Virtual Lab: a Java Application for Distance Learning

A. Di Stefano, F. Fazzino, L. Lo Bello, O. Mirabella

Universita' di Catania - Facolta' di Ingegneria
Istituto di Informatica e Telecomunicazioni
Viale A.Doria, 6 - 95125 Catania (ITALY)

Abstract - **Recent progress in telecommunications has made it possible to develop software offering users new features such as access to remote resources from different geographical locations. At the University of Catania we have developed a Virtual Laboratory for distance learning. Our lab is realized in Java and uses the Internet or/and an Intranet to provide complete access to the lab's resources. Users can consult tutorials, run simulations and intervene in the production cycle of an educational industrial plant from any Internet node. This paper describes the general organization of the VL and the implementation choices made in order to guarantee the efficiency of resource management, provide adequate communication and security for the information exchange and ensure flexible, customizable service configuration.**

I. INTRODUCTION

Distance Learning is an emerging reality in which students, teachers and researchers in different geographical locations can interact by accessing remote resources dedicated to learning, usually software. Recently, thanks to the development of new communication infrastructures, the functions offered by distance learning have been extended to real laboratories whose physical processes can be remote-controlled. This has led to the setting up of industrial labs with varying degrees of complexity and even real plants for teaching or research purposes [1].

In this paper we describe a project implemented at the Engineering Faculty of the University of Catania, in which various functions have been integrated in a single project and made accessible in what we can call a Virtual Lab (VL). The VL is a distributed application which combines software and hardware to provide users with access to the resources of a remote lab. The lab resources can be in the form of software, like tutorials and simulators, or real manufacturing plants specially configured for remote access. Users can link up with the VL from any geographical location to consult documents, run simulations or even monitor and carry out experiments on real plants.

The services relating to the use of a real plant or lab include distance consulting of documents about the functioning of the specific plant and the operations that can be carried out on it.

The VL was developed first of all to meet the learning requirements of students on Industrial Engineering courses. One of the greatest problems that emerge is the insufficiency, in terms of both equipment and space, of the labs available for student use: they are frequently not able to meet the students' need to experiment with the notions they have received during lectures, especially the theory of process control, problems regarding computer networks, FieldBus applications, etc. Besides acting as an interface between the user and a real plant, the VL can have other functions. If, for example, the user's aim is to carry out a series of measurements on a plant with certain inputs, to obtain data about the plant's behavior for learning or research purposes, results could be supplied, not only by a real plant, but also by a simulator. Here again it is useful for the lab to offer on-line consultation of tutorials which teach users about the features of simulators and how to use them.

It is also useful, for both real plants and simulators, not only for user access to be "guided", i.e. through interaction with preconstituted interfaces, but also for individual users to be able to implement their own applications using the resources of the remote lab.

The application is written in Java and allows the user to link up with the lab and interact with the available resources directly through his Web browser.

To link up with the remote lab, a user connected to the Internet only needs a Java-enabled Web browser (besides the HTTP address of the lab). All users can use the most recently updated VL interface which is transferred on to the local machine when the connection is set up. Through this interface, a user can interact with the resources the lab provides.

In Section 2 we will give further details of the Virtual Lab paradigm, its scope, the resources that can be made available and the need to configure plants in such a way as to make them accessible. In Section 3 we will describe the interfaces the lab offers the users and the programmers. In Section 4 we will illustrate how communication occurs over the network between the user and the lab. In Section 5 we will present the architecture and possible topologies of the lab. In the 6th and final Section we will summarize what has been achieved so far.

II. THE VIRTUAL LAB

The Virtual Lab is one of the new possibilities offered by current technology as an aid to teaching and research. It is a distributed software which makes it possible for a user to access a remote lab through the Internet or an Intranet, allowing distance interaction with the lab to carry out operations on the lab's resources which are normally performed locally.

