THE POINT LEPREAU DESKTOP SIMULATOR

Macey MacLean, Jim Hogg, Hubert Newman
New Brunswick Power, Point Lepreau Nuclear Generating Station
Point Lepreau, New Brunswick, Canada, E0G 2H0

INTRODUCTION

The Point Lepreau Desktop Simulator runs plant process modeling software on a 266 MHz single CPU DEC Alpha computer. This same Alpha also runs the plant control computer software on an SSCI 125 emulator. An adjacent Pentium PC runs the simulator's Instructor Facility software, and communicates with the Alpha through an Ethernet.

The Point Lepreau Desktop simulator is constructed to be as similar as possible to the Point Lepreau full scope training simulator. This minimizes total maintenance costs and enhances the benefits of the desktop simulator. Both simulators have the same modeling running on a single CPU in the same schedule of calculations. Both simulators have the same Instructor Facility capable of developing and executing the same lesson plans, doing the same monitoring and control of simulations, inserting all the same malfunctions, performing all the same overrides, capable of making and restoring all the same storepoints. Both simulators run the same plant control computer software - the same assembly language control programs as the power plant uses for reactor control, heat transport control, annunciation, etc.

This is a higher degree of similarity between a desktop simulator and a full scope training simulator than previously reported for a computer controlled nuclear plant.

The large quantity of control room hardware missing from the desktop simulator is replaced by software. The Instructor Facility panel override software of the training simulator provides the means by which devices (switches, controllers, windows, etc.) on the control room panels can be controlled and monitored in the desktop simulator. The CRT of the Alpha provides a mouse operated DCC keyboard mimic for controlling the plant control computer emulation. Two emulated RAMTEK display channels appear as windows for monitoring anything of interest on plant DCC displays, including one channel for annunciation.

MOTIVATIONS

The engineering people in the Lepreau simulator maintenance group consist of a hardware person, a system software person, two modelers and a supervisor. So there is not sufficient resources available to maintain multiple separate simulators. Therefore there was a powerful incentive to make the desktop simulator out of the same components as the training simulator, to the greatest possible degree.

From a training department point of view, there is substantial potential advantage to having the two the same. Instructors can prepare lesson plans using either the training simulator or the desktop simulator, and may come to prefer the desktop simulator because it runs faster than real time. Students preparing for licensing exams would have the potential to get more simulator time. The range of students gaining simulator access can be expanded.

The training simulator is starting to see use in the tuning of plant process systems. In particular the condensate system controller parameters were recently set to new values in the plant with some success (and still more work to do) based on experience gained with the simulator, which the plant system

engineer is thinking of presenting as a paper at a future CNS conference. The desktop simulator makes simulator time more available for system engineers.

Persons developing or refining or validating plant operating procedures have found access to the training simulator useful, and now that access may become easier.

Persons developing or refining or validating plant control computer software have found access to the training simulator useful, and that access can become easier.

The greatest obstacle to simulator access is the difficulty of access to persons who have adequate knowledge of overall plant control. It takes years to find one's way through the more than 10,000 pages of plant operating manuals[1], and that same obstacle to general use is present in the simulator.

There are more sophisticated non-real time simulation codes than the process models used in the real time training simulator. The greatest fidelity potential of the desktop simulator is through modifying it to incorporate some of the more sophisticated codes. The training simulator is unexcelled in the fidelity and scope of its control logic. It is also unexcelled in the scope of its process modeling. So the overall fidelity of simulation can be enhanced by combining the best modeling assets available from whatever sources. A means of combining the real time simulator code with non-real time variable step length models is described later in this document.

A DESCRIPTION OF THE DESKTOP AND TRAINING SIMULATORS

Figures 1 and 2 show the components of both training and desktop simulators respectively. Each component is a box in the respective figures and each box has a description and an italicized alphabetic number from *A to R.*

Box A. Both simulators run the same DCC application program images, with almost no differences compared to the plant. There are a few patches to DCC application programs for the simulators. The simulators' DCC behaves as DCCX in the plant in almost all situations. However, three of the programs have been patched so that in the instance of certain failure modes these programs think they are running in DCCY of the plant.

Box B. The simulator DCC executive software is the DCC operating system. Like any operating system it schedules the running of application programs (in Box A), and provides hardware services (such as reading inputs or writing to outputs) for those application programs.

The simulator DCC executive is very different from the DCC executive in the plant because it was written to run on a different computer (the SSCI 125, not the Varian V73 of the plant), talking to some different I/O devices, and providing a broader range of services than the plant DCC executive. However, the two simulators run almost identical executive DCC software with only a few words patched to be different between the two simulators. These patches cover odd little details like "dummying out" the gateway driver for control computer data logging. This interface is present on the training simulator, but seldom used, and it is not yet emulated on the desktop simulator.

---

[1] Current plans call for all the operating manuals and emergency operating procedures to be on-line to every PC in the plant, but there is no specific date by which this is supposed to be achieved.

Boxes C and D. In Figure 1, the training simulator DCC hardware is represented by these boxes. This provides the same user interface as control room operators have in the plant main control room. The DCC keyboards and associated RAMTEK displays are set out in the training simulator in the same manner as in the main control room, and since the software in Box A is the same as for the main control room, so too are all the operator interactions.

Box C-D. In Figure 2, the desktop simulator replaces this DCC hardware with the DCC emulator software providing the same functionality. The emulator is described in the companion paper reference (1). The only difference as compared to the training simulator is that in the training simulator there are a total of 9 DCC keyboards and associated RAMTEK display channels plus 2 annunciation RAMTEK channels, whereas on the desktop simulator there is only 1 keyboard, one annunciation channel and the RAMTEK display channel that goes with the keyboard.

Similar to the main control room, the simulator software allows all functions of any DCC keyboard to be easily accessed from any other keyboard and display. So the desktop simulator allows one to do pretty well anything that could be done in the training simulator or main control room.

The main reason for the limitation on the number of keyboards and RAMTEK channels on the desktop simulator is the amount of space (and cost) it would take in an office cubicle to have all the screens required to emulate many keyboards and display channels.

Box E. The simulator modeling includes 173 different modeling modules. Most of the modeling modules are organized as being process modules, control logic modules, or valves and motors modules. The modeling covers everything that has the possibility of dynamic interaction in the nuclear plant with the control room and has safety significance in an accident. This includes all systems that have or potentially have a bearing on fuel cooling, or radioactive releases to the public.

The modeling on the desktop simulator is completely identical to the training simulator.

Box F. Figure 2 is accurate in respect to Box F, but Figure 1 Box F is a bit of a simplification.

There are about 5000 I/O points between the DCC and the modeling. About 2000 of these are for contact alarms which are only communicated on an exception basis (when they change from TRUE to FALSE or vice versa). The other 3000 I/O points are constantly being copied back and forth from modeling to the DCC in both simulators. This copying operation affords the opportunity to insert override values selected by the simulator instructor in place of the normal process values.

The copying of DCC I/O between modeling and the DCC is done in 2 stages on the training simulator but through only one stage on the desktop simulator. Both the desktop simulator and the training simulator have a so called YD family of programs that run in close fixed timing association with the modeling. These programs look after the overrides.

In the training simulator there is also another stage of copying that is done through a special bus interfacing computer called a DUSC. This second stage of copying through the DUSC is the only planned difference of any potential importance to comparative dynamic performance of the two simulators. This arises because the DUSC runs asynchronously and introduces further communications delays ranging from about 70 milliseconds up to about 1.4 seconds - the latter only in respect to very slow moving values that would not change much in that time frame.

