INTEGRATION OF AN ENGINEERING-GRADE, RELOAD SPECIFIC CORE MODEL INTO A REAL-TIME TRAINING SIMULATOR

Jeffrey Borkowski, Kord Smith
Daniel Hagrman, Joel Rhodes, III
Studsvik Scandpower
504 Shoup Avenue, Suite 201
Idaho Falls, ID 83402
jeff @soa.com

1. INTRODUCTION

This paper describes the integration of SIMULATE-3R (S3R) into the Oconee Nuclear Station Simulator's Windows Integrated Simulator Environment (WISE). The primary purpose of this effort was to enable reload-specific simulation by direct usage of nuclear core design data. The sections that follow explain the relationship of the simulator core model to the core design models, the integration with existing simulator components, and the usage of the integrated WISE at both the Oconee Simulator and at SSP.

2. OBJECTIVES

The integration procedure was shaped primarily by the objectives of the ONSS staff, the Duke Power Core Physics Group, and Studsvik Scandpower. All groups had the overriding objective of cycle specific simulator performance, but with some differences in focus.

- The focus of the core physics group was to facilitate direct usage of core design data in the simulator without any ad hoc collapsing and/or tuning and the associated time burden of that sort of data manipulation.

- One focus of the ONSS staff was to perturb the usage environment of the current simulator as little as possible, allowing the training staff to maintain as much as possible of their current patterns, techniques, and approaches to scenario generation. A second major objective was to integrate S3R into their structured development environment, preserving consistent and longterm software configuration control.
- The Studsvik Scandpower (SSP) focus was primarily to satisfy the above customer objectives but simultaneously preserve the general structure of the software such that it remained consistent with the core design codes from which it originated.

3. PURPOSE FOR DEVELOPMENT OF S3R

3.1 STATUS OF PREVIOUS GENERATION MODELS

Many real time core models of the previous generation tended to use relatively simple core models as a result of hardware limitations at the time of their development. Several of the simplifying assumptions include:

- transient simulation is performed in the axial direction only,
- azimuthal effects are treated synthetically via externally determined tilting functions, and
- thermal hydraulic feedback is treated through the lumping of coolant channels into representative sub-regions in the core.

The level of detail is constrasted in Figure1

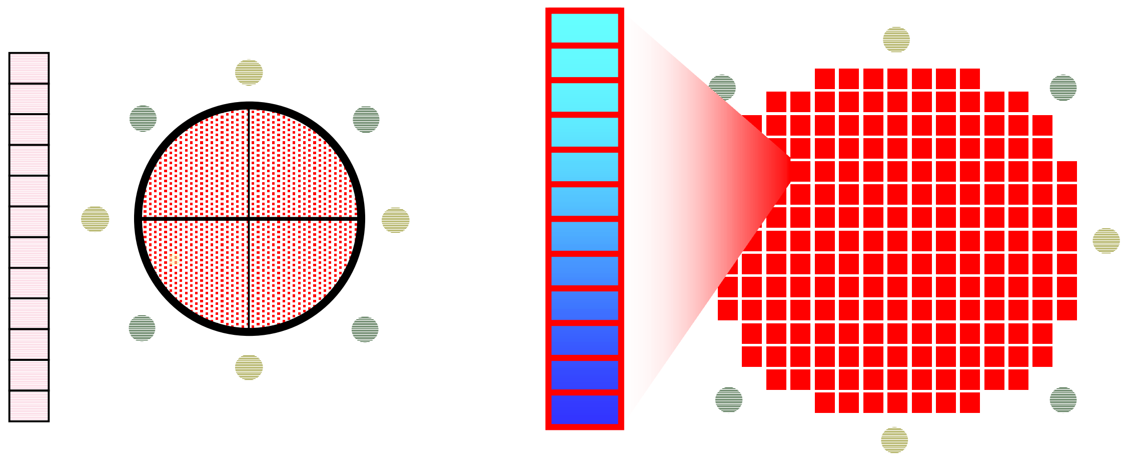


Figure 1. Contrast in level of detail between a typical simulator core model and a nodal, three-dimensional transient analysis model.

