MANUAL

FARM SDD Number (to be generated)
Revision 0
Printed February 16, 2022

FARM Software Design Description

Haoyu Wang (ANL), Roberto Ponciroli (ANL), Andrea Alfonsi

Prepared by
Idaho National Laboratory
Idaho Falls, Idaho 83415

The Idaho National Laboratory is a multiprogram laboratory operated by
Battelle Energy Alliance for the United States Department of Energy
under DOE Idaho Operations Office. Contract DE-AC07-05ID14517.

Issued by the Idaho National Laboratory, operated for the United States Department of Energy by Battelle Energy Alliance.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

FARM SDD Number (to be generated)
Revision 0
Printed February 16, 2022

FARM Software Design Description

Haoyu Wang
Argonne National Laboratory
haoyuwang@anl.gov

Roberto Ponciroli
Argonne National Laboratory
rponciroli@anl.gov

Andrea Alfonsi
Idaho National Laboratory
andrea.alfonsi@inl.gov

Contents

1 Introduction

1.1 System Purpose

The ***FARM*** plug-in is a generalized module for supervisory control within RAVEN.
RAVEN is a flexible and multi-purpose uncertainty quantification (UQ), regression analysis, probabilistic risk assessment (PRA), data analysis and model optimization software. Depending on the tasks to be accomplished and on the probabilistic characterization of the problem, RAVEN perturbs (Monte-Carlo, Latin hyper-cube, reliability surface search, etc.) the response of the system under consideration by altering its own parameters. The data generated by the sampling process is analyzed using classical statistical and more advanced data mining approaches. RAVEN also manages the parallel dispatching (i.e. both on desktop/workstation and large High-Performance Computing machines) of the software representing the physical model. For more information about the RAVEN software, see [1] and the RAVEN website (`https://raven.inl.gov`)
The ***FARM*** module is a dedicated plug-in module of RAVEN. A RAVEN plug-in is a software/-module/library that has been developed to be linked to RAVEN at run-time, using the RAVEN APIs.

1.2 System Scope

The ***FARM*** plug-in's scope is to provide the capability to compute the optimal value and the admissible range of actuator variables, given the state-space representation of system dynamics, to avoid any violation of system operational constraints.

The main objective of the module (in conjunction with the RAVEN software) is to assist the engineer/user to:

- predict the future evolution of system state and output variables when imposing a given value of actuator;
- adjust the actuator value in order to keep the critical variables within the range defined by operational constraints;
- provide the actuator admissible range to aid further analysis, such as power dispatch optimization.

In other words, the ***FARM*** plug-in (driven by RAVEN) is aimed to be employed for:

- Dynamic system behavior prediction;
- System supervisory control.

1.3 User Characteristics

The users of the ***FARM*** plug-in are expected to be part of any of the following categories:

- **Core developers (FARM core team)**: These are the developers of the ***FARM*** plug-in. They will be responsible for following and enforcing the appropriate software development standards. They will be responsible for designing, implementing and maintaining the plug-in.
- **External developers**: A Scientist or Engineer that utilizes the ***FARM*** plug-in and wants to extend its capabilities. This user will typically have a background in modeling and simulation techniques and/or numerical and data analysis but may only have a limited skill-set when it comes to object-oriented coding, C++/Python languages.
- **Analysts**: These are users that will run the plug-in (in conjunction with RAVEN) and perform various analysis on the simulations they perform. These users may interact with developers of the system requesting new features and reporting bugs found and will typically make heavy use of the input file format.

1.4 Other Design Documentation

In addition to this document, an automatic software documentation is generated every time a new CR (see def.) is approved. This documentation is automatically extracted from the source code using doxygen (see def.) and is available to developers at `https://hpcsc.inl.gov/ssl/RAVEN/docs/classes.html` (together with all the other RAVEN plug-ins).

