PhysicsShell: A Solution to Multi-Code Linking

Jakub Szymandera[a,*], Larry Blake[a], Bac Gia Phan[b], Ovidiu Nainer[c]

[a] *Nuclear Safety Solutions, Ltd. Toronto, Canada*
[b] *Ontario Power Generation, Pickering, Canada*
[c] *Bruce Power Generation, Toronto, Canada*

*** Corresponding author**
jakub.szymandera@amec.com

Abstract

Proper safety analysis of nuclear reactors often requires the use of several computer codes to model a specific scenario. To complicate matters, the computer codes are often required to exchange information between each other. A controlled and generic method is required to govern such multi-code interactions. The solution proposed is the PhysicsShell program.

1.0 Introduction

Nuclear reactors comprise of many intricate systems operating as a cohesive unit. As a result, the continued assurance that nuclear reactors are being operated in a safe manner involves complex analyses that may involve multiple systems. In general, the modelling of physically different systems for safety analysis involves separate computer codes. For example, in analysis of CANDU reactors, the heat transport system is modelled using a dedicated thermal hydraulics codes, core neutronics are handled by physics computer codes, and detailed fuel and fuel channel behaviour is yet still modelled with a further set of codes.

Safety analysis of design basis accidents, such as for example analysis of a loss of coolant accident (LOCA), may involve three or more sets of disciplines. Further, the various codes will require as input the results of one or more of the other codes. The stage at which input is required from the other codes may necessitate a coupled execution, or a sequential execution.

It is apparent that an organized scheme is required to properly manage the execution of various codes during safety analyses.

Code coupling is an ongoing activity and examples in the CANDU context are SMOKIN_SHELL and rfspcb. The former linked the SMOKIN physics code with the thermal hydraulic code TUF and the latter the RFSP and CATHENA codes.

2.0 Method of Solution

PhysicsShell is a software tool that permits a common and relatively simple interface to be created between codes that are normally used in safety analysis. It has been developed using the PERL programming language. PERL was originally designed for text processing, it has grown into a sophisticated, general-purpose programming language with a rich software development environment complete with debuggers, profilers, cross-references, compilers, interpreters, libraries, syntax-directed editors, and all the rest of the trappings of a "real" programming language.

In safety analyses, many of the codes use the products of one, or another, of the codes as part of their input, a mechanism is needed to ensure that interconnections are made. As well, there may be other constraints or activities that need to be imposed upon the process by which the analysis is performed. For example, in some instance, not only is the result subject to an iterative approach to solution, but some of the activities are contingent upon the previous results.

While it is possible to create dedicated computer programs or ad hoc shell scripts to perform these tasks, a more general approach allows for consistency of application. As well, certain features can be included within the method that can serve to simplify the process.

The approach taken by PhysicsShell is to create a small scripting language that allows for the execution of any programs that can be executed by the user, as well as to allow for the creation of ad hoc utilities that can share a common infrastructure. Additionally, conditional and repetitive operations are defined. Finally, the use of variables and internal functions act to simplify the execution of several code streams.

