VIMS2002 International Symposium on
Virtual and Intelligent Measurement Systems
Mt. Alyeska Resort, AK, USA, 19-20 May 2002

Open Distributed Control and Measurement System Based on an Abstract Client-Server Architecture

F. Pianegiani, D. Macii, P. Carbone
DIEI – Dipartimento di Ingegneria Elettronica e dell'Informazione
Università degli studi di Perugia, via G. Duranti 93 – 06125 Perugia, Italy
Phone: +39 075–5853629, Fax: +39 075–5853654, Email: carbone@diei.unipg.it, macii@diei.unipg.it

Abstract *– This paper describes in detail a Java-based, client-server architecture specifically conceived to allow a flexible control of remote devices. The main attributes of the proposed solution are portability and flexibility. The former feature is assured by the employment of the TCP/IP protocol suite and by the Java language properties. The latter is achieved using a high level of abstraction in the implementation of the system, that addresses multi-user issues and a wide range of possible applications with high code reusability. The proposed architecture can easily be upgraded to control various kinds of devices, by simply adding a limited amount of code on the system server-side.*

Keywords *– Java, distributed measurement system, client-server architecture, remote calibration.*

1. INTRODUCTION

In recent years, the growing demand for improved interoperability between electronic instruments, the increase in PC computing and input/output capabilities along with the diffusion of standard buses specifications (e.g. IEEE 488, IEEE 1394, PCI/PXI and VME/VXI) and network protocols, have favoured the development of software tools oriented to the implementation of distributed control architectures. Such tools represent enabling technologies for the development of home automation networks [1], human-robot interactive applications [2]-[5], real-time collaborative telemedicine systems [6] and, more generally, distributed measurement systems (DMS) both for educational and industrial purposes [7]-[9]. Unfortunately, many of the commercially available software tools devoted to Virtual Instrument (VI) implementation (e.g. LabView™, Lab Windows™/CVI and HP VEE™) often require specific applications to be installed on client computers [10]. Moreover, since they are based on proprietary technologies, remote control applications can not be freely distributed or easily extended. Conversely, common object remote brokers and interfaces have been defined to create extensible and distributed programming environments. By using Corba technology, for instance, each application is the result of a collaboration between several objects distributed over communicating networks and coded using various languages. Thus, new programs can easily and quickly be extended to address dedicated purposes. A similar goal can also be reached with object oriented languages, like Java and C++, without modifying the original structure of the source code [11]. This is achieved by using abstract classes and dynamically software libraries such as dynamic link libraries (dll) on Windows platforms or shared object (so) libraries under Unix.

Even if many architectures have been proposed to control instrumentation remotely, a certain lack of detailed low-level descriptions of possible implementations has been observed. Therefore, one of the aims of this paper is to give a full description of a highly abstract Java-based client-server system. Unlike other distributed multi-server architectures [8][10] [12], the solution presented in this paper focuses mainly on the optimization of the communication systems between multiple clients and a single server. Its flexibility and ease of reconfiguration has been considered as an important feature for the development of future multi-layered distributed applications.

In the following sections, at first the overall operating environment is described and the design choices are explained. Then, it is shown that the integration of new instrumentation and PC-cards, such as CAN or IEEE 488 controllers, can be accomplished without modifying the code of the client-server architecture. Finally, an example of system implementation devoted to remote calibration purposes is presented.

II. DESCRIPTION OF THE DISTRIBUTED MEASUREMENT SYSTEM

Because of the rapid evolution of information exchange standards, any newly devised architecture for remotely controlling instrumentation should include enough features to accommodate actual hardware and software specifications, possibly anticipating future technological developments. This is why the proposed system has been designed to be highly abstract, easily extensible and user-friendly.

