A Control System Based on WWW Technologies

B. Jeram, M. Plesko, R. Sabjan, M. Smolej, G. Tkacik
J. Stefan Institute, P.O. Box 3000, 1001 Ljubljana, Slovenia
e-mail: bogdan.jeram@ijs.si

Abstract

Once used only in state-of-the-art accelerator and large experimental physics control systems, distributed computing has become part of our everyday life. In recent years we have witnessed an explosion of WWW applications and development tools. Therefore it makes sense to fully exploit the existing WWW-technologies provided conveniently by the computer industry as off-the-shelf products.

The control system of ANKA, a 2.5 GeV synchrotron radiation light source being built in Karlsruhe, Germany, uses WWW for both the graphical user interface and data transmission. All operator control is performed through a Web-browser with Java applets. Documentation and help are obtained as conventional html text, including JPEG images, Postscript documents, etc. An authentication system allows applets, which can only read data from the control system, to run on any Internet host in the world. Access to databases is possible through application/data servers in a three-tier architecture. Communication between applets and control system data servers is done through CORBA, RMI or bare sockets, if speed is the primary concern.

This paper presents a running prototype, which includes applets ranging from those responsible for control of single elements such as power supplies to those that perform complex actions such as monitoring and logging. Extensive use of Java features such as multithreading, object serialization, networking etc. is employed to access fully asynchronous and distributed objects. Results of the prototype include an evaluation of several Java developing tools for PCs (J++, Visual Cafe and SuperCede) and benchmarks which clearly indicate that the interpreted nature of Java is not a problem, as computing power is not such an important issue in accelerator control.

1 Introduction

We want to make a control system that will be as homogeneous as possible from the operator point of view. The control system is designed to use existing intranet/internet infrastructure and web technologies such as HTML/HTTP, web browsers, web servers and ObjectWeb with Java and CORBA/IIOP. This decision was made because nowadays a large proportion of people are familiar with web browsers and because the WWW standards provide equal user interface to all information regardless of its type.

Every operator's interaction with control system will go through web browser:

- control GUI (Java / CORBA)
- logbook forms (Java / JavaScript / CGI)
- help, documentation (HTML)
- notification (e-mail)

The core of the control system is based on Java – the de-facto Internet programming language – and distributed objects implemented according to CORBA specification - the combination known as ObjectWeb [1].

The architecture of CS is based on the Standard model. Essentially the same thing is called *a three-tier architecture* in the world of databases:

1. visualization layer consisting of control GUI
2. process control layer consisting of accelerator objects
3. fieldbus layer consisting of devices and databases tier, respectively

In the case of ANKA control system the first tier (where clients run) can be any platform computer platform (Windows 95/NT, UNIX, etc.) attached to the Internet/Intranet capable of running Java. Consequently, the decision about the platform can be postponed to the last moment. Servers on the second layer are running on PCs under Windows NT 4.0 operating system. For the field bus LonWorks [2] from Echelon is used.

2 Client: Java applets running within a web browser

In order to be platform independent clients are applications or applets written in Java. The following description which refers to applets is also valid for Java application. We have chosen Java because it is a modern object oriented programming language, it has well defined data types and API (Application Programming Interface), it allows easy use of graphic widgets, threads and other system tools without having to know the specifics of a given platform. Unfortunately, Java is also an interpreted language, so it is a little slower than compiled languages like C++, but we found out that by using JIT (Just-In-Time) compilers it is fast enough for our needs. Nonetheless, if the need for faster execution arises, several native code compilers can be obtained from the market (see table 3). On the other hand, the Java Virtual machine (JVM) consumes a lot of resources: CPU and memory. Hence only a limited number of clients can run simultaneously on one computer, which has to be taken into account when designing the control system.

We mainly benefit from the following Java characteristics:

- client applets can be downloaded from a web server and run in any web browser on any computer worldwide
- we do not need dedicated computers for running control clients
- it is not necessary to install clients on computers from

which we plan to have control; all we need is a computer attached to Internet/Intranet running a web browser of any sort or even only an appletviewer.

