

# Distributed Supervision and Control of Fieldbus-Based Industrial Processes

Martí, P.  
Automatic Control Dept.  
Technical University of Catalonia  
C/ Pau Gargallo 5, 08028 Barcelona, Spain  
pmarti@esaii.upc.es

Rolando, F.  
Dip. Automatica e Informatica  
Politecnico di Torino,  
C. D. degli Abruzzi 24, 10123 Torino, Italy  
fabio.rolando@write.me.com

Colomar, P.  
Automatic Control Dept.  
Technical University of Catalonia  
Pla de Palau 28, 08003 Barcelona, Spain.  
jcolomar@bec.fnb.upc.es

Aguado, J.C.  
Automatic Control Dept.  
Technical University of Catalonia  
C/ Pau Gargallo 5, 08028 Barcelona, Spain  
jaguado@esaii.upc.es

Velasco, M.  
Automatic Control Dept.  
Technical University of Catalonia  
Pla de Palau 28, 08003 Barcelona, Spain.  
mvelasco@bec.fnb.upc.es

Fuertes, J.M.  
Automatic Control Dept.  
Technical University of Catalonia  
C/ Pau Gargallo 5, 08028 Barcelona, Spain.  
pepf@esaii.upc.es

## Abstract

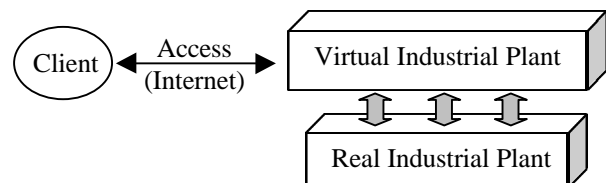
*This paper shows the application of an ongoing research project on the distributed control of processes. The objective of the project is to provide a framework to virtually model industrial plants, and to remotely, or even locally, supervise and control via Internet any industrial process by mapping it to a virtual plant. This framework provides, by using objected oriented methodologies, general patterns to implement communication needs and to represent virtual industrial processes and their control. The application of this architecture on a fieldbus-based process allows the control and monitoring of such systems with independence from both their physical implementation and location.*

## 1. Introduction

The increasing utilization of flexible and powerful distributed (control) systems and the expanding role of Internet and Intranets as communication backbones are presently characterizing the industrial automation field. These two tendencies are still presenting some drawbacks, the main of them being the excessive and unpredictable delay that a remote Internet connection nowadays implies. In spite of this problem, it can be accepted that process supervision and punctual control actions can nevertheless be adequately performed by

means of the existing communication networks and technologies, no matter how distant the plant can be. This possibility, mainly when it could be the only chance of minimally controlling a remote process, makes worthy to explore the Internet-based control.

A few attempts have been done in order to combine these two tendencies, but past implementations were only specific solutions for a particular application, and it is highly desirable to try to plan a more general approach. The goal of the proposed project [6] is to present a *general* framework and a general working methodology to accomplish a remote or local distributed supervision and control of any industrial processes via Internet (Figure 1).



**Figure 1. Project general scheme.**

The motivation of the project is twofold. On the one hand, it is aimed at finding a global solution that brings together the above-mentioned main tendencies in industrial automation, distributed control and Internet. It has to be remarked that this solution must overcome the lack of integration that physical components involve. On

the other hand, a practical application of the final product is pursued.

Our first goal, then, its to set industrial plants free from their physical reality, components and relationships among them, by means of virtual plants [5]. Those virtual plants will be worldwide accessible by means of the Internet and the TCP/IP (Transfer Control Protocol/Internet Protocol). It has to be stressed that at some point, when mapping our virtual plants to the real ones, our framework will have to deal with the particular operation characteristics of the real plant components.

The second objective of the project is the application of the developed architecture, in order to provide both the industrial and academic worlds (as in [11]) with a software tool that grants wider access to the supervision and control of industrial plants.

The aim of this paper is, after summarizing the most important project decisions and its development steps, to show its application on a fieldbus-based process.

Consequently, the structure of the paper is as follows. Section number two presents the project development and the third section describes the main parts of the framework, illustrating the general operation of the system. Section number fourth shows the application of the framework on a fieldbus-based process, and finally the fifth section draws some conclusions and points out some directions for future research.

## 2. Project development

The project development was divided into three main subprojects: preliminary study, communication scheme and virtual plant architecture.

