12th IMEKO TC4 International Symposium
Electrical Measurements and Instrumentation
September 25–27, 2002, Zagreb, Croatia

A MEASUREMENT LABORATORY OVER THE INTERNET BASED ON JINI

A. Furfaro, D. Grimaldi, L. Nigro, F. Pupo
Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria
I-87036 Rende (CS) – Italy
Email: {a.furfaro,grimaldi}@deis.unical.it {l.nigro,f.pupo}@unical.it

Abstract – A novel measurement laboratory, Jini-Lab, based on Internet and Jini services is proposed and experimented. The key features of Jini allowing the development of the distributed measurement laboratory are highlighted. The measurement experiments concern the monitoring of the power network characteristics. Attention is focused on the two services: the power network analyser service and the monitoring service.

Keywords: Distributed measurement systems, virtual laboratories, Internet, Java, Jini

1. INTRODUCTION

This work aims at experimenting with the development of distributed measurement systems [1–6] over the Internet using industry-based software technologies. Major goals are modular design and an exploitation of heterogeneous computing platforms for supporting open and dynamically configurable distributed measurement systems. Openness means, for instance, facilitating connections and access to a system by remote clients at runtime albeit in the presence of necessary authentication and security levels.

In the last years different tools and methodologies have emerged for designing open distributed systems. They include CORBA [7] and Java related efforts like Java RMI [8], JavaSpaces [9], Jini [10-12] and mobile agent systems [13–15].

CORBA aims at facilitating interactions between object-based, interoperable applications programmed into possibly different implementation languages and running on heterogeneous environments.

Java technologies are able to transform a network of heterogeneous machines in a network of homogeneous virtual machines. In particular, Java RMI, similarly to CORBA, allows to develop and publicize remote Java objects, hiding their execution location. Remote object methods can be invoked according to the usual synchronous method call semantics, normally used for local object methods in the context of a single Java Virtual Machine. JavaSpaces uses a Linda-like [16] model to share, coordinate and communicate tasks in a distributed system. Objects (tuples) can be saved on and retrieved from a JavaSpace by different applications, naturally supporting code reuse and download on-demand.

Mobile agents are autonomous computational entities which have the ability to migrate from a computing node to another after having accomplished a given task. They allow to conserve bandwidth in that many network message communications can be replaced by equivalent local communications after the agent has moved to the site of a destination object. Mobile agents can be exploited for building, monitoring and controlling complex distributed measurement systems [2].

2. AN OVERVIEW OF JINI

This work concentrates on the use of Java Jini [10] for the development of Internet-based measurement systems. Jini technology depends on Java RMI and JavaSpaces.

The Jini architecture is founded on the concept of *services*. A service can be associated to a device (e.g., a measurement instrument) or to a software component (e.g., a measurement algorithm or a virtual instrument). Services can dynamically be added to and retrieved from a Jini federation or community (e.g., a distributed measurement laboratory). The Lookup Service (LUS) is a special service which registers and publicizes a set of services which will be queried and accessed by remote *clients* (e.g., an end-user or another service).

The Lookup Services existing in a Jini community can be detected by means of the *discovery protocol*. Discovery can be performed according to a unicast protocol (the IP address of machine on which a LUS executes is known in advance and so the discovery is based on a URL) or through UDP multicast (no IP address for LUS is known and the discovery is accomplished through message multicast or broadcast). LUS discovery is followed by the *join protocol* to permit a service to register at one or more Lookup Services. A service unique identifier (service ID) is associated to each service which helps distinguishing the same service registered at multiple Lookup Services. Services are requested by clients on the basis of their types, attributes or service ID.

A predefined Lookup Service implemented by Sun is *reggie*.

Besides service dynamicity (addition/removal) ensured by the Lookup Services, a fundamental issue in Jini is *service modularity*. Services are accessed exclusively through their interface. The implementation of a service can be changed without affecting its clients.

General service management automatically ensures code mobility (delivering service code –actually a service *proxy*- on the client station after a successfully lookup operation) and normally relies on Java RMI as the underlying communication protocol.

The service proxy on the client can be

(a) fully computational (e.g., in the case of a virtual test simulation)

(b) dependent on a hardware-based remote test execution or
(c) a combination of the cases (a) and (b).

In order to support dynamic downloading of Java code, Jini requires a (possibly minimal) Web server to be running on a designated network node from which classes are retrieved and downloaded on demand through HTTP.

Services attributes can include one or multiple *user interfaces* useful for customizing the service behaviour.

