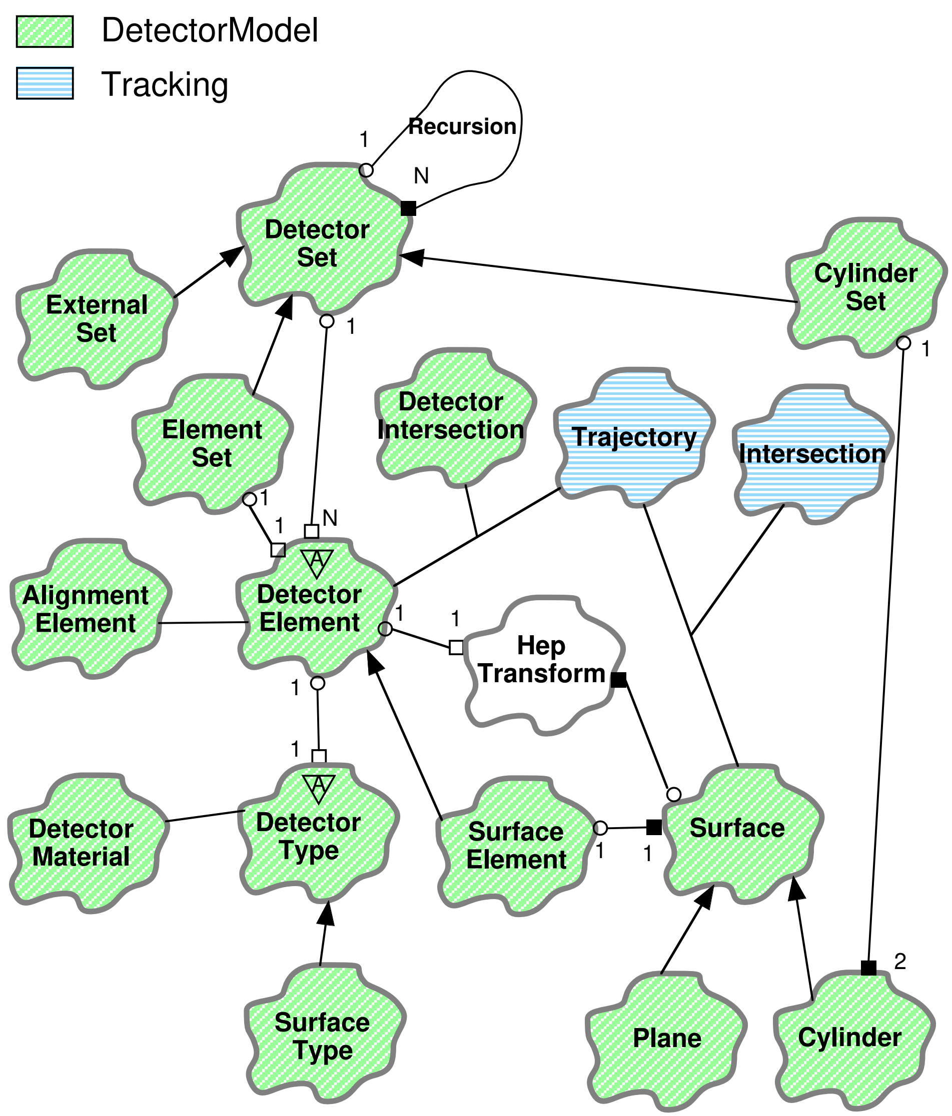
DetectorModel Package

The DetectorModel package provides base classes for describing the geometry and material of HEP detectors. It provides interfaces and services for tracking, nominal geometry, alignment, and material effects. This package will be used by the SVT, DCH, and DIRC and perhaps other subsystems to describe their detectors. This presentation provides a brief overview.

