Configuration and supervision of advanced distributed data acquisition and processing systems for long pulse experiments using JINI technology

Joaquín González[a], Mariano Ruiz[a], Eduardo Barrera[a,*], Juan Manuel López[a], Guillermo de Arcas[a], Jesús Vega[b]

[a] Grupo de Investigación en Instrumentación y Acústica Aplicada, Universidad Politécnica de Madrid (UPM), Ctra. Valencia Km-7, 28031, Madrid, Spain
[b] Asociación EURATOM/CIEMAT para Fusión, Avda. Complutense 22, 28040, Madrid, Spain

ARTICLE INFO

Keywords:
JINI
Long pulse
Distributed data processing
Remote participation

ABSTRACT

The development of tools for managing the capabilities and functionalities of distributed data acquisition systems is essential in long pulse fusion experiments. The intelligent test and measurement system (ITMS) developed by UPM and CIEMAT is a technology that permits implementation of a scalable data acquisition and processing system based on PXI or CompactPCI hardware. Several applications based on JINI technology have been developed to enable use of this platform for extensive implementation of distributed data acquisition and processing systems. JINI provides a framework for developing service-oriented, distributed applications. The applications are based on the paradigm of a JINI federation that supports mechanisms for publication, discovering, subscription, and links to remote services. The model we implemented in the ITMS platform included services in the system CPU (SCPU) and peripheral CPUs (PCPUs). The resulting system demonstrated the following capabilities: (1) setup of the data acquisition and processing to apply to the signals, (2) information about the evolution of the data acquisition, (3) information about the applied data processing and (4) detection and distribution of the events detected by the ITMS software applications. With this approach, software applications running on the ITMS platform can be understood, from the perspective of their implementation details, as a set of dynamic, accessible, and transparent services. The search for services is performed using the publication and subscription mechanisms of the JINI specification. The configuration and supervision applications were developed using remotely accessible (LAN or WAN) objects. The consequence of this approach is a hardware and software architecture that provides a transparent model of remote configuration and supervision, and thereby a means to simplify the implementation of a distributed data acquisition system with scalable and dynamic local processing capability developed in a fusion environment.

1. Introduction

Due to the special characteristics of fusion devices, the distributed data acquisition systems that are used require mechanisms to carry out both the monitoring and control remotely using systems that have a particularly low computational cost.

The vast amount of hardware devices (controllers, data acquisitions boards, processing boards, etc.) and software components (operating systems, modules, and drivers) that comprise these systems require the use of methods specifically designed to support remote operation, from initial setup to the supervision and monitoring of events.

The intelligent test and measurement system (ITMS) platform offers a technology based on a set of hardware devices and software components that allows for the development of advanced intelligent instruments with data acquisition and fast data processing capabilities that can be reconfigured through software [1,2].

JINI technology is a service-oriented software environment intended to offer a middleware solution which has been designed to develop distributed applications. JINI is defined through a specification that includes a set of components, such as an application program interface API, to develop objects and applications based on the service federation paradigm.

This paper presents an ITMS platform-based application of the JINI technology using a model that permits the implementation of scalable distributed data acquisition and processing systems that can be set up and monitored remotely. The result is a system

in which the software applications can be considered as a set of dynamic and accessible services.

2. ITMS architecture

Fig. 1 shows the hardware architecture of the ITMS platform. It includes the following hardware elements:

- A standard PXI chassis with an embedded CPU named the system CPU (SCPU). The SCPU is used to set up the data acquisition (DAQ) cards, distribute the acquired data according to the system setup, and process the data from selected acquisition channels.
- Several data acquisition cards.
- Several processing cards (peripheral CPUs (PCPUs)) connected to peripheral PXI slots. PCPUs are used to process the data from the acquisitions channels selected during setup. They significantly increase the processing capabilities of standard PXI platforms, which are based on a single CPU [3].

