Design changes for Detector Classes

The new design is only a minor change relative to the old, but I think it naturally handles the above concerns. Specifically, I've worked out how this modified design could be used in a Kalman filter fit involving both SVT and DCH hits. I think the design is also free enough to allow a natural description of the DIRC, probably closer to the implementation that's currently in the release. The new design also has a specific scenario for interacting with the SVT alignment, and for implementing a non-gaussian fit (though I don't discuss those here). I don't know how these ideas affect the DIRC, but I'm hopeful that they make your life easier, not harder, Gautier.

The basic change I'm proposing is to remove the 1<->1 association between a DetElem and a surface. Instead I've just specified the TRACKING INTERFACE to DetElem, leaving the implementation free. This interface satisfies all the needs of a Kalman fit (and I hope other fits) without ever making specific reference to a surface. We may want to put this interface 1 level below the DetElem (as a TrkElem or some such), depending on how things look with the DIRC. Clearly the DIRC would not want to implement all the below functions.

The implmentation of the SVT DetElem would still be a surface. I propose the DCH DetElem be modeled as a pair of surfaces. The DIRC could be simply a raw transform. The higher-level organization (DetSet and the classes that build them), as well as the lower-level DetType class, would remain as they now are.

The DetElem tracking interface would be the following (plus the ususal simple access functions, and any concatonation of the below which turn out to be useful):

1. HitTraj(HOT_info) This function creates the trajectory representing a single HOT. The HOT info is as follows:
    1. An integer flag (=phi/Z strip flag for SVT, DCH wire number)
    2. A floating value (=measurement for SVT, null for DCH)
2. Residual(TrackTraj,HOT_info) This function takes a track trajectory and combines it with the information in a HOT to find the residual. It calls the HitTraj function to get a hit Trajectory, then uses the geometric DOCA class. For the SVT DOCA is the residual, for the DCH we have to subtract the measured drift distance.
3. ResidualVector(TrackTraj,list of HOT_info). This function computes the residual as above for a list of HOTs, and combines them into a vector for parametric manipulation. For the SVT there would in general be 2 entries in the list, one for each of the two readout views. For the DCH there would only ever be one entry.
4. ResidualCovariance(list of HOT_Info). This function converts the measurement error into a covariance matrix. Correlations between the views in the SVT (coming from delta-rays for instance) could be applied here. Again the DCH only returns a 1X1 matrix. The result is ready for parametric manipulation.
5. Jacobian(TrackTraj,list of HOT_Info) This function computes the jacobian of the standard 5-parameter helix with respect to the measurements implied by the HOTs in the list. This will return a 5XM matrix, where M is the number of measurements (=1 for DCH)
6. Transport(TrackTraj,range limit) This function computes the 5X5 transport

matrix accounting for dE/dX and B-field irregularities in this element. The range limit is ingnored in the SVT, but is used in the DCH to determine the amount of gas traversed.

7. ScatMat(TrackTraj,range limit) This function computes the 5X5 scattering (and dE/dx uncertainty) matrix for this element. Again the range is ignored by the SVT implementation, but could be used by the SVT.
8. KalHit(TrackTraj,list of HOT_info) This function creates a single KalHit KalSite corresponding to the HOTs in the list. This would only be called with 1 list entry for the DCH, the SVT could have 1 or 2. This must NOT be called with independent hits in the same DetElem (for instance, multiple hits in the same DetElem from the top of a looping track in the DCHt cannot be combined here).
9. KalScatter(TrackTraj,trajectory range) This function combines the Transport and ScatMat functions above to create a single KalScatter KalSite.
10. KalSites(TrackTraj,list of HOT_info) This function combines the preceeding functions to create an ordered list of KalHit and KalScatter KalSitess from the list of HOTs and the trajectory. This is a 'smart' function (the only one so far), in that its behavior depends strongly on the input.
    1. For the SVT, this would be called with at most 2 hits for the 2 views. It would return one KalHit and one KalScat sites.
    2. For the DCH, this would be called with all the consecutive hits in a given DetElem. For instance, if there are 5 hits in the top arc of a spiral in a single layer, all 5 get passed in here.
        1. If there's a single DCH hit, it returns one KalHit and one KalScat, where the scatter corresponds to the total material encountered in traversing the layer (IE the gas between the 2 cylinder surfaces).
        2. If there're more than 1 DCH hit, it returns a KalHit and a KalScat for each one. The KalScat is computed using the material encountered along the trajectory moving from the DOCA of one hit to the next.
11. FlightDistance(TrackTraj) This function returns the flight distance to the 'intersection' point of the trajectory with itself. This could be the center of the DCH layer for example. This is used by the navigation function below.

In addition to the above DetElem functions, there is one tracking interface function to the DetSet (collection of DetElems). This is:

1. DetElemList(TrkTraj,range limit) This function returns an ordered list of pointers to detector elements (along with the flight distance to each of them) which are encountered by the trajectory within the specified flight distance range.

The above functions provide a sufficient interface for building a SVT+DCH Kalman fit, according to the following scenario:

1. The fit must be seeded by an ordered list of HOTs and an initial trajectory. the HOTs must know which DetElem they come from.
2. The DetElemList function is used to generate the list of detector elements

encountered by the initial trajectory.

3. Make sure that all the DetElems pointed to by the HOTs are on this list. If there's a missing DetElem, add it in sequence (according to the FlightDistance intersection).
4. The list of DetElems is iterated. for each one we do the following:
    1. Find all the adjacent HOTs coming from this element (would be a null set if the DetElem is a passive piece of material)
    2. Pass this sub-list of HOTs to the KalSites function of the DetElem. The returned sub-list of KalSites is appended sequentially to create a global KalSite list
5. The KalSite list is now Processed (sequential call to the Process function of every KalSite). This fits the track.
6. The track can be refit (re-linearized) by first Updateing the sites (sequential call to the Update function of each KalSite) and then re-processing them. This action doesn't change the number or order of the KalSite list.
7. The track KalSite list can be verified to make sure the final trajectory gives the same KalSite list. If the list is found to differ, the track can be re-fit starting from scratch, but using the (compound helix) trajectory of the last fit as a seed.

Clearly there's a lot missing from this list, but it's sufficiently complete to make me believe it's a workable scenario.