2017 International Nuclear Atlantic Conference - INAC 2017
Belo Horizonte, MG, Brazil, October 22-27, 2017
ASSOCIAÇÃO BRASILEIRA DE ENERGIA NUCLEAR – ABEN

Agência Brasileira do ISBN
ISBN 978-85-99141-07-6
9 788599 141076

DEVELOPMENT OF A NEW GRAPHICAL INTERFACE FOR THE LABIHS SIMULATOR USING LABVIEW

Silas C. Augusto[1], Guilherme D. Jaime[2], Marcos S. Farias[3]

Instituto de Engenharia Nuclear, CNEN/RJ
Universidade do Estado do Rio de Janeiro
Rua Hélio de Almeida, 75
21941-906 Rio de Janeiro, RJ
[1] silas@ien.gov.br
[2] gdjaime@ien.gov.br
[3] msantana@ien.gov.br

ABSTRACT

The LABIHS (Human-System Interfaces Laboratory) compact nuclear power plant (NPP) simulator is, since 2002, operating at the Nuclear Engineering Institute (IEN), Brazil. Due to the processing power required by the simulator software and the hardware available at the time, the simulator was developed under a PA-RISC architecture server (HPC3700), using the HP-UX operating system. All mathematical modeling components were written using the HP Fortran-77 programming language with a shared memory to exchange data from/to all simulator modules. In 2008, this hardware/software framework was discontinued, with customer support ceasing in 2013, what makes it difficult to maintain and expand. On the other hand, technological progress during this period enabled cheaper and more accessible computers, such as the PC (Personal Computer) architecture, to have enough processing power to run the simulator framework. It turns out that the PA-RISC computer architecture is incompatible with the PC one. Thus, our work group has been working, for some years, to port the simulator to the PC architecture. Part of this effort includes completely rebuilding all the MMI (Man-Machine Interface) using modern software tools and programming languages which are compatible with the PC architecture. In this work, we present two important steps for the NPP simulator migration from the obsolete architecture to the PC one. Firstly, we provide a new inter-process communication library which allows the data exchange between the NPP simulator currently running at the old PA-RISC architecture and other software running on modern PCs. This step is important because it allows us to work on the new Graphical Interface while the older NPP simulator is still operating. Secondly, we present the all-new MMI for the LABIHS simulator, currently under development using LabVIEW software suit. We then provide a comparison between the original and the proposed MMI development process.

1. INTRODUCTION

On industrial plants, operators interact with human-system interfaces (HSI's) to perform their tasks and functions, thus making the HSI's an important part of an industrial plant. They include information displays, controls, and alarms. All HSI is composed of hardware and software and is characterized regarding its physical significance and functional characteristics. The NUREG-0700 [1] reviews the physical and functional characteristics of HSI's. The design aspects of human factors engineering (HFE) of HSI's are covered by NUREG-0800 [2] and NUREG-0711 [3].

Based on the literature, interviews, and visits to industrial plants, O `Hara et al. [4] identified challenges in the technology of human-system interfaces and its potential effects on the performance of people. The topics were evaluated considering their impact on plant safety [5]. Concerning human-system interfaces of plants it was found that with the rapid development of computer technology, more and more human-computer interface systems have been introduced in the conventional man-machine systems. Currently, for example, some computers may represent a large control room with numerous gauges, buttons, and actuators of a nuclear power plant or a system of aid to the pilot of an

aircraft cabin. Gradually, MMI (Man Machine Interface) systems with conventional meters and actuators connected by numerous wires and panels are being replaced by systems with a few displays. This means that currently MMIs can be considered as human-computer interfaces and digital interfaces.

Besides, the control systems of industrial plants are becoming more elaborate and, consequently, what makes the design of these interfaces an increasingly complex task. When a system of a plant becomes more complicated, it becomes difficult to keep track of its dynamics. As a result, one can face difficulties while monitoring/controlling the system.

As in any complex system design, it is highly desirable to use a theory/methodology for the design of interfaces. The methods that guide the design of an interface must consider questions such as What, How and When any information or input control should be shown.

In recent years, to help to answer these puzzles, several techniques of building interfaces have appeared in the literature, among which we mention: the task-oriented interface [6]; ecological interfaces [7], and function-oriented interfaces [8].

The primary objective of this paper is to document and briefly present the development of two primordial components for the modernization of the LABIHS NPP Compact simulator.

We provide an Inter-Process Communication Module (ICM) which allows the data exchange between the NPP simulator currently running at the old PA-RISC architecture and other software running on modern PCs. This is important because it makes it possible for us to work on the new Graphical Interface while the older NPP simulator is still operating.

We, then, present the all-new MMI for the LABIHS simulator, currently under development using LabVIEW software suit. Finally, provide a comparison between the original MMI and the new one.

The remainder of this paper is organized as follows. Section 2 presents an overview of the LABIHS simulator. In Section 3 we describe the ICM. The MMI development process on the LabVIEW software tool is discussed in Section 4. In Section 5 we discuss differences and similarities between the LabVIEW software suite and the tool used to build Graphical Interfaces and the original NPP Simulator: the iLog Studio. Finally, Section 6 brings some conclusions and comments on the current stage of the MMI development for the LABIHS simulator.

2. OVERVIEW OF THE HUMAN-SYSTEM INTERFACE LABORATORY

2.1. LABIHS Objectives

The LABIHS laboratory has the following objectives:

- providing technical expertise in the design of MMIs;
- design systems to aid the operator;
- carry out modernization of control rooms;
- evaluate control rooms and interfaces considering aspects of ergonomics and human factors;
- analyze the interaction between operators and the various systems operated by them; and
- assess the reliability of human operators in simulated accident scenarios and regular operation.

2.2. LABIHS Advanced Control Room

A control room contains the systems and operation manuals needed to monitor and manipulate the operating conditions of an industrial plant, so to ensure that its service and shutdown processes are reliable and safe either under normal circumstances and during accidents [9].

Advanced control rooms of industrial plants consist of an array of systems and equipment, where operators can monitor, control and intervene in the process through various MMIs as well as monitoring stations. These interfaces have significant implications for the safety of the plant, since they affect the activity of operators, as it defines how operators receive information regarding the status of critical systems and determine the requirements necessary for operators to understand and supervise the system.

