Java Physics Generator and Analysis Modules

Michael T. Ronan
LBNL, Berkeley, CA 94720, USA

A Java software framework allows modules written in different languages to be used in a Object-Oriented (OO) environment. In this work, Java Native Interfaces (JNI) for Linear Collider (LC) physics event generators are used in defining a common generator interface package. Portable-JNI for TESLA and Asian JLC detector simulation modules have been written for performing comparisons to the American LC detector simulation. Physics and detector Java analysis modules based on prototype HEP class libraries provide high-level OO study tools. Complete physics generation, parallel detector simulations and event analysis for full 500 fb^{-1} simulated data samples are performed in single-pass batch jobs. Java histogram objects files are saved for final presentation using the Java Analysis Studio (JAS). The software architecture, JNI designs and overall performance is presented. Comparisons of American, Asian and European detector simulations of Higgsstrahlung events generated by Pandora, Pythia and Whizard are made.

1. INTRODUCTION

In performing Linear Collider physics and detector studies, one encounters a number of existing Fortran77, and new C++ and Fortran95 software modules for event generation, detector simulation and physics analyis. Stand-alone packages exist for Pandora(C++), Pythia(F77) and Whizard(F95) physics generation, and in some cases have been integrated into separate simulation packages for TESLA SimDet(F77), Asian JLC QuickSim(C++) and American Java Fast Monte Carlo detector studies. Individual physics analyses proceed by adopting one of the generator and detector simulation packages and often are hard to compare to similar analyses done in different environments. A Java software framework [1] allows the use of different event generators, detector simulation packages and analyis modules in the same high level Object-Oriented (OO) environment. An environment which provides modern development tools for Java physics and detector analysis modules using prototype HEP class libraries [2]. The model allows networked graphical interactive applications, or batch single-pass processing of complete physics generation, parallel detector simulations and event analysis for full 500 fb^{-1} simulated data samples. Java histogram objects files can be saved for final presentation using the Java Analysis Studio (JAS) [3].

In this paper, Java Native Interfaces (JNI) for Pandora, Pythia and Whizard Linear Collider physics event generators are described and used in defining a Java framework with a common generator interface package, Sec. 2. Sections 3 and 4 list the generator and detector simulation modules, respectively. A simple comparison of the different physics generators is made for a LC bench mark process in Sec. 5. Detector simulations using the U.S. Fast Monte Carlo, TESLA SimDet and Asian QuickSim packages are compared in Sec. 6. Details of interface implementations for the generators, and for executing SimDet and QuickSim detector simulations and accessing simulated quantities are outlined in Sec. 7. Java software package organization and a brief discussion of the documentation is left for Sec. 8.

2. JAVA ANALYSIS FRAMEWORK

Several prototype Java interfaces, classes and libraries have been written[1] to provide a high level object-oriented physics generation, detector simulation and analysis environment for e^+e^- Linear Collider studies.

2.1. Generator interface

A prototype Java object model has been developed to generalize the interface to physics generators such as Pandora, Pythia and Whizard. The following methods are implemented through Java Native Interfaces (JNI) to the underlying C++, Fortran77 or Fortran95 package:

- setup() - Set up generator configuration using default parameters.
- initialize() - Initialize for selected beams, energy and physics process with given parameters.
- generateEvent() - Generate an event.
- terminate() - Summarize generated data sample.

One additional method is required to provide information about the generator.

- getName() - Return generator name and version number.

Generator specific methods have been added such as

- list(int) - List particles in the event.

The usage of the Java generator interface is outlined in the following excerpt:

Presented at Computing in High Energy and Nuclear Physics, La Jolla, California, March 24 - 28, 2003

```
// Import the Pythia generator classes.
import hep.generator.pythia.*;

// Create a Pythia process.
  pythia = new Pythia();

// Set up a "User-defined" Pythia process
// with a String of parameters settings.
  pythia.give(parameters);

// Initialize the e+e- Center-of-Mass system.
  pythia.init("CMS","e+","e-",Ecm);

// Run Pythia to generate and list events.
  for (int n=1; n<=NEvents; n++) {
    pythia.generateEvent();
    pythia.list(1);
  }
```

Generator process interfaces have been defined for classes which dynamically load predefined process settings, such as PythiaProcess for initializing Pythia. A number of reference process classes have been written, such as **eetoZH** for Higgsstrahlung and **eetottbar** for Top-pair production processes.

