REAL-TIME TRAINING SIMULATORS BASED ON DISTRIBUTED REUSABLE COMPONENTS

MANUEL DIAZ, DANIEL GARRIDO, JOSE M. TROYA

Department of Languages and Computing Science
University of Málaga, Spain
Málaga, 29071
Email: {mdr,dgarrido,troya}@lcc.uma.es

Abstract: The development of training environments for the operations and maintenance of nuclear power plants is a long process where many components are not reused due to specific characteristics of each power plant. This paper presents an environment formed by a simulation kernel with a set of tools aiming at building reusable components. The simulators are being used for the real-time training of future operators in a safe way and they are based on a complex architecture of simulation models with real-time constraints involving many different applications where modern object-oriented methodologies and technologies have been applied. Additionally, we present a new real-time component model under development for the building of new applications.

Keywords: Simulator, distributed, RT-CORBA, real-time, nuclear power plant, components

1. INTRODUCTION

The building of a new training simulator for the operations and maintenance of a nuclear power plant is an expensive process in time and money. Many times, the software of previous simulators is not reused because it has been developed for a specific simulator with its specific features, architecture, applications, subsystems, etc. Nevertheless, the kernel of the simulators and its associated tools could benefit from common points presents in all the plants and simulators like simulation models, common tools (debugger, supervisor, etc.), communication infrastructure, etc.

The main contribution of this paper is the presentation of the utilization of modern object-oriented methodologies and technologies like UML [Booch et al, 1998]], and CORBA [Henning and Vinoski, 1999] (specifically RT-CORBA [Schmidt and Kuhns, 2000]) into complex distributed simulators aiming at the building of reusable components. Additionally, we present a component model and its associated environment for the development of real-time applications. We plan on using this model for the development of some new components in future projects. The main contribution of the environment is its support for real-time analysis at the component and application level. The real-time analysis is achieved by combining component meta-information in the form of an abstract behavior model and a method to measure worst-case execution times in the final platform. This component model was initially designed for embedded systems [Díaz et al, 2004] but we think that it is possible the readjustment for generic real-time systems.

The paper is situated within the context of simulators for a nuclear power plant located in Trillo (Spain) including a Full Scope Simulator intended for the training of future operators, allowing for the practice of different situations, from the most common ones like temperature monitoring, valve operation, etc. to most unusual situations like emergency situations that logically cannot be reproduced for practice in the real Control Room.

The simulator is an exact replica of the Control Room of the power plant, accounting for all details, from physical artifacts like furniture, control panels, etc. to software, simulating the applications running in the power plant. The kernel of the simulator is formed by simulation models which calculate the values of the distinct variables.
The software described in this paper has been carried out in a shared project between the company Tecnatom S.A. and the department of Languages and Computing Science at the University of Málaga. The work has been mainly related to the adaptation to CORBA and UML of existing software and to the creation of new applications needed for this concrete simulator, with special emphasis on the building of software components [Szyperski, 1999] that could be reused in future projects.

There are three main aspects in the software developed:

- UML with Rational Rose, obtaining documentation and improving the software process.
- Components: for the reusing of the software and the modularization.
- CORBA for the communications, obtaining interoperability, transparency and independence of platform and programming language.

Currently, there are several simulators for power plants developed and used in Spain, Germany and Tecnatom S.A. has new projects like a simulator for Laguna Verde in Mexico

The rest of the paper is organized as follows: The rest of this section presents information about RT-CORBA. The hardware architecture is presented in section 2. Section 3 presents the software architecture with implementation issues. Section 4 is about on-going projects. Section 5 presents the basis of the component model. The paper finishes with some conclusions and future work.

1.1. RT-CORBA Features

The Common Object Request Broker Architecture (CORBA) is a communications middleware that allows the communication of objects developed in different programming languages and running on different hosts or operating systems in a transparent way [Henning and Vinoski, 1999]. These objects (*servers*) define interfaces with operations provided to the *clients*. The *clients* only use these operations and there is not difference between invocations to local objects and invocations to remote objects. All the communication details are managed by CORBA.

Temporal predictability is a main aspect in the development of real-time applications. However, standard CORBA implementations are not suitable for real-time because they only support best-effort capacities in the communications and there is not guarantees about the temporal response on particular invocations to remote objects. So, the solution is the utilization of the Real-time CORBA specification. Real-time CORBA provides mechanisms that allow to configure and control: processor resources, communication resources and memory resources.

Native and CORBA priorities: Real-time CORBA applications can use CORBA priorities that allow to hide the heterogeneity of *native* priorities in the different Operating Systems of a distributed application. RT-CORBA priorities can be specified with ranges in value between 0 and 32767 and these priorities are used in a platform-independent way.

