Nuclear Power Plant Simulators: A Component-based Approach

Manuel Díaz, Daniel Garrido, Sergio Romero, Bartolomé Rubio, Enrique Soler, José M. Troya
Department of Languages and Computing Science, University of Málaga, Spain
{mdr, dgarrido,sromero,tolo,esc,troya}@lcc.uma.es

Abstract

The development of nuclear power plant simulators is a hard task where many different factors have to be considered. Software components can be a main factor in the improvement of the development cycle of complex simulators. This paper presents a new generation of simulators based on two different component technologies. On the one hand, RT-CORBA is a communication middleware, which allows the development of predictable real-time applications taking benefit from CORBA features. On the other hand, CCA is a new component model for scientific components, which allows high performance computing needed for these simulators.

Keywords*:* Component, real-time, RT-CORBA, CCA

1. Introduction

The development of simulators for Nuclear Power Plants is traditionally based on old techniques and languages such as FORTRAN, Ada or C. However, new techniques and methodologies such as CORBA [13] or CCA [18] can be applied in the development of these systems. This paper shows the experiences obtained from the development of complex simulators using these new technologies.

Specifically, we present a simulation environment focused on the simulation of control rooms of nuclear power plants. These simulators must allow the training of the operators for the operations and maintenance of the power plant in a safe way. Previous simulators [4,5] developed together with the company Tecnatom S.A. are being used in several power plants located in Spain, Germany or Mexico. Currently, we want to develop a new generation of these simulators combining several component models such as RT-CORBA and CCA and take benefit from software components. Based on component interoperability, the component programming style allows the creation of more flexible and adaptable software, promoting reusability of components already developed and verifed in other projects and increasing, this way, the reliability of the final product. We have used components in two ways: communications using CORBA and the simulator kernel using CCA.

The Common Object Request Broker Architecture (CORBA) is the core of the Object Management Architecture described by the Object Management Group (OMG) for the building of distributed applications. The development of the simulators using CORBA allows independence of platform, operating system and programming languages improving the reusing of code. Nevertheless, standard CORBA Object Request Brokers (ORBs) traditionally only support best-effort capacities without temporal guarantees. In these simulators we need predictable temporal behavior. Thus, we have used the Real-time CORBA specification (RT-CORBA) [20].

On the other hand, the simulator kernel is a clear example of scientific computing. We need a high degree of performance. Standard component models such as CCM [14], DCOM [10] or JavaBeans [7,11] share serious shortcomings for parallel and distributed scientific applications, due to the lack of the abstraction needed by parallel and distributed programming and poor performance. They also have trouble with the mechanism for encapsulating an existing scientific application (which might itself be a parallel-distributed application) into a component.

A large effort is currently devoted by the Common Component Architecture (CCA) forum [18,3] to define a standard component architecture for high-performance computing.

In this new approach, a simulator kernel can be constructed by connecting the corresponding simulation models, now encapsuled into software components. Apart from componentization, an additional aspect can be taken into account in order to improve the system. In the current simulators, simulation models are implemented using sequential procedures. However, codes from some of these models can be parallelized. For example, in [2] a parallel version of the thermo hydraulic code encapsulated into the TRAC simulation model is described. Parallelization of this model is specially important since it represents about 80% of the total simulation time.

The paper is organized as follows: A global overview of the simulators is presented in section 2. Section 3 presents the software architecture. The following sections show issues related to simulator kernel, communications and applications. The paper finishes with some conclusions and future work.

2. System overview

The simulation projects of Tecnatom usually include two simulators that influence on the hardware and software architectures. The first simulator is called *Interactive Graphic Simulator* (*SGI*), which through graphic applications (see Figure 1) allows the training of future operators. The second simulator is called *Full Scope Simulator* (*SAT*), which is an exact replica of the control room of the power plant (see Figure 2).

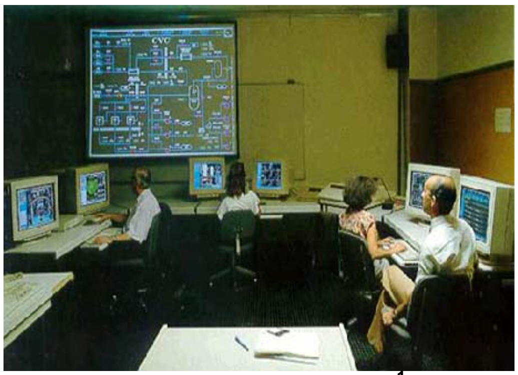


Figure 1. *SGI* simulator[1]

Figure 2. *SAT* simulator

[1] Pictures courtesy of Tecnatom S.A.

