TVA Browns Ferry Simulator EHC System Upgrade Using Woodward's NetSim™ Simulation Package

W. Todd Sneed, nHance Technologies, Lynchburg, VA
Van Miller, TVA Browns Ferry Nuclear Power Plant, Decatur, AL
Brian Baker, Woodward Governor, Fort Collins, CO

KEYWORDS
NetSim, Control Systems, TMR MicroNet

ABSTRACT
In the spring of 2000, the TVA Browns Ferry Nuclear Plant awarded a main turbine Electro-Hydraulic Control (EHC) system upgrade contract to GE Global Controls Services, the former Global Services Division of Woodward Governor. The EHC upgrade package consisted of two Woodward TMR MicroNet™ controllers, hard-panel and Human Machine Interface (HMI) operator displays, and an associated simulator upgrade. This paper outlines the various techniques successfully used to interface the Woodward NetSim™ simulation package with the existing Browns Ferry Nuclear Plant simulator.

NETSIM BACKGROUND
nHance Technologies, then the Simulation Services Division of Framatome Technologies, was first contacted by Woodward in 1995 to investigate the possibility of developing a simulation platform that would allow Woodward simulation engineers the flexibility to validate control logic and control logic modifications on a simulator, rather than by using the actual hardware. Of the numerous benefits to Woodward was the ability to single-step the control logic and step into the control algorithms. Since the process model was simulated as well, this provided Woodward engineers a unique ability not previously possible. Reference [1] describes the simulation package in more detail, which was subsequently named NetSim.

Today, Woodward and their systems integrators have successfully used the NetSim simulation tools to accurately predict control logic responses on more than 40 projects, including gas turbine generator packages, co-gen stations, mainline steam turbines, large process refrigeration systems, natural gas pipeline stations, hydro power generating stations, and marine applications. This paper documents the interface techniques used to expand the list of successful uses of NetSim to include nuclear power plant simulation.

NetSim Simulation Techniques
The basic concept behind the NetSim simulation was to use, to the greatest extent possible, the C-code generated by the Woodward Coder application. The early versions of NetSim accomplished this by post-processing the native C-code such that it could be compiled and executed on a PC. In order to accomplish this task in an automated fashion, a series of PERL scripts were developed. These scripts were initially faced with two main tasks. First, the Woodward-generated C-code contained a number of references to hardware memory locations, and second, the real-time operating environment used in the Woodward target hardware automatically sequenced the code for each rate group thread.

Memory Mapping
The generated C-code for the Woodward hardware typically stored and referenced variable and state information as hardware-specific memory locations. To avoid having to translate this information into variable names, and to assist with the save and restore feature, it was decided to take advantage of Windows NT's flat memory model, and allocate a region of memory on the PC of identical size and, where permissible, location as that used in the hardware. Using this technique, most of the current state of the system could be saved, and restored immediately upon request by simply dumping the allocated memory region to a file.

Rate Group Execution
During the processing of the Woodward generated C-code, the PERL scripts automatically maintain a list of each subroutine, and the rate group to which it belongs. Then, after converting all of the existing code for each rate group, a special function is created that calls, in the proper sequence, all of the functions belonging to that rate group. Using this technique, all of the code can be executed for a particular rate group by simply calling the rate-group's governing function. The resulting C-code is then compiled into a Windows Dynamic Link Library (DLL).

Control Executive
Converting the Woodward generated C-code into a Windows DLL solved a major portion of the simulation problem, but a way to execute the control code and communicate the control's Inputs and Outputs (I/O) to the control DLL was still needed. To accomplish these tasks, a standard application called the NetSim Control Executive was developed. The Control Executive was initially designed to load a control DLL, and cycle the rate groups.

In addition, it was responsible for communicating with the process model's main simulation executive to exchange I/O. The method decided upon for the synchronization between the Control Executive and the process model was to use Windows shared memory to store the control I/O. The process model would place the control inputs into shared memory and send an event to the control model that new control data was available. The Control Executive would copy the control inputs from shared memory into the control DLL's address space, then cycle the appropriate rate groups based upon the simulation time expired from the last call. Once the rate groups have been executed, the Control Executive would copy the control outputs to shared memory, and signal the process model that the control logic execution completed.

