Thursday

My editorial comments are in italics.

Morning Session

Thursday, 9:00 AM, Sierra conference room

Hans, Tim, Norman, Jeremy, Caroline, Nick, Rich, Dima, Rob

Tony's showed his Use Cases wiki page

- He wants us to add use cases to this page.

We noted that a cluster can have a gap in it if a strip is know to be bad dead or hot. For now clusters will be contiguous but we anticipate that we will need to have non-continguous clusters at a later date.

Jeremy:

- Virtual segmentation of sensors is not yet "canonicalized".
  - Virutal segmentation means tiling the surface of a sensor with pixels and strips. For example, the geometry system might hold one very long strip sensor. It might be broken up into several shorter strip sensors for purposes of assiging a hit to a readout channel.
- A Sensor is a specialized DetectorElement.

Rob mentioned the issue of the histogram code in the example to access the SimTrackerHits.

- Tne example is appropriate for the purpose of showing how to access this data.
- The example teaches new users a way to fill histograms that is elegant in that it is all done in a single line of code
- However if a users follows this model to create and fill many histograms, the resulting code can be very slow because aida has to search for the right histogram to fill based on the string identifier.
  - The group welcomes additional examples that show methods that execute faster.
- Tony mentioned that the search for which histogram to fill is optimized:
  - In many loops histograms get filled in the same order in each pass.
  - So aida's search algorithm is: if I just filled histogram h1, and usually after h1 I fill h2, then on the next call to fill a histogram I will first look to see if the requested histogram is h2.

Tony has calorimeter clusterers that are intended to work with a wide range of geometries, so long as the geometries support certain features. He wants to make sure we include these features. I missed some of the detail.

This lead to a broad discussion about one geometry element knowing its neighbours and how this should be implemented. Lots of it was calorimeter oriented. For example, What if the neighbouring cell is on a different detector element?

- Tim: This problem is in the plan but it is not yet implemented.
- The general algorithm might need to go up one level in the geometry heirarchy to solve this.
- What API needs to exist?
- Not a problem for tracking since the resolution of a hit is large compared to the gaps betweeen neighbours. But it is important in calorimetry.

Rich showed a new candidate for the cluster class.

- It forwarded a lot of geometry functions that we agreed to remove.
  - Just provide access to the geometry instead.
  - The stripped down class was shown in the afternoon session (below).
- We agreed that the getResidual() method did not belong. It belongs in a class with many more features.
- Tim did not want to provide a "getStripLength" feature since it can have a poorly defined meaning for a cluster.

Norman showed some TRF stuff:

- All tracks are fitted presuming the pion mass for energy loss and scattering.
  - There is no option to use a different mass.
- Still needs the bridge code to instantiate its geometry from the org.lcsim geometry.
  - Should be simple once that is decided.
- Language: hits and misses
  - hit can have material associated with it.
  - miss is material without a hit (eg. beam pipe, inefficient detector ....)
  - Point of closest approach to origin is one type of miss.
- He showed his hit class:
  - to use TRF you instantiate TRF hits from the clusters in the event.
- Native output includes track parameters + cov at all hits and misses.
  - No persistency yet
  - He wants to shoehorn his Hit class into TrackerHit.
- He allows kinks on tracks.
  - *I am not sure what this is for.*
- Cylinders for PCAO style tracks are always coaxial with the z axis
- Planar surfaces are always perfectly alighned but we could make new hit derived classes to fix that when needed - not a problem.
- He implements both thickscatter and thinscatter materials
- Straggling in energy loss does get into the covariance matrix.

- At the Saclay meeting last week, Ties presented a trajectory class that could hold track parameters + cov at multiple locations. Others prefered extending the existing Track class.

Afternoon Session

Hans, Tim, Norman, Jeremy, Caroline, Nick, Rich, Dima, Rob

Rich presented a stripped down cluster class that was discussed briefly. It was very close to the class Rob presented on Wednesday.

Dima presented an interesting idea to split the concept of a cluster into two pieces:

- ClusterData
  - a list of hits plus bookkeeping
  - no position or error, local or global, is available from this object.
  - This would be added to the event, in an organized container, not a flat list.
  - We do not expect this to be persistable.
- TrackCluster
  - something that can be created from ClusterData plus a Helix.
  - It is possible to specify the helix as null and the class must perform well defined actions when this is done.
  - This gives access to position information in both local and global coordinates.
  - Global information can be created by a lazy evaluation. *I think it must be evaluated this way*
  - The class allows the user to specify the algorithm to be used to compute the measurement.
  - We do not expect this to be persistable, *although we would really like it to be.*

A TrackCluster can create a TrackerHit when asked. There are two models for use:

- For each TrackClusterData, instantiate a TrackCluster using a null helix and add it to the event. People can drive their pattern recognition on this. Create new TrackClusters using helix information as it becomes available.
- Start up my pattern recognition code, create some TrackCluster for the hits I am interested in. Make more TrackCLusters as needed, including helix information when available.
- In both cases:
  - Once I have decided that a hit belongs on a track, I can create a trackerhit for it. We also need to create whatever sort of hit is needed by the track fitter being used.
  - We do not modify the objects in the event.

When I am finished with the pattern recognition, I will have a list of tracks, each of which will have a list of associated TrackerHits.

- *This inverts the original purpose of TrackerHit: it is now the end product of pattern recognition, not the input! It looks to me as if TrackerHits are now only useful for graphics and perhaps some monitoring.*
- *I am a little worried about the resource usage of this model, both memory and CPU, compared to a model in which clusters are much simpler objects. But we can go ahead with it for now.*

We still need a solution for bookkeeping which clusters are still available for use. While a TrackCluster object does keep a List of all TrackerHits that it created, this is not the correct bookkeeping solution:

- Objects in the event should be immutable.
- If a TrackCluster is not in the event, then the record is incomplete.
- There is no record of who is using it.
- We do not need to solve this now but we need to keep it on the list of things to do.

Someone pointed out that the TrackCluster has time field and we asked why. All of the cluster classes discussed today have it. In retrospect we are not sure that it belongs.

- Time does have meaning at the SimTrackerHit level.
- For strips and pixels it has no meaning at the RawTrackerHit and cluster level.
- For a TPC has meaning both at the RawTrackerHit and cluster level (it encodes arrival time, and is not synchronous with the beam crossings so it should be a double, as it is in Dima's class).
- For a clocked CCD the situation is less clear. Time does have a meaning: if you assume a different beam crossing, you get different solutions to the position in the clocked dimension. *It's not clear to me that you can even do clustering without assuming a t0.* Maybe it makes sense to return the assumed t0 with a getTime() method? In which case it should be an integer. *I don't see any meaning for a double getTime().*

We do not have a mechanism for recording the configuration information for the clusterfinder and TrackCluster code: what thresholds were used, what algorithms were used to form clusters, what algorithms were used to compute position once the track was associated.