The main hardware elements of *SAT* and *SGI* are:

Simulation Computers: These computers are responsible for the simulation process executing the simulation models and providing data to the other software and hardware components. They are the main elements of the simulators.

Instructor Console: The *Instructor Console* only exists in the context of the simulators. This element is used by the instructor of the simulation session and it allows the creation of scenarios that have to be solved by the students.

Physical panels: The *Physical panels* are exact replicas of the existing in the control room. Operators of the power plant carry out their actions mainly through these panels, which have a lot of indicators, hardware keyboards, valves, etc.

Subsystems: Depending on the nuclear power plant, there will be several subsystems that have to be simulated. These subsystems are very different, and so the code reusing is a fundamental element in the development of the projects

Student workstations: The *SGI* simulators additionally include the hardware needed for the *student posts* of the simulator. Basically a *student post* allows the practice of all the areas of simulation in a comfortable way with several monitors for each post and graphical applications (see Figure 1).

3. Software architecture

There are two well differentiated parts in the software architecture. In the first part, we have the components that act like a simulation server (because they offer a set of simulation services to the other tools and applications), forming the tools and applications used by students and instructor the second part of the system. All these components are distributed applications that can be executed on any node of the network, setting their communications through CORBA. New components can be added to the system without modifications in the software architecture using a suitable communication infrastructure. Some of the most important high level components (described in later sections) together their interactions can be seen in Figure 3.

A main goal in the projects of Tecnatom is the creation of software components that could be reused in future projects. All the applications and libraries have been developed with this purpose, obtaining a high level of software componentization, specially with the flexibility and reusing obtained with the communication libraries and the CCA kernel.

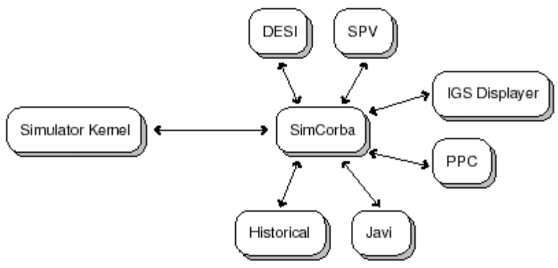


Figure 3. Main Software Components

4. Simulator kernel

SETRU is the kernel of the simulators of Tecnatom S.A. providing an execution environment for the simulation models required in a concrete simulator, so the execution, modifying, insertion, etc. of new simulation models is easy and the keeping of the real-time constraints of the models can be possible.

Simulation models contain the necessary code to simulate specific parts of the system. The execution of a simulation model usually requires reading or updating data variables which are computed by other models (we also refer to these data as simulation variables). The previous (classical) version of the simulator kernel resolved these types of inter-model data dependencies by declaring all shared data as global variables allowing access from any subroutine. Since we pursue the encapsulation of simulation models into separated software components, we must adopt a more appropriate mechanism for managing data dependencies. In our proposal, each component must report on:

-Simulation variables it needs, for reading or updating, from other models in order to be executed.

-Simulation variables it provides, which are computed and offered (exported) to the rest of models.

According to this, the programmer of a concrete simulation model component only has to declare, by implementing the corresponding methods, the simulation variables needed from/provided to the rest of models whereas a manager component integrated in the kernel is the one responsible for locating the requested variables (even if they are hosted in different processes) and providing them to the respective models. Specific inter-process communication schemes needed to resolve data dependencies efficiently are established in a configuration phase.

This way of managing data dependencies leads to a significant uncoupling among simulation models in both development and execution time. The programmer is not concerned about issues such as knowing the rest of models in the simulator or resolving data dependencies and so, he/she can be focussed on writing scientific code for the simulation model under development.

Simulation models are now encapsulated into (sequential or parallel) independent software components. Since all of them are CCA-compliant components implementing a concrete interface, we make sure they can be integrated into the same simulator, even if they are programmed using different languages or communication libraries. Our proposal makes the construction of different simulator kernels possible. This can be easily achieved by selecting and composing the corresponding simulation models from a component repository.

Figure 4 describes the Single Component Multiple Data paradigm used to implement the simulator kernel as a Ccaffeine [18] application. In this paradigm, all components together with the framework are instantiated in every participant process. Different components in the same process communicate among them through the CCA Port mechanism, which occurs, for example, when SETRU needs to call methods on models connected to it.

