EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - PS DIVISION

CERN/PS 2001-068 (CO)

A FRAMEWORK FOR JAVA APPLICATION PROGRAMS IN THE CERN PS CONTROL SYSTEM

M. Arruat, J. Cuperus, M. Gourber-Pace, R. Hoh, E. Roux

Presented at ICALEPCS 2001, San Jose, CA, U.S.A., November 27-30 2001

Geneva, Switzerland
13 December 2001

A FRAMEWORK FOR JAVA APPLICATION PROGRAMS IN THE CERN PS CONTROL SYSTEM

Michel Arruat, Jan Cuperus, Marine Gourber-Pace, Roger Hoh, Eric Roux,
CERN, 1211 Geneva 23, Switzerland

Abstract

The user interface for a system controlling 5 inter-connected accelerators is composed of a large number of windows organized in a tree structure of application programs with a console manager at the top. All programs run in a single Java Virtual Machine (JVM) without interfering with each other. The windows show accelerator data and interaction widgets. A framework has been built to design these windows and make them interact and cooperate with a minimum of effort from the programmers. A project manager controls the life cycle of the programs, including use of templates and CVS [3]. Program development and execution can be done on both the Microsoft Windows and Linux platforms through files that are visible to both systems.

1 INTRODUCTION

The CERN PS accelerator complex consists of 5 accelerators that deliver protons, antiprotons and ions for local experiments or injection into the SPS accelerator and later the LHC. Some can change completely their mode of operation from cycle to cycle. To control this complex we need a generic control system that is data-driven so that it can work with any accelerator. We are moving from an application layer written in C/C++ to one written in Java.

2 APPLICS AND FRAMES

The basic program unit is what we will call an **applic** which extends the AbstractApplic class. An applic can have an associated window or **frame**. An application consists of a tree of applics. Application trees can also form trees and so on but formally there is only one tree of applics (see Fig.1).

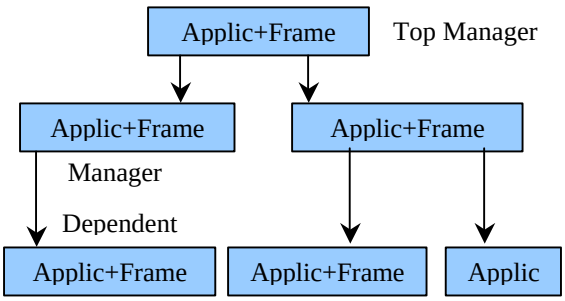

Figure 1: Tree of applics and associated frames.

Any applic can be started as a dependent of a manager applic and then executes in the same JVM. It can also be started independently in its own JVM, in which case it becomes a top manager with special privileges.

A console manager is a top manager with a frame that is essentially a large menu to start applications and to hide or iconify entire application trees. AbstractApplic contains code for managing these trees and the communication between applics.

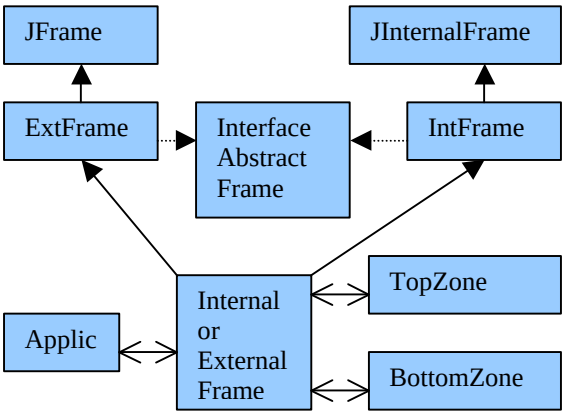

Figure 2: Class diagram for an applic frame.

A frame can be internal or external (see Fig.2) and if the programmer uses only the common AbstractFrame interface, he can transform one into the other by changing just one word in the code.

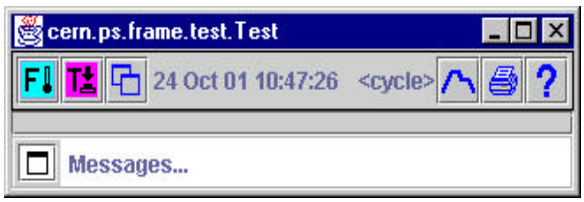

Figure 3: The default empty frame.

