SLAC-PUB-7963
October 1998

Java Analysis Studio

A. S. Johnson

Presented at International Conference on Computing in High-Energy Physics,
8/31/98ã9/4/98, Chicago, IL, USA

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

Work supported by Department of Energy contract DE AC03 76SF00515.

SLAC-PUB-7963
October 1998

Java Analysis Studio*

A.S.Johnson (tony_johnson@slac.stanford.edu)
Stanford Linear Accelerator Center, Stanford University, Stanford, California 94309

Presented at the International Conference on Computing in High-Energy Physics,
Chicago, Illinois, August 31st - September 4th 1998

Introduction - What is Java Analysis Studio?

Java Analysis Studio is a desktop data analysis application aimed primarily at offline analysis of high-energy physics data. The goal is to make the application independent of any particular data format, so that it can be used to analyze data from any experiment. The application features a rich graphical user interface (GUI) aimed at making the program easy to learn and use, but which at the same time allows the user to perform arbitrarily complex data analysis tasks by writing analysis modules in Java. The application can be used either as a standalone application, or as a client for a remote Java Data Server. The client-server mechanism is targeted particularly at allowing remote users to access large data samples stored on a central data center in a natural and efficient way.

Graphical User Interface

When the application is first started it presents the user with the interface as shown in figure 1. The goal is to present the user with a consistent interface from which all analysis tasks can be performed. The user interface is highly customizable to handle different user's preferences and features a number of "Wizards" which guide the user through complex tasks such as starting a new project.

Figure 1: *Java Analysis Studio implements several "Wizards" that guide users through tasks such as selecting data sources. The first page of the "New Job" Wizard is shown here.*

At any point the entire state of a project may be saved to a file and then later restored.

User Analysis Modules

Although some other graphical analysis environments allow users to define their analysis by wiring together pre-built analysis modules, we believe that the complexity of real-life physics analysis problems quickly makes such approaches unworkable. So although JAS allows some simple analysis operations to be performed using the graphical user interface, serious analysis is done by writing analysis modules in Java. Java is an excellent language for performing physics analysis since it is much easier to learn and use than C++, yet at the same time it is a very powerful and fully object-oriented language.

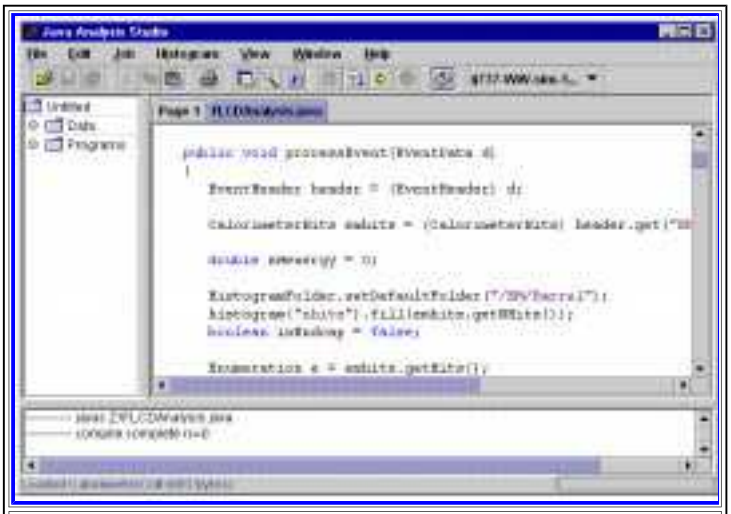

Figure 2: *Java Analysis Studio contains a built-in source code editor and compiler, so that analysis modules can be written, compiled and run without having to leave the analysis environment. Future versions of JAS will also feature an integrated debugger and code profiler.*

Java works by compiling user written classes into a machine independent format called bytecodes. As the bytecodes are executed they are normally translated on-the-fly into native machine code (a process known as just-in-time compilation) so that Java's execution is currently not too much slower than compiled C++. By early next year dynamic optimizers should be available[9] which optimize the code as it executes, and these promise to provide execution speeds which rival or even exceed statically optimized languages such as Fortran. Java modules can be dynamically loaded and unloaded from running programs, with no linking involved, which results in a very fast code, load, run, debug cycle excellent for rapid development of analysis algorithms.

