IEEE Instrumentation and Measurement
Technology Conference
St. Paul, Minnesota, USA, May 18-21, 1998

Multicast Control of Mobile Measurement Systems

Giancarlo Fortino, Domenico Grimaldi, *Member, IEEE*, Libero Nigro, *Member, IEEE*
Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria, I-87036 Rende (CS) - Italy
E-Mail: fortino@icsi.berkeley.edu, grimaldi@nwdeis1.unical.it, l.nigro@unical.it

Abstract ***- This paper proposes M^{3}A -Multicast Mobile Measurement Architecture- a software architecture for the development of flexible distributed measurement systems. M^{3}A centres at the large level on nowadays Internet technologies such as multicast and mobile computing and at the small level on a Java and Actor-based Framework. M^{3}A allows the building of open, portable and dynamically re-configurable systems. The paper describes the architecture and exemplifies it by some examples.***

Keywords: Open distributed measurement systems, mobile computing, multicast communications, Java, Internet, actors.

I. INTRODUCTION

Recently, several important trends have emerged in the context of the rapidly expanding Internet, including: (i) increased consumption of network bandwidth, (ii) strong consumer pressure for more powerful tools for an efficient access to information, (iii) strong consumer demand for services that are customised according to the user individual needs, (iv) open and dynamically re-configurable solutions.

Multicast and mobile agents technologies [1-3], which represent important areas of current research, possess many of the attributes required to address the above-mentioned trends.

Such technologies boost the achievement of open, inter-operable, modular, extensible and dynamically re-configurable Distributed Measurement Systems (DMSs) [4-8]. They allow to design flexible DMSs where the interaction patterns between the system components can dynamically be changed.

Mobile agents are autonomous software entities that are capable of travelling through the network and performing tasks on a user's behalf. They represent a powerful and promising software concept which can allow to minimise Internet communication overhead by having that the sender first moves to the receiver address space, makes there local requests, compute something and finally comes back to its originating address space with the result.

More in general, mobile computing allows for both code and data to be moved across a network. It makes possible for an already configured test method to migrate to and run in a remote host. Instead of requesting remote communications, it can be a test method itself, emulating the behaviour of a human operator, which moves to a destination site and executes the measurement task by interacting with local available instruments.

Multicast networking capabilities (e.g., IP-multicasting) [2] give the opportunity to create services like the following. Common addresses or group spaces can be defined to which a measurement station can send its measured data or a controlling node can transmit, for instance, start-up, shut-down events and other system-wide control commands. This way, all the monitoring nodes that want to observe the data originated from a measurement station have only to join the established multicast address so as to receive and render, possibly according to different views, the same measured data. The measurement station has no need to know how many and the identity of the nodes interested in the acquired data.

This paper proposes M^{3}A, a software architecture centred on multicast and mobile computing paradigms at programming in-the-large. At the programming in-the-small, a Java and Actor based Framework (JAF) [9-10] is adopted for structuring the test methods. JAF relies on light-weight actors, customisable message-based scheduling, and a modular support of timing constraints [11]. Java is a key for supporting code mobility and for improving software portability.

The remaining of this paper is organised as follows. In the next section the basic concepts of M^{3}A architecture are presented along with a description of the operational phases of a DMS developed according to M^{3}A. Some useful application domains enabled by M^{3}A are clarified. The paper goes on by discussing a prototype implementation of M^{3}A using the Voyager system [12]. After that, the application of M^{3}A is exemplified by some examples. Finally, an indication of the project status and some directions which deserve further research are given.

II. M^{3}A: MULTICAST MOBILE MEASUREMENT ARCHITECTURE

M^{3}A (see also Fig. 1) consists of four basic components: the Configuration and Control Node (CCN), the Measurement Station (MS), the Monitoring Node (MN) and the Code Server (CS).