The frame can receive a configurable TopZone and a BottomZone with a message area, expandable to a full frame, for looking at the latest messages. In between is the UserZone that can receive Java Beans and, more specifically, components for reading and setting accelerator parameters (Fig.3). The components of the default TopZone are (from left to right):

- Freeze/UnFreeze action
- One-shot action
- Show/Select dependent frames action
- Date+Time of latest action or update
- Present beam and cycle
- Beam and cycle trigger selector
- Print the frame action
- Popup html browser with applic help-text action

3 ERROR REPORTING

Class Display has static methods for reporting:

- showMessage(message, source)
- showWarning(message, source)
- showFault(message, exception, source)

Any component or service can call these methods. The framework tries to display the message in a relevant place and faults are guaranteed to be displayed. If object *source* is in any way associated with a frame, the message appears in the BottomZone of that frame, else in the BottomZone of the top manager, else as a message pane or, if there is no graphical interface, on the output stream. In addition, faults are logged on a relational database with JDBC (Java DataBase Connectivity).

4 APPLICATION ENVIRONMENT

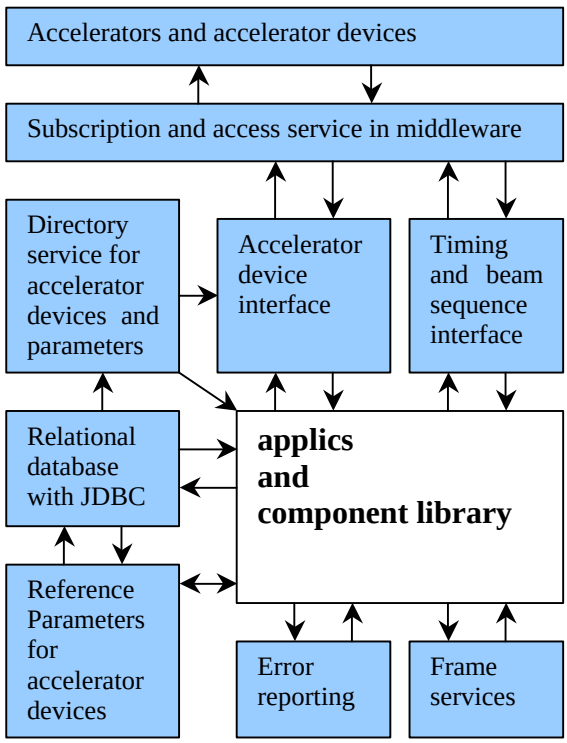

Figure 4: The environment for applics and components

The application environment is shown in Fig.4. Some of the subsystems are described in [1] and [2].

5 DESIGN TOOLS

A simple text editor is sufficient for maintaining Java files, but the use of a powerful design tool is more effective. JBuilder was chosen for the following reasons:

- It has powerful text editing capabilities.
- It works on Linux and Microsoft Windows.
- Graphical editing produces clear code so that graphical and text editing can be interleaved.
- It has good debugging facilities.
- It allows to maintain many projects and to switch from one to the other very quickly.
- Last but not least, it does not lock you in and you can move to another tool when convenient.

6 DEFERRED INITIALIZATION

Java Beans, dropped into the frame in a graphical design tool, may come to life in the tool if they are initialized completely. They may connect to the accelerator devices and this is not desirable. The Beans may therefore call a static method deferredInit(). In the operational environment, an applic is initialized in a dedicated thread and all deferredInit calls are put on a stack with the thread as a key. At the end of the initialization, the AbstractApplic superclass knows all these Beans, can call them with any parameters required by their class and may even automatically interconnect certain pairs of Beans.

7 TEMPLATES

For a new application or project *myproject*, the following skeleton files for the main applic can be generated from templates:

1. **Myproject.java**: the main applic. This may define a frame internally or refer to:
2. **MyProjectFrame.java**: a separate file for doing graphical design in a tool such as JBuilder.
3. **MyprojectHelp.html**: that will show when the Help button of the applic is pushed.
4. **Myproject.jpx**: defines the project for JBuilder.

(1) and (2) start with a section that should not be user modified. They can immediately be compiled and run and then produce a frame like Fig.3 to which you can add components.

Any time later, skeletons for dependent applics can be added, like: **Mydep.java**, **MydepFrame.java** and **MydepHelp.html**.

8 THE PROJECT MANAGER

We need to create new projects, safeguard them in CVS [3], install them and generate Javadoc. Also, we wish to work from Linux and Microsoft Windows interchangeably which produces long file paths for common visibility through the SAMBA [4] file system. The Project Manager automates all this.