- DetectorModel Classes
- Geomety and Material Interface
- Tracking Interface
- Alignment Classes and Interface
- SvtTrack Package Objects
- Proposed AbsEnv Database Interface

DetectorModel Classes



DetectorModel Classes

- The classes are transient
  - Not stored in the database (but can be built from such)
  - Accessable only after being built by a module
- The classes form a hierarchical tree
- The 'leaf' class is DetectorElement
  - A pure virtual base class
  - 1 class for each catagory of physical component
  - 1 class object for every physical object, describing the position, orientation, shape, size, and composition
  - General subclasses can be useful: eg SurfaceElement
- The 'branch' and 'trunk' class is DetectorSet
  - Contains a collection of DetectorElements
  - Contains a collection of DetectorSets
  - Provides a high-level tracking interface
  - Subclasses provide more sophisticated features
    - CylinderSet knows that its contained elements are arranged in a roughly cylindrical fashion (will be extended soon to SurfaceSet)
    - ElementSet has a 'special' relation with one element
    - ExternalSet is outside the tracking volume
- 'Leaf' internal structure is defined in the (pure virtual) DetectorType class

Geometry and Material

- The external geometric description of an element uses HepTransforms (CLHEP)
  - Every DetectorElement references a transform.
  - Transforms describe the position and orientation of objects in the BaBar coordinate system.
  - The transform also translates space points into 'local' points (DetectorCoordinate) and vice-versa.
  - Surfaces (planes and cylinders) are implemented using transforms.