The ITMS software architecture is based on three distributed modules that run in both the SCPU and the PCPUs (Fig. 2): Setup&DAQ, Dynamic Data Processing System (DDPS), and the Event Detection Module (EDM). These modules were developed using Linux kernel modules (Fedora Core 1) based on RTAI (version 3.3), the COMEDI data acquisition driver, and LabVIEW (version 8,2).

The Setup&DAQ module handles the data acquisition and distribution configuration, as well as the acquisition and distribution of data among the processing elements of the system (PCPUs and SCPU). The DDPS module is in charge of running the data processing routines defined by the user. These routines were developed in LabVIEW. The EDM is a client-server application that runs in each CPU (SCPU or PCPU). The client, which usually runs in the PCPUs, detects the events generated by the DDPS module and sends them to the server module as messages through TCP/IP. The server, which always runs in the SCPU, receives the messages from the clients and makes decisions accordingly.

3. JINI technology

JINI technology is a service-oriented architecture that defines a programming model which both exploits and extends Java technology to enable the construction of secure, distributed systems consisting of federations of well-behaved network services and clients [4]. In its conception, a device (defined as any type of system able to process information) that it is connected to the network is publicly announced, and its services

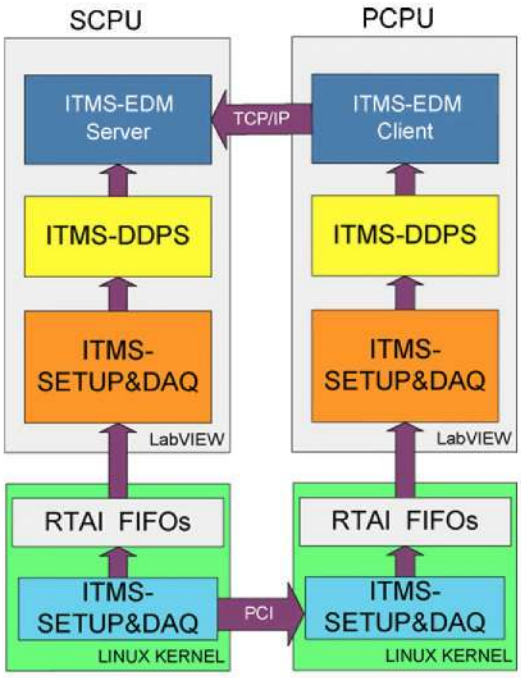

Fig. 2. ITMS standard software architecture.

are placed at the disposal of the other compatible JINI devices [5,6].

JINI's basic concept is based on service, so the network architecture is viewed as a service federation. JINI is defined as the specification for a set of components that constructs a middleware that permits the development of services encompassing all operational needs for subscribing and publishing. It also includes a set of standard services already implemented and ready to be used for deployment [7].

3.1. JINI system components

The following elements are present in a JINI system: services, clients, lookup services and class servers (Fig. 3) [8].

- Services are software mechanisms that make their resources available to the system. A service can provide access to a hardware device or to a software component.

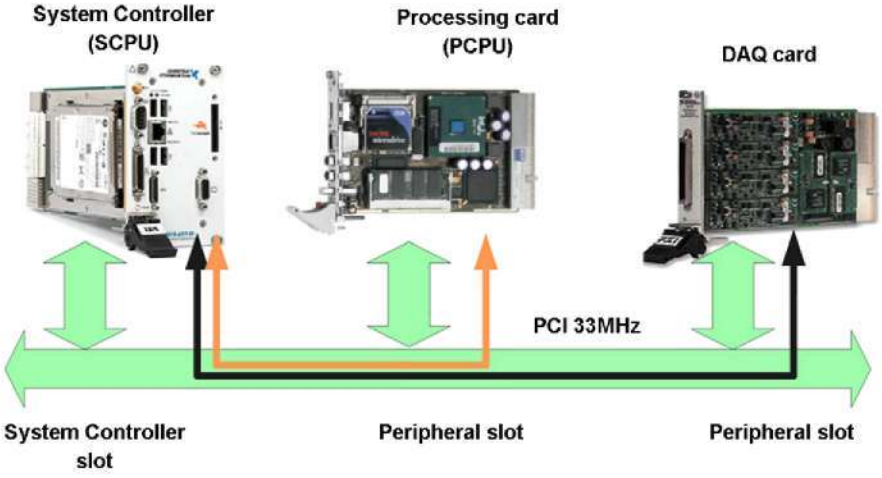

Fig. 1. ITMS standard hardware architecture.