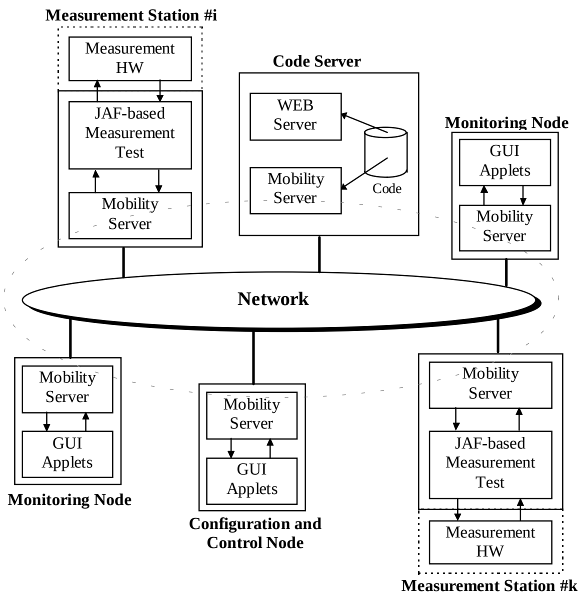


Fig. 1. Components of M^{3}A architecture.

CCN is responsible for interacting with the user to acquire the configuration parameters needed to set up the global and local measurement methods and instruments. It is composed of a Mobility Server sub-component that accomplishes multicast and mobile networking services; one or more Java applets for inserting/setting values, sending multicast messages and launching mobile agents. The Mobility Server is the fundamental block which contributes to the definition of a *virtual distributed measurement environment* over Internet.

MS directly interacts with the Device Under Test (DUT), e.g., by GPIB-controlled instruments and/or specialised hardware (e.g., an I/O board). The Measurement Test sub-component is based on JAF [9-10], a Java and Actor based Framework which is a variant of the Actor model [12]. JAF rests on programmer-defined scheduling and a modular specification of timing constraints [9-11]. The Measurement Test makes local measurements and data analysis.

The structure of a test method is object oriented and consists of three object layers: the *supervisor*, which models the test method itself and the requirements of the device under test; the *logical instruments*, directly operated by the supervisor, where objects closely mirror the features of corresponding physical or virtual instruments; the *device drivers*, where a driver object implements the services of logical instruments in terms of bus operations. The organisation maximises object reusability by having the supervisor which interacts with *abstract* logical instruments implementing a given interface of operations (*services*). Therefore, a logical instrument, e.g., an oscilloscope, can be replaced by another one provided that it implements the same service interface. Moreover, the approach transparently accommodates for the handling of both internal GPIB boards and external IP-addressable GPIB-ENET converters and I/O boards.

Portability of Measurement Test is enhanced by its design which is split in two blocks: the JAF-based actor part and an abstract Bus Server [14] (see Fig. 2). Both the JAF measurement software and the abstract Bus Server are platform independent and therefore totally portable. The Bus Server exports a collection of common bus operations. Multiple implementations of the Bus Server class can exist depending on the particular adopted standard interface board. A specialisation of the Bus Server can be achieved by using a pre-existing driver software and having it interfaced to the JAF-based measurement actors through Java native methods. Obviously, a concrete Bus Server is non-portable. In order to support the concurrent execution of multiple non-conflicting measurement tests, the operations of a Bus Server must be implemented as synchronized methods.

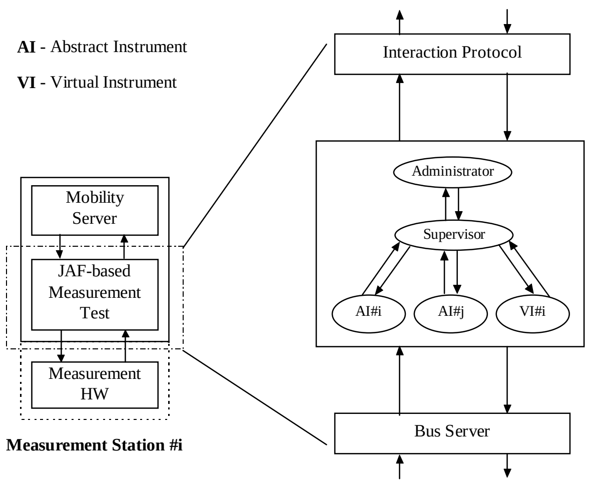


Figure 2. Details of a JAF-based Measurement Test.

Normally the Mobility Server and the JAF-based Measurement Test live in different address spaces of the same platform (e.g., Win95 PC). In order to support communication between them a portable Interaction Protocol abstract class is introduced whose (synchronized) methods are used for communications. An instance of a specialisation of this class is dynamically loaded at configuration time. As one can see from Fig. 2, the Measurement Test component includes also an actor *administrator* which is introduced for simplifying supervisor programming. The administrator is target of external messages, including configuration and monitoring control messages. An external message is created by an Interaction Protocol operation, e.g., requested by a mobile agent, scheduled by the JAF scheduler and eventually processed by the administrator and the supervisor.