In order to generate(locally) a hard copy in "html" or "latex", any user/developer can launch the following command (in the raven directory, with the FARM submodule configured and activated):

```
doxygen ./doc/doxygen/Doxyfile
```

Once the documentation is generated, any user/developer can navigate to the folder

```
./doc/doxygen/latex
```

and type the following command:

```
make refman.pdf
```

Once the command is executed, a "pdf" file named "refman.pdf" will be available.

The doxygen software is under configuration management process identified in " RAVEN Configuration Management " PLN-5553.

1.5 Dependencies and Limitations

The plug-in should be designed with the fewest possible constraints. Ideally the plug-in (in conjunction with RAVEN) should run on a wide variety of evolving hardware, so it should follow well-adopted standards and guidelines. The software should run on any POSIX compliant system (including Windows POSIX emulators such as MinGW).
In order to be functional, ***FARM*** depends on the following software/libraries.

- RAVEN (`https://raven.inl.gov`) and all its dependencies (listed in "RAVEN Software Design Description" - SDD-513)

2 References

- ASME NQA 1 2008 with the NQA-1a-2009 addenda, "Quality Assurance Requirements for Nuclear Facility Applications," First Edition, August 31, 2009.
- ISO/IEC/IEEE 24765:2010(E), "Systems and software engineering Vocabulary," First Edition, December 15, 2010.
- LWP 13620, "Managing Information Technology Assets"
- SDD-513, " RAVEN Software Design Description "
- PLN-5552, " RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan "

3 Definitions and Acronyms

3.1 Definitions

- **Baseline.** A specification or product (e.g., project plan, maintenance and operations [M&O] plan, requirements, or design) that has been formally reviewed and agreed upon, that thereafter serves as the basis for use and further development, and that can be changed only by using an approved change control process. [ASME NQA-1-2008 with the NQA-1a-2009 addenda edited]
- **Validation.** Confirmation, through the provision of objective evidence (e.g., acceptance test), that the requirements for a specific intended use or application have been fulfilled. [ISO/IEC/IEEE 24765:2010(E) edited]
- **Verification.**
  - The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
  - Formal proof of program correctness (e.g., requirements, design, implementation reviews, system tests). [ISO/IEC/IEEE 24765:2010(E) edited]

3.2 Acronyms

ANL Argonne National Laboratory

API Application Programming Interfaces

ASME American Society of Mechanical Engineers

CG Command Governor

DOE Department of Energy

HDF5 Hierarchical Data Format (5)

LWRS Light Water Reactor Sustainability

NEAMS Nuclear Energy Advanced Modeling and Simulation

NHES Nuclear-Renewable Hybrid Energy Systems

IES Integrated Energy Systems

INL Idaho National Laboratory

IT Information Technology

MOAS Maximal Output Admissible Set

NQA Nuclear Quality Assurance

O&M Operation and Maintenance

POSIX Portable Operating System Interface

PP Post-Processor

QA Quality Assurance

RAVEN Risk Analysis and Virtual ENviroment

RG Reference Governor

ROM Reduced Order Model

SDD System Design Description

XML eXtensible Markup Language

4 Design Stakeholders and Concerns

4.1 Design Stakeholders

- Integrated Energy Systems (IES)
- Open-source community

4.2 Stakeholder Design Concerns

The design of the ***FARM*** plug-in in terms of functionality and capabilities to be deployed has been performed in accordance with the funding programs reported above. No specific concerns have been raised during the design and deployment of the ***FARM*** software.