For remote access to a plant to be possible through a Virtual Lab, the plant has to have an appropriate hardware and software organization which will allow it to interface with the user who wants to interact with some lab resources, and also with the VL resources, which range from simple tutorials to simulators to real physical processes. More specifically, there have to be several host computers, each with software-driven digital control devices (dedicated boards or ports for communication with external devices) which will allow the user to modify certain plant reference parameters. Through a console, it is possible to read the significant process variables and make them available for monitoring through the Intranet/Internet. These computers also have to be connected to the Internet or a company Intranet using suitable architectures which will be described below.

In the version proposed here, the Virtual Lab can offer users access to three different types of resources:

-it has to act as an interface with real plants or laboratories in which limited experimental process control activities are reproduced. On their components read operations and, under appropriate safety conditions, configuration operations (e.g. setting of state variables) can be performed, according to certain specifications;

-it has to offer access to simulators by passing input parameters, and remote execution and reading of results;

-it also has to offer selective access to single portions of tutorial for reading and updating.

These resources have to made available for the following purposes:

Distance consultation: students must be able to access tutorials and simulators from home or the place in which they are studying;

Remote Monitoring: it must be possible to monitor the lab or remote plant in order to study its behavior in various situations (e.g. during normal production or in the presence of faults or malfunctioning);

Remote Performance Evaluation: it must be possible to perform both on-line evaluation of the performance of a real plant and evaluation of the performance obtained by simulating a model of the plant;

Remote Configuration: privileged users must be able to introduce modifications, from a distance, in some portions of the production process under study in the lab or in the plant. These modifications must involve limited areas and are subject to careful control to prevent them from causing any risk to the lab or the plant.

III. INTERFACES

The Virtual Lab provides its users with two kinds of interface.

First of all, a *user-friendly* interface through which the user can interact immediately to perform operations on the resources available. Interfaces of this kind are usually graphics-based. The graphical interface defined in the VL, which is called **GUI** (*Graphical User Interface*), allows guided access to the objects the lab contains.

The VL also provides an interface for the programming of applications, called **API** (*Application Programming Interface*). The API allows developers to interact with lab resources without following the pre-established scheme of the GUI, and to create local programs which access several lab resources at the same time, thus allowing highly flexible assembly of the interfaces offered with the various resources.

Unlike GUI users, API users require detailed knowledge of the programming interface they are offered so as to be able to manipulate the VL resources in as flexible a way as possible. Lastly, it should be pointed out that the GUI is not independent of the API; on the contrary, it may be considered to be the main application using its services.

A programmer can therefore implement an application using the API to perform all the required operations directly on the resources; he can even assemble various resources, directly interfacing the lowest level of the remote virtual lab (e.g. to carry out a joint experiment on several resources at once). The GUI, on the other hand, is an application provided by the API which allows the user to gain immediate access to resources simply by configuring the graphical interfaces he is offered. In providing VL services, interfaces have to meet requirements of *Upgrading Independence*, whereby the user is provided with transparent acquisition of each new version of the VL, and *Platform Independence*, whereby the interface the remote user is offered is independent of his hardware-software platform.

Meeting the first requirement by using a classical *telnet* mechanism, through which the user links up with the station running the lab via Internet and gives commands from a distance, is not a practical solution, as on geographical networks, like the Internet, *telnet* often only offers remote text access and this does not make it possible to interact through a GUI.

In addition, other mechanisms do not provide a type of access whereby the graphical interface seen by the user is executed on a remote machine, as geographical networks in general, and the Internet in particular, are not currently capable of guaranteeing sufficient bandwidth for the transfer of a continuous flow of information in graphical form. Therefore, as by nature graphical interfaces have to be managed by the computer on which they are to be displayed, the main way to implement a program that offers a user a powerful GUI is to use a high-level language, for example an object-oriented one, like C++. Such languages cannot, however, meet either *Upgrading* or *Platform Independence* requirements, as the graphics libraries they use are highly platform-dependent and require the software to be re-installed every time a modification is made.