The primary task of an operator is to keep the plant in service under acceptable conditions of safety and efficiency. The actions taken are supported by procedures which encompass features such as: starting and stopping the plant, emergency situations, alarm systems, communication systems, control systems, security systems and fault diagnosis. Operators also interact among themselves and with their support teams, such as the maintenance, testing, and planning ones.

2.3. LABIHS Simulator

The NPP simulator is a tool that simulates the process of generating electricity from nuclear fission, as well as all the main features of the systems that make up the nuclear plant. This is accomplished by using mathematical models to analyze the dynamic behavior of these systems. NPP Simulators are used as support for regulatory agencies; training and qualification of operators; and security analysis and validation of operating procedures. The simulator shall transmit a realistic view of the NPP, providing the illusion that they are working at a real one. To achieve this, it is important to both show the operation status of key systems, and enable one to experience actions they would perform in an actual control room.

The LABIHS simulator consists of a set of equipment and computer programs that simulate the processes of the primary, secondary and tertiary systems of a PWR (Pressurized Water Reactor) NPP. The simulated power output is of 930 MWe. It also includes a compact a control room with several advanced MMIs representing the various systems, as well as an Instructor Station. Figure 1 shows the main components of LABIHS.

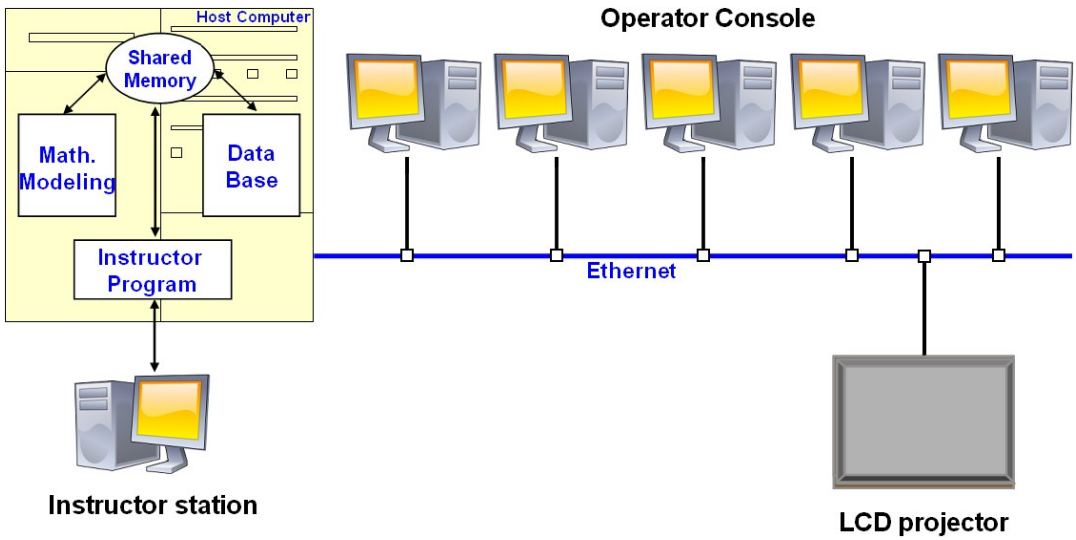


Figure 1. Basic components of LABIHS.

The simulator's operation crew consists of three operators: the one responsible for reactor core and the primary systems, the one that supervises the turbine and the secondary systems, and the supervisor. Each one of the two (non-supervisor) operators controls and monitors the systems under its responsibility (primary, secondary) through four LCD computer displays, each. In front of them are three 52" LCDs that show (1) all alarms from the alarms screens, (2) the overall operation of the nuclear plant, and (3) additional information as the insertion level of the control and shutdown bars on the reactor core and the pressure/temperature curve. These screens provide the operators with an integrated view of the reactor operation. Figure 2 presents an overview of the current LABIHS control room.

Figure 2. Current LABIHS advanced control room.

There is also a separate room within the control room where the instructor controls the LABIHS simulator software, which runs on an HP workstation running HP-UX (PA-RISC Architecture). Figure 3 shows the main screen used by the instructor.

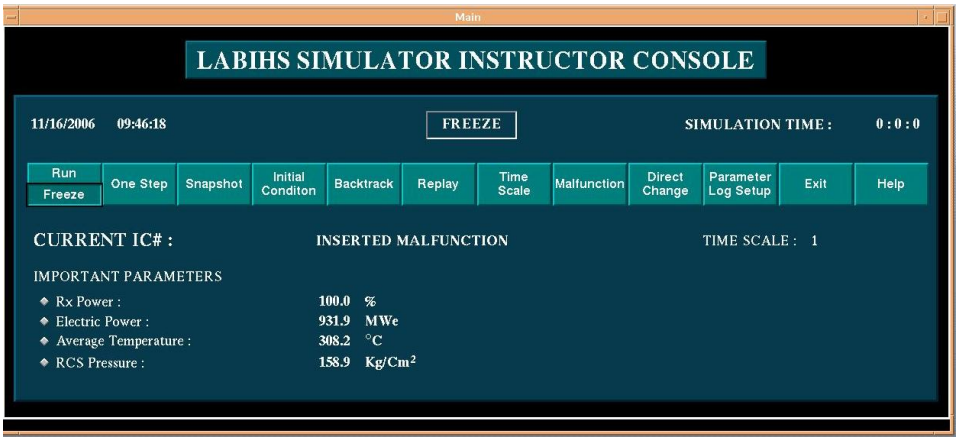


Figure 3. Instructor Console main screen.

The original simulator interface software (henceforth called MMI software, from Man-Machine Interface) also runs on the same HP workstation. Operators use the X-manager remote desktop software to remotely view the MMI software which runs locally on the HP workstation. A shared memory system provides the inter-process communication among the core simulator processes and the MMI processes. There is also a scheduler process coordinating the access to the shared memory and, thus, providing consistency between the simulation and all running instances of the MMIs (MMI processes).