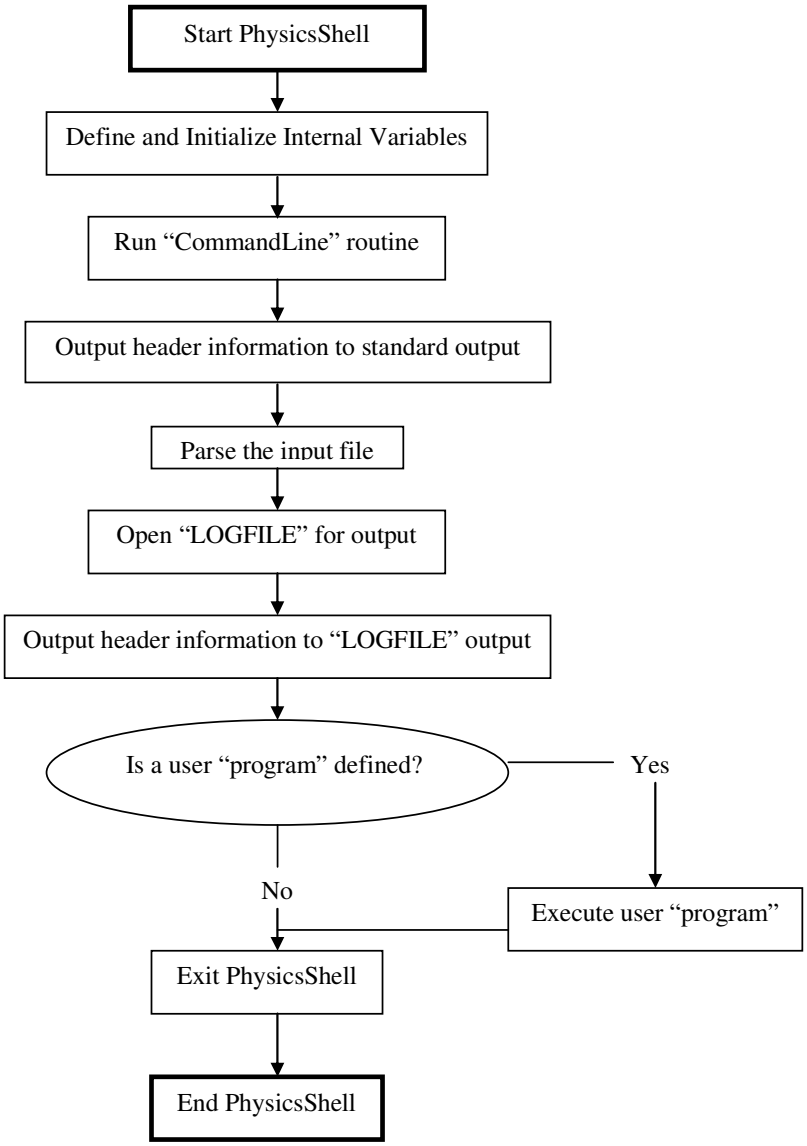
The performance of PhysicsShell is limited by, and governed by, the routines that are specified as input. In general the overhead attributable to the shell is minimal.

3.0 PhysicsShell Overview

A run consists of parsing of the input file and command line, and then execution of a user specified process. These user instructions, called programs, are input in either the user input file or a file read in using the **ReadPrograms** routine. As this process is user driven the actual execution will differ from run to run. Upon completion, PhysicsShell creates output and log files.

The overall process flow of the PhysicsShell script is shown in Figure 1.

Figure 1: Overall Program Flow



The PhysicsShell script contains numerous routines available to the user. The various types of routines are listed in Table 1.

Table 1: Routine Types

Routine Type	Description
Internal	Used by the PhysicsShell to perform its functions.
Syntax and template parsing	Used to perform the fundamental task of the PhysicsShell, to generate input based on a template and current variables.
Comparison	Performs some basic comparisons of variables.
Shell or system emulation	Emulates some system routines and reports status to the logfile.
External routine processing	Invokes the major analysis codes (e.g. RFSP).
Link processing	Routines used to simplify the link between two codes, such as the TUF/CERBERUS interface.
Displaying information	Outputs to either STDOUT or the logfile information regarding the state of variables or the current simulation.
Importing data	Allows for the importation of programs and routines into the PhysicsShell.
Variable manipulation	Modifies or allows modification of user variables.
List manipulation	Performs some basic manipulation of lists.
Array manipulation	Basic Array routines. Currently this includes importation of an array from a text file.
Specific information processing	Parses specific output files for information that is important in the link between two codes.

Apart from the internal routines, the above routines are available to the user to be invoked in a program statement, and their inclusion in the PhysicsShell script is to allow for a common approach to code interface.

This functionality can be increased by the INCLUDE routine. This routine reads a PERL file and includes it in the current environment. Thus, any routines that are defined in the included file can now be accessed as if they were defined in the PhysicsShell script itself. This functionality allows PhysicsShell to be extremely versatile and therefore be successfully applied to a multitude of problems, and safety analysis code interfaces.