For this reason we need one or more dedicated computers to run the web server and from where all clients, documentation and help files in HTML format can be downloaded. We can put Java client code in JAR (Java ARchive) files to make download faster. In this way we circumvent problems with installing and removing different versions of clients. New clients can be added on the fly to the control system. However, clients can also run as a JAVA stand-alone application, so even a web browser is not a necessity, if the Java virtual machine is avilable.

3 Communication: CORBA/IIOP

There are several possible ways of communicating between distributed programs including RPC and sockets. If we try to follow the lead of distributed object paradigm, we have several options to choose from:

- CORBA(Component Object Request Broker Architecture) [3]
- Microsoft's DCOM (Distributed Component Object Model) [4]
- Sun's RMI (Remote Method Invocation) [5]
- Objectspace's Voyager remote agent technologies [6]

DCOM exists only on Windows platform (Windows NT), Sun's RMI is a part of standard Java and allows only Java to Java communication and Voyager is still in development. Apart from those reasons we also took into account the platform and language independence of CORBA, and therefor we have chosen it as our preferred solution. We are considering using Voyager in the future, to benefit from other features like agents, remote Java Beans, multi-casting, one-way non-blocking communications, etc.

Currently, we are using Visigenic's CORBA also known as VisiBroker. We chose VisiBroker because it was one of the first CORBA implementation for Java, because its IDL-to-Java mapping will probably be standardized by OMG and because many well-known vendors (like Netscape and Oracle) use it. VisiBroker also supports IIOP (Internet Interoperable ORB Protocol) which allows communication between different CORBA implementations (in principle clients can thus participate in communication with CORBA servers of any implementation). Our comparison of VisiBroker and RMI revealed that RMI is slower. We measured binding, rebinding and remote call times on two 133 MHz Pentium machines under Windows NT 4.0 (see table 1).

Table 1: RMI and VisiBroker Comparison

	RMI	**VisiBroker Java to Java**
First bind	70ms	24ms
Rebind	18ms	70ms (?)
Remote redraw fillCircle	5.8 – 6.3ms	4.0 – 4.4ms
Ping [7]	5.5ms	3.6ms

Our clients use VisiBroker's ORB (Object Request Broker), which allows communication with remote objects on servers (on second-tier) using CORBA specification. The ORB class library, consisting of approximately 2Mb of code, is included with Netscape (Communicator version 4.0 or latter) so there is no need to download them. In our case, however, the prototype uses v3.0 release of Visibroker instead of v2.5 included with the Communicatior, and the libraries have to be downloaded anyway due to compatibility problems.

Since only a tiny layer of the client encapsulates all the communication details with the server, it can easily be rewritten should we want to use a different communication protocol instead of CORBA, without modification to the rest of the client code. In that way that we can replace the VisiBroker's ORB with any other ORB or even with entirely different architecture, like Sun's RMI, Microsoft's DCOM or pure sockets.

Device servers implemented along the guidelines of the TACO [8] system constitute the other side of data exchange. By implementing CORBA, TACO can export CORBA objects to the network. The need for speed and the necessity to communicates with external drivers require the servers to be written in C++. In the future, when we find appropriate Java development tools, we are planing to write device servers in Java – using JNI (Java Native Interface) and a native code compiler. The result will be only one platform and one development tool, which will greatly simplify the maintenance involved.

Device servers export objects to web using CORBA from VisiBroker (through its ORB) on one side and on the other communicate with tier three: devices attached to a fieldbus and DBMS (Data Base Management System).

The communication between clients and devices is completely asynchronous. The server's response to client requests are made via callbacks. In that way client acts like a server and server acts like a client. There is also a possibility of using "repeated callbacks". The idea is that clients are able to register with servers about which data they require and how frequently it has to be obtained.

The server is independent of the underlying fieldbus. And our Web-based concept can be also used with other control systems such as EPICS, etc.