Services are accessed according to a *leasing contract* negotiated with the resource *grantor*. A service is automatically released at the expiration of the leasing period, provided the lease has not been renewed in the meanwhile. Leasing is a Jini strategy which proves useful for releasing services and associated resources e.g. in the presence of system crashes. A service lease can either be exclusive or shared with other clients.

Another key feature of Jini is its *remote event model* which allows registered services to be notified when certain conditions are sensed in a selected service. The following objects can be distinguished: the object which has an interest for an event; the object which generates the event (*generator*); the object which receives the event notification (*listener*). The listener can possibly be a different object from that making the registration.

Registration to an event occurrence is itself a matter regulated by leasing.

Jini supports also the concept of a *transaction*, i.e. a sequence of actions which have a common destiny: either all commit or all their effects are cancelled with the computational status existing before the beginning of the transaction restored. A transaction protocol with a commit in two phases is used to ensure transaction atomicity. The concept of transaction can be exploited for programming complex distributed coordination and synchronization structures regulating the interactions among multiple measurement tests.

3. JINI-LAB

Jini-Lab is a novel distributed measurement laboratory which is based on the Jini infrastructure. The laboratory is remotely accessible to authorized users. A client can lookup for a particular measurement test service. In the case it is available, a lease is negotiated. The client then receives the corresponding proxy and proceeds with the configuration and controlling steps with the aid of a suitable friendly graphical user interface (service front-end).

Measurement instruments are themselves modelled as services which can be looked up as part of a measurement test service. Instrument measurement services can also be accessed individually by a client. Measurement services are normally accessed through an exclusive lease. Measurement data collected by a test service can be made available for reading by multiple clients. A service associated to a measurement instrument has the constraint to execute on the computer to which the corresponding instrument device is physically connected.

Jini-Lab is capable of accommodating both virtual or physical instruments as services according to recognized standards. For instance, to facilitate instrument interchanges, the recommendations of Interchangeable Virtual Instruments Foundation [17] can be adopted.

Three kind of users are distinguished in Jini-Lab: *managers*, *programmers* and *end-users*. They have different competences and responsibilities and can execute different operations.

Managers and end-users access the Lab by means of distinct passwords and separate graphical user interfaces. A manager has an in-depth knowledge of a measurement test and of the involved instruments. He/she is in charge of the setting up and initialising the instruments which make operational a test method.

Programmers are devoted to transforming a measurement task in Jini services. End-users (e.g., students) can utilize a measurement test and get the measured data in a suitable form (tabular or graphical), without having the possibility of doing operations which can cause troubles in the involved instruments.

4. A DISTRIBUTED MEASUREMENT EXAMPLE

A series of measurement experiments were implemented in Jini-Lab. They include remote instrument control (e.g., of a multimeter FLUKE45) by a GPIB/ENET, monitoring current and voltage at a remote load, frequency response of an amplifier, calibration of a DAQ board.

Fig. 1a exemplifies the approach applied to a distributed test method concerned with the monitoring of the power parameters of an electrical network. The measurement test is organized in three services:

(a) the power electrical network analyzer service
(b) the database service
(c) the monitoring service, which implements the measurement algorithm and controls the other two services.