In the early stages of the project, a preliminary study was made so as to determine which available technologies [2] [7] could be used to achieve the capability to perform remote supervision and control actions, and which methods would better suit our purposes. As a result, a first part of the communication scheme was developed. The next step mainly consisted in the study, as in [9], of the mechanisms able to provide the communication and information exchange between a real and a virtual plant. After that, the design and implementation of the overall communication scheme, which the virtual plant was to be built upon, was carried out.

As a final step, a brand new architecture was created to virtually model industrial plants, by means of object oriented methodologies, and with the objective of either remotely or locally supervise and control these plants. By following a client-server philosophy [1], the necessary structures were devised to represent a virtual plant, which was to be either partially or totally reflected in the client entity.

### 2.1. Preliminary study

As it was commented before, the first question was to determine the requirements of an environment aimed to modeling, supervising and controlling industrial processes. The first conclusion from these requirements was the necessity of employing an object-oriented methodology [3][4]. That way, the environment turns out to be flexible, scalable and reusable, meanwhile modeling each component of the plant as an object assures their intercommunication just by invoking their methods.

Once the general approach was chosen, the next aspect was the particular object-oriented language and its developing environment. The core of the whole project was the remote communication via Internet, and also an open interaction with other software and firmware. Two solutions appeared fulfilling those requirements, namely Java™ and CORBA [8].

The advantages of Java™, included, besides the possibility of programming the virtual plant, on the one hand. the Internet communication by means of the API RMI (Remote Method Invocation) [10] (see Fig. 2). On the other hand, it allowed by using JNI (Java™ Native Interface) the needed interaction with different software and firmware.

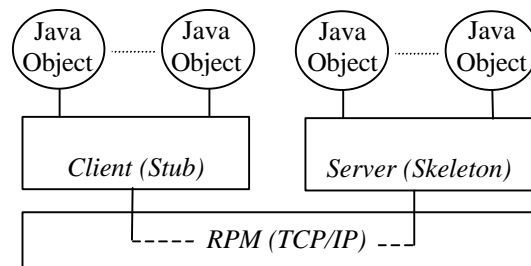
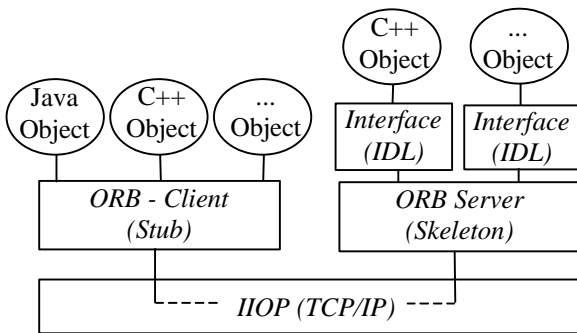


Figure 2. Java™ environment

CORBA stands for Common Object Request Broker Architecture. This platform permits remote and local interaction among processes written in different programming languages [8], thus solving both the communication through Internet (see Fig. 3) and the collaboration with other software and firmware. Nevertheless, it was clear that by choosing this solution, the problem of selecting a programming language for the implementation of the project was merely postponed.

Both platforms, according to their documentation, presented very similar characteristics, but in order to determine their true performance, a communication test was implemented. This experience tried not only these high-level solutions, but also included the low-level implementation of Sockets, in order to assess the delays induced by these more general architectures.



**Figure 3. CORBA environment**

It is not relevant in this context to describe all the experimental test details. The values found do not have an objective importance when considered alone, but are interesting to establish some comparisons among the technologies.

Talking about a distributed control environment, the obtained timing results are important whenever it is necessary to execute operations requiring a repeated invocation of the same method in the same object. If the execution of a simple punctual operation is considered, the timing results stop being that significant. In this case, a *set-up time* can be observed, during which the object is remotely identified, instantiated and initialized, before the client can actually invoke a method. The set-up time is certainly much longer than the time of a single invocation. The implementation with Sockets was actually the fastest one, followed by RMI and CORBA. But the task of writing code, using Sockets, able to run the communication between a client and a server, can be quite a challenging work.

RMI showed anyway quite a good performance. The new implementation of this technology adds important features that have been for years a privilege of CORBA developers. Some of them are the persistence of an object reference, the facilities for remote objects, the support for custom socket type and other improvements that, for instance, brought to a significant performance improvement between the releases 1.1 and 1.2 [7].

