Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

INFORME TÉCNICO

IGNATIUS
A Tool to Build Supervisory Systems

Silvia V. Benítez, Juan J. Seoane,
Gabriel A. Wainer, Roberto J.G. Bevilacqua.

Report n.: 96-006

Pabellón 1 - Planta Baja - Ciudad Universitaria
(1428) Buenos Aires
Argentina

http://www.dc.uba.ar

Title: **IGNATIUS: a tool to develop Supervisory Systems.**

Authors: **Silvia V. Benítez, Juan J. Seoane, Gabriel A. Wainer, Roberto J. G. Bevilacqua.**

E-mail: **silvia.v.benitez@ac.com, seoane@vnet.ibm.com, {gabrielw,robevi}@dc.uba.ar**

Report n.: **96-006**

Key-words: **Supervisory systems, SCADA, real-time system, software development tools.**

Abstract: In this work we address the design and implementation of a tool to build Real-Time Supervisory Systems called IGNATIUS. An architectural decomposition of all the Supervisory Systems in three service levels has been proposed: Interface Service Level, Particular Service Level and Basic Service Level. The ISL and PSL vary with each particular implementation, but the BSL is common for all of them. IGNATIUS encapsulates the BSL. This decomposition reduces the development cycle, letting the user to concentrate on high level design aspects. This approach allows to build complex SCADA applications reducing development and maintenance cost.

IGNATIUS
A Tool to Build SCADA Systems

Silvia V. Benítez
silvia.v.benitez@ac.com

Juan J. Seoane
seoane@vnet.ibm.com

Gabriel A. Wainer
gabrielw@dc.uba.ar

Roberto J. G. Bevilacqua
robevi@dc.uba.ar

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

ABSTRACT

In this work we address the design and implementation of a tool to build Real-Time Supervisory Systems called IGNATIUS. An architectural decomposition of all the Supervisory Systems in three service levels has been proposed: Interface Service Level, Particular Service Level and Basic Service Level. The ISL and PSL vary with each particular implementation, but the BSL is common for all of them. IGNATIUS encapsulates the BSL. This decomposition reduces the development cycle, letting the user to concentrate on high level design aspects. IGNATIUS provides the user with a complete development environment that can be easily used by inexperienced programmers. This approach allows to build complex SCADA applications reducing development and maintenance cost.

Keywords: Supervisory systems, SCADA, real-time systems, software development tools.

1. INTRODUCTION

At present, the advances and cost reduction in hardware has increased the number of computers used to control physical processes. This led to the automation of processes where manual intervention was a requirement (i.e.: avionics monitoring and control, chemical processes control, traffic light control in a city). This kind of processes need real-time response.

In a real-time system the results not only depends on the logic correctness of the computations but also on the moment when the results are generated. If the time constraints are not accomplished, a system failure occurs leading to catastrophic consequences. Hence, this kind of systems, must guarantee that timing constrains will be accomplished [Wai94].

There are many real-time applications in the industry, and a growing number of computers controlling industrial processes. Process control can be defined as "the exercise of a planed action, for what is considered object, to satisfy particular objectives" [Miy93]. A controller accepts the information coming from sensors, the information is processed, and the results are sent to the controlled object through actuators. The output information affects the controlled object.

In several cases exception conditions occurs, and objectives of the control system cannot be accomplished. To prevent this conditions ensuring correct system response, we can use a set of programs, called Supervisory Systems or SCADA (Supervisory Control And Data Acquisition) systems. To accomplish with these functions, the system must control the plant operation and provide the managers and plant engineers with a snapshot of the process status.

Supervisory systems are designed to coordinate, monitor and service system components. They handle input and output of messages and data, schedule the execution flow, assess priorities between application programs and carry out housekeeping functions. They also can process interrupts and deal with error and emergency conditions. They must be designed to coordinate the functions of the system under varying loads.

