JAVA-BASED OPERATOR INTERFACE.

G. Obukhov, Institute for High Energy Physics (IHEP) Protvino, Russia
M. Clausen, Deutsches Electronen Synchrotron (DESY) Hamburg, Germany
N. Kamikubota, High Energy Accelerator Research Organization (KEK) Tsukuba, Japan

Abstract

The proposal for Java-based Operator Interface (JOI) is discussed in this paper. Several examples were implemented for demonstration purpose. These examples can communicate with real equipment at DESY. JOI was implemented as 3-tier architecture: JavaBeans as reusable components on the client side, they communicate through CORBA/IIOP protocol with servers which provide connection to different control systems of DESY. Such approach enables to have multi-platform operator interface for diverse control systems. The development of JOI's JavaBeans and building of JOI displays was done under Windows NT. JOI can run without any modifications on any Java-enabled platform.

1. INTRODUCTION

Several different types of control systems are in use at DESY. This is because of distributed and large-scale nature of heterogeneous equipment involved into the projects. Each control system uses its own approach to building of Operator Interface (OI). Also they have different protocols for data interchange between OI and real-time layer of the control system. This leads to **incompatibility** of OIs. The developer has to create different OI for each system even if target equipment, processes have the same functionality. **Duplicating** of functionally of the same OIs for different control systems is inevitable in this case.

Different look and feel is another problem. The same actions in one OI can cause **different behavior** in another OI. It takes some time to learn not only OI itself but also differences between OIs.

Very often there is a need to add some custom elements to OI which are not currently in operator's disposal. This is a problem in **non-modular** architectures of OIs. Usually this leads to rebuilding of the whole implementation in case of the home-made OIs.

OIs which were built for usage on one platform mainly don't work on another one. This is a **portability** problem.

There are several possible implementations of Java-based operator interface which could avoid some disadvantages of current implementations of OIs described above. Java technology has many features well-suited for building of OIs. It has many useful embedded facilities. Many new advanced features appeared in the latest JDK 1.1 and Java 2 platform - JavaBeans, Java Foundation Classes (Swing), JavaIDL, Java2D, etc.

One of the possible implementations of JOI can be represented as the following 3-tier scheme (Fig. 1).

In order to provide one solution for any underlying control system one independent standard protocol should be selected. It can be either RMI or CORBA/IIOP. The last one was selected because it provides communication between objects implemented in any language (Java, C/C++ etc.).

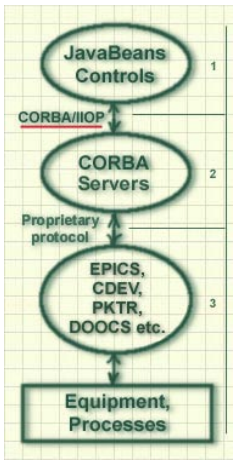

Fig. 1

Data Gateways (CORBA Servers) could be implemented as CORBA object in native (regarding language of the control system) language (C/C++ etc.). Such CORBA objects would be able to read/write data from/to the control system in proprietary protocol. Also it would be able to communicate with JOI JavaBeans (JavaBeans Controls) in standard way using CORBA/IIOP protocol. Implementing such CORBA objects doesn't take much time if you have ready-to-use CORBA development system in your disposal. VisiBroker (Inprise) was used for JOI development.

2. IMPLEMENTATION

JOI development process can be divided into three almost independent processes. First, implementation of **CORBA Server.** Second, development of all necessary **JOI JavaBeans**. Third, development of **JOI Displays** using those JOI JavaBeans.

All JOI JavaBeans mentioned in this paper were developed in VisualCafe (Symantec) under Windows NT. The examples of JOI displays were also prepared in

VisualCafe. The process of JOI display development looks very similar to the creation of GUI in any other visual builders (VisualBasic, VisualC++ etc.). A developer has just to clicks the appropriate icon (bean) in toolbar and drop this bean to working area. Then all properties of the bean can be changed in the "Properties list".