Java Analysis Studio is provided with a package of classes for creating, filling and manipulating histograms. This package uses ideas derived from LHC++[1], in particular binning of histograms is delegated to a set of "partition" classes allowing great flexibility in defining different types of built-in or user-defined histograms. The built in partition classes support histograming dates and strings as well as integers and floating-point numbers, and also support either traditional HBOOK style binning while filling, or delayed binning which allows histograms to be rebinned and otherwise manipulated using the GUI after they have been filled.

A second package of classes is currently being developed which will support simple operations on three and four vectors, plus event shape analysis and jet finding. This package is based loosely on the C++ CLHEP package[2] and the Jetset event shape and jet finding routines[3].

Data Formats

Unlike most other data analysis applications which force the user to first translate the data into a particular format understood by that application, Java Analysis Studio is able to analyze data stored in almost any format. It does this by requiring that for any particular data format an interface module be available which can provide the glue between the application and the data. The application is distributed with several built-in Data Interface Modules (DIMs), which provide support for paw[4] n-tuples, hippo[5] n-tuples, SQL databases (implemented using Java's JDBC database interface[6]), StdHEP files[7] and flat-file n-tuples.

Support is provided for analyzing either n-tuples or arbitrarily complex object hierarchies. While analyzing n-tuple data a number of graphical user interface options are available for plotting columns of data singly or in pairs, as well as applying cuts. The intention is to provide an interface similar to that provided by HippoDraw[8]. While analyzing n-tuple data can sometimes be convenient it is also rather limiting, and therefore we also support analysis events consisting of arbitrarily complex trees of objects.

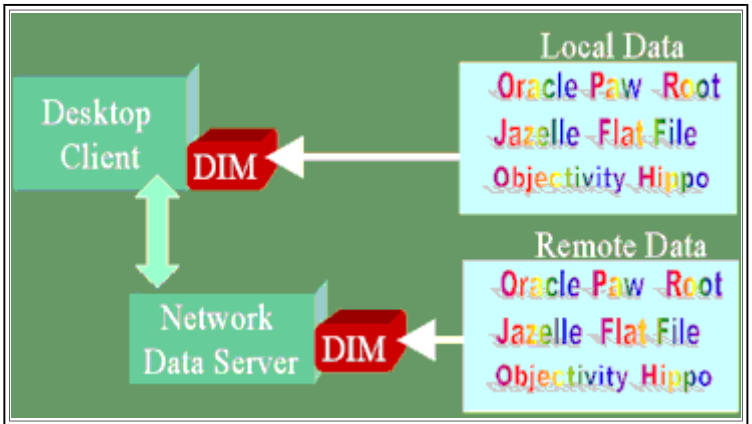

Figure 3: *Java analysis studio supports reading of data in a wide variety of data formats, and can read data stored locally or on a remote "Java Data Server".*

JAS can read data stored on the user's local machine, or stored on a remote data server. The application has been designed from the outset with this client-server approach in mind, and as a result the interface that is presented to the user is identical whether the data being analyzed is stored locally or on a remote server. When running in client-server mode the user's analysis modules are still edited and compiled locally, but when run the analysis modules are sent over the network and executed on the data server.

Since the analysis modules are written in Java and compiled into machine independent class files it is easy to move them from the users machine to the remote data server. The Java runtime provides excellent built-in security features to prevent user analysis modules from interfering with the operation of the data server on which they are running. When the user requests to see a plot created by an analysis module, only the resulting (binned) data is sent back over the network, resulting in a very modest bandwidth and latency requirements even when analyzing huge data sets. Due to its modest network requirements JAS works quite well even when accessing a remote data server via a 28.8 kb modem.