The VL we propose was implemented using the object-oriented Java language developed by Sun Microsystems [2][3][4][5][6]. With this language it is possible to meet both the requirements made by implementation of the Virtual Lab and it has proved to be particularly suitable for the implementation of the GUI.

Using Java it is, in fact, possible to create small programs (*applets*) which can be called by a remote user through a common World Wide Web browser, by which they are interpreted and executed locally. The Internet user therefore only needs a browser, which will present him on each connection with an updated GUI through which he can interact with the lab. Java also provides platform-independent graphics libraries. A Java applet (or application) can therefore be executed both on computers with Win32 (NT or 95), OS/2, or MacOS operating systems and on various platforms belonging to the UNIX family (above all the Sun Microsystems Solaris).

To sum up, a user equipped with a Java-enabled Web browser can access the VL through the Internet and interact with the GUI displayed on his screen. If, on the other hand, he wishes to implement an application of his own which goes beyond the possibilities offered by the GUI, the user can use the API functions and write the application in Java.

The use of an interpreted language endowed with portable graphics libraries makes it possible to implement GUIs which provide simple presentation of the various components with which the user can interact; for a chemical plant, for instance, the graphics could represent the various valves (input actuators) and thermometers or barometers (output sensors).

Obviously, when remote access is provided from any Internet node great security problems arise for the VL. The requirements for user access via interfaces are as follows: 1) it is necessary to provide locking mechanisms to ensure that the user modifying parts of a system has exclusive access, so as to guarantee the consistency of the system; 2) plant safety has to be assured, preventing the installation of dangerous assembly between its various parts; 3) security has to be guaranteed by differentiating between access on the part of various users; that is, it is necessary to specify which users can access which resources, so that only these requests for access are accepted.

IV. COMMUNICATIONS

On the basis of its mode of communication, the system being considered is of the Client/Server type [7]. In our case, the client is the user linking up with the lab to consult tutorials, run simulations or give commands from a distance, while the server handles the whole set of software and hardware resources of VL which meet the user's request for access. Client/server interaction in such models is usually of the *Request/Reply* type, i.e. the user sends his request and the server replies after execution by sending a message containing the processing results. When applied to the VL, this model is suitable for remote execution of simulation tools or distance commands, but may be insufficient for the remote monitoring of plants.

To overcome the necessity for the VL client to send periodical monitoring requests, thus creating a useless amount of work for the network and causing delays in the processing of messages by the server, a further communication model has been developed, called the *Continuous Flow* protocol. The client sends the server a single monitoring request and the server starts sending the data at the desired sampling frequency. The algorithms executed by VL clients and servers can be summarized in the following Java-like code (Fig.1). The code is a simplified one: it neglects, for example, details relating to the management of faults in the client or server, although these have been developed in the implementation of the VL.

```
// Client Algorithm (2 protocols)
String[] RequestReply(String[] request) {
 request.type = "Request/Reply";
 sendStrings(request);
 String reply[] = receiveStrings();
 return reply;
}
String[] ContinuousFlow(String[] request) {
 request.type = "ContinuousFlow";
 if (!request.equals(lastRequest)) {
    sendStrings(request);
    lastRequest = request;
 }
 String reply[] = receiveStrings();
 return reply;
}

// Server Algorithm
while (userConnected) {
request = receive();
 if (request.type.equals("Request/Reply")) {
    reply = exec(request);
    send(reply);
 }  else  if  (request.type.equals("Continuous
Flow")) {
    while (userConnected) {
reply = exec(request);
       send(reply);
    }
 }
}
```

Fig. 1: Algorithms executed by VL Clients and Servers

In implementing the Continuous Flow protocol, to allow the client to acquire monitoring results synchronously, the user can cyclically execute the *ContinuousFlow()* method and obtain data from the server without having to send any further requests after the initial one.

