INTEGRATION SOFTWARE FOR AN EDUCATIONAL VIRTUAL LABORATORY SUPERVISING FLEXIBLE MANUFACTURING CELLS

Daniela SARU[*], Sergiu Mihai DASCALU[**], Aurelian Mihai STANESCU[*], Adrian PETCU[*]

** Department of Control and Industrial Informatics, Faculty of Automatic Control & Computers, University Politehnica of Bucharest, Sp. Independentei 313, Bucharest, Romania*

*** Department of Computer Science, University of Nevada, Reno, 1664 N. Virginia St. Reno, NV, USA*

Abstract: *Creating and managing virtual enterprises represents the best modern approach for development and training in most business domains. Building such complex entities requires collaboration and integration of diverse hardware systems and software applications. Training software specialists and enterprise managers to design or simply choose the appropriate architecture is a very important and demanding educational task. This paper describes research and development aspects of building a software system for remote supervising and control of a virtual enterprise module. Conceived as a heterogeneous supervisor/control system and using Windows™ and QNX operating systems as main support and CORBA technology as integration tool, this software system is currently used as part of a complex virtual laboratory developed at the "Politehnica" University of Bucharest. The system's main purpose is to allow students at various remote locations to experience mechanisms specific to real-time operating systems, computer-integrated manufacturing, and control systems. By using Internet connections and installing simple client applications, students from other universities can also have access to the virtual enterprise module.*

Keywords: *e-learning, virtual enterprise, flexible manufacturing cell, client/server, distributed heterogeneous systems, middleware, CORBA, real-time, QNX.*

1. INTRODUCTION

Virtual enterprises can be viewed as distributed heterogeneous systems. Building such complex entities requires collaboration and integration of various hardware systems and software applications. From this perspective, it is very important to rely on a well designed IT infrastructure that offers capabilities such as scalability, interoperability, legacy application integration, and so forth [1]. A professional solution for this problem is to use an object-oriented middleware technology accepted by most IT hardware/software vendors [2]. Training software specialists and enterprise managers to design or simply decide on the appropriate architecture is a very important and demanding educational task. A virtual laboratory that allows remote control of various virtual enterprise modules represents a very useful and highly practical e-learning tool [3].

While working as the software integration team for the World Bank funded research project "FABRICATOR" at the Human Resource Training Center of the "Politehnica" University of Bucharest, the authors of this paper had the opportunity to design, implement, and test integration mechanisms and complex software components for a virtual enterprise pilot.

One of the main modules of this pilot was the DEGEM 2000 Flexible Manufacturing Cell (FMC), a training tool for students and enterprise employees [4]. The present paper describes several research and development aspects of building a software system for remote supervising and control of this cell. Conceived as a heterogeneous supervisor/ control system that uses Windows™ and QNX operating systems as main support and CORBA technology as integration tool [5], this software system is currently used as part of a complex virtual laboratory at the "Politehnica" University of Bucharest. Specific mechanisms of real-time operating systems, computer-integrated manufacturing and control systems can be experienced by students from remote locations. By using a web video camera for tracking all FMC movements the system allows geographical distributed participants to visualize the effects of different commands. The system, by its design, also guarantees conflict-free usage of the FMC.

2. SUPERVISOR SYSTEM ARCHITECTURE

Enterprise Application Integration (EAI) encompasses not only new software products but also old yet operational and still very important software systems referred to as legacy applications. Because we were dealing with such an application, we chose to integrate the FMC into the virtual enterprise pilot by wrapping the old software supervisor system into a CORBA interface [1].

The old structure of the control system has three main modules, as shown in Fig. 1. The module on the right-hand side of the figure is the ladder diagram of the Programmable Logic Controller (PLC). The ladder diagram specifies the elementary movements inside a working post of the flexible manufacturing cell. Each elementary movement is initiated by a particular state variable that can be set in the PLC by the computer. These computer-controlled state variables provide the interface for the communication link between the computer and the PLC.

The second module of the old control system is the communication driver. The presence of both a dedicated module and a distinct communication process between the PLC and the computer is justified by maintenance reasons. If one wants to use another PLC type the only necessary adjustment one has to make involves writing a new driver. The communication driver's task is to send commands from the user to the PLC. These commands specify actions that the equipment can perform and must be sent as messages that the PLC can understand. Consequently, the driver must create messages conformant to the PLC data format rules.

