IMTC 2005 – Instrumentation and Measurement
Technology Conference
Ottawa, ON, Canada, 16-19 May 2005

A MOBILE AGENT PLATFORM FOR REMOTE MEASUREMENTS

S. De Capitani di Vimercati [(1)], A. Ferrero [(2)], M. Lazzaroni [(1)]

[1] Dipartimento di Tecnologie dell'Informazione - Università degli Studi di Milano
Via Bramante, 65 – 26013 Crema (CR), Italy
Phone: +39 02 503.30073, Fax: +39 02 503 30010, Email: lazzaroni@dti.unimi.it

[2] Dipartimento di Elettrotecnica - Politecnico di Milano
Piazza Leonardo Da Vinci, 32 - 20133 Milano – Italy
Phone: 0039 02 23993702, Fax: 0039 02 23993703, Email: loredana.cristaldi@polimi.it

Abstract – *Industrial applications require suitable monitoring systems able to identify any decrement in the production efficiency involving economical losses. The information coming from a general purpose monitoring system can be usefully exploited to implement a sensorless instrument monitoring an AC motor drive and a diagnostic tool providing useful risk coefficients. The method is based on a complex digital processing of the line signals acquired by means of a Virtual Instrument. In this paper a Genetic Algorithm implemented in a Mathcad environment performs the evaluation of the risk indexes from the processed line signals. The combination of Genetic Algorithms and Neural Network is also investigated as a promising possibility for the development of a reliable diagnostic tool. The risk coefficients derived from this approach are evaluated, discussed and compared to other indexes – in particular Fuzzy Indexes - introduced by the authors in previous papers.*

Keywords – ***Mobile Agents, Measurements, Diagnostic, Testing.***

I. INTRODUCTION

Mobile code (mobile agent) technologies are receiving a great deal of interest from both industrial and academic world. The ability to move computations across the nodes of a wide area network allows deployment of services and applications in a more flexible, dynamic, and customizable way with respect to the well-known client-server paradigm. Mobile agents are an emerging technology that makes the design, implementation, and maintenance of distributed systems a very ease task. Mobile agents reduce the network traffic, provide an effective means of overcoming network latency and, through their ability to operate asynchronously and autonomously of the process that created them, help us to construct more robust and fault-tolerance systems.

Yet, the wide acceptance of the mobile agents approach is hampered by the security issues that arise when executable content and associated execution state are moved among different computational environments.

In this paper the authors propose a Mobile Agent Platform implemented for the calibration of the instruments located at the customer factory, instead of the calibration laboratory.

II. MOBILE AGENT PARADIGM

Mobile agent systems provide a computing infrastructure upon which distributed applications belonging to different (and potentially untrusted) users can execute concurrently. The execution of a program on a remote machine, that is, on a different machine from the one on which it is being operated, has been a current task since operations on Internet services, (e.g. *e-mail*, *ftp*, *telnet*), have been in use. Every service requests to activate a particular program able to serve it. This first model of remote code execution has definite limitations: it cannot put an arbitrary program in execution, since only the designated program that has been installed from the system administrator on the server, and only that one can be activated by a given service.

This type of operation is implemented as the realization of a virtual terminal, that is, it works as if the local keyboard and monitor and the remote machine were linked by "*long virtual cables*". In this case the execution is confined to only the application stored on the hard-disk of the machine where the code must be executed.

When applet Java and JavaScripts or programs written in VisualBasic have started to be integrated in the Web pages, the situation became more interesting: in fact, the executable program is not originally stored on the remote machine, but it is downloaded from the server and then executed. However their execution still remains confined in every single computer where the browser is installed and they are never executed on the server, where they exist only as latent codes.

The provision of services in an open, global, and mobile environment has significantly stimulated research work on new programming paradigms in order to enhance the flexibility of the traditional client/server model where client and server processes communicate either through message passing or remote procedure calls [1]. All proposed paradigms focus on the support of code mobility at runtime. In particular, a mobile agent is a program which represents a user in a computer network and can migrate autonomously from node to node in the network to perform some computation on behalf of the user. A mobile agent is then a program (encapsulating code, data, and execution context) sent by a server to a client. Unlike a procedure call, it does not have to return its results to the client but to the server. It could migrate to other servers, transmit information back to

```
void main()
  {
     int stay_here = 10;
     int counter = 0;
     while (stay_here--)
       counter++;
     migrate(new_IP_address);
  }
```

Fig. 1 – C-code of a simple program where a variable named *counter* is incremented. Moreover, the code is able to migrate on a different machine.

```
void main()
  {
     int stay_here = 10;
     int counter = get_previous_value();
     while (stay_here--)
       counter++;
     migrate(new_IP_address, counter);
    }
```

Fig. 2 – C-code of a simple Mobile Agent.

its origin, or migrate back to the client. It can be used for user-level applications, middleware, as well as system software.

