SLAC–PUB–8301
Feb 2000

JAVA ANALYSIS STUDIO AND THE hep.lcd CLASS LIBRARY [a]

M.T. RONAN
Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA

J. BOGART, G. BOWER, A.S. JOHNSON
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309, USA

N. SINEV
Physics Department, University of Oregon, Eugene, OR 97403-1274, USA

D. BENTON
Department of Physics and Astronomy, University of Pennsylvania, Philadelphia, PA 19104-6396, USA

The Java Analysis Studio and the hep.lcd class library provide a general framework for performing Java-based Linear Collider Detector (LCD) studies. The package is being developed to fully reconstruct 500 GeV to 1.5 TeV e^+e^-annihilation events for analyzing detector options and performance. The current North American LCD reconstruction effort is aimed at comparing different detailed detector models by performing full detector simulation and reconstruction. This paper describes the JAS/hep.lcd distributed analysis framework and some aspects of the reconstruction and analysis object modeling.

1 Introduction

The Java Analysis Studio [1] (JAS, pronounced jazz...) is a desktop data analysis application written mainly in the Java Language Environment [2]. It features a flexible modular design with a colorful histogram viewer and a well-tailored graphical user interface. The JAS framework allows users to perform arbitrarily complex data analysis tasks by writing Java analysis modules. The application runs as a local standalone program or as a client of a remote Java Data Server. The client-server architecture allows users to access large data samples stored on remote data centers in a natural and efficient way.

The hep.lcd class library, implemented entirely in Java, forms a framework for processing and analyzing High-Energy Physics (HEP) data. The current North American Linear Collider Detector (LCD) reconstruction effort includes first-pass pattern recognition, clustering, track-cluster association and jet finding for use in detailed detector studies.

[a] Presented at the International Workshop on Linear Colliders, April 28 - May 5, 1999 Sitges, Barcelona, Spain

Work supported by Department of Energy contracts DE–AC03–76SF00515 (SLAC) and DE–AC03–76SF00098 (LBNL).

In this paper, we describe some features of JAS and our Java analysis framework, and outline some aspects of the hep.lcd object modeling. We show results from our first pass at complete reconstruction of 500 $GeVe^+e^-$annihilation events.

2 Java Analysis Studio

The Java Analysis Studio is mainly written in Java, a modern Object-Oriented (OO) language which is easy to learn and use. It includes extensive core libraries for building graphical user interfaces, for network access and for many utility purposes, and has many other application programming interfaces (API's). The Java compiler writes byte-compiled code which is portable and dynamically loadable. The Java Language Environment, available on all modern machines, provides a Java Virtual Machine to interpret the byte code and, in most cases, a Just-In-Time (JIT) compiler to generate binary code for faster execution. Java thus allows rapid prototyping and transparent porting of code and compiled objects to any platform running the Java VM.

The main features of JAS are described below.

2.1 Graphical User Interface

A graphical user interface (GUI) is aimed at making the program easy to learn and use, and is well-tailored to the steps required for various operations. It is highly customizable to handle different user's preferences and features a number of "Wizards" which guide the user through operations such as starting a new project, selecting data sources or writing analysis templates.

JAS provides a package of classes for creating, filling and manipulating histograms. Built-in partition classes support histograming dates and strings as well as integers and floating-point numbers, and allow either fixed binning while filling, or delayed binning with subsequent manipulation and rebinning using the GUI.

2.2 User Analysis Modules

While JAS allows simple analysis operations and histogram viewing to be performed using the graphical user interface, serious analysis is done by writing analysis modules in Java. Two core analysis class libraries, hep.physics and hep.analysis, have been implemented. The hep.physics library provides basic particle physics definitions and utilities for displaying particle hierarchy and property information. The hep.analysis library contains definitions of event data types, generators and analyzers, as well as classes to define, partition and store histograms. JAS includes a "Wizard" for creating templates for event generation or analysis modules, and a built-in Editor and Compiler. Analysis modules can be dynamically loaded and unloaded while the program is running, with no linking involved, resulting in very fast turn-around during software development and debugging.