3.1 Inputs

The main input file provides the primary user interface to the program. It allows the user to:

- specify initial values of variables and lists,
- specify certain variables as being updateable based on simple rules, and
- specify the code to execute for the given run.

The code that is to be executed is found in ‘programs’. These are the active components of the input stream. The programs specify what the user intends to do with the data provided as input.

The program may perform one of the following tasks:

- create a program branch,
- call a routine,
- call a UNIX system call, and
- call an executable file in the users PATH.

In addition, the program statements may be combined with ‘while’ and ‘if’ loops. This functionality affords the user great flexibility in the control of the overall process.

PhysicsShell contains some general purpose routines for reading text files. Custom made routines to address specific input requirements may be created in the PERL language and made accessible to PhysicsShell through the use of the INCLUDE statement. In this manner, the treatment of input files is fully configurable by the user.

Another important input to PhysicsShell is the template file. Templates are files that have a special syntax that allows them to be altered by PhysicsShell. The altered file can then be used as an input to another routine or code. Through careful construction, one template can serve as a basis for many different input files for a given code. This is achieved by including ‘if’ statements in the template. For example, the control of RFSP [1] execution during a coupled RFSP/TUF [2] LOCA simulation requires the RFSP input file to include the call to the RFSP *TRIP_TIME module if a specific condition is reached. Instead of having two templates, one with and one without the *TRIP_TIME module, a single template can be used and the *TRIP_TIME module encompassed by a conditional ‘if’ statement. Templates are parsed, read and modified, by an internal routine.

3.2 Output

The output of PhysicsShell is normally to standard output; except as specified by an internal routine. This is not to be confused with the output generated by the various codes (e.g. RFSP, TUF) that are called by PhysicsShell. The output of these codes will be located in directories as defined by the user in the PhysicsShell input file.

During the course of an analysis, it is often required to manipulate the output generated by the codes. In PhysicsShell, this is achieved by calling specific sub-routines that are designed for the purpose of manipulating data and are internal to PhysicsShell or have been developed by the user. Two of the internal PhysicsShell sub-routines are the 'GrabPhysicsData' and the 'DrawGraph' sub-routines. An example below includes lines of code necessary to make use of these sub-routines when the RFSP code has been executed. Comments are provided within the bolded **{}.**

```
_PROG_NAME => RFSP {Execute RFSP.}
{Obtain the data to be graphed}
{Call the 'GrabPhysicsData' sub-routine. This reads through an rfsp_output file and grabs information from a CERBERUS run. It assumes one run per file. Information is stored in PhysicsShell lists_cerberus_time and _physics_*, where '*' stands for 'amp', 'rho', 'beta', 'lstar', and 'prompt'.}
_PROG_NAME => GrabPhysicsData
{Draw a graph of the _physics_amp list as a function of _cerberus_time list in a file called amplitude-1.eps }
_PROG_NAME => DrawGraph _cerberus_time _physics_amp amplitude-1.eps
{ Draw a graph of the _physics_rho list as a function of _cerberus_time list in a file called amplitude-1.eps }
_PROG_NAME => DrawGraph _cerberus_time _physics_rho reactivity-1.eps
```

This concept may be applied to any data that has been previously stored in a list.

A LOGFILE is created for each run of PhysicsShell. This file is found in the directory from which PhysicsShell was executed and has the default name PhysicsShell.log.

The contents of this file includes:

- The input file and command line options.
- The actual routines executed during the current run.
- The results of many internal programs are echoed to the LOGFILE.
- The final completion status.

Some error messaging is enabled and available in PhysicsShell where the error message will follow the program statement.

4.0 Application of PhysicsShell

The two sub-sections to follow document specific types of applications in which PhysicsShell performs the central role of managing a multi-code environment.

4.1 Best Estimate Analysis with Uncertainty (BEAU)

The proper execution of BEAU for large break LOCA requires on the order of several hundred large break loss of coolant accident (LBLOCA) simulations. Each LBLOCA is simulated using coupled physics and thermal hydraulic codes. Examples of computer codes used in this application are the physics RFSP and thermal hydraulic TUF codes. The PhysicsShell tool has been successfully employed in the automation of this complex process.