Originally the different scaling of the different DCC AI's (analog inputs) was dealt with in the DUSC, but to increase commonality and to ease simulator maintenance, the scaling operation was moved into

the YD AI program. Proper scaling is set up by an off line utility that reads the DCC software images. Both the desktop simulator and training simulator are the same in this respect.

In one important respect the YD family of programs for the desktop simulator are different than for the desktop simulator: they are set up to talk directly with DCC I/O arrays used by the DCC emulator, instead of talking to the DUSC as in the training simulator. The DCC I/O arrays used by the emulator are all in numerical order of the DCC I/O address to simplify and speed the operation of the emulator, whereas the order of the I/O seen on the interface between the DUSC and the YD programs was set up with other considerations in mind. The YD programs of the desktop simulator achieve the necessary reordering of the I/O addresses as part of the same copying operation into which DCC I/O overrides may be interposed.

Box G. Control Panel I/O programs at this time are identical between the desktop simulator and the training simulator. In the training simulator a simulator instructor can turn off a heat transport pump without touching the panel hardware, just using the panel override software. On the desktop simulator the pump handswitch is simply operated by using the same override software.

Box H. Instructor Facility I/O programs are reliant on a custom written communications program. This software is run on both the desktop and training simulators. In the longer term Lepreau will either replace the custom written software with TCP/IP which is more reliable, or remove the need for this communications by having one host for both the instructor facility and other simulator functions.

Box I. There are significant differences between the training simulator and the desktop simulator with respect to the program dispatching arrangement.

For the training simulator, real time is important and is achieved by a hardware timer interrupt driven dispatching program. Significant time is spent waiting for the next hardware interrupt to start the next "leg" of calculations. The wasted time is about 50% of capacity for our new VAX 4105 computer.

One could easily run the desktop simulator with a real time dispatcher. Instead the objective is to get simulation as fast as possible. When each task is finished, the next task starts without waiting for a hardware interrupt to match clock time advancement to the rate of model time advancement..

One concern with a higher speed desktop simulator is whether the instructor facility and its associated communications can keep up. When computing load was reduced so modeling was running three times as fast as real time, there were no problems nor do problems appear close to happening.

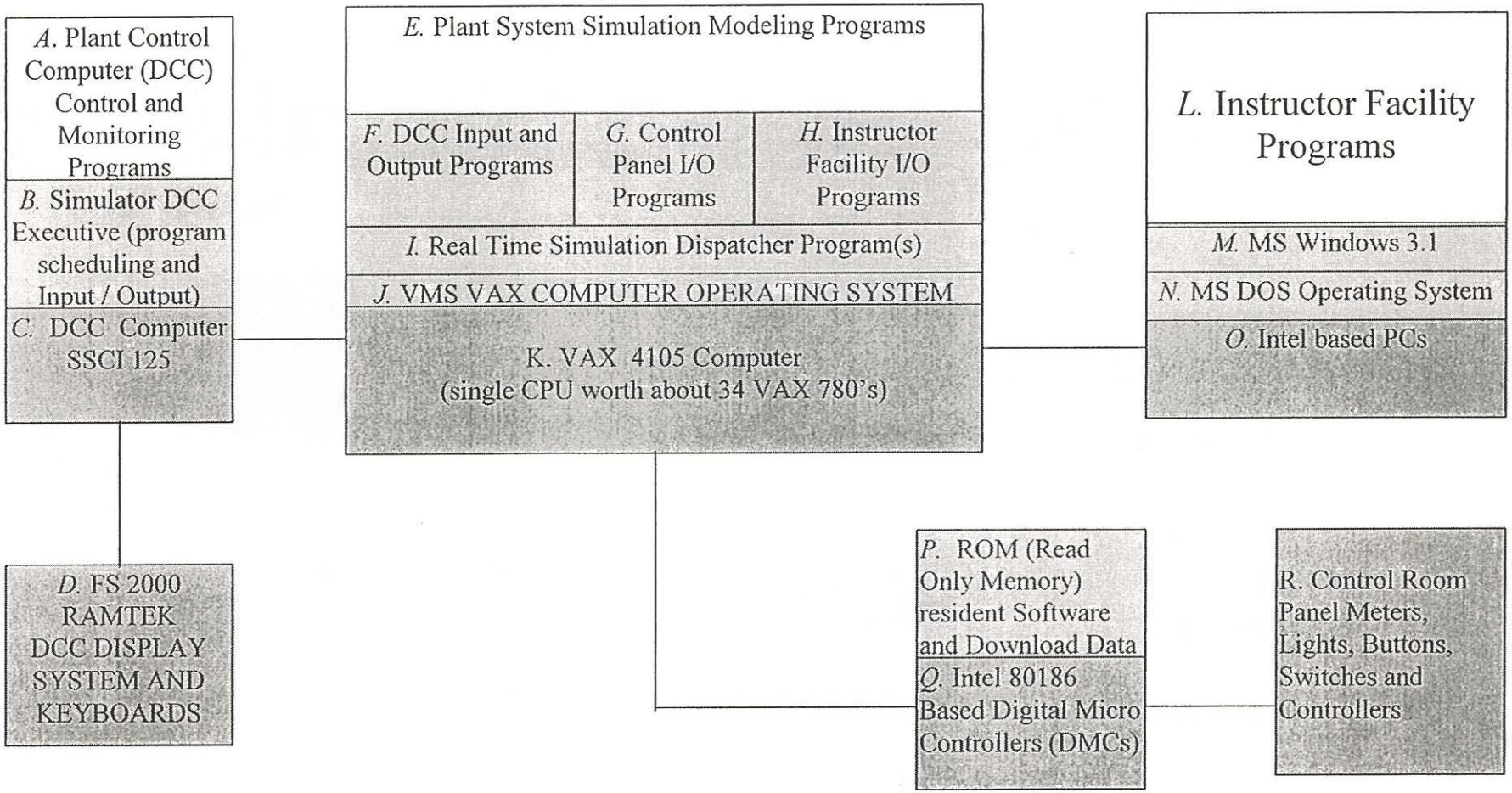
When the computer capacity of the training simulator was recently upgraded to the VAX 4105 from four VAX 3800's running in parallel, this provided an ability to exactly match the order of calculation of the modeling in the single CPU desktop simulator to the single CPU training simulator.

Another feature of the training simulator is that there are two distinct processes: the synchronous process which runs the modeling and an asynchronous process whose functions are deferrable - typically instructor control related functions. On the desktop simulator there is only one process, including all modules of both processes of the training simulator.

The DCC emulator module is called at the start of every leg of modeling calculations on the desktop simulator and is called to run for a period of time equal to the time up to the start of the next leg. On the training simulator the DCC hardware runs in parallel with the modeling computer - but since communications between modeling and the DCC is essentially a scheduled periodic process of the YD programs (see box F above), running in parallel is indistinguishable from running within a suitable dispatching arrangement.

