FULL EVENT RECONSTRUCTION IN JAVA

N. A. Graf, SLAC, Menlo Park, CA 94025, USA
for the LCD software development team

Abstract

We describe a Java toolkit for full event reconstruction and analysis. The toolkit is currently being used for detector design and physics analysis for a future e^+e^- linear collider. The components are fully modular and are available for tasks from digitization of tracking detector signals through to cluster finding, pattern recognition, fitting, jetfinding, and analysis. We discuss the architecture as well as the implementation for several candidate detector designs.

INTRODUCTION

The International Linear Collider

Detectors are currently being designed to fully exploit the physics discovery potential of e^+e^- collisions at $\sqrt{s}$=1TeV by performing precision measurements of complex final states. This will require exceptional momentum resolution,excellent vertexing capabilities, and hermetic Particle Flow calorimetry.

LCD Mission Statement

The Linear Collider Detector (LCD) Simulation and Reconstruction group is tasked to provide full simulation capabilities for the International Linear Collider (ILC) physics program, including physics simulations, detector designs, machine-detector interface and background studies. Due to the long leadtime of this project, we need flexibility for new detector geometries and technologies and innovative reconstruction algorithms. Limited resources demand efficient solutions and focused effort. The goal is to develop a common simulation environment used in ILC studies which allows sharing of detector designs, reconstruction algorithms, and code. The system should be flexible, powerful, yet simple to install, maintain and use. The metric of performance is the ease with which a physicist having an idea can implement and test its effect!

Why Java?

Java is a pure Object Oriented (OO) language which is simpler to learn and use than C++. The language design emphasizes ease-of-use over performance (e.g. garbage collector takes care of freeing unused objects, freeing developers from worrying about memory management). It is a new language, with no historical baggage and very powerful standard libraries. A large number of open-source libraries including libraries for scientific computing are available. It is also platform independent making it very suitable for the Grid environment. Compile it once and it just runs everywhere there is a Virtual Machine (Linux, Windows, Mac OSX, Solaris,). Physicists can concentrate on writing clean OO code to perform analysis tasks. Furthermore, the runtime performance of Java code is close to that of C++, in real life maybe 20-30% overhead typical.

"Particle Flow" Reconstruction

The desired precision and resolution of the ILC physics measurements require a new detector paradigm, intimately connected with the event reconstruction. We are aiming for a very tight loop connecting detector design to simulations, reconstruction and analysis, with the results feeding back into the design. The basic idea behind the "Particle Flow" algorithm is to uniquely reconstruct each particle in the event with high precision and efficiency. One is able to measure the momenta of charged tracks in the central tracker with superb resolution. One then attempts to associate the corresponding energy deposition from this particle in the calorimeter and remove those cells from further analysis. Photons are measured with high efficiency in finely segmented electromagnetic calorimeters with good energy resolution. Any remaining neutral hadrons are measured with reasonable resolution in hadron calorimeters.

$$\sigma^2_{Ejet} = \sigma^2_{Echarged} + \sigma^2_{Ephotons} + \sigma^2_{Eneut.had.} + \sigma^2_{confusion}$$

The confusion term, arising from the incorrect assignment of calorimeter energy deposition to charged and neutral hadrons, is the hardest term and cannot be correctly simulated with fast 4-vector smearing. Its understanding requires detailed detector designs coupled with realistic calorimeter shower simulation and a full *ab initio* reconstruction.

RECONSTRUCTION OVERVIEW

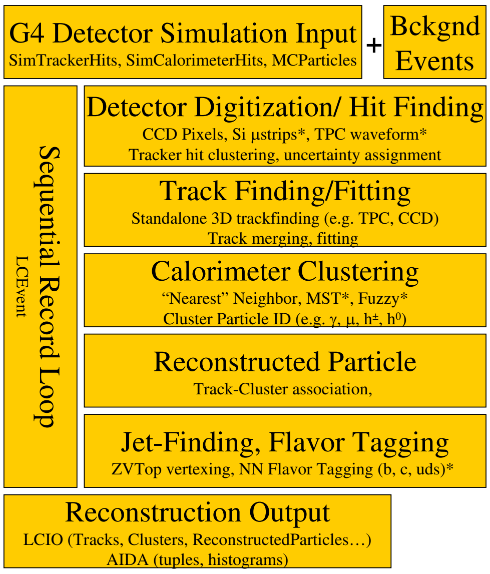
The reconstruction software runs either standalone or inside Java Analysis Studio (JAS3). A fast, 4-vector smearing Monte Carlo provides a best you can do target towards which the full reconstruction aims. One can overlay arbitrary combinations of beam and physics background at the detector hit level, as well as adding salt and pepper noise hits, or simulating inefficiencies by dropping hits. The Geant4 full simulation program writes out the full Monte Carlo hit information; detector digitization is deferred until the reconstruction phase. This allows detector readout schemes to be varied (within reason) to study the effects of