2.2. EventGenerator and analysis modules

The generator interface described above is used to provide "EventGenerator" modules for the American LCD framework[2]. A Java package **hep.lcd.generator** is abstracted to introduce different generator modules as alternative "EventSource's":

- Generator - Defines a standard HEP physics event generator module.
- GeneratedEvent - Defines a generated event.
- HEPEvent - Provides a standard /HEPEvt/ implementation of a generated event.

The usage of a LCD EventGenerator module is illustrated by

```
// Import the LCD generator classes for Pythia.
import hep.lcd.generator.pythia.*;

// Create a Pythia generator for given process.
  generator = new PythiaGenerator(processName);

// Run to generate and dump the first event.
  for (int n=1; n<=NEvents; n++) {
    event = generator.generateEvent();
    if (n==1) event.dump();
  }
```

Within the LCD Java framework, event processing and analysis modules are added as "Driver's" or "Processor's" defined in a **hep.lcd.util.driver** package. The following classes outline the use of detector simulation and analysis modules:

```
public class AnlPythiaZHJets implements Driver

  public AnlPythiaZHJets()

    add(detectorSimulation);
    add(analysis);
 or
    add(new MCFast());
    add(new SimDetModule());
    add(new QuickSimModule());
    ...
    add(new PythiaZHJetAnalysis());

class PythiaZHJetAnalysis implements Processor
  void process(LCDEvent event)
    particles = event.get("MCParticles");
    tracks  = event.get("SimDet Tracks");
    objects = event.get("Energy Flow Objects");
    ...
```

Note that the Java Virtual Machine (JVM) allows multiple detector simulations to run in independent computer memory space.

2.3. Analysis classes and tools

The utility of Java physics and analysis class libraries is illustrated in the following excerpts for the Particle physics class

```
int PDGID = particle.getType().getPDGID();
double[] PV = particle.getMomentum();
  double pT = Math.sqrt(PV[0]*PV[0]+PV[1]*PV[1]);
  double tanTh = pT/PV[2];
```

and the Histogram analysis class

```
histogram("Mass").fill(particle.getMass());
```

Java framework applications can be built using standard HEP analysis tools and run through convenient Java packages so that

- gmake - Compiles Java and native code then creates a shared object library.
- java -mx64M GenWhizard ... - Loads Java and all shared object libraries, executes Whizard and writes a xxx.javahist unbinned histogram file.

After completing several jobs, the Java Analysis Studio [3] can be used to combine histograms and make presentation plots.

```
jas &
  Open files: xxx.javahist yyy.javahist ...
  Overlay and format plots
  Save as PostScript files
```

3. MONTE CARLO EVENT GENERATORS

Three representative LC physics generators have been used in this development: Pandora V2.2, Pythia V6.2 and Whizard V1.22. Pythia which has been extensively tested by many experiments was used to provide checks on specific processes. Pandora and Whizard are new developments which also offer complete event generation capability.

3.1. Pandora Monte Carlo

The Pandora V2.2 [4] Monte Carlo provides simulations of selected two- and three-body processes and some illustrative beyond the Standard Model processes. Initial state radiation, beam polarization, and final state spin correlations are included for all processes. In stand-alone applications Pandora is used with the Pandora-Pythia V3.2 [5] interface package to generate final state events. A "straightforward" interface from the Java language to underlying Pandora C++ methods, Sec. 7, is provided in a **hep.generator.pythia** Java package, Sec. 8.