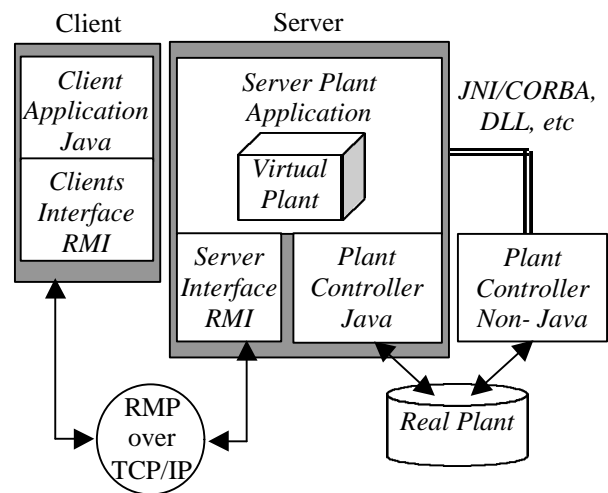
CORBA has the advantage of being a deeply tested solution for distributed computing. In addition, the support for most major programming languages makes it probably the better choice in case the developers are faced to the need of using many existing code sources in non-Java™ languages.

Nevertheless, CORBA is a heavier technology, compared to RMI, to build an independent and easy to use communication platform. Java™ was finally chosen as the programming environment because that way the user would only need a web browser –including the Java Virtual Machine- to remotely access any plant.

## 2.2. Communication scheme

After selecting Java™ as the design platform, the next task was the definition and implementation of a

communication scheme (Figure 4). In this scheme, an industrial plant is assumed to be always connected to a plant controller program that acts as a link between the real and the virtual plants. If that controller is written in Java™, the communication to the virtual plant can be straightforward. Nevertheless, the most common case implies an existing software performing the plant control. In this case, it is necessary to integrate the software libraries and drivers by means of DLL, JNI or CORBA to enable the communication. But the situation can be worst if the controller does not provide these drivers and libraries, and a specific solution has to be found.



**Figure 4: Communication Scheme**

The plant controller interacts with the *server plant application*, always coded in Java™, which models the virtual plant. This application, which will be further described within the next section, allows both the communication with the plant controller and a number of clients, by means of the application interface RMI.

It is important to remark that this communication architecture permits the existence of several virtual plants, either corresponding to real ones or not. On the other hand, it also permits the existence of several clients, accessing one or more virtual plants.

One of the goals of the project was to achieve full operation with a minimum number of resources. According to that philosophy, the client machine only needs to run a web browser including the Java™ Virtual Machine (JVM). In the same way, the server needs the JVM and (optionally) a web browser. For the full operation of the framework, it is also needed, in the server side or in another machine, a small non-commercial http server running the JVM.

In this communication scheme, it is basic to address the security issues, both from the point of view of Java-provided capabilities, and from the perspective of the programmed application. The aim is to have at our disposal a secure platform where Java-enabled

applications can run. The platform itself (JDK, Java Development Kit) provides tools and services to adjust the security of the platform and the applications that must run on top of it.

The Java platform has a security model that is changing in each release. In the very last Java release, every required level of security can be defined within local and remote code. RMI provides also the Security Manager which helps the developers to protect the local resources from code that could damage them and, in general, allows the programmer to implement quite a complete security system [8].

In the project, security levels are defined and every client is forced to fit in a given profile that grants read and modification access to any subset of a virtual plant. However, further research in this area is still needed.

### 2.3. Virtual Plant

The virtual plant structure is the key point where the whole project is based, and it comes determined by three important characteristics.

In the first place, a virtual plant has to be available to represent any real industrial plant, any of its components and any inner relationship among them. In other words, generality is needed.

A second important aspect is the necessity of a remote access to the plant. This implies, on the one hand, that the remote client must know how to get to the plant and its elements in order to construct its own partial or complete image of the virtual plant. On the other hand, it is necessary to assure the perfect correspondence between any reference to the plant or its components in the client and the plant or its elements in the server, in order to assure correct operation.

Finally, this virtual plant is designed in order to interact continuously with a real environment. This means that any configuration or change must be dynamic and allowed in execution time.