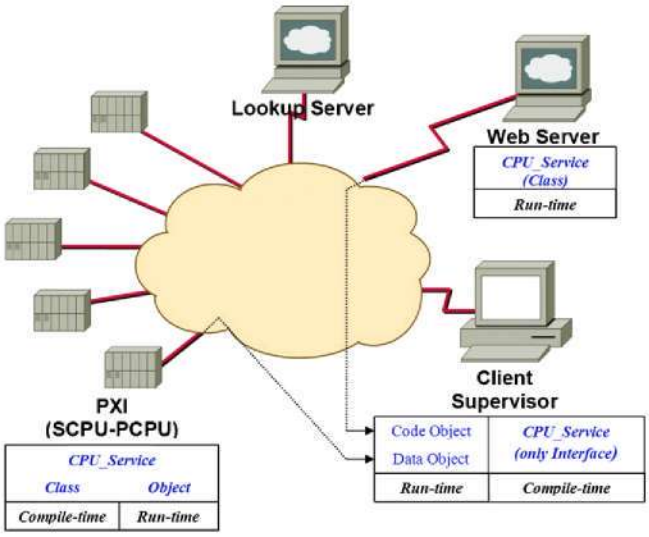

Fig. 3. JINI system components.

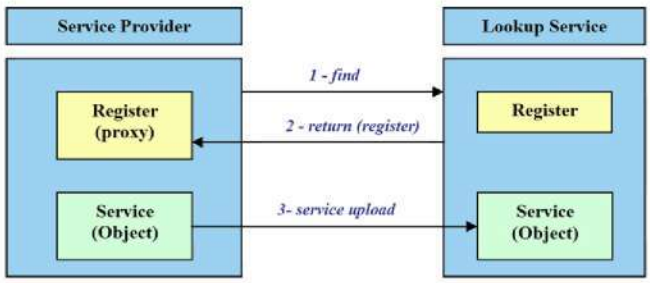

Fig. 4. Publication service process.

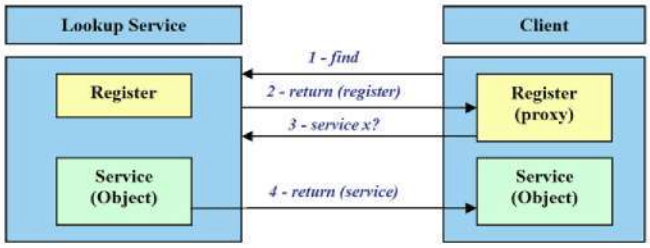

Fig. 5. Subscription to service process.

- Clients are systems that need to use services to perform other tasks. They use mechanisms to locate these services and to subscribe to them, enabling them to complete their objectives by integrating the service into their own business logic.
- The lookup service is a special service that permits clients to locate services. It allows the services to be published and, therefore, offers clients the ability to locate them and negotiate their subscriptions to them. Thus, this service is the key to the system's flexibility, particularly in situations where the network's topology constantly changes.
- Class servers play a key role in the exchange of objects that constitute JINI services. The Java object that implements the JINI service is created by the application server in its Java virtual machine. When a client receives the service, what it really gets is a "snapshot" of the object's state through the mechanism of Java serialization. This snapshot includes only the data value of the object, not the code. So, where is the code of the object? In other models of distributed programming the client must know the implementation of the class, however, in JINI technology, the client simply needs to know the interface of the class, because the implementation of the class will get run-time thanks to the server class. This circumstance offers a tremendous flexibility to update the code of a class because it allows clients to obtain the latest version of the code of the class at the time of subscribing to a service. The information on the location of the class server is part of the serialized snapshot and it is found in the property java.rmi.server.codebase. Once the client has received the snapshot and the class, it reconstructs the object and runs it in its own Java virtual machine.