An additional limitation of these core models is their ability to represent accurately the core as designed. One reason is the numerical models in the simulator are not as advanced as those in the core design tools. Another reason is that the basic nuclear data used in generic simulator core models does not accurately represent the behavior of modern lattice designs and reload strategies. As a result the static and dynamic characteristics of the training simulator may not represent those of the actual core. One possible approach to this problem is for the core physics group to make parametric adjustments to the nuclear data and external functions in the simulator, such that the simulator core is tuned to the core design. The limitations of this approach are:

- the tuning process must be performed every cycle,
- the process is labor intensive, and
- the result typically can be validated only with steady-state calculations.

An altenative method is presented here, which describes the extension of familiar core design codes and processes to a real-time transient core model. Such a model is inherently cycle specific and demonstrates high fidelty to the core design. Further, this approach uses no external data manipulation, collapsing, or other such tuning.

3.2 BASIC NUCLEAR DATA

The reload-specific data is generated by the core physics group as part of the familiar core design process (Figure 2). Basic two-group nuclear data is generated with the lattice physics code CASMO-4[1] (or its predecessor, CASMO-3). This data is stored in a binary library for use in SIMULATE-3. SIMULATE-3 uses an advanced nodal method (QPANDA[2]) to solve the three-dimensional neutron diffusion equation. In addition to loading and depleting the core model, SIMULATE-3 reconstructs local pin powers and detector responses. Every assembly and all reflectors are modeled explicitly.