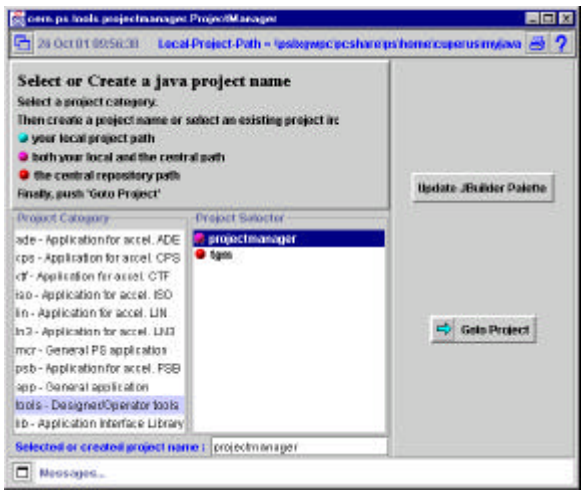

Figure 5: Select Project page of the project manager.

First, you select a project category and a project name (see Fig.5). Note that the project manager uses the framework and that it is used to maintain itself! When the *Goto Project* button is pushed, we get the design page shown in Fig.6. Many file operations are supported but not editing, for which you need a separate tool such as JBuilder.

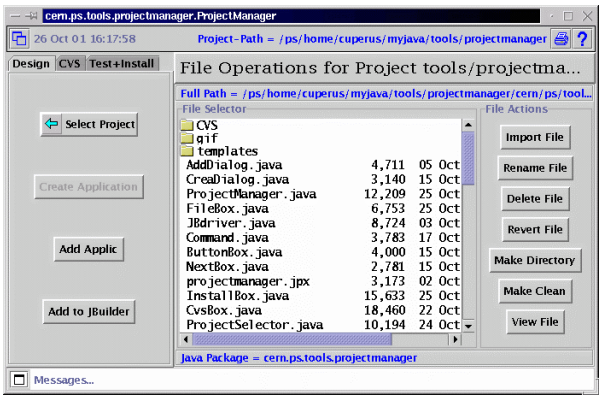

Figure 6: File Operations page (here on Linux).

The CVS operation page is shown in Fig.7. CVS is implemented on a server that can be addressed both from Linux and Windows. There is also a Test+Install page that looks similar to the CVS page and allows compilation, testing, Javadoc documenting, and installation as a .jar file.

At any time, you can expand the message zone, on the bottom of the frame, to view a complete log of your actions since the beginning of the session.

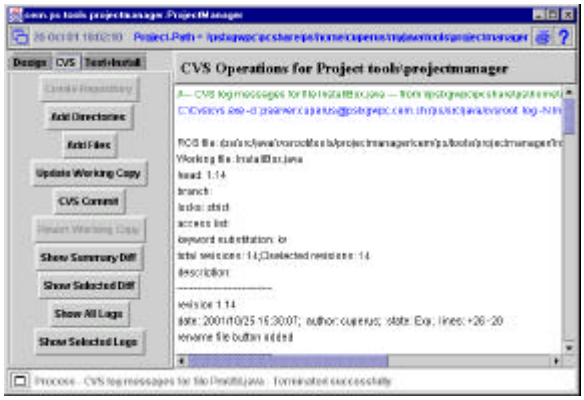

Figure 7: CVS operations page of the project manager.

9 CONCLUSIONS

With the framework, it is possible to write useful programs in a short time without necessarily being a professional programmer. The programmer, using a large library of proven components, can concentrate on his goals without being bothered by too many details.

Application programs using an early version of this framework have been in use for over a year. They perform satisfactorily as stand-alone Java programs called from a console manager in C++. This limits their number to 5 because of the large resources taken by each JVM. Also, a start-up time of 10-20s is rather long.

When enough Java programs exist, we should start them from a console manager in Java, all in the same JVM. Such a console manager must be able to run for several days without restarting. Also, the speed must be sufficient to update several hundred parameters every second. After recent improvements in the interfaces, the middleware, and the component library, we have confidence that we can reach these goals. A console manager in Java can also start (and to some extent control) C/C++ programs, but the ultimate goal should be an homogeneous Java control system.

REFERENCES

[1] A Directory Service for the CERN PS/SL Application Programming Interface, J.Cuperus et al, Proceedings Icalepcs99, p581.

[2] Remote Device Access in the New CERN Accelerator Controls Middleware, V.Baggiolini et al., this conference.

[3] Open Source Development with CVS, web address http://www.cvshome.org/ .

[4] http://us1.samba.org/samba/samba.html .