3.2. Subscription and publication mechanisms

Fig. 4 shows the procedure to publish a service. First, one or more lookup services are located by using one of two possible methods: if the server knows the location of the lookup service it sends a subscription request in unicast mode using TCP, if not, a request is sent using UDP packets in multicast mode. At the other end of the communication channel, the lookup service maintains a passive server ready to accept input connections on a predefined port. When it receives a request through that port it sends an object named "register" that will act as a proxy of the lookup service. From that moment on, any request that the server needs to make to the lookup service will go through the register object. The object includes a set of methods specifically developed to support this operation. Finally, using the object's methods the service provider publishes the service through the lookup service.

Subscription is the procedure that allows a client to obtain a service. Its mechanism of operation is shown in Fig. 5, it begins the process of locating the service by sending a request in the appropriate mode (unicast or multicast).

In both cases, the lookup service returns a register object that will act as its proxy, which provides the methods needed by the client to define the request by identifying the service being requested and its grant, if it is available. Finally, the client obtains the object that implements the requested service if this is available in the list of services maintained by the lookup service.

4. Implementation of the ITMS system setup and monitoring with JINI

A set of services developed with the JINI technology over the ITMS platform was deployed for use in the design and evaluation of a model that allows the extensive implementation of scalable data acquisition and processing systems that can be setup and monitored remotely.

The solution is based on a JINI service, named *CPU_Service*, that runs in each system's CPU and that is published for use by remote terminals (clients). This service can be used to perform the following tasks (Fig. 6):

- Obtaining the hardware configuration of a PXI system. This basically involves determining the hardware elements that form part of a system, such as the SCPU, PCPUs, and DAQ cards, and all of the setup information for each of these elements: IP addresses, acquisition channels, slots, etc.
- Transmitting processing algorithms to the CPUs in order to process data with the DDPS module.
- Receiving messages about the temporal evolution of the acquisition and processing mechanisms.
- Establishing a mechanism for subscription and notification of the events that can be detected during data acquisition and processing.

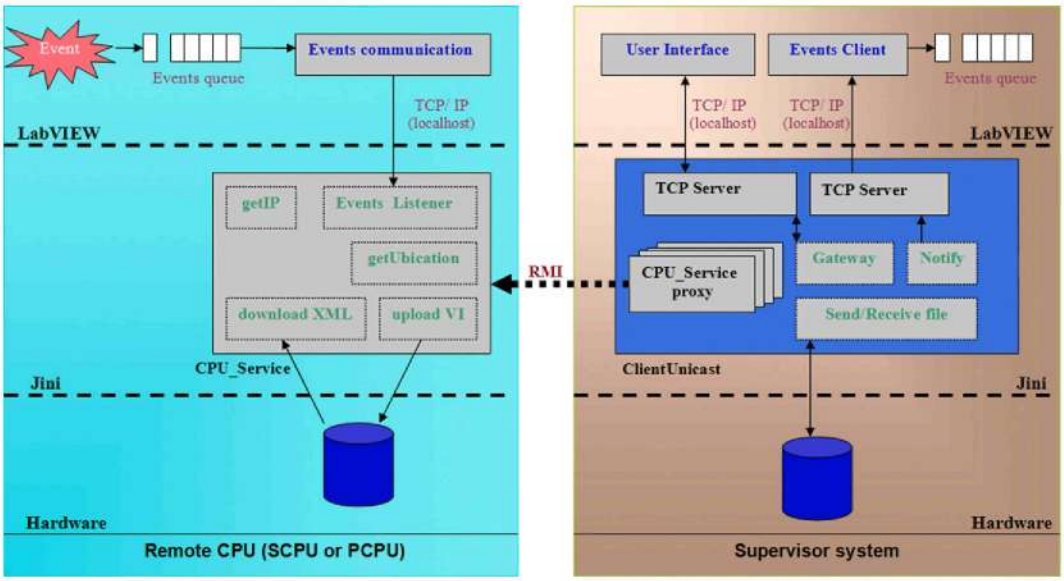

Fig. 6. JINI–ITMS services deployment.