2.1 CORBA Servers

Two different control systems EPICS and DOOCS which are in use at DESY were selected for implementation of real JOI examples. It was done intentionally to test the interoperability of the same JOI Beans with different control systems. The simplified CORBA IDL program describing the interface for both example servers follows:

```
module DESY {
        interface EPICS {
            any get(in String name);
        };
        interface DOOCS {
            any get(in String name);
        };
};
```

Only one method (get) was selected to be supported by both servers. This IDL file was compiled by VisiBroker's "idl2cpp" compiler. All necessary C++ client's stubs and server's skeletons were produced as a result of this compilation. This IDL program was not compiled by "idl2java" compiler because CORBA clients (JOI JavaBeans) will be implemented as a clients with Dynamic Invocation Interface (DII). This means that they don't need precompiled stubs, because they will find CORBA server dynamically during run-time period. All required source files for implementation of "get" methods were taken from appropriate EPICS and DOOCS libraries. Only getting of float values was implemented in those examples of EPICS and DOOCS CORBA servers. Server implementation should be linked with appropriate EPICS or DOOCS library and also with VisiBroker's CORBA library. This process is depicted in the following image (Fig. 2).

When both servers were successfully compiled and linked they were started under Solaris. After registration servers are waiting for requests from clients.

In order to increase reliability of the system two instances of the same server can be started on different computers over the subnet. They should have the same module, interface and object names. When one server will be stopped because of some reasons (crashing, switching off) clients (JOI Beans) will be seamlessly reconnected to the other server.

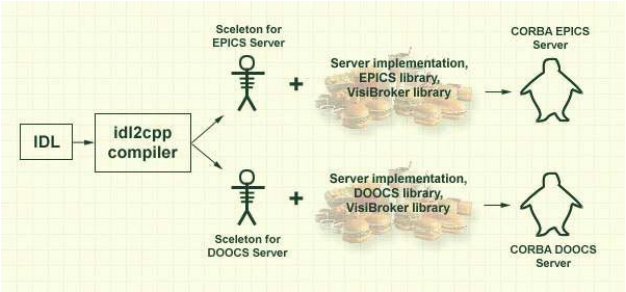

Fig. 2

This switching to another server will be done by VisiBroker's ORB transparently for clients. This approach can also be used for building of flexible loading mechanism when server "migrates" from one overloaded computer to another one with spare resources.

2.2 JOI JavaBeans

All JOI Beans could be divided into two general parts, such as "Indicators" and "Controls". "Indicators" means Beans which can visually represent value or state of the object under control. Voltmeter, oscilloscope, speedometer are the examples of indicators in real measurement technique. "Controls" means Beans which can be used for changing the value or state of the controlled object. Buttons, switches, knobs etc. are just several examples of real controls (Fig. 3).

There are several possible types of JOI Beans from the graphical representation point of view:

- Vector/Bitmap/Combined (vector and bitmap)
- With Double Buffering/Without Double Buffering
- With Anti-aliasing/Without Anti-aliasing
- Static/Dynamic (animated)

There are two possible algorithms of communication between JOI JavaBeans and CORBA Servers. First, polling is just periodical calling of communication task for reading remote data. Second, callback bares a strong resemblance to hardware interrupts. During callback client and server change their roles. A server notifies a client that some event interesting for client occurred on server side. For example, the status of equipment was changed. As a rule, client invokes relevant method upon this notification (e.g. fetching modified data).

Several additional JOI JavaBeans could be also developed in JOI framework. It could be JavaBeans providing the following functionality:

- logging and archiving
- reading CAD files (e.g. DWG, DXF)

• decoration elements in vector format (e.g. pipes, valves etc. for OIs of vacuum systems)

• implementing some kind of calculations (e.g. arithmetic, FFT etc.)