Mobile agents are able to navigate on the net by themselves. In particular mobile agents interrupt the execution of their task on a client, migrate to another client (or server) and restart the execution from the precise point when they have been stopped.

A Mobile Agent, is therefore *a "program that can be moved between several machines conserving its state"*, where the state represents the values assumed by the data structures of the program. In Mobile Agents this feature is mandatory and it is their long term memory.

From this point of view viruses and Internet worms, for example, cannot be considered Mobile Agents: they are able to replicate themselves from a machine to another one, but they do not have conscience, because, after arrival on a new computer their execution starts again from the beginning. The state possessed by the program when the request that generates the copy has been issued is not propagated in the generated copy. Fig. 1 reports a simple program where a variable named *counter* is incremented and, sometime, the code migrate on a different machine. The C-code reported in Fig. 1 is not a Mobile Agent: in fact every time it arrives on a new computer, it starts its execution from *counter* = 0, forgetting the value reached on the previous machine. If it is possible to suppose that, passing the value of the counter to function *migrate*(), the C-code is able to recover this data on the new machine, for instance by means of function *get_previous_value*(), then a simple Mobile Agent is obtained. Fig. 2 reports an example of C-code implementing a Mobile Agent. The program state (in this simple case only variable "*counter*") is now preserved correctly. The more interesting and difficult part of the program consists in the implementation of the *migrate*() and *get_previous_value*() functions.

These functions must: *i*) save all variables that represent the state of the program; *ii*) send the executable code and the variables of the Mobile Agent on the destination machine; *iii*) restore the state of the program, once it arrived to destination: *iv*) re-start the executable code. Obviously, these implies the presence of an application named, for example, *agent server*, on all machines involved.

Telescript [2] was the first system expressly designed for programming mobile agents. It was followed by several prototypes such as Tacoma [3] and Agent Tcl 0, where agents are written using script languages.

The wide diffusion of Java language and programming environment with its support for mobile code, led to develop object oriented agent systems, which represent the fusion of mobile object systems with mobile agents. Aglets [5], Voyager [6], and Concordia [7] are examples of Java-based mobile agent systems.

The adoption of the mobile agent technology is encouraged by many researchers in the distributed system area [8] due to the several advantages of the mobile agent technology (e.g., reduced network usage, increased non synchronicity among clients and servers, and so on) in comparison with traditional message passing or remote procedure call paradigms. The possible drawbacks of the mobile agent technology are represented by the security risks introduced by the need to host the execution of new computing entities that carry their own code. Furthermore, an agent may be attacked, modified, or deleted by a hostile agent platform on a malicious network host. Another typically stated and obvious concern related to mobile agents is the question if agent migration is always of more effective than message passing. For instance, it could probably be better to interact by message passing in case the agent code is bigger than the expected data volume to be exchanged. We will discuss the security problems more in details in the following sections.

III. THE PROPOSED MOBILE AGENTS ARCHITECTURE

As far as the proposed architecture is concerned, its most basic requirement is to provide a facility for executing visiting agents and transporting them to other hosts upon request. Agents need to access system-level resources on their host machines, such as data, files, network ports, and so on. To this purpose the agent infrastructure needs to define a binding between the visiting agent and its environment or between two or more agents which need to communicate or to coordinate their activities. The runtime system is structured as a client/server system. Fig. 3 shows the architecture of the realized system at the server side. The architecture is composed of five modules (classes):

- The *Agent Launcher* creates an agent within the server process. A service can be activated manually, through a graphical interface (module Agent Manager) or automatically, through a request on a TCP/IP port (module KKMulti ServerThread).
- The *Agent Manager* creates the graphical interface used to