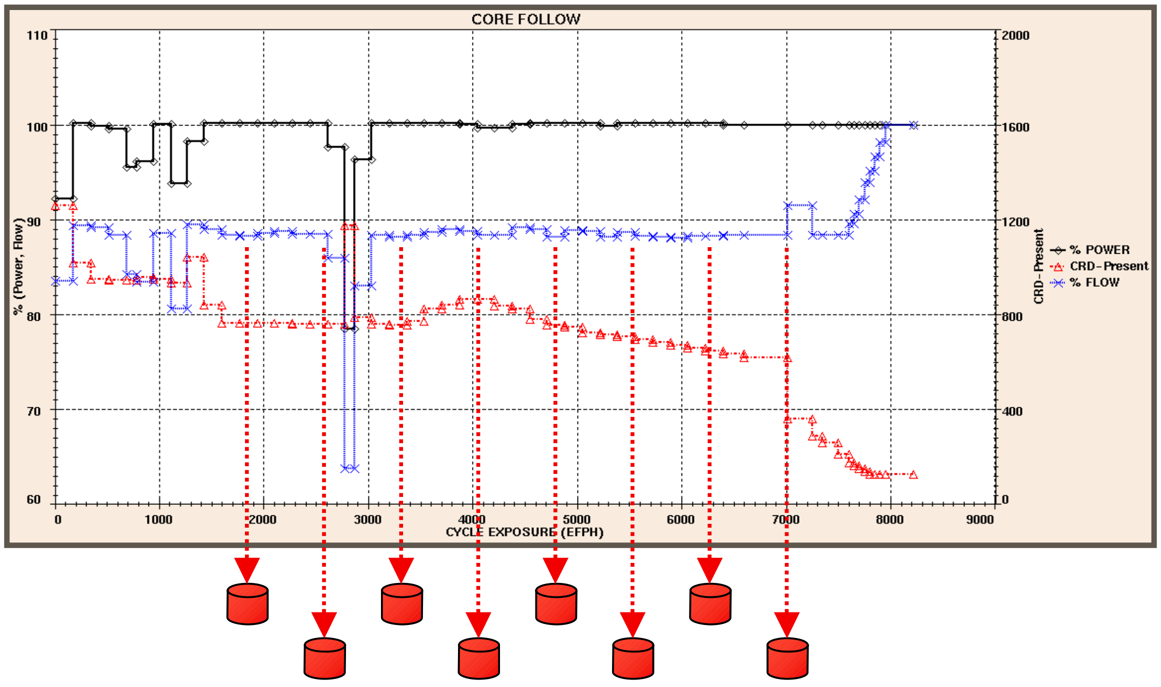


Figure 2. Schematic of In-Cycle Data Generation

5. OCONEE’S SIMULATOR ENVIRONMENTS (WISE and VISE)

5.1 OVERVIEW OF THE ENVIRONMENT

The Oconee Simulator runs native on a single DELL dual CPU machine running Windows NT. The software is designed to take advantage of the WIN32 API which incorporates a multi-process/multi-threaded design with a common shared memory architecture. This architecture allows the simulator code to be designed into small segments and processes based on common functionality. This significantly facilitates the design and debugging process as will be shown shortly. Presently the simulator has two environments:

- WISE (Windows Integrated Simulator Environment)

  Production/Training Environment
  Runs on the Production computers and is the Simulator Design Database software.

- VISE (Virtual Integrated Simulator Environment)

  Simulator Developer Environment
  Executes on the simulator developers personel development environment workstation and is maintained solely by the developer. This code is a superset of the WISE software code base and is considered volitile.

5.2 DEVELOPMENT ENVIRONMENT (VISE)

Visual Studio^{TM} 6.0 is the development environment used for design and debug of simulator software. The Oconee Simulator software is comprised of several different languages including C, C++, Visual Basic, and Digital FORTRAN. The language chosen for a particular task is dependent on the intended function. The language implementations are shown below:

- Executives - C and C++
- Modeling – FORTRAN
- Network communications – C and C++
- Instructor Station – Visual Basic and C
- Support Utilities – Visual Basic

The table below list the language with the associated activity:

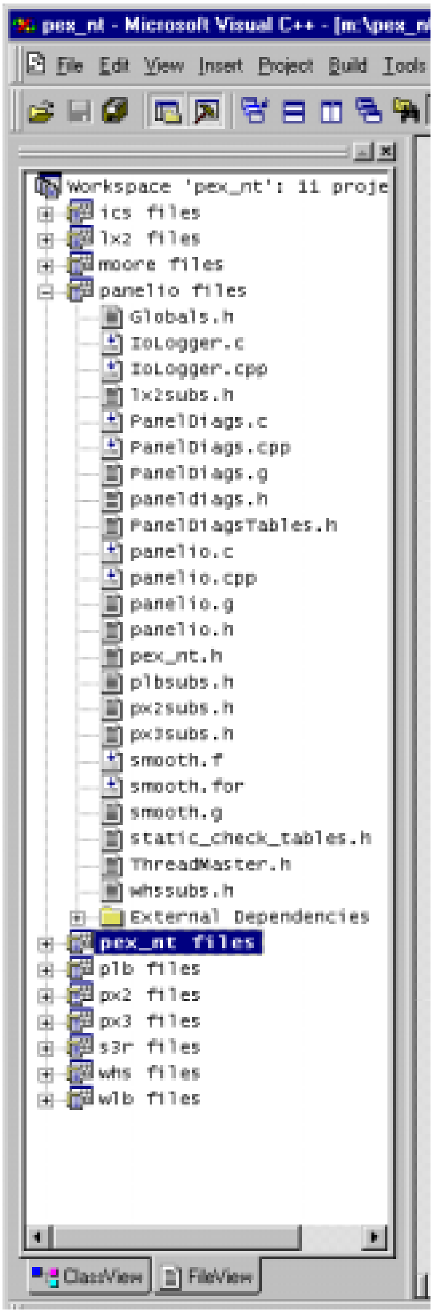
- FORTRAN – All thermodynamic modeling and legacy simulator software
- C and C++ - All process executives and network interface software

- Visual basic – Instructor Station, Offline support utility processes, and all database access software

Visual Studio^{TM} makes it possible to seamlessly integrate FORTRAN, C, and C++ software. This tight integration facilitates choosing a language for a particular function (one process may incorporate several different software languages) based on the inherent qualities of that language. With the proper Visual Studio^{TM} project design, the software developer can easily intermix C, C++, and FORTRAN in the same process.

