McVertex and McParticle

Purpose and use in applications:

1) output of generator (primary vertices and primary particles)
2) input to simulator (Geant4, Gismo)
3) kinematic of simulated event: simulator adds secondary vertices and particles to 1)
4) verification of reconstruction: comparison of simulated vertices and tracks with reconstructed ones, were the comparison is (normally) limited to the comparison of the particle quantities at the vertex of origin.

In more detail:

1) In Glast the Generators are flux generators
   - no real point of origin, the particles are generated according to the flux distribution on a surface related to the detector (and its position, orientation, etc). A single particle is generated with
   - particle properties (type, 4-momentum) at a (fictitious) vertex of "origin"

2) Geant4 expects as input
   - primary vertices and primary particles originating from these vertices

3) Kinematic of simulated event

use cases:
   - conversion of gamma: 1 incoming particle, 1 conversion vertex, two outgoing particles
   - interaction: 1 interaction vertex, 1 incoming particle, 1 interaction vertex, 1 to n outgoing particles
   - bremsstrahlung with gamma above threshold: 1 incoming particle, 2 outgoing particles (one of them has the same identity as the incoming particle)

4) verification of reconstruction requires
   - comparison of the reconstructed vertex with the simulated vertex position and the reconstructed track quantities with the properties of the trajectory at the vertex (=McParticle).

Vertex and particles:

case 2
   - separate objects are required for vertex and particles in Geant4

case 3
   - if the vertex would be part of the McParticle, the vertex information is repeated several times. Repeating information on a database is unfortunate. Consider an application, which plots the distribution of the conversion vertices. It has to figure out, which 'vertex' to ignore! Worse even for interaction vertices.

case 4
   - association of reconstructed and simulated objects. When McVertex and McParticle are separate, the mapping between reconstructed objects and simulated ones is a well defined task. If McVertex and McParticle are in the same object, the mapping becomes ambiguous.

case 1

- Because there is only a single particle simulated, vertex and particle could be described in the same object. The generator MUST not use the same class definitions as the simulator. So: should we have a FluxParticle? I think it is not worth the additional class definition, but it is a possibility.

The tree of McVertices and McParticles

It should be navigable. This is achieved, if the vertex knows all its daughter McParticles and each McParticle knows its end vertex. The McVertex has therefore a list of pointers to its daughters and the McParticle has a pointer to its end McVertex. In addition the start vertex has to be recognizable (either by a pointer from McEvent or from its vertexType).

This is sufficient to do all navigation. For convenience, one may provide also easy navigation 'up the tree': additional pointers can be provide (McParticle points to the McVertex of its "origin") or one can build a temporary map in TDS, which provides the additional navigation. The latter is a better choice, because it avoids duplication of information on the persistent database.

The additional pointers were chosen in my original proposal, Fig. 5 in [1] and related text Section 3.1.1. I would now opt for the temporary map. Within Gaudi, it is of advantage to choose SmartRef and SmartRefVector (loading of objects on request only) instead of straight C++ pointers.

The present implementation, as shown in Toby's message [2] defines a McVertex, which is neither a particle nor a vertex. A vertex is a position plus some identification, but should not contain 4-momenta.

Remains the fate of a particle when traversing material (energyloss, multiple scattering). This is described by the trajectory of the particle. If needed the trajectory or pieces of it can be stored also (e.g. the "end" of a trajectory) and relationships (pointers) established with vertices and particles. Note that the trajectory is the "source" of hits.

Summary

- The vertex and particle information in the kinematic event should be separated. If combined, information is duplicated and the applications become confusing and complicated.
- The output from the particle generation using the flux generator (or a single particle gun) could be a combined object with "origin" & "particle" information in the same object, a "FluxParticle".
- Much work has gone into the existing McParticle and McVertex definition, only the associations are not done correctly.

References

1) http://www.slac.stanford.edu/~hansl/glast/note/index.html

--> Raw data definition, version 0.7; Fig 5 and Section 3.1.1

2) http://www-glast.slac.stanford.edu/software/DataStructuresTF/20020128/McVertex.htm

3) Temporarily I have put the present full definition (UML style) on

http://hansl.home.cern.ch/hansl/glast/datamodel/

--> Ongoing work