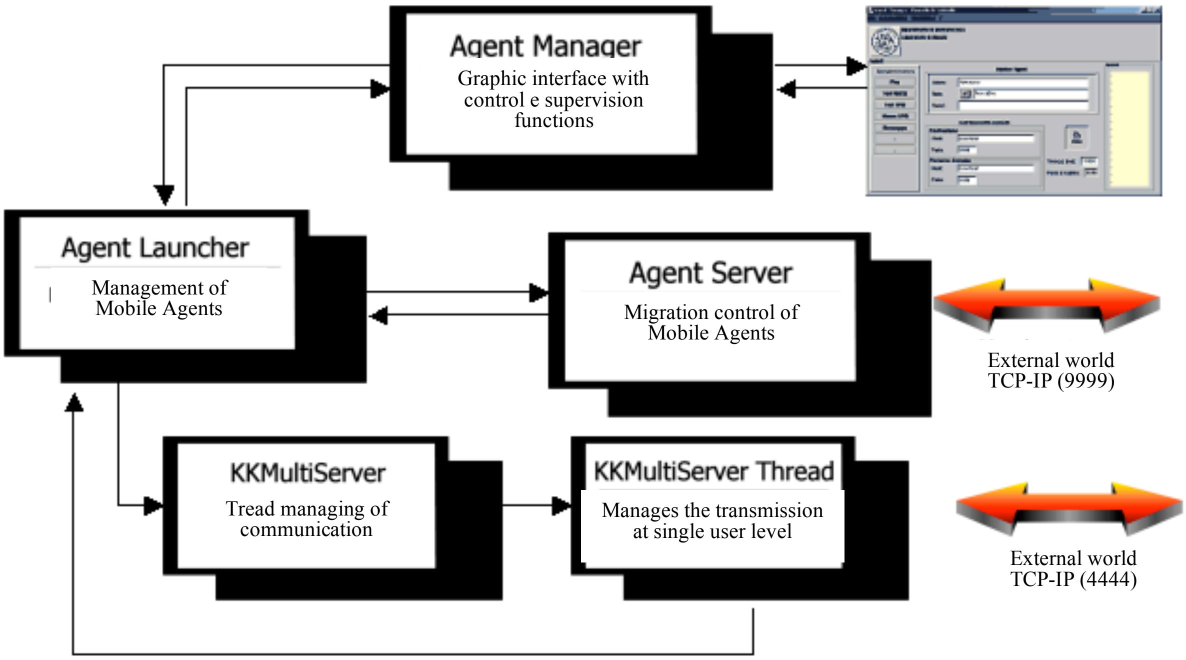

Agent Manager
Graphic interface with control e supervision functions
Agent Launcher
Management of Mobile Agents
Agent Server
Migration control of Mobile Agents
External world TCP-IP (9999)
KKMultiServer
Tread managing of communication
KKMultiServer Thread
Manages the transmission at single user level
External world TCP-IP (4444)

Fig. 3 - Architecture of our system at the server side.

interact with the Agent Launcher and takes care of monitoring the connected users and controlling the running agent.

- The *KKMulti Server* is responsible for the communication among servers, users, and agents. It creates a server socket and waits for a new connection. When a new connection is required, the KKMulti Server creates a new thread (KKMulti ServerThread) whose job is to serve that particular request.
- The *KKMulti ServerThread* manages the transmission of information such as the request of a service, the registration of a new user, and the error handling.
- The *Agent Server* takes care of receiving the migrated agents and executing agent code using an internal interpreter.

At the client side, the classes needed are those implemented in the Agent server module. In addition, the client has to notify to the server the termination of the service.

IV. IMPLEMENTATION OF THE PROTOTYPE

To demonstrate the applicability and the advantages of the solution in measurement applications, a prototype has been realized in Java by using the Aglets system. Aglets is a Java-based system developed by IBM where *agents* are named *aglets*. Java provides object *serialization* functions which allow to convert an object instance into a machine-independent array of bytes. The byte array can then be transmitted over a network to another host and de-serialized there (*i.e.*, converted back into an identical Java object). In our application, agents therefore are simply serializable Java objects. In Java, the source code for a class is first compiled into *bytecode*, which is then interpreted by a virtual machine. This *bytecode* can be transmitted over a network and reloaded as a class at the remote end. This implies that we can send both classes and objects from one machine to another. A remote agent server therefore does not need to have all of the agent's classes available locally beforehand. It can accept incoming agents and starts executing them. If the agent encounters a reference to a class which is not currently available, the Java virtual machine invokes the agent's class loader which can then contact the agent's home site to download the requisite *bytecode*. Agents migrate between agent servers (called *aglet context*) located on different network hosts. An interesting feature of Aglets is its callback-based programming model. The system invokes specific methods on the agent when certain events in its life-cycle occur. For instance, when an agent arrives at a server, its *onArrival* method is invoked. Agent mobility is realized adopting Java's object serialization. When an agent is reactivated at its destination, its *run* method is invoked. The programmer must implement further control flow in this method. Message-passing is the only mode of communication supported, that is, aglets cannot invoke each others' methods. The system provides a *retract* primitive that recalls an aglet to the caller's server. There is however no access control on this primitive and therefore it is possible for one user to retract another user's agent. We now briefly present the main classes of the realized prototype, to illustrate how the principles previously presented in the paper have been realized in practice.