3.2. Pythia Monte Carlo

The Pythia V6.2 [6] Monte Carlo program allows generation of high-energy physics events including hard and soft interactions, parton distributions, initial and final state parton showers, multiple interactions, fragmentation and decay. A Java-C-Fortran77 interface described below provides full Pythia functionality within the Java environment.

3.3. Whizard Monte Carlo

The Whizard V1.22 [7] Monte Carlo system is designed for the efficient calculation of multi-particle scattering cross sections and simulated event samples. Tree-level matrix elements are generated automatically for arbitrary partonic processes by calling external programs (O'MEGA, MADGRAPH and CompHEP). In this development, Java-C-Fortran95 interfaces to new Whizard procedures for initialization, phase space integration, event generation and summary was written to allow easy portability and compiler independence.

4. DETECTOR SIMULATION MODELS

Models of the expected LC detector response to 500 GeV e^+e^- annihilation events are being developed. Preliminary versions used in this study are described below, and compared in Sec. 6. The detailed Java interface implementations are outlined in Sec. 7.

4.1. LCD Fast Detector Simulation

The American LCD V1.4 detector simulation includes charged particle momentum smearing based on detailed error estimates, gaussian energy smearing for photons and neutral hadrons, and acceptance and energy threshold requirements. It doesn't account for confusion in reconstructing calorimeter energy deposits resulting in perfect energy flow separation of charged and neutral components of jets.

4.2. TESLA SimDet Detector Simulation

The European SimDet V4.0 detector simulation includes parameterized charged and neutral energy smearing based on full (**Brahms**) Monte Carlo simulations, acceptance requirements, and a realistic energy flow algorithm.

4.3. JLC QuickSim Detector Simulation

The Asian QuickSim detector simulation includes the generation of tracker hits followed by track and vertex reconstruction. Simulated calorimeter cell energy depositions are formed based on detailed detector simulations. Detailed cluster finding and track-cluster matching are performed before writing out combined track information..

5. PHYSICS GENERATOR COMPARISONS

The Pandora, Pythia and Whizard Linear Collider physics event generators are compared in Fig 1 for LCD Fast Monte Carlo detector simulations of Higgsstrahlung, $e^+e^-\to Z\ Higgs$, events. In these studies all generators were run at E_{CM} = 500 GeV, and the Higgs mass was set to 115 GeV. Pythia hadronization was used following all parton-level event generation processes. For Pandora, the NLC500 machine parameters were selected in simulating the beamstrahlung energy spectrum. While both Pythia and Whizard event generators used Circe beamstrahlung simulation of the TESLA machine. These different machine settings are not significant in this comparison.

Overall, the generators were found to be in good agreement. In this study, Whizard V1.22 generated events did not include the natural Z width as seen in the jet-jet mass distributions shown in Fig 1d; a Whizard deficiency that has been corrected following this work.

In related studies [13], the Java framework was used to compare the simulation of background processes for