Figure 1: Point Lepreau Training Simulator Architecture

(The Hardware Cost is $ Millions, the speed is real time)

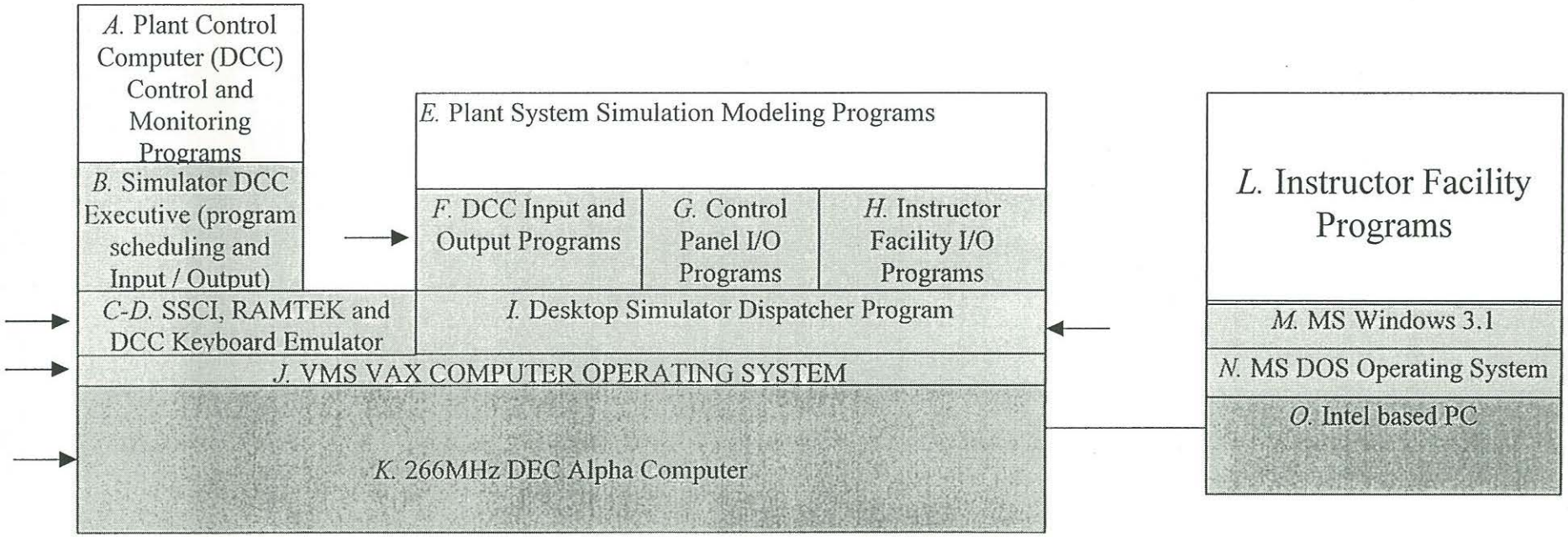


Meaning of Shading:

Hardware (bold)	System Software: programs that do the kind of jobs needed on every computer, deciding when to do things, and input/output operations	Application Programs: the main jobs that the computer was bought to do

Figure 2: Desktop Simulator Architecture

(Hardware cost is $16,000 or less per simulator, speed is up to 1.54 times as fast as real time)



Meaning of Shading:

Hardware (bold)	System Software: programs that do the kind of jobs needed on every computer, deciding when to do things, and input/output operations	Application Programs: the main jobs that the computer was bought to do

→ **Only the components with arrows pointing into them differ from the Training Simulator**

Box J. Both the desktop simulator and the training simulator utilize the VMS operating system. VMS was chosen for the training simulator because of its reliability and superior utility software, despite not being a true real time operating system. VMS was chosen for the desktop simulator to minimize the risks (and potential expense) of porting the software from the training simulator.

The most obscure technical challenge in setting up the desktop simulator, aside from creating the DCC emulator, was adapting the software used for setting up the shared global variables through which all the modeling software and many of the utility programs communicate. Associated with these variables is a whole series of coherently organized files. An off line utility called CDBP creates this family of files so vital to the simulator, and one of the files it creates is an object file that is linked to the modeling.

The alpha and VAX processors in Box K of the two simulators are utterly different, and so the object module format for the two machines is also different. Adapting the CDBP program was therefore a special task. However, the next time this adaptation is needed, it has been decided not to try to cope with the object module format of the target processor, but instead to create the source code for an equivalent FORTRAN block data subprogram, and let the FORTAN compiler handle the object module format details. This is the approach planned for moving the desktop simulator to run under NT.

Going to NT is anticipated to be easy. The motivation for doing so is to reduce the cost of desktop simulator hardware to well under $10,000, and also to reduce the cost of system software purchases such as TCP/IP communications software and compilers.

There has been difficulty in moving custom written real time disk I/O software from the VAX VMS environment to the alpha VMS environment. The adapted software functions, but is not reliable. This software is used by only the backtrack program which periodically stores simulator storepoints to disk during simulation so that the simulation can be brought back to one of these points in time, if desired.

Box K. The underlying computing power of the two simulators is derived from very different processors: a VAX 4105 in one case versus an alpha CPU in the other case. Our fastest alphas are only 266 MHz. For about $10,000 one can now buy alphas that run at 544MHz and which also have hardware that supports 8 and 16 bit data types. This likely offers speed of 3 times or more the rate of real time for a desktop simulator. A significant problem is that the modeling software uses a lot of bytes, whereas the current alpha hardware only works efficiently in 32 bit words. The control panel interfacing and the DCC interfacing on the training simulator both have built in constraints that make it difficult and expensive to think of moving away from using bytes.

Box L. The instructor facility programs of both simulators provide the following identical capabilities, which are exercised by use of a mouse on a large colour graphics display screen. The screen display varies according the functions being used. For many functions, the PC based instructor facility provides only a user interface to actions carried out on the modeling computer.

Freeze/Unfreeze - stop a simulation in progress, and then let simulation resume by unfreezing.

Store/Restore - store the present state of a frozen simulator so it can be recreated again at some future time by the restore operation. There are 28 different storepoints from which to choose.

Malfunction Insertion - the Lepreau training simulator and desktop simulator both have over 8000 malfunctions or instructor controls that allow operation of devices remote from the simulator control room, and which allow the simulator instructor or tester to cause faults in field equipment such as pipe ruptures, bus faults, failures of control logic, and inappropriate device actions.

Malfunctions and instructor control are mostly selected by clicking on device icons in a large family of plant system schematic diagrams. The same schematics report on the current status of numerous plant process values and the devices portrayed on the schematic.

Once the malfunction or instructor control is selected, its manner of use can be varied according to a wide range of options selected from a dialog box. Options include insertion based on a process condition first being satisfied, or based on some time delay, or put in a pending category for later action, or put in place immediately Values of such controls can be selected or often ramped. Included among the modeled malfunctions of the simulator are fuse failures for every wired device and chain of logic that is modeled in the simulator.

Override Insertion - the 5000 I/O points between the simulator DCC and the plant modeling can be overridden to assume values imposed by a simulator user. This is done as with other malfunctions: subject to some condition, delayed or ramped. On the training simulator there are also about 5000 overrides for the control panel I/O, but on the desktop simulator these enable the simulated operation of those panel devices.

Graphical Recorder - enables plotting of process values during simulation. The only values that can be plotted from the instructor facility are those in the modeling common data base.

Trainee Acton Monitor - provides a time stamped list of all control panel actions and instructor control actions. The desktop simulator and training simulator are the same in this respect except for the ability to track DCC keyboard actions. DCC keyboards involve the DUSC on the training simulator, and so it is a small enhancement to add DCC tracking to the desktop simulator.

Lesson Plan - lesson plan is a capability of setting up any of the foregoing functions in a manner, where the lesson plan can be saved and later executed as many times as desired in the same way. The same user interfaces used to invoke the previously described functions are largely used in setting up these actions within a lesson plan, which makes lesson plans easy to prepare.

Most simulator testing and exams are conducted by advance preparation of lesson plans. Lesson plans also play an important part in the delivery if training. The same lesson plan file format is used on the desktop simulator and the training simulator. So lesson plans prepared on one simulator are easily used on the other, with the same results.

Planned enhancements to the lesson plan software include changes to strengthen the way it can help in simulator validation and performance testing activity. This includes the ability to monitor DCC DTABs and the ability to program in DCC keyboard actions.