CCD pixel size, depletion depth, silicon microstrip pitch, TPC readout electronics, etc. Digitizing the hit information provides more realistic hits by simulating the effects of electronic noise and crosstalk, hit merging, ghost hits arising from strip pairing, and cluster dependent position measurement uncertainties. The full reconstruction package features *ab initio* track finding and fitting, calorimeter clustering, individual particle reconstruction (e.g. cluster-track association). The analysis suite includes a pure Java Neural Net implementation available for training and use, physics tools such as topological vertex finding, jet finding and jet flavor tagging. Analysis tools feature an LCD-specific WIRED event display, as well as a full suite of ntuple and histogram manipulation and fitting functions. The code and documentation (including examples and tutorials) can be downloaded from the hep.lcd homepage [1].

Event Loop

Utility classes are provided to read in events in various file formats (e.g. sio, lcio, root, ascii and Java native serialization). Reconstruction algorithms are implemented as Drivers with well-define callback hooks, e.g.:

```
add(Processor p);
beforeFirstEvent();
setDetector(Detector det);
processEvent(EventData event);
afterLastEvent();
```



G4 Detector Simulation Input
SimTrackerHits, SimCalorimeterHits, MCParticles
+
Bckgnd Events
Sequential Record Loop
LCEvent
Detector Digitization/ Hit Finding
CCD Pixels, Si µstrips*, TPC waveform*
Tracker hit clustering, uncertainty assignment
Track Finding/Fitting
Standalone 3D trackfinding (e.g. TPC, CCD)
Track merging, fitting
Calorimeter Clustering
“Nearest” Neighbor, MST*, Fuzzy*
Cluster Particle ID (e.g. γ, µ, h±, h0)
Reconstructed Particle
Track-Cluster association,
Jet-Finding, Flavor Tagging
ZVTop vertexing, NN Flavor Tagging (b, c, uds)*
Reconstruction Output
LCIO (Tracks, Clusters, ReconstructedParticles…)
AIDA (tuples, histograms)

Figure 1: The Event Reconstruction Flow.

Drivers implement the Processor interface, so they can be nested (i.e. Reconstructors can call other Reconstructors). The Event contains some predefined collection hooks,

```
event.getCalorimeterHits();
event.getTrackerHits();
```

However, because of the intrinsic OO nature of Java, arbitrary objects can also be added to and retrieved from the Event

```
event.put(''myStuff'', myStuff)
```

This allows users to rapidly prototype reconstruction algorithms without having to worry about the details of the event model. Successful features are then promoted to the formal event interface as part of the release procedure.

Detector Hit Digitization

As mentioned previously, the Geant4 full simulation package writes out the full Monte Carlo detector hit information. Since this would be a prohibitively large (and mostly useless) amount of information for each element of a particle shower in a calorimeter, the calorimeter hit (SimCalorimeterHit) information is quantized into cells in the Geant4 program. That is, only the total amount of energy deposited (and time of deposition) by each primary particle in a calorimeter cell is recorded. This, however, is done at a finer segmentation than is expected to be available in the real detector. Studies of the impact of various readout options such as segmentation, number of readout bits and thresholds can be undertaken by ganging cells and digitizing at the reconstruction stage. Hits in the tracking detectors (SimTrackerHit) preserve the full MC information for each hit: position, time, dE/dx, MC parentage, etc. The hits are then digitized at the reconstruction level, allowing the effects of strip pitch, pixel size, charge sharing, electronic noise, etc. to be studied.