3. THE ICM – INTERPROCESS COMMUNICATION MODULE

3.1. The Simulator Core: Overview

Broadly, the PWR simulator software is composed of a set of applications that communicates with each other, as shown in Figure 4.

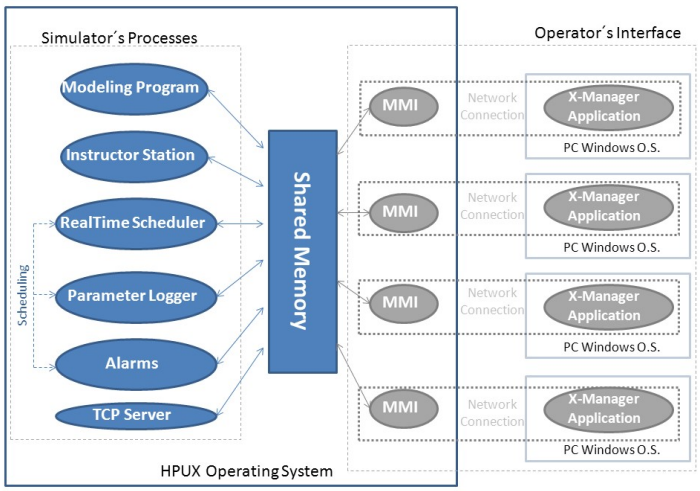


Figure 4: Current simulator's architecture overview.

The Modeling Program is the main component of the simulator. It contains all mathematical models that represent the most relevant characteristics of a pressurized water reactor power plant.

The Instruction Station module allows that an instructor interacts with the ongoing simulation to provide commands according to which scenario the instructor wants to make available to the operators.

Three other modules provide logs and reports:

- RealTime Scheduler is used to set timers and schedule tasks for both Parameter Logger and Alarm List modules;
- the Parameter Logger module stores, into files, a set of variables describing states through which the simulation has gone in the past;
- The Alarm List module provides alarms that guide the operators during the power plant operating procedures.

All communication among the presented modules is provided using a shared memory [14]. Shared memory is a RAM (memory) region in a computer system in which several running read and write data. Due to its inherent simplicity, a shared memory can only be accessed by local processes.
Because of this, the MMI (Man Machine Interface), which is the Graphical Interface for the Simulator, must run locally on the very same host (eg., HP UX Operating System, in Figure 4) as all

other modules/processes of the simulator. Since several persons generally operate the plant, some MMI processes also need to be run locally.

To accomplish this, each operator uses a PC running Windows operating system to run the X-Manager [15] application. This application allows the operator to remotely access the machine running the simulator, and start an instance of the MMI. Since MMI is running locally, it can write to or read from the shared memory, allowing the operator to read the current simulator state as well as to provide commands to it.

By referring to Figure 4, one can note that without the shared memory, different modules of the simulator would not communicate to each other, and the simulator would not work.

To allow the proposed MMI to access the simulator´s shared memory, we conceived a new module: the ICM (Inter-Process Communication Module, shown in Figure 5.

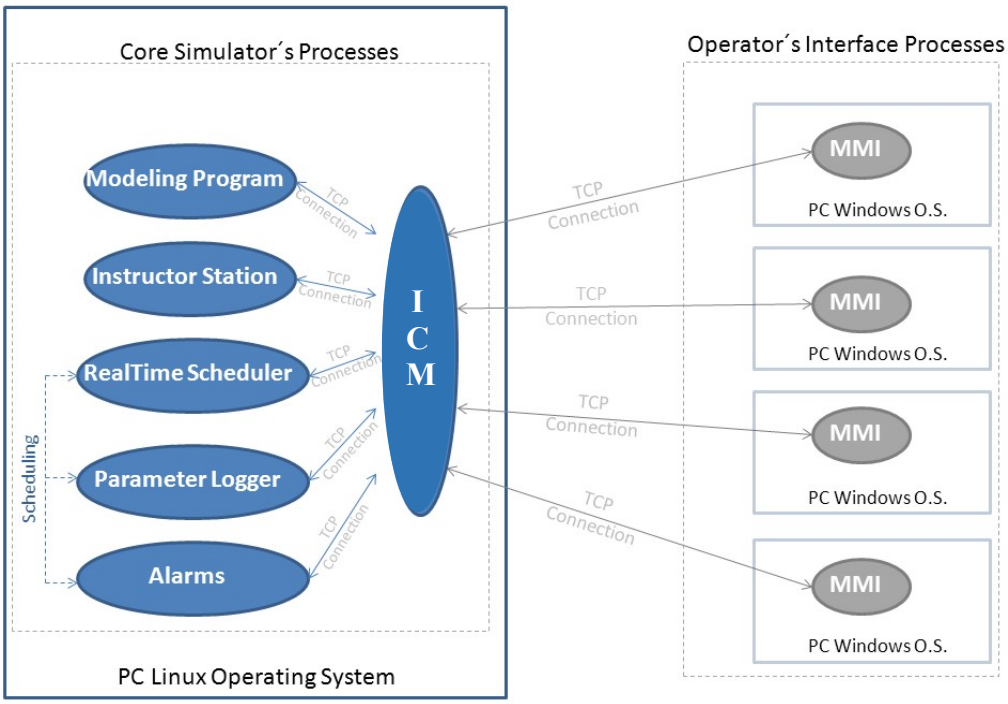


Figure 5: ICM providing communication between core simulator processes and remote MMIs.

This module can serve several network sessions at the same time. More details on the methods used to implement this concurrent network server are found in Section 3.2.

By referring to Figure 4, one can note that without the shared memory, different modules of the simulator would not communicate to each other, and the simulator would not work.

Through the ICM, all simulator´s modules, i.e., the Modeling Program, the Instructor Station, the Real-time Scheduler, the Parameter Logger, the Alarms and the MMI, will communicate to each other. Thus, this module will also be useful in the future, to the migration of the core simulator modules to the PC architecture.

Among all well-known inter-process communications implementation methods [14] (eg., Shared Memories, Pipes, Messages Passing, Remote Procedure Calls - RPC, Sockets), only two of them were conceived to provide network communications capability: RPC and Sockets. Between them, we opted for the Sockets choice since it the *de facto* solution used by the Internet. Therefore, the PWR simulator becomes a distributed system, and there are virtually no bounds† on where (different hosts) each module one can run.