In BEAU, simulations of LBLOCA are performed for a case matrix, which defines values of specific physics and thermal hydraulic input parameters. In this application, PhysicsShell is used to populate the RFSP and TUF input files with the correct parameters and control the RFSP/TUF linking. The various parameters and values are incorporated using the INCLUDE function in both the input file or the template, where appropriate. Therefore, from a practical point of view, a case name variable may be defined and PhysicsShell will include the appropriate parameters in the input and templates based on the current value of the case name variable.

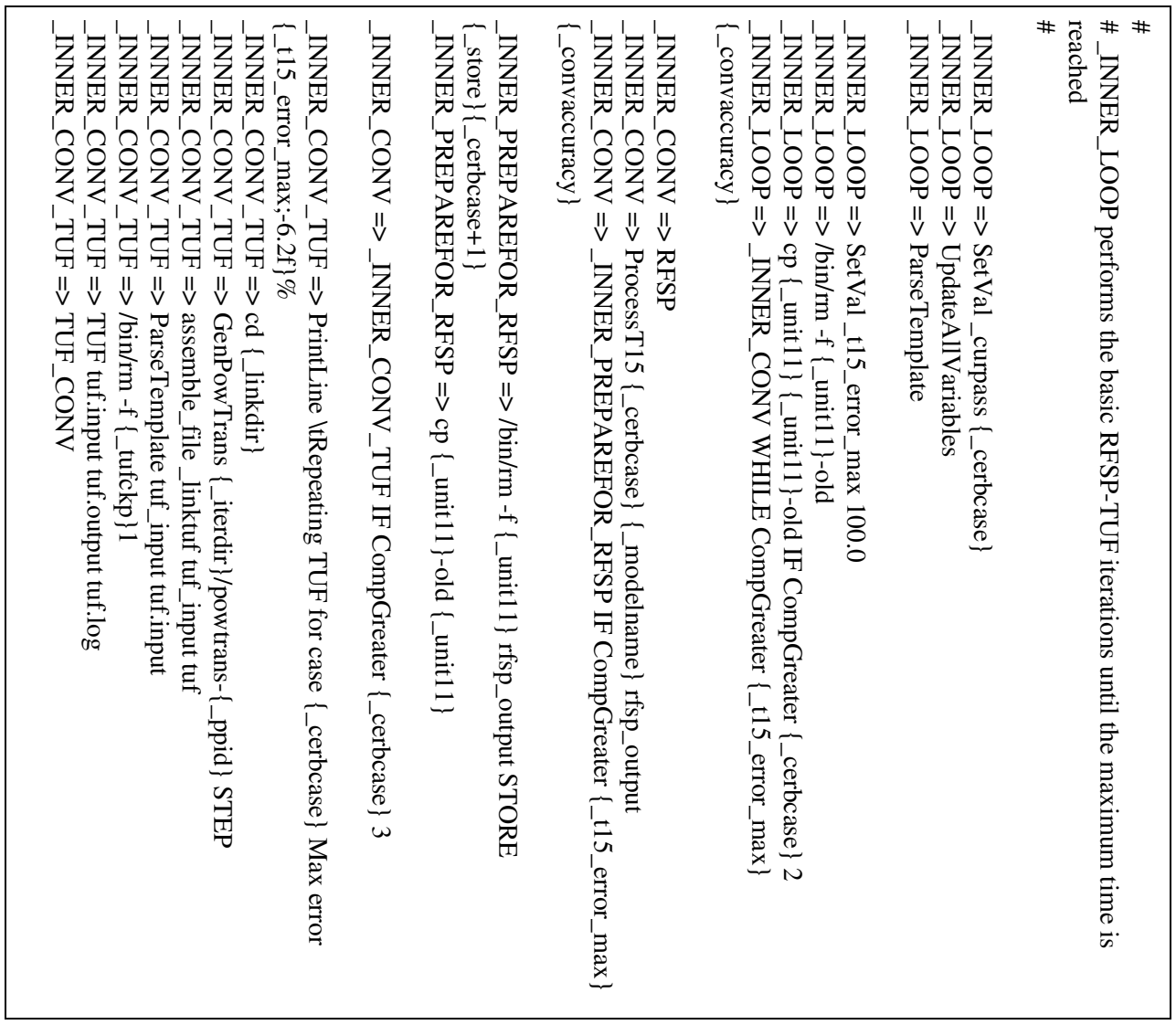
The RFSP/TUF link, in essence, is the manipulation and transfer of RFSP generated regional powers to TUF and TUF generated fuel temperature, coolant density, and coolant temperature information to RFSP. A LBLOCA transient is simulated in a step-by-step fashion. The total simulation time is divided into small segments with both RFSP and TUF executed at each step. PhysicsShell is used to control such functions as ensuring proper convergence between RFSP and TUF has been attained at each step of the simulation, determining whether the reactor has tripped based on a parsing of RFSP output, and insertion of shut-off rods at the correct instance.

