# A Java based environment to control and monitor distributed processing systems

I.C. Legrand

*DESY - Institut für Hochenergiephysik, Zeuthen, Germany*

Distributed processing systems are considered to solve the challenging requirements of trigger and data acquisition systems for future HEP experiments. The aim of this work is to present a software environment to control and monitor large scale parallel processing systems based on a distributed client-server approach developed in Java. One server task may control several processing nodes, switching elements or controllers for different subsystems. Servers are designed as multi-thread applications for efficient communications with others objects. Servers communicate between themselves by using Remote Methods Invocation (RMI) in a peer-to-peer mechanism. This distributed server layer has to provide a dynamic and transparent access from any client to all the resources in the system. The graphical user interface programs, which are platform independent, may be transferred to any client via the http protocol. In this scheme the control and monitor tasks are distributed among servers and the network controls the flow of information among servers and clients providing a flexible mechanism for monitoring and controlling large heterogeneous distributed systems.

*Key words:* Java, RMI; Distributed systems; Data Acquisition

## 1 Introduction

Current and future experiments in HEP require very large scale distributed processing systems for data acquisition, triggering and on-line data processing. High bandwidth, low latency data-switching networks are necessary to cope with the large amount of data which has to be processed in such systems with hard real-time constraints. For this reason the control and monitoring tasks for such systems should be decoupled from the real-time data processing parts. In this framework the monitoring of the computing nodes, the connecting network and the applications becomes of first importance to ensure safe running and guarantee the expected level of performance.

## 2 Java as a distributed processing environment

Java [1], is platform independent, object-oriented language well known as a tool to create Web applications. But Java goes well beyond this domain by providing a powerful programming environment for complex applications. Built in multi-threaded capabilities allows many concurrent activities inside one program with major benefits for interactive and real-time behavior. Distributed systems require communication between programs running in different hosts. For the basic communication mechanism, Java has extensive support through a socket package which is easy to use and flexible for general purpose interprocess communication. However, sockets require the partners to engage in a specific protocol which is application dependent. Usually such protocol increase the program complexity, are not easily updated and can be error-prone. An alternative to sockets is Remote Procedure Call (RPC), which abstracts the communication to the level of a procedure call. RPC, however does not translate well into a distributed object environment. In order to match the semantics of object invocation, distributed object systems require RMI. The RMI package [2], implemented in Java, consists of three layers: the stub/skeleton layer, the remote reference layer and the transport layer. To accomplish transmission of objects from one address space to another the technique of Object Serialization [3] is used. Another technique, called dynamic stub loading, is used to support client-side stubs which implement the same set of remote interfaces as a remote object itself.

## 3 System architecture

In the HEP environment the system architecture includes a so-called embedded system layer responsible for the detector specific sub-components. Quite simple local server programs may be used to control and monitor all these detector and sub-system specific parts which may have very different characteristics and real-time requirements. These local servers are connected to a layer of distributed servers. The aim of this distributed server system is to provide a transparent and dynamic access from all the clients to any resources in the system. On the other side the same distributed server system should provide a mechanism for error detection and logging of different types of information into specific files or data bases. A schematic view of the main components and how they interact is shown in Figure 1.

