Development of Distributed Self-adaptative Instrumentation Networks
Using JINI Technology

J-M. Dricot, Ph. De Doncker, M. Dierickx, F. Grenez and H. Bersini

Université Libre de Bruxelles
Service d’Electricité Générale
Elecgen CP 165/51
Avenue Franklin Roosevelt 50
1050 Bruxelles – Belgium.
E-mail: jdricot@ulb.ac.be.

Abstract – This paper describes a client-server architecture for the creation of dynamic and self-adaptative instrumentation over the network, either local or the Internet. The proposed solution allows multiuser, multi-instruments sessions by the means of a new cooperative concept known as Jini. Clients applications take advantage of this system-independent technology by using the Java programming language.

Keywords – Open distributed measurement systems, intelligent sensors, Jini technology, mobile networks

I. INTRODUCTION

To understand why building robust softwares for the network is difficult, it is useful to look at the assumptions a classical programmer relies on, and to submit to that they reveal to be false in the long run: network absolute reliability, zero-latency interactions and topology knowledge in terms of connectivity and instrumentation devices. Classical applications built on this model do not allow a real dynamic deployment of the instrumentation devices since they do not provide enough reliability for wide scale usage of mobile networks and they require the use of proprietary solutions. Changing a sensor by another is thus synonym of driver (re)installation or even of in-depth reconfiguration of experiment software.

A solution is offered by a new cooperative technology known as Jini[1]. Jini runs on top of the Java platform and was first introduced by Sun Microsystems in January 1999, with a host of licensees already on board including Xerox, Hewlett-Packard (Agilent), Cisco and Nokia. Jini enabled devices are able to communicate over the network[4] to provide each other services that can be discovered and assembled dynamically, while the devices and the consumers appear or disappear.

Using Jini one can provide dynamic access to the instrument at the time it is plugged into the network. Each device is thus not just seen as attached to a specific computer but as an independant and auto configurable network device. Client applications can browse for instruments available on the network and use them without any installation of any kind and without any code recompilation.

Since networks are inherently bandwith-limited and can experience major connection failures, Jini provides a quality and reliability guarantee mechanism useful to provide robustness. If an instrumentation device suddenly disappears or seems to experience failure, the whole community is advertised thanks to broadcasted warning messages.

Finally, since Jini is based on the Java technology which provides platform-independant deployment and object-oriented development [6][7], it enables integration with already well-used network-distributed applications like CORBA, LabVIEW or GPIB-ENET[5].

II. THE JAVA AND JINI TECHNOLOGIES

Java started out as an object-oriented langage called Oak, designed as a portable way to write platform-independant and network-oriented code for embedded processors. It provides common programming langage and deployment regardless the operating system by the mean of a computer abstraction layer called the Virtual Machine.

Simply put, the Virtual Machine takes care of reading a Java compiled code – known as bytecode – and interprets it for execution on the local system (Figure 1). Such a Virtual Machine is available for nearly any CPU/System combination, from PC/Windows to Sparc/UNIX and Linux. Embedded systems can also take advantage of Java technology using special Java edition known as MicroEdition. These run on Motorola and Intel processors relative to small and lightweight systems. Researches are also currently undertaken to develop a pure
Java processor, allowing thus to run Java directly over the target hardware, without slowing the execution with the interpretation process. Using these principles, one can develop and compile under a given machine and directly deploy the code over any other host that has a Java Virtual Machine or a Java processor installed.

Java is also backward compatible with old code and it can use any library written in C or FORTRAN. This technique, known as the use of native code allowed us for example to write a Java program to locally control standard GPIB devices using widely available C development libraries.

In some ways, the history of Jini is an extension to the history of Java – it fullfills the vision of a group of consumer-oriented electronic devices interchanging data and capabilities. Jini technology was first released January 25, 1999 by Sun Laboratories and is now available as a free of charge package that can be downloaded from the internet. While Jini implementation still varies from release to release, the specifications are publicly released and frozen. We can thus rely on them to develop a long term solution.