In addition, PhysicsShell is used to manage the execution of the fuel and fuel channel codes FACTAR-SS [3] and FACTAR [4]. The former code establishes the steady state conditions of the fuel and the later is used for transient analysis. Through the inclusion of PERL subroutines, PhysicsShell performs the necessary data manipulation, creation of specific input files, and execution of the two fuel and fuel channel codes.

PhysicsShell plays a central role in managing information flow and performing necessary actions for the successful coupling of RFSP, TUF, FACTAR-SS, and FACTAR in BEAU.

Figure 2 illustrates a portion of the syntax of the PhysicsShell program that facilitates the TUF and RFSP iterations.

Figure 2: Portion of PhysicsShell Program for TUF and RFSP Link



```
#
# _INNER_LOOP performs the basic RFSP-TUF iterations until the maximum time is
reached
#
_INNER_LOOP => SetVal _curpass {_cerbcase}
_INNER_LOOP => UpdateAllVariables
_INNER_LOOP => ParseTemplate

_INNER_LOOP => SetVal _t15_error_max 100.0
_INNER_LOOP => /bin/rm -f {_unit11}-old
_INNER_LOOP => cp {_unit11} {_unit11}-old IF CompGreater {_cerbcase} 2
_INNER_LOOP => _INNER_CONV WHILE CompGreater {_t15_error_max}
{_convaccuracy}

_INNER_CONV => RFSP
_INNER_CONV => ProcessT15 {_cerbcase} {_modelname} rfsp_output
_INNER_CONV => _INNER_PREPAREFOR_RFSP IF CompGreater {_t15_error_max}
{_convaccuracy}

_INNER_PREPAREFOR_RFSP => /bin/rm -f {_unit11} rfsp_output STORE
{_store}{_cerbcase+1}
_INNER_PREPAREFOR_RFSP => cp {_unit11}-old {_unit11}

_INNER_CONV => _INNER_CONV_TUF IF CompGreater {_cerbcase} 3

_INNER_CONV_TUF => PrintLine \tRepeating TUF for case {_cerbcase} Max error
{_t15_error_max;-6.2f}%
_INNER_CONV_TUF => cd {_linkdir}
_INNER_CONV_TUF => GenPowTrans {_iterdir}/powtrans-{_ppid} STEP
_INNER_CONV_TUF => assemble_file _linktuf tuf_input tuf
_INNER_CONV_TUF => ParseTemplate tuf_input tuf.input
_INNER_CONV_TUF => /bin/rm -f {_tufckp}1
_INNER_CONV_TUF => TUF tuf.input tuf.output tuf.log
_INNER_CONV_TUF => TUF_CONV
```

The text prior to the '=>' symbol is the loop identifier, and the text after the '=>' symbol is the instruction to be executed at this step. Below is a more detailed explanation of the '_INNER_CONV' loop with comments contained within the bolded **{}**.

