Discussion on Physics Event Model - November 13, 2001

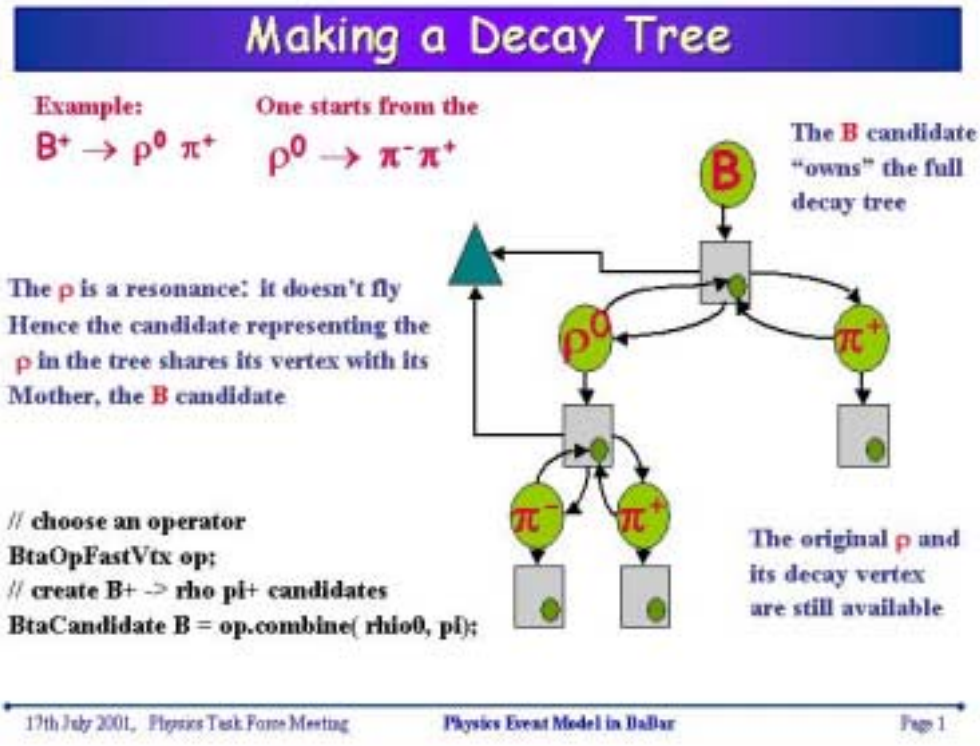
Present:
O.Callot, M.Cattaneo, Ph.Charpentier, G.Corti, M.Frank, P.Mato, E.deOliveira, O.Schneider

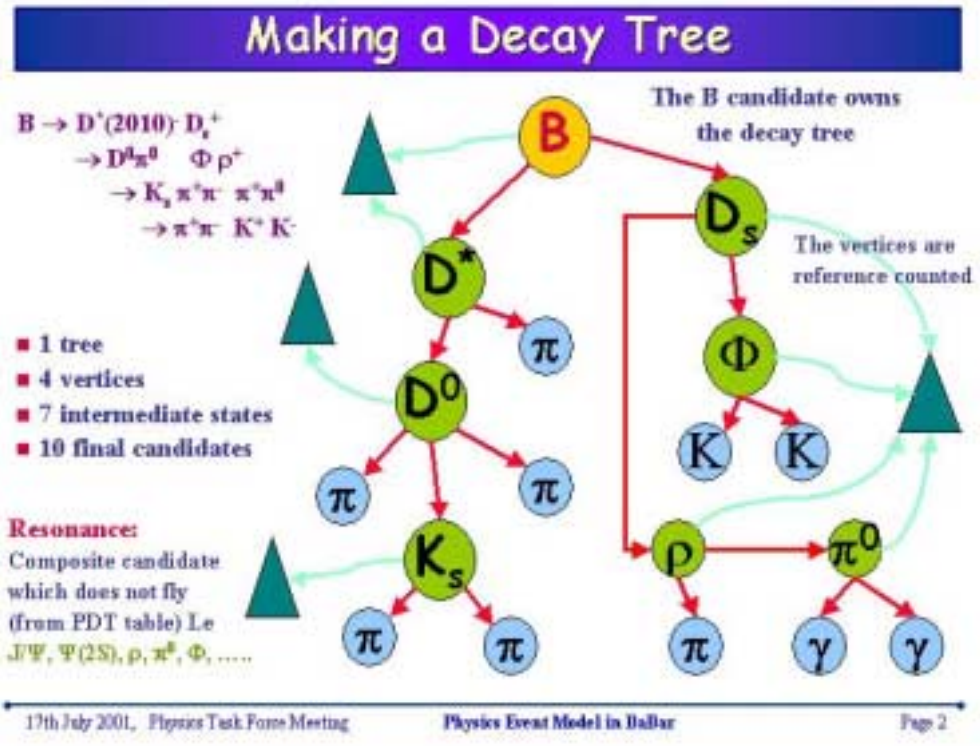
The discussion concentrated on the Relationship/History of particles: essentially how to represent the tree that particles build. The main question was if we want to store all info about the "tree particle" together with the relationship or not.