III. THE JINI INSTRUMENTATION NETWORKS

The Jini instrumentation networks we deployed rely on 4 key elements (see fig2):

1. the Lookup is a PC that manages the network, collects information about available services and interconnects dynamically clients and instruments. It also monitors device or network failures and dispatch events to registered clients.
2. the Services which are provided by the instruments and are to be used by the client applications. For example, a voltage measurement service is associated with voltmeter instrument. Advanced devices can provide multiple services, e.g. voltage/current/impedance measure capabilities for common multimeters.
3. the Client is the service consumer. It looks up services it wants to use via the Lookup and registers for events. The client simply specifies to the Lookup what kind of services it wants to use and then gets from the Lookup to give it the necessary information to connect to the matched services.
4. the Leases. Each device on the network is monitored by the Lookup using a leasing mechanism. This ensure flexible apparition or departures conditions and allows the detection of abnormal behavior of instruments – sudden disappearing or software crash.

The Jini model also supposes that the network layer is properly configured. This is easily achieved dynamically by well know techniques such as DHCP (Dynamic Host Configuration Protocol) which takes care of configuring on request a machine appearing on the network. Such a light management software is actually embedded in most network routers and could be also put in a future hardware version of the Lookup service.

Jini-enabled devices and softwares interact using a common scheme to create the local instrumentation system. One Lookup need to be present on the network to orchestrate the transactions. Redundant Lookups can be put to increase reliability or to act as bridges between two networks. Since a Lookup service is a java program, it can be run on top of a standard, commercial computer or it could be embedded into a pure hardware machine dedicated to this role. Hardware Lookup are thus low cost and easy to deploy.

IV. THE JINI MECHANISM

In a first step, the instrumentation device tries to discover a Lookup using a broadcasting technique[3]: it sends packets to all machines and waits for a positive response (see fig2)
to it a portion of code called a *proxy*. The Lookup is the repository of all the proxies coming from detected devices (see fig3).

Any client application on the network can ask the Lookup if a given service is present on the network (for instance a given device located in a given room). If the Lookup holds one or more proxies matching the needs of the client, it sends a copy of the requested proxy to the client (see fig4). This proxy has thus travelled point-to-point from the device to the consumer program without need of knowing each other network parameters, such as IP address or protocol.

The instrument proxy sent to a client can be seen as some kind of *driver* that is dynamically installed on request and that can be removed after use, without requiring human intervention or reconfiguration. In our case, the proxy is used by the client to call back the device. Jini does not lay down a specific protocol at that point and the remote control can be performed using CORBA (distributed objects industry standard), ENET-GPIB or Java’s RMI (Remote Method Invocation). The whole "callback" method is entirely included in the proxy and does not need to be known by the parties (see fig5).

It is important here to note that looking up for a proxy supposes that the client has prior knowledge of the kind of service it wants to use. For example, a voltage monitoring program wants to find a voltmeter or a more complex device offering at least the "measure voltage" ability. This mutual knowledge can be implemented by using special objects called *interfaces*. These interfaces define what the device is able to do in a standard form, but not how it does it. Although each manufacturer has the freedom to develop its own implementation of the service and put its own code in the service, everyone knows how it will look like "from the outside" and how to gain access to the instrument. Interfaces are used to define a common canvas of work and communication.

It is here obvious that enabling Jini in instrumentation technology needs the manufacturers to agree on a standardized set of common interfaces. Currently, no common interfaces are officially set by software manufacturers or standardization organism, but the in-depth analysis of the devices functionalities and the way they interact has led us to propose a first definition which could be used as a basis for further standardization. Since this definition is implemented in the Java programming language, it can be used to describe and develop applications in any object oriented language. CORBA or DCOM systems, largely used to deploy distributed applications over the network can rely on this descriptive and object oriented structure of the virtual instruments and take advantage of the prior knowledge of the attributes of the objects present on the bus.

V. BUILDING SELF-ADAPTIVE INSTRUMENTATION OVER JINI

Actual mobile instrumentation networks are bound to the knowledge of present hardware and the use of proprietary solutions[2]. Using Jini, gaining control to a voltmeter is as easy as plugging a network cable in the device. But Jini offers also solutions to robustness and reliability. Each proxy sent by a service to a Lookup is stamped with a leasing agreement established by the service and the Lookup. This lease can be cancelled properly at any time.
and is to be renewed periodically to ensure the device is still alive on the network. If it cannot be renewed, the Lookup in charge of the device broadcast messages to the client, indicating that devices network topology has changed.

