ODBMS for LHC detector simulation

Makoto ASAI

GEANT4 (CERN RD44) Collaboration
Hiroshima Institute of Technology
2-1-1 Miyake, Saeki-ku, Hiroshima, 731-51, Japan

One of the new and powerful features of the GEANT4 detector simulation toolkit is the newly designed "Hits + Digitization" category. Using this, the user can easily simulate responses of his/her own detector setup. On the other hand, it is well known that a huge number of simulated events are essential to large scale high energy experiments. Simulated detector responses are one candidate for the heavy use of an ODBMS. Following the first attempt to merge the efforts of CERN RD44 and CERN RD45, we report on the use and performance of ODBMS persistency for LHC detector simulation. The main topics are (1) design of persistent "hit" and "trajectory" classes, (2) use of the ODBMS for "hits" simulated data, and (3) the benchmark performance.

Key words: GEANT4; object oriented; detector simulation; toolkit; LHC; object persistency; ODBMS; Objectivity/DB;

1 Introduction

It is well known that a huge number of simulated events are essential to large scale high energy experiments. Simulated detector responses are one candidate for the heavy use of ODBMS. Also, trajectories of simulated particles should be recorded and associated with detector responses to verify the reconstruction/analysis programs and the triggering logic of the experiment.

The product of CERN RD44 (GEANT4, an object-oriented toolkit for simulation in HEP)[1] is being released as its alpha version. We have reached the stage of simulating a realistic detector of the scale of LHC experiments with event input from commonly used physics generators. Fig.1 shows hits on the ATLAS pixel and silicon detectors simulated with a Higgs event generated by PYTHIA[2].

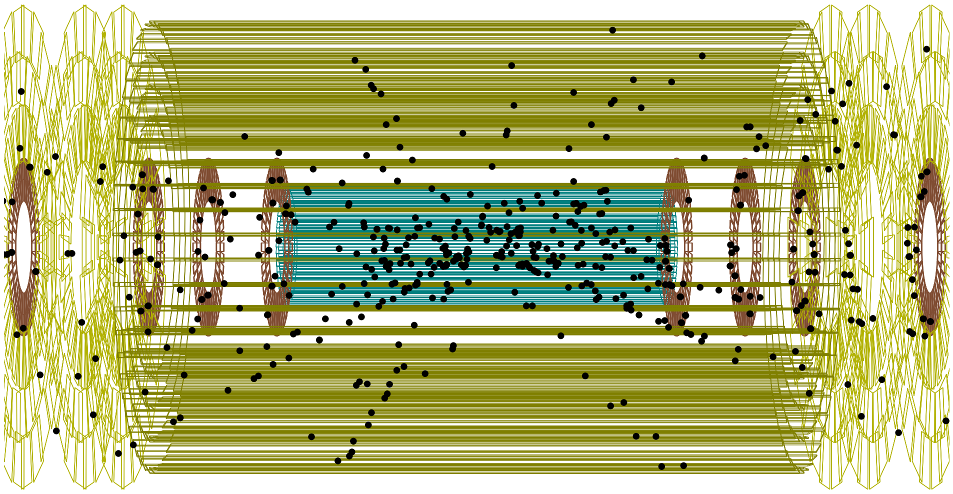

Fig. 1. Simulated hits on ATLAS pixel and silicon detectors[4]

On the other hand, CERN RD45 (A persistent object manager for HEP)[3] has already provided us with its first interface classes to Objectivity/DB. Following the first attempt to merge the efforts of CERN RD44 and CERN RD45, the use and performance of ODBMS persistency for LHC detector simulation is being investigated using "hit" and "trajectory" objects.

2 Hit related classes in GEANT4

The class diagram of "hit" related classes is shown in Fig.2. Persistent classes are written in *bold-italic* case. A `G4VSensitiveDetector` class object will construct `G4VHit` class object(s) when a track comes into its "sensitive" volume. All the information needed to generate a hit object are extracted from a `G4Step` class object. Hit objects are stored in a `G4VHitsCollection` class object, which has a *ValOrderedVector*[1] of the particular hit type. All `G4VHitsCollection` objects associated to an event are stored by their references in a *PtrOrderedVector* of `G4HCofThisEvent` class objects. `G4HCofThisEvent` has a one-to-one association with a `G4Event` class object which represents a particular event.

As the contents of a "hit" class object depends on the detector type, they are user dependent, and the user has to derive his/her own classes of sensitive detectors and hits from base classes `G4VSensitiveDetector` and `G4VHit`, respectively. Note that the `G4VHitsCollection` class object has a *ValOrderedVector* to minimize the disk space overhead; thus the user also needs to derive

[1] *ValOrderedVector* and *PtrOrderedVector* are concepts from RogueWave *Tools.h++* template library, which is extensively used in GEANT4.