The third module is the graphical user interface, installed on the same computer as the communication driver. This module offers the user the possibility to specify commands to be sent through the communication driver to the PLC.

Designed to be installed in the vicinity of the FMC the old software control system was running under the QNX Real Time Operating System (QNX RT-OS) on a PC compatible machine [6].

The main goal of our recent work was to build a new, network- and Internet-oriented, TCP/IP-based remote control software system for the flexible manufacturing cell. The new system has two main components: a client component, *Command Module*, designed to be installed on Windows™ platforms, and a specialized server module that wraps the old software control system (bottom-right part of Fig. 2). The server module is working with the QNX operating system as support and is installed on the same hardware platform as the controlled process.

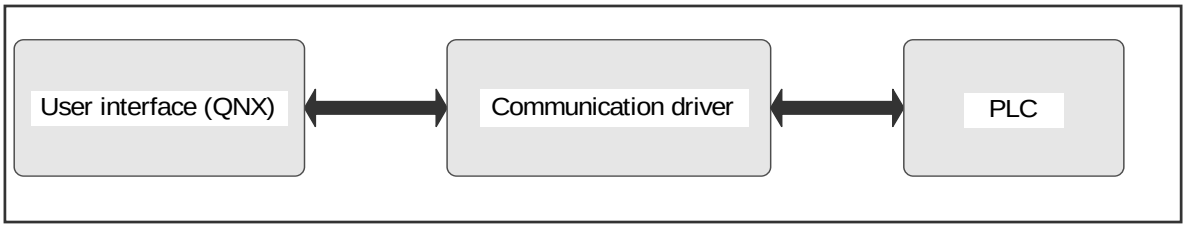

Fig. 1. Old control system architecture

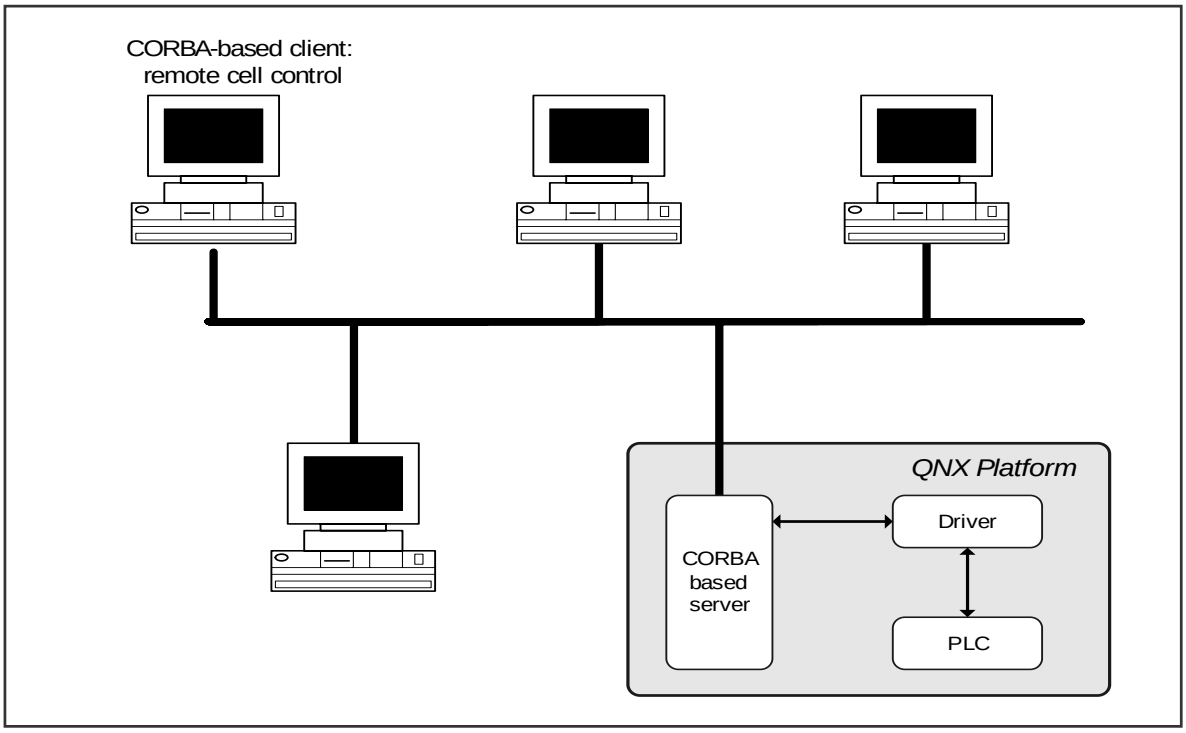

Fig. 2. New control system architecture

Client/server communication [7] uses CORBA technology, a relatively new middleware standard that offers excellent interoperability capabilities for heterogeneous distributed software applications [2]. Such a heterogeneous distributed architecture offers the possibility to install an industrial FMC in dangerous or noxious environments while the *Command Module* and the *Graphical User Interface Module* can be placed in a separate remote and safe room. Both the cost and the usability of this architecture are very attractive. By installing only client modules on different campus platforms and by using the Internet, several universities can access the same FMC for e-learning activities.

In order to avoid the intensive use of the FMC during the testing phase of the software system's life-cycle we created and worked with a Windows server simulator instead of the real QNX server. This solution not only allowed us to focus on client module issues concerning technical and ergonomic aspects of the system but also provided us with a practical testing tool that can be re-used for future developments, as various new client module types are likely to be added later.

3. REASONS FOR USING CORBA AS THE INTEGRATION TOOL

During the last decade, OMG's CORBA standard specifications were accepted by the most prominent software vendors as the performing client/server communication and application interoperability tools of choice [8].

Since 1991 complex software utilities based on the CORBA standard were built for a wide range of different hardware platforms, operating systems, and programming languages. Standardization brings the significant advantages of application compatibility, portability, and interoperability. By using CORBA, a customer can build a system using applications delivered by different vendors, according to specific business requirements [9].

