in order to construct $\bar{C}$

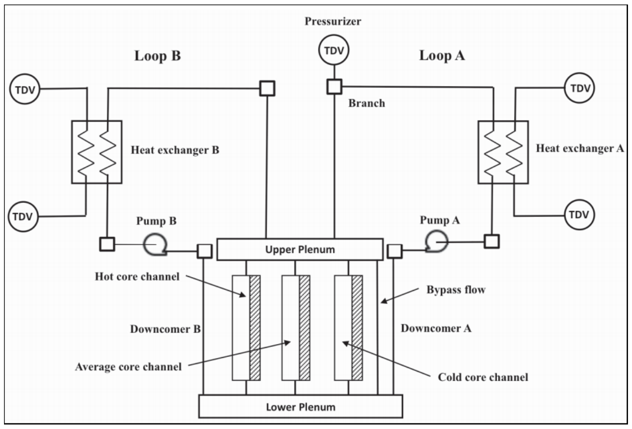


Figure 6: PWR model scheme.

4. 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 determined
5. RAVEN asks RELAP-7/MOOSE to compute the solution $\bar{x}$ for the following time step
6. Repeat from 2 to 5 until the end of the calculation or an exit condition is reached (e.g., clad failure)

6. DEMO FOR A PWR PRA ANALYSIS

In order to show the capabilities of RAVEN coupled with RELAP-7/MOOSE, a simplified PWR PRA analysis has been employed. Figure 6 shows the scheme of the PWR model. The reactor vessel model consists of the Down-comers, the Lower Plenum, the Reactor Core Model and the Upper Plenum. Core channels (flow channels with heat structure attached to each of them) were used to describe the reactor core. The core model consists of three parallel core channels and one bypass flow channel. There are two primary loops, i.e., loop A and loop B. Each loop consists of the Hot Leg, a Heat Exchanger and its secondary side pipes, the Cold Leg and a primary Pump. A Pressurizer is attached to the Loop A piping system to control the system pressure. A Time Dependent Volume (pressure boundary conditions) component is used to represent the Pressurizer. Since the RELAP-7 code does not have the two-phase flow capability yet, single-phase counter-current heat exchanger models are implemented to mimic the function of steam generators in order to transfer heat from the primary to the secondary. In order to perform a PRA analysis on this simplified model, it has been necessary to control unconventional parameters (i.e. inlet/outlet friction factors), since RELAP-7 still has limitations for the component controllable parameters and models. In the following paragraph, the PRA station black out sequence of events is reported.

6.1. Station Black Out (SBO) analysis

The Probabilist Risk Assessment analysis has been performed simulating a Station Black Out accident, making Monte-Carlo samplings on the recovery time of the diesel generators t_1 (delta time from reactor scram signal) and the clad failure temperature TCf. Two sets of Monte-Carlo calculations have been run:

- 400 runs, randomizing t_1 (Normal distribution, mu = 120 s, sigma = 20 s) and TCf (Triangular distribution, xPeak = 1477.59 K, xMin = 1255.37 K, xMax = 1699.82 K)

- 400 runs, randomizing only t_1

The SBO transient is based on the following sequence of events (starting from a steady-state operational condition of the Nuclear Power Plant [5]):

- 60.0 seconds, transient begins
- 61.0 seconds, loss of power grid and immediate shutdown of the reactor(scram):
  - Pump coast-down;
  - Decay heat power;
  - Diesel Generators and residual heat removal system (RHRS) not available.
- t_1, recovery of the diesel generators
- t_2, end of transient either for clad failure or 300 seconds of simulation (PRA success)

Since the scope of this demo is to show the capabilities contained in RAVEN and RELAP-7 capabilities are not optimized for long simulation times, the transient has been accelerated in order to simulate a maximum of 300 seconds. In the following paragraph, the simulations results are shown and explained.

6.2. Results

Figure 7 shows the distribution of the maximum temperature reached by the clad in the core channels (blue histogram) and compares it with the distribution of clad failure temperature (red histogram). Although there is large overlapping of the two distributions, which indicates a high failure probability of the system considered, the scope of the analysis was just to show RAVEN capabilities to perform stochastic analysis of relatively complex systems.

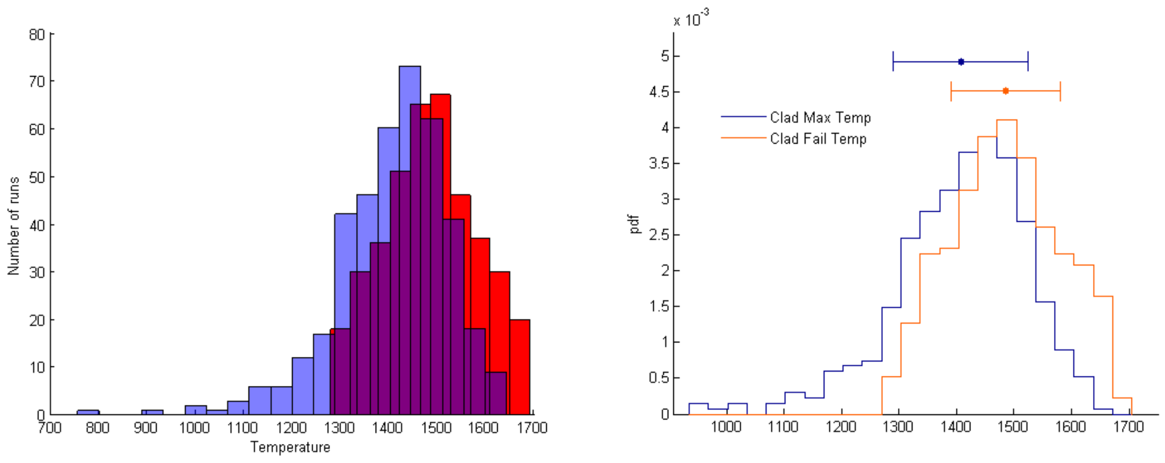


Figure 7: Comparison between max reached clad temperature and clad failure temperature distributions: binning results (left) and associated probability distribution functions (right)