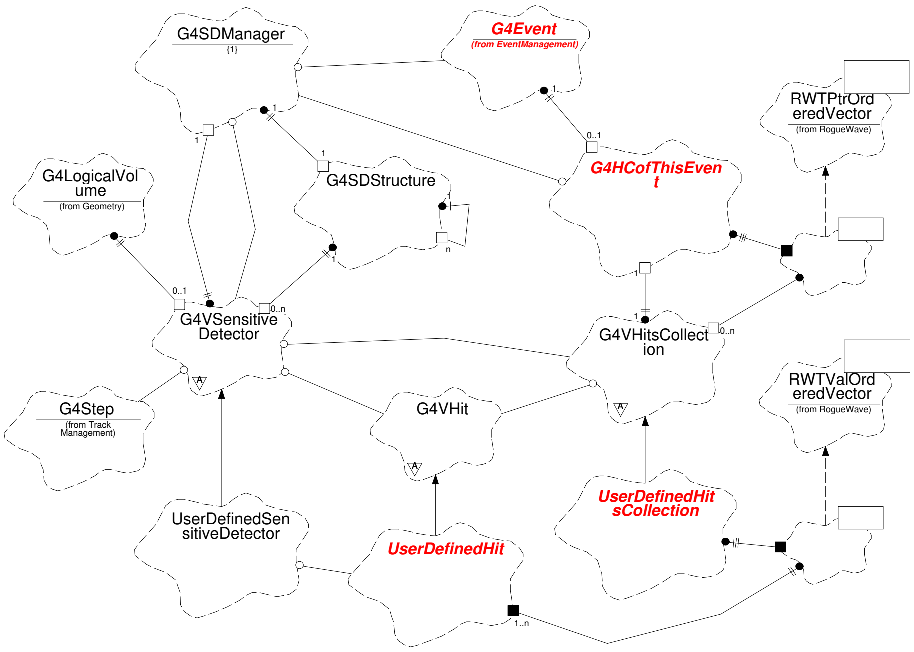

Fig. 2. Class diagram of hit related classes

his/her own collection classes from `G4VHitsCollection`, and instantiate them with the derived hit classes.

3 Results of performance tests using hit objects

Persistence for two types of hit object has been introduced using Objectivity/DB. In the case of tracker-type hits, each object is accessed only once at construction time, but the number of objects per event is unpredictable and huge. On the other hand, in the case of calorimeter-type hits, objects are accessed many times as the energy deposition is accumulated, but the number of objects is usually much smaller than the number of calorimeter cells.

Two different implementations have been tested for each of above two types:

(i) storing each hit object as a separate persistent object in the data base;
(ii) storing all hits of one kind in a persistent *VArray*.

Fig.3 and Fig.4 are the header files of a user-defined hit class for transient and persistent implementations, respectively. Modifications required to make a class persistent are written in **bold** case.

In both cases, as can be seen from the Tables 1 and 2, the overhead introduced by making the objects persistent is very small. In the case of tracker hits, two collections which contain 1900 and 1700 hits respectively are created and each

```
#ifndef MyCalorimeterHit_h
#define MyCalorimeterHit_h 1

#include "G4VHit.hh"
#include "G4Allocator.hh"

class MyCalorimeterHit :
  public G4VHit
{
  public:
      MyCalorimeterHit();
      ~MyCalorimeterHit();
      MyCalorimeterHit(const MyCalorimeterHit &right);
      const MyCalorimeterHit& operator=(const MyCalorimeterHit &right);
      int operator==(const MyCalorimeterHit &right) const;

      inline void *operator new(size_t);
      inline void operator delete(void *aHit);

      void draw();
      void print();

  private:
      G4double edep;
      G4ThreeVector pos;

  public:
      // Some set/get methods here...

};

extern G4Allocator<MyCalorimeterHit> MyCalorimeterHitAllocator;

#endif
```

Fig. 3. A transient hit class

```
#ifndef MyCalorimeterHit_h
#define MyCalorimeterHit_h 1

#include "G4VHit.hh"
#include "Hep.h"

class MyCalorimeterHit :
  public d_Object, public G4VHit
{
  public:
      MyCalorimeterHit();
      ~MyCalorimeterHit();
      MyCalorimeterHit(const MyCalorimeterHit &right);
      const MyCalorimeterHit& operator=(const MyCalorimeterHit &right);
      int operator==(const MyCalorimeterHit &right) const;

      static HepClusteringHint clustering;

      void draw();
      void print();

  private:
      G4double edep;
      G4ThreeVector pos;

  public:
      // Some set/get methods here...

};

#endif
```

Fig. 4. A persistent hit class

Table 1
Individual persistent objects

	Tracker hits (in sec)		Calorimeter hits (in sec)	
	Transient	Persistent	Transient	Persistent
User time	7.96	9.63	8.80	13.09
Real time	12.20	14.22	9.63	26.33

Table 2
Persistence by containment in *VArray*

	Tracker hits (in sec)		Calorimeter hits (in sec)	
	Transient	Persistent	Transient	Persistent
User time	8.66	8.37	9.66	8.89
Real time	10.96	15.87	11.28	14.41

hit is accessed only at its construction time. In the case of calorimeter hits, two collections of 19 and 17 hits are created and each hit is accessed 100 times to accumulate energy deposition. The tests were performed on IBM SP-2 at CERN. Table 1 and Table 2 give the resulting performance for the two implementations defined above.