Though several changes have occurred in the process control area, most SCADA applications are not flexible enough and have a very limited range of use. With this scenario in mind, our proposal addresses the design and implementation of a tool to build SCADA applications. Several existing semi-formal Design Techniques have been studied, and the tool has been adapted to the selected technique. Also, system software has been selected to provide a complete development environment that can be used by programmers to improve security, maintainability and correctness of the developed applications. The tool has been designed to let the user build a wide variety of supervisory applications with minimum effort.

This work is organized as follows: in section 2 we describe the development tools chosen, in section 3 we describe our approach to build SCADA systems, in section 4 there is a description of the tool components. Finally, in section 5 we show the use of the tool to build SCADA applications.

2. TOOLS SELECTION

The first stage of our work was to select a set of tools to develop our environment. This included the selection of a development methodology, an operating system and a programming language. In this way we can provide a complete development environment that can be integrated with our tool to achieve the goals mentioned in section 1. In this section we will explain the tools we choose and the motivations for these selections.

2.1 Design Methodology

Real-time control applications are usually composed of several tasks executing concurrently at different speeds. Also, tasks that execute in asynchronous mode, need to be synchronized to accomplish functional requirements. Hence, at the design stage of real-time applications, we have to consider additional requirements of those considered by conventional design methodologies.

We analyzed several design methodologies to meet these goals, including: DARTS (Design Approach for Real Time Systems) [Gom84], Structured Development for Real Time Systems [War86], Deutsch [Deu88] and MASCOT (Modular Approach to Software Construction, Operation and Test) [Mas87]. The following essential requirements for real-time applications design have been evaluated [Gom84]:

1. Data Flow Oriented Design.
2. Task Communication and Synchronization.
3. Information Hiding.
4. State Dependency in Transaction Processing.

We decided to use MASCOT as design methodology because it is very simple to use and understand by inexperienced designers and it considers all the requirements mentioned above. It also can be easily integrate with our tool [Wai93] and provides most of the entities provided by the Operating Systems we studied. In this way, a complete development environment to build complex supervisory applications with low cost and high productivity can be constructed.

2.2. Operating System

Several operating systems available in the market have been analyzed. In Table 1 some of their major advantages and disadvantages have been highlighted.

The use of the DOS operating system was not considered, mainly because it has no multitasking capabilities. The same happened with Windows. The main reason to discard UNIX was that often, the scheduling algorithm has no preemptive multitasking. This is an important requirement for real-time applications development. The Windows-NT bad performance led us to the situation of deciding between QNX and OS/2. Though QNX is a widely used real-time Operating System, OS/2 was chosen due to very good performance, its priorities preemptive algorithm and its OOUI features. The lack of SCADA tools for OS/2 when the decision was taken, also encouraged us to select this Operating System.

OPERATING SYSTEM	DISADVANTAGES	ADVANTAGES
MS-DOS	. No multitasking capabilities . Bad use of new processor features (i.e.: 32 bit bus, large amount of RAM, etc.) . Poor memory management . No graphical interface . Few facilities to develop complex concurrent systems.	. Massive acceptance . Easy to use . Low cost hardware configuration required . Full access to hardware capabilities
MS-Windows	. No native Operating System . No multitasking capabilities . Poor memory management . Not stable development platform	. Massive acceptance. Easy to use . Graphical User Interface . Low cost hardware configuration required
UNIX[1]	. No preemptive priorities multitasking (in several versions) . High cost hardware configuration (developed in the version) . Difficult to develop graphical user interfaces	. Multitasking . Widely used in the industry and academy . Low cost hardware configuration required (depending on the version) . Full access to hardware capabilities . Complete security system
QNX	. Not widely used in non-industrial environments . Few development tools . Few software developed . High cost hardware configuration . Difficult portability	. Real-Time oriented . Multitasking . Distributed . Several scheduling algorithms . No swapping . Plenty of timing capabilities . Widely used for control applications
Windows-NT	. Bad performance . Not scaleable . Not stable development platform	. Widely used . LAN oriented . Easy to use . Hardware Abstraction Level . Low cost hardware configuration required
OS/2	. Difficult to configure some devices . High cost hardware configuration . Difficult portability	. Very good performance . Preemptive multitasking with priority management algorithm . 32 bit operation . OOUI (Object Oriented User Interface) . Event driven . Protected mode operation . Interoperability facilities . Large number of tools

Table 1. Operating Systems comparison

2.3 Programming Language

The major characteristics considered for the programming language selection were the low-level coding capabilities. It is also easy to make use of the Operating System features (semaphores, messages, queues, pipes, etc.), as well as the high-level facilities to build user interfaces easily.

