Status of new tracking data model
“End-user” Classes

Jose Hernando (CERN), Eduardo Rodrigues (NIKHEF)

In short ...

Driving idea:
- agree on the interfaces
- move the code adiabatically in steps
  - code always working, smooth implementation of the interfaces

Visible to the user:
- tracks & states
- propagators

To help the tracking/pattern recognition developers:
- nodes & measurements
- projectors

Details on status of implementation:
http://cern.ch/eduardo.rodrigues/lhcb/tracking/event_model/index.html

( note: place of evolving ideas/implementations ... )
**Working plan**

1. **Reach agreement on TrTrack, TrState, extrapolators interface**
   - **GOAL:** standard output of all fitting algorithms, online & offline
     - client interface: how to use the tracks, ...
     - minimal interference with present code
       - e.g. leave measurements untouched at this stage
   
   - **STEPS:**
     1a) reach final agreement on this set of classes (today?)
     1b) implementation of converters
       - TrFitTrackToTrTrackCnv & TrgTrackToTrTrackCnv
         - needed to make sure the TrTracks work!
     2a) make TrFitTrack & TrgTrack inherit from TrTrack
       - at this point we could already imagine some full MC production with new model!

   - **TIMESCALE:**
     1a) end of the week
     1b) 1 week after agreement (implementation + test)
     2a) 2 weeks
Working plan

2. Reach agreement on TrMeasurement, TrNode, projectors interface

- GOAL: have some common base classes for pattern recognition and fitting algorithms
  - re-use as much as possible the existing code and packages
  - Tr(g)Event minimally/not touched
  - end-user/client code is unchanged!

- STEPS: 1a) get some first agreement on proposed structure
  - projectors as Gaudi tools, need for TrMeasurement & TrNode classes
    1b) re-build all necessary info from persistency
  2a) XxxClusterOnTrack replaced by TrNode,
    XxxCluster derived from TrMeasurement
  2b) pattern recognition and fitting algorithms make full use of projectors and new measurement classes
**TrTracks (1/2)**

### Data members

- **m_type**: OK. Related enums are no problem to persistency
  - variation: make use of bitfields

- **m_historyFlag**: OK. Enum.
  - variation: change the name?

- **m_errorFlag**: renamed to **m_flag**. Uses bitfields with contents = { valid, ... }

- **m_chi2**: becomes **m_chi2PerDoF**
  - more commonly used

- **m_nDoF**: keep it?
  - variation: remove it
    - pros: saves space, but needs some calculations to get it back

- **m_closestState**: state closest to beam-line, to be persistent
  - more commonly used
  - variation: call it **m_firstState**

- **m_states**: proposal to have a **SmartRefVector**
  - persistency on demand: subset of states to be persistified (Beam-line, TT, T)
  - variation: use a std::map with location as key (e.g. AtTT)
Public methods

- end-user methods for fast/easy access to general track information
  - e.g. StatusCode **positionAndMomentum** (double z, **ITrExtrapolator** *extrapolator,
    ParticleID pid, HepPoint3D &pos, HepVector3D &mom, HepSymMatrix &cov6D)

- **methods to get direct access to closest state, by z-pos or plane**
  - commonly used methods in tracking code
    - pros: friendly and simple interface

- **methods with extrapolator interface as argument:**
  - pros: hide repeated operations
    (e.g. aTrack->closestState()->extrapolate() )
  - variation: move these methods to the extrapolator interface

- virtual methods: # brought to a minimum
  - closestState(z/HepPlane3D), reset(), clone(), producedByAlgo()
TrStates

Data members
- m_type: enum {StraightLine, HasMomentum} Fine
- m_location: enum key for state locations (e.g. AtTT, BegRICH1, ...)
- other data members: z-position, state vector, covariance

Public methods
- # of "setter" methods minimized
- methods related to position/slopes: implemented
  - no need to be virtual
- methods with Q/P or momentum: all virtual

Derived classes
- 2 strategies to derive from TrState:
  - TrStateL -> TrState (Q/P and 5 components)
  - TrStateL, TrStateP -> TrState
**Tools - extrapolators**

**ITrExtrapolator**

**Public methods**

- extrapolators to z and HepPlane3D
  
  ```cpp
  virtual StatusCode propagate (TrState *state, double z, ParticleID partId=ParticleID(211))=0
  ```

- **Issue**: need for an extrapolation to some kind of surface

  - How to define a surface?

    Can use for the moment a typedef to HepPlane3D in a "Surface" class

- **interface** ITrExtrapolator defined and essentially agreed ...

**TrLinearExtrapolator**

**Public methods**

- already implemented

- serves as (simple) example