The slightly better user time in the case of "persistent by containment" comes from the improved optimization in the persistent collection class, which is based on the Objectivity/DB *VArray*, whereas the transient case uses a Rogue-Wave collection class. Further optimization of the persistent versions are possible; for example, the performance of the "individual objects" implementation should improve if multiple persistent objects are created at the same time, rather than individually, as is the case shown in the Table 1.

The low overhead introduced by the database is striking, and can be compared with that incurred for storing ZEBRA objects in an RZ file. In a very simple test done by RD45, e.g., using a linear chain of 1900 banks, each bank containing 10 data words, the I/O overhead represents a small factor, i.e. the performance is several times worse in the persistent case, rather than a fractional increase as is seen to be the case of using ODBMS.

4 Trajectory related classes in GEANT4

Fig.5 shows the classes related to "trajectory". The `G4TrajectoryPoint` class object stores only the three-dimentional coordinate of a single point where a step of a track terminates. A `G4Trajectory` class object has a *ValOrdered-Vector* to store `G4TrajectoryPoint` class objects, and it also has some infor-

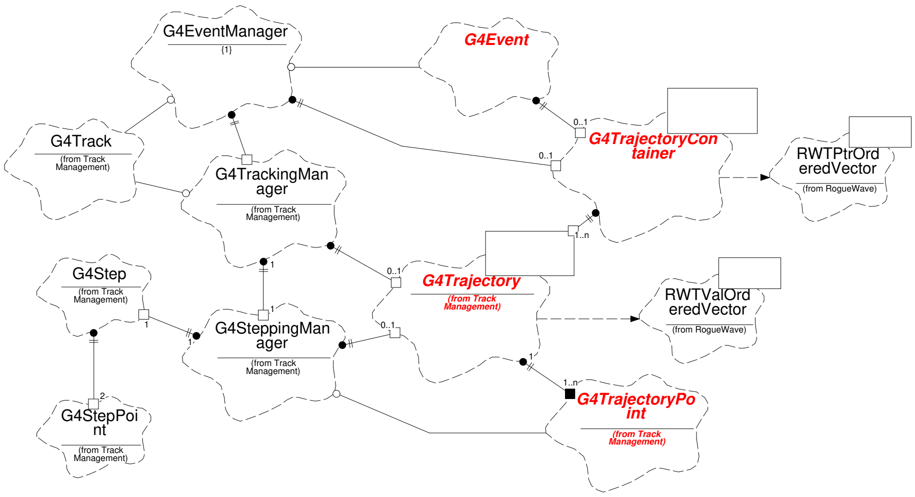

Fig. 5. Class diagram of trajectory related classes

mation about the particle, e.g., particle code, charge, track ID, etc. The user can construct `G4Trajectory` class objects selectively according to the particle type and the nature of the particle, e.g., selecting only for charged primaries. `G4TrajectoryContainer` is a simple container class of `G4Trajectory` pointers and it has a one-to-one association to the `G4Event` class object.

We are going to study the use and performance of an association between a trajectory and tracker hits. This association should be a one-to-many bi-directional association.

5 Summary

The first joint efforts of CERN RD44 and RD45 produced use and performance figures for ODBMS persistency for LHC detector simulation which resulted in remarkable satisfaction. On the way to the beta-release of GEANT4, which is scheduled to be one year from now, we will continue to make various kinds of classes persistent and to provide tools for users to make their derived classes persistent too. Also it should be remarked that the use of ODBMS is a user choice and the interfaces to ODBMS in GEANT4 classes will be abstract. Thus the user can execute his/her own application in both transient and persistent ways with minimal modifications to the source code.

6 Acknowledgements

The author, on behalf of the GEANT4 (CERN RD44) Collaboration, would like to acknowledge the contribution of CERN RD45, especially that of Jamie

Shiers and Dirk Duellmann, CERN IT-ASD, to this work.

References

[1] A.Dell'Acqua et al., RD44 Collaboration, "GEANT4: an Object-Oriented toolkit for simulation in HEP", CERN/DRDC/94-29 (1994), CERN/LHCC/95-70 (1995). User requirement document, class and object diagrams, and prototype source codes are available at
`http://wwwcn.cern.ch/asd/geant/geant4.html`.

[2] T.Sjöstrand, "PYTHIA 5.7 and JETSET 7.4 physics and manual", CERN-TH-7112/93 (1993), LU TP 95-20 (1995).

[3] J.Shiers et al., RD45 Collaboration, "A persistent object manager for HEP", CERN/DRDC/94-30 (1994). Documents are available at
`http://wwwcn.cern.ch/asd/cernlib/rd45/index.html`.

[4] ATLAS Technical Proposal, CERN/LHCC 94-43 (1994).