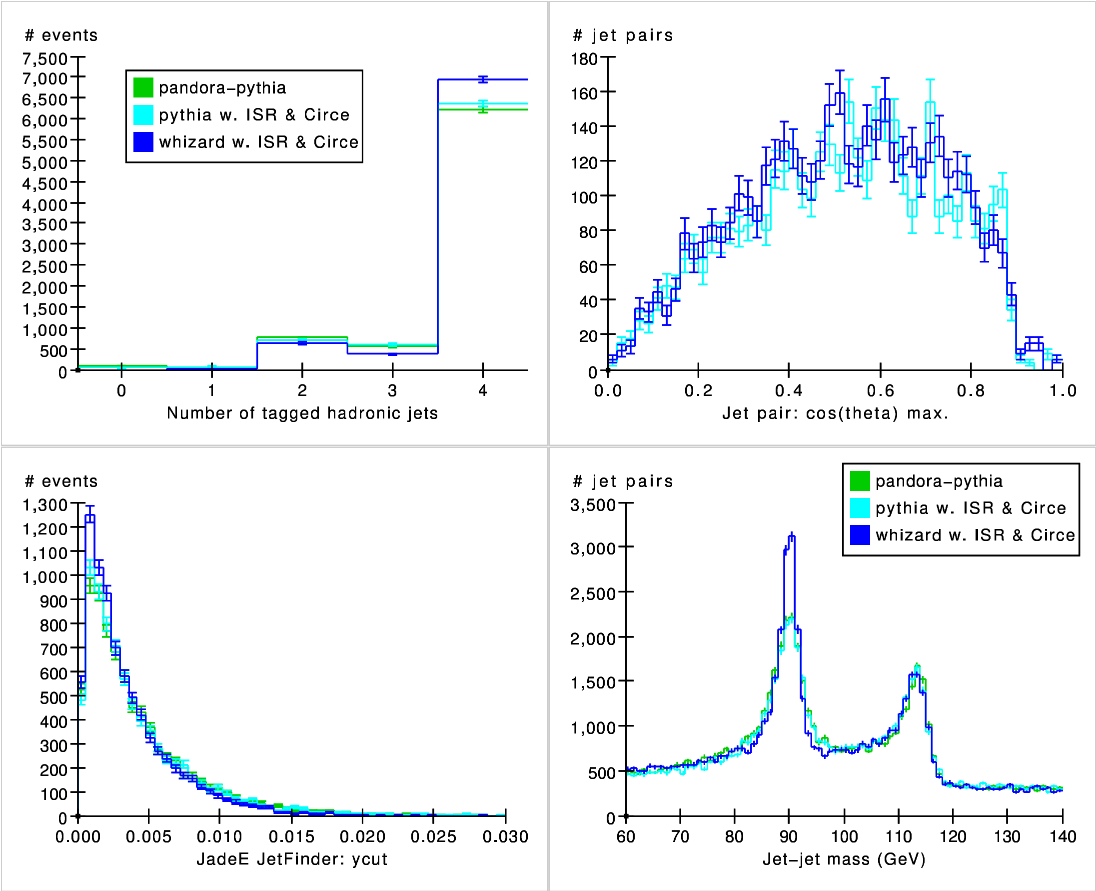

Figure 1: LCD Fast Monte Carlo Particle jet distributions comparing Pandora, Pythia and Whizard generated Higgstrahlung events: a.) number of "correctly" reconstructed jets, b.) angular distribution ($\cos(\theta)_{Max}$) of jets, c.) jet finder final "ycut" value, d.) jet-jet mass distributions showing Z and Higgs hadronic decay signals and the combinatorial background.

WW, ZZ and $q\overline{q}$ production from Pandora with explicit final state simulations in Whizard. These comparisons showed discrepancies due to the more precise treatment of polarization in Pandora for exclusive processes. The simulation of inclusive, interfering processes in Whizard was found to be the most complete, and the Whizard MadGraph amplitude calculation provided handling of parton color flow information.

6. DETECTOR COMPARISONS

In a e^+e^- Linear Collider the Higgs can be reconstructed through hadronic decays[13] even in multi-jet final states. The jet-jet mass distributions for the American LCD Fast Monte Carlo (FMC), TESLA SimDet and JLC QuickSim detector simulations of Higgstrahlung signal-only events with the Z and Higgs decaying hadronically are shown in Fig 2. In these current simulations, the LCD FMC jet-jet mass resolution is significantly better since it assumes "perfect" energy flow. More-realistic TESLA SimDet and JLC QuickSim detector simulations give comparable jet energy resolutions but different mean reconstructed jet-jet masses. Both these simulation models may be improved with the development of new techniques to resolve charged and neutral calorimeter clusters, and to correct for the loss of low energy tracks and clusters.