2.3 Data Formats

Data in different formats are input to JAS through interface modules. JAS is provided with several built-in Data Interface Modules (DIMs), which provide support for PAW n-tuples, SQL databases and standard (StdHEP) generator files. JAS can read data stored on the user's local machine, or stored on a remote server. Once the data source is selected, the interface presented to the user is identical whether the data being analyzed is local or remote.

2.4 Extending the Java Analysis Studio

JAS has been designed to be extended by end users for different uses or experiments. A number of API's have been defined for this purpose:

- The Data Interface Module API for writing interfaces to new data types.
- The Function API for writing new analytical functions.
- The Fitter API for writing new data fitters.
- The Plugin API for writing user or experiment specific extensions to the JAS client.

The Plugin API allows extension of the GUI by supporting the creation of new menu items, dedicated windows and dialogs, and provides methods for interacting with the event stream.

3 Reconstruction and Analysis Software (The hep.lcd Class Library)

The present North American Linear Collider Detector (LCD) simulation effort is focused on detailed Monte Carlo tracking and event reconstruction studies of two design options, known as Large and Small.[3] The Large detector is based on a central Time Projection Chamber (TPC) tracking system with a 3 Tesla superconducting coil mounted outside of Electromagnetic (EM) and Hadronic (HAD) calorimeters. The Small silicon tracking detector has a 6 Tesla field with the coil located inside of the Hadronic (HAD) calorimeter. Both detector models include an inner vertex tracking system based on similar CCD designs, and an outer muon identification system. In the LCD simulation system[4], final state particles from standard StdHEP generators are tracked through detailed detector models in a Gismo C++ tracking package.

The hep.lcd class library provides an extensible Java framework for running sophisticated analyses such as fast parameterized Monte Carlo (FastMC) simulation and full event reconstruction. The LCD simulation system allows analysis of generator-level four vectors, FastMC quantities or space point data from the full tracking package. Use of the Java Language Environment allows for rapid development of reconstruction algorithms and excellent through-put in processing. A suite of physics analysis tools, such as histograming, event display and jet finding, enable users to build on proven code. The overall design emphasizes flexibility and extensibility, typically providing multiple algorithms in following object-oriented design

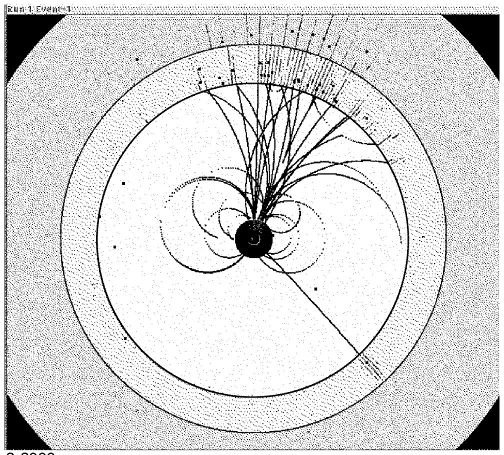

Figure 1: Display of a simulated 500 $GeVe^+e^-\to W^+W^-$event for the Large TPC detector model in the (x-y) bend plane. The EM and HAD calorimeters are shown in light and dark gray, respectively. For this detector model, the large number of contiguous space points measured in the tracking detector at 144 radial layers allows straight-forward pattern recognition.

rules. The framework can be used as a standalone or run inside Java Analysis Studio.

Several Java packages have been written to provide a flexible framework for developing reconstruction and analysis software

- event - describes the event structure and provides event handling methods
- geometry - specifies the detector being analyzed and its segmentation
- io - includes methods for reading generator output and LCD data files
- mc - contains fast Monte Carlo and full detector point smearing methods
- physics - 3- and 4-vector utilities, event shape determination and standard ("y-cut") jet finders translated from the JETSET library [5]
- plugin - a simple event display shown in Fig. 1
- recon - full reconstruction packages for tracking, cluster finding, etc. discussed below
- util - includes LCD analysis framework building blocks (Sec. 3.1) and general reconstruction utilities such as a helical track swimmer.