  - The same transform is used to describe passive (material) and active (hits) aspects of an element.
  - Alignment methods act on this transform.
- The internal geometric description of an element can be 2-D or 3-D
  - Interface (DetectorType) is mostly virtual.
- Material properties are described by DetectorMaterial helper class.
  - Multiple scattering, energy loss, etc.
  - Collaboration with Geant4 on these is sensible

Tracking Interface

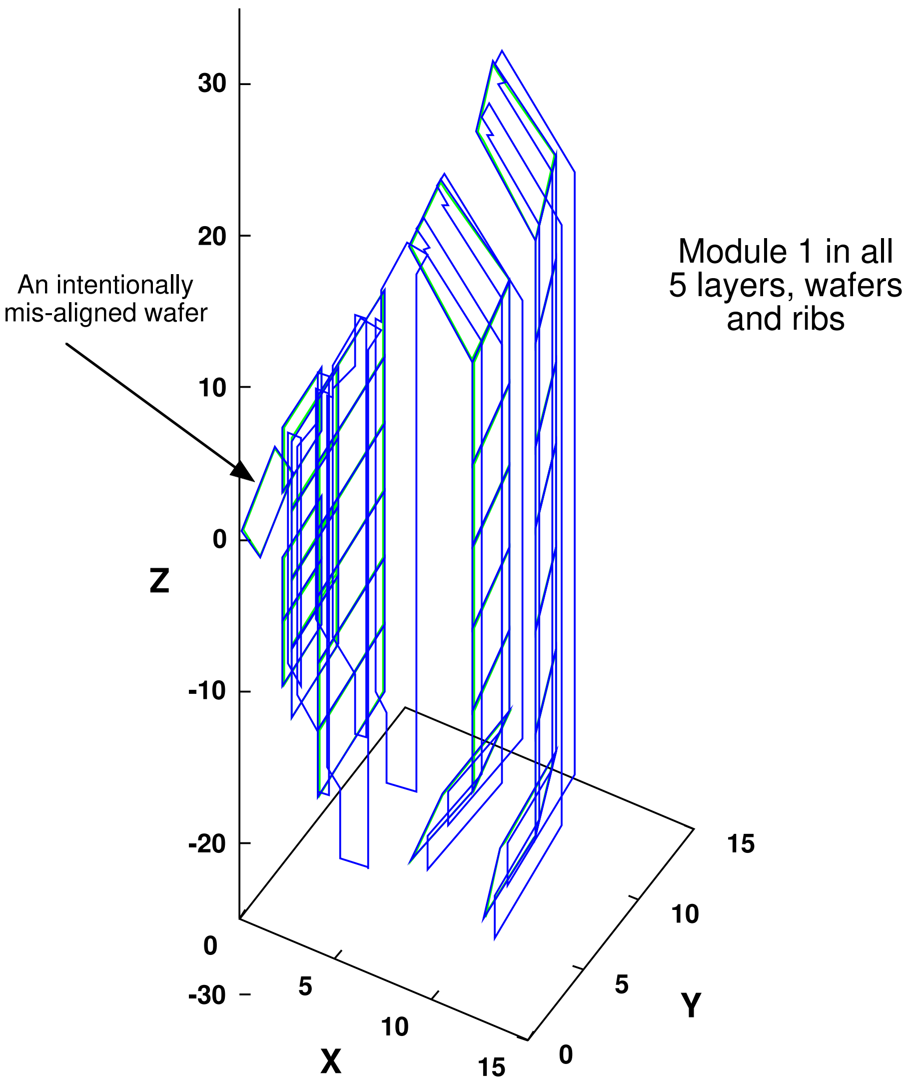
- The tracking interface is through Trajectories
  - A Trajectory defines a 1-D object in 3-D space.
    - A Track must provide a trajectory.
    - A hit must provide a trajectory.
    - Residuals are defined as the distance between 2 trajectories.
  - A DetectorElement provides methods to determine if and where a (track) trajectory intersects it.
    - DetectorIntersection describes entrance, exit, and effective interaction point as path lengths along the trajectory.
  - A DetectorElement can describe the material effects (multiple scattering, energy loss, etc.) corresponding to a DetectorIntersection.
  - DetectorSet provides navigation methods.
    - Next intersection (next DetectorElement intersected).
    - All intersections (within a specified path length range).
    - Daughter classes (CylinderSet) are designed to do this very efficiently: Navigating the SVT (~500 elements) takes ~ 15msec*.
- Active elements provide methods to translate a measurement into a hit trajectory.
  - Reconstruction details are hidden, as measurements are described in TypeCoordinates.
  - DetectorType objects might be shared with ‘clustering’.
- Kalman tracking uses the material interface
  - Used in KalMaterial constructor and functions.