The Code Server (CS) contains all the software code of the system (test methods, applets, html pages, etc.). The user and the mobility server can access the code through a Web Server. The user can download html pages and applets, and by support of Mobility Server can upload objects remotely and launch mobile agents.

The Monitoring Node (MN) is capable of displaying measured data on a Applet GUI on a demand basis by using mobile agents or by joining a multicast address.

A. Operational Phases

A measurement system based on M^3A is characterised by phases during which the components interact one to another to accomplish specific tasks. The interaction is regulated by system-wide policies which ensure that the overall operational sequence is safe.

The starting point to make operational an M^3A system is a *bare virtual measurement machine*, where each MS runs an instance of the Mobility Server. Subsequently, the individual MS nodes are uploaded with the relevant and configured measurement actors and brought into operation.

It is worthy of note that each remote test method is only responsible of localised measurement tasks. System-wide co-ordination and policing are ensured by a *global test method* which purposely can rely on mobile agents and multicast control.

The basic phases to bootstrap (1.-3.) and manage (4.-6.) a system are the following.

1. *Configuration*. Every local test method is configured at the CCN site, i.e., the set up parameters of the involved instruments and the test method itself are established. To accomplish this task a *configuration applet* for each measurement test is downloaded.
2. *Dynamic Upload*. All the configured objects (instruments and supervisor) and the JAF runtime support (message scheduler and dispatcher) are remotely uploaded into the target MS. This operation adds contents to the bare virtual measurement machine. The runtime support of JAF goes into a stand-by state, waiting for a start event. In this phase an instance of the classes Interaction Protocol and Bus Server of Fig. 2 are dynamically created and loaded.
3. *Start-up*. The global system operator at the CCN site brings into activity the various JAF subsystems on each MS by sending them either a multicast start message or a round-trip mobile agent which delivers locally the start message.
4. *Control*. When the global system is running, specialised mobile agents can be used to fulfil the needs of a global test method. As an example, a mobile agent equipped of a suitable system itinerary may be launched. The mobile agent can ask each MS for some parameters. According to them it may perform actions (measurement enforcement), move to another site or turn back to its originating node etc.
5. *Monitoring*. Observers at MN sites can acquire and watch the data of particular MSs. An observer typically launches a mobile agent or join multicast addresses to gather information such as waveforms (e.g., of voltage, current), parameters (e.g., coefficients, single measured data), analysed data (e.g., by FFT), historical archived data etc.
6. *Replacement*. A remote test method can be halted and substituted by another one through a new upload and start-up sequence. All of this contributes to dynamic re-configuration and openess of an M^3A system, which can be affected by changes at runtime.

B. Application Domains

The following describes some significant use contexts of the measurement services provided by an M^3A based system.

Measurement Laboratory on Demand

a) *Data acquisition.* This application concerns the possibility for a remote client which has a measurement problem and local equipment to join temporarily a M^3A based laboratory as an MS. The client should be able to provide only the specification of the involved instruments and the adopted standard interface, then he/she should have that the remote laboratory automatically accomplishes the measurement task and sends back its results.

b) *Data analysis.* This application concerns getting some measurement data and analysing them by exploiting the services of a specialised MS, for example provided of non portable virtual instruments. A mobile agent can purposely be introduced which is in charge of the necessary movements and computational steps. The mobile agent eventually returns the analysis results to its originating client.

Integrated DMSs

This application domain consists of the possibility of having multiple, concurrent and independent (e.g., based on non conflicting sets of physical instruments) DMSs which execute on the same physical nodes of a M^{3}A network. Each DMS uses different instances of the software components (e.g., voyager server and JAF-based measurement test).

III. A VOYAGER ENABLED M^{3}A

The Java programming language comes with some library classes and mechanisms [15] especially useful for code mobility (object serialization, reflection, dynamic class loading and instance creation, Internet programming and so forth). However, for prototyping reasons, a first implementation of M^{3}A was achieved on top of an existing Java-centred mobile agent tool in the public domain.

In particular, the ObjectSpace's Voyager system [12] was chosen since it is 100% pure Java, it is simple to use and provides an innovative migration mechanism. It supports both mobile objects and autonomous agents and is featured with persistency services, multicast communications and directory services.