Backtrack/Replay - the backtrack and replay capability of the desktop simulator is somewhat hobbled at the moment due to the reliability problem with the adaptation of the real time disk I/O routines to the alpha, and due to limitations in the desktop simulator monitoring of DCC keyboard actions. These problems are not seen as major difficulty nor major priority. Nevertheless, there is now a functioning, if somewhat unreliable, backtrack and replay capability.

Performance Monitor - this feature of the instructor facility software is not much used at Point Lepreau. It inserts information into the trainee action monitor record based on the occurrence of pre-specified process conditions. There are no known differences between the desktop simulator and training simulator in this respect.

Database Access - one can examine, monitor or modify any value of any variable in the modeling database. This ability is accessed from a screen button on both the desktop and training simulators.

Most functions of the training simulator used to be accessed from a touch sensitive "keyboard" which was a touch sensitive screen of a special "Emerald" computer linked to the instructor facility

through a serial line. Prior to the creation of the desktop simulator, these emerald touch sensitive keyboard computers were eliminated from the training simulator and replaced by screen mouse activated buttons (actually "windows") at the base of the normal instructor facility display screens. This increased simulator reliability by reducing the number of hardware components, and opened the way for a common user interface between both the desktop simulator and the training simulator.

Hardcopy - any instructor facility screen display is dumped to a printer by a mouse activated "hardcopy" screen button - the same on both simulators.

Box M and Box N. The operating system used on the training simulator instructor facility for years has been a combination of MS DOS and windows 3.1. This was the initial situation for the desktop simulator as well, although the instructor facility demonstrated at the time of presenting this paper is running Windows 95. Neither windows 95 nor Windows 3.1 offer the stability and reliability of Windows NT. Both the training simulator and the desktop simulator and the training simulator are planned to eventually run Windows NT. This contributes to the possibility of running the entire simulator on a single host, thereby achieving a further reduction in costs.

Box O. Both the desktop simulator and training simulator have instructor facilities using Intel based PC's. The training simulator uses two instructor facilities: one at the rear of the simulator suitable for the conduct of exams, and the other movable about the room for the delivery of instruction. The desktop simulator has a single such PC. The PCs used by simulator modelers for normal office functions are used as a desktop simulator instructor facilities, although the demands for a high resolution graphics display make these office PCs more costly than for the normal user at Lepreau.

Boxes P, Q, and R. These boxes are unique to the training simulator, as the desktop simulator has no actual control panel hardware.

OBSTACLES TO REPEATABILITY

Validation of the desktop simulator compared to the training simulator is facilitated if both machines perform in a repeatable manner. To the degree this does not happen, it lengthens the period for a definitive comparative evaluation.

The more thought that is given to the problem of repeatability, the more reasons surface for having a problem. Attainment of 100% accurate repeatability can only be done over time, with a concerted effort on many fronts.

Simulator Fully Restorable

The most fundamental obstacle to simulator repeatability is the excellence of the simulator restore operation prior to the commencement of any test. This is a problem that never goes away because the DCC software and the modeling continue to be updated throughout the life of the simulator, which is always introducing the need to update the definition of what is included in a simulator storepoint.

Any process value which implicitly incorporates history into it, which has escaped inclusion in the simulator storepoint becomes a potential cause of unrepeatable performance. The history prior to each restore operation tends to be somewhat different, and if that history can impact on simulation after the restore operation, it contributes to unrepeatability.

At Lepreau there has been much effort to ensure the storepoints used by modeling are appropriately defined. Perhaps the only tool needed for doing this is a good compiler with the ability to identify local (i.e. non-database variables) which are used before they are computed. The fact that it is not a severe problem to have a good analytical tool is due to the consistent way the modeling software has been written specifically to facilitate simulation.

Finding the definition of DCC storepoints is more difficult. The DCC storepoint software at Point Lepreau is table driven. A DCC storepoint is defined in a table as a series of addresses in the normal memory of the SSCI, and also a series of addresses within the different BMU logical units. When a storepoint is made, a copy is made of the contents of all these addresses. When a storepoint is restored, the values of all of these addresses are restored to a previous state. Because the DCC software is not written to facilitate simulation, there is no automatic way of spotting any error in the derivation of a storepoint. Until recently, the only way to derive a storepoint was by direct inspection of the DCC programs to see where information is written that is used in subsequent processing. For the most part, Lepreau DCC software is clearly organized which facilitates this activity.

Beyond the definition of the data in a DCC storepoint is the question of the state of DCC hardware devices after a restore operation. The countdown registers must be in a known state consistent with when the storepoint was made.

Now that the Lepreau DCC emulator exists, it opens up the possibility of automated logging of all write operations that have storepoint definition implications. This can provide us with proper assurance of the integrity of our storepoint definition table. In the longer term, there is some interest in exploring the possibility of using cheap massive computing power and disk space to remove the need for any kind of analytical activity.

Asynchronous Communication of Control Information

The Lepreau training simulator communicates most DCC I/O asynchronously between modeling and the SSCI DCC using the previously noted DUSC interfacing computer. Because the timing of the DUSC is uncertain relative to the timing of the modeling and the SSCI, what is seen by the programs running on the two machines can vary. In the future, Lepreau hopes to go to a synchronous DUSC to remove this problem. Some delays may be artificially introduced into the YD programs on the desktop simulator to exactly match the timing behaviour of the two simulators.

The DCC itself may be argued to be an asynchronous machine. It is not tightly tied to schedule of modeling calculations in the training simulator, whereas it is in the desktop simulator.

The main consideration in the timing behaviour of the DCC is the timing behaviour of the various countdown registers: in particular the two CDR's which are used to set the timing of the so called fast and slow control programs, and also the CDR for timed DO's. So long as the elapsed time accuracy of these CDR's taken over a period of an hour or so is in good agreement with the timing behaviour of the CPU clock on the modeling computer, then the training simulator behaviour should not differ from the desktop simulator behaviour for reasons of not moving to the same "time drummer". For example, if the slow program CDR takes only 58 minutes to reach an hour's worth of millisecond ticks, then most of the slow control programs will have run at least an extra 120 times more than they should have run in the first hour of simulation. The new DCC emulator offers the possibility of easily and safely exploring the possible consequences of problems of this nature, as for example it effects integral control.

Trigger Each Test at the Same Time

It is desirable to perform all tests using pre-programmed scenarios defined in the lesson plan software. Unfortunately for both the training simulator and desktop simulator, the triggering of the first step of the lesson plan after the initial simulator restore operation is performed manually. A future enhancement is to remove this feature so that the first step of a lesson plan becomes the same as any other step - capable of being triggered by any desired pre-defined process condition - such as waiting 3 minutes after the initial unfreeze.

Other Factors

Progress in achieving greater restorability has been underway at Point Lepreau. Among the actions taken thus far is going to a single CPU VAX 4105 training simulator, as compared to having the former 4 CPU VAX 3800 simulator. This eliminates totally any CPU races in the modeling software, although we ran for years without this being noted as a problem of particular concern.

Another small action taken recently has been an adjustment in the handling of the system service for random numbers, so the seeds of the random numbers have become restorable. The same restorable method of generating pseudo random numbers is used on both the training and desktop simulators. One can readily modify the seeds used in any simulation under instructor control if one does not want repeatability.