The system architecture is shown in Fig. 1. It consists of a group of distributed client-server applications that can be upgraded to control general-purpose instrumentation over the Internet. These instruments are either plugged directly into PCs (e.g. PCI data acquisition boards) or interfaced via a bus controller (e.g., IEEE 488 and IEEE 1394 cards). The access and the possible sharing of the available resources are managed using the multithreading approach. When a user requests the execution of any control operation, the server application runs a new dedicated thread. Since more than one thread can be run and processed independently, the system allows the execution of multiple operations at the same time. A portion of the server application has been conceived to engage, release and share the requested resources and to

0-7803-7344-8/02/$17.00 ©2002 IEEE

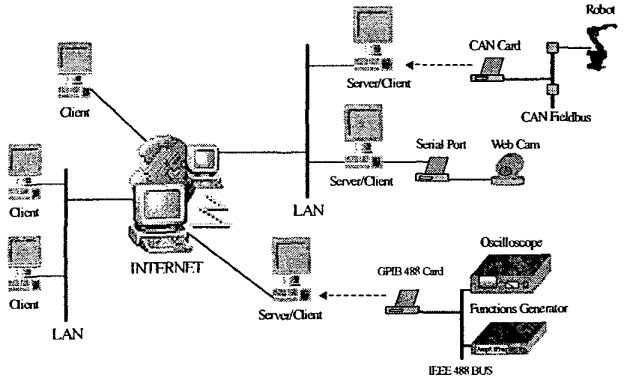

Fig. 1. Extensible distributed measurement system.

manage simultaneous accesses. The client-side application is started automatically when a user connects to the server homepage using a Java-enabled browser. Users are presented with a graphical user interface (GUI) showing the list of controllable interface cards and instruments, directly installed on the server. Moreover, if a standard bus controller (e.g. IEEE 488-PCI bus interfaces) is chosen, an additional search is performed to find devices connected to the bus. A list of all resources is shown on a graphical panel, enabling selection of a specific device. Then, by using socket connections and the TCP/IP communication protocol, information can be exchanged with the server unit (instrument control commands, parameters and reports). Finally, measurement data can be saved locally or visualized on the screen. Notice that, the use of full Java-based graphical panels has been preferred over servlet technology, because it reduces the loading time of GUI, thus improving system performance.

Unlike other valuable Java-based multithreading solutions that need the insertion of several Java server classes to manage any instrument newly connected to the server [13], the most important feature of the proposed architecture is its easy extensibility. In fact it is possible to manage new cards and instruments installed on a server unit without recompiling or modifying any code component. It is only necessary to add a limited portion of upgrading code on the server-side. This operation is feasible either locally or remotely. In the following subsections the chosen programming tools will be described, along with a functional and temporal analysis of the proposed architecture.

A. Software Tools

As mentioned in the introduction, several languages are used to implement distributed control systems. The architecture described in this paper has been coded using the Java and C languages. Java simplifies the development of data exchange mechanisms between client and server, by offering GUIs and client-server communication methods based on socket connections and applet executions. Moreover, the Java Secure Socket Extension (JSSE) set of packages implements a Java version of Secure Socket Layer (SSL) and TLS (Transport Layer Security) protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Accordingly, security issues related to any connection established between client and server ports could be addressed easily by using SSL socket methods.

On the other hand, C is the language most frequently used to create drivers and libraries for the low-level control of devices at the highest execution speed. In order to translate programming elements from Java to C, the Java Native Interface (JNI) and native methods have been employed [14]. However, native methods demand knowledge of both Java and C languages. Better results and lower development efforts could have been achieved using Microsoft® Visual J++™ [13]. However, this product was recently withdrawn from market.

In the following, to describe the system at the architectural level, the unified modeling language (UML) has been employed. This is a language useful to describe organizational and technical systems. In fact, it provides several types of diagrams divided into three main categories: diagrams to model static application structures, diagrams to represent different aspects of dynamic behaviors and diagrams to organize and manage the application modules [15].