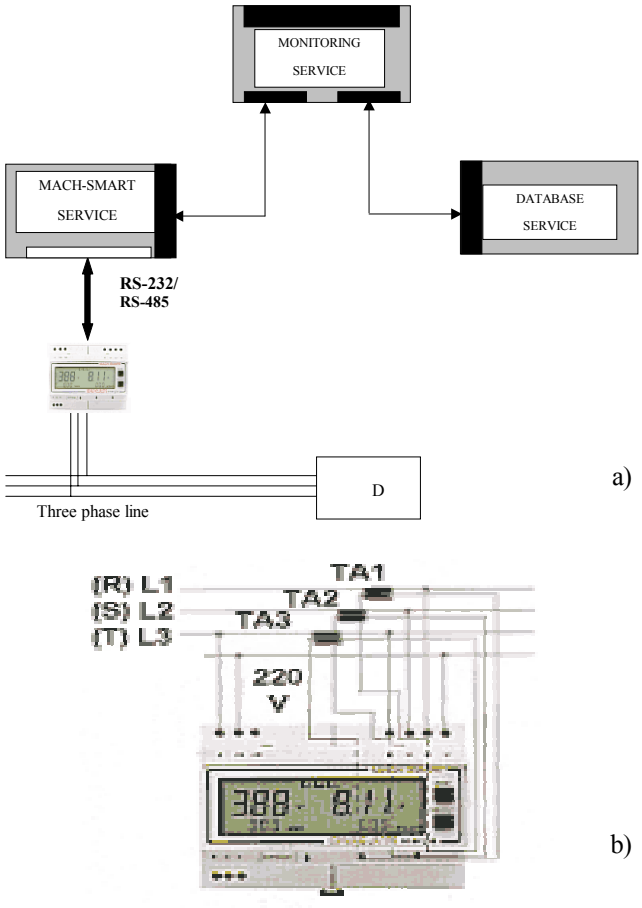

Fig. 1. a) Monitoring schema of a power electrical network
b) Mach-Smart power electrical network analyzer

All the measurement services can remotely be controlled through distinct graphical user interfaces (GUIs) separately exported for managers and end-users. The services are used through an exclusive leasing. However, a client which finds a service temporarily inaccessible can reserve it. Reservation is also managed by leasing. An expired reservation frees the service from sending a corresponding notification event to the client.

The monitoring service periodically reads the electrical parameters from the power network analyzer and sends them to the database for archiving purposes and off-line analysis, and to registered clients interested in watching the data. The database service exploits the Java Database Connectivity (JDBC) [18] package which allows to manipulate a data base, e.g. using Microsoft's ODBC driver.

4.1 Power network analyzer service

Such a service makes it available all the functions of the DUCATI Mach-Smart power network analyzer [19], Fig. 1b, which measures and processes the electrical parameters. The power network analyzer is controlled by a computer through a proprietary protocol or by the Modbus-RTU standard.

The Mach-Smart enters the receiver mode upon the arrival of an STX (ASCII 02H) character. In this mode remains until an ETX (ASCII 03H) character is received. After that correctness of the checksum accompanying the message is checked and the content of the data packed examined. An example of a Host-to-Mach serial communication follows:

<STX>H01*b*<data-packet>*b*<checksum><ETX>

where H indicates the Host is the source, 01 is the address of the instrument, *b* denotes a blank. The data packet of a message sent by Host can specify a required electrical parameter (character ? followed by the electrical parameter code) or a command (character ! followed by the command code). A response from the instrument then carries the value of the electrical parameter. For instance, the following two commands respectively transmitted by Host and by Mach-Smart, require and return the network frequency:

<STX>H01*b*?00b<checksum><ETX>
<STX>M01*b*50.0*b*<checksum><ETX>

The end-user GUI is displayed after a recognized user has been authenticated by its password. Fig. 2 depicts the Mach-Smart GUI for data reading.

The Mach-Smart manager GUI (see Fig. 3) is organized as a tabbed pane. The Join sub-pane (shown in Fig. 3) allows one to associate the power network analyzer service to existing local Jini Lookup Groups or communities or to external Lookup Locators. The serial port sub-pane makes it possible to change the address of the serial port to which the device is connected. The password pane permits entering the password which controls user-level interactions with the service. Other sub-panes are devoted to setting up the TA/TV parameters, to check the service status (free or occupied), to remove or suspend the service.

