Alignment* framework status

- **Working definition**
- **Three ways**
- **The third way**
  - –Idea
  - –Status
  - –Details
- **Summary**
  - –What is there
  - –What is not there
- **Implicaitons/Points for discussion**

Juan P. Palacios
CERN

***AKA Geometry framework**

(Current) working definition

- **Provides users of detector geometry description with geometrical information derived from both a nominal alignment of all the sub-components of LHCb, and deviations from the nominal alignment**
- **Provides a mechanism for users to modify the deviations from the nominal alignment for a given sub-component and ensures that the modifications are propagated coherently to the geometry description**
- **Provides a mechanism to write a new set of deviations to the conditions database**

(Current) working definition / 2

- **Says nothing about alignment strategies, algorithms or their implementation**
- **Provides the basic functionality needed for such algorithms to function**
- **This definition could evolve**
  - Commonality between different alignment algorithms/tasks?
- **“Alignment framework” misleading, “Geometry framework” a bit general**
- **Maybe we should think of (yet another) name for this…**

Usage of detector geometry service

- **Typically, calls to DetectorElement::geometry() methods**
- **Returns IGeomertyInfo* which has local to global and global to local transformations and a host of other services**
  - See **http://lhcb-release-area.web.cern.ch/LHCb-release-area/LHCB/doc/html/class_i_geometry_info.html**
- **Is there any usage** other than this? Speak now!

Yes I know, position of things at the simulation level Is not necessarily coupled to the detector element of each volume. Will discuss this later.

Notes on detector geometry usage

- **The local to global and global to local require traversing geometry tree from branches towards trunk**
- **This means state of IGeometryInfo and DetectorElement could depend on state of *parents***
- **This has implications for any update mechanism!**

Three ways I: three stores

- **Idea from Sebastien P.**
- **Will Bell has tried hard to make it work**
  - Complications!
- **After some time trying to understand the way the (DetectorDataSvc) stores work, I capitulated!**
- **For details, see talk of W. Bell**

Three ways II: move PVolumes

- **Both VELO and RICH run alignment studies using PVolume::applyMisalignment**
  - Very handy but requires DISTINCT PVolumes in mother volume of interest
- **Example: VELO with distinct ½ stations: can only really move ½ stations, not sensors within.**
  - PVolumes are only moved within their mother LVolume. If this is replicated, the transformations are also replicated
- **Would need either one master volume (VELO) with individual sensors (or modules):**
  - NO GRANULARITY FLEXIBILITY: stuck with one parent and set of daughters
- **Or different LVolumes for each PVolume:**
  - Applicable only for moderate numbers of volumes
  - Done for VELO modules and sensors
  - Might be only way of **misaligning in the simulation**