* on a DEC ALPHA 250 MHz- model V

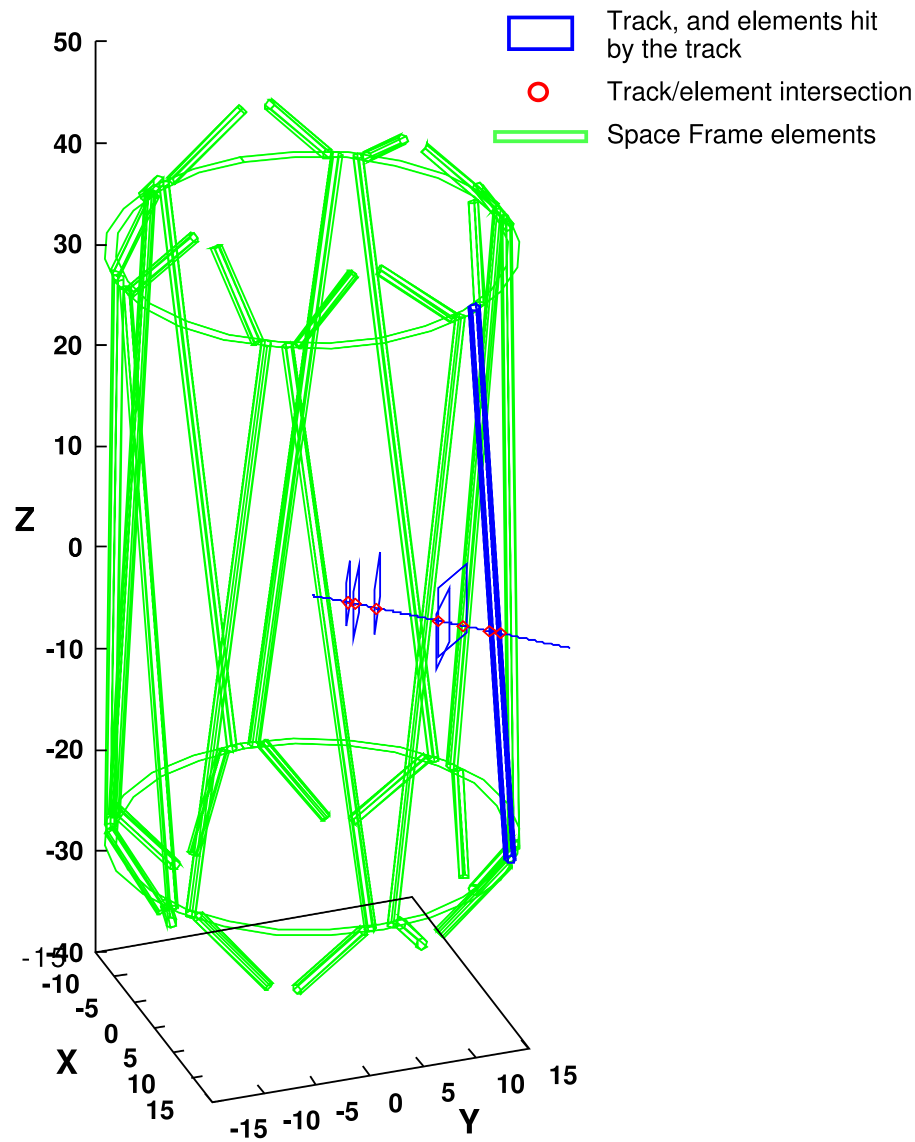
Alignment Interface

- DetectorModel implements alignment using the AlignElement class.
  - Parameters represent small deviations from nominal.
  - A HepTransform can be constructed from these.
  - Alignments can be *applied* and *removed.*
    - Apply sets the element transform to the product of the existing transform and the alignment transform. T *= A.
    - Remove does the same for the inverse alignment transform.
  - Alignment can be applied *globally* or *locally* .
    - global alignment multiplies the element transform on the left side.
    - local alignment multiplies the element transform on the right side.
    - The AlignElement ID must match exactly for local alignment.
  - Both global and local alignment can be applied through DetectorSet.
    - Global applies a single alignment to all contained elements.
    - Local takes an array of alignments, one for each DetectorElement.
- The aligned position of a DetectorElement is implicit in its transform.
  - Hit trajectories and material navigation are ‘automatically corrected’ for alignment with no overhead.
- No alignment bookkeeping is kept internally.
  - Avoids inefficient checks (efficient checks are outside).
  - Avoids problems when computing alignment.