Human Machine Interface

The control executive is capable of using the Modbus protocol to communicate with a Modbus-based HMI using the same configuration information as the hardware. Since the Woodward Modbus configuration information is stored in the control logic generated by the Woodward tools, the control interface uses the same configuration. Thus, the NetSim product can use the same configured HMI application as the true Woodward hardware.

BROWNS FERRY PROJECT

The Browns Ferry Simulator Upgrade Project consisted of upgrading the Browns Ferry simulator to account for the EHC System upgrade.

The division of responsibility for the simulator upgrade portion of the Browns Ferry Project was as follows:

- Brian Baker of Woodward Governor was responsible for development of the interface spreadsheet, and providing advice, support, and guidance on the NetSim application.
- Van Miller of TVA was responsible for all aspects of programming on the Browns Ferry UNIX-based simulation computer. This included obtaining data from the process model, implementing the network interface routines, and synchronizing the network interface routines with the process model.
- Todd Sneed of nHance Technologies was responsible for designing the interface program for NetSim, making modifications to NetSim as required to support the simulator integration, and designing the TCP/IP network messages.

Challenges

Since the NetSim product existed at the time of the Browns Ferry contract award, the challenge was not how to emulate the Woodward control system. Rather, the challenge was how to interface the existing PC-based control emulation to the Browns Ferry Simulator. To accomplish this task, it was decided to provide Browns Ferry with a Windows 2000-based PC to execute the NetSim control emulation, and interface the control emulation with the simulator using standard TCP/IP-based network communications. To accomplish this, a series of basic messages was designed to handle the data exchange and executive communications. These messages were then implemented on both the PC and UNIX boxes. Once the communication messages were designed and tested, the task became one of interfacing the network interface applications with their corresponding simulation executives, and ensuring proper control model and process model synchronization.

TCP/IP Messages

The TCP/IP message definitions started with a standard message structure that included enough overhead information to ensure proper message processing as well as the message data itself. The standard structure is presented in Table 1.

Table 1, TCP/IP Message Structure

Name	Description
TotalLength	Total message length, including CRC
MessageKey	Type of message being transmitted
MessageID	Sequential Message ID
ResponseID	ID of message being responded to
Pbuf	Message dependent data
CRC	CRC check on entire message

Once the overall message structure was designed, it was necessary to work on the individual messages. The messages were divided into two categories: those messages sent from the Process Model and those sent from the Control Model. Representative lists of messages sent from the process model and control model are presented in Tables 2 and 3, respectively.

Table 2, Messages Sent from Process Model

Message	Description
InitComs	Sent to Initialize communications
DataExchangeCI	Sent when new control inputs are available
SaveDataImage	Request for control model to save its state
LoadDataImage	Request for control model to load a previously saved state

Table 3, Messages Sent from Control Model

Message	Description
CmdResponse	Response to various commands
DataExchangeCO	Control Outputs sent to process model
ComInfo	Communications & Message version information

Process Model Integration

The process model integration effort consisted of exchanging simulator global and control model data in addition to interfacing with the simulator executive controls. However, this was only half the integration effort. Once the two models were physically integrated, the real integration effort began; namely, interfacing a control system designed to work in plant conditions with a comparably large time-step based discrete model. Thus, two challenges were encountered during the process model integration effort; first, a method had to be developed to physically exchange data, and second the "real" plant environment verses simulation effects had to be minimized.

Physical Data Exchange using Sim2WGC

To physically exchange process model and control model data a simulator interface application, named Sim2WGC (Simulator to Woodward Governor Control) was developed. Sim2WGC was designed to be responsible for communicating with the process model directly via global data, and the control model via TCP/IP, in effect, acting as a communication "bridge" between the two models. However, directly exchanging model data was not always proper, as the control model and process model sometimes had different engineering unit requirements. To address this need, Sim2WGC was designed such that it could manipulate the process model data passed to the control model, and control model data passed to the process model as necessary to accommodate the different engineering unit.