_INNER_CONV => RFSP **{1st step in INNER_CONV, call to execute the RFSP code}**

_INNER_CONV => ProcessT15 {_cerbcase} {_modelname} rfsp_output **{2nd step in INNER_CONV, a call to the 'ProcessT15' sub-routine that interprets a select portion of the RFSP output}**

_INNER_CONV => _INNER_PREPAREFOR_RFSP IF CompGreater {_t15_error_max} {_convaccuracy} **{3rd step in INNER_CONV, a conditional 'if' statement that starts the _INNER_PREPAREFOR_RFSP loop if an error is greater than the convergence criteria stored in the '_convaccuracy' variable}**

4.2 Reactor Regulating System Emulator (RRS Emulator)

The purpose of the Reactor Regulating System (RRS) in the CANDU reactor is to monitor and regulate the reactor power distribution by either increasing or decreasing the reactivity within the core. In addition, RRS performs reactor power measurements and calibrations; calculates power demand for controlling the reactor power; performs both bulk (global) and spatial power control and also controls the zone controller system response and regulates other reactivity devices.

An emulator of RRS has been developed for the RFSP code using PhysicsShell as the interface governing the interactions between the RRS Emulator and RFSP. . The fundamental logic of this RRS Emulator is based on that used in the SMOKIN [5] code.

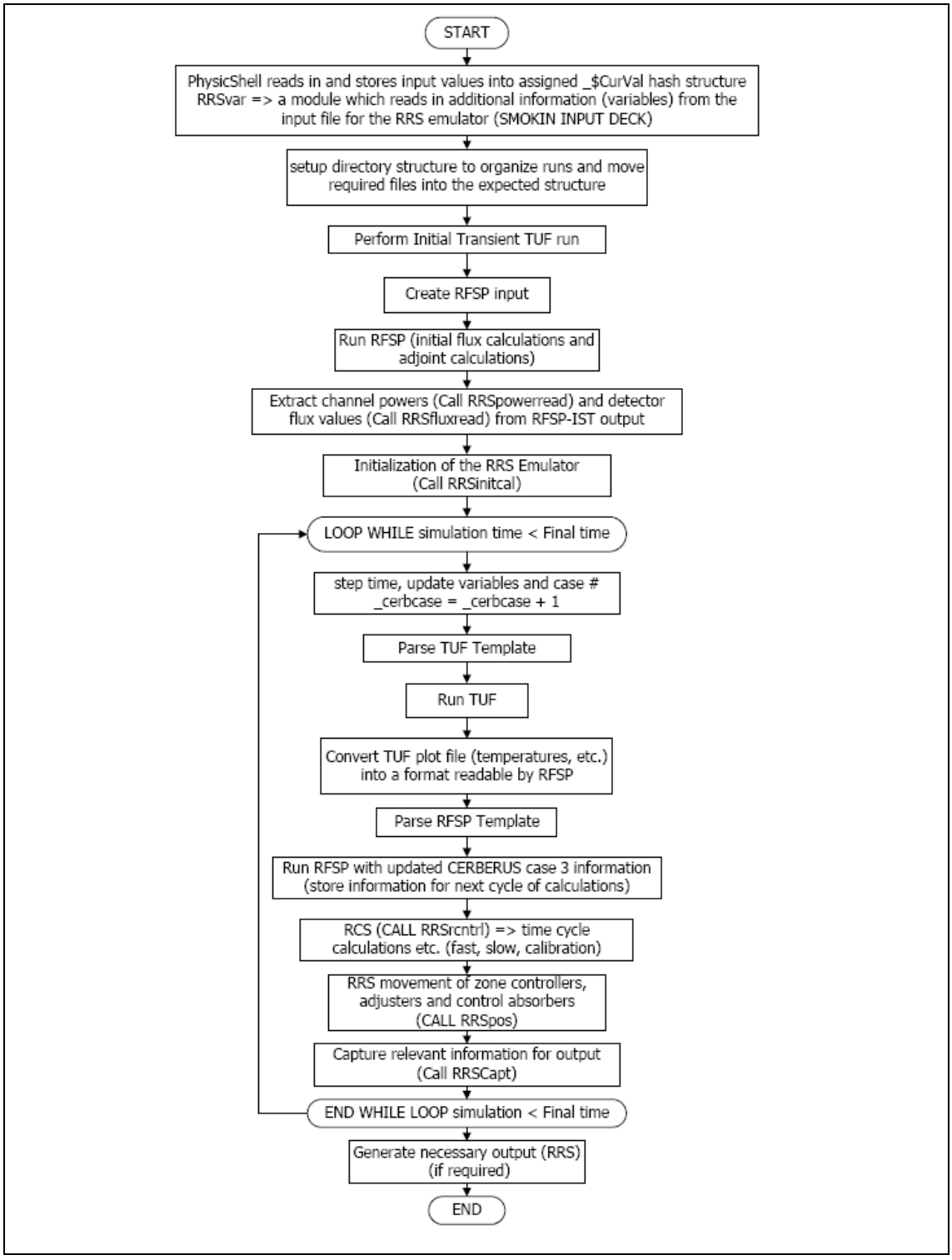
The use of PhysicsShell enables a coupling of a thermal hydraulics code. Thus, transient simulations incorporating RRS feedback may be performed.