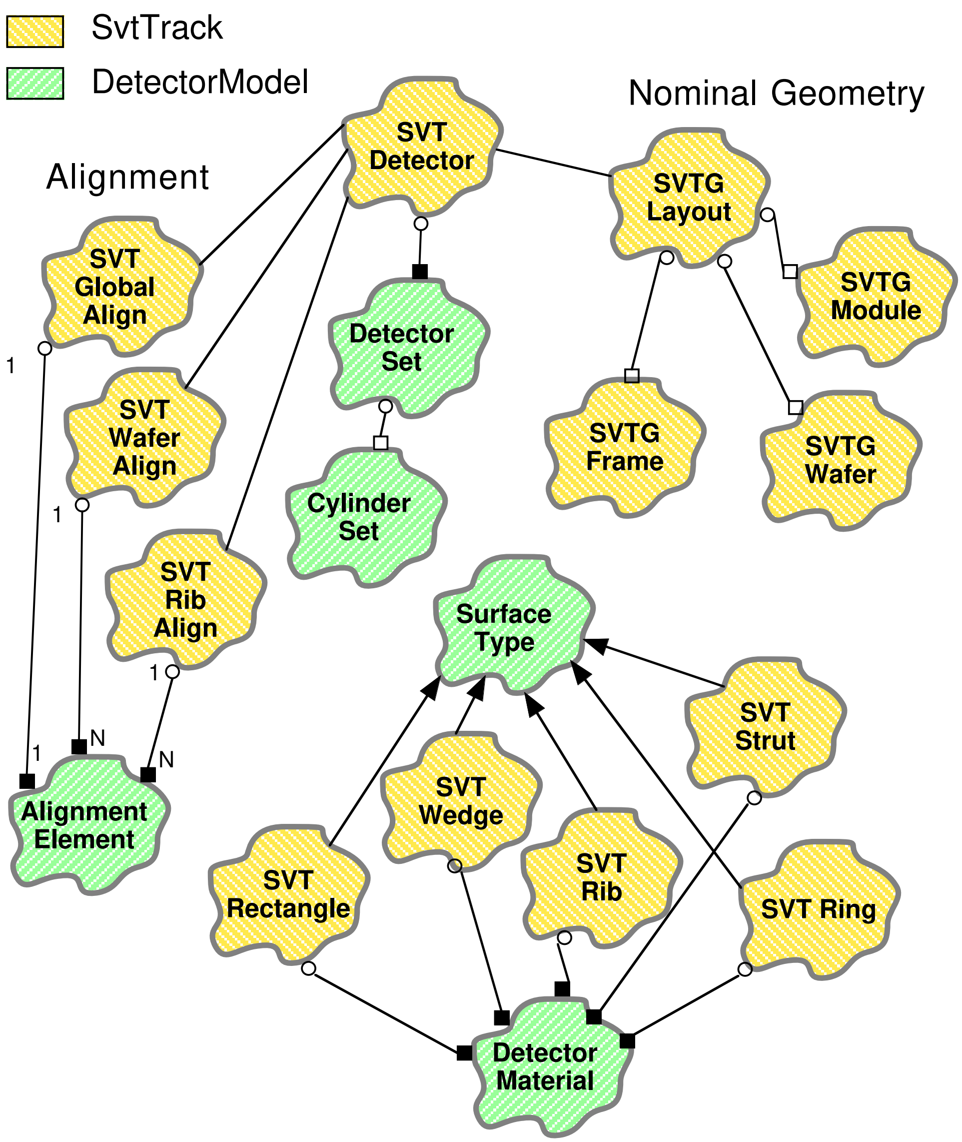
Svt Model



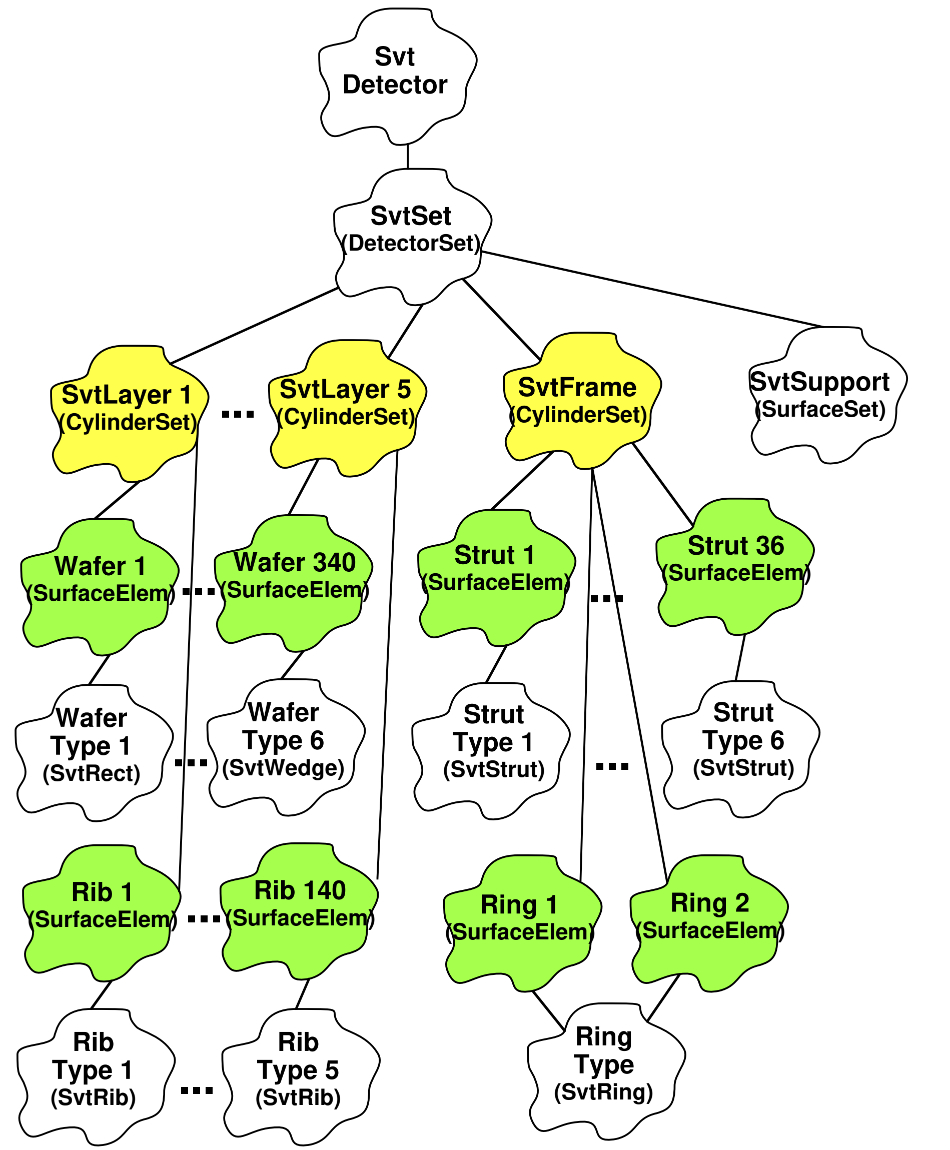
Svt Model and Tracking



SvtTrack Classes



SvtTrack Transient Objects



SvtTrack Classes and Objects

- Svt has only SurfaceElement objects.
  - Wafers, ribs, and rings are based on planes
  - Strusts are cylinders
  - Specific behaviors are given by the DetectorTypes
- A singleton object SvtDetector owns the entire DetectorSet tree.
- A CylinderSet is used for each layer
  - Holds ribs and wafers
  - Speeds up track navigation by 5X DetectorSet
- SvtDetector is constructed from nominal and aligned geometry information (classes)
  - SvtDetector class can also work for testbeam
  - SvtG* classes describe the nominal geometry in a hierarchical, minimal way
    - classes are mainly data, with almost no methods
    - constructed from an ascii file (today)
    - can be stored in/read from the database ('tomorrow')
  - Alignment objects are collections of AlignElements
    - Global describes the Svt Global position (1 AlignElement)
    - Ribs can also be individually aligned
    - Wafers and Ribs alignments are in different objects (different generation methods, update frequencies)
    - can be stored in/read from the database

AbsEnv Interface

- AbsEnv is being re-implemented using the ProxyDict of Ed Frank
  - Database accesses is hidden behind proxies
  - Proxies stand in for transient objects
    - They hold a pointer to the actual object
    - if the pointer is null, they call a fault handler to build the object
    - Users request the transient objects through their proxy