B. Functional Description of the Client-Server Architecture

The architecture is composed of a client, a server and an Internet units. The implementation code is divided in 2 parts: an Abstract Client-Server Architecture (ACSA) that represents the permanent part of the system and a code portion to be upgraded every time a new device is added. The working principle is based on three main functions responsible for client-server communications, abstract management of the available physical resources and upgrading operations. In order to describe the functional and time relations between objects and classes of the ACSA architecture and the upgrading code, the UML collaboration diagram shown in Fig. 2 has been used. In this diagram, users, developers and managers are represented as three actors that allow data exchange between client and server for remote automation purposes. The sequence of operations is as follows. The *system manager actor* or the server operating system starts the server application asynchronously by opening the *AppletServer.htm* file. It contains the number of the server port accepting connections and a list of the available resources. This data is essential to access and to share over the Internet the devices connected to the server. Then, *Appletserver.htm* loads an applet that reads the information stored in the HTML file through the *getParameter()* method. Finally, the applet starts the *MultiThreading* class that allows the server to enter a waiting state, expecting connection requests. The *user actors* load the client application by opening the server homepage and executing

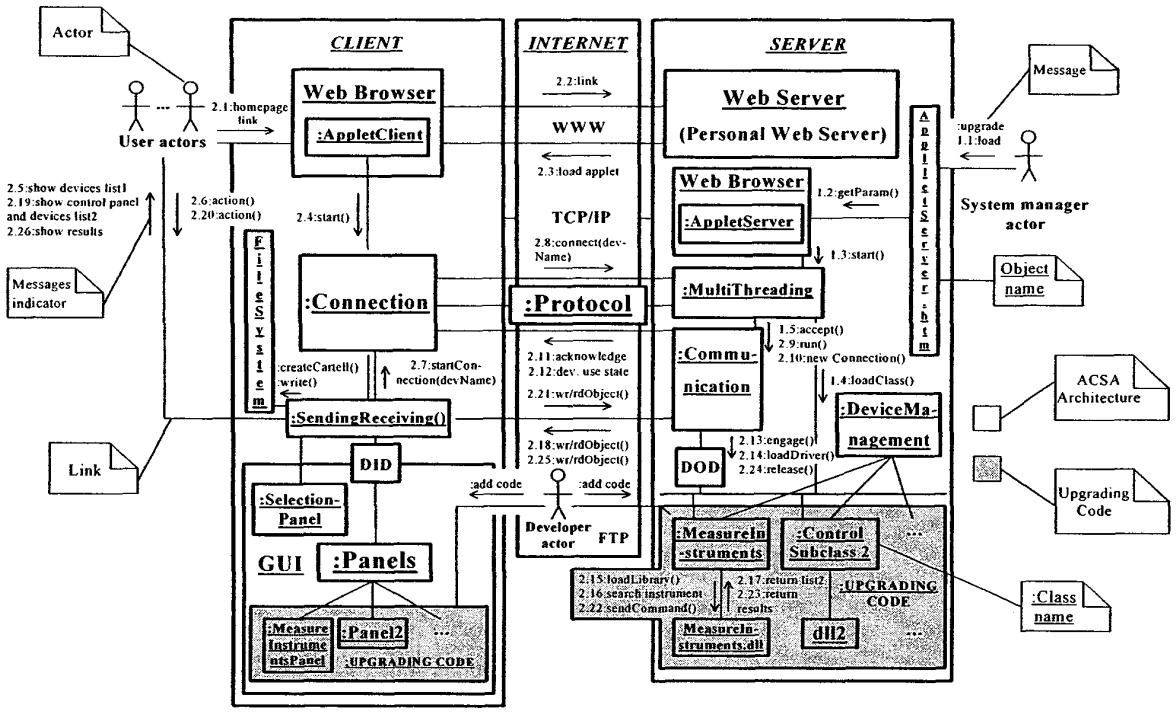

Fig. 2. UML collaboration diagram of the client-server architecture.

the AppletClient applet. They can establish a link with the server by means of the *Connection* class which generates a socket connection to *MultiThreading*. The connection or the disconnection to or from the server, can be executed by the *user actors* through a dedicated menu in several graphical panels. They are used to choose the instruments, to insert control commands and parameters, and to visualize responses from the server.