Server declared and Client propagated priorities: Two different policies are used to transmit priorities. In the SERVER DECLARED model, the server declares the priorities at which an invocation made on an object will execute. The CLIENT_PROPAGATED model allows the transmission of the client's priorities that must be honored by servers, avoiding priority inversions problems.

Thread pools: The pools allow the pre-creation of threads in a way that a thread manages each invocation on a particular object. This way, the cost and unpredictability of dynamic threads are avoided. The thread pools can contain static and dynamic threads and they can be created with lanes with different priorities, allowing redistribute the invocations attending to the priorities of the clients.

Priority transforms: This mechanism allows to change the priority at which some invocations are performed on particular objects when particular situations require the transformation of the original priority.

Mutexes: The RT-CORBA mutexes are the standard synchronization mechanism of RT-CORBA that allow for the priority inheritance and priority ceiling protocols [Burns and Wellings, 2001].

Protocol properties: The underlying transport protocol used by a particular Object Request Broker (ORB) (e.g.: IIOP - TCP/IP) can be configured by RT-CORBA allowing to benefit from special features, such as ATM virtual circuits, etc.

Managing connections: Explicit binding and Private connections can be used to avoid the unpredictability related to the implicit activation of objects and multiplexed connections of standard CORBA ORBs. These mechanisms allow to pre-establish non-multiplexed connections and control how client requests are sent over these connections.

2. HARDWARE ARCHITECTURE

There are two different simulators that influence on the hardware architecture and the physical infrastructures. 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 a power plant (see Figure 2).

The high-level hardware components of *SAT* and *SGI* includes the *Simulation Computers*, a *Plant Process Computer*, the *Instructor Console* and the *Physical panels*.

Figure 1: *SGI* simulator[1]

Figure 2: *SAT* simulator

The *Simulation Computers* are responsible for the simulation process executing the simulation models and providing data to the rest of software and hardware components.

The *Plant Process Computer* (*PPC*) is a complex subsystem responsible for carrying out the simulation of this component of the Control Room and whose main task is the monitoring of data values and presentation of alarms, graphs and reports.
The *Instructor Console* only exists in the context of the simulators, and it allows the creation of scenarios that have to be solved by the students.

Physical panels are exact replicas of the existing in the Control Room. They are situated in a large room (17x18 m.). Operators of the power plant carry out their actions mainly through these panels, which have a lot of indicators, hardware keyboards, valves, etc. These panels allow the connection between hardware and software components.

[1] Pictures courtesy of Tecnatom S.A.

The *SGI* simulator additionally includes the needed hardware for the student workstations of the simulator. Basically a student workstation allows the practice of all the areas of the simulation in a comfortable way with several monitors for each student and graphical applications.

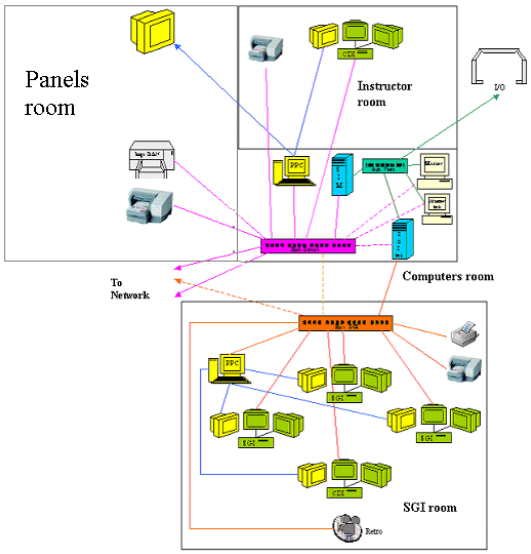


Figure 3: *SGI-SAT* simulator

Figure 3 shows a global overview of the simulators. At the top of the figure we can see the instructor room, where the instructor manipulates the training session. Near this room we find the computers room with the simulation computers and the computers simulating the subsystems. In the SAT simulator, there is another room with the physical panels and finally there is a room with the different workstations for each student in the SGI simulator.

3. SOFTWARE ARCHITECTURE

Two different development environments were used: Unix and Windows. The Operating System IRIX 6.5 of Silicon Graphics was used in Unix. On the other hand, Windows NT and Windows 2000 with Microsoft Visual C++ 6.0 were used in Windows environments. Different languages had to be used: C++, FORTRAN, Java, etc. In the case of CORBA the implementation used was TAO 1.2. [Levine et al, 1998], freely available CORBA ORB that due to characteristics like predictable timing and robustness is very adequate for its utilization in real-time applications.

New components can be added to the system without modifications in the software architecture using a suitable communications infrastructure. Some of the most important higher-level

components together their interactions can be seen in Figure 4.