When first invoked, in fact, the *ContinuousFlow()* method sends the server a monitoring request and, at the same time, stores the parameters of the user request; whenever it is subsequently invoked, if the parameters have not changed, the method just receives the results the server cyclically continues to send the client to meet the Continuous Flow request.

Without going into further detail regarding the actual communication protocol used to implement the VL, it should be pointed out that it uses the connection-oriented socket classes provided by Java for TCP/IP protocols. The solution adopted thus consists of implementing the VL by providing for the setting up of a connection between the client and the server. This choice was made mainly because it guarantees greater safety in communications between the two and provides an opportunity to control the reception of messages, checking that each request and reply is received once and only once and in the right order.

The setting up of a connection between the client and the server allows better safety management, based on a preliminary user authentication mechanism thanks to which the Lab Server can ascertain the identity of a user requesting a connection. Using symmetric (DES) and public key (RSA) encryption mechanisms [10], user authentication can be achieved by means of passwords and it is also possible to establish a session key to be used during communication.

V. ARCHITECTURE

The VL could physically be made up of a single server that the users of the client machines can access via the Internet (or a company Intranet). However, certain design decisions can be made to improve the characteristics of the VL. It is possible to achieve greater system reliability, power and flexibility by replicating the lab server, to avoid the presence of a centralized component constituting a single point of failure and a bottleneck for the processing of all the requests received.

With replication, the client side of the application has a complete list of all the available servers and the services they can support. This list of network addresses is embedded in the LabClient class. In this way, the user does not need to know the servers that can be contacted; when he wants to set up a connection, his client will choose the server. The choice is made on the basis of a *multicast requests* assignment algorithm: the client sends the connection request in multicast to all the servers at the same time and chooses the first to reply. This algorithm guarantees the best performance if the servers' delay in replying to a connection request is proportional to their current workload, and is based on the assumption that the first server to reply is the one with the lowest workload and therefore the one which will in the future ensure the best response times for interactive users.

The other design choice concerns distributed allocation of work, i.e. the possibility a server has of not executing a client request but delegating actual execution to further machines, called 'slaves'. This choice may be in reality an obligatory one, due to the *inherent distribution* of some labs or industrial plants. In addition, it may also be convenient to provide forms of distributed allocation for the execution of remote simulations on powerful machines: on receipt of a simulation request, the server could decide to assign it to one of the available slaves, applying a job allocation algorithm. In the case of our VL, the aim of this algorithm is to minimize response times, first by establishing a scale of correspondence between the weights of the operations requested (e.g. simulation requests, read/write operations on tutorials) and the capacity of the machines (which depends on both their processing power and their current workload), and then trying to assign each job according to a *best fit* policy (i.e. the job is assigned to the machine with the smallest capacity suitable for execution of an operation of that weight).

In reality, this job allocation algorithm can be applied not only to simulations and tutorials, for which the availability of several machines with equivalent functions is reasonably certain, but can also be used for the execution of operations involving measurements and monitoring remote plants or labs, according to specific assembly modes. The decision may be made, for example, to provide redundant hardware for interfacing with the plant, so as to allow access through several slaves. In this way it is possible for the VL to be used simultaneously by several users who can customize different portions, activating locking mechanisms to allow exclusive access to some lab resources for the setting of portions of the plant according to the user's specifications. The use of several slaves gives the desired degree of fault tolerance for access to the plants, preventing the failure of a single slave from completely jeopardizing access to the resource.

In short, after making the desired design choices (server replication and distributed job allocation), to enhance the performance offered by the VL (in terms of both response times and fault tolerance) the overall organization is as follows: a remote Internet user uses his browser to link up with any one of the available servers (applying the multicast request algorithm).

While the connection is being set up the user is authenticated and submits his requests. The server chosen in turn applies the job allocation algorithm to select the slave which is most suitable for execution of the services requested (simulations, remote monitoring, etc.). The results of the operation then take the reverse direction, from the slave to the server and then to the client used by the user.