Other mobile Java-enabled technologies include IBM Aglets, General Magic Odissey, and Concordia [16].

A. An Overview of Voyager

The main concept in Voyager is the Virtual Object, the key communication framework and tool to support inter-agent communication and control. Voyager makes available a virtual code compiler (vcc) which takes any existing Java class (source or class file) and modifies it to create a *Virtual Object*, i.e., remote enabled, which mirrors the source class. A Virtual Object is a class concept of which instances can be created and moved around a network. Communication with them is possible in an RPC-like fashion. One such instance can have its own life cycle.

The communication facility in Voyager is very flexible. It allows asynchronous, synchronous and future remote method calls (the latter is an asynchronous call followed, on demand, of a blocking receive of the result). References to remote virtual objects can be freely passed as parameters to methods and can be serialised. As a consequence, a dynamically re-configurable framework can easily be achieved. A virtual object can be moved from server to server and the ability to interact with it kept transparently as it moves. Object *forwarders* are left on the hosts where the object passes which help the communication system to forward messages toward its actual residence.

Besides virtual objects, Voyager offers the mobile Agent class whose extensions, virtualised by vcc, can freely and autonomously move around a network according to an itinerary, i.e., a sequence of hosts to visit and act upon. On its arrival on a given host, the mobile agent can perform some computation, i.e., interact locally with other objects in order to achieve some desired goal.

To support mobile computing, the Voyager system, like other similar systems [16], relies on the *agent server* concept, named also voyager, which must be launched and stay resident on each host taking part in a virtual machine. A voyager server is identified by the tuple <site's IP-address, port number>. The move operation on virtual objects and mobile agents takes the identifier of a destination voyager server.

The Voyager system also provides the *space* notion, i.e., a group of virtual objects which can be affected by multicast communication. Sending a message to a space has the same effect of sending a copy of the message to individual members of the space. When the space concept is achieved on top of IP-unicast Java mechanisms, it only offers a limited form of multicast. True multicast support requires the use of special protocols [17-19] and hardware provisions (router awareness) which can repeat automatically a message directed to a group of remote members.

Finally, is worth mentioning that Java applets are also featured by Voyager. Every time that an applet is downloaded a temporary Voyager server is spawned.

B. Mapping M^{3}A on Voyager

A one-to-one mapping is possible between M^{3}A concepts and Voyager concepts. The Mobility Server coincides with the Voyager Agent Server. It normally deals with a specific Measurement Test distinguished by an assigned port number.

JAF actors and runtime support classes can be made remote enabled by converting them with vcc to Virtual Objects. This way an entire test method along with a copy of the JAF runtime system can be cross-developed and uploaded during the upload phase.

It should be noted that the only class which isn't affected by mobility and which doesn't require virtualisation is the Bus Server class. It must be available on the relevant host for it to be dynamically loaded during the upload phase.

The Interaction Protocol is achieved as a thread-safe virtualised class implementing a shared buffer.

A global test method can be achieved by a mobile agent. In order to support multicast control for system-wide control messages, the various remote test methods can be grouped in a space. However, unicast communication remains always possible with remote test measurement subsystems and mobile agents.

IV. EXPERIMENTAL EXAMPLES

The M^3A concepts were tested by developing two concurrent and independent DMSs integrated in the system portrayed in Fig. 3.

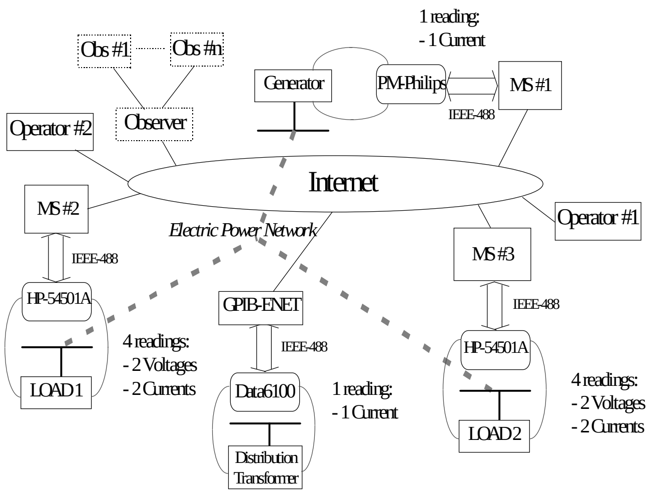


Fig. 3. An experimental M^3A system.