4 Security / authentication

4.1 Java security problems

Java client applications can access remote objects directly without any interference from device servers, but applets cannot because of web browser security restrictions placed on Java applets. This, the so called Java sandbox security, prevents an applet from making network connections except to the host from which the applet itself originated, and also prevents the applet from accepting any incoming connections. In our case this means that we can communicate only with the device server running on the computer where the web server runs, which is a great limitation for the control system. One solution of these problems is to digitally sign applets, but then they will run

without restrictions only on computers, which can authenticate the applet digital signature. Another alternative would be to use some kind of gateway between applets and remote objects servers. Visibroker offers such a gateway named Gatekeeper. In the latest version (3.0) of Visibroker callbacks are also supported. With Gatekeeper every applet can access a device server irrespective of where they run. Secondly, the problems with firewalls can be circumvented on both client and server side. And finally the Gatekeeper can be used as a web server.

4.2 Client authentication

When discussing the authentication issues of WWW environments, one usually thinks of the applet. Our case is different. We must not allow all clients to have control privilege of the machine. Therefore we have to provide an authentication mechanism which will grant read access to all clients, and restrict full access (reading and setting) to but a few applets. We plan to export objects through two different interfaces (IDLs): restricted and non-restricted. The restricted interface exports only "read" methods that can be invoked from everywhere – worldwide, while the non-restricted exports all ("read" and "set") methods, which in turn can only be invoked from the computers in the control room. To export the same object through two or more different interfaces we can use the CORBA tie-mechanism. Tie-mechanism using C++ template enables us to export already implemented objects onto the net.

Authentication of clients will be carried out on the basis of client IP host address check and/or authentication data that are generated by client. We can authenticate client on two levels:

- bind level: client is authenticated only when it tries to bind to the CORBA server.
- request level: for every client request the server has to authenticate the client.

Authentication on a request level can slightly slow down the server response time.

We can enforce the authentication by using CORBA interceptors a specified by OMG or by using VisiBroker's feature of handling ORB communication events.

We implement the authentication of client's IP address, in the bind interceptor or in the bind event handler. That's allowing clients from everywhere to bind using the restricted interface and only clients from control room to bind using the non-restricted interface.

5 Database access

The main idea of three-tier architecture is that clients don't access the database directly but through CORBA server.

Our design of the control system involves having three databases:

- static database that stores configuration parameters like names, constants, calibration coefficients, attributes, alarm levels, fieldbus addresses, etc.
- snap-shot data base that stores the state (i.e. all settings) of machine
- historic data base that logs data over long periods of time

We are investigating two options for databases:

- relation database RDBMS (Relation Data Base Management System)
- object oriented database OODBMS (Object-Oriented Database Management System)

We are porting the TACO database API from UNIX to Windows NT 4.0. The first version will use a hash table based database. Our control system is fully object-oriented so we will evaluate the option of using an OODBMS to ease our work of storing data. OODBMSs are ideal for storing complex object hierarchy. We are currently examining and studying POET OODBMS.

We have tested and compared the following database implementations: simple hashed database, object oriented database (POET) and relational database (InterBase). Tests were performed on a Pentium 100MHz with 32Mb of RAM under Windows 95. We measured the basic database operations that are common to all tested databases: insert, read and delete. Each measurement involved 10.000 simple data objects which contained, on the average, 70 bytes of data. The results (see table 2) indicate that POET is relatively slow. The shortcomming of the analysis is that we have made tests with a very simple data model which can be represented with only one table in relational and one object in object oriented database. We expect POET would perform better when dealing with complex object hierarchy present in thecontrol system environment.

Table 2: Database comparison between hashed, relational and OO database

	Hashed DB	OODBMS (POET)	RDBMS (InterBase)
Insert	21,09s	108,63	38,77
Read	23,45s	29,78	14,39
Delete	27,02	117,88	28,13

6 The prototype

The demonstration of already working prototype can be seen on-line on URL: http://kgb.ijs.si, which includes on-line documentation.

Clients-Java applets can be run even in any Java 1.1-enabled web browser. We tested our clients with popular web browsers on UNIX, Windows 95/NT and Macintosh environments: Netscape Navigator 3.x and Communicator 4.x and Microsoft Internet Explorer 3.x and 4.x. Our CORBA device servers run on a PC under Windows NT 4.0. For a web server we use Microsoft IIS (Internet Information Server) which is included in Windows NT 4.0, but others could well have been used instead.