5.3 DESIGN VERSUS RUNTIME

The design and runtime environments are equivalent (code is not optimized for WISE execution) on the simulator and this facilitates full source code debugging within the runtime environment, regardless of language selection. The Oconee simulator source code in the Visual Studio^{TM} is WYSIWYG. The Visual Studio^{TM} project includes all source (each source module is explicitly process activity. In other words, standard practice is to execute the simulator from within the Visual Studio^{TM} when developing code. The developer seldom leaves the Visual Studio^{TM} environment when designing and debugging simulator code. This is invaluable for tracking hard to find anamolies and bugs which may occure during training sessions or for tracing program execution to ensure correct program execution. With the production and debugging environments being the same, this makes it much easier to reproduce simulator anomolies (all any aborts or math errors report the actual address that can be viewed in the Visual Studio^{TM}). At Oconee, each simulator software developer has at his desk the complete simulator source code, runtime database, simulator initial condition file sets, and Windowed Instructor Station (WIS). This allows each developer to work in a Virtual simulator environment which is totally isolated from the production environment. When in VISE mode, the simulator initializes all I/O data structures so that it appears that I/O is available. Note that when in this mode, all other simulator processes that handle external interfaces are not executing and only the model dynamics are running. This significantly minimizes the amount of dead code that must be carried with the simulation. When the software developer determines that the code is ready for test, the software developer will attach the WISE computer to the developers VISE workstation (via network share) and execute the test process on the WISE computer (the VISE environment switches to a WISE environment), thus allowing complete testing of the developers


pex_nt - Microsoft Visual C++ - [m:\pex_nt
File Edit View Insert Project Build Tools
ClassView
FileView

```
Workspace 'pex_nt': 11 proje
+ ics files
+ lx2 files
+ moore files
- panelio files
    Globals.h
    IoLogger.c
    IoLogger.cpp
    lx2subs.h
    PanelDiags.c
    PanelDiags.cpp
    PanelDiags.g
    paneldiags.h
    PanelDiagsTables.h
    panelio.c
    panelio.cpp
    panelio.g
    panelio.h
    pex_nt.h
    plbsubs.h
    px2subs.h
    px3subs.h
    smooth.f
    smooth.for
    smooth.g
    static_check_tables.h
    ThreadMaster.h
    whssubs.h
  + External Dependencies
+ pex_nt files
+ plb files
+ px2 files
+ px3 files
+ s3r files
+ whs files
+ wlb files
```

Figure 3. Typical Visual Studio Project view.

software without ever making a single change on the WISE computer. This design has significant benefits compared to the old primary/secondary development system approach (Note: each software developer has a separate and independent VISE workstation at his desk). Once the developers software is deemed ready for production (fully tested and validated), the developer will integrate the appropriate software modules (subroutines) into the simulator design database. This code will then be compiled into the WISE production process and distributed to all WISE computers.

The VISE design was utilized by SSP in this project. SSP received a copy of the Oconee VISE model thus allowing the developers at SSP access to all of the same development tools and environments that are available at Oconee. SSP developers could then work with the full simulator load just as if they were at Oconee. This has proven to be very successful and very efficient.

5.4 NT PROCESS SEGMENTATION

As shown earlier, WISE is comprised of five different processes. The list below describes this segmented functionality and software languages used:

- Simulator Executive and Dynamic modeling (S3R model)
    - Language: C, C++, FORTRAN
- I/O Executive
    - Language: C
- Plant Compuer Executive
    - Language: C, FORTRAN
- Radiation Display
    - Language: C, FORTRAN
- XRVLIS Display
    - Language: C, FORTRAN

This segmented design functionality yields great benefits from the software developers perspective. The developer has no need to carry the additional functionality of external interfaces during the design and debug phase. Generally, the software developer will only execute the simulator dynamics executive during this phase of development and the VISE environment is totally isolated from any other simulator modification activities. This gives the developer significant freedom in designing code as the development and test environments are completed in total isolation. Only after the change has been fully validated will it be incorporated into the simulator design database.