The chosen structure for the representation of the virtual plant is very simple. Maybe the main feature of an industrial plant is the capability of recursively include other plants, and that property can be represented by a structure of “containers”. Thus, a *container* will be the basic unit of a virtual plant, and it will present two important treats:

- A *container* can contain and be contained by another *container* or group of them.
- A *container* can be remotely configured and can remotely supply information about its content.

The first property allows the existence of complex data structures. The structure would be tree-like with intermediate nodes and leaves.

The second property allows the characterization of each plant component (container) by using properties, collecting every property of the included *containers*.

The *container* structure can be created and properly connected by means of the configuration procedure (figure 5) in both the server and the client side. After its application, the whole plant and any of its components make their properties available.

So as to configure and access the plant, both server and client are provided with the same graphic interface that shows the tree structure as it is being created. This environment also allows determining the properties of any container, finding out if its properties are structural or physical and characterizing its remote access policy.

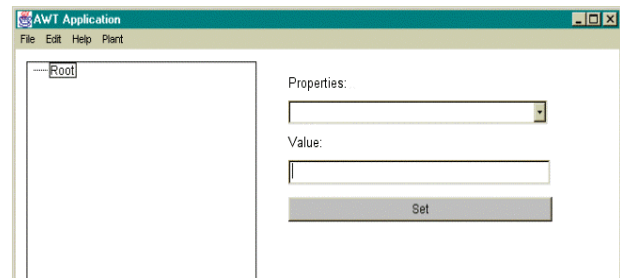


Figure 5. Configuration procedure

When creating a new node (container), a few characteristics have to be set (see Figure 6). First, whether the new node will be a local element or an instance of a remote element. On the server side, all new nodes are usually local elements, meaning that they are virtual representations of the real plant elements where they are attached. On the client side, all nodes will usually be instances of a remote element (node on the server side).

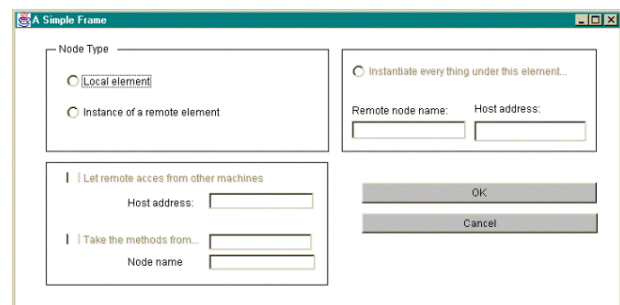


Figure 6. Container configuration

Additionally, it is important to determine if the current new node will permit remote access from other machines. Usually, on the server side, every node will be accessible, but it will not be so on the client side. Every node will also have a name, and in the case of local element enabling remote access, the host address where the object can be found has to be provided. If the new node is an instance of a remote element, the remote node name must be provided as well as the host address where it can be accessed.

Within the configuration procedure, other operations are allowed. There is a property specially worthy of being remarked: the real location of the variable, which

will completely determine by its value the way it could be accessed.

The implementation of *containers* in Java™ is accomplished by means of an object derived from the available *HashTable* structure. Its methods have been properly modified to enable remote accessibility. Each *container* includes four methods that allow both server and user to build up any virtual plant.

### 3. Operation

The real operation of the presented framework can distinguish three logical entities (not necessarily physical ones), clients, plant servers and an object server. Let us analyze each element.

#### 3.1. Plant Servers

The Plant Servers (PS) are the entities where the control and supervision processes are executed. The Plant Servers, that model virtual plants, execute the software able to communicate directly with the real plant through the appropriate interfaces, if it is required. Depending on the plant, one or more Plant Servers can be necessary. Obviously, the communication path between PS and real devices is bi-directional.

A single Plant Server can be described as a software entity able to execute one or more processes, providing the services needed to represent, control and supervise the plant. Each one of these services will be called Server Service (SS). A SS that interacts with a plant controller has certainly to be developed closely integrated with the real plant. In different contexts, the SS could be part of PS application (both written in Java, probably by the same developers), or part of a stand-alone application written in whatever language.

#### 3.2. Clients

The most general happens when there is more than one client, and they are placed remotely and connected to the Object Server through a TCP/IP network. The existence of local clients or a single client are simpler possibilities, which are included in the general case. The requirements for a client are basically simple: it has to be a device able to browse the Internet through a Java 2 compliant browser. At the moment, this requirement means the existence of a generic computerized device running an operating system and supporting the any of the most popular browsers.

