

A Java Based Analysis Environment

A.S.Johnson

*Stanford Linear Accelerator Center, Stanford, California, USA
(tony_johnson@slac.stanford.edu)*

Abstract

This document describes some preliminary thoughts on the creation of a JAVA based analysis system for future HEP experiments. The aim is to create a true client-server based analysis package, allowing physicists to work in a natural way using a desktop GUI system while still accessing the full power of a large central data server/processing system. The system will be designed to work with relatively low bandwidth between the desktop and the central data server, allowing remote access to a central data processing farm.

Keywords: Java, Analysis, Interactive

1 Overview

There are three main components to the project proposed here.

- The client side implementation which would present the desktop graphical interface to the user. This part of the system would deal with the display of data (histograms, n-tuples, scatterplots etc.), perform analysis tasks such as fitting, minimum likelihood calculations, and data correlation studies (for example by applying "real-time" cuts on one variable and studying the effects on other variables). It should also be possible to hook experiment specific modules into the client side implementation, for example to produce the capability of easily displaying experiment specific event displays.
- A data analysis language which would be used by the physicist for all non-trivial physics analysis tasks. This would ideally be based on a standard, well supported language extended to be HEP specific through the use of library functions. The physicist would write analysis routines using the graphical desktop system, and these could then be run either on the local machine, or on the remote data analysis engine. Where they actual run should be largely transparent to the end user. There are many languages that could be used for this, but one of the most promising is Java. (Don't stop reading at this point, despite the hype surrounding the Internet and Java, Java is just a programming language and is used in this context quite differently from its use in creating useless applets to "spice up" web pages!the pros and cons of using Java for this task are discussed in detail below.)
- The server side implementation would run on the data server (or servers). This would be primarily responsible for processing large amounts of data, scheduling analysis requests from different users and for providing an interface for experiment specific data and MC/reconstruction/analysis modules. Again the server side implementation would be extensible so that hooks could be provided to data in experiment specific formats, and to give access to analysis and simulation modules written in any language.

The relationship between these modules is summarized in figure 1. There are many implementation options to be researched for each of these components, but some possible strategies are discussed in the following sections. Regardless of the implementation details the project should address the goals described in the next section.

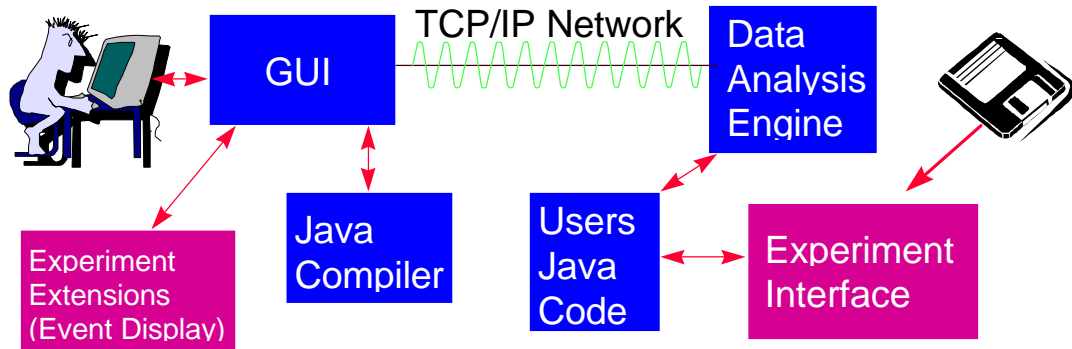


Figure 1: The components that comprise the proposed client-server system

2 Goals

- Modular design
 - easy to add new functionality or to replace specific modules to extend existing functionality
 - easy to "farm out" modules to be implemented independently
- Powerful GUI interface designed to make typical "real-world" physics analysis tasks as straightforward as possible.
- Experiment independent. Easy to port to new experiments, platforms.
- Designed to use real-world networks, to make it possible for physicists at universities to exploit the data handling capabilities at the major laboratories. (Primarily this means designing the system to use moderate bandwidth, high latency networks.)
- It should be possible to use the GUI system in a "standalone mode", i.e. without a connection to a remote data server.
- It should be possible to run analysis developed using the GUI interface in a batch environment, especially for production jobs such as filtering/simulation/reconstruction.

3 Implementation

3.1 The Data Analysis Language

The data analysis languages would be used by physicists to write all non-trivial physics analysis code. (Trivial analysis tasks, such as defining cuts, fitting histograms *etc.* could be done directly from the GUI interface - conversely major components of reconstruction such as track finding, pattern recognition *etc.* would not be addressed by the current project, but would be implemented outside of this framework and run on the data server described below.) Which ever language is chosen it will need to be "extended" by the provision of library routines that give access to the physics data (or objects) as well as access to tools such as histogramming, n-tuple building, fitting *etc.*

There are many existing languages that could be used as the basis for the data analysis language, but one language seems to stand out as particularly appropriate, namely the Java programming language from Sun. (For a project using C++ as the data analysis language see the ROOT project at CERN [1].) Java offers a number of advantages:

