IGNATIUS: A TOOL TO BUILD SCADA SYSTEMS

Silvia V. Benítez sbenitez@vnet.ibm.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
Pabellón I - Ciudad Universitaria
(1428) Buenos Aires - Argentina

ABSTRACT

In this work we address the design and implementation of a tool to build SCADA Systems called IGNATIUS. An architectural decomposition of all the Supervisory Systems in three service levels is proposed: Interface Service Level, Particular Service Level and Basic Service Level. The ISL and PSL may vary with each particular implementation, but the BSL is common for all of them. IGNATIUS encapsulates the BSL, reducing the development cycle of SCADA tools. 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, software development tools.

1. INTRODUCTION

At present the advances and cost reduction in hardware have increased the number of computers used to automatically control physical processes. Many control applications are in the industry, and a growing number of computers controlling industrial processes.

In several cases, exception conditions in the controlled system occurs, and the automatic control of the desired process fails. To prevent these conditions, and to ensure correct system response, Supervisory Systems can be used.

Complex supervisory systems (sometimes called or SCADA, Supervisory Control And Data Acquisition) must be 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 process interrupts and deal with error and emergency conditions. They must be designed to coordinate the functions of the system under varying loads. They also must provide the managers and plant engineers with a snapshot of the process status.

Several SCADA applications are not flexible enough and have a very limited range of use. The proposal here presented addresses the design and implementation of a tool to build SCADA applications. Several existing semi-formal design methodologies have been studied, and the tool has been adapted to the selected technique. Also, a stable development environment system has been selected to provide an adequate tool that can be used by inexperienced 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.

2. AN APPROACH TO BUILD SCADAS

After analyzing several available SCADAs, three *Service Levels* were identified as common to all them:

1. *Interface Service Level (ISL)*: this level encapsulates the routines providing an 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.

2. *Particular Service Level (PSL):* this level encapsulates the routines 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. For instance, it can 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.

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

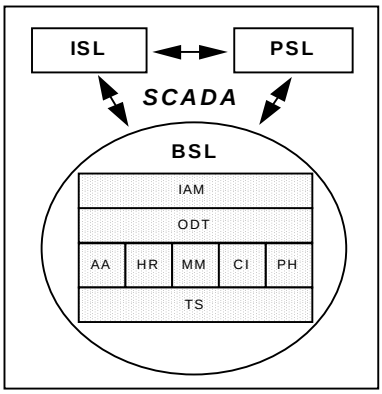

Figure 1. SCADA Service Level Decomposition.

According to the [1] classification, IGNATIUS acts as an interface with a programming language. This type of tool is very flexible though it 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 Client/Server class library. IGNATIUS acts as a 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 (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 main components of the EE are the High Level Task Scheduler (TS), the Alarm Analyzer (AA), the Historic Recorder (HR), the Mimic Manager (MM), the Command Interpreter (CI) and the Port Handler (PH).

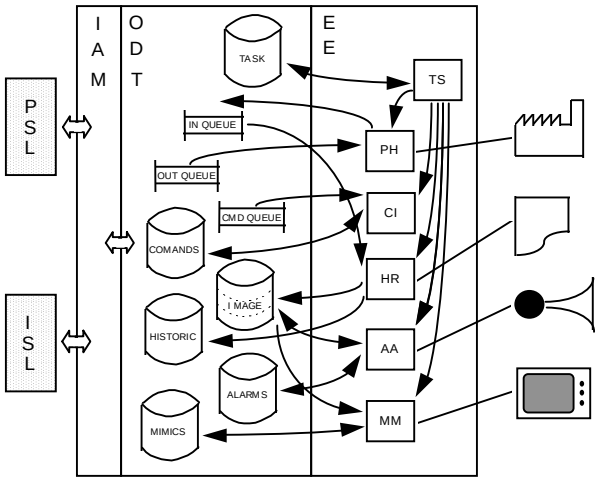

Figure 2. IGNATIUS component interaction.

The tool was designed to be coupled with most existing semiformal real-time design techniques, but specially tailored to MASCOT [2]. Several operating systems were analyzed, and

OS/2 was selected. The class library has been built using C++, and Vispro/C++ was used as graphical front-end. To obtain further information about these decisions, see [3,4].