It should be pointed out that this scheme is a very general one and can be used by client, server and slave machines communicating via the Internet. It is the loosest possible communication scheme and can be modified to cope with requirements of efficiency, reliability and security. The servers and slaves, for example, can belong to the same company Intranet based on a high-speed LAN (e.g. FDDI), so as to speed up server/slave communications and increase security against outside access (Fig.2).

In the VL scheme outlined so far, all the user/slave interactions have to go through the server, which always acts as an intermediary. It is, however, possible to envisage a scheme in which the server only acts as an intermediary in a preliminary contracting and authentication phase, and then leaves the client free to communicate directly with the slave. The two possibilities each have their advantages and disadvantages: bypassing the server in client/slave communications speeds up the transmission and processing of messages, but keeping the server as an intermediary guarantees greater security as the server centralizes management of user access to the various resources.

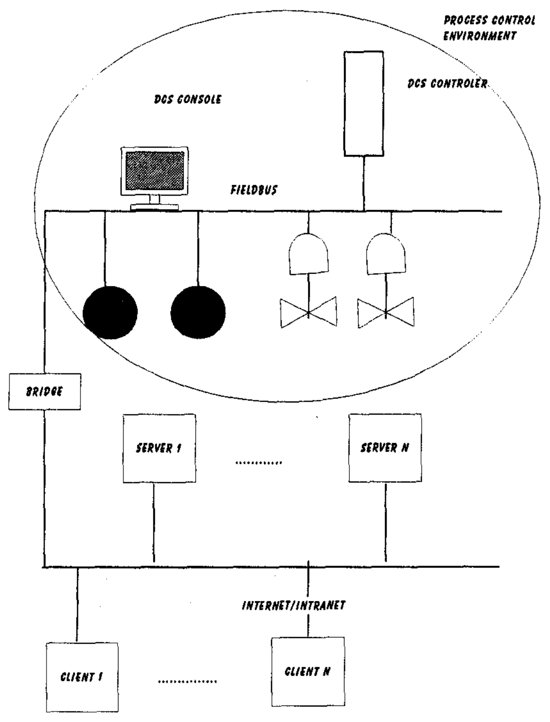

Fig. 2: Architecture of VL

A. Security

Security is a critical issue in the VL. Two security aspects have been taken into consideration:

- access security, which is ensured by developing a model derived from the one proposed by Bell and La Padula (BLP) [8][9] using an authentication protocol [10] and various encryption algorithms [11][12];

- security in operations performed on real plants. This is dealt with by the Lab Security Manager, an active programmable entity in the server whose aim is to check that requested operations will not lead to any dangerous situations. As said, the Lab Security Manager is an active entity, in that it can autonomously survey the plant before deciding whether the new operation is safe or not. It is programmable in the sense that it is implemented as a Java class, so it is possible for it to contain high-level programming of all the physical constraints of the plant and interactions between its various parts.

Using a high-level language like Java it is possible to implement a safety scheme for operations on the plant that does not refer to predefined models but is customized according to the requirements of the application. To do so, before authorizing the execution of an operation, the Lab Security Manager performs all the checks needed to ensure that the physical constraints of the plant are respected (e.g. constraints linked to the type and topology of the devices involved).

More specifically, the algorithm used by the Lab Security Manager to verify the correctness of the operations requested is as follows:

```
   // Lab Security Manager Algorithm

   if (user authenticated) {
    while (true) {
       request = receive();
       if              (checkAccess(request.user,
request.resource) &&
          checkCommand(request.resource,
request.command)) {

          reply = exec(request);
          send(reply);
       } else {
          disconnect(request.user);
       }
    }
   }

   // Check user authorization for resource

   boolean checkAccess(user, resource) {
    return (AccessMatrix(user,resource));
   }
```

```
  // Check safety of command

  boolean checkCommand(resource,command) {
   if (CurrentBlockingSet(resource))
      return false;
  return
(ResourceInteractionSet(resource,command));
  }
```