Among the different languages evaluated (Pascal, Rexx, C, C++), we found that only C++ satisfies the requirements. It has all the low-level power of the C language, whereas the classes provide high-level facilities. In addition, as an Object Oriented language, C++ provides the mechanisms of Information Hiding, Encapsulation, Inheritance and Polymorphism.

To build the ISL and PSL that will be mentioned later, we used a visual programming tool. VisualBuilder, Dialog and Vispro/C++ were evaluated. Vispro/C++ was chosen because it is the easiest to use and can be fully integrated with IGNATIUS, providing the user a complete development environment.

3. AN APPROACH TO BUILD SCADA SYSTEMS

After analyzing several available SCADAs, we identified three *Service Levels* common to them:

1. *Interface Service Level (ISL)*: this level encapsulates all the programs that provide the interface between the SCADA and the operator. It provides graphical displays, alarms, screen messages, etc. The implementation of this set of programs changes according to the software and hardware platform selected for the development.

[1] Main Unix versions were included for this analysis: Aix, Solaris, SCO, Linux, Minix, etc.

2. *Particular Service Level (PSL):* this level encapsulates all the programs that implement the particular requirements for a specific SCADA. This includes, for instance, alarm assistance routines, event handlers, user command assistance routines, etc. The implementation of this set of programs changes for each SCADA according to the particular service characteristics.

3. *Basic Service Level (BSL):* this level encapsulates all the programs that implement the basic supervision services common to all the SCADAs. In this level we include plant image point internal representation, alarm detection routines, historic storage, message transmission, user command execution, high level task scheduling, process modeling, etc. From a functional point of view, this service level is the same for every SCADA.

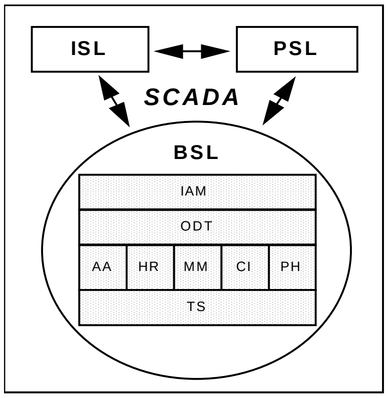

Figure 1. SCADA Service Level Decomposition.

IGNATIUS was built to avoid the development of the BSL services (common to all SCADA) minimizing and facilitating the programming task.

According to the [Kap95] classification, IGNATIUS acts as an interface with a programming language. This type of tool is very flexible though it is a library that can be adapted to control any process of the real world. It is not end-user oriented, but intended to be used by programmers that need to develop specific supervisory applications with minimum effort.

The tool was implemented as a class library using the Client/Server model. IGNATIUS acts as the server, and the ISL and PSL are the clients that vary for each particular supervisory system implementation. Therefore, IGNATIUS can be seen as the same BSL server that provides services to different ISL and PSL clients. IGNATIUS provides three services through the following classes:

- **OBJECT DATA TABLES (ODT):** This is a set of tables that store the data of the different real world objects, including plant image, alarms, I/O ports, mimics, etc.

- **INFORMATION ACCESS METHODS (IAM):** The way to use the data stored in the ODT is through the Information Access Methods. They are implemented as class methods that enable the user to store, read, update and delete the ODT information. The only way to use the ODT is through the IAM.