Choosing a product necessarily involves considering its performance, price, and licensing. From this perspective, ORBACUS/E, an Object Oriented Concepts (OOC) product, offers enhanced flexibility [10]. At the time we chose to use it there were available free downloadable versions for non-commercial purposes for Microsoft Windows and many UNIX/Linux flavors, all of them designed to be used on embedded real-time systems running on operating systems such as QNX. Importantly, Orbacus/E was answering appropriately the strict time response and memory size requirements of such systems, including the specific requirements of our system. Also, this novel integration tool was distributed open source for C++ and Java, which suitably fitted our needs.

4. APPLICATION REQUIREMENTS

The analysis phase of the software's development cycle has revealed several major requirements for the system, as follows.

Firstly, the server calls need to be very fast. The server's final version is intended to run on a QNX platform, a real-time operating system, thus it must conform to strict response time requirements that guarantee correct and efficient FMC functionality.

Secondly, another important requirement is to ensure a small network data traffic amount during method calls. Because the server state is inspected at regular time intervals, the number of server polling sessions can be high enough to generate intensive network traffic. The human user can modify the number of server polling sessions in order to obtain more accurate state information. Additionally, using the TCP/IP protocol as a CORBA communication support, it is no longer mandatory to place both the server and the client in the same LAN (Local Area Network) – they can be placed anywhere in the Internet. This option can lead to communication problems related to traffic bandwidth fluctuation in the network, thus the information sent and received through the network during method calls must be very concise.

Third, besides the minimum data necessary for command and state visualization, a potential client also needs time and synchronization information, stored and managed in a simple data base. Such data can be used for tracking the usage of the FMC or to allow a statistical analysis of the tool. Remote access option allows multiple clients to concurrently access the FMC. Avoiding unauthorized access as well as conflicting resource usage is another prominent requirement of the software.

Finally, as an FMC command module, the client application must include at least two functions: *visualize* and *modify* the FMC state. These two functions can be modeled inside the software interface as two methods: one for reading server states, the other for sending the command code.