The ICM module needs to be portable, just as all other modules of the simulator. This way, the C programming language [16] is being used to implement it, and the module is being carefully crafted to comply with the POSIX standard [14,17]. POSIX stands for Portable Operating System Interface for Unix and is the name of a family of IEEE standards to define the API, for software compatible with variants of the Unix operating system. These choices are important, as the ICM module needs to consistently compile and run in both computer environments (i.e., PA-RISC HP UX and x86 Linux), especially during the test phase of this project.

In fact, sometimes it is not enough to follow a programming standard such as the POSIX. Figure 6 shows an example of a piece of C language code. Even conforming with the POSIX standard, some platform-specific code needs to be written if one wants to compile it in both PA-RISC and Linux operating system. We carefully treated this type of situation when writing the ICM.

```
#ifdef LINUX
        if( socket_conectado < 0 )
#endif
#ifdef HPUX
        if( *socket_conectado < 0 )
#endif
```

Figure 6: Following POSIX standard sometimes is not enough, and there is a need for writing specific code to assure portability

3.2 Providing Scalability and improving efficiency

The choice of using the Sockets API to provide inter-process communication on the new version of the simulator is also used to solve the scalability issue.

There are two main features required for the simulator software to be considered scalable, according to the MMI modernization project requirement:

- the possibility to the simulator to run several instances at the same time, and
- Each instance of the simulator must be able to serve several MMI sessions at the same time.

The first feature is easy to achieve, once the ICM is based on the Socket API. When one used sockets for enabling inter-process communication, he needs to provide the port number to which the software will listen for connection requests. It is, indeed, straightforward to write a network server software that accepts the port number as a user parameter, as shown in Figure 7.

```
gdjaime@linux$ ./ICM 5050
```

Figure 7: User starts the ICM module, and commands it to listen to port number 5050

Thus, if there is a need to start another instance of the ICM, the user needs only to start another terminal a run the module again, providing a different port number. Figure 8 illustrates the execution of two instances of the simulator at the same time, one is using port number 5050, and the other one is using port number 5051.

† As the PWR Simulator is a real-time software, the communication delay between modules must remain strictly under a certain value (i.e. 100ms)

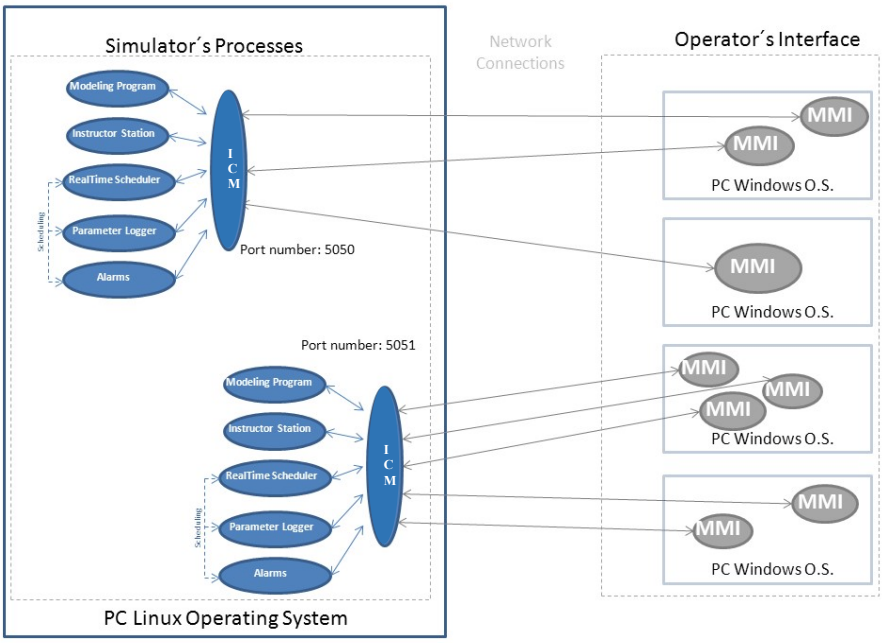


Figure 8: Two instances of the simulator running at the same time, MMI remotely running at operators´ workstations. Ports 5050 and 5051 are being used.

The other required feature, namely, the ability of each ICM to communicate to several MMI concurrently, is more complex to provide. Two POSIX standards options are currently available for implementing concurrent service at a network server software: processes (fork) [14] and threads [17].

Both are advanced programming techniques, but the main difference between them is that when a process calls a fork, the operating system creates a copy of the process and now two processes are running in *different areas of the computer memory*: the parent and the child. Thus, any communication between them needs to be explicitly implemented as if they were to strange processes. For example, a shared memory could be implemented so both processes can communicate, or they could open a socket to exchange data. Figure 9 (b) shows an example with three processes created by using the fork system call. Note that each process has its data (variables) memory area.

On the other side, as shown in Figure 9 (a), if a process uses threads to create another execution instance, the original process continues to run, and the operating system creates a thread *into the same memory space of the process*. Then, if the thread writes a value to a global variable, the main thread will be able to read it. In other words, the communication between them is trivial.

Refer to Figure 5, in which a single ICM is communicating to several processes: all simulator´s core modules to the left and four instances of the MMI to the right. If an operator sitting on any of the operator´s workstation interacts with the simulator, all other modules need (including other three MMIs) to be aware of this.

Suppose that there is one thread of the ICM running to serve each module currently active. As all threads share the same memory area, if any change is made to a global variable, all other threads will be aware of this. It is clear, then, that the threads solution fits just right as the programming technique.