The system is made up of three measurement stations to which several instruments are connected through GPIB interfaces; one operator site for each DMS performing CCN functions; one or more observers at MN sites.

A. A DMS for the Daily Monitoring of Current/Frequency

This DMS is devoted to the daily monitoring of the Current/Frequency at the output of a Distribution Transformer. The measured data are employed to monitor the quality of service. The instrument involved in the measurement is a Data Precision Analyzer 6100 interfaced by the GBIP-ENET converter. The measurement test consists in acquiring one current phase and analysing it by using the FFT algorithm embedded into the instrument. Data display is supported in a monitoring applet.

B. A DMS for Detecting Current Distortion in Power Networks

Such DMS is specialised to the detection of the distortion of the current signal and the identification of the source of distortion in a three-phase and equilibrated electrical circuit constituted by a generator node and two load nodes. The circuit was assembled in the context of a laboratory, but it takes into account a typical real situation that happens in an electrical power network distributed on a geographic area.

Many actual loads are connected to the same power network supplied by a common sinusoidal generator and a load is assumed non-linear whereas all the others are linear. In this case, the current of the non linear load is distorted and owing to the generator and network impedances can cause the voltage to be distorted too. All the others loads are affected by the distorted supply voltage. Hence the current of the linear loads is also distorted. Of course, the source of distortion is only the non-linear load which must be identified among all the loads.

The identification of the source of distortion is easy if the generator and network impedances are null. Indeed, the identity of the non-linear load can be detected by monitoring the harmonic components of the current which are different from those of the supply voltage. On the contrary, the identification problem becomes difficult when the impedances are not null. In this case the identification of the distorting load is based on the following measurement steps:

1. the current signal is monitored at the generator node and, by utilising the FFT algorithm, the harmonic components are detected and the distortion index computed
2. if the distortion index exceeds a fixed value the source of distortion is identified by monitoring the line voltage and current in the metering sections at each load node and by applying the method proposed in [20].

The DMS consists of three test methods (see Fig. 3) located at the generator and at the two loads. They are designed to be independent and periodic. The global measurement test is delegated to a mobile agent which is installed into the generator MS and periodically tests for the presence of distortion.

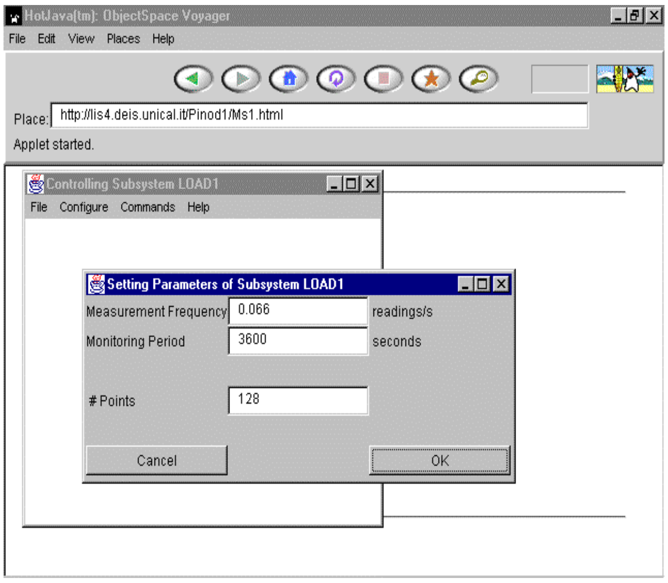
When a distortion is sensed, the mobile agent moves to the loads according to an itinerary and parks on the load MS which is the source of distortion. The mobile agent then sends information to the operator node for a policy decision. Alternatively, it could be the mobile agent itself which is in a position to take some action on the distorting load. After corrections the mobile agent is re-installed at the generator MS and so on.

Load test methods repeatedly read voltage and current, pre-process them according to the algorithm described in [20] and prepare a non linearity coefficient (Cnl) which will allow the mobile test method to detect whether the load is a source of distortion or not. All local test methods use a virtual instrument for computing the FFT of a sampled signal. This software instrument was achieved by interfacing C library code by Java native methods.

Fig. 4 shows the configuration applet of a load Measurement Test under the HotJava 1.1.2 browser. In particular the setting parameters sub-window is displayed.