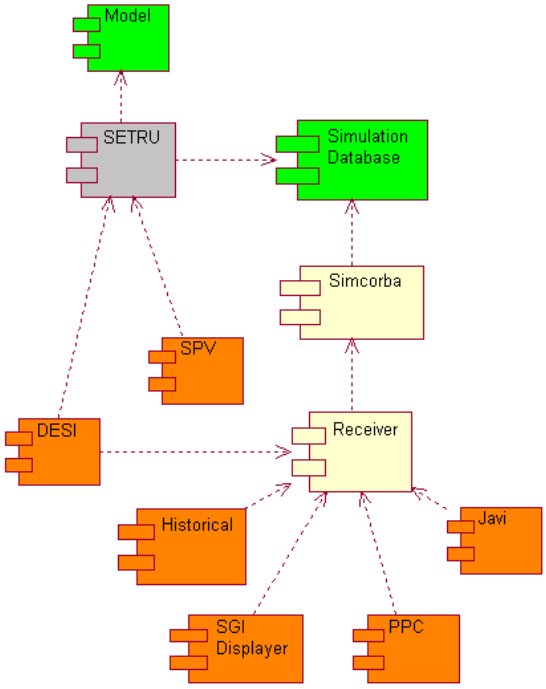


Figure 4: Main software components

There are two well differentiated parts in the final software architecture. The first part includes the components that act like a simulation server (because they offer a set of simulation services to the rest of tools and applications). The second part includes the rest of tools and applications used by the students and the instructor. All these components are distributed applications that can be executed on any node of the network, setting their communications through CORBA.

Basically, we have a simulation server providing data to the different applications and tools. The simulation server hides the source of alarms, users actions, etc. So, the two simulators (SGI and SAT) can interoperate between them and the interaction between tools, simulation models, etc. is performed through variables.

3.1. Simulator Kernel

SETRU is the kernel of the different 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 would be easy keeping the real-time constraints of the models.

The simulation models are responsible for the precise simulation of physical components of the real system, like valves, sensors, actuators, etc. providing to the applications a set of simulation variables representing the physical components and which can be queried, modified, etc. Each model has a different execution priority [Burns and Wellings, 2001] determined by the number of executions in a second. So, the models with more executions have a higher priority than other models and the operating system and the CORBA implementation give priority to their executions.

The number of execution of the models can be configured in files as the following:

```
RUN interfa1   1  1  1  1  1  1  1  1
RUN TRAC_rt    1  1  1  1  1  1  1  1
RUN interfa2   1  1  1  0  1  1  1  1
```

In this example, there are three models: *interfa1*, *TRAC_rt* and *interfa2*. The numbers '1' indicate the number of executions in a second. So, the two first models are executed 8 times in a second and they have the same priority. On the other hand, the last model is executed 7 times and it has a lower priority.

The communication between *SETRU* and the rest of components is done through the *Simcorba* component. For a large part of the applications, it acts like the "simulation server" because it offers a set of services like periodic transfer of variables, updating of variables, etc. The clients of *Simcorba* can use the *Receiver* component that acts like a passive data container hiding the communication details.

RT-CORBA allowed the definition of different types of clients where each type has a different CORBA priority respected along the overall system in a transparent way. The clients with a higher priority are attended firstly. Figure 5 shows an example of *Simcorba*/*Receiver* with *SGI* and CDI clients where the priority of the *CDI* post type (1000) is the highest.

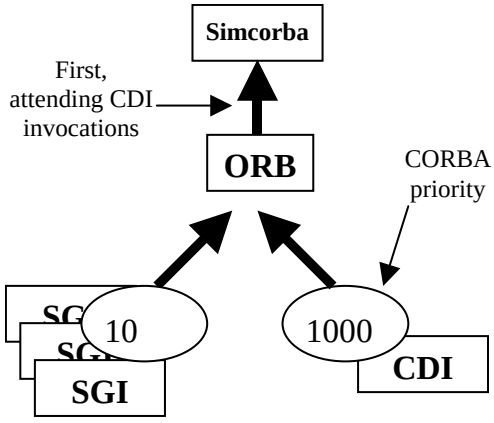


Figure 5: Priorities and client types

The applications that use *Receiver* components do not know anything about the internal implementation of its corresponding *Receiver* component. The behavior 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 with new simulation data. Furthermore, the *Receiver* component offers a set of additional services like the possibility of changing the values of variables.

Figure 6 shows two different applications and the *Receiver* component.

