MONTE CARLO EVENT GENERATION IN A MULTILANGUAGE, MULTIPLATFORM ENVIRONMENT

Norman Graf **Tony Johnson**

Stanford Linear Accelerator Center

Abstract:

We discuss techniques used to access legacy event generators from modern simulation environments. Coding to a standard interface and use of shared object libraries enables runtime selection of generators, and allows for extension of the suite of available generators without having to rewrite core code.

International Linear Collider (ILC)

Designed to exploit the physics discovery potential of $e^+ e^-$ collisions at $\sqrt{s} \sim 1$TeV.

Will perform precision measurements of complex final states and require:

- ❖Exceptional momentum resolution
- ❖Excellent vertexing capabilities
- ❖"Particle Flow" calorimetry
- ❖Hermeticity

Require access to many different event generators to fully investigate interplay of machine, physics and detectors.

Problem Statement

❖ Physics and detector simulations for the International Linear Collider are being conducted by an amorphous and heterogeneous group of high energy physicists, working mostly part-time on this project.

❖ Simulation software needs to be lightweight, yet flexible and performant over a wide variety of development platforms.

❖HEP community has mostly completed it transition to modern, object-oriented programming languages such as C++ and Java

e.g. GEANT4, ROOT, JAS, …

One exception is event generators. Most event generators producing unweighted events with stable final state particles appropriate for detector response simulations are either written in FIRTRAN or depend on FORTRAN-based libraries for fragmentation, e.g

PYTHIA, ISAJET, HERWIG

pandora-pythia

Interfacing Legacy Event Generators

✓Most generators can target the HEPEVT common block.

✓stdhep provides a binary persistence binding

×stdhep not well supported on all platforms, somewhat tricky to implement for casual users.

❖Could interface with generators simply through persistent output:

FORTRAN program → stdhep file → user app.

×Disk storage can be prohibitive in large statistics fast-simulation physics analyses.
×End users need to know idiosyncrasies of each package, i.e. no standard main programs or runtime input commands.

❖ Alternative is to abstract out a common generation interface and provide standard implementations.

A pure Java Solution

❖Providing a simulation and reconstruction framework written in Java has proven to be well matched to this environment.

✓ Object Oriented and easy to use.

✓ Many support libraries.

✓ Platform independent.

× Few event generators are written in Java!

✓Do have a "diagnostic" generator for simple events.

A Multilanguage Solution

❖Use Java to provide a consistent platform-independent interface.

❖Use Java Native Interface (JNI) to call FORTRAN event generators via C++.

❖Use shared-object (.so) or dynamic link library to delay binding and provide runtime felxibility.

Encapsulation

Philosophy is to push as much as possible onto the native event generators. Control is through ASCII files which are parsed by FORTRAN routines. These communicate with the COMMON blocks in the event generator code to set parameters.
Can select processes, beamstrahlung, decay channels, etc. at run-time by editing text file.
Output is stdhep files

Java calls C++ class through JNI with minimal interface:

```
public native void initialize();
public native void nextEvent(
      int[] nev,
      int[] isthep, int[] idhep,

      int[] jmohep, int[] jdahep,
      double[] phep, double[] vhep);
public native void finish();
```

C++ communicates with FORTRAN:

```
extern "C" void initialize_();
extern "C" void nextevent_();

extern "C" void finish_();

typedef struct // HEPEVT common block
{
 int nevhep;          // event number
 int nhep;            // number of entries
 int isthep[4000];    // status code
 int idhep[4000];     // PDG particle id
 int jmohep[4000][2]; // pos'n of 1st, 2nd mother
 int jdahep[4000][2]; // pos'n of 1st, last daughter
 double phep[4000][5]; // 4-mom, mass
 double vhep[4000][4]; // vertex position & time
} hepevtcommon;
```

✓FORTRAN code simply fills HEPEVT common block for each event.
×This code has to be written by someone with expertise in the generator, but then can be used by anyone.
•Control is through initialize_ call, and is generator-specific.

Runtime Control

✓Interact natively with event generators

ISAJET has well-defined set of control "cards"

Input file same as for FORTRAN job.

PYTHIA has command-parsing capability

Simply pass these commands to PYGIVE

HERWIG has neither

abstract out a "reasonable" set of parameters.

Similarly for other generators

×Needs input from an "expert" in the generator.

✓Interaction from Java/C++ is only through initialize() method.

✓All .dll or .so libraries respect same interface, so can dynamically select and load at runtime

>java EvtGen libToLoad

✓Catalog of available generators can be expanded in the future, without users having to modify any of their existing code. Simply distribute new .so or .dll!

```
import hep.analysis.EventGenerator;
import hep.analysis.EventData;
import hep.analysis.EndOfDataException;
import hep.io.stdhep.adapter.StdhepAdapter;
/**
 * LCD interface to stdhep event generators.
 * @author Norman Graf
 */
public class StdhepEventGenerator extends EventGenerator
{
  StdhepAdapter _stdadapter;
  private EvtGen _eventgen;

 /**
  * Constructor loads native library
  */
  public StdhepEventGenerator(String generatorName)
  {
    _stdadapter = new StdhepAdapter();
    System.loadLibrary(generatorName+"evtgen");
    _eventgen = new EvtGen(); // Constructor calls initialize()
  }

  /**
  * Generate a single event
  */
  public EventData generateEvent() throws EndOfDataException
  {
    _eventgen.nextEvent();
    return _stdadapter.convert(_eventgen.genStdhepEvent());
  }
  /**
  * Finish up
  */
  public void afterLastEvent()
  {
    _eventgen.finish();
  }
}
```

Summary

❖Use of Java as the user interface and JNI to connect to legacy code has proven to be a useful solution.

❖Definition of a minimal interface allows generators to be selected at runtime without code modification and allows smooth future upgrades.