Figg. 5-7 respectively illustrate the monitoring windows of one phase current FFT, data waveform and Cnl values in the case of a load which is responsible of distortion.

The Monitoring Window is attached to and handled by the Monitoring Applet. The data in the Monitoring Window is periodically updated by the monitoring mobile agent enabled by the Monitoring Applet and located at the LOAD1 site.


HotJava(tm): ObjectSpace Voyager
File Edit View Places Help
Place: http://lis4.deis.unical.it/Pinod1/Ms1.html
Applet started.
Controlling Subsystem LOAD1
File Configure Commands Help
Setting Parameters of Subsystem LOAD1
Cancel
OK

Measurement Frequency 0.066 readings/s
Monitoring Period 3600 seconds
Points 128

Fig. 4. A subsystem load configuration applet.

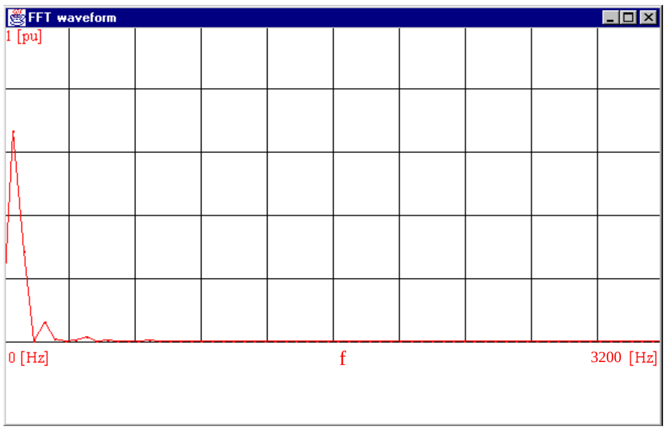


Fig. 5. FFT of one phase current.

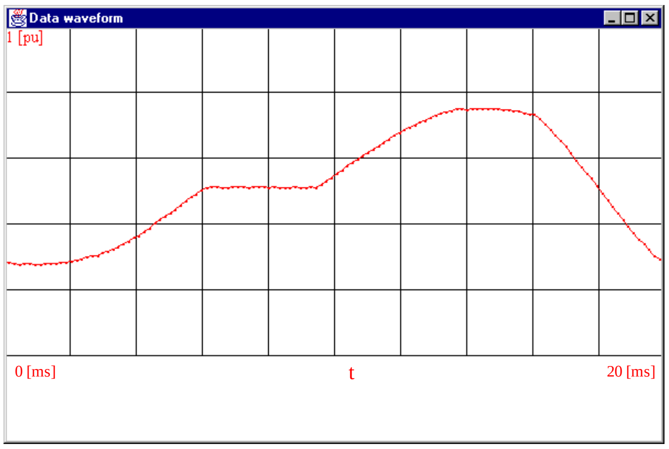
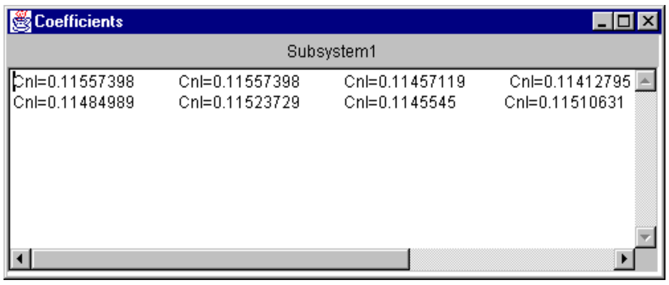


Fig. 6. Data waveform of a current phase.



Subsystem1

```
Cnl=0.11557398   Cnl=0.11557398   Cnl=0.11457119   Cnl=0.11412795
Cnl=0.11484989   Cnl=0.11523729   Cnl=0.1145545    Cnl=0.11510631
```

Fig. 7. Values of the non-linearity coefficient.

V. CONCLUSIONS

Multicast communications, mobile computing and the Java programming language represent timely Internet technologies which have the potential to favour a development of flexible, open, high portable and dynamically re-configurable distributed measurement systems.

This paper proposes the software architecture M^{3}A which is directly based on the above concepts. A prototype implementation of M^{3}A was achieved on top of the ObjectSpace Voyager mobility system [12]. Measurement tests are programmed according to the actor paradigm [13,9-10] and can be remotely configured, uploaded, activated, monitored and supervised by mobile agents.