If *MultiThreading* accepts the connection request, it starts the *Communication* class in a dedicated thread and waits for a new request. In this way, more than one *user actor* can have access simultaneously to the server system and control one of the available instruments. To allow data exchange between client and server connection, sockets and object input/output streams are defined. Initially, such streams are used by the server to send acknowledge messages concerning the connection state; then, they are used by the client to receive the list of the available resources. The selection of resources is performed in 2 stages: at first, *user actors* have to choose one of the devices or controllers in the received list shown through *SelectionPanel*. Then, if a controller for a particular input/output standard bus is chosen (e.g., IEEE 488, IEEE 1394), a search for devices physically connected to the bus is started automatically. This mechanism is carried out by a native method implemented in the driver of the bus controller. After completing the first selection stage, *Connection* sends to the *Communication* class the name of the chosen device and waits for responses about its availability. The rules allowing the dialog between *Connection* and *Communication* are defined in the *Protocol* class. If the requested resource is available, the corresponding panel, subclass of *Panels,* will be shown to the *User actors.* So they can select one of the instruments found in the second search stage, and can insert measurement data and control commands in the panel fields. The user requests are sent to the server through the *Sending-Receiving* class. This operation employs an input serializable object of the *DeviceInputData* (DID) class whose attributes are suitable to represent all kinds of device input data. Finally, *Communication* and the subclasses of *DeviceManagement*, transfer to the instruments the requests received from the client and return measurement results, if available. This occurs through an output serializable object of the *DeviceOutputData* (DOD) class.

For each new resource installed on the server, the *developer actor* has to upgrade the system by adding new management code (highlighted in grey in Fig. 2). In particular, a new subclass of *DeviceManagement*, as well as one or more dll drivers for the low-level control of the device and a new dedicated GUI panel, have to be implemented. Moreover, when the new card or instrument is connected to the server, the *manager actor* has to upgrade the parameters list in the *AppletServer.htm* file.

III. ABSTRACT MANAGEMENT OF THE AVAILABLE RESOURCES

The easy extensibility of the system depends on the abstraction features of the ACSA. This has been achieved on the basis of three design choices:

- the declaration of device-independent attributes in the DID and DOD classes;
- the dynamic loading, on the client-side, of the control panels inheriting attributes from the *Panels* superclass;
- the dynamic loading, on the server-side, of the device specific subclasses inheriting methods from *DeviceManagement*.

The two former features in the list enable the exchange of management and measurement data through a common interface, regardless of the kind of controlled devices. This means that control messages, either written in text fields or set by clicking on checkboxes, are encapsulated in a unique data record before being transferred. A dual mechanism is used to return measurement results to the client. All of the fields of these records are declared inside the *Panels* superclass as shown in Fig. 3(a), in which a UML class diagram describes the hierarchic relationship between *Panels* and its device-dependent subclasses.

As regards the third feature in the list, *DeviceManagement* allows the management of any controllable resource. This is accomplished by declaring abstract methods such as *shareable(), engaged(), engage(), release()* and *loadDriver(object: Object)*. As shown in the UML class diagram plotted in Fig. 3(b), all of these methods are implemented in the *DeviceManagement* subclasses whose structure depends on the different characteristics of devices connected to the server. The device-dependent methods, unknown to the ACSA, are called by the *Communication* class through an instance variable of *DeviceManagement*.

The methods *shareable()* and *engaged()* are devoted to detect the device availability following a specific user request, while *engage()* and *release()* enable respectively the control of the device and its release when the user stops controlling the resource. Finally, *loadDriver(object)* loads dynamically the dll or so drivers containing the C native methods necessary to control the requested instrument.

IV. AN APPLICATION EXAMPLE OF THE ACSA: A REMOTE CALIBRATION PROCESS

A test application of the ACSA architecture has been developed to run remotely several calibration procedures on measurement instruments located in the laboratories of the University of Perugia. In particular, after installing an IEEE 488 controller board on the server-side of the system, an HP 34401A digital multimeter (DMM), an HP 54603B oscilloscope and a Fluke 5500A multifunction calibrator were connected to the GPIB bus. This approach may be used, for instance, to provide traceability for a customer's calibrator by means of a transportable DMM, calibrated, in its turn, via a reference calibrator maintained at a national laboratory or at an accredited standards laboratory [16].