Three ways II: PVolumes

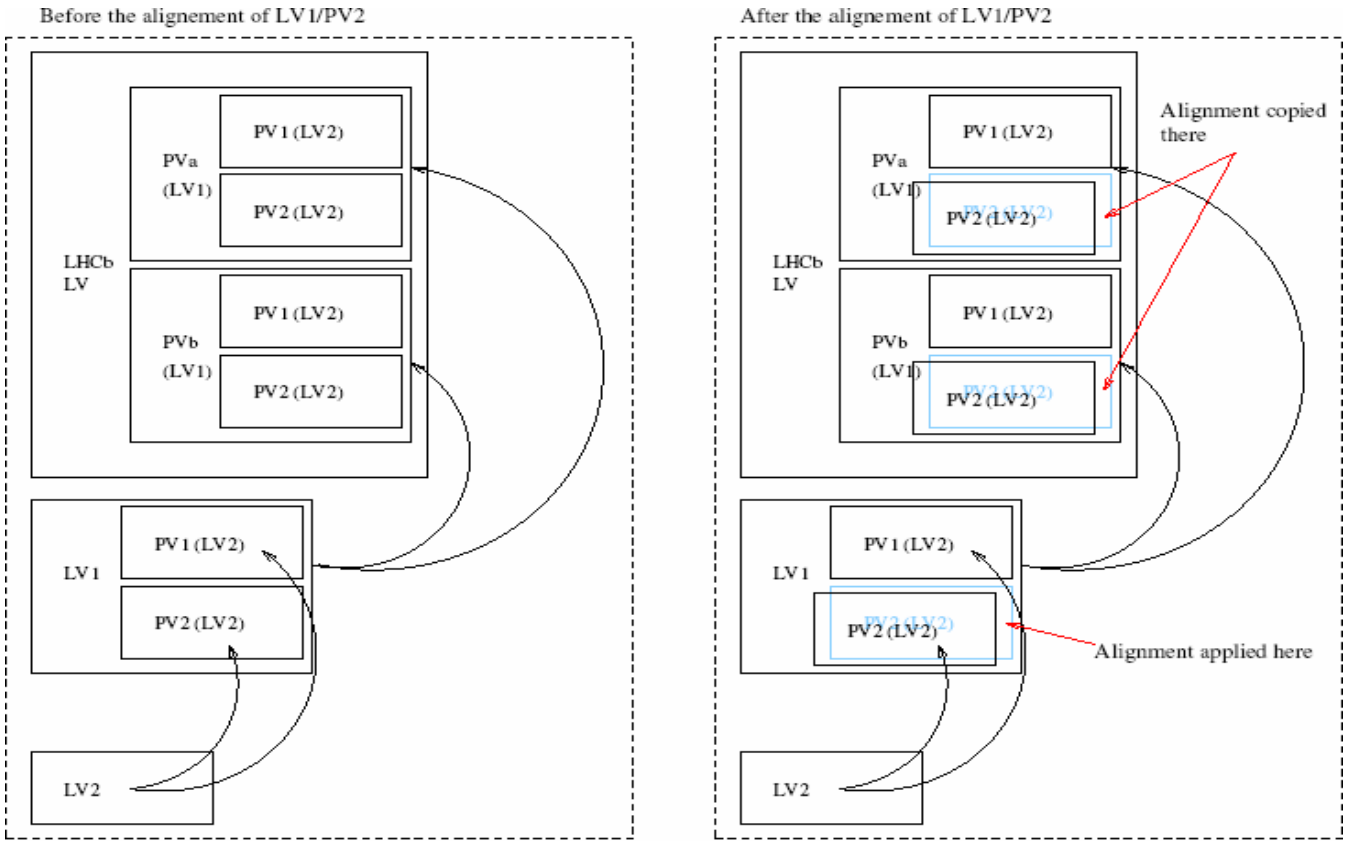

Diagram showing problem with assigning transformations to the PVolumes (Marco Clemencic)

Three ways II: PVolumes

- **With distinguishable PVolumes, approach would be:**
  - Have one detector element (can be default, just need a detelemref and support path) per ***alignable*** sub-component (to handle its own and parents geometry information
  - Have one physical volume corresponding to each ***alignable*** sub-component
  - Assuming the structure already exists in the geometry description. This must also be supplied.
  - Have a “conditions” tree with one-to-one mapping to structure (DetElem) tree: one transformation per detector element
  - Assign the address on this tree to the relevant PVolume

Three ways II: PVolumes

- **This would allow misaligning in simulation and to see effects directly in Panoramix**
  - But overlapping volumes etc. would have to be carefully taken care of
- **Implications:**
  - Potentially a lot of XML to write
  - A lot of volumes created in memory

Three ways III: Detector element

- **Basic idea (from Pere's suggestion):**
  - Give the transformations to the Detector Element and let it do the work
- **Two limits:**
  1. One detector element per aligneable object. Needs basically one transformation.

     THIS IS MY FAVOURED APPROACH!
  2. A detector element that knows enough about its PVolume daughters (and their daughters, and their daughters' daughters, and their daughters' daughters' daughters, and …) to assign the correct transformations to each

     MESSY IN ALIGNMENT CONTEXT, BUT MAYBE NECESSARY FOR OTHER CONDITIONS LIKE CALIBRATIONS?

Three ways III: detector element

- **Keep some ideas from PVolumes exercise**
  - "Tree" of conditions
  - One detector element per alignable component
  - One-to-one mapping between the two
- **No longer care about individual PVolumes or LVolumes or whatever**
  - Avoid drastic changes to XML
    - Use "templated" detector element files
    - Small changes for conditions tree
    - VELO examples available
  - No changes to LHCb detector geometry description philosophy
  - Few changes to DetDesc and DetDescCnv code

The third way: status

- **XML conditions tree**
- **AlignmentCondition class**
- **Possible implementation: AlignmentInfo**
- **Missing:**
  - Real CondDB or data store access
  - Update mechanism
  - CondDB write mechanism

3rd way status: conditions tree

- **Test conditions in XML file.**
- **structure.dtd already had some useful stuff**
  - Only necessary to add an

    `<alignmentinfo condition="somePath" />`

    in each <detelem …. /> definition
  - Then store the condition somewhere:

```
<condition classID="6" name="somePath">
     <paramVector name="dPosXYZ" type="double"> 0. 0. 0.</paramVector>
     <paramVector name="dRotXYZ" type="double"> 0. 0. 0.</paramVector>
     <conditionref href = "#someBranchPath"/>
</condition>
```

Added conditionref to DTD to allow hierarchical construction (conditions can have conditionrefs now)

3rd way status: AlignmentCondition

- **Specialisation of Condition class**
  - ParamValidDataObject
- **Returns transformation matrix and inverse relating position of associated detector element to mother**
  - Matrices constructed from six parameters in data store
- **Has no knowledge of parents or daughters, only a transformation**
  - Simple, atomic condition
- **Being DataObject fits easily into update mechanism (ask Marco!)**

Creation of an AlignmentCondition

- **At the moment, via XML and dedicated “converter”**
- **classID = 6 maps to XmlAlignmentConditionCnv**
  - This is a little template which simply instantiates the right kind of condition
- **AlignmentCondition has access to the parameters and constructs the matrices**

```
<condition classID=“6” name=“/dd/Conditions/LHCb/myDetector/Module67/”>
     <paramVector name=“dPosXYZ” type=“double”> 0. 0. 0.</paramVector>
     <paramVector name=“dRotXYZ” type=“double”> 0. 0. 0.</paramVector>
     <conditionref href = “#Plank82”/>
</condition>
```