A dual-layer software architecture that includes the existing ITMS software modules was designed to fulfill the requirements. The first layer deals with communication with the standard ITMS software modules. In order to provide a mechanism for the user to access remote objects, a user interface based on LabVIEW was developed exclusively on the client side to provide remote communication with the lower layers of remote objects. This module is also in the first layer.

The second layer is based on JINI technology and was developed in Java. It provides a service federation system that allows publication and remote access of the objects that make up the services. Considering special characteristics derived from the use of Java in software applications, and in order to simplify their maintenance as much as possible, a client-server communication based on local TCP/IP messages was used to link the JINI and LabVIEW modules. This second layer consists of two JINI applications.

The first application, named *CPU_Service*, runs in each Java virtual machine (JVM) located on the SCPU and the PCPU. This application is in charge of publishing the available resources, and their functionality, in the lookup services available in the network. The *getIP* and *getUbication* methods are responsible for obtaining the IP address and the name of the machine that runs the service for identification. The download XML method is designed to obtain the current configuration document and make it available to clients upon request. The *upload VI* method allows receiving processing units that will be run by ITMS platform. And finally the *Events Listener* method that is responsible for subscribing clients who wish to listen to the events from the upper layer and generate the necessary notifications so the subscribed clients will receive them.

On the client runs the application named *ClientUnicast*. Its task is to subscribe to all *CPU_Service*s published and to register as a listener of remote events, to put these events into the events queue, and to play the actions related to user interface messages. The *gateway* module provides a tunnel of communication between LabVIEW and the Java application to send and receive operations and results from the user interface. The *Notify* module transmits the events received to the LabVIEW application which monitors and stores them in a queue of events to be processed according to the requirements. The module *Send/Receive file* is responsible for transmitting and receiving configuration and processing files.

5. Evaluation

The configuration and remote monitoring model proposed is not developed under restrictions on real-time operation. This is why it has been possible to work with JAVA technology and conventional Ethernet networks. Therefore the quality of service (QoS) will depend on the state of the operating systems involved and on the network load. It can be done this way since the monitoring tasks are not critical in this system.

When evaluating the implemented solution, the parameter of greatest interest is the latency time of the notification of JINI events between the *CPU_Service* and the system supervisor (client).

In order to evaluate this latency, an experiment was developed consisting of the detection of a change in the signal spectrum of a channel acquired and processed by one SCPU, and required communicating this detection event to one of the clients who was subscribed to the *CPU_Service* of that CPU.

The signal was digitized using a data acquisition card allocated in a PXI crate managed by the SCPU. The signal data were processed in LabVIEW to obtain the signal's power spectrum and to detect the instant of time in which a frequency change was produced. Immediately after the change was detected, a JINI event was produced and transmitted to the *supervisor system* (client) through the JINI architecture.

The obtained results show that the mean response time (latency) of the architecture is about 36 ms. This time includes: signal acquisition, signal processing in the SCPU to estimate frequency, event generation by *CPU_Service*, event transmission through the JINI architecture, and reception of the event by the client (Fig. 7).

6. Discussion

The architecture developed here is based on the service federation paradigm offered by JINI. All of the CPUs that constitute the ITMS platform make their availability public through the network lookup services. The *supervisor system* (client) needs only to know the object interface, while JINI manages the rest of the processes in a form that is completely transparent for both the user and the programmer.