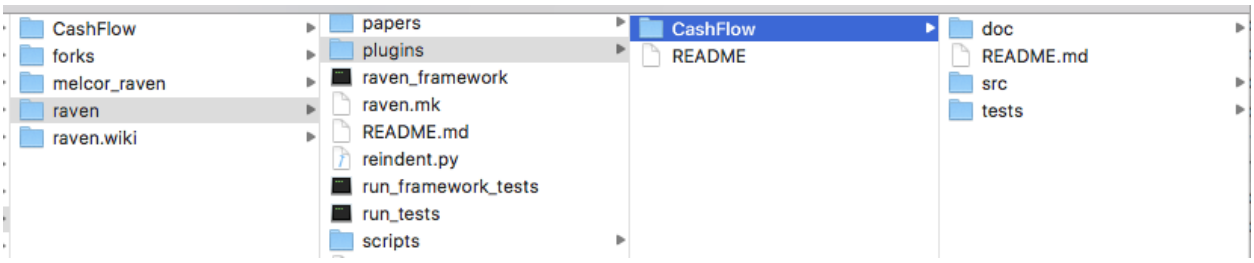

Figure 1: Plugins Location

5 Software Design

5.1 Introduction

The ***FARM*** plug-in scope is to provide the capability to compute the optimal value and the admissible range of actuator variables, given the state-space representation of system dynamics, to avoid any violation of system operational constraints. The ***FARM*** plug-in has been coded using the language *Python*. The ***FARM*** plugin can only be used within RAVEN (for installation and usage instructions, see [1]). The input of ***FARM*** is an XML file, which can be read and executed by RAVEN.

5.2 System Structure (Code)

The ***FARM*** plug-in is based on the RAVEN plug-in system. This system is aimed to ease the creation of external models and modules by external developers without the need to deeply know the internal structure of the RAVEN software. This system is a transparent API for RAVEN external models.
The addition of a plugin does not require modifying RAVEN itself. Instead, the developer creates a new Python module that is going to be embedded in RAVEN at run-time (no need to introduce hard-coded statements). This plugin (FARM for instance) needs to be placed in a folder located in (see figure 1):

```
path/to/raven/plugins/
```

In order to install the FARM plugin (if not downloaded by RAVEN submodule system), the user can run the script contained in the RAVEN script folder:

```
python path/to/raven/scripts/install_plugins.py  **directory**/FARM
```

where $**directory**$/FARM should be replaced with the absolute path to the FARM plugin directory. (e.g. “path/to/my/plugins/FARM”).

5.3 FARM Structure

The ***FARM*** plug-in contains the following methods (API from RAVEN):

```
from PluginBaseClasses.ExternalModelPluginBase
  import ExternalModelPluginBase

class RefGov_parameterized_SIMO(ExternalModelPluginBase):
  def run(self, container, Inputs)
  def _readMoreXML(self, container, xmlNode)
  def initialize(self, container, runInfoDict, inputFiles)
  def createNewInput(self, container, inputs, samplerType, **Kwargs)
```

In the following sub-sections all the methods are explained.

5.3.1 Method: `run`

```
  run(self, container, Inputs)
```

In this function, the reference governor, kernel of FARM, is coded. The only two attributes this method is going to receive are a Python list of inputs (the inputs coming from the XML block **<Input>** and a "self-like" object named "container". All the outcomes of the FARM module will be stored in the above mentioned "container" in order to allow RAVEN to collect them.

5.3.2 Method: `_readMoreXML`

```
  def _readMoreXML(self, container, xmlNode)
```

In this method, the FARM input is read and made available to the plug-in and RAVEN. The read information are stored in the "self-like" object "container" in order to be available to all the other methods, specifically the **run**, **initialize** and **createNewInput** methods. The method receives from RAVEN an attribute of type "xml.etree.ElementTree", containing all the sub-nodes and attribute of the XML block **<ExternalModel>**.

Example XML:

```
  <ExternalModel name="RG1"
     subType="FARM.RefGov_parameterized_SIMO">
  <!-- 3 output variables -->
  <outputVariables>V, V_min, V_max </outputVariables>
```

```
  <!-- 4 variables: Issued Setpoint(PwrSet), Adjusted
     Setpoint(V1), bounds of V1(V1min & V1max) -->
  <variables> PwrSet, V, V_min, V_max </variables>
  <!-- steps in MOAS calculation, "g" value -->
  <constant varName="MOASsteps"> 360 </constant>
  <!-- lower and upper bounds for y vector, will be internally
     checked -->
  <constant varName="Min_Target1"> 2.5 </constant>
  <constant varName="Max_Target1"> 55. </constant>
  <constant varName="Min_Target2"> 2.5 </constant>
  <constant varName="Max_Target2"> 55. </constant>
  <!-- System state vector "x", optional, with elements
     separated by comma(,) -->
  <constant varName="Sys_State_x"> 30.,0 </constant>
</ExternalModel>
```