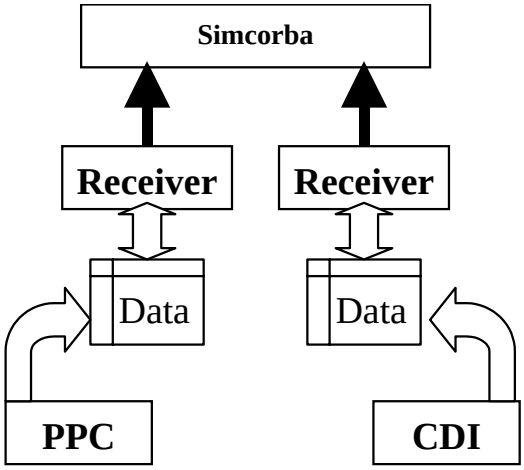


Figure 6: *Receiver* component

3.2. Common Tools: Debugger and Supervisor

The *Variables Debugger* (*DESI*) and the *Supervisor* (*SPV*) are two applications that can be used in different simulators.

DESI (Figure 7) allows the query and modification of the value of variables associated with the simulation models, being very useful for the validation of these models and for its use in training sessions.

Figure 7 shows a window of *DESI* running for a given scenario and some variables: *TIMET* (Simulation Time), *NSTEP* (simulation step), *PN* (pressure), *FA* (area).

One important aspect of the simulation models is the division of the power plant into components and cells (see Figure 8).

Figure 7: *DESI* application

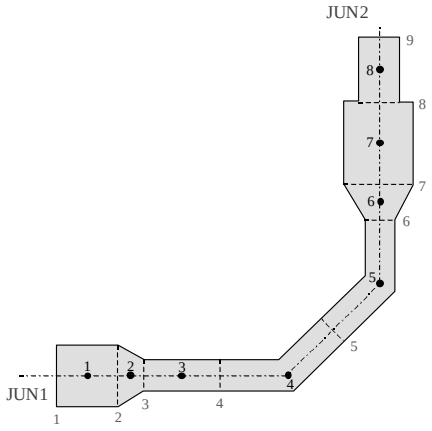


Figure 8: Component and cells

The Plant is divided into a large number of components and each component is divided into a set of cells. The role of the simulation models is to update the values of different variables (pressure,

velocity of liquid,) for each component and associated cells.

SPV allows the modification of different simulation features like executed models, timing control of the models, etc. In Figure 9 we can see the main window of the *Supervisor* with the run-time information about some simulation models: *interfa1, trac_rt, interfa2. SPV allows to the instructor* the control of the simulation models of the session.

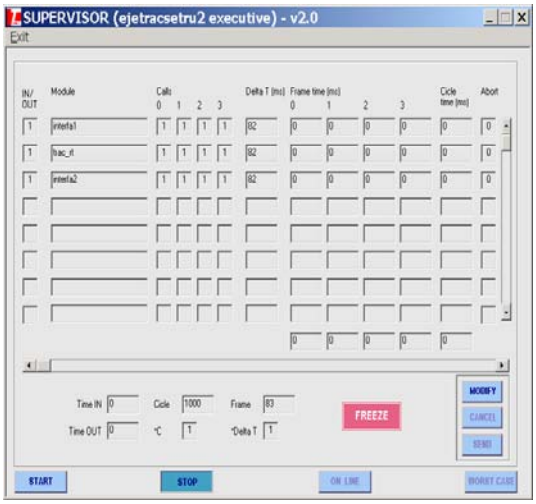


Figure 9: SPV application

Contrary to *Simcorba/Receiver* where the main intention was the transfer of huge data volumes from the *server* to classical *clients*, the communication between *SETRU* and these tools use a client/server model where the client applications (*DESI* and *SPV*) will send different commands to *SETRU.*

Thread pools and mutexes of RT-CORBA were used to guarantee sufficient resources for the tools and to guarantee mutual exclusion zones in *SETRU* avoiding, for example, the simultaneous execution of contradictory orders like *RUN* or *FREEZE*.

3.3. New Applications

The SGI-SAT project required the development of new applications that could be reused in future projects. These applications include: *SGI Displayers*, the *PPC* subsystem and *Javi.*

SGI Displayers. A *SGI Displayer* allows the visualization of graphical sheets (Figure 10) with the different components existing in the Control Room. In these sheets the students can perform the same actions as in the SAT simulators.

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.

The communication software of the *SGI Displayers* was organized into a dynamic library as in the case of *Receivers*; the underlying communication layer is hidden from the applications that use it.

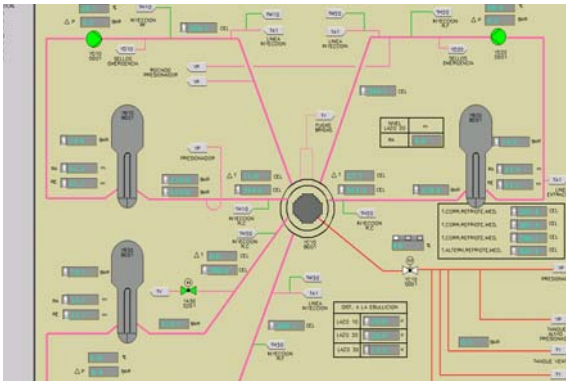


Figure 10: Example of *SGI* application

The Implementation Repository [Henning and Vinoski, 1999] of TAO was used to carry out the automatic starting of the students. The instructor additionally offers services to the student posts like connecting to started simulation sessions, to request information of existing sheets, etc.