To perform the remote calibration, three software units have been implemented: a subclass of the *DeviceManagement* class, a C program and a subclass of the *Panels* class. Inside the first unit the *loadDriver* method calls two native methods, defined in the dll-compiled C program. The first native method seeks all of the measurement instruments connected to the GPIB bus, while the second carries out the calibration procedures after receiving the necessary input calibration parameters specified by an enabled remote *user actor* on a client computer. This data consists of the name of the instrument to be calibrated (Device Under Calibration-DUC), the kind of measurements that the DUC has to perform (e.g., volt AC/DC, current AC/DC, resistance 2/4 wire), the measu-

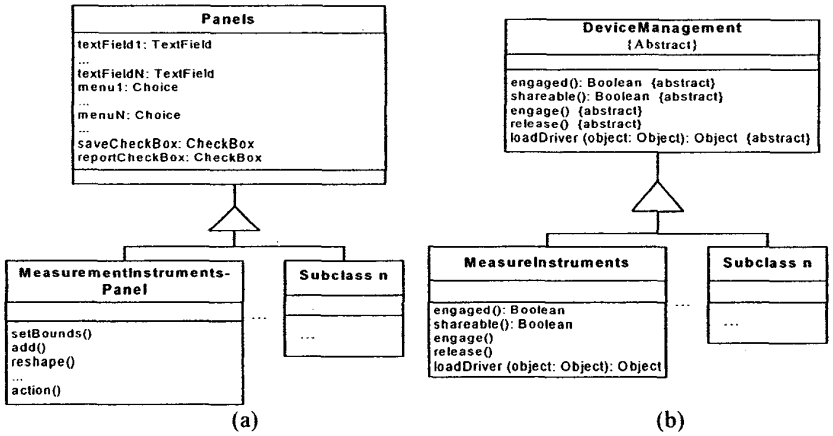

Fig. 3. UML class diagrams of *Panels* (a), *DeviceManagement* (b) and their subclasses.

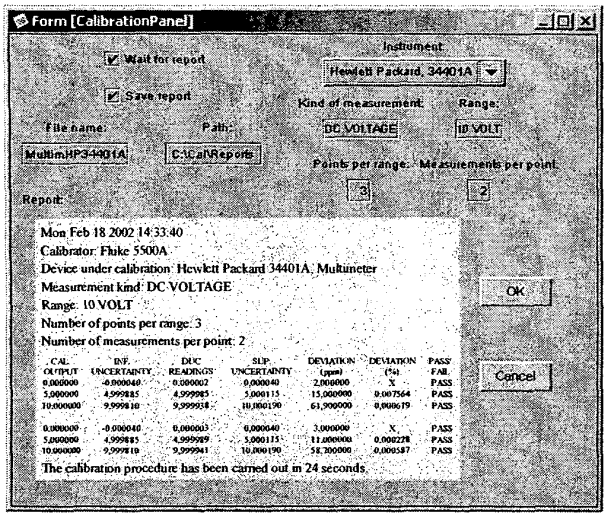

Fig. 4. Client-side panel of an application devoted to the management of remote calibration operations.

rements range, the number of points per range value and the number of measurements per calibration point. Then, the same native method processes the measurements results and returns the calibration report. This report contains the date, some information about calibrator output data, the minimum and maximum measurement values permitted by the reading and range uncertainty of the DUC, the measurements carried out by the DUC, the deviations between measured data and calibrator outputs and the calibration timing.

The third unit of the upgrading code, that is the graphical user interface to be downloaded on the client-side, has been realized by using the *Swing* components and Forte™ for Java Integrated Development Environment. An example of GUI is the virtual panel shown in Fig. 4, that reports the outcomes of a remote calibration procedure executed on the HP 34401A multimeter.

CONCLUSIONS

In this paper, an abstract client-server architecture devoted to the control of measurement instruments over the Internet has been described. This architecture exploits the flexibility, the portability and the network-oriented features of the Java language, thus avoiding the use of proprietary, license-bound software tools. Moreover, the system has been conceived to be easily updatable whenever new hardware resources are connected to the server computer. This attribute results from the high level of abstraction coded in both client- and server-side applications. In fact, while low-level device control is performed by C routines, the use of abstract classes and the dynamic loading of native methods enable the client to communicate with every instrument, regardless of its specific properties. The proposed solution to the management of remote instrumentation is further improved by the multi-threading mechanism, that allows multiple users to perform simultaneous measurements independently. In order to verify the correct operation of the system, an application based on the ACSA architecture has been developed to run remote calibration procedures on several measurement instruments connected to a server computer via a GPIB bus.