The overall performance of the Java framework, event generators, detector simulations and jet analyis was quite acceptable for full 500 fb^{-1} production runs. Each generator was run separately with all three detector simulations being performed for the same generated event. The Java framework itself and event generation and fragmentation typically took much less than 100 msec per event, with the longest times for multi-body processes. The detailed native SimDet and QuickSim detector simulations each took about 300-500 msec per event, depending on the type of events. The Java jet-finding which included multiple passes to merge jets and the subsequent analysis took roughly 100 msec per event for each type of particle jet being found, e.g. for Monte Carlo

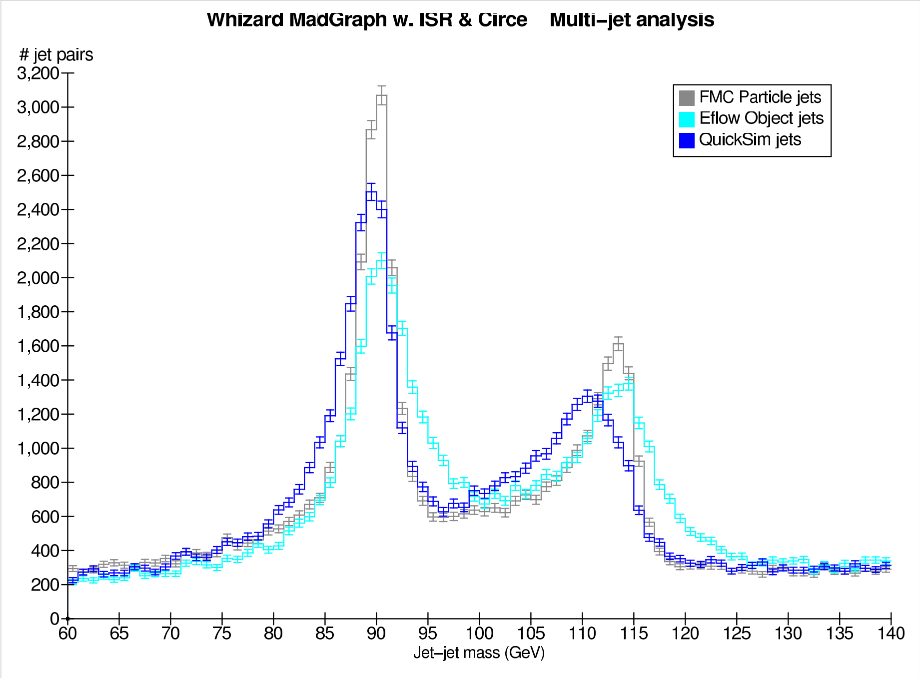

Figure 2: Direct reconstruction of Z and Higgs through hadronic decays is shown for Higgsstrahlung signal events only. Jet-jet mass distributions for U.S. LCD Fast Monte Carlo (FMC), TESLA SimDet and JLC QuickSim detector simulations are reconstructed for Whizard-MadGraph Monte Carlo events including ISR and Circe beamstrahlung effects.

particles, for jets found from reconstructable tracks and clusters, or for Energy Flow object jets. The typical overall event generation, multiple detector simulation, and jet analysis took about 1-1.5 seconds per event.

Batch jobs were labeled by the generator name and parameters that were set, and histogram folders were created for the different detector simulations. In the analysis of results, JAS would open separate output histogram files in different folders and use the folder names to automatically generate plot legends.

7. JAVA NATIVE INTERFACE (JNI) IMPLEMENTATIONS

The physics generator, physics event, detector simulation and software framework implementations[1] for all developed software modules are outlined in Table I.

In generating events through the Pandora-JNI, a Java-C++ interface is used to allow the setting of different beam configurations, and to create physics processes to be added to Pandora's Box. Pandora processes parameters, such as the Higgs mass, are required for the Java interface construction. Once a Pandora process has been created no Java object to C++ object interface is needed. Pandora generates primary partons which are accessed through the **HEPEvt-JNI** and passed to Pythia for fragmentation.

The Pythia-JNI allows for independent event generation and fragmentation processing. The interface can be used to pass configurations through pygive or to use Java classes such as “eetoZH” to configure Pythia for simulating the Higgsstrahlung process. The Pythia interface will be quickly replaced when new C++ versions of Pythia are made available and tested.