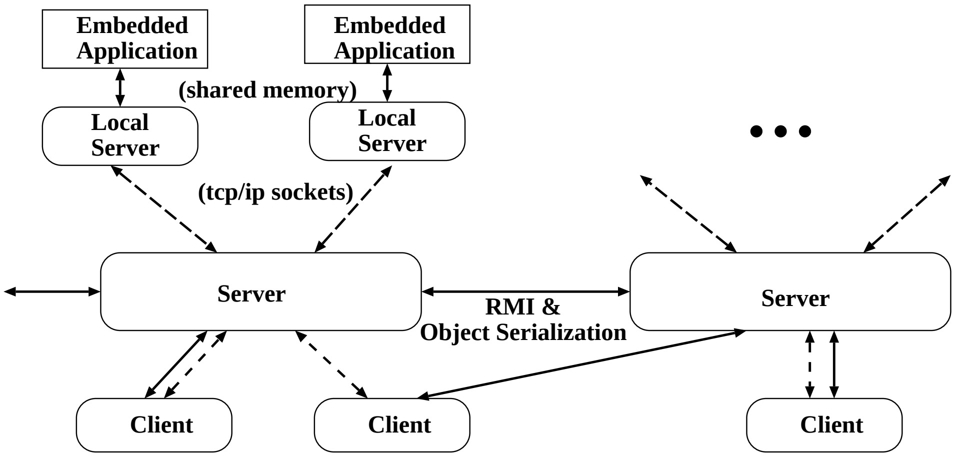


Fig. 1. A schematic view of the distributed server system for controlling and monitoring heterogeneous embedded systems

### *3.1 The local server*

Each such local server controls and monitors a specific part and connects (via tcp/ip sockets, shared memory or native methods) to a distributed server layer which makes all the resources available to any client. A local server controls and monitors a specific task through shared memory (or other communication channels) with the application program. As such local servers may also run in embedded controllers and under small real-time kernels, they should be small C programs which perform the interface between the distributed server and different local applications.

### *3.2 The distributed server*

The distributed server layer consists of more identical Java tasks running on different hosts which collect information from different sub-systems and provide to any client a transparent access to the entire system. Servers continuously update the information from their specific sub-components through independent threads. For each new local server or client a communication thread is dynamically created to provide a better interactive behavior. Local server tasks can create or join a specific service. A service is a logical entity which can be distributed among several server tasks. Each server keeps a table of all the active servers and the type of objects for which they are able to provide information. A new distributed server task is trying to connect to an other server (from a list of possible locations) and gets all the information about the running servers and services.

Servers communicate between themselves through RMI by passing an Event object which is used to inform servers when a server, service, client or local server appears/disappears in the system. This is done through the RMI registry interface which provide a bootstrap service for retrieving and registering objects. Clients have access to all the services in the system and could subscribe/unsubscribe to receive updated information for specific sub-systems with different frequencies. The real-time information from each sub-system is encapsulated into a Java object which is serialized and continuously send to the clients as long as they are register to receive it. A client request for an object which is updated by an other server is forwarded to the appropriate server task for an efficient use of the network resources.

### *3.3 The Client*

A client can connect to any server, through tcp/ip sockets, allowing the user to monitor different parameters in the entire system and to set the configuration for any component. The Graphical User Interface (GUI) is also done in Java, allowing the user to down-load the code, which is platform independent, via the http protocol. The GUIs are also design as multi thread applications. Clients can subscribe to receive detailed updated information for different sub-components. Such a request is forwarded to the appropriate server task which will continuously update the GUI frame of the client. The GUI implements a RMI interfaces which allows the server to update it by sending an serialized Java object. The detailed information for sub-components is transfered in a hash table. The key is used to label parameters. Known key names will trigger special frames for monitoring the corresponding objects. Unknown keys will generate simple 2D history plots for the corresponding parameters. In this way, the monitoring frames are dynamical constructed once such an "information object" is received from the server.

Dedicated client like programs (without a GUI) may be used as plug-in applications for logging data into specific data bases or for run-time optimization between different services which are correlated.

## 4 Summary

The Java programming environment together with RMI and Object Serialization packages provide the tools for developing a distributed, object-oriented system to control and monitor large data acquisition systems. The distributed server layer was implemented using RMI and Object Serialization. Using the same distributed object model in Java, the clients can transparently access any resources. This scheme provides a robust and flexible mechanism to control and monitor large distributed systems.

## References

[1] Java, Sun Microsystems, http://java.sun.com

[2] RMI on JDK, Sun Microsystems, http://chatsubo.javasoft.com/current/rmi

[3] Object serialization, Sun Microsystems, http://chatsubo.jasoft.com/serial