VALIDATION OF THE EMULATOR

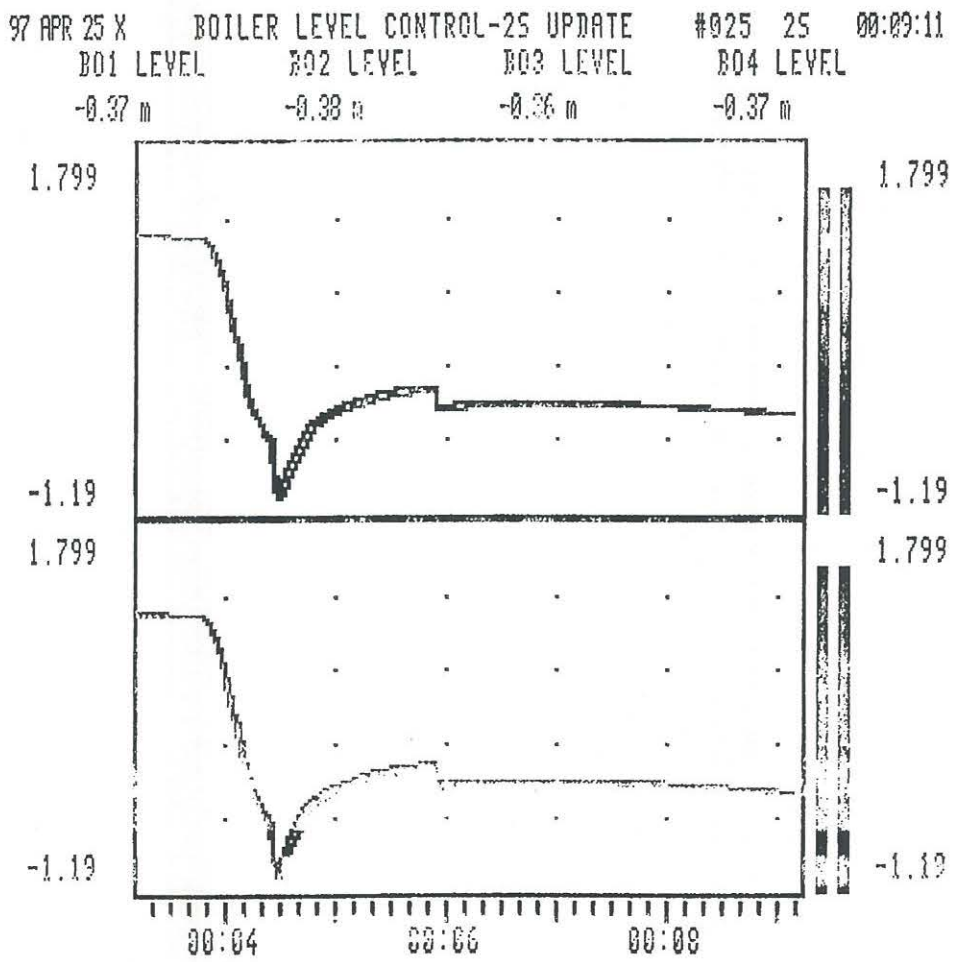
Based on the foregoing factors which impact on the repeatability of simulations, one would not yet expect absolute total agreement between two events run on the desktop simulator and the same events run on the training simulator.

Nevertheless a remarkable degree of agreement is achieved. You have to examine the DCC trends carefully to pick out slight differences. The annunciation listings include the same alarms in about the same order and time delay from the start of the event.

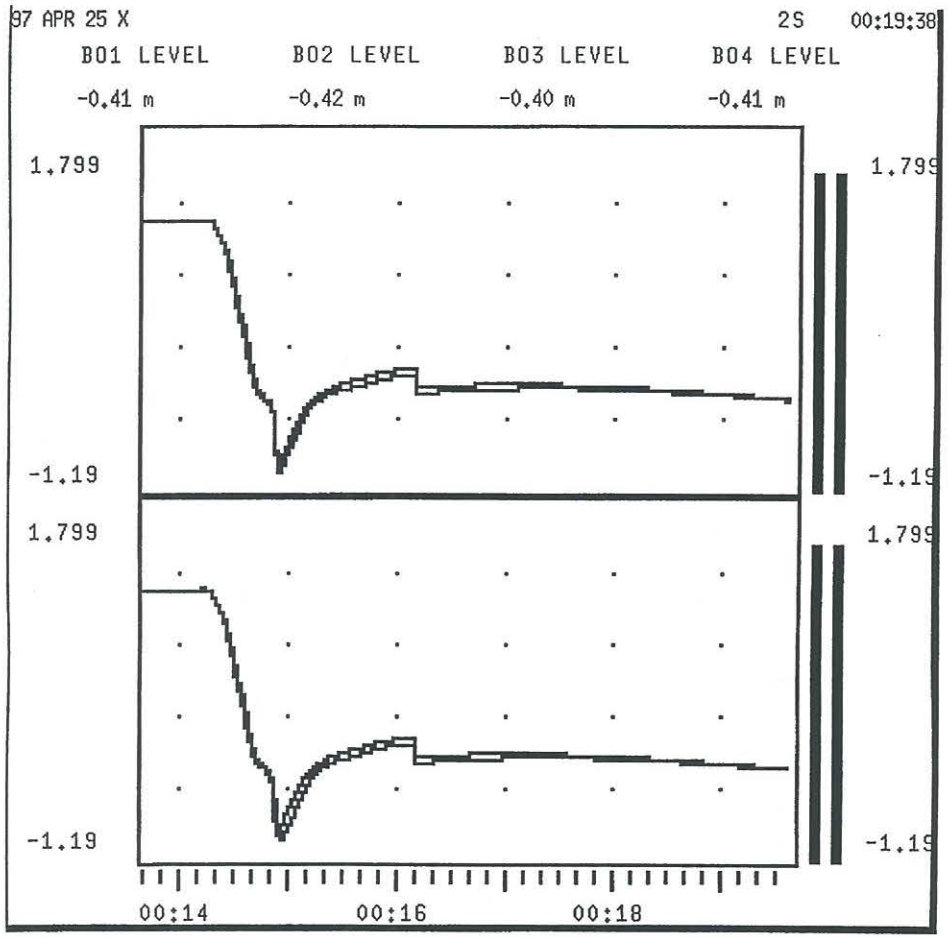
Among the events where simulation results have been compared are loss of class 4 power, manual reactor trip, heat transport pump trip, turbine trip, and large loss of coolant accident. In all cases good results have been observed. The following figures show a comparison of the manual reactor trip response on the two simulators. The graphs speak for themselves.