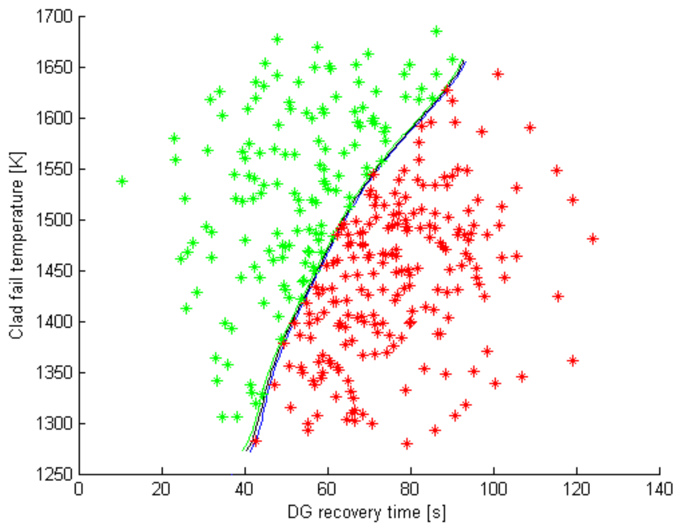


Figure 8: Limit Surface for the SBO analysis of a simplified PWR model

In addition, Fig. 8 shows the limit surface, i.e. the boundaries between system failure (red points) and system success (green points), obtained by the 400 Monte-Carlo simulations. Since only two uncertain parameters have been considered (i.e., DG recovery time and clad fail temperature), this boundary lies in a 2-dimensional space. The slope of the limit surface pictured in Fig. 8 also shows, in this particular demo, how the DG recovery time has a greater impact on the system dynamics then the clad failure temperature.

It has also been performed a new set of 400 Monte-Carlo simulations in which, now, the clad failure temperature is fixed at a predefined value $T_{Fail} = 1477.59$ (i.e., there is no triangular distribution associated to it). As expected, the probability of core damage was slightly different among thee two set of simulations. This fact shows how modeling of uncertainties can impact risk evaluation.

7. CONCLUSIONS

In this paper it has been presented RAVEN as a tool to perform dynamic PRA through Monte-Carlo sampling. In particular, the software structure and all the components that are involved in the computation have been presented, including system simulator (i.e., RELAP-7) and the control logic, characterized by monitor system dynamics and on-line control of selected parameters. An example of PRA analysis has been also presented for a SBO-like case for a simplified PWR loop. The description of the implementation for such case demonstrates how the flexibility of the software framework provides the basic tools to perform Dynamic PRA, uncertainty quantification and plant control. Next capabilities, to be implemented to RAVEN and that are currently under development, include dynamic event tree generation [6], adaptive sampling [7] and more advanced data mining algorithms [8].

REFERENCES

[1] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, and R. Kinoshita, “Reactor analysis and virtual control environment (raven) fy12 report,” Tech. Rep. INL/EXT-12-27351, Idaho National Laboratory (INL), 2012.

[2] C. Rabiti, A. Alfonsi, D. Mandelli, J. Cogliati, and R. Martineau, "Raven as control logic and probabilistic risk assessment driver for relap-7," in *Proceeding of American Nuclear Society (ANS), San Diego (CA)*, vol. 107, pp. 333–335, 2012.

[3] D. Gaston, G. Hansen, S. Kadioglu, D. A. Knoll, C. Newman, H. Park, C. Permann, and W. Taitano, "Parallel multiphysics algorithms and software for computational nuclear engineering," *Journal of Physics: Conference Series*, vol. 180, no. 1, p. 012012, 2009.

[4] D. Mandelli and C. Smith, "Integrating safety assessment methods using the risk informed safety margins characterization (rismc) approach," in *Proceeding of American Nuclear Society (ANS), San Diego (CA)*, vol. 107, pp. 883–885, 2012.

[5] D. Anders, R. Berry, D. Gaston, R. Martineau, J. Peterson, H. Zhang, H. Zhao, and L. Zou, "Relap-7 level 2 milestone report: Demonstration of a steady state single phase pwr simulation with relap-7," Tech. Rep. INL/EXT-12-25924, Idaho National Laboratory (INL), 2012.

[6] A. Hakobyan, T. Aldemir, R. Denning, S. Dunagan, D. Kunsman, B. Rutt, and U. Catalyurek, "Dynamic generation of accident progression event trees," *Nuclear Engineering and Design*, vol. 238, no. 12, pp. 3457 – 3467, 2008.

[7] D. Mandelli and C. Smith, "Adaptive sampling using support vector machines," in *Proceeding of American Nuclear Society (ANS), San Diego (CA)*, vol. 107, pp. 736–738, 2012.

[8] D. Mandelli, A. Yilmaz, and T. Aldemir, "Scenario analysis and pra: Overview and lessons learned," in *Proceedings of European Safety and Reliability Conference (ESREL 2011), Troyes (France)*, 2011.