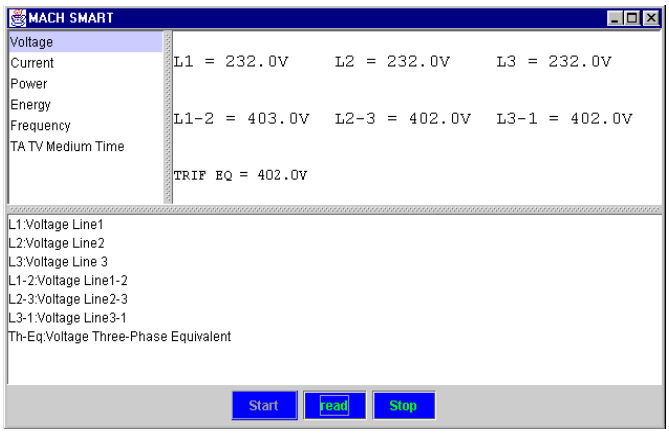

Fig. 2. Mach-Smart end-user GUI for data readings

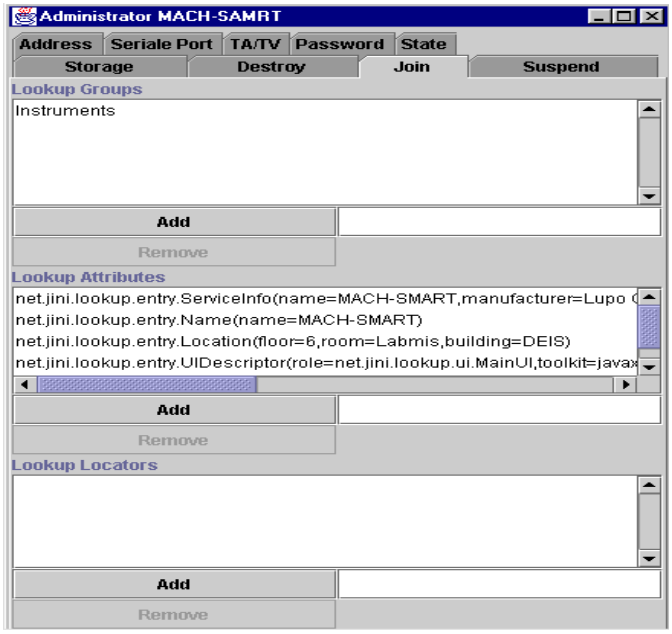

Fig.3. The administration GUI for the Mach-Smart service

4.2 Monitoring service

It first gets the references to the other two services, possibly by reservation if currently they are inaccessible. Then it spawns a thread which implements the measurement test. During the initialisation, the monitoring service creates the database table on which subsequently data readings from the power network analyzer are stored. Fig. 4 portrays the end-user GUI.

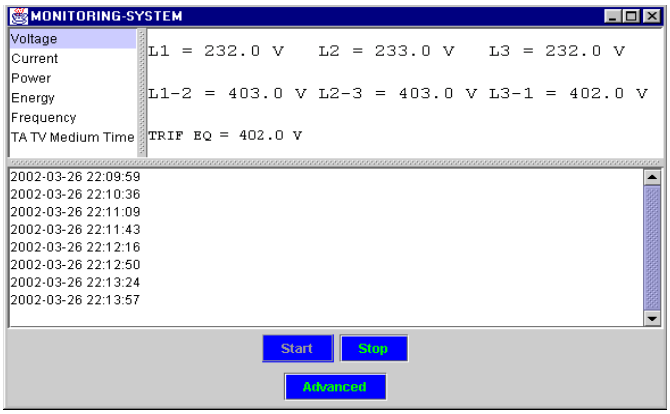

Fig. 4. End-user GUI for the monitoring service

The *start* button in Fig. 4 allows the measurement data to be displayed. The *advanced* button makes it possible to display the data in a tabular or graphical form (Fig. 5).

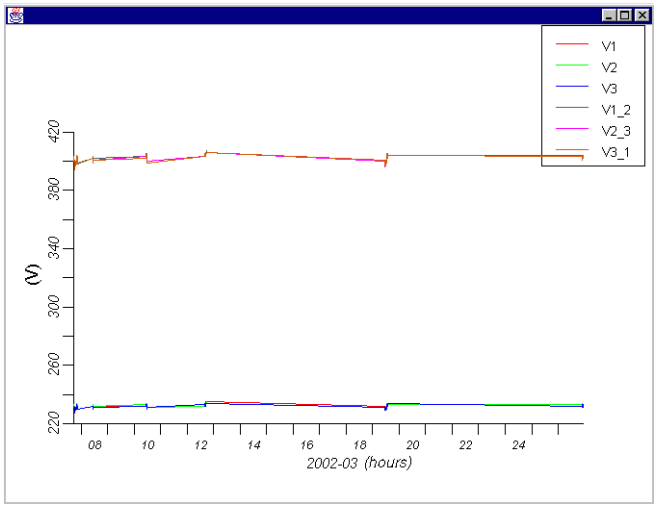

Fig. 5. A graphic displaying the measured voltages

The administrator GUI for the monitoring service is similar to that shown in Fig. 3 for the Mach-Smart service and admits sub-windows for setting the test parameters, i.e. the period of measurements, entering the information about the particular power network analyzer and database services to use, specifying the Lookup Groups where the monitoring service has to be published (join sub-pane), defining the access password.

4.3 Jini Service Finder

To facilitate lookup and management of measurement services in Jini-Lab a Finder application was developed. Finder is able to display graphically the services available in the community Lookup Services. Fig. 6 shows the services in the leim10 Lookup Group and the selection of the Mach-Smart measurement service which activates a pop-up menu. The pop-up menu allows to enter the administration window which the authorized manager can use for management purposes.