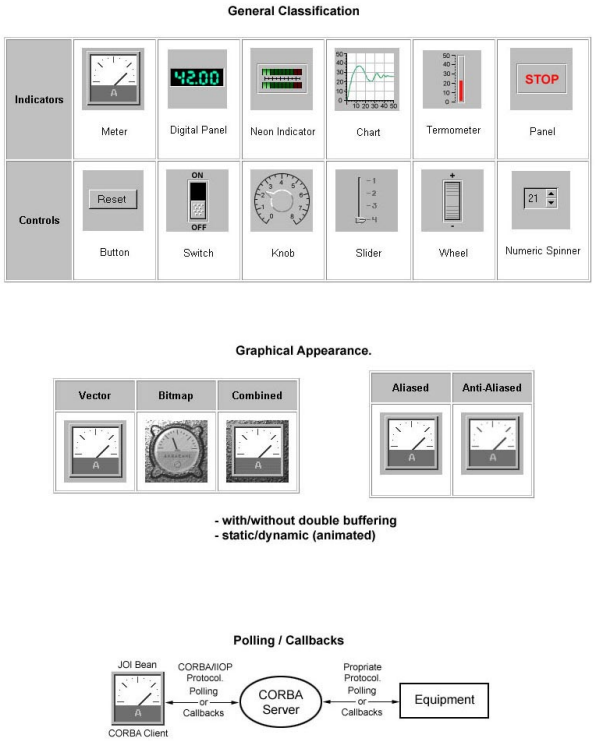

Fig. 3

Not all these Beans need GUI. But it's possible to change their properties in JavaBeans development tools as well. Some of these Beans could also have ability to communicate with CORBA Server. For example, calculation Bean could get data from equipment, make some calculations and forward the result to "indicator" Bean for graphical representation.

All JOI JavaBeans should support three main functions:

- read data from control system
- make graphical representation of this value
- fire event in case of alarm value

The properties and different appearance of "Digital Indicator" JOI JavaBeans follow (Fig. 4).

It's properties related to CORBA communication (with “Comm” prefix) should be properly configured during design time before real communication. For this purpose should be specified such properties as the name of control system (e.g. EPICS, CDEV, ...), the name of process variable and scanning period in seconds (if polling mechanism is in use). All these properties have to be just typed into properties window of corresponding Beans' development tool. During run-time period this information will be used by communication thread for ORB initialization and for reading real data. The name of

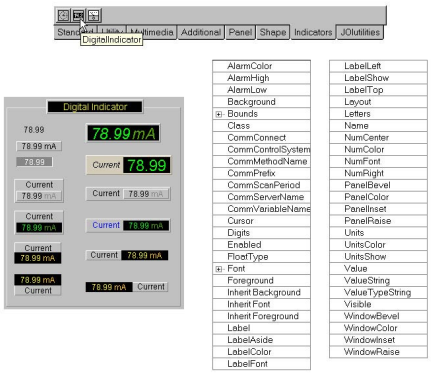

Fig. 4

control system is used only during ORB initialization phase for binding to CORBA server. This server should have corresponding name (e.g. EPICS, CDEV, ...). The scanning period is used as a period of communication thread activation. That thread invokes method (“get”) for reading value from control system. Standard CORBA Exception messages will be printed to Java console in case of wrong process variable name or bad network connection to CORBA server.

JOI JavaBeans have a Communication Customizer which allows to change the server’s name, process variable name and scanning period during run-time period (Fig. 5). If another server name was specified the JOI JavaBean will be reconnected to the server with new name.

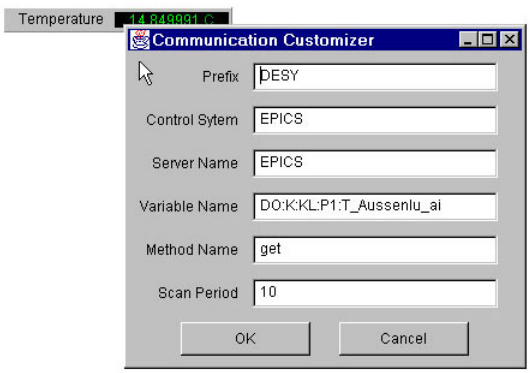

Fig. 5

The user can invoke this customizer clicking right-mouse-button when mouse pointer is inside of Bean's window (Fig. 5).