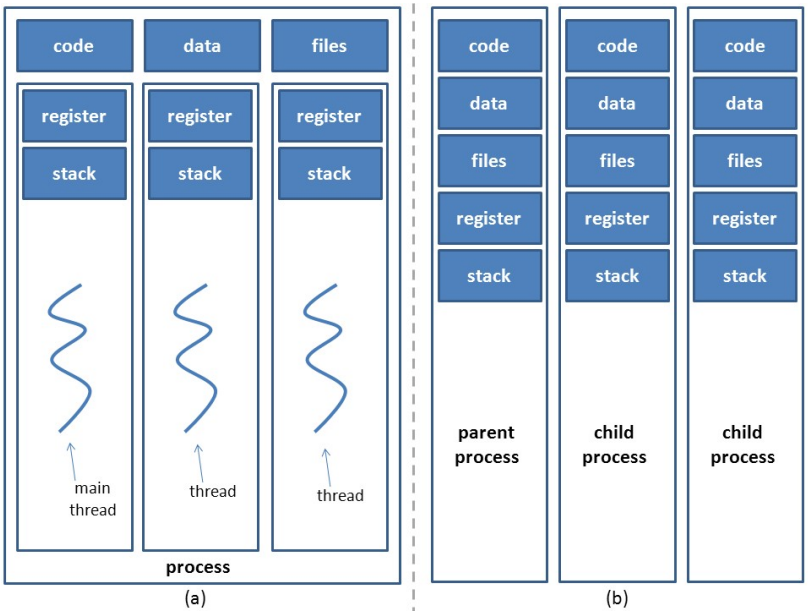


Figure 9: (a) Concurrent threads always share its data memory area; and (b) when a *fork* is called it created a child process, and each child will have its own (separated) data memory area.

One may argue that, by using threads to serve several network connections, there is a need to implement some kind of access control at the global variables, so that only one operation is executed at a time, to avoid inconsistency. To avoid this kind of inconsistency, the ICM uses the Mutex API [14] to provide the required access synchronization to global variables.

Besides the fact that threads fit to what is the ICM when comparing processes and threads, there are efficiency characteristics that make threads the best choice. We highlight the most important below:

- threads have direct access to the data segment of its process;
- threads can directly communicate with other threads of its process;
- threads have almost no overhead, while processes (fork) have considerable overhead;
- it is easyer to create threads than processes.

Another important efficiency issue regarding the MMI modernization is related to the hardware incompatibility in the Byte Order. To deal with this, all communication between the ICM and other modules will rely on the External Data Representation – XDR standard [17]. In short, this standard provides for hardware independent data representation.

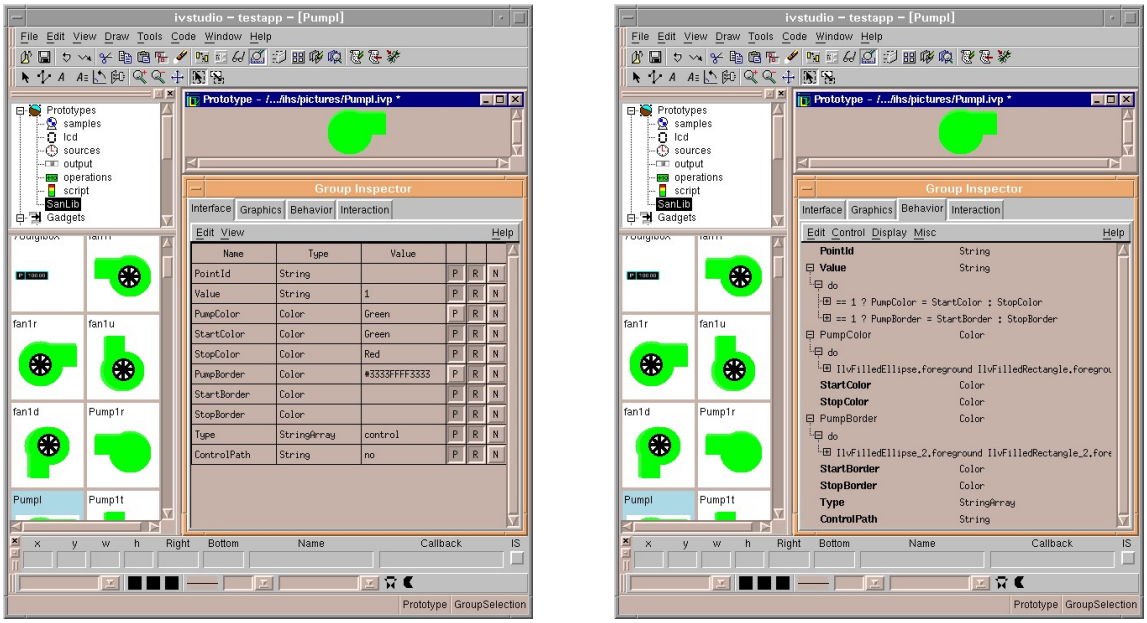
4. LABIHS SIMULATOR NEW INTERFACE DEVELOPED IN LABVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a system-design platform and development environment for a visual programming language from National Instruments. LabVIEW is commonly used for data acquisition, instrument control, and industrial automation on a variety of operating systems, including Microsoft Windows. [10]

The scope of this article is the development of a new MMI for the LABIHS simulator using LabVIEW. For achieve that purpose, the LabVIEW DSC module was used, and the ICM presented in Section 3 supports the communication between the new MMI and the NPP simulator. The following sub-sections describe the MMI development on both the original (old, using legacy tools) and on the new one (using LabVIEW).

4.1. Component

On the original MMI (i.e., the one for HP-UX), the software iLog Views Studio is the software used to develop new screen components. The user needs to draw the component and to define its properties and behaviors. Figure 10 shows the iLog Views Studio editing the pump element.



(a) (b)

Figure 10. iLog Views Studio editing the pump (a) Interface tab, defines component attributes; (b) Behavior tab.

On LabVIEW, the DSC module was used. It provides a ready-for-use component library for valves, pumps, and tanks, among others. Figure 11 shows the LabVIEW Controls Pallete with the DSC module expanded and the graphics of a 2D valve and 2D pump from the DSC module.

(a) (b)

Figure 11. (a) LabVIEW Controls Pallete with DSC module expanded; (b) 2D valve (left) and 2D pump (right) graphics from the DSC module.