- **Some questions to guide discussion on Relationship/history**
    a. in the class or in the container
    b. is the tree a container or a navigator
    c. which if the container that owns new particles when in transient store
    d. persistency of the relationships
    e. dropping of intermediate steps
    f. overlapping of trees (for parallel analysis)
    g. how easy is for user to work with a tree: creation/retrieval

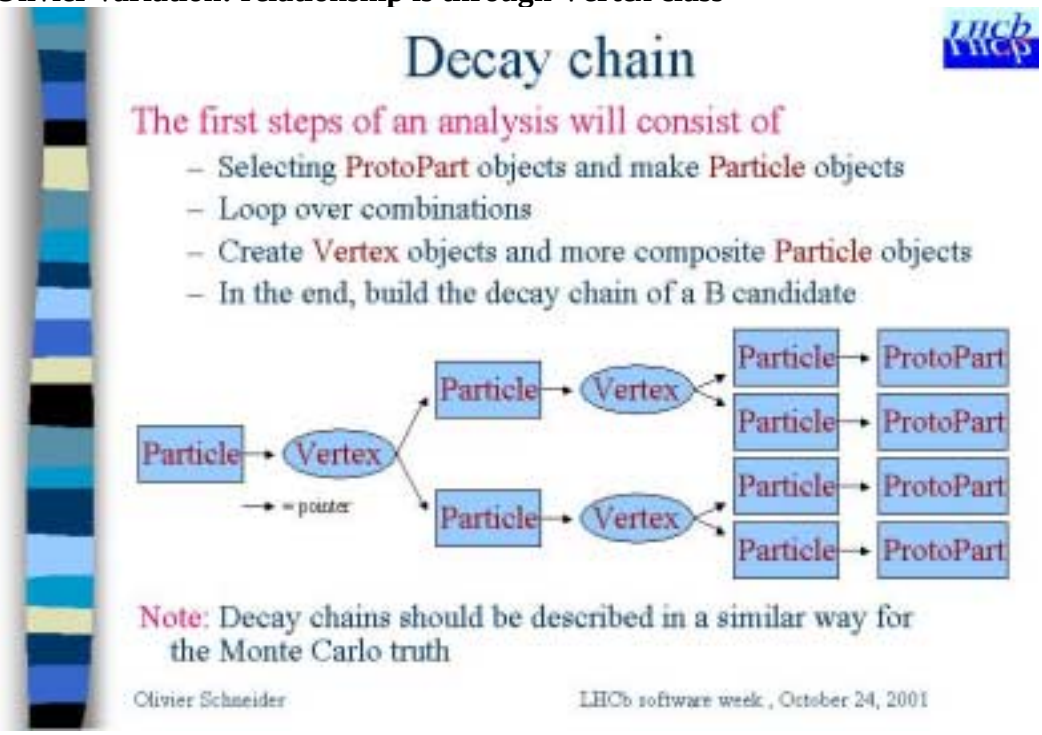
Babar has opted for storing the relationship history in the classes itself, something similar but with always vertices has been proposed by OlivierS., Delphi has opted for having the relationship completely external to the classes and putting particle and vertices in a tree/graph container. Attached are the transparencies illustrating the 2 approaches + variation.