Fig. 3: Algorithms used by the Lab Security Manager

As the fragment of Java-like code in Fig. 3 shows, the Lab Security Manager authorizes requested operations only if both the Boolean methods *checkAccess()* and *checkCommand()* give a positive result. These methods use the information stored in the data structures of the security model. More specifically, *checkAccess()* verifies in the *Access Matrix* whether the user has the right to access the resources (authorization is provided on registration of new users but can be modified subsequently), while *checkCommand()* verifies that the remote command requested is not dangerous. This method uses the information stored in the *Current Blocking Set* structure, which contains the set of resources which cannot be accessed currently because they are being used, and the *Resource Interaction Set*. The latter is the active programmable entity in which the physical constraints of the plant and those on interaction between its various parts can be wired.

Lastly, the VL security model has another structure, the *User History*, which logs the activities each user has performed on the lab resources since his registration. This is useful in various ways: it provides information as to which user has performed a certain operation, and can implement accounting mechanisms or even assign timed passwords.

VI. CONCLUSIONS

In this paper we have presented the VL, which allows distance learning using a remote lab. Such a lab has to provide various tools and real resources, here classified as simulators, tutorials and real instruments. The implementation of a tool of this kind is made possible by the appearance in the Internet and network scenario in general of distributed programming languages oriented towards flexible implementation of Client/Server interaction.More specifically, the VL was implemented using the object-oriented Java language. In the paper we have identified the type of interface the user can be offered, assuming that he can interact with the lab at various levels of expertise. The GUI, for example, supports simple and menu-guided interaction, while the API allows interaction independently of a rigid scheme, which may require detailed knowledge of the characteristics of the lab.

We have described the general organization of the VL and the implementation choices made in order to increase the efficiency of resource management, provide adequate communication for the information exchange typical of the lab and ensure flexible, customizable service configuration.

Great importance has been attached to protecting the lab, in terms of security by preventing unauthorized access to the resources available and in terms of safety by prohibiting dangerous operations.

Devised as a tool for distance learning, the VL can be endowed with other functions to support remote-access control of a real industrial plant.

When applied to the control of industrial systems, the concept of Virtual Lab allows the monitoring and remote control of plants of any type, making it unnecessary for whoever is in charge of establishing the production cycle, or diagnosing and solving any malfunctioning, to be physically present in the plant itself: ideally he can access the plant from any part of the world via the Internet.

VII. REFERENCES

[1] B. Aktan, C.A. Bohus, L.A. Crowl, and M.H.Shor, 'Distance Learning Applied to Control Engineering Laboratories', *Tech. Rep., in IEEE Transactions on Education*, vol. 39, August 1996, pp. 320-326.

[2] J. Gosling, and H. McGilton, 'The Java Language Environment: A White Paper', *Tech. Rep., Sun Microsystems*, 1995.

[3]Sun Microsystems, 'The Java Language Specification', *Tech. Rep*., 1995.

[4] M. Campione, and K. Walrath, 'The Java Tutorial', *Tech. Rep., Sun Microsystems*, 1996.

[5]Sun Microsystems, *Basic & Advanced Java Programming*, 1995.

[6] J.A. Bank, 'Java Security', *Tech. Rep., Massachusetts Institute of Technology*, 1995.

[7]A.S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992.

[8]D. E. Bell, L.J. La Padula, '*MITRE Technical Report 2547*', Vol.1 e 2, March 1973.

[9]S.Castano, M.Fugini, G.Martella, and P.Samarati, *Database Security*, Addison-Wesley, 1994.

[10]B. Schneier, *Applied Cryptography (2nd edition)*, John Wiley & Sons, 1995.

[11]W. Cheswick, and S. Bellovin, *Firewalls and Internet Security*, Addison-Wesley, 1995.

[12]W. Stallings, *Network and Internetwork Security*, Prentice-Hall, 1995.