Simulation models can be programmed using both sequential and parallel programming styles. Instances in different processes of the same parallel component, e.g. TRAC in Figure 4, communicate with each other by using a parallel communication library such as MPI [16] or PVM [8]. In this case, all simulation variables computed by the model are distributed across the different processes. On the other hand, instances of a component that represents a sequential simulation model, e.g. SeqMod in Figure 4, do not communicate among them and the same whole computation is executed on every process, having their simulation variables replicated on all processes.

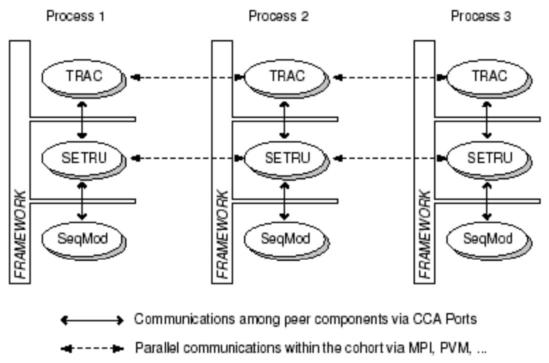

Process 1
Process 2
Process 3
TRAC
SETRU
SeqMod
FRAMEWORK
Communications among peer components via CCA Ports
Parallel communications within the cohort via MPI, PVM, ...

Figure 4. SCMD-parallel Simulator Kernel

Other situations can also be considered: for example, a parallel simulation model having both distributed and replicated data variables, or a sequential model being executed on one process only (not replicated). The proposed simulator kernel supports the integration of all these different types of simulation models together, maintaining data consistency for both distributed and replicated data.

5. Communications

We have an optimized simulator kernel. Now, we have to supply simulation variables to the other applications and tools of the simulator.

Simcorba acts like the "Simulation Server" for a big number of client applications because it offers a set of services like periodic updating of variables, actions over the simulator, etc. On the other hand, the component ***Receiver*** is responsible for the communication between *Simcorba* and the rest of applications. Basically, it is a data container supplying variables. There is a *Receiver* component for each application that needs to retrieve data from *Simcorba*.

A main goal of this pair is transparency. Any new application, component or tool, which wants to receive data only have to use the component *Receiver* and automatically it will have data from simulation provided by *Simcorba*. So, the applications that use *Receiver* components do not know anything about the internal implementation of its corresponding *Receiver* component. The behaviour of the *Receiver* component from the client's point of view is like a passive data container with arrays that can be consulted and automatically updated. Furthermore, the *Receiver* component offers a set of additional services. Figure 5 shows two applications and the *Receiver* component.

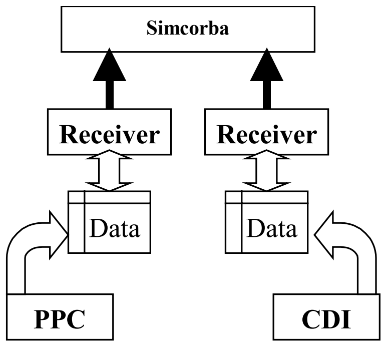


Figure 5. *Receiver* component

The *Receiver* components periodically request data to *Simcorba*. *Receiver* and *Simcorba* use the CLIENT PROPAGATED [20] model of RT-CORBA where the requests of the clients are performed with the same CORBA priority in all the system. In this way, the ORB can help to avoid priority inversions.

There are different client types that determinate the data received by the client. The client type ("post type") is determined by the application, existing types such as *PPC*, *SGI*, CDI, etc. *Simcorba* has the necessary information to know the variables needed for each type in a flexible way that allows for the incorporation of new types without needing to modify the developed code. Furthermore, each type has its own priority. In this way, different applications have different importance for *Simcorba*. For example, the CDI type represents to the *Instructor Console* whose orders are more important than the updating of *SGI* posts. Figure 6 shows an example of *Simcorba*/*Receiver* with *SGI* and CDI clients where the priority of the CDI client is the highest.