REFERENCES

[1] T. Saito, I. Tomoda, Y. Takabatake, K. Teramoto, K. Fujimoto, "Gateway technologies for home network and their implementations," Proc. *Workshop Distributed Computing Systems* (DCS), pp. 175-180, 2001.

[2] D. Buhler, W. Kuchlin, G. Grubler, G. Nusser, "The Virtual Automation Lab-Web based teaching of automation engineering concepts," Proc. *Engineering of Computer-Based Systems* (ECBS), pp. 156-164, 2000.

[3] A. Speck, H. Klaeren, "RoboSiM: Java 3D robot visualization," Proc. *International Conference on Industrial Electronics, Control and Instrumentation* (IECON), Vol. 2, pp. 821-826, 1999.

[4] A. S. Sekmen, Z. Bingul, V. Hombal, S. Zein-Sabatto, "Human-robot interaction over the Internet," Proc. *IEEE Southeastcon*, pp. 223-228, 2000.

[5] P. G. Backes, K. S. Tso, J. S. Norris, G. K. Tharp, J. T. Slostad, R.G. Bonitz, K. S. Ali "Internet-based operations for the Mars Polar Lander mission," Proc. *International Conference on Robotics and Automation* (ICRA), Vol. 2, pp. 2025-2032, 2000.

[6] Mee Young Sung, Moon Suck Kim, Myung-Whun Sung, Eom Joon Kim, Jae Hong Yoo, "CoMed: a real-time collaborative medicine system," Proc. *Computer Based Medical Systems* (CBMS), pp. 215-220, 2000.

[7] Wang Lihui, B. Wong, Shen Weiming, Sherman Lang, "A Web-based collaborative workspace using Java 3D," *Design Computer Supported Cooperative Work* (CSCW), pp. 77-82, 2001.

[8] L. Benetazzo, M. Bertocco, F. Ferraris, A. Ferrero, C. Offelli, M. Parvis, V. Piuri, "A Web-based distributed virtual educational laboratory," *IEEE Trans. Instr. and Meas.*, Vol. 49, no. 2, pp. 349-356, Apr. 2000

[9] K. Michal., W. Wieslaw, "A new Java-based software environment for distributed measurement systems designing," Proc. *Instr. Meas. Tech. Conf.* (IMTC), Vol. 1, pp. 397-402, 2001.

[10] P. Arpaia, A. Baccigalupi, F. Cennamo, P. Daponte, "A measurement laboratory on geographic network for remote test experiments," *IEEE Trans. Instr. and Meas.*, Vol. 49, no. 5, pp. 992-997, Oct. 2000.

[11] D. Buhler, G. Nusser, W. Kuchlin, G. Gruhler, "The Java Fieldbus Control Framework-object oriented control of fieldbus devices," Proc. *Object-Oriented Real-Time Distributed Computing* (ISORC), pp. 153-160, 2001.

[12] M. Bertocco, M. Parvis, "Platform Independent Architecture for Distributed Measurement Systems," Proc. *Instr. Meas. Tech. Conf.* (IMTC), Vol. 1, pp. 648-651, 2000.

[13] D. Grimaldi, L. Nigro, F. Pupo, "Java-based distributed measurement systems," *IEEE Trans. Instr. and Meas.*, Vol. 47, no. 1, pp. 100-103, Feb. 1998.

[14] Sun Microsystems, web address: *http://java.sun.com.*

[15] Object Management Group, web address: *http://www.omg.org/uml.*

[16] N. Oldham, M. Parker, "Internet-based test service for multifunction calibrators," Proc. *Instr. Meas. Tech. Conf.* (IMTC), Vol. 3, pp. 1485-1487, 1999.