Sim2WGC Data Manipulation

As mentioned above, the Sim2WGC application would sometimes have to manipulate the data sent to and received from the control model. For example, the control model assumes various input data is coming from plant transmitters that normally generate a 4-20 milliamp signal with gain and offset differences between individual transmitters. As such, the control model input blocks (or, sometimes a downstream block) usually applied a gain/offset to the input signal and/or would convert the 4-20 milliamp signal to engineering units for internal calculations. Since the simulator process model transmitters were "perfect" transmitters and represented the actual process value in engineering units, the Sim2WGC program converted the engineering units into a 4-20 milliamp signal based on the signal range expected by the control model and applied the gain/offset values as appropriate. This in effect, converted the process model data into a format expected by the control model. A similar approach was used when necessary to convert the control model outputs into units required by the process model.

Minimizing "Real" Effects on the Simulation

The Woodward TMR MicroNet control system has the capability to divide up the control code execution into separate rate group threads. The GE Global Controls control model design employed for the Browns Ferry EHC plant upgrade used a minimum rate group of 10 ms. Thus, in the "Real" plant environment, the control model receives various inputs from the plant every ten milliseconds. However, in the existing simulator model, the Turbine Controls were executed at 4 cycles per second (cps), or once every 250 milliseconds. Ideally, for the new simulation, the process model would be configured to execute the steam turbine simulation in 10 millisecond intervals (or 100 cps). Although raw computing power has increased dramatically over the past few years, a 10-millisecond execution rate for the process model was simply not practical. Therefore, the process model simulation can't execute at the same rate as the control model and the control model must perform several cycles using the same model data, resulting in slightly different control responses than in the "Real" plant. These different control effects had to be minimized.

The first, and easiest way to minimize the cycle discrepancy effects is to minimize the number of control cycles that must be executed using the same process model data. In other words, increase the cycle rate of the appropriate simulation systems as much as reasonably possible. Thus, the simulator models in the areas of the main turbine model, turbine control, main steam and reactor thermal-hydraulics were increased to 12 cps (83 milliseconds).

At a 12 cps cycle rate, the normal sequence of events is as follows: At the end of each cycle, the Sim2WGC application collects a list of control model inputs, and passes them to the control model. The NetSim Control Executive compares the current model time with its last model time, determines that 83 milliseconds has expired, and cycles the appropriate rate groups the appropriate number of times, collects the control outputs and passes them back to the process model. For example, on a typical cycle, the control model would cycle its 10ms rate group 8 times, its 20ms rate group 4 times, its 40ms rate group 2 times, and its 80ms rate group 1 time, in the appropriate order (10, 20, 10, 20, 10, 20, 10, 20, 40, …). Basically, since the rate groups were triggered with a single data exchange message from the process model, the control logic code executing in the rate groups uses the same control inputs. This sequence of events is of particular importance, for example, in the PID type controls, which are normally in the 10ms rate group. In the normal cycle outlined above, these controls integrate a static signal for 8 cycles before passing the control output (such as actuator demand) back to the process model. In general, from a simulation perspective, there are only three responses to this situation, namely:

- Execute the process model faster
- Modify the control model
- Analyze the impact, and live with the results.

As stated above, the first response was used somewhat, by increasing the model execution frequency from 4 cps to 12 cps, but increasing it to 100 cps was not practical.

The second possible response is a valid option, however, it was not desirable to modify the actual control logic, as it was extremely important to use the same control logic in the simulator as in the plant. However, the Woodward DCS allows for the manipulation of various "Tunable" parameters without actually modifying the control logic design itself. For example, one particular problem was the control model sampling of turbine speed input probes to determine if a probe was good or bad. The test used by the control model, which executed in a 10 ms rate group, was if the speed demand and current speed were different by a specific delta, which was "tunable." On fast speed changes and with simulator speed input from the simulator model changing only once per eight times for the control model demand request, the speed signals would sometimes be marked "Bad." This is where the simulator's control model was "de-tuned"; In other words, the set point was relaxed so the control model would allow a greater tolerance before marking the speed signal "Bad."