Comparison of Manual Reactor Trip Response

Training Simulator

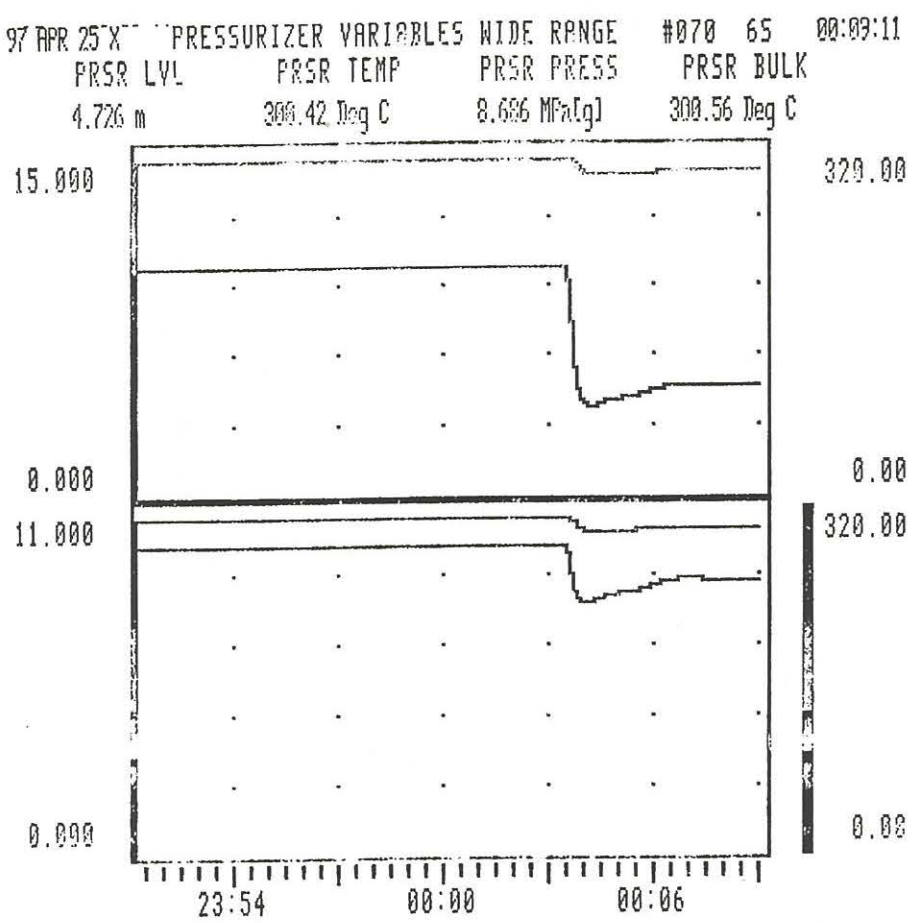


Desktop Simulator

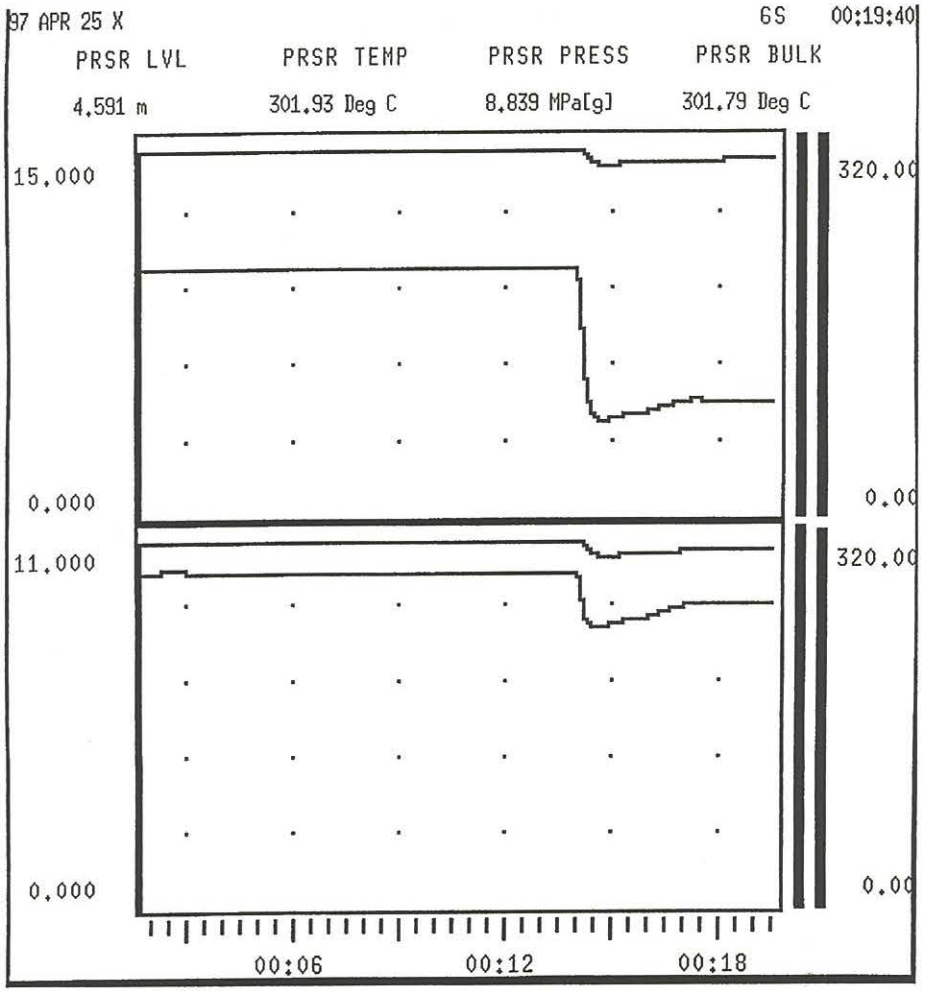


Comparison of Manual Reactor Trip Response (Cont'd)