#### 3.3. Object Server

This component has a crucial but simple role in the system. Its main task as a naming service is to operate as an interface between the Clients and the Plant Servers. Plant Servers will publish objects and methods on the Object Server (OS) that will be accessible by Clients.

RMI offers a simple application called *rmiregistry* that provides a naming service. A service offered by the

*rmiregistry* is used by the servers to register the names of the objects and methods they offer. The *rmiregistry* keeps track of the objects' names and offers them in a way they that can be invoked by the clients. The client has to make a request to the *rmiregistry* in order to have access to the remote object.

An http server with the JVM is running on the OS providing the registration procedure, which can contain all the information about the plant to be controlled, and the services offered by the system. One or more applets are available in this server so that servers and clients can download and execute them to carry out all steps to set the framework up for the supervision and control.

Due to limitations on the RMI capabilities, a Java application, called Central Object Register (COR), has been implemented in the OS to accomplish a crucial task. The Object Server running the http server executes the *rmiregistry* and runs the COR. Each Plant Server communicates with the COR using RMI, but acting as a client in this context. The objects and their methods are then registered in the central *rmiregistry* by the COR. Although the clients can also communicate with the COR to retrieve the information about the available objects in the system, usually they directly contact the *rmiregistry* in order to obtain it. It is at this point where the http server is needed to facilitate the exchange of the skeleton and stub files between clients and the *rmiregistry*. After this phase, the clients can invoke the methods communicating directly with the plant server, without using the Object Sever any more.

It is necessary to remark that the role of the Object Server is important because a breakdown of this entity can be critical. There can be a number of clients and Plant Servers but the only Object Server has to work properly.

#### 3.4 The operation step by step

The operation step by step can be seen in Figure 7. Before starting the normal operation of the framework (building a new virtual plant and allowing clients to access the published elements), the Object Server components (http server, *rmiregistry* application and COR application) must be running on a machine, and they must be accessible through a web page from where the configuration procedure (Figure 5) can be downloaded by both servers and clients. Therefore, the only thing that clients and servers have to know to start to work with this framework is an internet address.

The normal operation begins when the Plant Server, through the configuration procedure obtained from a known web page, registers (step 1) the plant and each of its elements using the *rmiregistry* (via the COR) running in the Object Server, in order to make them accessible to the clients. From that moment on, a client can contact the same web page by means of a web browser (step 2) and can also obtain all the necessary tools (configuration procedure and communication profile) that will allow to

start the configuration of the virtual plant in the client side (step 3).

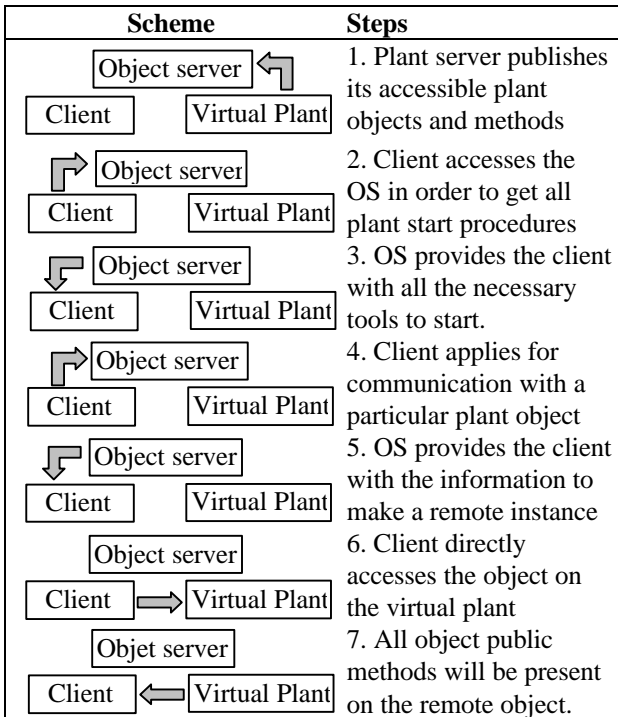


Figure 7. Step by Step

Beyond this point, any new *container* (element of the plant) that the client knows (through the COR or *a priori*) and wants to access, has to be first “ordered” to the *rmiregistry* on the Object Server (step 4). After that, the OS provides the Client with all the necessary information to make the remote instance of the “ordered” element (step 5).