The third response is not really a response, but rather a way to determine how much of a problem exists. As it turns out, the fidelity of the simulator response due to this interface difference was analyzed after the cycle rate was increased, and the tunable parameters were de-tuned for the simulator. The result was that the simulator was able to adequately match plant data.

Control Model Communication Interface

Once the network communication message structure was outlined, a new application, named SimCom, was developed to implement the PC side of the communications.

Since SimCom was designed to act as a bridge between the Sim2WGC application (process model), and the control model, it needed to know how to "talk" to both. As mentioned above, the TCP/IP messages provided the methods necessary to communicate with the process model. These messages were implemented using the standard Microsoft Foundation Classes (MFC) socket classes while configuring SimCom to function as a "server," at least in the respect that it listened for connection attempts from the Process Model. Once the server socket received a connection request, it created a communication thread designed to listen for messages from the server and respond to them. In order to adequately respond to the messages, SimCom had to know how to "talk" to the control model.

In general, the approach to develop communications between SimCom and the control model was simply a matter of designing SimCom to emulate S_Master, nHance Technologies' simulation executive for which NetSim was designed to interface. To accomplish the S_Master emulation the S_Master communication routines were compiled into an interface DLL, which was loaded by SimCom to provide the interface functionality. A block diagram of the overall communication interface is presented in Figure 1.

Synchronization

In order to ensure proper execution of the control model, it was necessary to ensure the control model and process model were synchronized, meaning that the control model and process model executed their respective "cycles" at the same time relative to one another's cycle, thus ensuring repeatability of calculated events.

On the control model side, synchronization was built in by default, given the fact that the control model was continuously in a state of "freeze" until it received the control logic inputs from the process model. One of the parameters passed along with the control logic inputs was the "global simulation time". From this time parameter, the NetSim control executive could cycle the control model's rate groups as required to bring the control model to the desired state at the equivalent model time. After all the control model rate groups had been cycled, the NetSim Control Executive would collect the control model output parameters, and pass them along to the SimCom application. SimCom then forwarded the messages to the process model, allowing the process model the opportunity to process the outputs and continue its cycle.

This technique automatically addressed the Freeze/Run/Step states for the simulator, but it had one flaw. The process model did not contain a "global simulation time," rather; it contained a variable for "problem time." The basic difference between simulation and problem time is that problem time is reset to zero upon restoration of an Initial

Condition (IC). Since the control model uses simulation time as a replacement for the hardware clock, resetting the time to zero was unacceptable. To address this problem, a variable was created on the process model that was incremented with each process model cycle, and passed along to the control model to ensure cycle synchronization.

Given the techniques outlined above, tight synchronization with the process model would be guaranteed if the process model had the ability to collect the control model inputs and process control model outputs at the same point in every execution cycle. However, in this particular case, the Browns Ferry simulator model executed in a real-time operating system, thus presenting a further complication.

To address the synchronization issue on the process model side, it was decided to collect the control model inputs at the beginning of every "frame," which started every 1/12 of a second. The inputs were then passed along to the control model to process, and the outputs were received and incorporated into the process model at the end of every frame. Thus, synchronization was guaranteed only if the control model could process the inputs and provide the outputs within the 1/12 of a second time-slice. Since we could not guarantee this tight synchronization, a method was developed to identify when the control model and process model fell out of synchronization. It was then subsequently determined that this condition does not occur during normal operation of the simulator.

EXISTING LIMITATIONS

One of the existing limitations of the control model is that its data images are stored by dumping memory image to disk. This technique provides for quick responses to data image loads and restores, but at the expense of tagging the data image to a unique instance of the control model. In other words, if the control model changes even slightly, the memory location for the data, and or the size of the data could change, therefore invalidating the data images stored on disk, and forcing a complete rebuild of all the simulator IC files, which can be very time-consuming. This limitation is mitigated somewhat by the fact that some minor changes may permit reusing old IC's, although, this is only recommended for testing purposes, and not for training. It should be pointed out here that the modification of tunable parameters does not invoke this problem, as this can be accomplished by simply loading each IC, setting the tunable parameter, then resaving the IC.