Training Simulator

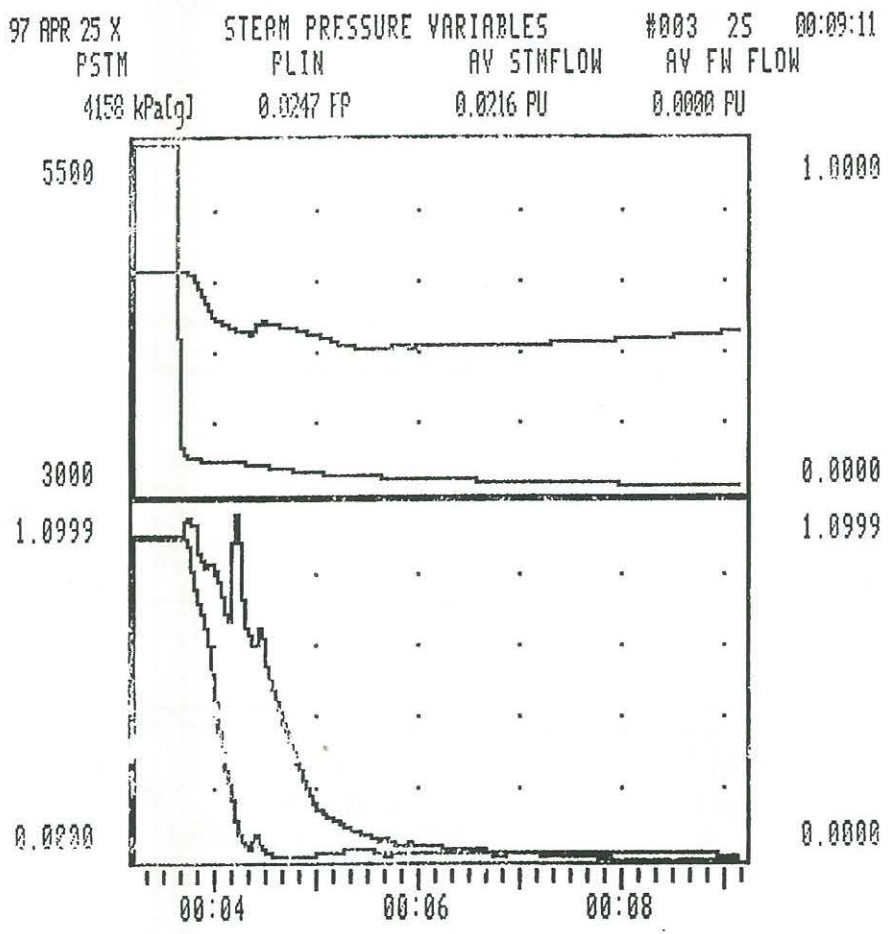


Desktop Simulator

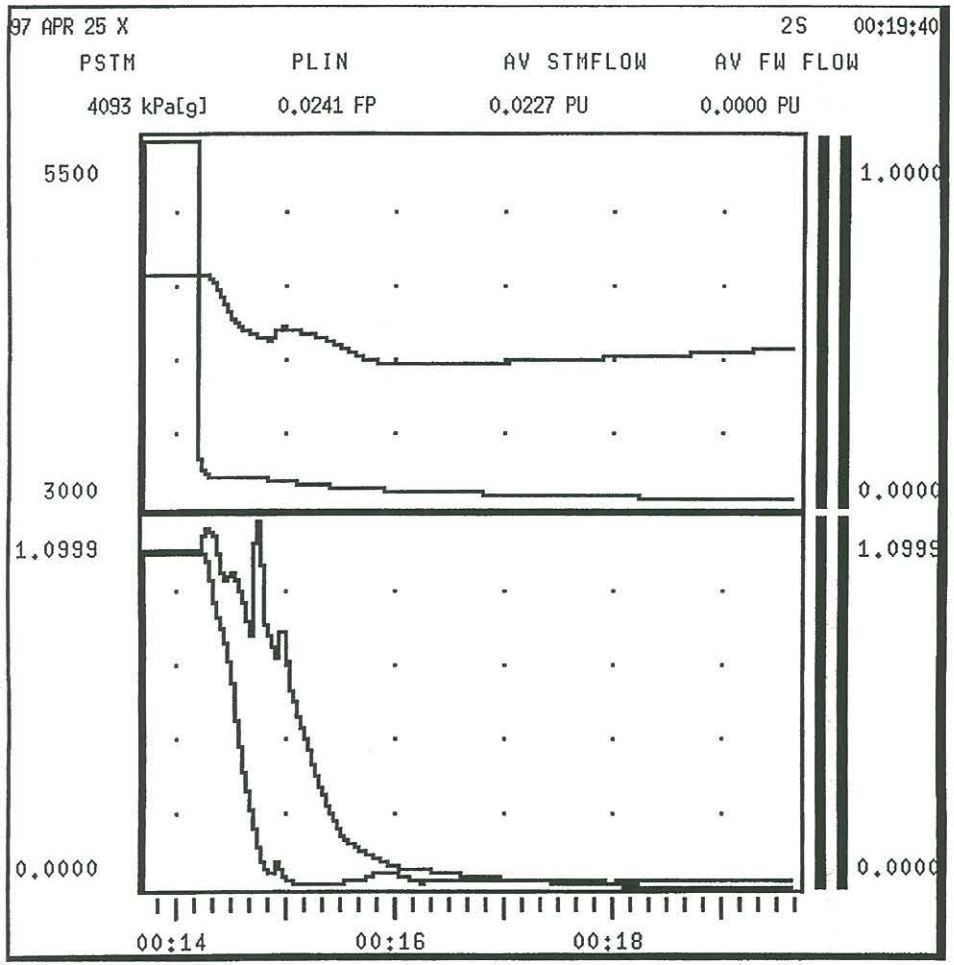


Comparison of Manual Reactor Trip Response (Cont'd)

Training Simulator



Desktop Simulator



ADAPTATION TO WORK WITH NON-REALTIME CODES

The potential exists to create a non-real time desktop simulator of superior fidelity by bringing together the modeling code of the training simulator with non-real time codes originally developed for analytical purposes. There are some challenges in doing this including:

1. the management of the step length in the simulation which is the amount of time by which the modeling is moved forward at each stage of the calculations;

2. setting up practical boundaries between the different sections of the modeling, and arranging for communications across those boundaries; and

3. arranging for initialization and control of the simulation.

Step Length Constraints in the Real Time Modeling

In large scale continuous process simulations such as simulating a power plant, most of the equations are algebraic: for example, the mass equals a new volume times a new density. Every purely algebraic equation is merely an expression of a relationship that must always be true in the modeling, and it therefore says nothing about how the state of the modeling changes dynamically through time.

A relatively small fraction of the equations is involved in dynamically moving the simulation forward in time by numerical integration, differentiation, accounting for pure time delays, or other more exotic calculations. All of these few calculations that move the simulation forward in time implicitly or explicitly take the size of the time step forward into account in the calculation. The size of the time step between successive iterations of the calculation is called the step length, or equivalently, the number of steps per unit time is called the iteration rate. For example, the new mass in a tank may be computed as the old mass plus the new net inflow rate into the tank times the step length over which we say the new net inflow rate has been occurring.

The programming standards used to develop the Lepreau training simulator required every modeler to explicitly use a symbol for step length (or its inverse, the step frequency) in every calculation where simulator dynamics is driven. Thus when the simulator dispatcher calls a modeling module representing a particular portion of the power plant, the dispatcher provides the modeling module with the step length and/or step frequency needed to do the dynamic calculations. This is a sound and practical way to organize the calculations. When this approach is properly applied it allows modeling modules to be easily called by the dispatcher to run at whatever iteration rate is required to ensure adequately stable and accurate dynamic calculations.

In non-real time simulations the size of the step length between iterations is tightened up to be shorter when rapid changes are being simulated, and is then lengthened when things are changing more slowly. The control of the iteration rate or step length is based on an estimate of the errors of approximation in the calculations, and/or stability considerations. This overhead is justified in non-real time calculations because of the added efficiency in the use of computer power when one is able to stretch out the step length in the slower moving parts of any transient. It is assumed that the logic for managing the step length in any non-real time code must be retained even when the code is integrated into the desktop simulator, although the maximum step length may have to be limited at times to satisfy constraints imposed by other parts of the modeling - to be explained.

In real time simulation there is no incentive to manage step length dynamically. Managing step length adds to the complexity of estimating errors and the complexity of retrying those steps forward which

are not assessed to be successful. In real time calculations one is expected to have dedicated computing hardware, and the power of this computing hardware has to be sufficient to get through the most severe transient to be simulated. The step lengths of modules is set so that they are small enough to get through the worst transient, and big enough to fit within the available computing power. Having done that, there is no point in making any further adjustments because the simulation is required to move forward at a constant real time rate, regardless of whether one is in steady state or in the midst of a severe transient.

Because there is not much incentive to play with step length in a real time simulator, there is not much incentive to ensure that the step length is rigorously and properly applied within all the equations. This is a problem that never goes away because it is always potentially affected by changes made in the modeling during simulator maintenance activity. It should therefore not be too much of a surprise to observe that the step length variable has not been adequately incorporated into the modeling of the Lepreau simulator. In a number of instances, the modeler has implicitly used knowledge of the step length to set up dynamic equations where a constant numerical value takes the place of the step length variable. The result is that when one changes the step length of that module in the dispatcher program, the module is called according to a different schedule, but some of the calculations in the module continue to be performed as if the module were on the former schedule. The result is that modeling fidelity is destroyed.

Incorrect use of step length is difficult to find in the 100,000 or more equations of the training simulator, but by running the same modeling in the non-real time dispatcher of the desktop simulator one has the potential to identify where the problems lie. If the non-real time dispatcher of the desktop simulator calls every module with almost 0 step length, the result should be almost zero change on any variable. However, in those equations where step length has been hard coded to a higher value, significant change will continue to be observed. Using this idea one can systematically track down the various errors that have crept in over the years. However, until such an exercise is done, ***one has to assume that all the modeling modules of the training simulator must continue to be run at the same iteration rates as they are required to run in the real time training simulator.***

Interfacing between Modules Developed Apart from Each Other

All the modules of the desktop simulator were developed in a mutually coherent environment where information is communicated between modules through shared FORTRAN Common blocks. The population of these common blocks is about 80.000 variables in the Lepreau simulators. These common blocks are not seen as changing to accommodate any modules "foreign" to the training simulator. Nor is it seen as practical to consider renaming of variables to accommodate modules not used in the training simulator.

Variable step length codes may also have a very large body of variables. It is not seen as practical to expect their nomenclature to be revised for purposes of being integrated with the training simulator modeling.