- **EXECUTION ENGINE (EE):** The Execution Engine is a set of routines that start the system execution. These routines execute concurrently, each in a different thread, synchronized by a high level task scheduler. The following components are part of the EE:

  - High Level Task Scheduler (TS)
  - Alarm Analyzer (AA)
  - Historic Recorder (HR)
  - Mimic Manager (MM)
  - Command Interpreter (CI)
  - Port Handler (PH)

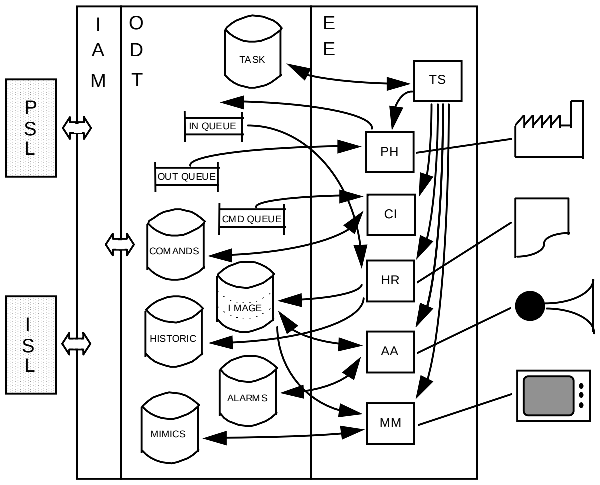

Figure 2. IGNATIUS component interaction.

4. CLASS DESCRIPTION

As previously stated, IGNATIUS is built as class library implementing objects to build SCADA applications. It is composed of the following classes:

4.1. IMAGE

This class represents the plant image point values and the methods to control it. All monitored values must be stored in a central repository implemented with this class. The image is divided into three virtual segments representing integer, analog and digital values. The implementation of virtual segments allows the user to manage the image points without regarding about their internal storage. The class stores the information for each of the image points that it represents. This includes the point value, set-point, alarm status, quantity of updates made, update percentage (for historical recording) and date and time of last update.

There are methods to add, delete, read or update an image point. We also can increase in one the quantity of updates, or modify the status of an image point. Finally, we can return the number of image points stored in the object, return the virtual segment where an image point is stored and acknowledge an alarm status.

4.2. ALARM

This class represents the values of the alarm conditions that must be evaluated by the EE to detect alarm scenarios, and the alarm conditions methods as well. It also links alarm conditions with real world image points. This link allows the user define multiple many-to-many relationships between alarm conditions and image points. The PSL alarm assistance routines may analyze the information of the alarm scenarios detected by the EE to give the required treatment.

The class stores information for each of the alarm conditions it represents, including name, priority, address of an alarm assistance routine, lowest and highest values allowed, evaluation criteria (high, low, rate of changes). It also stores quantity of evaluations made, evaluation frequency (in milliseconds), wait time since last evaluation (in milliseconds), maximum and minimum change rate allowed and time of last update.

The relations between alarm condition and image point include alarm id, image point id and image point virtual segment location.

The associated methods let add, update and delete alarm conditions, read conditions. It also allows to increment the number of times the condition was evaluated, calculate lowest and highest values based on a set-point value and a tolerance percentage. Finally, it lets search the alarm condition linked to an image point, search the image point linked to an alarm condition or link and alarm condition with an image point.

4.3. QUEUE

This class represents the messages sent from the SCADA to the physical devices (**In Queue)** and the messages sent from the physical devices to the SCADA (**Out Queue)**. It also represents the methods associated to these messages. It is used as the communication channel between external control elements (such as PLCs) and the SCADA.

4.4. HISTORIC

This class reads the In Queue messages and stores historical information of the image point included in the message, based on an update parameter specified by the user. This class lets the user retrieve information about the different values of an image point during a period, allowing the user to analyze this information and detect changes or problems in the environment. The update ratio lets the user specify the rate of historic recording for each image point.

4.5. TASK

This class synchronizes the execution of the different components of the EE. Its use can be extended to synchronize user written routines (like a routine to refresh data on the screen). It stores all the information of the task components of the EE. The High Level Task Scheduler (TS) is responsible to synchronize the other components using system semaphores. To do this work the class stores name, process id, execution frequency in milliseconds, wait time in milliseconds since last execution, and the event handler.

