A NEW ALARM SYSTEM PROCESSOR FOR ELETTRA

Paolo Michelini, Claudio Scafuri,
Sincrotrone Trieste, Padriciano 99, 34012 Trieste, Italy

ABSTRACT

The new ELETTRA alarm system is a set of programs used to log alarm events, report them to control room operators and feed them to analysis programs. The design of the new alarm system is based on an object oriented distributed architecture with an extend client/server communication model; moreover it has the added constraint of supporting the old RPC based alarm delivery system. The new system, written almost entirely in C++, shows that the object oriented paradigm is a natural choice for building event driven programs and that it is possible to encapsulate RPC based servers into C++ classes. The Motif graphical user interface is also written in C++ and is based on the well known concept of component by Douglas Young.

1 ALARM SYSTEM

1.1 Definition of alarms

Alarms are events which are triggered when a set of predefined conditions is satisfied. Generally the conditions to generate an alarm are related to malfunctions, anomalous conditions or potentially hazardous situations for the personnel or plant.
Alarms can thus be used to start some automatic safety procedure and to alert the human operators. In our system the first task is autonomously carried out by the interlock system [1], the second is handled by ELETTRA Alarm System.

1.2 Overview of the ELETTRA Alarm System

The ELETTRA Alarm System is integrated in the ELETTRA Control System [1]; alarms are generated at the lowest layer (EIU) of the system by an alarm processor, a task which periodically checks a set of field variables. The alarm notification is sent to the intermediate layer through the MIL1553b field bus [2]. Here an appropriate process formats the alarm notification into a standard alarm message, and notifies the alarm server by means of a remote procedure call (RPC) based on the cern/nc package. The alarm server is a task running on the upper layer of the Control System, which collects and processes the various alarms.

2 THE NEW PROJECT

2.1 Goals an constraints

The project for a new alarm system started with the following goals:
- robustness;
- responsiveness;
- clear, Motif compliant user interface;
- open architecture for future expansions;
- support for many contemporary users;

The new project had to respect the constraint of using some of the existing components:
- lower layer system;
- RPC interface;
- configuration files.

2.2 Functionality and architecture

The purpose of alarm system is to alert the control room, either human operator or a diagnostic system, of the occurrence of alarms and to manage a list of the active alarms. An alarm becomes active when an alarm message is detected. It becomes inactive after a re-entry message for that alarm is detected and it has been acknowledged from the control room. All these transitions must also be logged into an history system.

A distributed architecture has been chosen, with an extended client/server communication model. A unique server manages the list of active alarms in order to guarantee the correctness and coherency of the information. The list of alarms is altered by the server in response to external events:
- alarm messages (RPC) coming from the field;
- acknowledge request messages from client programs;

Incoming alarm messages are also checked against a configuration file, containing the list of legal alarms and their characteristics.
Anytime there is a change in the alarm list the server notifies the event to each of the connected clients. and writes a record into a log file. The communication from server to clients is done via TCP/IP streams. The alarm server exports also an auxiliary RPC for reading the complete alarm list. The server capability of sending unsolicited message to its clients is an extension of the traditional client/server model of distributed computing.

The client is a Motif program which displays the alarm notifications generated by the server. It is also known as the Alarm_interface. The main window displays the synthetic descriptions of each alarm; the operator can request more information about an alarm to be displayed in a secondary window. These information is extracted

from the alarm configuration file and from the actual value of the alarm. In addition to the visual notification the alarm interface generates also an audio signal to attract the operator attention. The audio signal remains on until it is turned off explicitly or by requesting an alarm acknowledge. It is possible to run many Alarm interfaces concurrently, because the alarm server will maintain the alarm list correct.

3 SOFTWARE DESIGN

3.1 General methodologies

The project has been designed keeping in mind some of the principles of object oriented programming [3]. Both the client and the server have been decomposed into a series of separate entities (objects) which interact by means of messages. The reception of a message triggers the execution of some procedures, known as methods, inside the object; some of these methods will themselves send an appropriate message to other objects.
In this way the program is completely event driven, ensuring a fast response when an alarm must be processed and a negligible waste of processing resources when there are no events to handle.
The programming language used for the development is mainly C++[4] with some modules written in ANSI C .

3.2 Server structure

The server program is composed of the following objects:
- AlarmServer: receives and validates alarm messages from the field.
- AlarmConfiguration : parses the alarm configuration file and returns information about the various alarms
- AlarmStatus: manages the list of active alarms, also handles the requests of acknowledge.
- AlarmLog: manages the writing to disk of the all the relevant events.
- Talker: handles connections with clients, sends notifications of events to clients.