When a better quality non-real time process model is to be integrated with the better quality control logic and broader scope modeling of the training simulator, it is necessary to study the duplication of calculation between the new process model and the previous training simulator equivalent model. A logical physical boundary[2] needs to be chosen between the two sets of modeling, and the portion of the training simulator modeling inside that boundary is excluded from the new pattern of calculations, and is replaced by the superior non-real time calculations. The control logic is likely the domain of the

[2] Where pipes join vessels may be the most common choice for boundary points.

training simulator as all of the code has been prepared directly from the elementary diagrams of the nuclear plant logic, without any attempt to summarize or simplify.

The interfacing calculations between the remaining training simulator modeling and the new modeling can be a problem to set up depending on the similarity between the organization of the two duplicate sets of calculations formerly used in the two sets of modeling.

What would be nice at these interfaces is simple copying operations between things that directly correspond. The closest one can get to this is probably simple algebraic relationships that do not include equations with dynamics - i.e. no integration, no differentiation, etc.

Furthermore the interfacing should be set up out of algebraic equations that have been solved by sequential substitution, rather than as part of an explicitly simultaneously solved system of equations.

The modules at either end of the boundary are going to be called sequentially one after another by the dispatching program, and so the equations on the interface need to be based on values that are "final" for this moment in time when these values are passed from one module to the next.

If anything algebraic is going to be solved iteratively, it should be done within a single module, and not across the interface with some other module. If anything is going to be solved as part of an explicitly simultaneous system of algebraic equations, it must be done where all the variables that enter into the calculation are within scope of the module that does those calculations, so that all the values used in the calculation are values which are valid.

The requirement that the equations on the interface be algebraic and be solved only by sequential substitution is easily met in the context of the training simulator, because 95% or more of all the equations in the modeling have these characteristics. However, the requirement may not be as easily met for other codes never before integrated with the modeling of the training simulator.

To accomplish the necessary interfacing there need to be two extra modules between the two mutually foreign parts of the modeling.

The pre-interfacing module looks after all the "input" requirements of the variable step length module, ensuring that any values that have to be picked up from the rest of the modeling are copied into a form and place that is accessible to the variable step length modeling.

The post-interfacing module is concerned with copying those values computed by the variable step length module which are used elsewhere in the modeling.

Together, these two copying operations do not take much time relative to the rest of the calculations. They facilitate each section of the modeling keeping its original nomenclature for variables. They can also facilitate the retry operation of the variable step length module.

One of the great advantages of the training simulator as compared to the heavy duty analysis modeling codes is the availability of the instructor facility to make simulation results more visible through plant schematics that depict current process values, and through plotting facilities. The post interfacing module can play an important role in copying results from the non-real time code into a form which is available for display and monitoring on the instructor facility.

Simulation Control

Instructor controls accessed through the instructor facility do nothing unless they can reach into the portion of the plant modeling they are supposed to affect. So any instructor control for equipment modeled by a new non-real time module is either supported by logic within the new module or it won't work. It also must be supported in the copying operation of the pre-interfacing module.

The most onerous aspect of simulation control is simulation initialization. The existing Lepreau training/desktop simulator, including the plant modeling and the DCC, supports up to 28 different possible initializations. It is not expected that any non-real time code will have this capability. Instead the integrated facility would likely have a single state to which it could be initialized that would work approximately as follows:

- the normal simulator restore operation is performed on the training simulator database using the program XRV which is set up for this purpose;

- the slightly modified XRV also makes a new call to the new non-real time module which causes it to be initialized according to whatever mechanism is already designed into the non-real time code.

The XRV program is invoked by the restore command from the desktop simulator user interface, where the storepoint chosen for the training simulator modules is the one which corresponds to the initialization state created by the initialization process associated with the non-real time code.

Dispatching Considerations and a Possible Algorithm

The training simulator modeling works adequately when each module is given its input values computed to be up to date exactly when that module runs. So the dispatching arrangement respects the former dispatching schedule such that inputs to all retained modules get computed up to the points in time corresponding to the old schedule. If a new variable step length module has the possibility to step beyond the normal time for an input to an old module, the dispatcher constrains the new module so it steps only up to the time from which the old module is accustomed to getting its inputs. Similarly, the old modules are constrained only to provide their outputs at the times to which they stepped in the old schedule. Therefore any new module using values from the old modules must be content only to have those values updated according to the old schedule - that is the limit of the capability of the old module, and these timing characteristics are built into the way the dispatcher calls the module.

If a new variable step length module finds it has tried to step further forward in time than it can properly do, it must let the dispatcher know that a retry is required with a shorter step length. When the step length is judged to need to be lengthened, there is no requirement for a retry, but the dispatcher must nevertheless be informed of the requested longer step length.

From the point of view of the DCC hardware emulator, one has no reasonable means to approximate the running of the computing hardware over a given period of time. Every single machine instruction that would be executed during a given period of time must be executed. So the dispatching program calls the DCC hardware emulator for a time step appropriate to the needs of the plant process modeling. The emulator then runs every machine instruction that would be executed by the DCC during that period, adjusting its outputs according to whatever may arise from those calculations, and reading its inputs according to whatever the state of the plant modeling was at the start of the DCC emulator running. This is what has been done with success in the existing desktop simulator, and it

continues to be the approach that would be taken for simulations that may involve some variability of step length for some of the modeling modules. The only difference is that the DCC emulator may be required to run for a shorter period of time, if that is what the dispatcher program says is required based on what it knows of the various modeling modules.

When the emulator is run together with a variable step length integration method it would seem more efficient to let the variable step length module run first so that it can find the step length it wants. After it has stepped forward in time successfully, the DCC emulator could then be called to run for a similar period of time[3]. In general, the variable step length module is expected to be run from one 50 millisecond leg boundary of the training simulator modeling exactly up to the next such boundary, taking as many or as few steps as its integration method desires. After each such step, the DCC emulator can be brought forward up to the time so far reached by the variable step length module. When the 50 millisecond leg boundary is reached for the training simulator, then the normal dispatching sequence for simulator modeling modules in that leg would be run. Then the variable step length module is run and the whole pattern repeats.

A possible dispatching algorithm for the case where there is only one variable step length module is as follows.

A. Compute the delay until the scheduled start the next 50millisecond leg of the training simulator modeling.

B. Compare delay time in A with the step length currently being requested by the variable step length modeling, and choose whichever time is shorter as the next step length.

C. Call the pre-interfacing module of the variable step length module.

D. Call the variable step length module.

E. Examine the information returned by the variable step length module, and if it is required to be called again with a shorter step length return to either step C or step D, according to the known interfacing characteristics of the module. Otherwise, continue to step F.

F. Call the post-interfacing module of the variable step length module including a call to the DCC emulator to bring it up to the time achieved by the variable step length module.

G. If the time just reached is on the boundary of a 50 millisecond modeling leg of the training simulator computing schedule, then call all the modules in that leg which remain in scope of simulation.

H. Return to step A and repeat.

Using the foregoing approach, or something similar, it is hoped that the desktop simulator can evolve to becoming as high a fidelity non-real time modeling platform as Point Lepreau finds it useful to become. Extending the existing dispatching mechanism of the desktop simulator as suggested also provides a benefit in bringing the non-real time code under instructor control for purposes of freezing and unfreezing the simulation, and controlling simulation subject to features of the instructor facility.

---

[3] Using the ability of the dcc software to make dcc storepoints and to restore these storepoints one can do retries on the dcc, but it is more efficient to organize the dispatching so this is unnecessary.

REFERENCES

Refer to the following companion paper for a more complete description of the desktop simulator of which the Point Lepreau DCC emulator is a component.

(1) M. MacLean, et al, New Brunswick Power, "SSCI 125 Emulation at Point Lepreau G.S.", 18th Canadian Nuclear Society Annual Conference, Toronto, June 8-11, 1997.