We developed Java applets based on JDK 1.1.x using Symantec Cafe 1.5 and Visual Cafe 1.1e but also tested other Java development tools: Asymetrix SuperCede, IBM Visual age for Java, Borland JBuilder and Microsoft Visual J++. We have made a comparison considering the features we need the most (see table 3).

In our prototype there are several client applets and several servers: power supply, ramping, vacuum valve, vacuum pump and vacuum gauge. For simplicity we made

only one server available for tests over the Internet: the power supply server, together with two clients: PSPanel and Trend.

PSPanel client is a front-end for controlling and monitoring a power supply. With it we can set current, turn it on/off and reset the power supply. It monitors power supply's ADC and DAC currents, status and alarms. The value of ADC is shown using a gauge widget, which we downloaded freely from the Internet [9]. Values for status, ADC and DAC are sent to the client using repeated callback mechanism with frequency of 1Hz.

Trend is a client for logging currents of one or more power supplies. We plot current using sophisticated widget, which we also downloaded from Internet [10]. With the Trend client we can log current with acquisition time from 20ms to 10s. However, the max rate of 20ms can be achieved only on the local Intranet. We can change speed of drawing the current curve, and it is possible to choose which power supply and which current to log: DAC or ADC.

We measured that writing one set point needs between 22 and 40ms. If we compare this result with the result for Java - CORBA communication – approx. 4ms (see table 1) we can find that most time is spent on fieldbus communication and for this reason Java is fast enough for our control system.

7 Conclusion

The analysis of the subject and the working prototype clearly demonstrate that the idea of WWW-based control system is a good one and ripe for deployment. More important, though, is the global perspective: Java/CORBA technologies are (or at least will be) widely supported in the near future, and the investment in them seems worthwhile. Important aspects of communication, such as security, database access, transparent method invocations etc. are already precisely defined by the existing standards and implementations are available from many software vendors; the only remaining task is to pick the correct software components and integrate them in a reliable, reusable and easy-to-maintain manner.

Acknowledgments

We thank the ANKA-team at the Forschungszentrum Karlsruhe and D. Einfeld and H. Schieler in particular for creating a pleasant atmosphere of collaboration and for their hospitality during our visits. Thanks a lot to the iOsupport – team for technical supporting in using VisiBroker. Many important questions were raised and suggestion given from the colleagues in the TACO collaboration, notably A. Goetz, S. Hunt and W.D. Klotz. We thank them all.

References

[1] R. Orfali, J. Edvards, CORBA, Java and the ObjectWeb, BYTE vol. 22 no.10 (1997) 95.

[2] B. Jeram, G. Mavric, M. Plesko, M. Smolej, Experience with LonWorks as a Fieldbus for the Light Source ANKA, this conference.

[3] http://www.omg.org/

[4] http://www.microsoft.com/activex/dcom-f.htm

[5] http://www.javasoft.com/products/jdk/rmi/index.html

[6] http://www.voyager.com

[7] R. Orfali, D. Harkey, Client/Server Programming With Java and Corba.

[8] J. Mayer, Redesigning a Radio Frequency Control System with TACO, this conference.

[9] http://www.bonsai.com/java/gaugedemo

[10] http://isd.cme.nist.gov/staff/shackleford/diagapplet/plotter

Table 3: A short preview on features of different Java development tools

	Symantec Café 1.5	Symantec Visual Cafe 1.1	Asymetrix SuperCede	IBM Visual Age for Java 1.0	Microsoft Visual J++ 1.1	Borland JBuilder 1.0
Two-way RAD	no	yes	No	yes	no	yes
JavaBeans Supporting	no	no*	no*	yes	no	yes
JavaBeans Creating	no	no	no*	yes	no	yes
CORBA Support	no	no	no	yes	no	yes
Native code compiler	no	no*	yes	no	no	yes
Support for ActiveX	no	no	yes	no	yes	no
Byte code Compiler	JIT	JIT	flash			JIT

* will be available in next versions