This organisation is fully modular, and allows us to modify the internal structure and functionality of each object provided that its external interface is not changed. For example, it would be very easy to modify the internal structure of the AlarmLog object to store the events into an external database system instead of the local disc file. The programming effort would be directed only to the modification of the AlarmLog object, all the other client objects would remain unchanged.
The AlarmServer object is particularly interesting, because it encapsulates the functionality of a traditional RPC server. Although the cern/nc rpc library is designed for ANSI C we have found that with some small modifications the automatically generated stubs can be easily used inside a C++ class. If further integration of RPC services into C++ objects will be needed, we think that it will be possible to modify the cern/nc package to generate directly C++ compatible code.
Some of the events to which the client reacts are generated outside the main program and must be reported to the appropriate internal objects. In our case all the external events are in the form of incoming network messages. A simple event loop based on the well known select() UNIX call fulfils this task and is the core of the main function of the server.

3.3 Client structure

The client program is subdivided into the following objects:
- Listener: this is the companion of the client Talker object. It opens the TCP connection with the server and handles the notifications sent by the client.
- AlarmConfiguration: this object is identical to the one used in the server program.
- FanIn: this object extracts the notifications from the Listener objects and distributes them to object of the class FanOut through UNIX pipes. In practice it works as a sort of multiplexer. It has the additional task of sending the acknowledge requests to the server.
- FanOut: implements the receiving end of the pipe, where the messages are used.

With this scheme it is possible to share the services, that is notifications and acknowledge requests, of one server among many client modules, using the minimum number of TCP and RPC connections.
In the current alarm client the objects sharing the messages extracted by FanOut objects are:
- AlarmBeeper: handles the generation of the audio signal whenever there is an alarm occurrence.
- AlarmUI: handles the graphical representation of alarms in a Motif [5] window. This object has been designed according to the "User Interface Component" methodology by D. Young[6] and uses the "Matrix" widget from Bellcore [7], which has been put in the public domain. Internally it handles the list of displayed alarms with a linked list data structure.
- ShowDetail: another "User Interface Component" used to show detailed information about one alarm. It reads and formats information from the AlamUI and AlarmConfiguration objects.

3.5 User Interface Component

The "User Interface Component" methodology is a technique to use Motif or other X11 "widgets" inside C++ classes to be used as the building blocks of the graphical user interface of a program. The idea is not to encapsulate directly the native widgets, but rather to write classes containing one or more widgets to extend and customise the behaviour of the widget library. This technique exploits some very useful characteristics of the X11/Motif environment to improve the easy of use of components; for example we can use the resource mechanism to set the

values of some internal variables (properties) of our components at creation time. Although these operations can be done "by hand" using low level X11 library functions, the use of the predefined UIComponet class as the parent of our class makes this task easy and straightforward.

With this technique the need to write custom widgets [8], a very difficult and time consuming task, is almost eliminated.

4 CONCLUSIONS

We have seen that an object oriented design technique implemented with an objected oriented language like C++ is a good basys for writing an event driven distributed system, specially if a good collection of base classes is available; in our case the UIcomponet class. These techniques require more emphasis on the analysis and design phase of the project, which must be complete and accurate; on the other hand the implementation part is be easier, since the use of standard classes, available with almost any commercial C++ implementation, relieves the programmer from writing again and again the basic algorithms and data structures. The time needed to complete the project has not been substantially shorter than if we had used a traditional approach, but we were able to produce a very robust system, with satisfactory performances which, up to now, showed very few problems. Moreover, the classes and structures we have produced will allow us to extend the system functions easily.

REFERENCES

[1] D. Bulfone, "Status and prospects of the ELETTRA control system"; Nuclear Instrumentation and Methods in Physics Research A 352 (1994) 63-66 North Holland

[2] D. Bulfone et al. "The ELETTRA Field Highway System"; Proc Int. Conf. on Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japan, 1991.

[3] P. Coad, E. Yourdon, “ Object - oriented Analysis”; Prentice-Hall, 1991.

[4] B. Stroustroup, "The C++ Programming Language"; Addison-Wesley , 1987.

[5] D. Heller, "Motif Programming Manual"; O'Reilly & Associates,1992.

[6] D. Young, "Object-Oriented Programming with C++ and OSF/Motif"; Prentice-Hall, 1995.

[7] A. Wason, "XbaeMatrix(3x)"; Bellcore inc. 1995. (availabe in the public domain distribution of the XBAE Widget Set).

[8] D. McMinds/J.P. Whitty,"Writing Your Own OSF/Motif Widgets"; Prentice-Hall,1995.