Agent. It is the most important class that represents the agents. It is declared as abstract and therefore some methods are implemented when the agent is loaded. The agent migrates from a server to another one when a connection between the servers has been established.

Agent Server. It listens for agents. It is a multithread server, that is, multiple simultaneous connections are handled.

AgentHandler. It handles a single connection with a client. If there is the needed to use classes that are not stored in the server, it has to redefine the class loader through the class AgentLoader.

AgentClassLoader. It loads classes when needed. It uses an hash table with two columns: the first column is the class name and the second column is an array of bytes (the corresponding *bytecode*). This table includes all classes needed for the working of the agent.

AgentLoader. It is responsible of the agent serialization and deserialization.

V. SECURITY CONSIDERATIONS

Three apparent security problems arise when applying mobile agents:

- authentication (i.e., the determination of the identity of an agent or an agent system): the mobile agents need to be authenticated and authorized at the servers;
- secure communication and agent transfer: to ensure the integrity of the data; it must be transmitted in a secure communication channel;
- access control of resources/services depending on the requesting agent.

To guarantee confidentiality and integrity, crucial information such as code and state of a migrating agent should exploit, for example, public-key cryptographic encryption before transfer over an untrusted network.

In our prototype these issues have been addressed by setting up a Virtual Private Network (VPN) connection. A VPN is a connection that allows data to be sent securely over a shared or public network, such as the Internet. From the user's point of view, a VPN connection is a point-to-point connection between the user's computer and the server. VPN connections leverage the IP connectivity of the Internet and use a combination of tunneling and data encryption to securely connect remote clients and remote offices.

In the final paper a detailed discussion concerning security will be reported. In particular, a discussion concerning security in measurement applications will be reported and the proposed and implemented solution for security improvement of the realized application will be presented.

VI. USING MOBILE AGENTS IN MEASUREMENT: THE PROPOSED APPLICATION

We have seen how the technology of the Mobile Agents opens many doors. However, the use of the Mobile Agent is actually limited to the data-sharing application, database, and so on. Actually, the use of the Mobile Agent technology can be extended to control hardware and peripherals (such as calibrators, instruments and other devices), provided that they are connected to a host client through standard interfaces (RS232, IEEE 488, etc..) and specifics agents are developed to access the peripheral through these interfaces.

If these issues are correctly addressed it is possible to use Mobile Agent Technology to gain a full control of an instrument (or more instruments!). In particular the aforementioned technology can be used to calibrate a measurement instrument implementing the simple architecture depicted in Fig. 4.

Fig. 5 shows the Control Panel of the prototype Agent Manager for Mobile Agent launching implemented in Java. The realized Platform is fully operative and detailed discussion on obtained results will be reported on final paper. Finally, in Fig. 6 an example of Mobile Agent implemented is reported: the Mobile Agent is able to recognize the instruments connected by IEEE 488 to the PC.

VII. CONCLUSIONS

In the final paper a detailed discussion concerning the architecture of the Mobile Agent technology implemented will be reported. The designed and realized Mobile Agent System is able to perform a remote calibration of the measurement instruments and the obtained results, reported in the finale paper, are very interesting.

REFERENCES

[1] B.H. Tay and A.L. Ananda. A survey of remote procedure calls. *Operating System Review*, 24(3):68-79, July 1990.

[2] J.E. White. Mobile agents. Technical report, General Magic, Inc., October 1995.

[3] D. Johansen, R. van Renesse, and F.B. Schneider. Operating system support for mobile agents. In Proc. of the 5th IEEE Workshop on Hot Topics in Operating Systems (HotOS-V), May 1995.

