Discussion on Physics Event Model - November 27, 2001

Present:
O.Callot, M.Cattaneo, Ph.Charpentier, G.Corti, M.Frank, A.Kahn, P.Mato

Resuming last week discussion it is felt that we should not force a base class for all particles. The commonality is minimal and for similar methods it can be ensured by having the same nomenclature.

When wanting to apply, for example, vertexing tools to MCParticles the Particles can be filled with MCParticles. An algortihm performing this operation will be provided.

The particle class in the Physics Event Model will be simply named Particle, following the general LHCb Event Model convention that prefix are for MonteCarlo related stages.

Particle and Vertices relationship will have to be represented the same way in the MC and Phys world.

The "originator" of initial Particles can be a ProtoParticle or a MCParticle. The Particle will have as a data member the key of the originator (see LHCb Event Model general discussion). This will be resolved by the software only when following it, via a template method.

```
Ex:   Particle P;
      ProtoP*   pp = P.getOriginator( ProtoP* )
      MCPart*   mp = P.getOriginator( MCPart* )
```

Composite Particles will have an invalid key and return void pointers to ProtoP.

There will be one collection of Particles and one collection of Vertices.
Should resulting particle containers be provided? Or a single container with all composite particles? This would be in memory and if necessary a part of this container can be saved…

There will be in any case in memory lists of Pointers (key?) to the original containers with the various "type(ParticleID)" of particles. These "lists" will be created starting from the original containers, via tools that "fill" these lists.
If later we see the need to save such lists the pointers can be converted to the key of the objects.

In this approach we would start with 2 separate containers one for Particles, one for Vertices, where the one with Particles is filled by a preprocessing stage with Particles populated with ProtoParticles and that of Vertices is empty.
The idea would be for Particles produced by a specific analysis to be put in a new separate container specific to the analysis, with clones of the original Particles.

This way an analysis is self-contained and a separate algorithm would start with the same setup. (Comment: but wouldn't a separate algorithm anyway start with maybe a different set of Particles?)

The K0s problem: A user will want to find K0s in the particle container without having to change its code. On the other hand, this should be found in the same container….
So DaVinci should provide a jobOptions for running of files where the K0s has been reconstructed in which case the Transient store is populated from the Persistent store, another in which the store is populated by a K0s producer algorithm.

If particles in a "tree" are modifiable by refitting then each time they are attached to a "mother" they have to be copied. This implies that there will be copies of the "same" particles in the container, both accessible…this would imply that kinematic fitting would only modify the parameters of the mother, in which case no copies are necessary.

Cloning of Particles implies that only mono-directional links are allowed.

Philippe raised the issue that maybe it is not necessary to have a separate container with Particles from ProtoParticles and a container with what an analysis has used and produced.

At the end the point of internal/external three relation is still open.
Gloria will write an example of user code with one and the other possibility and we will choose based on that.