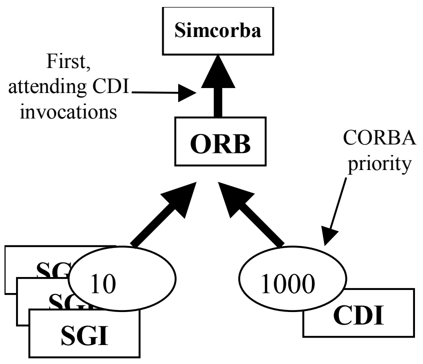


Figure 6. Simcorba/Receiver and Priorities

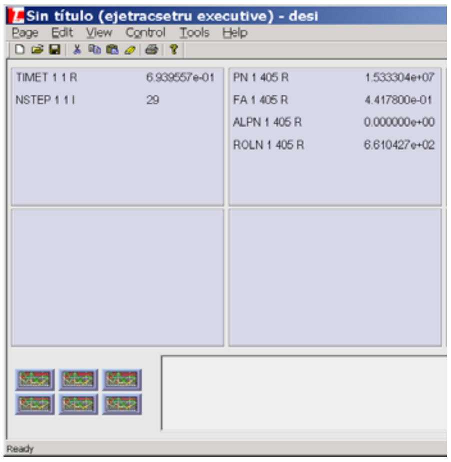
The number of static clients is known when *Simcorba* is started. So, *Simcorba* creates thread pools with lanes [21] for the static clients and for possible dynamic clients. The thread pools are configured according with the priorities of the different post types. On the other hand when *Receiver* components are initialised, the connection with *Simcorba* is verified using the explicit binding mechanism of RT-CORBA. In this way, the *Receiver* components will have reserved the suitable network resources. Private connections with priority banding are also used to guarantee QoS between *Simcorba* and *Receivers.*

6. Applications and tools

The simulation kernel is supported by a set of applications varying in each simulator. In this section we present some of the most important tools and applications.

The applications and tools are different in each simulator and all these elements are integrated into a simulator through the *Receiver* component, which allow the data recovering from the simulator kernel.

There are applications that always are used such as the ***Variables Debugger*** (*DESI*) and the ***Supervisor*** (*SPV*). *DESI* allows the query and modification of the value of existing variables associated with the simulation models, being very useful for the validation of these models and for its use in training sessions. *SPV* allows the modifying of different simulation aspects such as executed models, timing control of the models, etc. Figure 7 shows a fragment of *DESI* showing some variables: *TIMET* (simulation time), *NSTEP* (simulation step), *PN* (pressure), *FA* (area), etc.


Sin título (ejetracsetru executive) - desi
Page Edit View Control Tools Help
Ready

```
TIMET 1 1 R      6.939557e-01     PN 1 405 R       1.533304e+07
NSTEP 1 1 I      29               FA 1 405 R       4.417800e-01
                                  ALPN 1 405 R     0.000000e+00
                                  ROLN 1 405 R     6.610427e+02
```

Figure 7. *DESI* application

The ***SGI Displayers*** allow the training of operators in a classroom differing the simulation in the *SAT* simulator, where the different components of the control room are directly manipulated. A *SGI Displayer* allows the visualization of graphical sheets with the different components existing in the control room. So, for example, the students can perform actions like opening and closing of valves, alarms recognition, etc. Furthermore, the actions taken by the students will have repercussion in the global state of the simulation session and they will affect other students. The *SGI Displayers* are organized into an instructor/student scheme, where the instructor can carry out the same actions that students can, and additionally other actions related to the management of the simulation session. Examples of these actions are the control of connected students, modification of special variables, etc.

Figure 8 shows an example of simulation session with *SGI* posts and the reusing of the *Receiver* component. Each *SGI* post uses a *Receiver* component for the communication with the simulator.

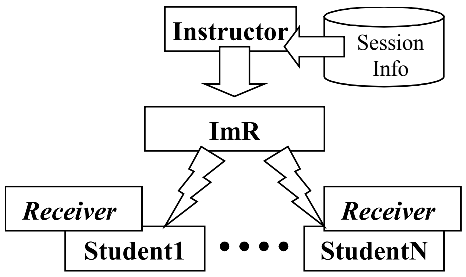


Figure 8. *SGI Displayers*

The communication software of the *SGI Displayers* was organized into a dynamic library and as in the case of *Receiver*s; the underlying communication layer is hidden from the applications that use it. The instructor additionally offers services to the student posts like connection to started simulation sessions, information about sheets, etc.

The *Receiver* component has to provide data to the *SGI Displayers* at a rate of 250 ms. With this rate, the user has the sensation of a correct animation. Higher delays will cause an uncomfortable sensation and the user could make errors by the incorrect animation of the graphical sheets.

The Plant Process Computer (**PPC**) subsystem was developed for a specific simulator (Trillo). It is an example of big subsystem integrated into a simulator. The main goal of this subsystem is to report the state of the plant in the control room through alarms and graphics. It has several computers, physical panels, keyboards, etc. and the simulation of this subsystem includes software and hardware of the original *PPC*. Some real components of the *PPC* have been replicated into the *SGI Displayers*. So, for example, the keyboards of the *PPC* have their counterpart in the software keyboards (shown in Figure 9).