5. IMPLEMENTATION DETAILS

5.1. The IDL Interface

Based on the above mentioned analysis of requirements, we decided to design an IDL interface [2] containing more than one method for each base operation type: polling, command, request access, and release access key. The IDL interface structure employed is shown in Fig. 3.

```
// time and data structure type def.
struct inftime
{    short msec;
     short sec;
     short min;
     short hour;
     short day;
     short month;
     short year; };

interface cellcom

{//  returns an access key only if the
 //  server is  available; returns 0
 //  otherwise

long registerKey();

//   releases the access key; server
//   becomes available

void releaseKey(in long key);

//   obtains state information; returns
//   0 for success

short display
(in long key, out long states);

//   obtains time and state information;
//   returns 0 for success

short display_t(in long key, out long
states, out inftime time);

//   sends a command for the FMC;
//   returns 0 for success and a
//   value greater than 0 for failure

short command(in long key,in short cmd);

//   sends a command for the FMC and
//   obtains state information; returns
//   0 for success

short display_cmd(in long key, in short
cmd, out long states);

//   sends a command for the FMC and
//   obtains time and state information;
//   returns 0 for success

short display_cmd_t(in long key, in
short cmd, out long states, out inftime
time); };
```

Fig. 3. IDL interface structure

Access keys are tools that enforce mutual exclusive access to server to those clients that send commands to the FMC. This is a classical solution for the well known "readers/writers" IPC (InterProcess Communication) problem [11]. Specifically, "reader" processes are client modules that only request information about the

FMC (state, time and date, etc.) while ”writer” processes are client modules that send commands to the FMC to change its state.

The *registerKey()* and *releaseKey()* methods must be used by the “writer” processes before issuing any other calls. The server returns a new, distinct access key for the caller process if and only if there is no other valid access key currently used by a different client. In this context, it is necessary to note that “currently used by a client” refers to the time interval between obtaining and releasing the access key by that client.

Due to an access key inspection mechanism, if a client fails to continue the communication with the server after receiving a key, other clients can compete to obtain the access.

For any *registerKey()* call the server checks the current access key for the last usage moment. If the non-usage time interval is greater than the one set as limit by the specific analysis requirement (for example, 5 seconds), the current access key is considered “not valid anymore”. The server can then grant the new *registerKey()* request. Keeping “alive” (valid) an access key is possible if and only if the client uses the server with an appropriate frequency. This mode of operation satisfies the system’s requirements without decreasing the system’s overall performance.

Access keys are crucial only for c*ommand(), display_cmd()* and *display_cmd_t()* methods. If they belong to a “reader” client, the other two methods, *display()* and *display_t()* can use any value as their first call parameter. Otherwise, the first parameter must be the registered access key. These method calls help the “writer” client not only to obtain information about the FMC but also to preserve the validity of the access key. Figures 4 and 5 show two possible client/server scenarios.