The Whizard interfacing to Fortran95 was simplified by creating a few simple Fortran95 modules. Whizard is configured by reading a “whizard.in” data file. Future developments of the Whizard Java interface are planned.

Java interfaces to the HEPEvt data structure and StdHEP output were written for multi-purpose use. After a generator fills HEPEvt, framework modules read the information and create event objects. These objects are later downloaded into HEPEvt structures that are accessed by fragmentation processes or by the different detector simulations. The design allows any combination of physics generator, fragmentation process or detector simulation. User analysis interfaces have also been written to allow one to use existing analysis packages.

[1] http://www-lc.lbl.gov/software/docs/

An interface to Circe [8] is used to handle the initialization of beamstrahlung simulation parameters from the Java framework for the Pythia and Whizard generators. The Tauola [9] τ-decay package was used to handle polarization effects. A simple Java interface was written to allow testing of individual Tau decay modes.

The detector simulation packages have simple interfaces to set up, initialize and execute the simulations. Care was taken to pass events into the packages through separate HEPEvt interfaces. The simulated detector responses for SimDet and QuickSim are accessed from underlying storage into Java track and cluster classes to form reconstructed objects that are added to the LCD event for subsequent analysis. The SimDet and QuickSim interfaces follow a similar design model which could be generalized for other applications.

8. JAVA SOFTWARE PACKAGE AND API ORGANIZATION

The interface software and LCD modules have been organized into several Java packages or class libraries. The interfaces to the imported physics generator and detector simulation modules are in the HEP packages listed in Table II. The LCD framework modules are organized in the packages listed in Table III. The LCD modules provide reference implementations using the more generic interfaces to the HEP packages.

The LCD event generator and detector simulation modules are designed for simple use, see Sec. 2.2. The Generator, Fragmentation and GeneratedEvent interfaces allow the design of abstract models for sourcing events to be based to the different detector simulation modules. Simulated reconstructed objects implement a basic Particle class allowing the development of abstract analysis methods to reconstruct event quantities from different detector models.

Special utility packages, **pandorapythia** and **whizardpythia**, based on the Pandora-Pythia [5] software model for handling the physics interfacing from Pandora and Whizard generated parton color flow information to Pythia hadronization, are located in the **hep.lcd.util** package.

Software Application Programming Interface (API) documentation on all Java packages is available on the web[2]. Class inheritance and method signature documentation is automatically generated. Some documentation exists for most classes while comments on many self-explanatory methods are missing.

[2]http://www-lc.lbl.gov/software/java/api/

In building stand-alone event generation, detector simulation and physics analysis packages, the Java classes are compiled according to the following rule

```
# For a collection of Java source files
 JAVAFILES := $(wildcard hep/.../*.java)

# the corresponding class files are obtained
 CLASSFILES:= $(patsubst %.java, \
                %.class,$(JAVAFILES))

# and compiled with
%.class: %.java
   @echo "Compiling" $(*F)
   @javac $(JAVACFLAGS) $<
```

Header files for the JNI implementations are automatically generated from the Java source code using a utility

```
$(HeaderFile): $(JNISource)
   @if [ -f $(TargetFile) ]; \
      then rm $(TargetFile); fi;
   @javah -jni -d $(TargetDir) $(Source)
```

Shared object libraries and native code are compiled following

```
   gcc -shared -I$(JavaJDK)/include \
     -I$(JavaJDK)/include/linux \
     ...
     -L/usr/lib/gcc-lib/i386-redhat-linux/2.96 -lg2c \
     -L/usr/lib -lc \
     -o $(SharedLib)
```

where JavaJDK locates the Java development kit (JDK), the include files are used by the JNI implementations, and the library list usually includes Linux specific libraries and general libraries that normally would be loaded into an executable image. The shared object library is located by the LD_LIBRARY_PATH environment variable and loaded into the JavaVM following

```
public class Pythia
{
  ...

  /** Static initializer loads Pythia library. */
  static {
    String libname = "pythia";
    System.out.println(" Load lib"+libname+".so");
    System.loadLibrary(libname);
  }
}
```