The proxy can also be enhanced to provide more functionalities. Beside offering remote measurement, the proxies we developed hold a set of graphical user interfaces for purpose of displaying devices front panels, installation wizards or even debugging environments. As usual with the proxies, no prior knowledge or installation are required. These softwares are sent directly by the device and are automatically uninstalled after leaving the local area network, thanks the Java cleaning mechanism known as garbage collection: objects present on the machine that are not bound to any particular usage and that are not required anymore by any piece of code are periodically collected and removed from the memory. This mechanism is designed to ensure that there will be no leak of dynamic memory on the local machine and can be seen here as a sort of uninstallation of the code that was downloaded from the service. Many proxies can thus be downloaded and used for a given time, and once the experiment is finished, the programs uninstalls properly, cleaning the client side of the experiment. Because Jini concepts are not limited to the control of devices, Jini-enabled software can also be put on the network to offer assistance and intelligent decision mechanism available for all elements taking part in the measurement process. One could develop mobile agents or intelligents agents available from the local network to perform decisional-level tasks, data collection or to increase reliability via monitoring processes.

Virtual instrumentation softwares (e.g. LabVIEW) integration is possible with special VI’s that connect on the network as clients and provide dynamic presentation of devices available for measurement. These devices can be added or bound to diagram elements just as if they where classical GPIB instrumentation devices with the advantage that no reconfiguration is needed if an instrument is replaced by another with the same capabilities – they just have to present the same interface. We have shown that such an integration requires no in-depth redevelopment of current applications. The use of Jini functionalities is possible by the means of a handling layer written in Java and working with the application. This layer has been designed to manage every aspect relative to Jini and it presents the available instruments to the application. The layer and the application interact using a very minimal set of functions, for example a simple VI library in LabVIEW, demonstrating that it can be viewed as an add-on or plugin, rather that a piece of code that must be compiled in the software.

Finally, Jini technology can be used to improve reliability, scalability and maintenance in large scale experiments or enterprise measurement environments. If one device fails during the time of the experiment, it can be directly replaced by another Jini-enabled device as much as the replacement device is able to provide the same functionality. Heterogenous devices networks, mostly in term of manufacturers, are no more a problem because the devices are seen as functionalities present on the network, not as a particular instance of a given device locally present. Device replacement thus means only to unplug the defective device to plug in a replacement one.

VI. REMOTE INSTRUMENTATION

We have seen that Jini services and clients can act on a local network, such as a classical enterprise network, when they are able to detect the local Lookup using a broadcasting technique. Jini also offers the possibility to bypass the broadcasting-based discovery process to designate the address of a prior known local Lookup. Since the location used in this method must only be a valid TCP/IP address, one could reference a Lookup located anywhere on the Internet. The services using that Lookup will thus be available for the clients connected with that remote Lookup. Many use cases are then possible:

1. control of remote devices that are looked up by a client on a distant location
2. making available local devices from a remote laboratory anywhere on the Internet
3. using a company-centric Lookup to put largely spread clients and services in touch. These clients or services can be thus distributed far from each other but, as they communicate with a central point, they are available as if they were present locally.

This static vision of Jini is in fact a complementary of the dynamic one we have exposed upper. The innovative approach compared to the well-known technique of remote control is the common scheme of development enabled by using Jini-based on a set of well known interfaces. One can thus develop an application that will be globally functional, because everyone knows what kind of objects and functionalities will be available on the remote side and that the application developped for local control of devices uses the same principles in case of distant measurement experimentation.

VII. THE INNOVATIVE APPROACH OF JINI

Many solutions exist for remote control or distributed computing, but Jini offers some enhancements that enable
totally new possibilities for distributed virtual instrumentation.