3. CLASS DESCRIPTION

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

1. IMAGE: This class is used to represent the plant image points. The image is divided into virtual segments, allowing the user to manage the image points without regarding about their internal storage. It is used to keep point value, set-point, alarm status, quantity of updates made, update percentage (for historical recording), last update time. The methods allow to add, delete, read or update an image point, count updates, set up the number updates, or modify the status of an image point. Alarm states for each point can also be obtained.

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. The class keeps information for each of the alarm conditions it represents.

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. 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.

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.

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.

7. MESSAGE: This class encapsulates the messages sent by the class MIMIC to the SCADA. 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.

8. PIPE: This class implements 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.

9. MIMIC: This class links the plant 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.

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 relations between them. To do so, it uses the In and Out Queues. It also initializes sets the I/O ports (baud rate, parity control, data bits, etc.).

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.

4. BUILDING SCADA APPLICATIONS

To test the tool and show its usage, different SCADA systems have been built. The tool has been used as an educational tool. In both examples we considered the requirements of a drying oven process application. First, the requirements of this control application were considered. Then, the MASCOT methodology was used to specify a design document. Finally, the SCADA was implemented following the design specification, using IGNATIUS and the visual programming tools stated earlier.

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 first example allowed to test the utility of the tool to build a complex SCADA systems. It also served to determine the effort required to build a SCADA using all the facilities of the tool. The second example allowed to measure the utility of the tool to build a simple SCADA and determine the minimum effort required to build a SCADA using the tool.

The priorities preemptive scheduling algorithm of the chosen Operating System, and the high level scheduling policy chosen for the TS (Rate Monotonic [5]), avoided 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, the routines were executed with intervals greater than the second and the expected response times were accomplished. The system was tested, and the minimal periods were obtained, providing more accurate responses (see Table 1).

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 1. EE Minimal Interval

The use of the tool allowed to reduce the time expended to design, code and test the examples. A total of 120 man hours was used for the first example, while only 24 man hours were used to develop the second example. These time values clearly reflect the tool usefulness and a reduction in the development cycle time, lowering the complexity and letting the user concentrate on high level design aspects, mainly considering that both systems were developed by groups of undergraduate students.

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.

5. CONCLUSIONS AND FUTURE WORK

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

Supervisor systems are are usually not flexible enough to support several application: they can have a very limited range of use, or its complexity is excessive for simple control applications. To avoid these problems, the main features of supervisory systems were identefied, and divided in three different levels. The Basic Service Level was encapsulated in the present tool, allowing to have a flexible environment to develop general and particular purpose SCADAs.

The utility of the tool and the effort required to built SCADAs with different complexity requirements was tested. Performance has been tested to check the response time constrains for different EE task intervals.

The proposed decomposition in three service levels, the use of Object Oriented techniques, the provision of a stable develpment environment, and the use of IGNATIUS, reduced the SCADA development cycle to the building of the ISL and

PSL, shortening the development times 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. It can help the programmers to maintain the systems, make changes, and develop correct applications. This could be checked by using the tool with educational purposes in undergraduate programs. The tool will also be integrated with other projects where another tool [6] is being used at present.

6. REFERENCES

[1] G. Kaplan and R. House. Data Acquisition Software for Engineers and Scientists. *IEEE Spectrum*, (5),1995. 23-39.

[2] MASCOT. *The official handbookof MASCOT, version 3.1*.(Computing Division, RSRE, Malvern. 1987).

[3] S. Benítez, J. Seoane, G. Wainer and R. Bevilacqua. 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.

[4] S. Benítez, J. Seoane, G. Wainer and R. Bevilacqua. IGNATIUS: a tool to develop Supervisory Systems. *Technical Report 96-006. Departamento de Computación, FCEN/UBA* . 1996.

[5] J. Lehoczky, L. Sha and Y. Ding. The Rate Monotonic Scheduling algorithm - exact characterization and average case behavior. *Proceedings IEEE Real-Time Systems Symposium*. CS Press, Los Alamitos, Calif. 1986.166-171.

[6] G. Wainer. SSDT: a tool to develop Real Time Supervisory Systems (in Spanish). *Proceedings of the Jornadas Chilenas de Ciencias de la Computación*, 1993. 44-52.