BaBar approach: history relationship in Particle class





Olivier variation: relationship is through Vertex class



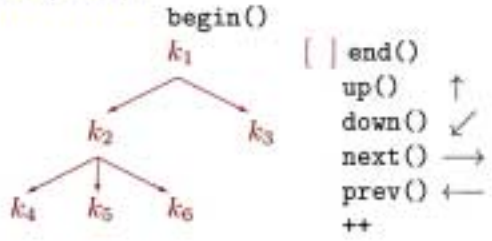
Idea's approach: relationship is in container

History: Trees of Particles and Vertices

Sequence

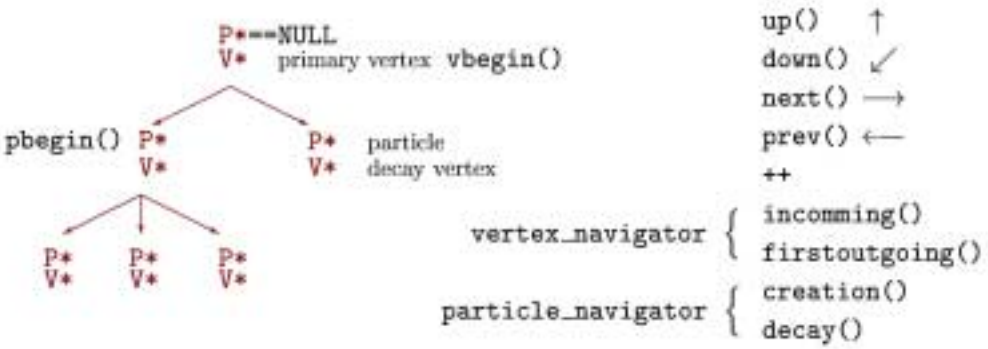

```
for(iterator it=c.begin();it!=c.end(),++it){ cout<<*it<<endl;}
```

Tree/Graph




```
navigator nav=t.begin();
if(nav.down()){  cout<< *nav.down()<<endl; }
```


Particle-Vertex-Tree/Graph



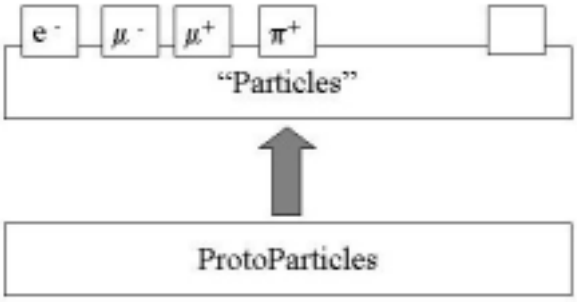

```
particle_navigator p_nav=t.pbegin();
if(p_nav.down()){  cout << **p_nav.down()<<endl; }

vertex_navigator v_nav=t.vbegin();
if(v_nav.down()){  cout << **v_nav.down()<<endl; }
```

- Dropping of intermediate stages is mostly a MC issue and should be solved by the algorithms that drops the information keeping consistency from a physics point of view. Both approaches of storing the relationship have the problem.

- The first step main goal of a physics analysis will be to "reconstruct" a decay tree → to build and navigate a tree must be as easy as possible.

- The construction of the tree starts from the base by combining Particles "selected" from ProtoParticle to make a new Particle: for example starting from muons to make J/Psi. The J/Psi "belongs" already to a tree structure: its own. Head of these simple trees are then combined to make a more complex tree: J/Psi and Ks for example to make B^0. **Only this final tree with all its particles could/should be saved.** There are about ~ 10 particles in a final tree.