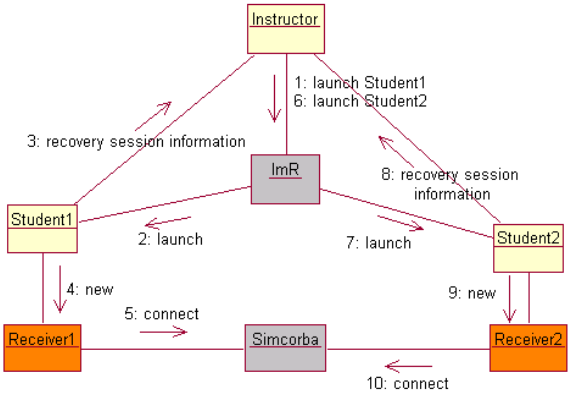


Figure 11: Session starting

Figure 11 shows how a simulation session is started with two remote students. The instructor configures the workstations. After this, the Implementation Repository of TAO starts the *SGI Displayers* of the students; each active *SGI Displayer* asks for

configuration information of the actual simulation session (information like simulator, sheets allowed, etc).

When the *SGI Displayer* has all the information needed, it proceeds to initialize the associated *Receiver* component. At that point, the *SGI Displayer* begins to receive data through *Simcorba* and the *Receiver* component uses such data to correctly display the sheets.

The main real-time constraint of this application is related to the updating of the screens.
This update must be done 4 times in a second and it is very related with the data reception from the *Receiver* component used by the *Displayer*.

***PPC* subsystem.** The main goal of this project was to completely simulate the behavior of the *Plant Process Computer* of the power plant. The mission of this computer is to report the state of the plant in the Control Room through alarms and graphics on screens. This is an example of complex subsystem integrated with the rest of applications and tools in the *SAT* simulator.

Some real components of the *PPC* have been replicated into the *SGI*. So, for example, the keyboards of the *PPC* have their counterpart in the software keyboards (showed in Figure 12).

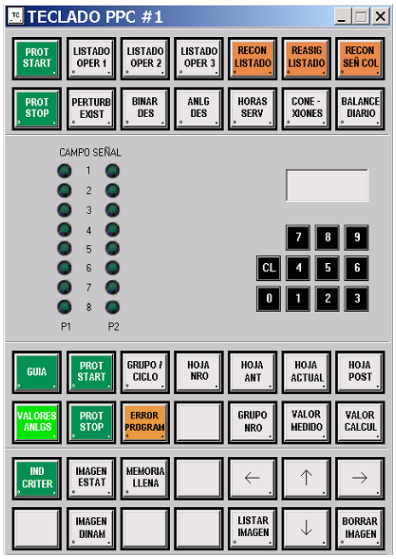


Figure 12: PPC keyboard

The communication areas of the simulated *PPC* (Figure 13) use the pair *Simcorba/Receiver* for each simulated computer. The *PPC* Kernel uses a thread pool for the requests of keyboards, screens and computers. The highest priority is for the computers; secondly, the screens and with the lowest priority the keyboards.

The computers periodically (at least 4 times in a second) check for the existence of new alarms and provide these data to the *PPC* Kernel. The execution cycle of the *PPC* Kernel is responsible for processing these alarms and it creates the information in a suitable format for the screens.

The actions of the users are aperiodic and are processed by the keyboard manager contained into the *PPC* Kernel.

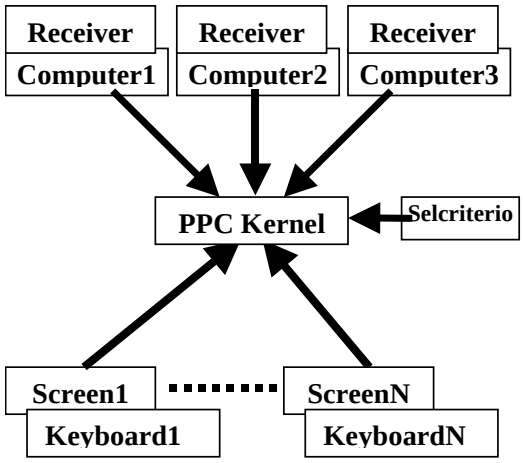


Figure 13: *PPC* RT-CORBA architecture

Finally, explicit binding of CORBA objects with private connections are used to improve the connection between keyboards, screens and computers whose number is fixed and known in starting time.