- **User accesses condition via data service:**

```
SmartDataPtr<AlignmentCondition> cond(datasvc(),
                    “/dd/Conditions/LHCb/myDetector/Module67”);
```

AlignmentConditions in the det. Elem

- **Use AlignmentInfo. Why?**
  - Pointer exists in DetectorElement class
  - Mechanism to instantiate from XML exists in XmlDetElemCnv: only need to include the following in detelem definition:

```
<alignmentinfo condition="/dd/Conditions/LHCb/myDetector/…" />
```

  - In is a public ConditionInfo and a virtual public IAlignment!
    - This I don't care about! We should review the role of some of these interfaces
    - In fact, ConditionInfo and IAlignment BOTH are IInterfaces. Is this safe?
- **Real answer: "because it is there"**
  - But this is temporary, the functionality is, or should be, limited and should possible be implemented elsewhere…

What does AlignmentInfo do?

- **Relevant public interface:**

```
virtual HepPoint3D toLocal(const HepPoint3D& pt) const;
virtual HepPoint3D toGlobal(const HepPoint3D& pt) const;
Virtual HepPoint3D toMother(const HepPoint3D& pt) const;
```

- **Transformations to and from local to global (LHCb) frames**
- **Transformations to and from local to frame of parent volume**
- **Also accessor methods to get corresponding matrices**

These are available to the detector element and to its users

How does AlignmentInfo do what it does?

- **Constructor uses path to get AlignmentCondition**
  - "mother" matrices obtained directly
  - Maybe should not be done in constructor (facilitate updates)
- **Scans down tree picking up parent transformations**
  - Iterative procedure finding parents from IDataManagerSvc and IRegistry (tree navigation)
  - Constructs the global-to-local martix
- **"Local" matrix can be updated by user**
  - Re-calculates "global" matrix automatically
  - At the moment this is de-coupled from automatic update mechanism

How do we want to use this information?

- **This is really a question to users of the detector description**
- **Currently the “ideal” transformations are obtained from the geomerty() methods of the DetectorElement PI.**
- **Do we want to keep this interface?**
  - Do we want geometry() to point to ideal?
  - Do we want geometry() to point to ideal+deviations?
- **It should always be possible to obtain ideal and deviations separately. Question is how do we want to handle the merged information.**
  - Intuitively one would make geometry() return ideal + deviations and provide separate accessors for ideal and deviations

How should this be implemented?

- **At the moment I keep all deviations in AlignmentInfo.**
- **Client code most likely uses geometry() which returns IGeometryInfo***
  - Would need to propagate deviations into GeometryInfo
  - Or make AlignmentInfo be an IGeometryInfo?
- **Do we really need all these ISomethingInfos??**
- **AlignmentInfo is not data object**
  - Problems to implement updates?
- **Can we use the opportunity to break things now and clean up DetectorElement and related interface?**
  - I hope so because there is a lot in there that I don't like!

Requirements from detectors

- **A detector element per alignable object**
  - Must have path to an alignment condition
  - In current scheme this means an <alignmentinfo/> with the correct path in the XML
- **A hierarchical tree of conditions with one-to-one mapping to the detector element tree**
- **A geometry description where the alignable objects can be associated to an LVolume**
- **We don't even consider the intelligent detector element case here. One-to-many mapping not implemented…**

All this has been implemented and tested for the VELO (missing ½ velo volumes)
Contact me for details!

Summary: what is there

- **XML conditions tree**
  - Changes to structure.dtd condition definition and XmlBaseConditionCnv to allow for conditionrefs inside conditions
  - VELO example tree of references
    - Working for now with dummy XML alignment parameter vectors
- **AlignmentInfo class**
  - Instantiated in DetectorElement from XmlBaseDetElemCnv
  - Modifications to both the above to pass condition address
- **AlignmentCondition class**
  - Assigned **classID 6** and “Wrote” XmlAlignmentConditionCnv
  - Instantiated in AlignmentInfo constructor using path to get alignment parameters from (XML based for now) data store.

We can get the right transformations to the detector element

Summary: what is not there

- **Access to conditions via Marco's data store interface (only dummy XML for now)**
- **An update policy!**
- **How to actually use the information and give it to users**
- **How to modify alignment in an algorithm**
  - Presumably a "set" method for AlignmentInfo?
- **How to write alignment parameters to CondDB. In fact, how to write them anywhere at all!**
- **Surely I must have forgotten something?**

Points for discussion

- **Does this approach make sense?**
  - Are there any problems?
  - Is any obvious functionality missing?
- **How do people use and intend to use the "ideal+deltas" alignment information?**
  - Will anyone need "ideal"? And "deltas" only?
  - How is DetectorElement used in tracking, alignment?
- **Update policy when using information coming from parents in tree**
  - Can dependencies be somehow factored out ?
  - Some re-design of the providers of "toLocal" and "toGlobal" functionality?