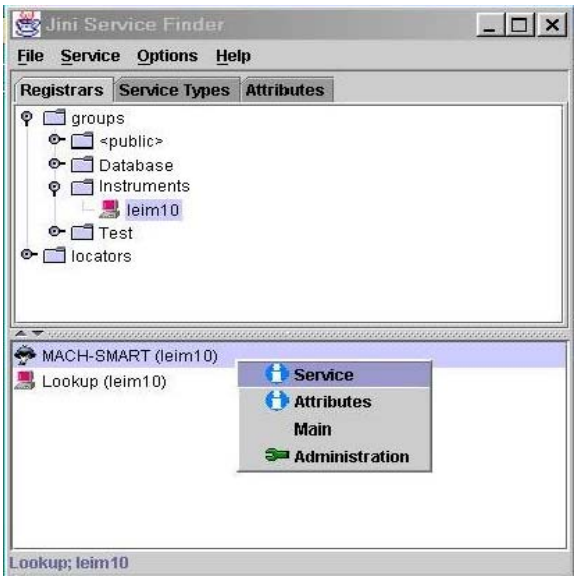

Fig. 6. Detecting Jini-Lab services by Finder

5. IMPLEMENTING JINI-LAB SERVICES

The following outlines some details of current implementation of Jini-Lab.

A measurement service which is in control of a remote physical instrument executes on the measurement station to which the instrument is attached (e.g. through a GPIB bus). As a consequence, a measurement service is decomposed in two objects: the *service proxy*, which is registered to relevant Lookup Services, and the actual *service object* which is remote in the sense of RMI and interacts with the instrument according to the measurement test. Service proxy and service object communicate via RMI protocol.

The user of a measurement service obtains from a Lookup Service a copy of the service proxy and associated attributes, e.g. the graphical user interface which facilitates user interactions.

A service proxy implements the following Java interface:

```
public interface Instrument extends Remote{
    public Object getInstrument( String userID, String password )
        throws RemoteException;
}//Instrument
```

where the *getInstrument()* method returns, would the password be correct, the remote service object on behalf of which the user invokes the measurement commands. The remote object corresponding to the Mach-Smart power network analyzer implements the following interface:

```
public interface MachSmart extends Remote{
    public Object leaseFreeInstrument( long duration )
        throws RemoteException;
    public EventRegistration register( MarshalledObject data,
        RemoteEventListener listener, long durationEvent,
        long durationService) throws RemoteException;
    public double [] getPhaseVoltages( Object key )
        throws RemoteException;
    public double []getLineCurrents( Object key )
        throws RemoteException;
    …//all the other measurement methods
    public double [] getAllValues( Object key )
        throws RemoteException;
}//MachSmart
```

The MachSmart interface defines methods for carrying all the possible measurements on the Mach-Smart power network analyzer. The method *leaseFreeInstrument()* acquires a leasing, if the instrument is free, according to the proposed duration parameter. The method returns a unique object (key) which represents the acquired leasing. The leasing is renewed during the utilization session of the service. If the instrument is found occupied at the invocation time of leaseFreeInstrument(), the method returns null but the user can register so as to be notified by an event when the object becomes free.

The *register()* method specifies the kind of desired service, the listener which will be notified for the lease acceptance, the proposed duration of the registration and the proposed duration of the service.

Each measurement method in MachSmart receives a key object representing a lease and proceeds with the measurement provided the key object is valid (leasing was not expired). At the server side, the AccessLease class is used for lease renewal or cancellation. When a lease expires, a user which still has a non expired registration to the service is notified.

Service administration allows to an authorized user to set up remotely all the operating parameters of a measurement service. To permit remote administration, the service proxy implements also the following interface:

```
public interface Administrable extends Remote{
    public Object getAdmin( String userID, String password )
        throws RemoteExcetion;
}//Administrable
```

The *getAdmin()* method, provided the password is correct, returns the remote object which is responsible of administration. Such a remote object is expected to implement the RemoteAdmin interface which comprises the Jini interfaces [12] JoinAdmin, StorageLocationAdmin and DestroyAdmin which respectively control service participation in the join protocol, specify the location where service persistent data are kept and make it possible to destroy a service by releasing persistent data etc, and the InstrumentAdmin and Remote interfaces. In the case of Mach-Smart, the InstrumentAdmin interface defines all the possible methods for controlling the operation of the power network analyzer (see Fig. 3).