CONCLUSION

This successful conclusion of the Browns Ferry Simulator Upgrade Project demonstrates how a well-designed control system simulation can be interfaced with a different platform based simulation model in a cost effective manner. NetSim has once again proven that it is fully capable of accurately and reliably emulating a wide array of Woodward hardware for use in control system validation, as well as a real-time operator training simulator environment.

REFERENCE LIST

[1] McWhorter, Scott, Brian Baker, and Greg Malan, "Simulation System for Control Software Validation." Presented at SCS Simulation Multi-Conference, April 6-10, 1997, Atlanta, GA

BIOGRAPHY

W. Todd Sneed is the President of nHance Technologies, which was established in March of 2001 when Framatome ANP divested the Simulation Services Division. Before founding nHance Technologies, Todd worked for 10 years at Framatome, initially performing Loss of Coolant Accident (LOCA) Analysis using the RELAP5 safety analysis code, and later, as the principal contributor and software architect for MMS Simulation Tools. Todd holds a B.S. in Nuclear Engineering from North Carolina State University, and an M.B.A. from Lynchburg College. Todd can be reached via e-mail at: todd.sneed@nhancetech.com

Van Miller of Tennessee Valley Authority (TVA) is the lead engineer on the Browns Ferry Simulator. Van has also worked on TVA's Sequoyah and Bellefonte simulators. His responsibilities include maintenance and modifications of simulator models, instructor station, I/O interfaces, stimulated controls, and system administration. Van holds a B.S. and Masters in Mechanical Engineering from Tennessee Technological University. Van can be reached via e-mail at: vnmiller@tva.gov

Brian Baker of Woodward Governor is the NetSim product champion. Brian joined Woodward in 1995, after having previously worked in the simulation business for ESSCOR and General Dynamics. Brian used his knowledge of simulation, and its benefits for control validation to convince his managers to fund and support NetSim, which has turned out to be a success story at Woodward. Brian holds a B.S. in Mechanical Engineering from Oregon State University. Brian can be reached via e-mail at: NetSim@woodward.com.

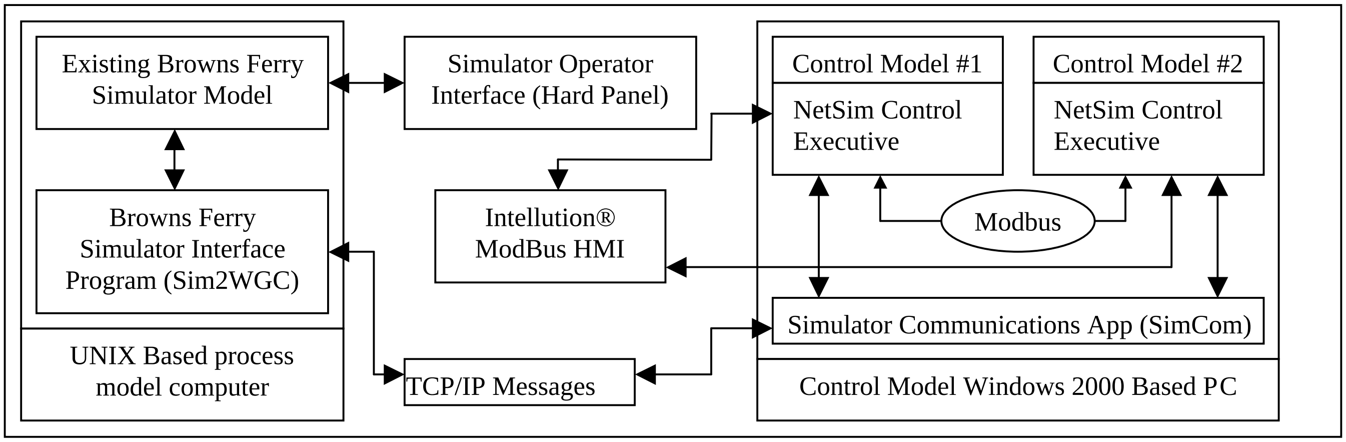

Figure 1, Communication Overview