***Javi* application.** *Javi* is a new application initially not used in the Trillo Project. This application has reused previous components, specially the ones related to the communications, using the same interfaces as the rest of other components.

The *Javi* application allows the graphical representation of the state of the plant through color codes and has been developed using Java-2D (Figure 14) and Java-3D (Figure 15).

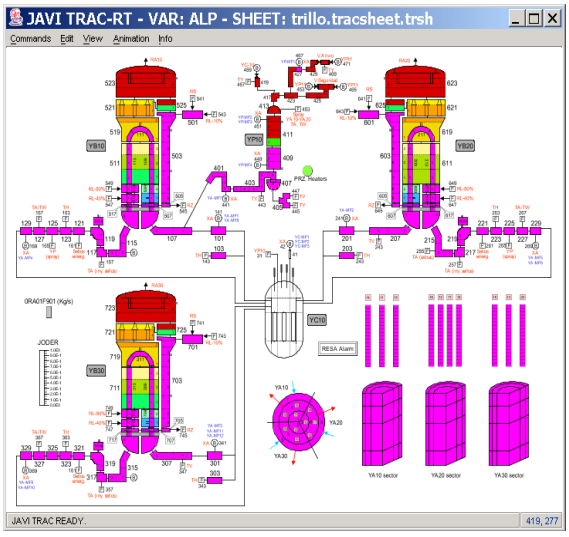


Figure 14: Javi application

The main novelty of this application is the utilization of Java. Thanks to Java, the same application can be used into Linux, Windows, Unix, etc.

Figure 15 shows a 3D-graphic of Javi. With these graphics, the operators can have a global overview of the vessel of the power plant and easily detect problems.

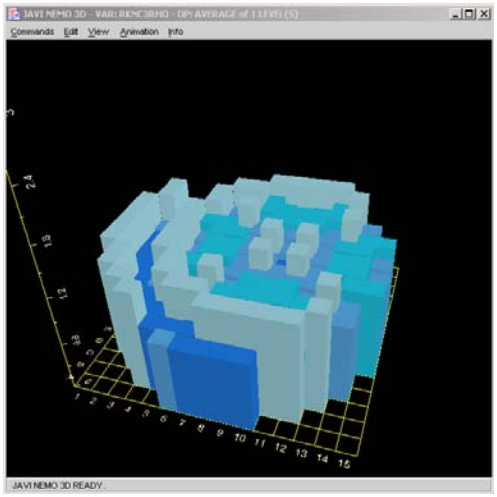


Figure 15: 3D-Javi graphic

4. CODE GENERATORS

Building of simulation models is a tedious task prone to errors. On the other hand, many simulation models are similar and the generation of the code could be automatic. Due to these reasons, Tecnatom S.A. and the University of Málaga have new projects for the automatic generation of code. The first of these tools is METRANE. Using METRANE, the user can indicate instructions to calculate the value of some simulation variables. Additionally to METRANE, we are developing TEAMLOGIC, a tool for the design of simulation models in a graphical way. This tool will allow for the automatic generation of code (FORTRAN, C++, Java) avoiding errors related to programming.

4.1. METRANE

METRANE allows the automatic generation of code for static and dynamic variables.

In the case of static variables, the user writes expressions indicating input variables, modifiers for the value of the variables (add, multiply, lower value, higher value and initial value) and one function to apply to the input variables. This point of view is closer to the developers of the simulation models, who frequently are physicist or mathematician (not software developers). The following is an example of expression to calculate a variable:

```
RSUM
#Name type add mult. Ll  Hl    init
VAR1  R8   0.0 1.0   0.0 1.0E20 0.0
I1    R8   0.0 1.0   0.0 1.0E20 0.0
I2    R8   0.0 1.0   0.0 1.0E20 0.0
```

The modifiers change the original value of the variable adding a constant or multiplying the value and finally applying limits to the final value of the input variables. The input variables are applied to the function RSUM and the result is stored into the output variable.

In the case of dynamic variables, the user can create new variables through functional expressions. These variables can be used for every cell and component of the power plant. The following example shows some of these variables and their expressions.

```
# Liquid mass of cell
CellMassL R4 FUNC CELL
        'VOL(cell) * (1-ALP(cell)) * ROL(cell)'

# Water vapor mass of cell
CellMassV R4 FUNC CELL
        'VOL(cell) * ALP(cell) * ROV(cell)'
```

These expressions are again easier to understand by the developers of the simulation model. However, in this case the variables cannot be modified because they are based on functions.

Note that in both cases (static and dynamic) these variables can be used by the rest of tools (e.g. DESI)

and simulation models. Additional details about METRANE can be found in [Díaz et al, 2003].