Two additional utility JOI Beans were developed. First, "Imager" Bean loads GIF and JPG images according to URL specified during design-time. It can be used for loading bitmap backgrounds for JOI displays. "CGM Viewer" is another utility Bean which can be used for loading files in vector CGM (Computer Graphics Metafile) format. Almost all widely used

vector graphics packages have possibility to export files in CGM format.

2.3 JOI Displays

Any JavaBeans development tool can be used for building of JOI Displays (Fig.6). All JavaBeans developed for JOI can be placed in Beans' palette of corresponding development tool (VisualCafe in this example).

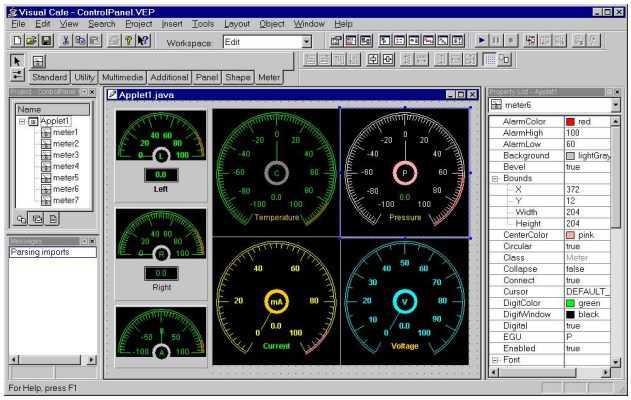

Fig. 6

In order to add new element to JOI Display the developer merely has to click its icon in toolbar and drop it to the working area ("Applet1" in our example). Then it can be resized. Also any property of selected "Meter" can be changed in "Property" window. When the whole operator interface is completed it can be started either under development environment (for debugging) or under appletviewer (in case of Applet) or under Web browser. If JOI will be used in Web browser the best way is to use Java plug-in and HTML converter for this plug-in.

3. EXAMPLES

Two different JOI displays were created for demonstration of JOI facilities. The first one is a remake of the real MEDM display which is in use at DESY in MKK group (Fig. 7). This JOI display consists of almost all process variables (22) of its predecessor. These variables are supported by EPICS control system. The second display consists of 8 process variables (Fig. 8). It was created to demonstrate the ability of the same JOI Beans (Digital Indicator) to read data from any control system (e.g. EPICS and DOOCS) using standard CORBA/IIOP protocol. In contrast with previous display (Fig. 7) this one reads data from DOOCS control system.

These two examples demonstrate also utilizing of different types of backgrounds for JOI displays. First example uses vector image for background and the second one uses bitmap background.

The background for the first example was prepared in CorelDraw and exported into CGM file. Some standard vector clipart elements were used during drawing of this background. Afterwards, it was downloaded into VisualCafe by "CGM Viewer" JOI JavaBean. Then Digital Indicators were placed in proper positions over the screen. At last, all necessary labels (from AWT package) were set and at this point the visual part of JOI display was completed.

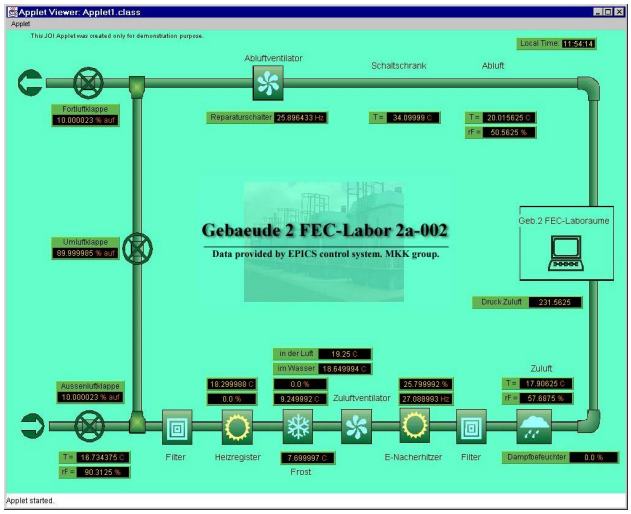

Fig. 7

Bitmap background for the second example (Fig. 8) was prepared in painting program (Adobe Photoshop) and saved in JPEG format.