6. S3R INTEGRATION INTO WISE

6.1 S3R AS A PROCESS MODULE

The S3R model was added to the WISE Visual Studio^{TM} project as a WIN32 static library project. All S3R source code modules reside in this project group. Additionally, a global "Preprocessor Variable" was defined which allowed for a single source base to be maintained thoughout the project (Old core model exist along side S3R). Even though the S3R source was included in the project, its inclusion into the linked process could be turned on and off by the "Preprocessor Variable", and during the compile process the source code would automatically configure itself for proper execution. The interface to S3R is through a single subroutine call with two passed arrayed parameters. One array is an input array and the other array is an output array. A small amount of wrapper code was added to pack and unpack the input and output arrayed data.

6.2 TRANSIENT AND REAL-TIME PERFORMANCE

S3R is a specialized version of SIMULATE-3K[3]. SIMULATE-3K is the transient analog of SIMULATE-3, also using the QPANDA nodal model. Thermal-hydraulic conditions are modeled using a five-equation Drift Flux fluid model coupled to a pin conduction model[4].

All reload core data comes from SIMULATE-3 restart files which are generated at multiple points in the cycle depletion. SIMULATE-3K reads the core follow nuclear data library directly.

Real-time performance in the S3R implementation is achieved via parallel and serial accelerations of SIMULATE-3K. Parallel acceleration is achieved using threads. The major threaded modules are:

- the tabular cross-section evaluation,
- the thermal-hydraulic channel sweep,
- the neutronic flux iteration,
- the delayed neutron precursor integration, and
- the fission source evaluation.

These modules account for 80 per cent of the required CPU execution time. The speed-up factor achieved via threading depends on the hardware. On average, 75 percent theoretical speedup is seen when taken across all threaded components.

Serial acceleration is achieved by modifications to time-step, nodalization, and nuclear data evaluation. In all cases, S3R solves its field equations for every assembly radially and typically uses the following run time configuration:

- 0.25 s time step,
- 12 axial, 1 radial node per assembly,

• Two-dimensional cross-section table lookup, and,

• Periodic recalculation of the nuclear coupling coefficients matrices, typically once per second.

No physical models differ between S3R and SIMULATE-3K, so as these restrictions are relaxed, the SIMULATE-3K solution is obtained. It should be noted that because the physical models of SIMULATE-3K and SIMULATE-3R are identical, SIMULATE-3K can be used in a slower than real-time mode to assess the performance of the real-time mode.

Real time performance for the ONSS was obtained using a 2 CPU Pentium II 400 MHz machine with approximately 50 per cent spare margin. Additional details on the acceleration techniques used in S3R, as well as timing studies, have been published by Rhodes et al[5].

Table 1: Configuration Differences Between S3K and S3R

S3R	S3K
12 Axial Mesh	24/25 Axial Mesh
PWR: 1 radial nodes/asm	PWR: 4 radial node/asm
BWR: 1 radial node /asm	BWR: 1 radial node /asm
0.2 - 0.25 sec time step	Variable and Automatic
Period Coupling Coefficent Updates	Countinuous Coupling Coefficient Updates
Fixed Fluxed Iterations	Variable Flux Iterations
No Implicitness Iteration	Full Implicitness Iterations
Full Pin Power Reconstruction	Synthetic Pin Power Reconstruction

6.3 CYCLE SPECIFIC DATA

From a core physics point of view, the primary advantage of the S3R core model is it uses the exact same data that the core physics group uses, and uses that data in the same format. The core physics group designs, depletes, and analyzes the reload using the SSP products CASMO-4 and SIMULATE-3. In this process, two fundamental binary data sets are generated and used:

- A basic nuclear data library which represents the fuel characteristics functionalized against burnup and feedback variables.

- One or more restart files. These files are generated during depletion analysis and capture the instantaneous core state at a particular burnup.