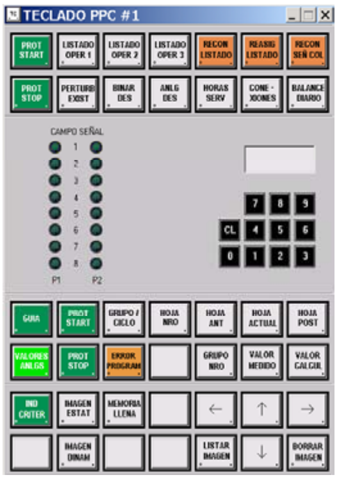


Figure 9. PPC keyboard

PPC includes several *Physical panels*, computers controlling the state of the Plant (each computer is responsible for different subsystems of the power plant), screens (at least 10) and keyboards associated with these screens. Each screen can be used in different execution modes and in these modes the operator can see alarms, query variables, etc.

7. Conclusions

The development of simulators for nuclear power plants is a hard task where the software is developed with *ad hoc* solutions. Software components can improve the development of these simulators.

This paper has presented a new generation of simulators with components in two main areas: communications and the kernel of the simulator.

RT-CORBA has allowed the development of communication components, which hide details to developers. On the other hand, CCA has been used to obtain an optimized and parallel simulator kernel.

The techniques used in the simulators improve the reusing of code allowing lower costs in the development of simulators and more flexible simulators.

8. References

1. Allan, B.A., Armstrong, R.C., Wolfe, A.P., Ray, J., Bernholdt, D.E., Kohl, J.A., The CCA Core Specification in a Distributed Memory SPMD Framework, Concurrency and Computation: Practice and Experience, 14, 5 (2002), pp. 323-345.
2. Alvarez, J.M., Díaz, M., Llopis, L., Rus, F., Soler, E., Practical Parallelization Strategies of a Thermohydraulic Code, in Proceedings of Euroconference in Supercomputation in Non Linear and Disordered Systems, pp. 254-258, Madrid, Spain, 1996.
3. Components@LLNL: Babel, home page http://www.llnl.gov/CASC/components/babel.html.
4. Díaz, M., Garrido, D., Applying RT-CORBA in Nuclear Power Plant Simulators, in 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2004), pp. 7-14, IEEE Computer Society, Vienna, Austria, 2004.
5. Díaz, M., Garrido, D., A Simulation Environment for Nuclear Power Plants, in 8th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2004), pp. 98-105, IEEE Computer Society, Budapest, Hungary, 2004.
6. Díaz, M., Rubio, B., Soler, E., Troya, J.M., SBASCO: Skeleton-Based Scientific Components, in Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2004), pp. 318-324, IEEE Computer Society, La Coruña, Spain, 2004.
7. Englander, R., Developing Java Beans. O'Really&Associates, 1997.
8. Geist, Al., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, R., Sunderam, V.S., PVM: Parallel Virtual Machine. MIT Press, 1994.
9. Heineman, G.T., Council, W.T., Component-Based Software Engineering: Putting the Pieces Together. Addision Wesley, 2001.
10. Horsmann, M., Kirtland, M., DCOM Architecture, Microsoft White Paper, 1997. Available from http://www.microsoft.com/com/wpaper.
11. Monson-Haefel, R., Enterprise Java Beans 3th edition, O'Really&Associates, 2001.
12. Nieplocha, J., Harrison, R.J., Littlefield, R.J., Global Arrays: a Portable Shared Memory Programming Model for Distributed Memory Computers, in Supercomputing'94, pp. 340-349, Los Alamitos, California, USA, 1994.
13. Object Management Group, CORBA home page http://www.corba.org.
14. Object Management Group (OMG), Specification of Corba Component Model (CCM). http://www.omg.org/technology/documents/formal/components.htm.
15. Smith, B., Bjorstad, P., Gropp, W., Domain Decomposition. Parallel Multilevel Methods for Elliptic P.D.E.'s. Cambridge University Press, 1996.
16. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J., MPI: The Complete Reference, volume 1-The MPI Core. MIT Press, 1998.
17. Tecnatom S.A. home page http://www.tecnatom.es
18. The Common Component Architecture Forum, home page http://www.cca-forum.org.
19. Vanneschi, M., The Programming Model of ASSIST, an Environment for Parallel and Distributed Portable Applications, Parallel Computing, 28, 12 (2002), pp. 1709-1732.
20. Schmidt, D.C., Kuhns, F., An overview of the Real-time CORBA Specification" in IEEE Computer special issue on Object-Oriented Real-time Distributed Computing, June 2000.
21. Levine, D.L., Mungee, S., Schmidt, D.C., The Design of the TAO Real-Time Object Request Broker, Computer Communications 21, 1998. pp. 294-324, 8.