4.6. PROCESS

This class models a plant process, i.e.: drying oven process, evaporation plant process, chemical reactor vessel, etc. This information can be used by the ISL to show process data on the screen.

4.7. MESSAGE

This class encapsulates the messages sent by the class MIMIC to the SCADA through a system pipe. Each message contains information about an image point and its attributes on the screen: position, color, etc. This information is used by the ISL to display the image points of the process in display.

4.8. PIPE

This class implements, through system pipes, the communication channel between the MIMIC class and the SCADA for the message delivery. The MIMIC class writes MESSAGEs in the PIPE while the ISL read this messages to display the image point information on the screen.

4.9. MIMIC

This class links real world processes with image points and their screen attributes: position, foreground color, background color, etc. The Mimic Manager (MM) writes the attributes of the points belonging to the process to display in the PIPE. This is done with a frequency defined by the user.

The classes MESSAGE and PIPE are part of this class. They are used for implementation purposes and serve as the communication vehicle between the MM and the ISL.

4.10. PORT

This class handles the I/O ports that are used to connect the computer to the physical devices (especially control industrial devices such as PLCs or other controllers). The class allows to link physical devices with image points letting the user to define multiple many-to-many relations between them. To do so, uses the In and Out Queues. It also initializes sets the I/O ports (baud rate, parity control, data bits, etc.).

4.11. COMMAND

This class provides a mechanism to define and manage commands entered by the operator. It also validates the commands and parameters entered to determine if they are correct. In this approach, the command definition, handling and validation are separated from the command execution. While this class encapsulates the former functions, the command assistance routines must be implemented in the PSL. The command assistance routines execution can be accomplished in two different ways: scheduled or immediate. In the first case, the command entered is added to a command queue and executed once all the previous commands in the queue have finished their execution. In the second case, the command entered is executed immediately without waiting for the commands in the queue to finish their execution.

5. BUILDING SCADA APPLICATIONS

To test the tool and show its usage we built two different SCADA systems.

The first one, shows the usage of the tool to build a customizable SCADA. This SCADA lets the operator dynamically configure the resources: define image points, alarm conditions, processes, physical devices, screen representations, update the image point values, alarm conditions, etc. The flexibility given by the dynamic configuration, allows us to use of the SCADA for any particular purpose implementation.

The second shows the usage of the tool to build a single purpose SCADA. In this case, the resources are statically defined. The operator has no capability to dynamically configure resources.

In both examples we considered the requirements of a drying oven process application [Ben93]. The components are dried by being passed through an oven divided in three sections: preheat, drying and cooling. The components are placed on a conveyor belt that conveys them slowly through the drying oven. The oven is heated by three gas-fired burners placed at intervals along the oven. The temperatures in each of the areas heated by the burners is monitored and controlled. An operator console unit enables the operator to monitor and control the unit operation. The system is controlled by a hardwired control system. The requirement is to provide supervisory tools for the control system.

Based on these requirements, we made a system design using MASCOT and we implemented it using IGNATIUS and the visual programming tools stated earlier.

Example 1 let us measure the utility of the tool to build a complex SCADA. It also allwed us to determine the effort required to build a SCADA using all the facilities of the tool. Example 2 let us measure the utility of the tool to build a simple SCADA and determine the minimum effort required to build a SCADA using the tool.

We tested the examples sharing the CPU with other applications and changing the SCADA execution mode to background. The SCADA response time was always within the expected time frame due to the Operating System scheduling algorithms and the design of the TS. In this scenario, the priorities preemptive scheduling algorithm avoids the excessive use of the CPU by processes that could delay the execution of the EE. The EE routines were scheduled within the TS with different intervals. In a first stage, these routines were executed with intervals greater than the second.

With an average CPU load, the expected response time was accomplished. In the last stage, the minimal interval was obtained to let the system respond within the expected time frame considering an average CPU load (see Table 2).