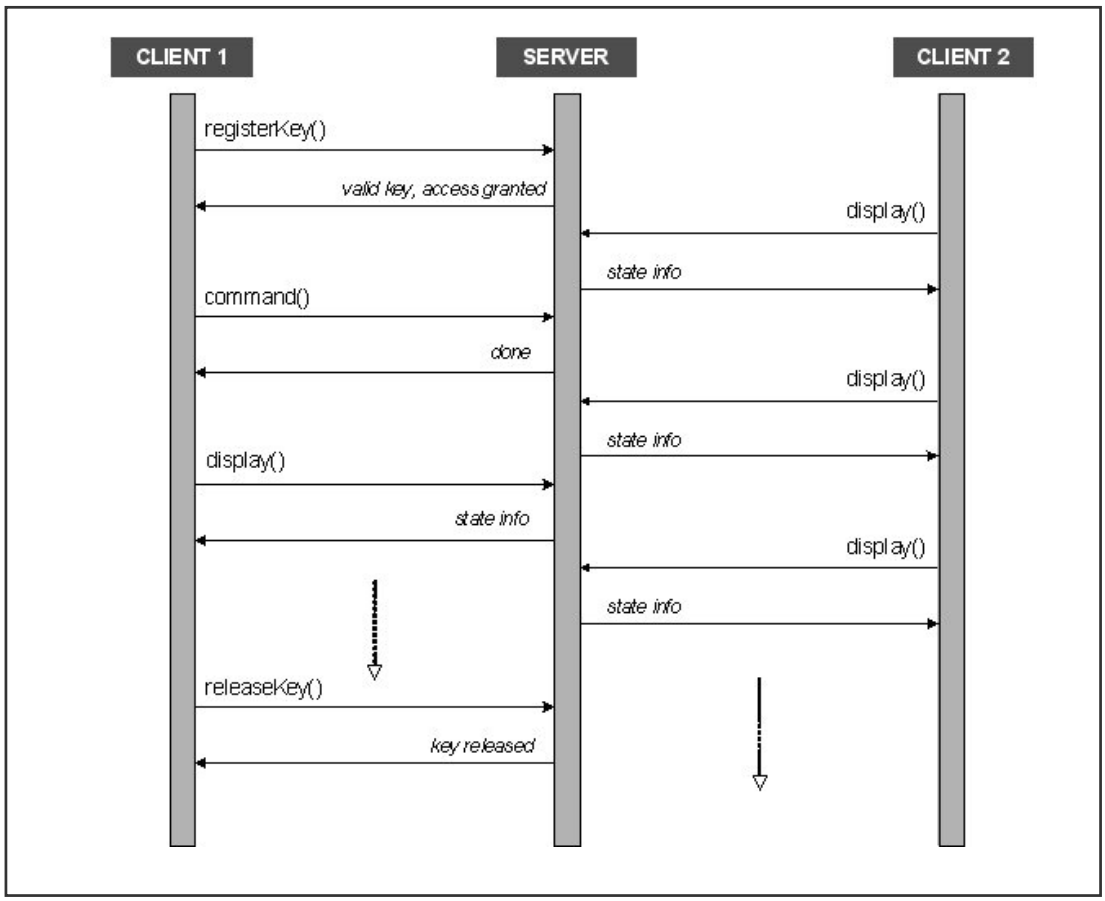

Fig. 4. First client/server scenario

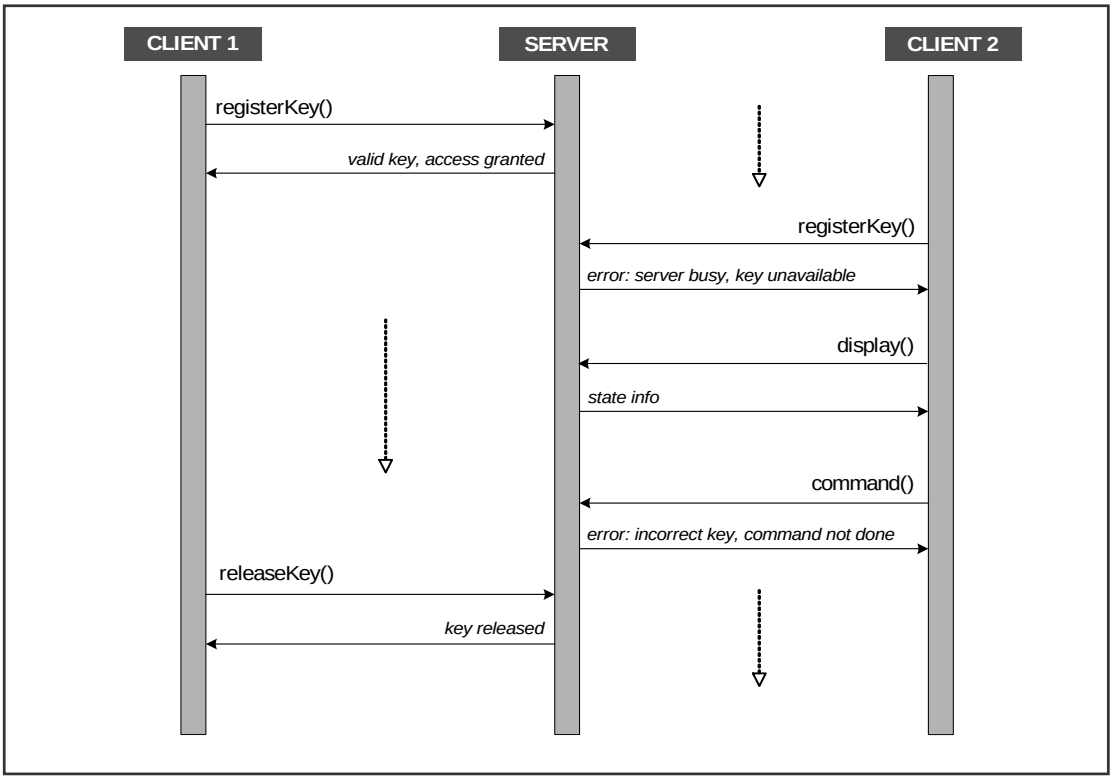

Fig. 5. Second client/server scenario

Specifically, Fig. 4 provides an example of CORBA interaction for a "writer" client (Client 1) that requests an access key, sends a command for the FMC, and reads state information. Concurrently, a "reader" client (Client 2) not owning an access key can only inspect FMC evolution, obtaining state and/or date and time information from the server. Figure 5 presents the interaction scenario for two "writer" clients and the server. If the first client has already obtained an access key and is currently using the server, the second client must wait. Its request can not be granted until the first key is released or expires. Until then the second "writer" client can only act like a "reader", inspecting FMC state through server services.