First of all, Jini distinguishes from the other distributed systems protocol, like CORBA or RPC (Remote Procedure Call), in that it does not only call a function or use a remote object (in term of object-oriented programming) but also transfers a piece of program from the service to the client. This vision leads to the usage of smart proxies that can be used to communicate with the remote instrument but also could make the post-treatment of the data acquisition on the client, for example FFT analysis, graphical display. All these possibilities are contained in the device.

Jini also differs from electronic systems used for remote control thanks to the dynamic deployment of the device. The usage of remote instruments actually suppose the knowledge of their TCP/IP address and require to have the correct driver to be installed on the client machine. Because all the necessary software is embedded in the device itself, a Jini device is far more flexible and deploys faster.

VIII. PRACTICAL JINI DEPLOYMENT EXAMPLE

All of the above explained concepts were used in an experiment taking place at the University of Brussels (ULB). The experiment concerned digital voltmeters used in a hypothetical measurement experimentation in our laboratory. Each device is controlled by a standard computer using GPIB connectors. Since the Jini service is built on the Java platform, we used here the native code usage capability of Java to control the GPIB bus using C libraries on the local machine (see Fig. 6). The proxy of the voltmeter service is able to furnish the voltage. It also provides a graphical interface that can be used to set location of the device, and a small comment helping to fix its role in the experiment. These information are used later to browse the devices on the network and to choose the one we want to connect to.

A second service represents a digital multimeter that is able to measure voltage, current, impedance and frequency. It provides a proxy similar to the one of the first device but also one for current measurement, one for the frequency and one for the impedance. That device offers thus 4 services.

As soon as the devices are plugged in the network, they appear in the client browse list and are categorized on the basis of their functionalities. We can see on Fig. 7 that the second device is available in four categories: one for each service it offers. Note here that it is just a question of software to enable or disable a service that a device offers on the local Jini bus. One can restrict the field of action of an instrument by selecting which interfaces the proxy of the device implements.

The device can now be used in the client application or be asked to transfer on the client-side its graphical configuration interface. Using that interface, one can set up new parameters for the device and, as soon as the new arguments are set, the client are notified by a propagating event that the device has changed its configuration.

Remote events are also used when the device disappears suddenly. When the leasing renewal sequence can no more take place between the service and the Lookup, the Lookup discards the local copy of the proxy it holds and sends an alert event to the clients. Many "crash tests" have been practised using typical security time as small as ten seconds, all showing confident reliability in the monitoring of the presence and the availability of the devices.

Finally, connecting the two devices to the same source
IX. ENABLING JINI IN DEVICES

Jini is built on top of Java to offer code mobility and dynamic interoperability. Allowing a device to join a Jini community requires the device to be able to run Java programs. Such hardware exists in the form of the PicoJAVA processors. They are designed to execute Java bytecode on an embedded electronic device, allowing to integrate Jini technology directly in the device. This technology is already available by major electronic manufacturers and it offers cheap solutions for building Jini-enabled embedded devices. Another solution is to consider the use of the KVM, a small signature Java Virtual Machine that was designed to run over Intel x86 and Motorola processors. These processors are widely used in small devices and provide two major advantages: their low cost and the use of well known and easy to find development environments.

Finally, an easier to deploy solution is the use of standard PC controlling instruments via standard ports such as GPIB, serial or parallel controllers. The computer acts then as a “translation box” that allows to run the software needed to join a Jini community. The remote control instructions received by the service are translated into instrument-specific instructions and sent locally to the device. This configuration and the use of GPIB-ENET allows also to create a centralized set of services that can be used as a bridge between GPIB-ENET networked instruments and the Jini services.

X. CONCLUSIONS

Using Jini technology enables dynamic instrumentation in an easy to implement fashion. It provides open programming model interfacing with industry-standards solutions like CORBA and LabVIEW. Absolute interoperability between the actors of measurements is achieved by abstraction mechanisms which hide vendor-specific implementation requirements. The existing devices can easily become Jini-enabled by putting a small embedded system in the device itself or on GPIB connectors. Helper and intelligent modules can be developed and added “on-the-fly” to help improving instrumentation network capabilities and meeting local engineering needs. Remote instrumentation on distant systems can be achieved using the same scheme of development as for local control of instruments, enabling an easier scheme of development.