4.2. TEAMLOGIC

TEAMLOGIC is a new tool under development. This tool allows the building of circuits where the variables are connected to icons representing functions. The tool automatically obtains the code equivalent to these circuits and the icons are similar to libraries with encapsulated code without errors.

After the code generation, the tool allows the integration of this code in some remote simulator and the execution of these new simulation models with the automatic insertion in makefiles, creation of variables in the simulation database, etc.

Figure 16 shows an example of sheet developed with TEAMLOGIC.

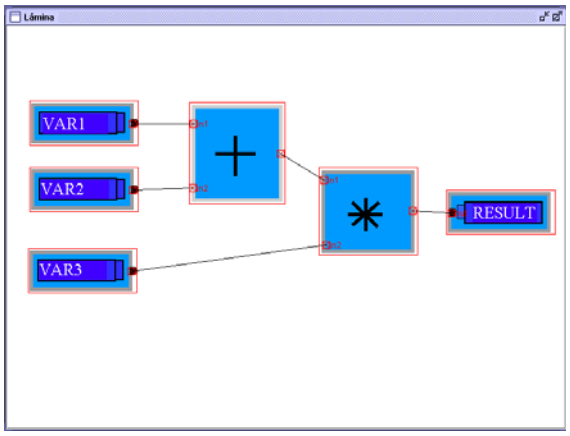


Figure 16: TEAMLOGIC

The tool is in its first stages but FORTRAN code can currently be generated for single circuits and integrated into executables.

5. REAL-TIME COMPONENTS

CORBA is very valuable for the development of distributed applications. However, sometimes the development of CORBA-based applications is not easy for new developers. Furthermore, CORBA is object-oriented, so the need for modifications in CORBA for component-based applications is widely recognised [Szyperski, 1999] [Henning and Vinoski, 1999]. The approach of OMG is the utilization of the CORBA Component Model (CCM) [O'Ryan et al, 1998]. Nevertheless, CCM is not a solution for real-time applications and the additional features provided, for example, by RT-CORBA need to be used. So, we plan on using a component model developed in the University of Málaga for the development of real-time distributed applications. Our approach [Díaz et al, 2004] is based on the development of components with our model and the deployment of these components on a predictable execution environment like, for example, RT-CORBA ORBs.

The model considers two different views: the component developer view and the component user view. In the first one, we describe the basic types of components and how they can be combined during the development phase. In the user view the component is described through its interfaces, communication mechanisms, configuration parameters and real-time constraints. This view does not include any details about component internals and it is used by the assembling and analysis tools.

Component Types. There are two main component types: primitive and generic. Generic components are containers of primitive and, possibly, other generic components.
Primitive components can be either active or passive. Active components are threads that share information through passive components or other generic components (by means of method invocations or raising events).

A generic component can have any arbitrary number of active and passive components. Active and passive components are declared inside a generic component as it is shown in Figure 17. Passive components are used to encapsulate shared resources inside a component.

Components Interactions. The model allows for communication between components through interfaces and events.

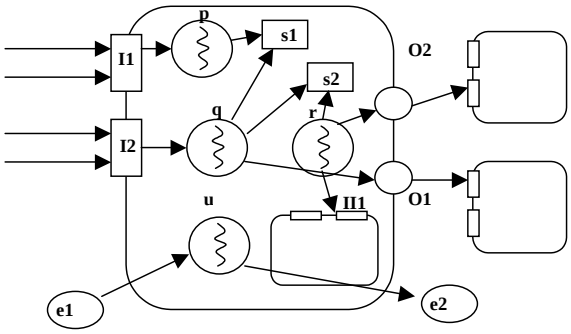
The interfaces are defined outside the component using a CORBA-based Interface Definition Language (IDL) and they are the standard communication method between components. We distinguish between input interfaces and output interfaces. Input interfaces provide a set of services that can be used by other components. The components need the services provided by other components, so we define output interfaces with the services required by our components.

In our model, it is possible to develop and distribute components that will be combined with third-party components through the interconnection between input and output interfaces. Additionally, the developer performs the connection between the input interfaces and the part of code (Active, Passive or Component) that will execute the invocation.

The events (data structures) allow to communicate components in an anonymous way [Henning and

Vinoski, 1999]. The components declare the different type of events that they can *consume* and the different type of events that they can *publish*. The events are raised in an asynchronous way and they are caught by all the components waiting for a particular type of event. This way, it is possible the communication between components without common interfaces improving the decoupling between components.

