Summary of persistency scheme discussion

SLAC, Dec 10-13, Ties Behnke, Frank Gaede, Norman Graf, Tony Johnson

Summary done by Ties Behnke

1 GOAL

The goal of this week of discussion was the formulation of a common data model and persistency scheme between the American and the European Linear Collider studies.

2 Brief summary of discussion

Over the course of the week the following topics were discussed:

- Data model for the step between simulation and reconstruction
- Data model for the storage of reconstructed objects
- Definition of the interface between the data model and the user
- Definition of the interface between the data model and the persistency scheme
- Implementation of the persistency scheme in a particular framework

3 Data Model

The data model has to be defined at two (possibly three) stages:

- After the simulation, before the reconstruction
- After the reconstruction

After some discussion the following definitions were agreed:

- The split between the simulation and the reconstruction is defined to happen after the generation of hits by GEANT, and – if necessary – after the combination of GEANT hits into detector-cell hits. **No smearing due to detector resolution is to be included in these hits.** A detector-cell is defined as the smallest readout unit in the detector, which might however contain more than one GEANT hit. In this way it is possibly to change the detector layout within some limits (essentially making its granularity worse) without having to rerun the simulation. Studies of detector resolution effects can be done without rerunning the simulation.
- Reconstructed objects are stored at the level of reconstructed particles. The exact definition of a reconstructed particle might depend on the reconstruction algorithm used. However in general reconstructed particles do not include objects derived based on these particles, like jets, vertices etc. They are treated separately.

Eventually a third step might be needed, after the digitization of the hits. This will be needed for real data, if e.g. test beam data are to be processed in this scheme.

To ease the discussion of the data model a simple XML based scheme has been proposed by Tony Johnson as a way to define the data model. At the moment this is only used to

document the data model, however it is conceivable that in the longer run such a scheme can also be used to define and dynamically create the data model together with the data.

3.1 The HIT data model

For each event one record is written, which is subdivided into several objects:

- EventHeader
- MCParticle
- TrackerHit
- CalorimeterHit

All types of detectors are described by two generic types of hit objects: TrackerHit or a CalorimeterHit. A mechanism is implemented which allows the extension of each of the pre-defined Hit classes by the program developer in a transparent fashion. Both TrackerHit and CalorimeterHit provide pointers back into the MCParticle.

Comment: The CalorimeterHit can be defined in either a long or a short format. In the **long format** each cell records each individual particle type which crosses the cell. Both the energy and the MCParticle producing this hit are stored. Hits produced by the same type of particle are summed. The **short format** stores one entry per cell and per shower which produced this entry. Showers are defined as having the same MC-parent at the point of entry into the calorimeter. A more exact definition of the parent of a shower is needed.

3.2 The Reconstructed Particle data model

The result of the reconstruction procedure is a list of particles (or pseudo particles) which are then used as base for the further analysis of the event.

For each event, the event record is extended to include information on these reconstructed particles:

- OutputHeader
- ReconstructedParticle
- TrackObject
- ClusterObject

Each reconstructed particle is built up from TrackObjects and ClusterObjects. Pointers exist from the reconstructed particle to the TrackObject and ClusterObject. The same mechanism defined for the hits can be used to extend the information in the predefined objects in a transparent way. Optionally pointers exist from the Track of Cluster Objects back to the hits.

Objects constructed from reconstructed particles (jets, vertices etc.) are stored in a fifth object type, the ReconstructedObject class. Its data and methods are very similar to the

reconstructed particle, but its pointers point only to ReconstructedParticle objects, not to Track or Cluster Objects.

3.3 The Extension mechanism

The defined data format fulfills the common requirements for linear collider simulation studies. Nevertheless the need to store additional information in the files for a particular analysis might arise. Two extension mechanisms are foreseen.
One, the user will be able to add additional information to any entity described in 3.2 and 3.1 (e.g. CalorimeterHit, TrackObject) in form of an ExtensionObject. Such an object has an arbitrary but fixed number of ints and floats, to be defined by the user at run time. Access to these objects in read mode is gained through a unique name, also defined by the user (at writing time). This additional data will not be read if not required by other users, using the same file.
Two, users will be able to store arbitrary objects in the event, given they provide an implementation of a corresponding HandlerClass for the particular persistency format (SIO).
With those two mechanisms, users have all the flexibility they need for their analysis using the LCIO package. If it turns out, that some additional information is used by a lot of users, it can be easily incorporated into the data model in a new release.

3.4 Interfaces

It is planned to provide interfaces to the data model for a JAVA based environment, for a C++ based environment, and for a FORTRAN based environment. To ease the development of the interfaces, the interface definition is done in AID (Abstract Interface Definition), a package, which allows the definition of the interface in a language independent way, and which provides tools for an automatic code generation of the interfaces in JAVA and C++.
It is foreseen, to define one interface per entity, defined in 3.2 and 3.3, that is read only, i.e. has get() methods only. We will provide two implementations for these interfaces in terms of concrete classes. One for reading the event, which is read only as well and one that has also set() methods. The latter is a convenient implementation for writing the events, that the user might or might not use. For existing code users will probably have existing classes. In order to use these, all they have to do is implement the interface.
There will be interfaces for a Reader and a Writer, that have methods for the extension mechanism described above.
The Fortran interface will be based on the C++ interface (i.e. users have to include a C library in their programs). A possible way of providing all the functionality of the OO-implementations in Fortran is to implement a C-function for every class member function, that has a pointer argument (Integer in Fortran). Thus, actually providing a OO-like interface in Fortran. There are two advantages to this approach:
One: code has to be written only for Java and C++ (plus trivial wrappers for C/Fortran).
Two: users will have to think in terms of OO, facilitating their moving to Java or C++ in the near future.

This comes at the price of two additional function calls per data member access, which would discourage the development of new software in Fortran, something not necessarily to be regarded a disadvantage.

4 The persistency package

The choice of the actual persistency package is governed by a couple of considerations:

- Simplicity
- Availability for different platforms
- Support for pointers in OO languages
- Compact storage of data (packing supported)

It was decided to base the first implementation of the above scheme on the SIO package. In the context of SIO the following mapping will be done:

LC data model		SIO implementation
Run Header		Record
Event		Record
	Event Header	Block
	MCParticle	Block
	TrackerHit	Block(s) (one per detector)
	CalorimeterHit	Block(s) (one per detector)
	Output Header	Block
	Reconstructed Object	Block(s) (optional)
	Reconstructed Particle	Block(s) (one per object type (jet, …)
	Track	Block(s) (one per track type)
	Calorimeter	Block(s) (one per detector type)

Note: since SIO only supports pointers within records, not across records, the complete event including hits and reconstructed objects has to be stored as one record. This potentially has a drawback in the efficiency with which events can be read, when only the reconstructed information is desired.

For all standard objects the user will be completely shielded from the SIO package.

5 Open Questions

At the moment no interface of the SIO package to FORTRAN exists.

6 References

1.A.P.Waite: Serial Input Output. 8. November 1999, Version 1.0
2.AID: http://java.freehep.org/lib/freehep/api/org/freehep/aid/package-summary.html