These data sets provide all the necessary data for S3R initialization. Further, the numerical methods employed in S3R are the same as in the core design codes, so the fidelty of the S3R physical methods to the offline codes is extremely high. In practice, the core design group actually generates many more restart files than is practical for simulator usage. A pratical subset was established as several Effective Full Power Day (EFPD) statepoints:

- 0 EFPD, for Zero Power Physics Testing and startup
- 5 EFPD, for BOL training
- 250 EFPD, for MOL training
- 440 EPFD, for EOL training

In this procedure, the effort burden on the core physics group is basically file transfer to the ONSS staff.

7. INTEGRATION STRUCTURE

7.1 PRIMARY CONSTRUCTS

The ONSS development environment is based on two primary constructs:

- a modular source code configuration which is fully integrated into the Microsoft Visual Studio^{TM} development system; and
- a shared memory global database, which handles variable registration and passes information among the modules and to the simulator display.

The Visual Studio^{TM} project is organized into a hierachy of several subprojects. These subprojects have been created along the lines of physical function, or in some cases to differentiate among source code from multiple vendor.

7.2 ONSS SIMULATOR COUPLING

The S3R core model is coupled to the pre-existing reactor coolant system (RCS) and control system. Thermal-hydraulic boundary conditions and rod positions are obtained from these models. S3R returns thermal power, excore detector response, incore detector response, and thermocouple temperatures to the RCS and control system.

This coupling is implemented via the ONSS Windows Instructor Station (WIS) developed by the Oconee Simulator Support Group. The WIS acts as both an executive and developmental system for the simulator as a whole. The WIS is fully integrated in the MS Visual Studio^{TM} environment. As such, each individual physical model is a unique project in this environment, addressing both local memory and a partition of shared (or “global”) memory. The contents of global memory are typically those data segments that are required for panel display, snapshots, or for coupling to another model.

7.3 USAGE AND VERIFICATION

Because the SIMULATE-3 restart files are accessed directly, the simulator may be initialized at any cycle exposure. Typical base starting exposures are 0 EFPD for physics testing and approximately 5, 250, and 450 EFPD for in-cycle training, as stated above. Instantaneous conditions of the core model may be saved via the intrinsic snapshot capability of the WIS.

Many off-nominal transients do not have a measured assessment capability. However, it is possible to assess the performance of the S3R model against zero power physics tests (ZPPT) for a known cycle. When assessed against Oconee 1 Cycle 18, the S3R based model matched the measured critical rod position at reference boron to within 1%.

CONCLUSION

The inclusion of the S3R core model in the ONSS has facilitated the direct usage of reload-specific core design data in a simulator training. The procedures required for this effort are greatly simplified because the simulator model accesses directly the core design calculational result. Additionally, the usage of modern programming tools in this development effort has facilitated rapid and robust development.

REFERENCES

1. M. EDENIUS, K. EKBERG, B. H. FORSSEN, D.G. KNOTT, "CASMO-4 A Fuel Assembly Burnup Program User's Manual," Studsvik/SOA-95/1, Studsvik, 1995.

2. K. SMITH, D. VERPLANCK, M. EDENIUS, "QPANDA: An Advanced Nodal Method for LWR Analyses," Trans. Am. Nuc. Soc., **50**, 532 (1985).

3. J. BORKOWSKI, J. RHODES III, P. ESSER, K. SMITH, "A Three-Dimensional Transient Analysis Capability for SIMULATE-3," Trans. Am. Nuc. Soc., **71**, 456 (1994).

4. D. KROPACZEK, K. SMITH, J. BORKOWSKI, "A Fully Implicit Five Equation Channel Hydraulic Model for SIMULATE-3K," Joint Intl. Conf. on Math. Methods and Supercomputing for Nucl. Applications, 1421, (1997).

5. J. RHODES III, K. SMITH, D. HAGRMAN, J. BORKOWSKI, "Real-Time Reactor Simulation with a Multi-Threaded Shared Memory Version of SIMULATE-3K," Advances in Nuclear Fuel Management II, EPRI TR-107728-V2, 20-1 (1997).