Component Configuration. Component configuration is the process of providing the domain and platform dependent information to the component environment tools in order to make it possible to create a running instance of a component. We define constant configuration parameters through the use of *configuration slots*. A configuration slot is a section in the component definition that is public to the user (as public interfaces are). The component user must supply values for all the parameters in the configuration slots. If a specific parameter is unknown in the actual developed component then this value must appear in the configuration slots of the component.




```
Component comp_name {
  Interface I1 {
      void op1(<parameters>);
      oneway void op2();
      }
  ...
  Input     I1,I2;
  Output O1,O2;
  Consumes event_type1;
  Publishes event_type2;
}
Component implementation comp_name {
  Active p,q,r,u;
  Passive s1,s2;
...
}
Event void event_type1(<parameters>);
Event void event_type2(<parameters>);

    ……
```

Figure 17: Generic component definition

Real-time Constraints. The user can indicate real-time constraints in the components. These constraints can be deadlines, timers, events, etc. and together with the configuration slots mechanism allow for the propagation of the real-time constraints from the most external components to inner components. Finally, the analysis model is only a flat model of threads and resources that can be easily analysed.

6. CONCLUSION

The adoption of new technologies in a market dominated by costs requirements, development time, etc. is a big problem for companies that want to adopt stable and secure technologies. In the area of nuclear power plants Simulators, the developers frequently adopt *ad-hoc* solutions that prevent the reusing of code.

In this paper, the adoption of object-oriented technologies like CORBA and UML has been presented in the developing of simulators for nuclear power plants. These technologies improve the reusing of components allowing for the simulator kernel and the tools to be reused in new simulators.

The utilization of RT-CORBA has been presented in relation to several applications and tools. RT-CORBA has allowed different priority levels in tools, simulation models, etc allowing the building of reusable component for real-time systems.

The entire project was designed using UML like development methodology. Its adaptation was verified in a project developed by a large number of developers located even in different geographical locations.

A new predictable component model for real-time systems has been presented. This model is going to be used in the development of new applications improving the reusing of code.

The simulators are currently being used and there are new projects where the developed components are being reused.

7. REFERENCES

Booch G., Jacobson I. and Rumbaugh J. 1998, “The Unified Modeling Language User Guide”, Addison-Wesley Professional.

Burns A. and Wellings A. 2001, “Real-Time Systems and Programming Languages. 3rd Ed.”, Addison-Wesley.

Díaz M., Garrido D., Soler E. 2003, “Un lenguaje para la definición en tiempo de ejecución de variables físicas asociadas a un simulador”, In *Proc.*

III Jornadas de Programación y Lenguajes, Alicante, Spain, November. Pp. 1-15.

Díaz M., Garrido D., Llopis L.M., Rus F., Troya J.M. 2004, "Integrating Real-time Analysis in a Component Model for Embedded Systems", To appear in *30th EUROMICRO conference*, Rennes, France.

Henning M. and Vinoski S. 1999, "Advanced CORBA Programming with C++", Addison-Wesley Longman.

Levine D.L., Mungee S. and Schmidt D.C. 1998, "The Design of the TAO Real-Time Object Request Broker". In Computer Communications 21. Pp. 294-324, 8.

O'Ryan C., Schmidt D.C. and Wang N. 2000, "Overview of the CORBA Component Model". In Component-Based Software Engineering (Heineman G. and Councill B., eds.), Reading, Massachusetts, Addison-Wesley, 2000.

Schmidt D.C. and Kuhns F. 2000, "An overview of the Real-time CORBA Specification". In IEEE Computer special issue on Object-Oriented Real-time Distributed Computing.

Szyperski C. 1999, "Component Software: Beyond Object-Oriented Programming", Addison-Wesley Longman.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support from the División de Simulación at Tecnatom S.A. who contributed to the creation of this paper.

AUTHOR BIOGRAPHIES

Manuel Díaz received his M.S. and Ph.D. degree in Computer Science from the University of Málaga in 1990 and 1995, respectively. From 1990 to 1995 he was an Assistant Professor in the Department of Languages and Computer Science of the University of Málaga. Since 1995 he has been an Associate Professor in the same department. He has worked in the areas of distributed and parallel programming and in real-time systems, especially in the areas of software engineering for this kind of system. He has published several papers in international refereed journals and representative congresses in the area.

Daniel Garrido received his M.S. degree in Computer Science from the University of Málaga in 1999. From this year he has worked in different public and private projects related to distributed programming and real-time systems, especially in the areas of software engineering. He is currently a Ph.D. candidate at the University of Málaga.

José M. Troya received his M.S. and Ph.D. degrees from the University Complutense of Madrid in 1975 and 1980 respectively. From 1980 to 1988 he was an Associate Professor in that University, and since 1988 he has been a Full Professor in the Department of Languages and Computing Science of the University of Málaga. He has worked on parallel algorithms for optimization problems and on parallel programming and software engineering for distributed real-time systems. He has directed twenty Ph.D. thesis and published more than fifteen papers in international refereed journals.