This is page where you can put your inputs/comments/... For new topics, please start a new item, and for comments to existing items, please add subitems.

Fits for different hypothesis

- GerhardRaven (2005/04/05 17:16)

How to handle fitting the same 'track' as a pion and as an electron? The electron fit may look very much different from the pion one, and may include things like the addition of bremsstrahlung in the trackfit itself (which changes the momentum at some location). This type of change is 'easy' mathematically in a Kalman formalism (it is in principle no different from the effect of energy loss in material). Is a new 'track' made for a different hypothesis? How is the bookkeeping done all the way through to the analysis level, i.e. how is the 'overlap' between the 'electron' and 'pion' versions of the track handled?

- MarcelMerk (2005/04/06 10:38)

This is a good point. We already have such a use case if we decide to refit electron tracks with our current implementation of the electron fit. The trajectory can have larger local changes of curvature and the covariance matrix is larger. In the past we indeed considered the possibility to add photons into the electron track, but was not implemented due to lack of time. In that case we clearly would not only have to create a new protoparticle but also the underlying track changes.

Some simple implementation of this could be to add a flag to the track specifying the particle hypothesis used in the fit; e.g. *MIP* or *electron*. In the standard reconstruction tracks are fitted as MIP, in a later stage they can be refitted with other PID hypotheses.

- SomeOtherPerson (some date) Some clever comment on the above subject goes here...

Is there a need to have a seperate/dedicated class representing trajectories

- GerhardRaven (2005/04/05 17:40)

At this point, there is no seperate class which represents a trajectory, and its functionality is fully 'absorbed' into the track. The open question is whether a seperate object makes sense.

- MarcelMerk (2005/04/06 10:13)

This is a bit of a change with respect to the current model. The track becomes a lighter object and a trajectory object is added. A difference is that in this case a finite geometrical representation of a trajectory exists that can be obtained analytically, i.e. without making use of the transporter. I think if parabola, helixes, lines, etc. A use case could be vertexing. It is still not clear to me whether one would intersect **trajectories** or **tracks** with surfaces or track,since in most use cases (apart from vertexing) extrapolation will happen over distences longer then the validity of the trajectory.

- SomeOtherPerson (some date) Some clever comment on the above subject goes here...

Is there a need to specify algorithm settings in the history?

- GerhardRaven (2005/04/05 17:40)

To me this seems more in the reign of general Gaudi provenance, i.e. how was the executable which did some processing configured...

- MarcoCattaneo (2005/04/06 15:50)

I think the answer is no, this is indeed a general Gaudi problem. For a given event we must know the version of the program was used, and hence its configuration. It is a property of the event, not of the track. The same applies for conditions data. In fact, I don't really understand the current definition of history, in the current implementation the track type is sufficient because the sequence of algorithms needed to produce that track type is unique for a given program version.

Concrete Track ans State classes

- OlivierCallot (2005/04/12 18:05) I think that we shouldhave a Track class and a State class which are not base classes (with virtual

methods) with the real object being a derived class, but the one and only one class for tracks and states. There is almost no use case for states without QOverP, and so little difference in pattern and fit tracks. Nothing prevents from adding new functionality to these classes, like for example adding a timevalue to the State. The only difference is that one can not overload an existing function. In case, when we have a real use case, and that we need to overload a function, this would require to make it virtual (-> not inlined) inState (or Track), which is simple and has only minor side effects: We re-compile everything when there is a new release, due to the Project management of our software. Then all code will see the new signature, which iscompletely backward compatible.

In short, take the existing proposal, remove all the virtual (and tests on type of state) for the State class. For Track, add the few bitfields needed bythe pattern recognition (R Sector for Velo RZ result, some bits for 'bad track' and 'used', and the ancestor pointer vector we agreed on. Add a std::vector<Measurement*> for the fit, or whatever object we will have to combine Measurement and Node, and that's it. For me this would be great, as this is in fact the closest I can imagine from the TrgTrack classes ! So the adaptation of the Trg code would be a piece of cake.

I hope this proposal is clear. If you would like to have an xml implementation, I think Jose and Eduardo can provide one quite easily. Let me (us) know. Cheers

- JoseAngelHernandoMorata (2005/04/13 10:27) Yes I will say that we 'agree' that with the extended Track class we have

now the functionality that we need for most (all?) part of the reconstruction code (pattern and fitting) as we saw on monday. The same is valid with the State. Then Track and State will be them real useful objects for the tracking. This still not implies that people will not derive them :) as adding the time is the state it is trivial use of derivation.

Ok, having in mind the extended Track class and the revisiting of the classes following the requirements that we did on monday, this is what Eduardo and myself we are thinking to do for the classes:

1) for Track as data members: a vector of Track* for the ancestors a vector of Measurement* for loading the Track from LHCbIDs a vector of Node* for the Fit to link State-Measurements with residuals and local-chi2 None of them will be persistent by default. as method: (the 'flags' will be now set using strings or int, instead of the enum that are 'fix' in the header file, this will require to have an external class with the valid type of 'flag', here for 'flags' we understand: History (PR, and Fit), Type, Status, etc, in this way the user can add its own flags without re-compiling). a method to set the History for PatRec and Fit algorithms a method to set the Status ("PatRec","ReadyToFit","Fitted","Failed") Olivier, if you do not find a better way, with a external helper class for example, to give you the R-sector for the RZ tracks, we will let some space in the bitfield for

you to locate them.

2) for State as seems that our base State it is a favorite one, we can remove the 'virtual' from the method q/p(). we can always put the virtual if someone really need it. So we have a State class with 5-dimension.

3) For ITrackExtrapolator We keep in mind that in the future they will need to be 'smarter' and that we should provide extrapolations to points and lines.

Except for the 'flags' the other changes are easy to incorporate. If there is no complains, Edu and myself we will go ahead with the changes, and we hope that on monday Gerhard can annonce good news of the result of the task force.

- OlivierCallot (2005/04/13 11:59)

It is always possible to add information on a track outside a track: One can derive a class with another flag word,one can have another objects with attributes to a track by key,... It is clear that the persistency doesn't need too much information concerning the intermediate statuses of the final tracks, as bad tracks are removed, Velo Sectors are used only for building space tracks from R-Z ones, and so on. Having an auxiliary object is not too nice, but this is a way to be independent of the track fitting for example. If this is what is preferred by the task force, I will go this way. If not, here is the list used in the Trg tracks.

-
  - bitfield name = 'badTrack' length = '1' desc = 'Track should be ignored'
  - bitfield name = 'alreadyUsed' length = '1' desc = 'Track has already be used by another algorithm'
  - bitfield name = 'backward' length = '1' desc = 'If the Velo track is going backward'
  - bitfield name = 'selectedIP2D' length = '1' desc = 'Track was selected by the impact parameter in 2D'
  - bitfield name = 'selectedMuon2D' length = '1' desc = 'Track was selected by the 2D muon match'
  - bitfield name = 'hasMissedVeloStations' length = '1' desc = 'Track has no hits is stations before first measure'
  - bitfield name = 'RSector' length = '4' desc = 'R sector this 2D track was measured in'
  - bitfield name = 'vertexNumber' length = '4' desc = 'Primary vertex number'

Next Topic goes here...

-- GerhardRaven - 03 Apr 2005

This topic: LHCb > TrackmodelInputs
Topic revision: r9 - 2005-04-21 - GerhardRaven