3.1 A Java Object-Oriented Framework

The object modeling of the hep.lcd.util.driver package allows flexible modular designs. Modular **Processor** objects carry out the event processing, implementing defined processing methods such as start, process and stop. **Driver** objects, which have methods for adding **Processors** to be executed, pick up the **LCDEvent** data generated by the Monte Carlo tracking package. Each loaded driver runs through its list of **Processors** in following the prescribed data processing sequence. Each processor receives the event for processing and can add its reconstructed quantities to it. **Drivers** also implement the **Processor** methods so that they can also be driven by other drivers in assembling an arbitrarily complex sequence. A **ProcessorContext** object is held by each **Processor** to specify the processing, histogramming and debugging options for that processor.

3.2 Track Finding and Fitting

The hep.lcd.recon.tracking package[6] contains full pattern recognition and track fitting software, including simple vertex detector hit association. Track reconstruction interfaces are specified for the **Tracker**, **TrkFinder**, **TrkFitter** and **VertexDetector** objects. Common base implementations are provided in each case: An abstract **AbsTracker** class, implementing the **Processor** methods (Sec. 3.1), picks up the **LCDEvent** data and orders the **TrackerHits** by tracker layer for use by the track finder. The **AbsTrkFinder** class implements common methods for returning track information, leaving the actual pattern recognition to concrete implementations. The **AbsVertexDetector** processor class accesses the **VXDHits** from the **LCDEvent** data and orders them by vertex detector layer. The base **AbsTracker** class is extended in specifying the dimensions and parameters of the tracker by detector specific classes such as **TPCReco** and **SiliconReco** for the Large and Small detector options, respectively. For track finding, a 3D pattern recognition class, **TPCPat2** ported from existing C++ code, extends the **AbsTrkFinder** methods in using various triplets of layers to find helical tracks that originate from the origin and satisfy minimum p_t requirements. Track fitters based on existing algoritms have been ported to Java and are being developed. A **CCDReco** class, extending the **AbsVertexDetector** class, specifies the CCD vertex detector option and associates the vertex detector hits with the reconstructed tracks. Fully reconstructed tracks, **ReconstructedTracks**, which implement the **hep.lcd.event.Track** definitions are added to the event by a separate processor **AddReconTrks**. The track reconstruction package is controlled by the **TrackReco** driver which extends the **Driver** class (Sec. 3.1).

3.3 Cluster Finding

Electromagnetic and Hadronic Calorimeter clusters, **hep.lcd.event.Clusters**, are formed from the **CalorimeterHit** cells contained in the **LCDEvent** data. A number of calorimeter clustering algorithms have been implemented. The **ClusterCheater** uses Monte Carlo generator information to “cheat” in finding the clusters due to individual particles. In cells where the showers from more than one **MCParticle** have contributed, the corresponding deposited energy is unfolded to allow complete separation of the clusters due to different parent particles. Other realistic cluster finders are under development. A **SimpleClusterBuilder** finds clusters which consist of contiguous hit cells, and a **RadialClusterBuilder** builds

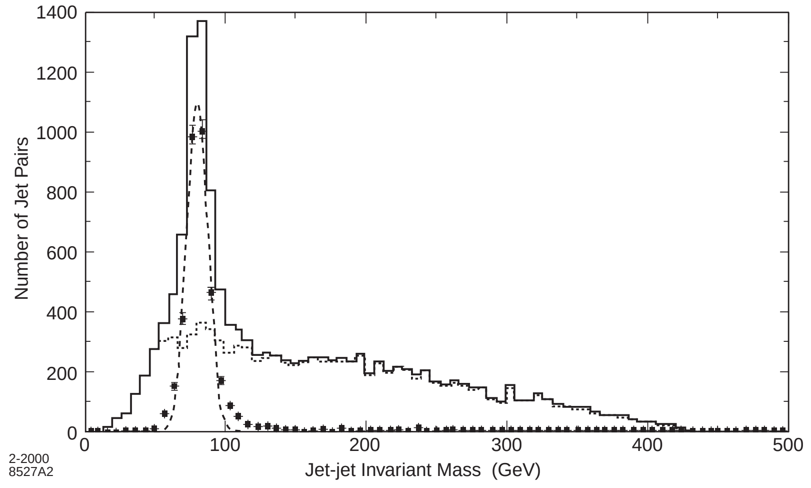

Figure 2: Inclusive jet-jet W boson reconstruction in simulated 500 $GeV e^{+}e^{-}\rightarrow W^{+}W^{-}$reactions for the Large TPC detector model. Individual jets are found following a standard Energy-Flow algorithm. The W mass is then reconstructed in events with both W's decaying hadronically. The solid histogram represents all jet pair mass combinations. The data points and light dashed histogram are constructed for jets from the same or different W-boson parents. The heavy dashed histogram is from a Gaussian fit to the central W-boson mass region with a fitted mass and resolution of 80.5$\pm$0.15 GeV and 7.4$\pm$0.14 Gev, respectively.