- Java is a modern object-orientated (OO) language whose syntax is based closely on that of C++, but lacking many of the ugly complexities which make C++ difficult to learn and use. There are many books available to make learning Java straightforward. Java is also a convenient "stepping stone" for people who eventually want to learn C++. (It should be added that some commentators liken the statement "Java is easier to learn than C++" to saying "K2 is easier to climb than Everest.")
- Java is not a "toy" language, it is a fully functional OO language with a large (and rapidly expanding) set of libraries to go with it. Java can be compiled into machine independent "bytecodes". This is ideal for the proposed application since it enables programs to be compiled on the desktop machine and debugged there, and then moved to the data analysis machine for number crunching. Security features of Java make this a "safe" procedure too, it is unlikely badly written Java code could cause problems for the data analysis engine. Debuggers are available for Java, and these also support remote debugging.
- Interpreted execution of the machine independent "bytecodes" would be fast enough for the type of analysis task envisioned here, but many "just-in-time" (JIT) compilers are now also available for Java. This allows the machine independent bytecodes to be converted to native machine code just before execution, thus providing a very efficient method of executing Java code. Using this technology it is likely that Java programs would be efficient enough to be used for large analysis tasks, such as track-cluster matching for example.
- Java is supported by major industry players such as Sun, Microsoft and Netscape. This makes it likely that the language will be around for some time.
- Unlike most other commercial languages which have been "improved" by each vendor and thus become non-portable, Java has been well defined by Sun, and its use for developing Internet applets mean that it is likely to be implemented without "extensions" on any new platforms, and to remain highly portable.
- The Java compiler and language interpreter is freely available for many platforms.

3.2 The Client Side Implementation

The client implementation is potentially the most important and the most difficult. Since this is what the end user will primarily see it needs to be very well designed, and have plenty of tutorial and reference information. It should also use "Wizard" technology as a way of introducing users to the more tricky aspects of its use, such as learning and using the Data Analysis Language.

It seems likely (to the author at least) that Windows NT will be the dominant desktop platform for physicists by the year 2000 (if not before) . It provides all of the advantages of a MAC/PC desktop, such as access to a very wide variety of commercially developed GUI applications, while at the same time acting as a good X-terminal for access to legacy UNIX and VMS systems, and provides a solid foundation for building new applications on.

On the other hand most predictions about success/failure of computing platforms turn out to be wrong, so any new application should be written to be as portable as possible. Unfortunately the goals of portability and functionality are not easy to resolve.

One possible solution would be to write the entire GUI interface in JAVA using the Java windowing toolkit (AWT). This would have the advantage that the entire application would be portable to windows, MAC and UNIX (X) as well as other future platforms. However there are two major disadvantages:

- While the Java windowing environment is portable across all platforms, its performance and capabilities are at present limited to least-common-denominator functionality. This means it is not possible to provide a fully functional , industrial strength, GUI interface at present using Java. This is a serious limitation, since experience shows that the last few percent in functionality of a GUI interface has a very large effect on the perceived ease of use.
- Java , and the AWT, probably do not provide the performance required for "real-time" displays as required, for example, by rapidly updating histograms and plots.

Another obvious option is to use visual C++ and the Microsoft Foundation Classes (MFC). This is probably the easiest way to build true professional quality GUI applications for the Windows environment. (Visual Basic is probably another good option, it is used for most commercial windows applications, but it seems like a waste of time having people learn it rather than C++). Using Visual C++ also makes it possible to exploit the Microsoft OLE technology (now cryptically renamed ActiveX) to build reusable components. For example the histogram display could be implemented as a separate OLE component that could be used in other future C++ based applications. C++ also has disadvantages:

- There is a fairly steep learning curve for programming in visual C++ and learning how to use the MFC.
- Applications built using C++ and MFC are primarily windows applications. They can be made to work (with some effort) on MAC's (in the unlikely event MAC's are still around when this application is ready for use) but there is not currently any easy way to port such applications to UNIX/X.

In the case of both C++ and Java it is possible that the disadvantages listed above may be mitigated by future developments. It is also possible to mix the two approaches listed above. For example a prototype could be readily developed using Java, and then completely or partially re-implemented using C++ later.

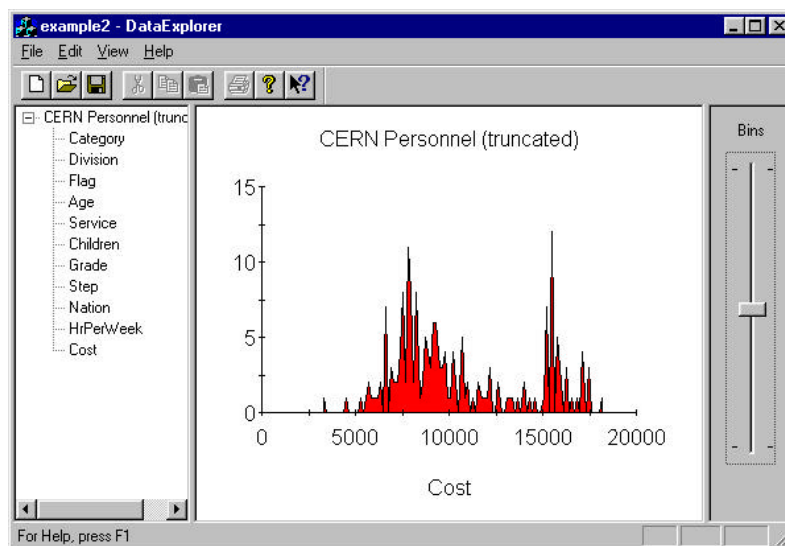


Figure 2: A pre-prototype client side implementation using MFC and a commercial plotting module.

3.3 The Server Side Implementation

This is probably the easiest part of the application to write. It is fairly clear that it should be written in portable C or C++, and if written in C++ use some portable class library such as CLHEP and or the standard template library (STL).

The client is responsible for providing an interface between the data analysis language and the experiment's data and main reconstruction/simulation and analysis code.

4 Conclusions

The project described here is currently in the initial design phase. The reason for presenting it here is to solicit input from the community as to the usefulness of the proposed system, as well as input on the choice of implementation technologies.

We are also very interested in collaborating with others working on similar projects, or in finding others interested in working with us on this project. For more up-to-date information on the status of this project see reference [2].

References

1. For more information on the root project see:
<http://root.cern.ch/>
2. For an update on the current state of this project see:
<http://www.slac.stanford.edu/~tonyj/JAW/>