After that, the client can directly access the element, which will be a remote instance of a local object on the server side (step 6), and get all the public methods (step7) that will be “present” on the remote object. Once the virtual plant has been configured within the client, the Object Server no longer appears in the communication scheme.

The RMI technology always imposes a certain time offset or delay in the remote object re-creation. Fortunately, it only happens in the instantiation, meanwhile Internet completely determines the response times when in the regular operation.

The remote representation of a plant can be addressed from two different points of view. A first possibility is obtaining the structure of the first element, and then performing the remote instantiation of every found object. After that, the process is repeated and the tree-like structure is completely explored level by level. The advantage of this approach is the availability of every element in execution time. But of course the main drawback is the accumulated delay when the connection is first established.

A complementary solution creates instances of the elements only when the user needs them. Obviously, this policy implies a fast first access to interesting variables, but a little delay whenever the crucial structures have not been instantiated yet.

#### 4. Application

Applying this framework on fieldbus-based industrial processes is one of the possibilities of our project. The following example (see figure 8) illustrates the overall system behavior when monitoring and controlling the speed of an engine accessible through Profibus DP from an industrial PC.



Figure 8. Application Example

In this example, the proposed framework is used to model a simple fieldbus-based process. It must be pointed out that the resulting virtual plant on the server and client side are just particular representations of the same plant. Nevertheless, more detailed or even more general designs of the same real process can be also possible. Moreover, servers and clients can have different views of the same process.

In this case, to start the operation, the (plant) server needs to register the engine. To do so, the Object Server has to be in execution: the COR must be running somewhere (Fig. 9), along with the *rmiregistry* application and the *http* server. These three elements can be located in different machines, but usually they are located in the same one, under an initial web page or identified by the same IP address.

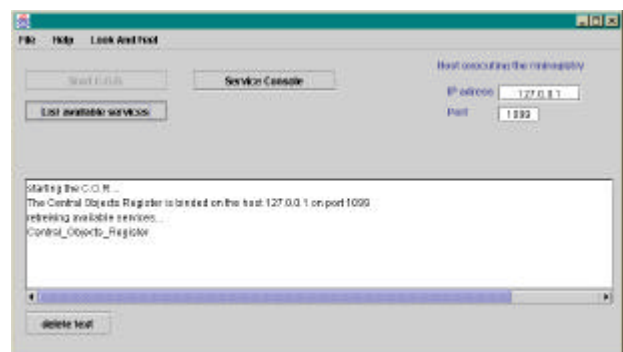
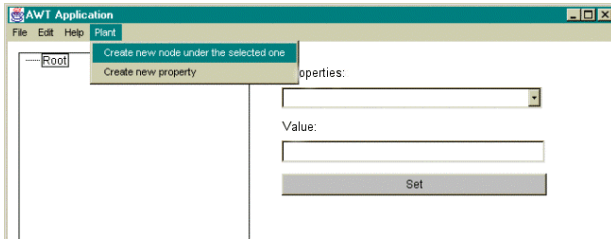


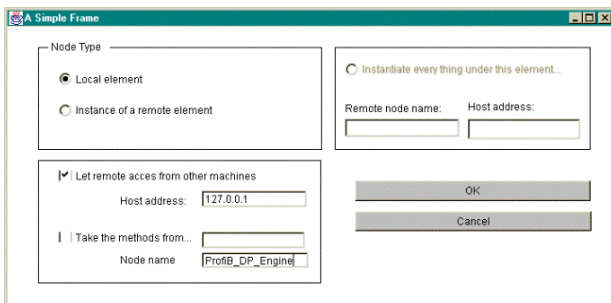
Figure 9. Central Object Register

After downloading the configuration procedure from the Object Server web page, the operation of modeling a virtual plant begins on the server side. In this simple case, the operation starts when the engine is created, as a first node (container) of the virtual plant, pending from the root tree-like structure (Figure 10).



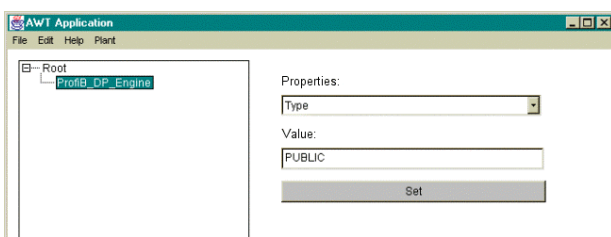
**Figure 10. Creating a new virtual plant on the server side**

As it was said before, the creation of a new node implies the need to set a few characteristics. Firstly, the new node must be identified as either a local element or an instance of a remote element. In this case, the engine will be a local element, enabling remote access from other machines and being accessible from the object server (see figure 11).