Prosecution of the research activity aims at

- experimenting with the M^{3}A architecture in the development of complex distributed measurement systems, particularly toward the achievement of the *Laboratory on Demand* concept, i.e., a powerful domain to publicise and support measurement services
- implementing an optimised mobile agents framework in Java directly based on the actor paradigm in order to exploit recent real-time and reliable protocols [17-19]

which have been proposed for multimedia applications [10] and which give better support to multicast communications. All of this should improve the flexibility and the efficiency of the programming in-the-large level.

ACKNOWLEDGMENTS

Work carried out under the financial support of the Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the Project “Methodologies and Tools of High Performance Systems for Distributed Applications.”
The authors wish to thank Giuseppe Pangallo and Francesco Pupo for their contribution to this paper.

REFERENCES

[1] J. Vitek and C. Tschudin (eds): “Mobile Object Systems: Towards the programmable Internet,” *2nd Int. Workshop MOS'96*. Selected Presentations and Invited Papers, Springer-Verlag, *LNCS 1222*, 1997.

[2] S. Deering: “Host extensions for IP multicasting,” *IETF RFC 1112*, August 1989.

[3] V. Kumar: “*Mbone: Interactive multimedia on the Internet*.” New Riders Publishing, 1996.

[4] P. Daponte, D. Grimaldi, L. Nigro, and F. Pupo, “Distributed measurement systems: an object oriented architecture and a case study,” *Computer Standards and Interfaces,* **18**(5), pp. 383-395, June 1997.

[5] G. Fortino, D. Grimaldi and L. Nigro, “Distributed measurement patterns using Java and web tools,” *Proc. of IEEE AutoTestCon97*, pp. 624-628, 1997.

[6] D. Grimaldi, L. Nigro, and F. Pupo, “Java-based distributed measurement systems,” to appear on *IEEE Trans. on Instr. and Meas*., 1998.

[7] P. Arpaia, F. Cennamo, P. Daponte and Savastano M., “A distributed laboratory based on object oriented systems,” *Proc. of IMTC/96*, pp. 27-32, Brussels, June 1996.

[8] P. Arpaia, A. Baccigaluppi, F. Cennamo and P. Daponte, “A distributed measurement laboratory on geographic network,” *Proc. of Imeko TC-4*, pp. 294-297, Budapest, September 1996.

[9] Nigro and F. Pupo, “A modular approach to real-time programming using actors and Java,” *Proc. of 22nd IFAC/IFIP Workshop on Real-Time Programming*, pp. 83-88, 1997.

[10] G. Fortino and L. Nigro, “QoS centred Java and actor based framework for real/virtual teleconferences,” *Proc. of SCS EuroMedia98*, Leicester (UK), Jan. 5-6, pp. 129-133, 1998.

[11] B. Kirk, L. Nigro and F. Pupo, “Using real time constraints for modularisation,” Springer-Verlag, *Lecture Notes in Computer Science* 1204, pp. 236-251, 1997.

[12] ObjectSpace Voyager, http://www.objectspace.com.

[13] G. Agha, “*Actors*: *A model for concurrent computation in distributed systems*,” MIT Press, 1986.

[14] D.J. Rawnsley, D.M. Hummels and B.E. Segee, “A virtual instrument bus using network programming,” *Proc. of IEEE IMTC/97*, vol. **1**, pp. 694-697, 1997.

[15] Sun Microsystems Inc. Java, http://java.sun.com/, http://chatsubo.javasoft.com.

[16] J. Kiniry and D. Zimmerman, “A hands-on look at Java mobile agents,” *IEEE Internet Computing*, pp. 21-30, July-August 1997.

[17] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, “RTP: a Transport Protocol for Real-Time Applications,” *IETF RFC1889*, January 1996.

[18] T. Liao, “Light-weight Reliable Multicast Protocol as an extension to RTP,” http://monet.inria.fr/lrmp.

[19] S. Floyd, V. Jacobson, C. Liu, S. McCanne and L.A. Zhang, “A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing,” *ACM SIGCOMM’95*, pp. 342-356, August 1995.

[20] L. Cristaldi, A. Ferrero: A digital method for the identification of the source of distortion in electric power systems, *IEEE Trans. on Instr. and Meas*., **44**(1), pp. 14-18, 1995.