In addition, the services functionalities are easily extensible, and their deployment in the network is extremely simple, thanks to

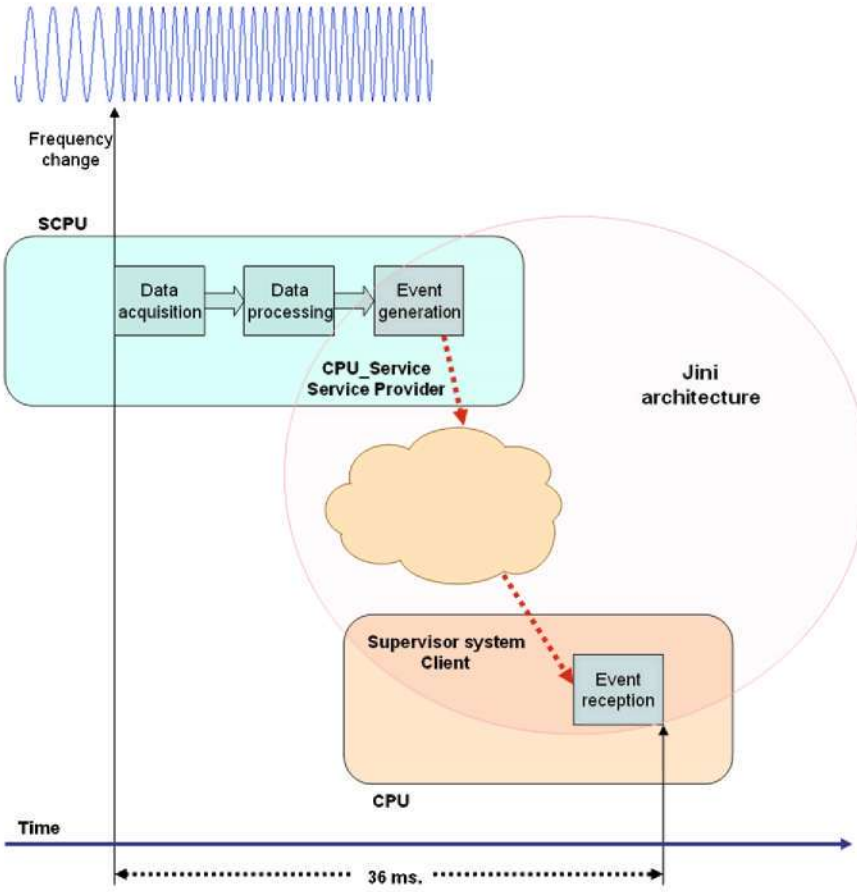

Fig. 7. Architecture response to events.

object serialization and the existence of the HTTP servers which dynamically provide the last version of each service.

The system bears the activation and deactivation of hardware resources in a standard form, allowing the experiment fusion platform to be reconfigured at any time.

This architecture provides to the ITMS platform a methodology to implement data acquisition and processing systems devoted to long pulse and steady state experiments. The ITMS systems can be configured and monitored remotely. This point is extremely important to know the temporal evolution of the experiment by means of user-defined event communication.

Acknowledgement

This work was partially funded by the Spanish Ministry of Science and Technology under the Project DPI 2006-06624.

References

[1] E. Barrera, M. Ruiz, S. López, D. Machón, J. Vega, PXI-based architecture for real time data acquisition and distributed dynamic data processing, IEEE TNS 53 (3) (2006) 923–926.
[2] M. Ruiz, J.M. López, G. de Arcas, E. Barrera, R. Melendez, J. Vega, Data reduction in the ITMS system through a data acquisition model with self-adaptive sampling rate, Fusion Eng. Des. 83 (2008) 358–362.
[3] M. Ruiz, E. Barrera, S. López, D. Machón, J. Vega, E. Sánchez, Distributed real time data processing architecture for the TJ-II data acquisition system, Rev. Sci. Instrum. 75 (10) (2004) 4261–4264.
[4] http://www.jini.org/wiki/what_is_Jini.
[5] http://www.sun.com/software/jini/index.xml.
[6] http://www.jini.org/wiki/Main_Page.
[7] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann and B.J. Krämer, Service-Oriented Computing. A Research Roadmap, Dagstuhl Seminar Proceedings (2006). http://drops.dagstuhl.de/opus/volltexte/2006/524.
[8] Jan Newmarch, "A Programmer's Guide to Jini Technology" Apress 2000.