4.2. Screen

On the original MMI, the HSI Builder software is used to create and modify screens. Each screen is saved as a separate ILV file. Components are organized in tabs on the windows' left corner. A component can be dragged and dropped into a new or existing screen to create a new instance of it, and have its properties' values altered. Especially, the POINTID property's value associates the component's instance to a value on the simulator's shared memory. When a screen is loaded the MMI software binds itself to the simulator's shared memory, so that it can read or modify simulation variables. When a value is changed, all other MMI instances will be able to see it, as they are also attached to the shared memory. Figure 12 shows the HSI Builder window editing the RCS.ILV screen.

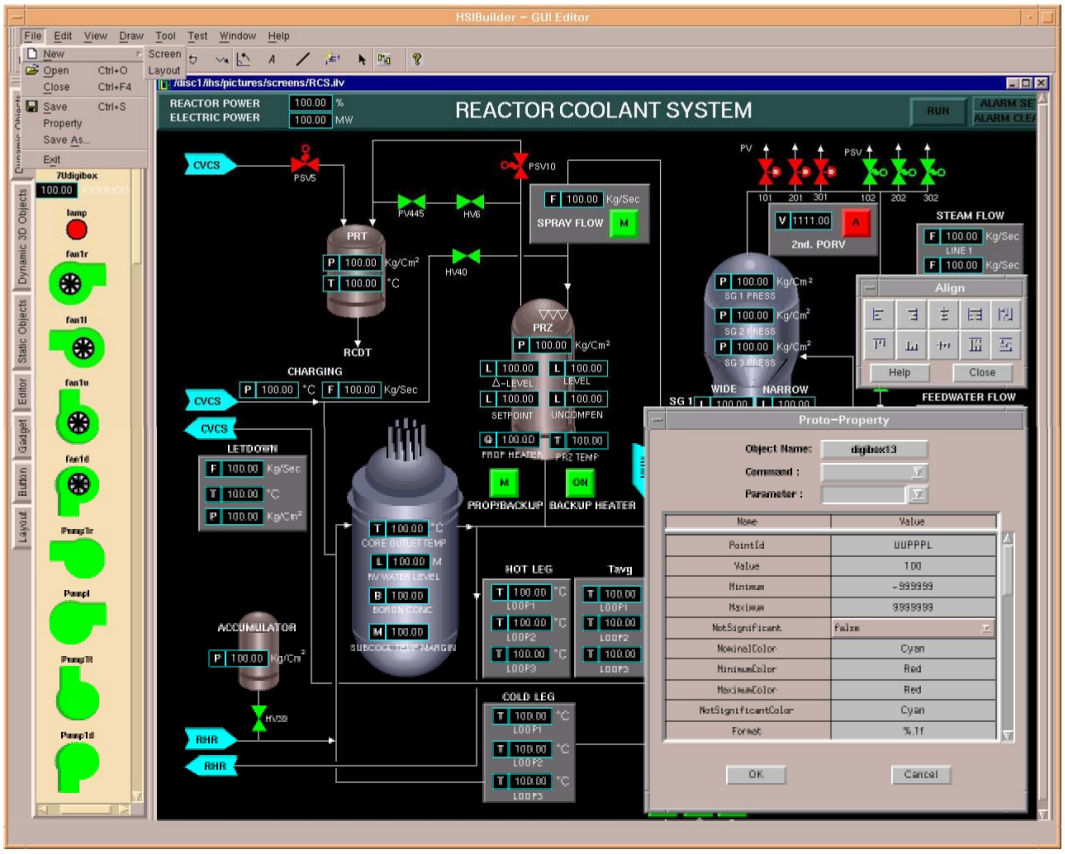
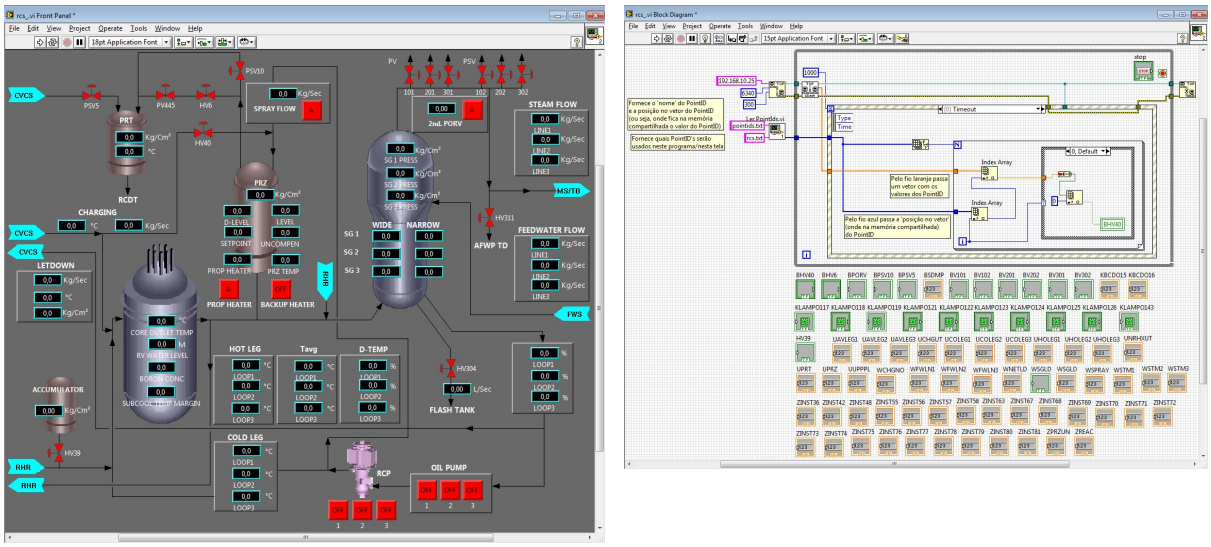


Figure 12. HSI Builder window editing RCS.ILV screen.