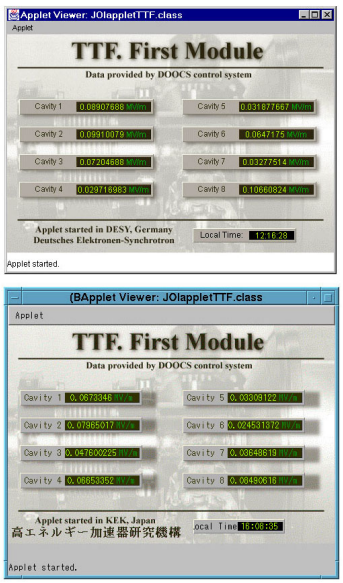

Fig. 8

It was downloaded to VisualCafe by "Imager" JOI JavaBean. "Local Timer" JOI JavaBean was added to this Applet for displaying of the current time. All communication parameters for this Applet were specified for reading data from DOOCS. After

compilation the Applet was started in appletviewer under Windows NT (Fig. 8 top).

In order to check global CORBA network the same JOI Applet was started in KEK, Japan under Digital Unix. After establishing connection between Applet (KEK, Japan) and CORBA-DOOCS server (DESY, Germany) it was possible to read data from DOOCS control system (Fig. 8 bottom).

4. CONCLUSION

Some benchmarks were done to estimate different communication schemes from efficiency point of view. For the first test EPICS' Channel Access library for Windows 95/NT was in use. Simple GUI for this test was created in Microsoft's VisualC++. This program just periodically reads one process variable.

Another testing program was written in Java. It was started on the same computer and communicated with the same IOC (Input/Output Controller) as previous testing program. This Java Applet implemented two different tests. First of them periodically reads data from IOC via CGI (Common Gateway Interface). The second one reads the same data but from CORBA-EPICS server. CGI gateway was implemented as a Perl script which invokes EPICS' "caget" utility program.

All results of these tests are collected in the table (Table 1):

Table 1

	Channel Access	CORBA Server	CGI
One cycle (ms)	8.45	10.85	290.8

Any Java program cannot work faster than program written in C/C++ because of its interpreter nature. But from the table follows that Java and CORBA overhead adds only about 2.4 ms to Channel Access time. The advantage of CORBA approach against CGI one is obvious.

There are only few ready-to-use widgets in standard JDK package which you can use for JOI JavaBeans (e.g. Button, TextField etc.). The rest of elements you ought to implement yourself or buy from third-parties. For creation of such widgets you can use the same Java development environment as for building JOI displays, for example VisualCafe.

ACKNOWLEDGMENTS

The authors wish to thank Kazuro Furukawa and Shiro Kusano from High Energy Accelerator Research Organization (KEK) Tsukuba, Japan, who provided the testing of JOI applications.

REFERENCES

[1] Client/Server programming with Java and CORBA. R.Orfali et al. 1997 ISBN 0-471-16351-1

[2] The essential distributed objects survival guide. R.Orfali et al. 1996 ISBN 0-471-12993-3

[3] Instant CORBA. R.Orfali et al. 1997 ISBN 0-471-18333-4

[4] The Java class libraries: an annotated reference. Patrick Chan et al. 1996 ISBN 0-201-63458-9

[5] VisiBroker for C++. Programmer's Guide. Version 3.0

[6] VisiBroker for Java. Programmer's Guide. Version 3.0

[7] JavaBeans Homepage. http://java.sun.com/beans/

[8] The Free CORBA homepage. http://patriot.net/~tvalesky/freecorba.html

[9] The Experimental Physics and Industrial Control System (EPICS) homepage. http://epics.aps.anl.gov/asd/controls/epics/EpicsDocumentation/WWWPages/EpicsFrames.html

[10] The Distributed Object Oriented Control System (DOOCS) homepage. http://tesla.desy.de/doocs/doocs.html