EE routine	**Minimal Interval with average CPU load (in seconds)**
Alarm Analyzer (AA)	0.15
Historic Recorder (HR)	0.45
Mimic Manager (MM)	0.35
Command Interpreter (CI)	0.65
Port Handler (PH)	0.55

Table 2. EE Minimal Interval

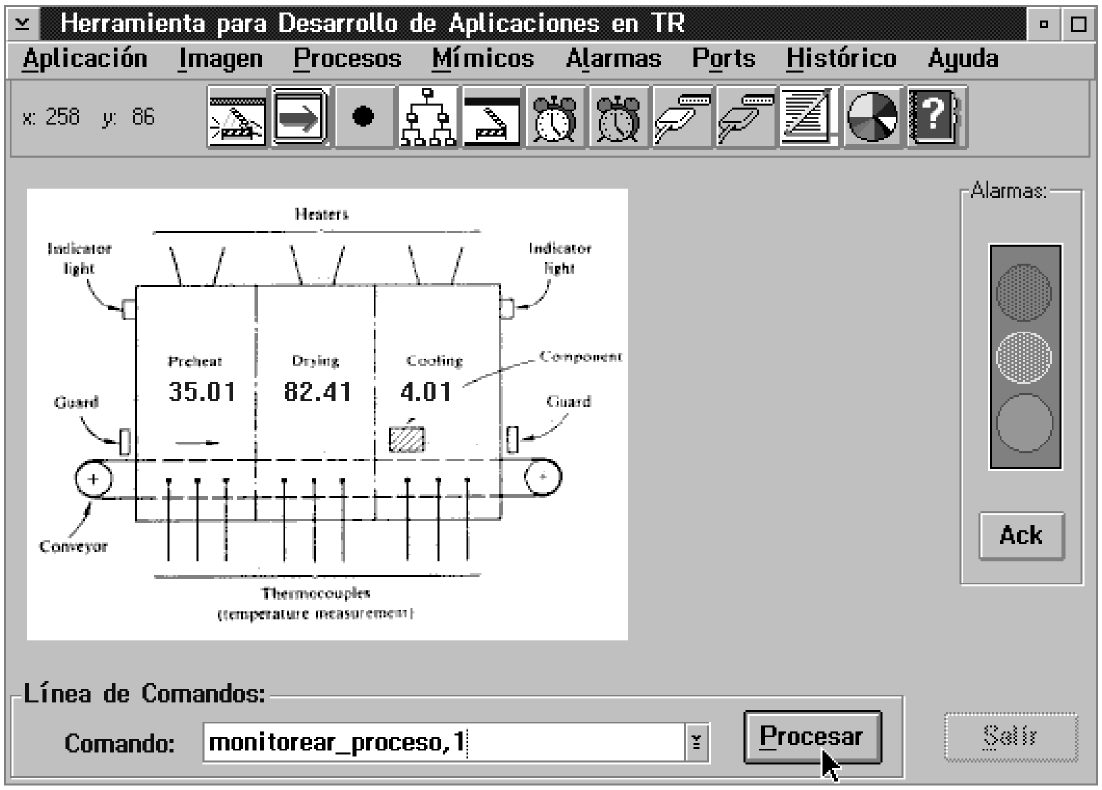

Figure 3. Example 1 - Main Window.

This lets the user execute other tasks while the SCADA is running, sharing the CPU without letting the SCADA off line. For instace, we can analyze historical records, draw or scan new processes, launch an Operating System session, add alarm conditions, processes, mimics, etc.

The time expended to design, code and test the examples was minimal. A total of 120 man hours was used for the example 1, while only 24 man hours were used to develop the example 2. These time values clearly reflect the tool usefulness and the development cycle time reduction, reducing also the complexity and letting the user concentrate on high level design aspects.

The development effort is proportional to the complexity of the particular problem environment. As we could see in our examples, the effort is minimum for medium complexity environments.

6. CONCLUSIONS AND FUTURE DIRECTIONS

In this work we have presented a tool to provide an integrated and flexible development environment to develop SCADA applications.

Though several control techniques have not changed much since the beginning of the use of computers for process control, process supervision was involved in several changes. In most cases the applications are not flexible enough: they can have a very limited range of use, or its complexity is excessive for simple control applications.