5.3.3 Method: `initialize`

```
def initialize(self, container, runInfo, inputs)
```

The **initialize** method is implemented to initialize the FARM simulation based on the current RAVEN status and the XML input file.
Indeed, RAVEN is going to call this method at the initialization stage of each "Step" (see section [1]). RAVEN will communicate, thorough a set of method attributes, all the information that are needed to perform an initialization:

- runInfo, a dictionary containing information regarding how the calculation is set up (e.g. number of processors, etc.). It contains the following attributes:
  - `DefaultInputFile` – default input file to use
  - `SimulationFiles` – the xml input file
  - `ScriptDir` – the location of the pbs script interfaces
  - `FrameworkDir` – the directory where the framework is located
  - `WorkingDir` – the directory where the framework should be running
  - `TempWorkingDir` – the temporary directory where a simulation step is run
  - `NumMPI` – the number of mpi process by run
  - `NumThreads` – number of threads by run
  - `numProcByRun` – total number of core used by one run (number of threads by number of mpi)

- `batchSize` – number of contemporaneous runs
- `ParallelCommand` – the command that should be used to submit jobs in parallel (mpi)
- `numNode` – number of nodes
- `procByNode` – number of processors by node
- `totalNumCoresUsed` – total number of cores used by driver
- `queueingSoftware` – queueing software name
- `stepName` – the name of the step currently running
- `precommand` – added to the front of the command that is run
- `postcommand` – added after the command that is run
- `delSucLogFiles` – if a simulation (code run) has not failed, delete the relative log file (if True)
- `deleteOutExtension` – if a simulation (code run) has not failed, delete the relative output files with the listed extension (comma separated list, for example: 'e,r,txt')
- `mode` – running mode, curently the only mode supported is mpi (but custom modes can be created)
- *expectedTime* – how long the complete input is expected to run
- *logfileBuffer* – logfile buffer size in bytes

- inputs, a list of all the inputs that have been specified in the "Step" using this model.

5.3.4 Method: `createNewInput`

```
def createNewInput(self, container, inputs, samplerType, **Kwargs)
```

In this method, the state-space matrices file will be read and analyzed, with the following information made available in the "container" for the **run** method:

- `Tss` – time interval for the A,B,C matrices in discrete time domain
- `n, m, & p` – the dimensions of state variables, actuator variables and output variables
- `para_array` – the array of scheduling parameters (only available in parameterized FARM)
- `UNorm_list` – the list of actuator variable nominal values corresponding to each scheduling parameter
- `XNorm_list` – the list of state variable nominal values corresponding to each scheduling parameter

- `XLast_list` – the list of state variable last values corresponding to each scheduling parameter
- `YNorm_list` – the list of output variable nominal values corresponding to each scheduling parameter
- `A_list` – the list of state-space A matrices (state matrices) corresponding to each scheduling parameter
- `B_list` – the list of state-space B matrices (input matrices) corresponding to each scheduling parameter
- `C_list` – the list of state-space C matrices (output matrices) corresponding to each scheduling parameter

5.4 Data Design and Control

The data transfer in the ***FARM*** plug-in is fully standardized via the API for External Model Plug-In, via Python dictionaries.
The documentation of this API is reported in the RAVEN user manual ([1])

5.5 Human-Machine Interface Design

There are no human system integration requirements associated with this software.

5.6 Security Structure

The software is accessible to the open-source community (Apache License, Version 2.0). No restrictions for downloading or redistributing is applicable.

6 REQUIREMENTS CROSS-REFERENCE

The requirements are detailed in FARM SPC & RTM Number (to be generated), “FARM Software Requirements Specification (SRS) and Traceability Matrix”.

References

[1] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, P. W. Talbot, D. P. Maljovec, and J. Chen, "Raven user manual," tech. rep., Idaho National Laboratory, 2017.

Document Version Information


INL
Idaho National Laboratory