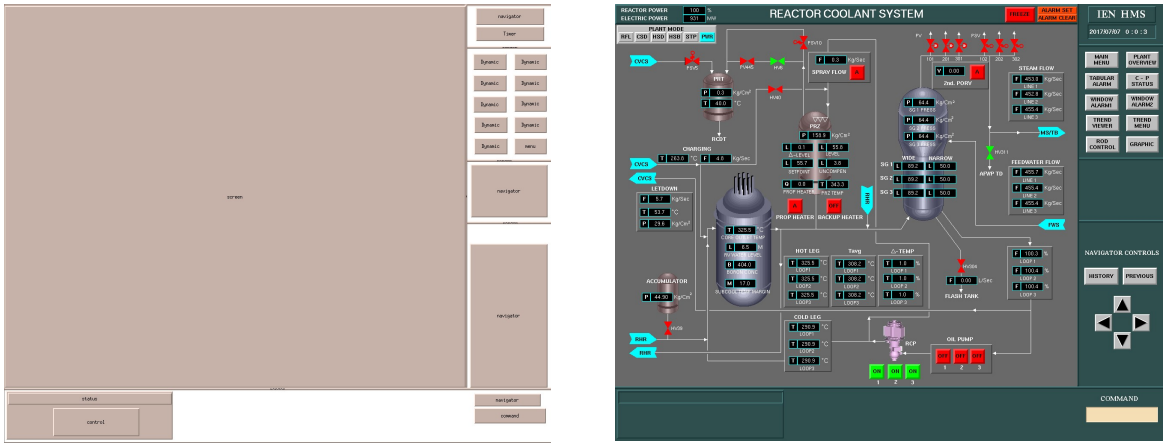
On LabVIEW, each screen is saved in a VI (Virtual Instrument) file. Each VI is composed of (1) a Frontal Panel, which holds the screen's graphical user interface, and (2) a Block Diagram, which contains the screen's programming logic. Components are organized on the floating window called Controls Pallet. A component can be dragged and dropped into a new or existing screen to create a new instance of it, and have its properties' values altered. The corresponding component's icon on the Block Diagram is then integrated properly on the screen's programming logic so that the component can function correctly. Figure 13 shows the LabVIEW software editing the RCS.VI screen, showing its Frontal Panel and Block Diagram.



(a) (b)

Figure 13. Frontal Panel (a) and Block Diagram (b) of the Reactor Coolant System screen.

On the original simulator MMI, the HSI Builder software is also used to create and modify the MMI's layout. The layout is saved as a LAYOUT file. The default LAYOUT file defines a screen area and right/bottom borders. On the right edge of the display, there are buttons used to switch between screens. Figure 14 shows the layout used on the original simulator MMI and a screen built on this layout.



(a) (b)

Figure 14. Original simulator MMI layout: (a) layout; (b) screen built on the layout.

On the new MMI, to change the current screen, two LabVIEW components were used. The first one replaces the screen's title bar and functions as a drop-down list, where the operator chooses the screen to be shown. The second one loads the chosen screen on a subpanel. Figure 15 shows the new MMI made on LabVIEW with the Reactor Coolant System screen loaded.

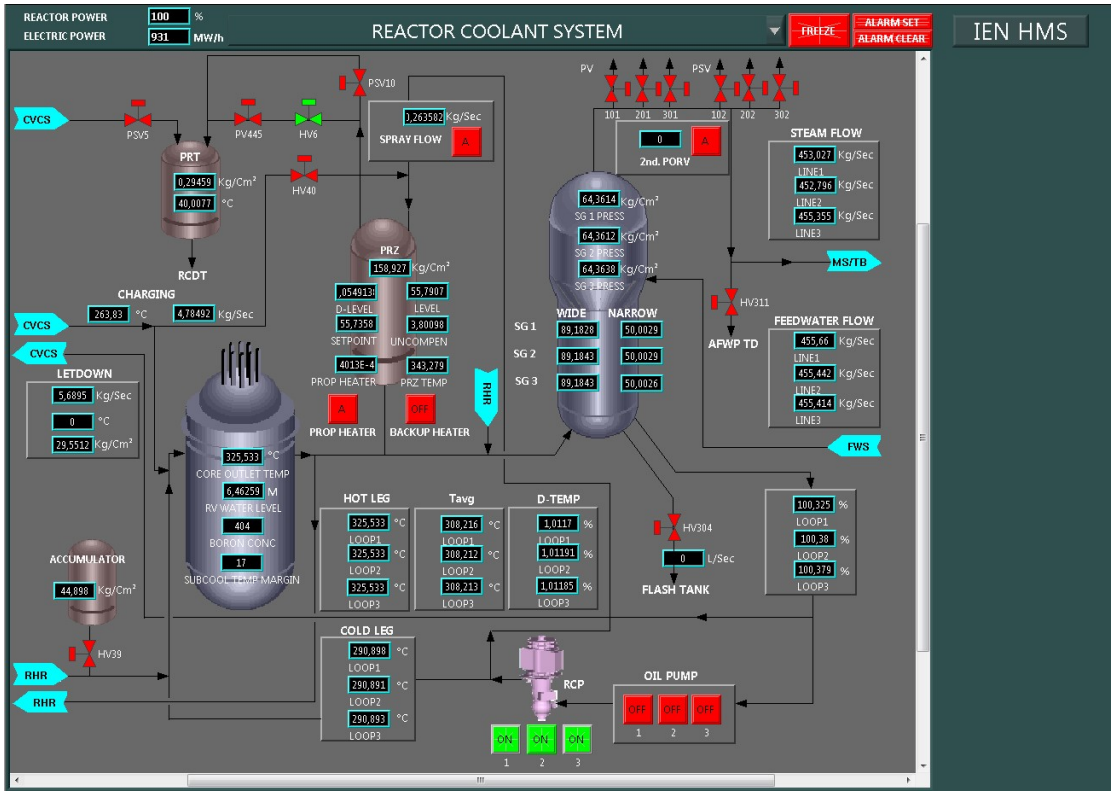


Figure 15. New MMI built using LabVIEW.

5. INTERFACE DEVELOPMENT COMPARISON BETWEEN THE ORIGINAL INTERFACE AND LABVIEW INTERFACE

The MMI development is quite different between the original one [11-13] and the one built using LabVIEW. On the original one, there are several specialized software used for specific tasks: the iLog Views Studio for component development, the HSI Builder for screen and layout development, and a software called MMI for running the man-machine interface. On the new solution, the LabVIEW (coupled with the DSC module) is used for most tasks: screen and layout development, and part of communication functionality. Besides LabVIEW, only the ICM, which runs on the workstation, is required for the new MMI, since its functionality was not originally present on the simulator software. The following sections present a comparison of the development process on the original MMI and on LabVIEW.