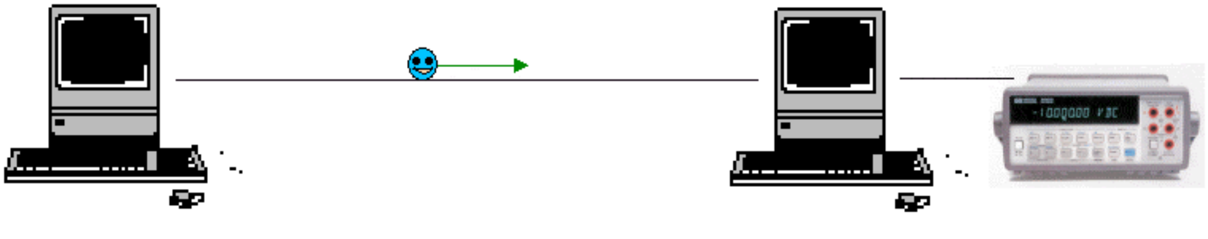
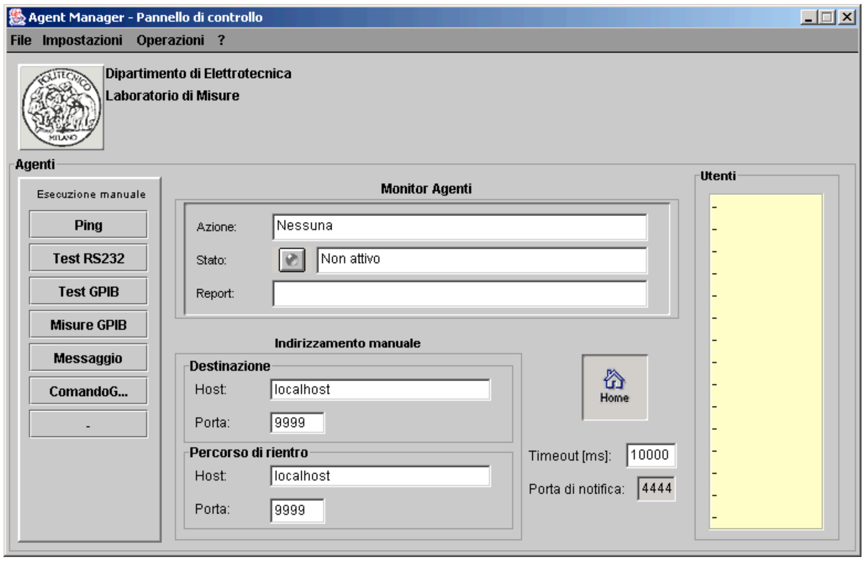


Fig. 4 – An example of application of Mobile Agents in Measurement.



Azione: Nessuna

Stato: Non attivo

Report:

Destinazione

Host: localhost

Porta: 9999

Percorso di rientro

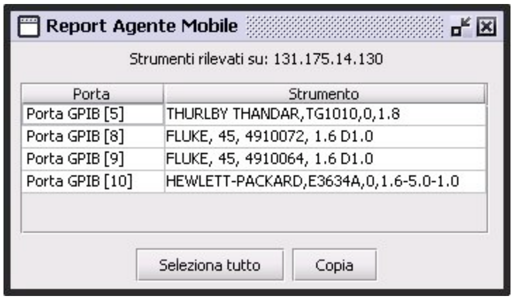
Host: localhost

Porta: 9999

Timeout [ms]: 10000

Porta di notifica: 4444

Fig. 5 – The Control Panel of theAgent Manager for Mobile Agent launching.



Strumenti rilevati su: 131.175.14.130

Porta	Strumento
Porta GPIB [5]	THURLBY THANDAR,TG1010,0,1.8
Porta GPIB [8]	FLUKE, 45, 4910072, 1.6 D1.0
Porta GPIB [9]	FLUKE, 45, 4910064, 1.6 D1.0
Porta GPIB [10]	HEWLETT-PACKARD,E3634A,0,1.6-5.0-1.0

Fig. 6 – An example of an implemented Mobile Agent which is able to recognize the instruments connected by IEEE 488 to the PC.

[4] R.S. Gray. Agent tcl: A Flexible and secure mobile-agent system. In Proc. of the Fourth Annual Tcl/Tk Workshop (TCL'96), July 1996.

[5] G. Karjoth, D. Lange, and M. Oshima. A security model for aglets. *IEEE Internet Computing*, pages 68-77, July-August 1997.

[6] Voyager application. http://www.recursionsw.com/voyager.htm.

[7] Mitsubishi Electric. Concordia: An infrastructure for collaborating mobile agents. In Proc. of the 1st International Workshop on Mobile Agents (MA'97), April 1997.

[8] D. Chess, C.G. Harrison, and A. Kershenbaum. Mobile agents: Are they a good idea? In G. Vigna, editor, Mobile Agents and Security, LNCS 1419. Springer Verlag, 1998.