We could see that all the supervisory systems can be composed of three levels. We have encapsulated the Basic Service Level in the tool allowing to have a flexible tool to develop general and particular purpose SCADAs.

The utility of the tool and the effort required to built SCADAs with different complexity requirements was determined. Performance tests have been applied to both examples to check the response time constrains for different EE task intervals.

The Object Oriented technology simplified the design and development of the tool. Code error detection was also easier because of the data and related code encapsulation. This paradigm also provided reusability and simplified maintenance characteristics.

The OS/2 Warp 3.0 Operating System provided a stable development environment. The protected mode execution avoids system crashing due to involuntary errors introduced in the code, which is very important for this kind of applications. Multitasking facilities increased the productivity during the development cycle.

OS/2 also isolates the hardware level from the application software level. Applications cannot operate directly with the physical devices: the device drivers must act as an interface between the application programs and the hardware. While this isolates the application software from physical device characteristics, it makes it dependent upon the type of device driver loaded to handle the device (i.e.: the result of a console write operation may change depending if the device driver is ANSI or not). Some devices are available to the application programs only if the appropriate device driver has been previously installed (i.e.: the serial port cannot be opened by an application until a communications driver is loaded).

We could see that the proposed decomposition in three service levels and the use of the tool reduces any SCADA development to the building of the ISL and PSL, shortening the development cycle and reducing the complexity.

The tool also lets the user concentrate on high level design aspects, providing a complete development environment that can be easily used by inexperienced programmers, helping them to maintain the integrity of the system, make changes and develop correct applications.

At present we will start to use that tool in the Real Time Systems course in our Department. It also will be integrated with other projects of the department where a previous version of the tool is being used [Wai93].

Several new developments can be faced in the future as an extension to our work:

1. Migration of the tool to other operating systems.
2. Design and implementation of a supervisory system using a distributed Client/Server architecture. The BSL will reside in the server and the ISL and PSL will reside in the client.
3. Design and implementation of the BSL as a single server (single data base) providing services to different supervisory systems.
4. Addition of interfaces to different industrial PCs and PLCs.
5. Test of the tool in rough application environments (including its embeding in ROM).
6. Develop other supervisory applications: electronic worksheets, statistics generator, etc.

7. REFERENCES

[Ben93] BENNET, S. “Real-Time computer control: an introduction”. Prentice-Hall International. 2ND Edition, 1993.

[Ben96b] BENITEZ, S.; SEOANE, J.; WAINER, G.; BEVILACQUA, R.J.G."Development and implementation of a tool to build supervisory systems". (in Spanish). M.Sc. Thesis. Computer Sciences Department. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1996.

[Deu88] DEUTSCH, M. “Focusing real-time systems analysis on user operations”. IEEE Software, September 1988. pp 39-50.

[Gom84] GOMAA, A. “A Software Design Method for Real-Time Systems”. Communications of the ACM, September 1984, pp 938-949.

[Kap95] KAPLAN, G.; HOUSE, R. “Data Acquisition Software for Engineers and Scientists”. IEEE Spectrum, May 1995. pp. 23-39.

[Mas87] MASCOT. “The official handbookof MASCOT, version 3.1”. Computing Division, RSRE, Malvern. 1987.

[Miy93] MIYAGI, P.; PEREIRA RIBEIRO BARRETO, M.; SILVA, J. “Domotic: Control and automation” (in Portuguese) Vol II, VI EBAI. 1993.

[WAI93] WAINER, G. “SSDT: a tool to develop Real Time Supervisory Systems ” (in Spanish). Proceedings of the Jornadas Chilenas de Ciencias de la Computación, October 1993. pp. 44-52.

[Wai94] WAINER, G. “Introduction to the development of Real Time Systems” (in Spanish). On Line publication, http//zorzal.dc.uba.ar/pub/materias/str/libro, 1994.

[War86] WARD, J.; MELLOR, P. “Structured development for real time systems”. Yourdon Press, 1986.