CCD Digitization

As an example of a tracking detector hit digitization, the vertex detector hits from simulated events are converted int CCD pixels. The package finds charge deposited in each pixel, adds electronics noise and digitizes the resulting signal. Clustering software then associates contiguous pixels into clusters, splitting if necessary. The coordinates of the found cluster centroids are used to replace TrackerHits in the events. Further event processing (track finding, fitting, and so on) proceeds as before. The user can set CCD parameters (like thickness, depleted layer depth, epitaxial layer thickness and so on), electronics parameters (noise, ADC conversion scale, pixel and cluster thresholds), and algorithm processing parameters (like cluster center calculation method).

Track Finding and Fitting

The released reconstruction software features full pattern recognition in 3D detectors. For 2D outer trackers, such as silicon microstrip trackers, tracks found in the vertex detector can be extrapolated outwards. The current software is specifically tuned for two strawmen detectors, but work is ongoing to generalize the pattern recognition software. The default track fitter is a weight matrix implementation to account for material effects, and single detector or combined fits are supported. This is sufficient for many preliminary studies, since the proposed tracking detectors are very lightweight and the magnetic fields quite uniform, but to extract the maximum tracking resolution and make the fitting more flexible, a Kalman filter algorithm is being developed.

Calorimeter Clustering

Associating energy depositions in the calorimeters to particle showers is at the heart of the ”particle flow” algorithm, putting extra emphasis on calorimeter clustering. A generic cluster interface has been defined, and several clustering algorithms are currently implemented. A MC cheater is available to associate cells based on the known MC track depositing the energy to provide a target towards which to aim. A Nearest-Neighbor algorithm, with user-definable neighborhood domains for association is also available. A fixed-radius cone algorithm has shown promise as a fast, efficient clustering algorithm for finding electromagnetic showers. To assist developers, a quality assurance package working only on the Cluster interface has been written to provide efficiency and fake rates. Clustering refinements include combining clusters found in different calorimeters (e.g. hadronic and electromagnetic) and across detector boundaries (e.g. barrel and endcap). The fine granularity of the proposed calorimeters allows the particle identity of clusters to be identified with reasonable confidence based on neural net analyses of the cluster characteristics such as shape and energy moments.

Reconstructed Particle

Particle Flow algorithms are being developed with minimal coupling to specific detector designs. The photon-finding and muon reconstruction packages are fairly mature. Although several approaches are being investigated, the emphasis for charged hadron reconstruction has been placed on a track-following algorithm. Tracks found in the central trackers are extrapolated into the calorimeters and energy deposited in the readout cells is added to the track until a threshold is reached, either in the matching of energy to momentum of the track, or a distance from-the-track metric is exceeded. The high granularity of the calorimeter readout design allows the traces of minimum ionizing particles (either muons or charged hadrons before they shower) to be reconstructed with high efficiency and reasonable precision.

Systematic investigations of the dijet invariant mass resolution as a function of the central magnetic field, the calorimeter inner radius and barrel-endcap aspect ratio, calorimeter transverse readout cell area, longitudinal segmentation, material and readout technology are being undertaken, employing a Particle Flow algorithm.

FUTURE DEVELOPMENTS

As part of an ongoing program to truly internationalize the ILC physics and detector simulation effort, a simple and lightweight persistence format was developed, called LCIO [3]. The event data model for the simulation and reconstruction software is being rewritten to target this data model and also to incorporate new features of the Java language which have become available since hep.lcd was first being developed. Experience gained in the use of the existing packages is also being used to refactor much of the software. The improved code will be released as a collection of packages in the org.lcsim namespace [2].

CONCLUSIONS

Designing detectors for the International Linear Collider is an area of active development with many choices, requiring close coupling of design, simulation, and reconstruction. Ease of use and speed of development are essential for physicists conducting ILC studies part-time and in a heterogeneous environment. A fairly complete suite of simulation tools written in Java exists and is being successfully used for these studies.

ACKNOWLEDGEMENTS

I would like to acknowledge and thank all the LCD developers who have contributed to this project, in particular Gary Bower, Ron Cassell, Tony Johnson, Saurav Pathak, Mike Ronan, Nick Sinev, and Wolfgang Walkowiak.

REFERENCES

[1] http://www-sldnt.slac.stanford.edu/jas/Documentation/lcd/

[2] http://lcsim.org

[3] LCIO - A persistency framework and data model for the Linear Collider, Frank Gaede *et al.*, these Proceedings, and http://lcio.desy.de