Acknowledgments

I thank Tony Johnson for many useful discussions on Java object modeling and for presenting this paper at the conference. Tony provided an initial interface to the HEPEvt data structure. Michael Peskin and Tim Barklow provided guidance on interfacing to Pandora and on passing the parton color flow information to Pythia for fragmenting high-energy e^+e^- jets. The Pythia generator interface was extended to include pygive & pyinit in consultation w/ Stephen Mrenna. Tim Barklow, Wolfgang Kilian and Thorsten Ohl helped in defining the Whizard interface and in implementing simple Fortran95 procedures. I thank Heinz-Jürgen Schreiber and Akiya Miyamoto for reviewing the detailed interfacing to the TESLA SimDet and JLC QuickSim detector simulation packages. The Java physics generator and detector simulation modules described in this paper are used within the Java based software framework under development by the American LCD software group[1–3].

Work supported by the U.S. Department of Energy contract DE-AC03-76SF00098.

References

[1] All modules and tools used in this study are available from **http://www.lbl.gov/~ronan/docs/lcstudies**.

[2] Norman Graf et al, Proceedings of the International Workshop on Linear Colliders, Jeju Island, Korea, August 26-30, 2002 and references therein.

[3] Tony Johnson, SLAC, *Java Analysis Studio*, **http://jas.freehep.org**.

[4] PANDORA V2.2, **http://www-sldnt.slac.stanford.edu/nld/new/Docs/Generators/PANDORA.htm**.

[5] PANDORA-PYTHIA V3.2, **http://www-sldnt.slac.stanford.edu/nld/new/Docs/Generators/PANDORA_PYTHIA.htm**

[6] PYTHIA V6.2, **http://www.thep.lu.se/~torbjorn/Pythia.html**.

[7] WHIZARD V1.22, **http://www-ttp.physik.uni-karlsruhe.de/Progdata/whizard/**.

[8] CIRCE V7.0, **http://heplix.ikp.physik.tu-darmstadt.de/lc/beam.html**.

[9] TAUOLA V2.6, **http://hpjmiady.ifj.edu.pl/programs/node6.html**.

[10] SIMDET V4.0, **http://www-zeuthen.desy.de/linear_collider/**.

[11] JSF V2002-1.0, **http://www-jlc.kek.jp/subg/offl/jsf/docs/usersguide/html/**.

[12] QUICKSIM is part of LCLIB, **http://www-jlc.kek.jp/subg/offl/lclib/index.html**.

[13] Michael T. Ronan, *Multi-Jet Higgsstrahlung Event Analysis"*, LCWS 2002, ibid.