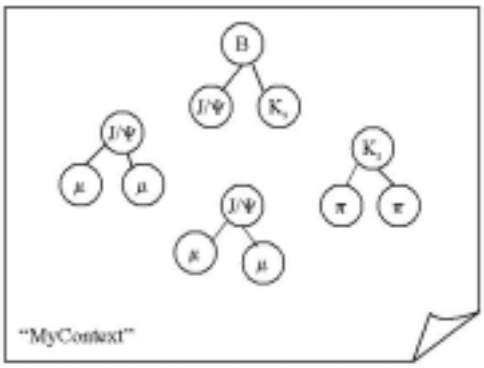
- ProtoParticles are in a flat structure and are produced as last stage of Reconstrutcion. There are ~ 70 ProtoParticles/event. Each ProtoParticle has all possible particleID hypotheses. A "pre-processing" stage in an analysis will be to make a flat list of Particles (assigning a particular particleID) from ProtoParticles.



- Intermediate stages should be seen as separate trees. Copies of the particles should be made when putting a head of a tree as a leaf of a new tree.

- The transient store should not be inflated with various strings for all possible trees

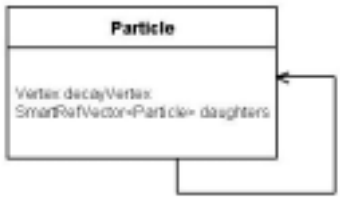
- Each analysis will have its own context to which all partial trees will belong. If a final tree needs to be saved (transient or persistent store), a copy of the tree can be saved.



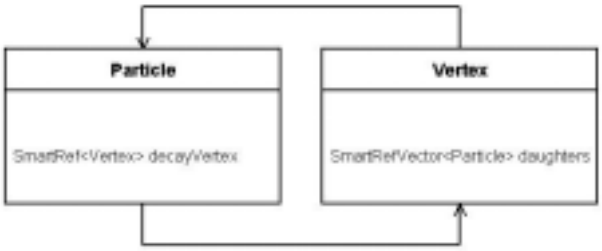
- Different particles can have SmartRef to the same ProtoPart. Only when using the head of a tree made with this "reconstructed" Particles as a leaf of a new tree the Particles can be checked to be exclusive: particle operator that combines the particles should check for a tree to be self-exclusive. This is delaying the check at the end.

- Considering all of this navigation issues it seems to be easier to have the history relationship in the classes. In this case there are various options for where to put the relationship and connect Particle and Vertices.

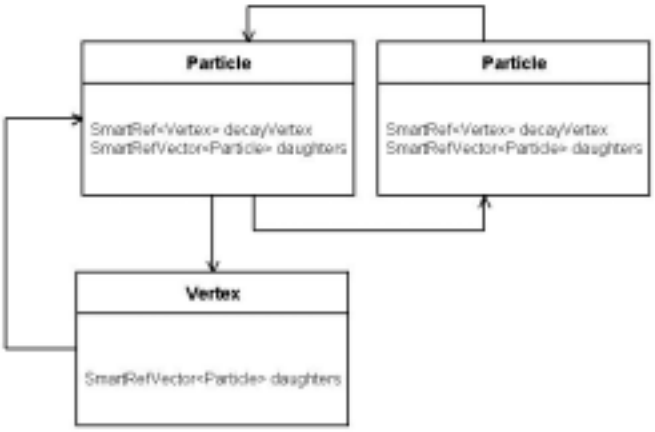
  Option 1



  Option 2



Option 3



- A vertex should not always be internal to a Particle (Opt.1): ProtoParticles will not have decay Vertices
- Particle with zero lifetimes would not intuitively described by Option 2 but more by option 3. One the other hand Option 2 is “simpler”. Particle with zero lifetime could be taken care of by the particle having a zero lifetime flag for which the particle used to make the tree have to be taken. Vertexers should then check this flag. Option2 while in some sense equivalent to Option1 is more flexible.

.