Event Generators

Norman Graf
(SLAC)

May 20, 2003

Problem Statement

- HEP community has mostly completed its transition to modern programming technologies (object-oriented, C++, Java).
  - GEANT4, ROOT, JAS, …
- One exception is event generators which remain mostly FORTRAN based.
- Need to accommodate legacy packages
  - e.g. Pythia, Herwig, Isajet

Interfacing I

- Could interface with generators simply through persistent output.
  - FORTRAN program $\rightarrow$ stdhep file $\rightarrow$ user app.
- Requires users to gain proficiency in setting up and implementing each generator.
  - Different requirements, interfaces, etc.
- Requires FORTRAN compiler!
- Disk storage can be prohibitive in large-statistics fast-simulation physics analyses.

Interfacing II

- Alternative is to abstract out a common interface & provide standard implementations.
- HEPEVT common block is an accepted standard for event generators.
- STDHEP provides common persistent format.
- Java provides a consistent platform-independent interface.
- Java UI/API $\rightarrow$ C++ $\rightarrow$ FORTRAN (.dll or .so)

Legacy Physics Generators

- 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.

Physics Generators

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];     // position of first, second mother
 int jdahep[4000][2];     // position of first, last daughter
 double phep[4000][5];    // 4-momemtum, mass (GeV)
 double vhep[4000][4];    // vertex, production time in mm
} 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.

Physics Generators

- Any generator producing HEPEVT-format output can be used as input to LCD simulations.
- Provide precompiled versions of
  - PYTHIA 6.2
  - ISAJET 7.48
  - HERWIG 6.5
- Interfaced to CIRCE (beamstrahlung) and TAUOLA (τ decays) and controlled at run-time by ASCII files.

Runtime control

- Interact natively with event generators
  - ISAJET has well-defined set of control "cards"
    - Input file is same as would be used for FORTRAN job.
  - PYTHIA has command-parsing capability
    - Simply pass these commands to PYGIVE
  - HERWIG has neither
    - abstract out a "reasonable" set of parameters which user can modify.
- Interaction from Java/C++ is only through initialize() method.

Runtime execution

- 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.
- However, only one generator can be used at a time.

Alternate approaches

- Mike Ronan has developed a set of explicit interfaces to various event generators which allow different generators to be run at the same time.
  - Very useful for comparisons
- e.g.

```
Pythia pythia = new Pythia();
pythia.give(parameters);
pythia.init("CMS","e+","e-",Ecm);
```

Diagnostic Generator

- Often convenient to analyze simple events over which user has complete control.
  - Single particles for tracking (momentum) and calorimeter (energy) resolution studies.
  - Resonances
    - π^0 to study photon separation
  - Few-particle events with well-defined properties
    - Fixed impact parameter or decay point for vertexing
    - Multiple tracks to investigate two-track separation
    - charged+neutral track to study energy flow algorithms

Diagnostic Generator II

- Implemented interface to PYTHIA allowing runtime control of arbitrarily complex user-defined events.
  - Number and type of particles
  - Energy, θ, φ, x, y, z distributions
  - Repeat as necessary
  - Arbitrary number of PYTHIA commands to control final state decays.

Event Generation with LCDRoot

- Implements PYTHIA via TPythia class
- Set up processes and event control through method calls at runtime via CINT.
  - methods similar to PYTHIA calls
- Can output events in stdhep binary format, or as root file.
- Can also simply loop through events using fast simulator.
- Interfaced to pandora-pythia (preferred mode)

Event Generation with LCDRoot

```
TPythia6 pyth6;
 pyth6.SetPMAS(6,1,175.);
 // Select gamma*/Z0 production process
 pyth6.SetMSEL(0);
 pyth6.SetMSUB(1,1);
 // Initialize process
 pyth6.Pyinit("CMS","E-","E+",ECM);
 for (iEvent = 0; iEvent < nEvent; iEvent++) {
    pyth6.GenerateEvent();
 }
```

LCDMcDiag

- Generate single particles (e,μ,π,n,p,K)
- Specify momentum, cos(theta), phi

```
LCDMCDiag mc(fname);
mc.Set_Momentum(10.0, 0.5, TMath::Pi()/2);
mc.Set_Type(211);
mc.Set_Particles(1);
mc.Set_Seed(0);
for(Int_t i=0; i < nEvent; i++){ mc.doit(); }
```

Summary

- Event generators are most important class of legacy code, and require special handling in multi-language, multi-platform environments.
- Several techniques exist to allow users to generate their own event samples.
- We are following developments of generators (e.g. PYTHIA++) and event record formats (e.g. HepMC) as replacements.