Table I Java Native Interface (JNI) Implementations

Package	**Java method signature**	**Native invocation**
Physics Generators		
Pandora		**C++ → C++**
Create a new beam.	native newBeam(Eb,pol,in,out);	b1 = new ebeam(Eb,pol,in,out);
Create a new process.	native newProcess(name,param);	P = new processName(param);
Add process to Pandora's box.	native add(PandoraProcess process);	Box→add(*P);
Generate an event.	native nextEvent();	LEvent LE = Box→getEvent();
Pythia		**C → F77**
Give a parameter list.	give(String s) { give(s,s.length); }	
	native give(String s,int len);	pygive_(string,length);
Initialize CMS frame.	native init(String frame, ...);	pyinit_(frame,b1,b2,&E,i,j,k);
Generate an event.	native exec();	pyexec_();
Whizard		**C → F95**
Read input data file.	native readInput();	whizard_read_input_();
Do phase space integration.	native integrate();	whizard_integrate_();
Generate an event.	native event();	whizard_event_();
Circe		**C → F77**
Initialize beamstrahlung simulation.	native init(double X1M, ...);	circes_(&x1m, ...);
Tauola		**C → F77**
Set decay mode of Tau+ or Tau-.	native setDecayMode(int i, int mode);	if (i==1) jaki_.jak1 = mode;
		else if (i==2) jaki_.jak2 = mode;
Physics Events		
HEPEvt		**C → F77**
Get momentum vector component.	native getPHEP(int i, int j);	return hepevt_.phep[i][j];
StdHEP		**C → F77**
Initialize output file.	native initialize(String outfile, ...);	stdhinit_(outfile,&ngen);
Write an event.	native writeEvt();	stdhwrit_(&nevt);
Detector Simulations		
SimDet		**C → F77**
Initialize detector model.	native init();	siinit_();
Set detector parameters.	native detr();	sidetr_();
Simulate detector response	native exec();	siexec_();
QuickSim		**C++ → C++/F77**
Create a JSF package. .	native newJSF();	jsf = new JSFSteer(...);
Set up QuickSim process.	native setup();	sim = new JSFQuickSim(...);
Set type of generator.	native setGenerator(type);	sim→setGenerator(type);
Simulate detector response	native exec();	result = jsf→Process(eventNum);
Software Frameworks		
JSF		**C++ → C++**
Create a new JSF framework.	native newJSF();	jsf = new JSFSteer(...);
Add a LCFull process.	native addLCFull();	full = new JSFLCFULL(...);
Set up defaults:	native setup();	sim = new JSFQuickSim(...);
Set type of generator.	native setGenerator(type);	sim→setGenerator(type);
Add an analysis process.	native addAnalysis();	anal = new MyAnalysis();
...		

Table II HEP Java software package

Project	Java classes	Java package name
HEP Generator packages		hep.generator
Pandora	Pandora, PandorasBox, Beam, PandoraProcess, eetoee, eetoZHiggs, ...	hep.generator.pandora
Pythia	Pythia, PythiaProcess, eetobbbar, eetoZH, ...	hep.generator.pythia
Whizard		hep.generator.whizard
also		
Circe	Circe	hep.generator.circe
Tauola	Tauola	hep.generator.tauola
HEP Event package		
HEP	HEPEvt, StdHEP	hep.event
TESLA packages		
SimDet detector simulation	SimDet, SimDetCHAPAEntry, SimDetCLUSEntry, SimDetEFLOWEntry, ...	hep.tesla.simdet
JLC		
JSF Software framework	JSFSoftwareFramework, JSFGenerator, JSFGeneratorParticle, JSFPythiaGenerator	hep.jlc.jsf
QuickSim detector simulation	JSFPackage, QuickSim, QuickSimParticleEntry, QuickSimTrackEntry, QuickSimCalHitEntry, ...	hep.jlc.quicksim

Table III LCD Java software package organization

Package	**Java classes**	**Java package name**
Generator packages		
Physics generators	Generator, Fragmentation GeneratedEvent, HEPEvent	hep.lcd.generator
Pandora generator	PandoraGenerator, PandoraEvent, LCDPandoraEvent	hep.lcd.generator.pandora
Pythia generator	PythiaGenerator, PythiaFragmentation, PythiaEvent, LCDPythiaEvent	hep.lcd.generator.pythia
Whizard generator	WhizardGenerator, WhizardEvent, LCDWhizardEvent	hep.lcd.generator.whizard
Monte Carlo packages		hep.lcd.mc
U.S. Fast MC Detector Simulation	MCFast, ReconTrack, ReconCluster, ...	hep.lcd.mc.fast
TESLA SimDet Detector Simulation	SimDetModule, SimDetDriver, SimDetAdapter, SimDetParticle, SimDetTrackList, SimDetClusterList, ...	hep.lcd.mc.simdet
JLC QuickSim Detector Simulation	QuickSimModule, QuickSimDriver, QuickSimAdapter, QuickSimParticle, QuickSimTrackList, QuickSimClusterList, ...	hep.lcd.mc.quicksim
Software framework packages		hep.lcd.framework
JLC Software Framework	JLCSoftwareFrameworkDriver, JSFEventGenerator, JSFGeneratorParticle, ...	hep.lcd.framework.jsf