It is worthy of note that the chosen implementation framework of Jini-Lab services centres on interface design. All of this enhances service modularity and reusability because a remote object implementation can be changed without affecting end-users or other service clients.

6. CONCLUSIONS

Jini-Lab is an experimental measurement laboratory supporting the development of distributed measurement systems. The following points summarize key factors of the approach:

- object-oriented implementation of measurement services in Java, which favours code reusability and portability
- immediate deployment of measurement services publicized in Lookup Groups and made accessible, in a controlled way, to manager and end-users
- easy configuration of a measurement service through a graphical user interface which is attached as an attribute to the service
- location transparent distributed control of measurement services
- possibility of accessing measurement services in isolation or in combination according to a test method logic.

Prosecution of the research aims at

- improving Jini-Lab implementation by reducing dependences from RMI
- experimenting with complex distributed measurement systems and associated coordination and synchronization problems
- integrating Jini-Lab with multimedia-based distributed virtual instruments [4]
- making a quantitative comparison with a recent achieved mobile agent architecture [20] which is being applied to the development of a distributed measurement laboratory like Jini-Lab.

ACKNOWLEDGMENTS

The authors wish to thank Corrado Lupo for his contribution to Jini-Lab prototype implementation.

REFERENCES

[1] D. Grimaldi, L. Nigro, F. Pupo (1998). Java based distributed measurement systems. *IEEE Trans. Instrum. Meas.*, **47**(1), pp. 100-103, February.

[2] G. Fortino, D. Grimaldi, L. Nigro (1998). Multicast control of mobile measurement systems. *IEEE Trans. Instrum. Meas.*, **47**(5), pp. 1149-1154, October.

[3] G. Fortino, D. Grimaldi, L. Nigro (1999). An agent-based measurement laboratory over the Internet. *Proc. of IEEE AutoTestCon'99*, 30 Aug.-2 Sept., Texas, USA, pp. 61-66.

[4] G. Fortino, L. Nigro (1999). Development of virtual data acquisition systems based on multimedia internetworking. *Computer Standards and Interfaces*, **21**(5), pp. 429-440.

[5] P. Arpaia, A. Baccigalupi, F. Cennamo, P. Daponte (1998). A measurement laboratory on geographic network for remote test experiments. *Proc. of IEEE IMTC/98*, Min. (USA), pp. 206-209.

[6] D.J. Rawnsley, D.H. Hummels, D.E. Segee (1997). A virtual instrument bus using network programming. *Proc. IEEE IMTC/97*, pp. 694-697.

[7] Object Management Group (1997). CORBA services: Common object services specification version 2. *Technical report, Object Management Group*, June. www.omg.org/corba/.

[8] Sun Microsystems Inc. JavaSoft (1996). Remote Method Invocation Specification. www.javasoft.com/products/jdk/rmi/.

[9] Sun Microsystems Inc. JavaSoft (1998). JavaSpaces. www.javasoft.com/products/javaspaces/

[10] J. Waldo (1998). Jini Architecture Overview. www.javasoft.com/products/jini/

[11] W. K. Edwards (1999). *Core Jini*. Prentice Hall, Java Series.

[12] R. Flenner (2002). *Jini and JavaSpaces application development*. SAMS.

[13] J. Baumann, F. Hohl, K. Rothermel and M. Straßer (1998): Mole – Concepts of a Mobile Agent System, *World Wide Web*, Vol. 1, Nr. 3, pp. 123-137, www.informatik.uni-stuttgart.de/.

[14] R. Gray, D. Kotz, G. Cybenko, D. Rus (2000). Mobile agents: motivations and state-of-the-art systems. Dartmouth College Computer Science Department, *Technical Report TR2000-365*. ftp://ftp.cs.darthmouth.edu/TR/TR2000-365.ps.Z.

[15] ObjectSpace, Inc. (2001). Voyager, www.objectspace.com

[16] N. Carriero, D. Gelernter (1990). *How to write parallel programs*. MIT Press.

[17] Interchangeable Virtual Instruments Foundation, www.ivifoundation.org

[18] Sun Microsystems Inc. JavaSoft (1999). Java Database Connectivity. www.javasoft.com/products/jdbc/

[19] N. Goffredo (1998). *Mach-Smart User's Manual* (in Italian), v. 1.08, Ducati Energia SPA.

[20] A. Furfaro, L. Nigro, F. Pupo (2002). ActorServer: A Java middleware for programming distributed applications over the Internet. *Proc. of International Network Conference (INC2002)*, July 16-18 , University of Plymouth, UK.