clusters radially outward. For ease in program development, higher level objects are defined which extend the **Cluster** definitions to include additional cluster information, such as the innermost point on a cluster for track-cluster associations, and corresponding accessor methods.

3.4 Full Event Reconstruction

In completing a first pass reconstruction of simulated Monte Carlo events, each subsequent processor uses the reconstructed tracks and clusters from the **TrackList** and **ClusterList** stored in the **LCDEvent** data. A **TrackClusterAssociator** processor separates charged and neutral clusters in following a standard Energy-Flow algorithm. A hybrid Monte Carlo system is used to mix fully reconstructed tracks and clusters with a parameterized simulation for any particles that may have been missed. The resulting mixed particles (both reconstructed and added) are used by the **JetFinders** to find the expected number of jets for the types of events being analyzed, e.g. 2,3 or 4 jets for $e^{+}e^{-}\rightarrow W^{+}W^{-}$depending on whether the W's decayed leptonically or hadronically. The jets are used to study invariant mass reconstruction for W's, Z's and top quarks for different detector designs. The jet-jet W mass reconstruction[7] for $e^{+}e^{-}\rightarrow W^{+}W^{-}$simulated for the Large detector design is shown in Fig. 2. The fitted W mass and width are consistent with Fast Monte Carlo simulations. This full reconstruction technique also demonstrates non-gaussian tails to the mass distribution as might be expected.

4 Java-based Computing

There are a number of advantages to the chosen Java-based computing model.

4.1 Distributed Analysis Model

The client-server architecture of the LCD reconstruction framework allows users to access the sizable Monte Carlo data samples in performing Java-based distributed analyses. The user's analysis modules are still edited and byte-compiled locally, then sent over the network to be executed on a remote server located at a data center. The user can monitor histograms as the job is executing, or disconnect and reconnect later. When the user requests a plot created by an analysis module, only the resulting selected histogram data are sent back over the network. The data are then manipulated, binned and fit on the local client machine, thus taking advantage of the powerful graphical features and computing capabilities of desktop machines.

4.2 Overall Performance

The use of Java and the JAS distributed analysis environment has several performance advantages. Since Java is easy to read, the OO design of the code is more transparent and allows good performance optimization in the object modeling. The Java compiler provides clear programming messages in enforcing the design model, and dynamic loading results in quick turn-around in program development. Java itself is fast! With a good JIT compiler, the GUI's are responsive and program execution times are quite good. We find that the calorimeter cluster finding and the track pattern recognition run in 0.1 sec/event and 1-2 sec/event, respectively, for 500 GeV $W^+W^-$and $t\bar{t}$ events on affordable PC's. The computing model makes good use of centralized data centers, eliminating the need to transport data to remote sites, and makes the never-ending task of porting code to different platforms obsolete.

References

1. A.S. Johnson, *Java Analysis Studio*, http://www-sldnt.slac.stanford.edu/jas.
2. Sun Microsystems, Inc., http://java.sun.com.
3. J. Brau, in these proceedings.
4. R. Dubois, *LCD Small and Large Calorimeter Single Particle Resolutions*, in these proceedings.
5. G.R. Bower, *Reconstruction of High Mass Particles from Hadronic Jets at a High Energy Lepton Collider*, in these proceedings.
6. M.T. Ronan, *Tracking in Full Monte Carlo Detector Simulations of 500 GeV e^+e^- Collisions*, in these proceedings.
7. M.T. Ronan, *W Reconstruction in Full Monte Carlo Detector Simulations of 500 GeV e^+e^- Collisions*, in these proceedings.