**Figure 11. Setting engine properties on the server side**

After that, the new created object, ProfiB\_DP\_Engine (Fig. 12), which is the virtual representation of the engine, is ready to be characterized by new properties.

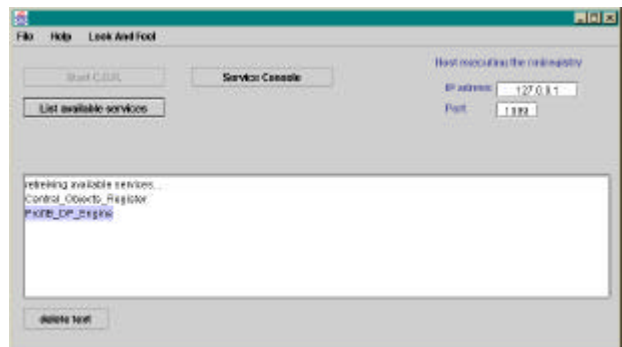


**Figure 12. Virtual engine on the server side**

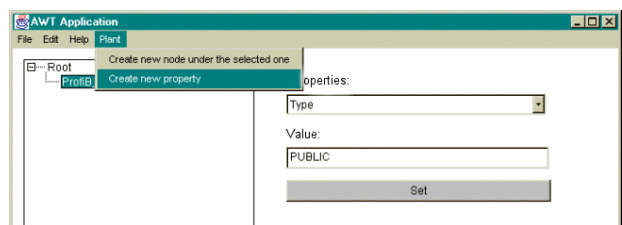
It is possible to keep track of this process also in the COR, because every object that is created has also been registered in the *rmiregistry* through the COR application (Fig. 13). The COR application will also allow clients to be informed of the different available services (registered objects) that the plants servers have made accessible.

Following the same procedure, a full virtual industrial process can be created in a tree-like shape. For every intermediate or leaf node, several properties can be defined. In this case, the new node corresponds to a real engine accessible through Profibus DP. Let us suppose

the speed of the engine is the real variable to be supervised and controlled. Therefore, a new property named Speed must be created (Figure 14) in order to have access to the real speed of the engine.

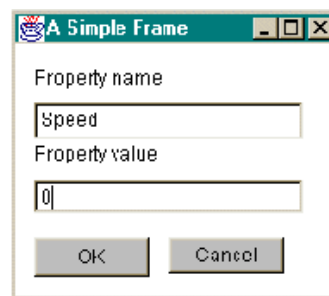


**Figure 13. Engine registered through the COR application**



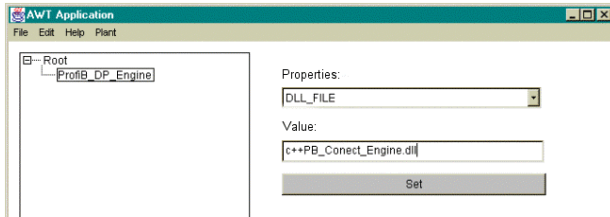
**Figure 14. Creating the speed property.**

When creating a new property (Figure 15), its name must be specified along with the initial value of this property. In the actual implementation of the framework, some default properties are also provided. The most important is the *DLL\_FILE*. Its value will specify the procedure used to access a real variable. Usually, this property will be filled up when it belongs to a final node in the tree structure because final nodes (leaves) are the candidates for representing the real variables of the plant.



**Figure 15. Defining a new property**

Once the property speed has been created, it is necessary to link this virtual property to the real speed of the engine. So, to finish with the configuration of the virtual engine on the server side, this link must be stated by filling the *DLL\_FILE* property. The value of this field will tell how to access the real speed value (see Fig. 16).

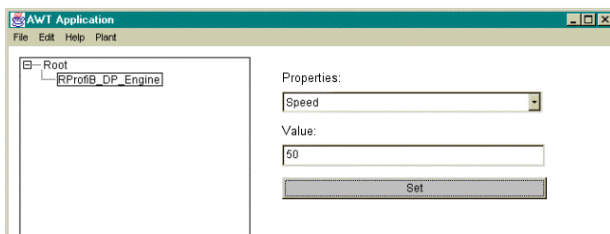


**Figure 16. Setting DLL\_FILE properties on the server side**

At this point, on the server side, the value of the speed of the engine is accessible by consulting the property value of the engine node.