5.1. Component

The original MMI had its component library developed in conjunction with the HSI Builder and a software called MMI, already providing all the needed components for composing all MMI screens. Additionally, the iLog Views Studio software can be used to create new components. The new MMI made in LabView already has all the needed components, with the most specific ones, like pumps and valves, provided by the acquired DSC module. Thus component development on LabVIEW was not explored.

5.2. Screen

On the original MMI, editing screens on the HSI Builder software is a very simple task, just needing the dragging-and-dropping of the desired component onto a screen and setting its properties accordingly. On the new MMI, however, editing a component is a more complex task, requiring more work and caution, due to the need to advanced programming code on the screen's Diagram Block, and the way it receives and sends values from/to the simulator.

It should be noted, however, that the HSI Builder and MMI software were developed in C/C++ programming language using the iLog Views GUI and graphics toolkit, and made specifically for use with the simulator. While editing a screen on HSI Builder is currently easier than on LabVIEW, programming the HSI Builder software was a more demanding and complex task than simply acquiring the LabVIEW software and creating a new MMI with it.

While the old MMI uses shared memory to read/write to simulator's variables, on the LabVIEW software, this is done by manually programming each Block Diagram. The user needs to write specific code so that the component will use the network communication module, described in Section 3, to exchange data through a TCP/IP network with the simulator's instance at the HP Workstation.

5.3. Interface Layout

On the original MMI, the HSI Builder software was designed to create and modify a LAYOUT file that defines the MMI layout, making layout modification easy. On the new simulator interface developed in LabVIEW, it is difficult to reproduce accurately the original simulator MMI, so some adjustments were used, such as the drop-down list to change the current screen.

6. CONCLUSIONS AND COMMENTS

As computer technology evolves constantly, the obsolescence of software and hardware must be faced.

The MMI migration from the old PA-RISC to the PC architecture will ease the software support on the long run since PC hardware and software is the de facto standard today.

One of the issues identified is the complexity of creating and modifying screens, as it requires, for each rebuilt screen, writing advanced programming code of the Diagram Blocks. A more modular approach, separating as much as possible the programming from the Diagram Blocks on commonly used modules could ease screen creation and modification. Another issue is the desirable replication of the original MMI in a closer fashion, so that simulator operators face a smoother transition from the original MMI to the new one.

Currently, the LABIHS simulator is still running using the HSI developed with iLog Views Studio running on PA-RISC workstation. This happens because there are still many screens which need to be rebuilt using LabView.

To facilitate development process, we're currently studying a more modular approach for the programming logic present on the screens' Diagram Block. Also, we're applying an effort to achieve a more accurate reproduction of the simulator MMI layout. When both stages are complete, we will be able to work on the migration of all simulator screens to the PC architecture using LabVIEW.

It is important to note that the development of the ICM enabled us to make the LabVIEW software to interoperate with the core simulator processes still running on our legacy PA-RISC architecture. This way, we have been able to work on the MMI modernization presented in this paper, while the simulator core is still running on the old framework. In fact, even after the core simulation software is migrated to the PC architecture, the network communication module will still be a primal component not only for the operation of the simulator core modules but also to allow for the development of new simulator MMIs in the future.

REFERENCES

1. NUREG-0700, "Human-System Interface Design Review Guidelines", **U.S. Nuclear Regulatory Commission Research**, Washington & (2002).
2. NUREG-0800, "Standard Review Plan, Chapter 18 Human Factors Engineering", **U.S. Nuclear Regulatory Commission Research,** Washington & United States (2004).
3. NUREG-0711, "Human Factors Engineering Program Review Model", **U.S. Nuclear Regulatory Commission Research**, Washington & United States (2004).
4. O´Hara, J. Stubler, W., e Nasta, K., "Human-system interfaces management: Effects on operator performance and issue identification", **BNL Report W6546-1-1-7/97**, Upton, Brookhaven National Laboratory, Nova York & United States (1997).
5. Stübler W., Higgins J., and O´Hara J., "Evaluation of the potential safety-significance of hybrid human-system interface topics", **BNL Report J6012-T2-6/96**, Upton, Brookhaven National Laboratory, Nova York & United States (1996).
6. Balbo S., Draheim, D., Lutteroth, C., "Appropriateness of User Interfaces to Tasks", TAMODIA, Gdansk, Polônia pp. 26–27, (2005).
7. Vicente K.J., Rasmussen J., "Ecological interface design: theoretical foundations", ***IEEE Transactions on Systems, Man and Cybernetics***, **Vol**. 22 (4), pp. 589-606 (1992).
8. Lin, Y., "Experimental study based on eye gaze measurement for computer interface", **Technical Report (AEDL-2000-L7Z01)**, Advanced Engineering Design Laboratory, Department of Mechanical Engineering, University of Saskatchewan & Canada (2000).
9. CNEN-NE-1.01, "Licenciamento de operadores de reatores nucleares", **Comissão Nacional de Energia Nuclear**, CNEN, Brasil (1979).
10. "LabVIEW" https://en.wikipedia.org/wiki/LabVIEW (2017).
11. iLog, "iLog Views Studio 4.0 - User's Manual" (2000).
12. iLog, "iLog Views Controls 4.0 - User's Manual". (2000).
13. HSIL Simulator - Human System Interface Laboratory Simulator, **HIS Builder User's Manual**, Doc. ID: IEN-HSIL-DOC-06-APPENDIX 2, Instituto de Engenharia Nuclear, Brasil (2002).
14. "The X-Manager Software: All-in-One Network Connectivity Solution", http://www.netsarang.com/, 2011.
15. Kane, G., "PA-RiSC 2.0 Architecture", Prentice Hall, 1996.
16. Nichols, Bradford. *Pthreads programming: A POSIX Standard for better Multiprocessing*, O´Reilly, United Stated, 1998.
17. "RFC 1832 - XDR: External Data Representation Standard (RFC1832)", http://www.faqs.org/rfcs/rfc1832.html, 1995.