AIM

What we need:

1. a new transient representation of a V0 as a two-track object for use in HLT and Brunel. The LHCb::RecVertex is not sufficient because it does not store any momentum information. The LHCb::Vertex is only for use in DaVinci since is is based on Particles.

2. a transient representation of a V0 as a Particle for use in DaVinci. This exists, but it is not clear how we get to this from step '1'. In the end, the user wants

- a list of Ks candidates
- a list of Lambda candidates
- a list of Lambda-bar candidates

and shouldn't need to worry about how these are created.

3. a persistent representation of a V0 to get from Brunel to DaVinci. This does not exist either. It could well be the transient class '1'.

There are several steps in between, such as an eventual representation as a ProtoParticle. Ideally, there are as few classes as possible to get a V0 in any form we like.

Particles and ProtoParticles

A ProtoParticle represents a reconstructed particle. There is currently a single ProtoParticle class. It holds pointers to tracking, RICH, MUON and CALO information corresponding to a single particle. ProtoParticles are created 'at the end of Brunel'.

A Particle can represent both a stable particle and a composite particle. There is only a single Particle class: It contains a pointer to a ProtoParticle (for a stable particle), a list of pointers to daugters and a pointer to a Vertex (not RecVertex) and a particle 'hypoithesis' (type). If the particle is a composite, the pointer to the ProtoParticles is zero. If the patricles is a stable particle, the vertex and daughter information is void.

See also the https://twiki.cern.ch/twiki/bin/view/LHCb/LHCbPhysicsEventModelTaskForce#Particle

Vertexing V0s

Among all composites, V0s are somewhat special, because they are abundantly produced in minimum bias and easy to identify because of their long lifetime. We would like to reconstruct V0s early in the reconstruction code (ie. Brunel), because a) we have more information on the tracks available in Brunel (the knowledge that the daughter tracks come from a V0 may help to inmprove the tracks) and because they are useful for monitoring tracking performance when little data is available.

There are two arguments why the representation of a V0 could be different from that of other composites:

- a. we do not assign a particle hypothesis to the daughters, ie daughters are not of the Particle type. Rather, we reconstruct the vertex once from the tracks (or ProtoParticles) and then reuse the vertex for the different Particles, corresponding to the different V0 species.
- b. for now, we should refrain from using Particles in Brunel.
- c. V0s fly: For most composites the tracks states at the beamline (stateAtBeamline) are good enough for vertexing because the vertex is close to the beam line. This is in general not the case for V0s. We may want to store dedicated states on these tracks.

A possible reconstruction sequence of a V0 could be:

- a. select tracks (or ProtoParticles), select combinations, 'fill' a vertexobject
- b. create (theoretically) up to 3 Particles from this vertex

Given the requirement on the use of Particles, step 'a' could be done in Brunel/HLT, step 'b' only in DaVinci. As discussed in the meeting, one problem with this approach is that we would actually need a way to propagate from step 'a' to step 'b' what kind of V0 we are dealing with, ie if it passed the mass cut for a Ks, a Lambda and/or an anti-Lambda: In particular, we do not really want to perform the mass selection more than once.

Representing the result of a 2-prong vertex fit

Sean and I have represented different models suitable for representing V0s. They essentially correspond to the two different ways of representing the result of a fit of a vertex to two stable particles, namely

- a) by storing the vertex position plus the p4 of the mother (7 parameters)
- b) by storing the vertex position plus the p3 of each daughter (9 parameters (or 11 if daughters are composites))

Model 'b' is the model used in a vertex fit: the parameters from model 'a' are extracted after we have determined the parameters of 'b'. This does not change if we do not actually perform a vertex fit, but just substitute the daughter momenta with those extracted from the unconstrained track. The advantage of model 'b' is that it is independent of the flavour of the final state particles, because (for non-composite daughters), we can change the 'mass' of the final state particles without affecting the rest of the vertex calculation.

Model 'a' is the model used in a Particle. The main advantage of model 'a' is that it uses less space and that it caches exactly that information used further vertex fits or a physics analysis. The disadvantage is that the representation assumes a certain particle hypothesis (mass, energy) of the two daughters: We know only the p4 of one of the 3 solutions. Sean has suggested that to solve the latter problem one could calculate the p4 of the mother assuming pions for the daughters and then add additional data for the invariant mass of the p-pi and pi-p pair, to be filled by the algorithm that creates the vertex. Note that, ignoring for the moment cov matrices, the model is then as large as model 'b'.

With these things in mind, I argue that if we want to use a vertex object that represents more than one species of V0, then we should just use model b. Let's call this a TwoProngVertex 

V0s and (Proto)Particles

At the end of the Brunel-Davinci chain we need V0s represented as particles. Assuming that we have a vertex object, how do we get the Particle?

Patrick would like to fit the V0s into the existing Track/Calo/Rich/Muon --> ProtoParticle --> Particle scheme, changing as little as possible. He has suggested that we extend the ProtoParticle class with a pointer to a reconstructed vertex. I am not convinced that this is a good idea.

First, as was pointed out by Marco, if the vertex class is not based on ProtoParticles but on Tracks, there is no easy way to access the PID information. Second, there is no 'data' overlap between a ProtoParticle containing from Track/Calo/Rich/Muon information and a ProtoParticle pointing to a V0 vertex. (This is the same reason I do not like that we use a single class to represent composite and stable Particles.) So, if we really need to create ProtoParticles for V0s, then it would be better to create a new class 'ProtoV0', which has a common base with ProtoParticle.

However, I wonder why we need the ProtoV0 at all. I do not understand why we could not just create Particles directly from the TwoProngVertices, skipping the ProtoParticle step. The (simple!) algorithm would take the vertex list, find the (Proto)Particles corresponding to the tracks and output 3 new Particle lists. It could eventually refit the vertex with a standard vertex fit. (See also the persistency below). It could run directly after the algorithm that creates the standard Particle lists (for pions, kaons, muons etc.) It can at any point be swapped with the existing algorithms for creating K-shorts in DaVinci.

Persisting V0s

I'll discuss here the persistence of V0s to get from Brunel to DaVinci. In the HLT everything needs to go in a summary bank, so that's an entirely diferent story. Probably, in the HLT there is no reason to treat the V0 differently from other composites.

I think that there are two alternatives for persisting V0s. First, we could persist just a pair of pointers to tracks. To reconstruct the V0 in DaVinci you would redo the vertex fit. The disadvantage of this approach is that in DaVinci the tracks are less complete than in Brunel. You may wonder if it is worth to persist V0s at all in this scheme.

The other alternative is to persist the full TwoProngVertex. It contains sufficient information to fill the V0 Particle with the same data that a standard vetrex fit in DaVinci would give. The problem with this approach is disk space: including a chisquare and a covariance matrix, we are talking about the equivalent of about 60 doubles. That's seems a lot, but it is actually not that much compares to a track: a track LHCb::State is about 20 doubles. Persisted long tracks have at least 3 states (?), plus quite a bit of other stuff. If the number of V0s per event is limited to a few, the disk space should be quite acceptable. (It get's a lot better once we implement the compression explicitely, because we don't care to much about detail in the covariance matrix.)

-- WouterHulsbergen - 19 Nov 2007

This topic: LHCb > V0Persistence
Topic revision: r2 - 2007-11-19 - WouterHulsbergen