- SvtDetector is the (proposed) proxied object for the Svt DetectorModel.
  - The fault handler builds the SvtDetector object if none already exists.
    - The fault handler knows which database objects are needed to construct the SvtDetector object, and can retrieve them.
    - These are passed to the SvtDetector constructor.
  - The fault handler updates the SvtDetector every event.
    - Each event, the first access to the proxy invokes a fault handler.
    - If the database objects have not changed since last read, the cached pointer to the old SvtDetector object is returned.
    - If any database objects have changed, SvtDetector is updated.
      - Global alignment changed -> remove old alignment and apply new.
      - Local alignment changed -> remove old alignment and apply new (?)
      - Nominal geometry changed-> rebuild SvtDetector from scratch
      - The proxy counts the number of apply/remove cycles. If this gets too high, there is a risk of loss of numerical precision, and the the SvtDetector object is rebuilt from scratch.

Conclusions

- The DetectorModel package being used to implement the material/geometry model of several BaBar subdetectors
  - SvtTrack
  - DrcGeom
- The package uses existing code.
  - CLHEP transforms.
  - RW template libraries.
- The package provides a standard interface to tracking.
  - Material navigation.
  - Hits.
- The package provides a standard implementation of alignment.
- Collaboration with Geant4 on a material description class is desireable.
  - A working prototype version is provided as a stopgap.