5.2. The Command Module

The command module client used as a control panel for the flexible manufacturing cell, the *Remote Cell Control*, has been designed for Microsoft Windows platforms and special attention has been paid to the graphical user interface, the information supplied, and its presentation.

Equipped with a robust and intuitive user interface (shown in Fig. 6), the *Remote Cell Control* implements not only command functions but several other useful facilities such as statistical and graphical views of the FMC state transitions. Using server supplied information, *Remote Cell Control* builds animated graphics for each of the twenty-two variables that describe the current state of the supervised object. The animation is done by shifting each graphic to the left side of its window, such that older events can be viewed along with the current time value. On the right side of the window, the new events are showed. Every important state variable of the FMC can be supervised by simply clicking the appropriate button placed below the graphic. The human operator receives information about state transitions from several visual elements with on/off LEDs appearance placed on the left side of each graphic. Clicking one of the twenty-two buttons implies sending a command to the server through the network, using CORBA communication mechanisms. The command is then delivered to the FMC and visual confirmation will be received by the operator within a short time delay. This delay is due to both network data propagation characteristics and FMC's reaction latency. Some settings are allowed in order to reduce the information display refreshing rate and minimize the delay of visual confirmation. The right-hand side smaller panel of the graphical user interface allows the operator to control the connection with the server, to access different options, and to freeze graphics by clicking the *Freeze* button.