It is hoped that the client-server features built into Java Analysis Studio will prove

particularly useful to researches who typically access data from Universities where it is not possible to store the Petabyte sized data samples typically generated by today's HEP experiments. Using Java Analysis Studio such researchers can still take advantage of the powerful graphical features of their desktop machines, while analyzing data which is stored remotely. The performance of JAS is such that it is quite possible to forget that the data is not located on the local machine.

Histogram and Scatterplot Display

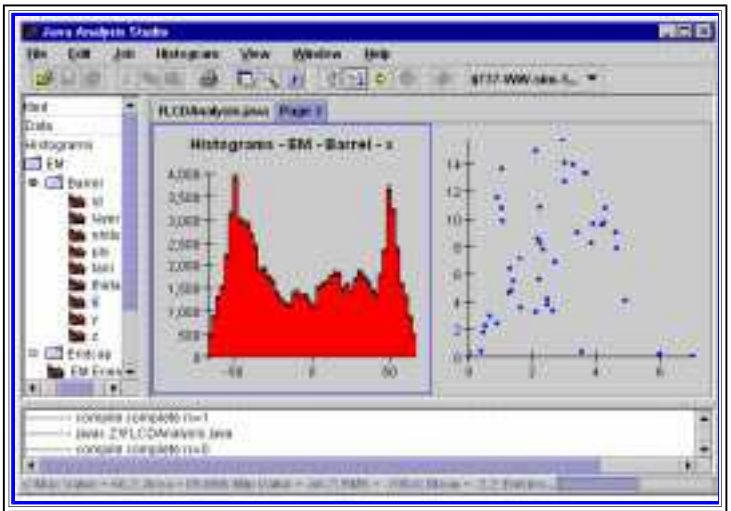

Figure 4: *Plotting features currently include the ability to display histograms and scatterplots.*

Earlier versions of Java Analysis Studio made use of a commercial charting widget. While there are many excellent commercial charting packages written in Java, they tend to be aimed primarily at developers of web pages or other applications requiring fairly basic charting facilities. To achieve ease of use they are normally packaged as Java beans which export a multitude of properties which can be set by the end user to customize the look of the chart. This approach works well as long as the required features are a subset of those already foreseen by the chart developers, but it does not work well when the chart needs to be extended to implement features that were not foreseen, since these beans are typically not extensible in an object-oriented way.

In the most recent version of Java Analysis Studio we have switched to a new set of charting tools which we have written ourselves, while still not precluding the possibility to switch back to a commercial display widget at some point in the future if some better product becomes available. We have developed two separate packages of charting tools, a low level package (called jas.chart) which consists of very general purpose classes such as Axis, Label, ChartAreas, Title, DataOverlay*etc.* which may be extended and combined together to form higher level charts. The second package (jas.hist) is built on top of the lower level package and provides less general-purpose but easier to use histograms, scatterplots, function fitting *etc.* Both of these packages have been designed with the intention that they could be used outside of the Java Analysis Studio application for other charting and histograming applications or applets.

In writing the new charting packages we have concentrated on performance and support for direct interaction with the GUI. The charts are very efficient at redrawing themselves, so that they can easily display rapidly changing data. By interacting with the GUI, end users can easily change the title or legends just by clicking on them and typing new information, and can change the range over which data is displayed just by clicking and dragging on the axes.

Extending Java Analysis Studio

Java Analysis Studio has been designed to be extended by end users and/or by experiments. A number of API's have been defined to make it possible to build extensions without having to understand the details of the Java Analysis Studio implementation. Currently supported extension API include:

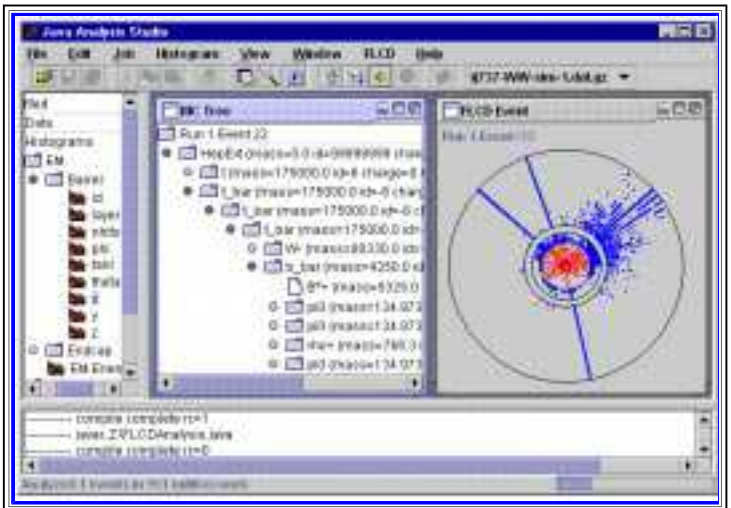

Figure 5: The Plugin API allows experiment specific extensions to be tightly intgrated into the graphical user interface.

- The Data Interface Module API for writing interfaces to new data types.
- The Function API for writing new functions.
- The Fitter API for writing new fitters. This allows user defined fitters to be used in place of the built-in least squares fitter. (A fitter which used Java's Native Interface to call Minuit would be a useful extension).
- A plugin API for writing user or experiment specific extensions to the JAS client. The Plugin API allows extensions to be tightly integrated into the client GUI by by supporting creation of new menu items, creation of windows and dialogs, and interaction with the event stream. Figure 5 shows two plugins that have been developed for Linear Collider Detector studies, one shows the MC particle hierarchy as a Java tree, and the other is a simple event display. In the future we hope to make available a plugin that will allow a full WIRED[11] event display to run within JAS.

Implementation

The application has been built as far as possible on industry standards and using commercial components where consistent with the goal of making the final application redistributable with no runtime license fees.

Availability

At the time of writing Java Analysis Studio version 1.0 Beta 1 is available for download from our web site at: http://www-sldnt.slac.stanford.edu/jas. Currently we *support* for Windows (NT/95/98), Linux and Solaris, however since the application is written entirely in Java (except for the optional paw, hippo and stdhep DIM's) it should work on any platform with a JDK 1.1 compliant Java Virtual Machine.

References

1. Histograms as Objects, Y.Adesenya, presented at CHEP97. http://www.ifh.de/CHEP97/paper/192.ps
2. CLHEP - A Class Library for High Energy Physics, http://wwwcn1.cern.ch/asd/lhc++/clhep/index.html
3. T. Sjöstrand, Comp. Phys. Comm. **82** (1994) 74.
4. PAW - Physics Analysis Workstation,http://wwwcn.cern.ch/asd/paw/
5. HippoPlotamus, Michael F. Gravina, Paul F. Kunz, Tomas J. Pavel, Paul E. Rensing. Contributed to 10th International Conference on Computing in High Energy Physics (CHEP 92), Annecy, France, 21-25 Sept. 1992.
6. The JDBC Database Access API, http://java.sun.com/products/jdbc/index.html
7. StdHep, Lynn Garren and Paul Lebrun, http://www-pat.fnal.gov/stdhep.html
8. HippoDraw as Electronic Notebook, P. Kunz, HEPVis 98, http://www-sldnt.slac.stanford.edu/hepvis/Papers/Web/5/
9. The Java HotSpot Virtual Machine Architecture, David Griswold, http://www.javasoft.com/products/hotspot/whitepaper.html
10. Java Grande Forum, http://www.javagrande.org/
11. World-Wide Web Interactive Remote Eventy Display M.Dönszelmann, http://wired.cern.ch/

Acknowledgements

A lot of the work of developing the Java Analysis Studio code has been done by Kevin Garwood and Jonas Gifford and Azhar Zuberi, students working at SLAC from the University of Victoria, British Columbia. We would also like to thank Mike Ronan for his enthusiasm concerning Java and for trying out many of the early versions of JAS and giving us useful feedback.

---

* Work supported by Department of Energy contract DE-AC03-76SF00515.