Now, on the client side, a similar configuration process has to be carried out. The main differences occur when setting up the properties of the engine node, here called RProfiB\_DP\_Engine. The new node, with this new name, will be an instance of the remote element ProfiB\_DP\_Engine. Therefore, the remote node name and the host address where the remote node can be found (where the *rmiregistry* is running) must be specified.

Finally, by setting the property value on the client side (Fig 17), the value of the speed object on the server side will be modified, and the real speed of the engine attached to Profibus DP will change to the specified one.



**Figure 17. Speed value on the client side**

Also, if the purpose of the client is just to monitor the speed of the engine, just by selecting the property speed, the actual value of the speed will appear in the value field due to an automatic refreshing activity.

## 5. Conclusions

The main objective of the project was the design and implementation of a software architecture that grants distributed supervision and control of industrial plants via Internet.

The project contribution takes profit from two main forces in the industrial control field, the wide-spread usage of Internet and the improvements in computer networking capabilities, which have started to permit real distributed control, or at least remote supervision.

The applicability of the project has been shown at the end of this paper, by employing this framework for monitoring a fieldbus-based process through its virtual representation both on the client and the server side.

Future work will focus on security issues of the communication scheme and data consistency of the distributed framework. Also, real-time performance of some parts of this architecture is desirable. Finally, the framework will be tested in a real and complex industrial process to check all its real capabilities.

## Acknowledgements

The presented research has received support from Spanish CICYT project ref. TAP98-0585-C03-01.

## References

- [1] Aldrich J., Dooley J., Mandelsohn S., Rifkin A., (1998) "Providing Easier Access to Remote Objects in Client-Server Systems", Proc. of the 31st Hawaii Int. Conf. on System Sciences, vol. 7, pp. 366-375.
- [2] Curtis D. (1997) "Java, RMI and CORBA" a white paper by the Dir. Platform Technology Object Management Group. <http://www.omg.org/library/wpjava.html>
- [3] Dollimore, J. (1997) "Object-based Distributed Systems" Draft material for 3<sup>rd</sup> edition of Distributed Systems – Concepts and Design. DCS, University of London.
- [4] Guijun W., Ungar L., Klawitter D. (1998) "A Framework Supporting Component Assembly for Distributed Systems", Enterprise Distributed Object Computing Wk., 1998. Proceedings, pp. 136-146.
- [5] Lump T., Gruhler G., Kuchlin W. (1998) "Virtual Java Devices. Integration of Fieldbus Based Systems in the Internet", IECON'98. Proc. of the 24<sup>th</sup> Annual Conf. of the IEEE Ind. Electronics Society, Vol. 1, pp. 176-181.
- [6] Martí P., Aguado J.C., Rolando F., Velasco M., Colomar J. and Fuertes J.M. (1999) "A Java<sup>TM</sup>-Based Framework for Distributed Supervision and Control of Industrial Processes". 7<sup>th</sup> IEEE Int. Conf. on Emerging Technologies and Factory Automation, Vol 1, pp. 33-41
- [7] Medeiros D. J., Watson E. F., Carson J. S. and Manivannan M. S. (1998) "Distributed Simulation Modelling: a comparison of HLA, CORBA, AND RMI", Proceedings of the 1998 Winter Simulation Conference.
- [8] Orfali R., Harkey D. (1998) "Client/Server Programming with Java and CORBA" Second Edition. John Wiley & Sons, Inc., New York.
- [9] Reiter H., Kral C. (1997) "Interaction between Java<sup>TM</sup> and LonWorks<sup>RM</sup>", 1997 IEEE International Workshop on Factory Communication Systems, pp. 335-340.
- [10] Sutherland D. (1997) "RMI and JAVA<sup>TM</sup> Distributed Computing", JavaSoft<sup>TM</sup>, A Business Unit of Sun M., Inc. <http://www.javasoft.com/features/1997/nov/rmi.html>
- [11] VCLab (1999) Virtual Control Lab 2.1. Control Engineering Laboratory. Dept. of Electrical Engineering and Information Sciences Ruhr-University Bochum. <http://www.esr.ruhr-uni-bochum.de/VCLab/main.html>