The RRS emulator has been developed in PERL using PhysicsShell conventions of programs and routines in mind. As a result, the user interface of the RRS emulator is identical to that used in any other PhysicsShell application. This includes functions such as setting up appropriate directory structures, defining and altering variables, and manipulating input and output files as necessary.

The proper coupling of the RRS emulator, physics code, and thermal hydraulic code is achieved by assembling a master program that governs the overall sequence of events for all three codes.

A typical PhysicsShell program flow is shown in Figure 3.

Figure 3: Typical PhysicsShell Program Flow with RRS Emulator



The sub-routines in Figure 3 that begin with 'RRS', for example RRSinitcal, and RRSrcntrl, are external to PhysicsShell and have been developed specifically for the RRS Emulator application.

5.0 CONCLUSIONS

Safety analysis of nuclear reactors often requires the application of several computer codes to model a specific scenario. To complicate matters, the computer codes are often required to exchange information between each other. A controlled and generic method is required to govern such multi-code interactions. The solution proposed is the PhysicsShell program.

The PhysicsShell program has been successfully used to manage multi-code analyses such as Loss of Coolant Accidents, Best Estimate Analysis with Uncertainty, and modelling of the Reactor Regulating System. In addition, PhysicsShell is generally used to provide coupling in analyses where there is a need to perform repeated code executions and has been used to overcome an RFSP code limitation imposed by the fixed number of CERBERUS cases.

Overall, PhysicsShell offers a flexible platform for controlling a multi-code environment.

6.0 REFERENCES

1. B. Rouben, “RFSP-IST, The Industry Standard Tool Computer Program for CANDU Reactor Core Design and Analysis”, Proceedings of the 13th Pacific Basin Nuclear Conference, Shenzhen, China, 2002 October 21-25.

2. Cheng, D., “TUF User’s Manual – Generic Input, Code Version 1.0.3.5.1,” Report No. N-REP-06631.02-10009 R03, September 2001.

3. Sills, H.E., and Liu, Y., “FACTAR_SS Initial Fuel Conditions for Fuel Channel Transient Simulations,” Paper presented at Fifth International Conference on Simulation Methods in Nuclear Engineering, Montreal, Canada, September 1996.

4. Westbye, C.J., Brito, A.C., Mackinnon, J.C., Sills, H.E., and Langman, V.J., “Development, Verification and Validation of the Fuel Channel Behaviour Computer Code FACTAR,” Presented at the 35th Annual CNA/CNS Conference, Saskatoon, Saskatchewan, Canada, June, 1995.

5. M. Gold, J.C. Luxat, “SMOKIN – A Code for Time-Dependent Three-Dimensional Neutronics Calculations in CANDU-PHW Reactors Based on Nodal Kinetics Theory – Application to Analysis of Loss of Coolant Accidents,” Presented at the International Conference on the Physics of Reactors: Operation, Design and Computation, April 23-27, 1990 Marseille, France.