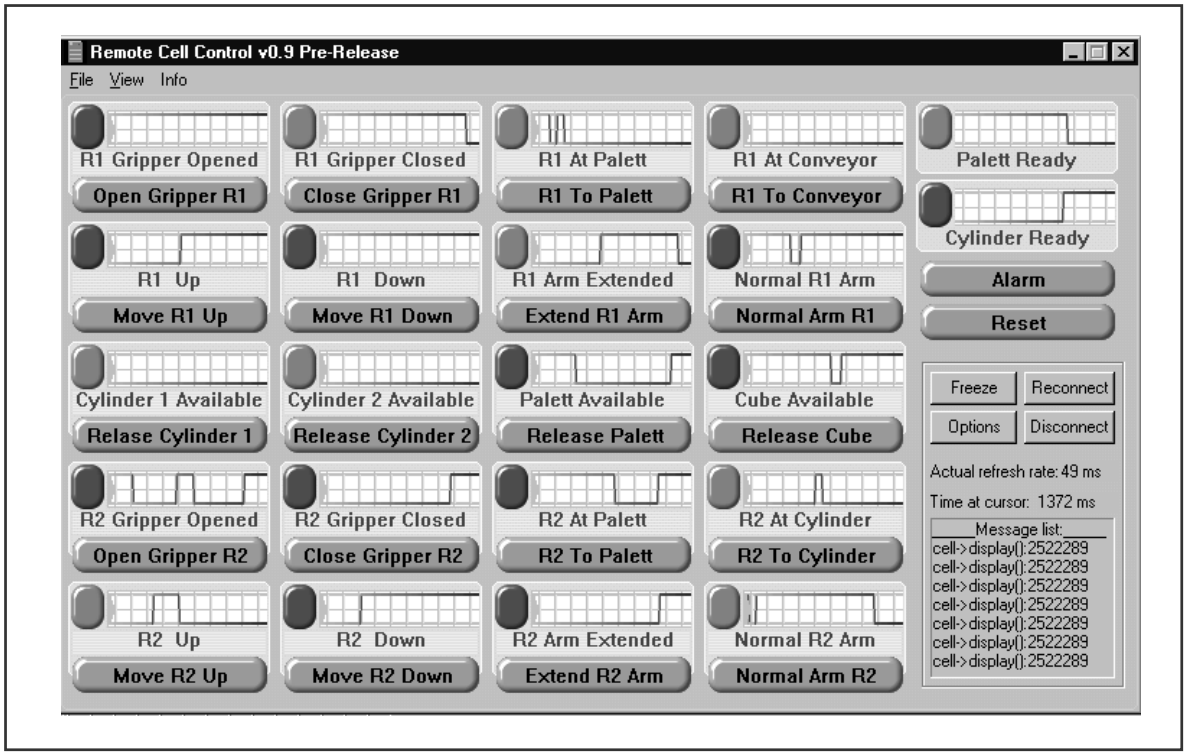

Fig. 6. Screenshot of the front panel

The *Freeze* option allows a thorough study of state transitions, from a given moment of time until the freeze command is issued. It is easy to determine the ordering of events by simply placing the mouse on a certain point of the graphic and reading the corresponding time information *(Time at cursor)*. This information provides the elapsed time between the occurrence of the cursor-selected event and the moment when the *Freeze* button was pressed. Compared to a simple visual observation, this method is more accurate. When clicked, the *Freeze* button changes its label to *Continue*. Clicking this button again resumes the previous mode of presentation and the graphics resume their animation. Although not showed on the screen during the freeze state, new events that occurred during this state appear now on the graphic, on the left side of the point that corresponds to the current moment. It is necessary to point out that only the graphics can be frozen, any change in the server states being immediately indicated to the operator by the panel's LEDS.

The *Options* button allows the operator to choose the name and IP address of the server and the communication port number and the *Reconnect* and *Disconnect* buttons control the connection with the server. The *Reconnect* button can be used for both activating new functional characteristics and for re-establishing the connection with the server after it was closed or lost. The small *Message list* panel on the bottom-right corner of the screen contains all the messages sent to the server.

6. CONCLUSIONS

This paper has presented the main aspects related to the design and implementation of a software system for remote supervision and control of a DEGEM 2000 Educational FMC. Built as a heterogeneous supervisor/control system that uses the Windows ™ and QNX operating systems as main support and CORBA technology as integration tool, this software system has been recently integrated into a complex virtual laboratory at the "Politehnica" University of Bucharest.

Designed primarily for educational purposes, the system allows students at remote locations to experience and study mechanisms specific to real-time operating systems, computer integrated manufacturing, and control systems. Through the use of a web video camera that tracks all the movements within the FMC, it also makes available to all the participants the visualization of the effects generated by various commands

issued. In addition, by its design, the system guarantees conflict-free usage of the FMC.

Another significant advantage of the described system is that the same control system architecture can be used for industrial flexible manufacturing cells, including cells in hazardous environments. This is possible because client modules such as the *Command Module* can be placed separately in remote, safe rooms.

Lastly, it is worth noting that the described software system has high scalability and flexibility: whenever needed, it is possible to add new types of client modules and to create clients based on additional hardware and software platforms, the only constraint being to use the same IDL interface and CORBA technology as the integration tool.

7. REFERENCES

[1] Serain, D., - "Enterprise Application Integration. L'Architecture des Solutions E-business", Dunod, Paris, 2001.

[2] Siegel, J., - "CORBA 3 Fundamentals and Programming. Second Edition", Wiley Computer, New York, 2000.

[3] Licks, V., Quiroga, M., Jordan, R. and Correa, J. S., - **"**Building Virtual Laboratories - A Web Based Experience", *Proceedings of ICECE 2000 -International Conference on Engineering and Computer Education, Sao Paulo*, pp. 157-160, 2000.

[4] DEGEM 2000 flexible manufacturing cell documentation.

[5] Saru, D. and Ionita, A. D., - "Object-oriented Software Systems" (in Romanian), ALL Educational, Bucharest, 2000.

[6] QNX – Watcom C – Library Reference Manual.

[7] Orfali, R., Harkey, D., and Edwards, J., - "Client/Server Survival Guide", Wiley Computer, New York, 1999.

[8] OMG - Object Management Group, http://www.omg.org, accessed February 10, 2003.

[9] CORBA website, http://www.corba.org, accessed May 5, 2003.

[10] OOC - Object Oriented Concepts Inc., http://www.ooc.com, accessed January 15, 2002.

[11] Tanenbaum, A. S., - "Modern Operating Systems. Second Edition", Prentice Hall, Amsterdam, 2001.