Detector Description in LHCb (Extended Version)

Detector Description Workshop
4 July 2002
S. Ponce - CERN
Contents

- Gaudi Architecture Overview
- Transient Store Mechanism
- Detector Description
- XML Persistency
- User extensions of the schema
- Visualization
- Simulation: Interfacing Geant4
- Condition Database
Definition of Terms

- **Algorithm**
  - Atomic data processing unit (visible & controlled by the framework)
  - Written by physicists, Called once per physics event
- **Service**
  - Globally available software component providing some functionality
- **Data Object**
  - Atomic data unit (visible and managed by transient data store)
- **Transient Store**
  - Central service and repository for objects (load on demand)
Interfaces
Interfaces in Practice

IMyInterface.h

class IMyInterface {
    virtual void doSomething( int a, double b ) = 0;
};

ClientAlgorithm.cpp

#include "IMyInterface.h"

ClientAlgorithm::myMethod() {
    // Declare the interface
    IMyInterface* myinterface;
    // Get the interface from somewhere
    service(“MyServiceProvider”, myinterface);
    // Use the interface
    myinterface->doSomething( 10, 100.5);
}
Gaudi Services

- JobOptions Service
- Message Service
- Particle Properties Service
- Event Data Service
- Histogram Service
- N-tuple Service
- Detector Data Service
- Magnetic Field Service
- Tracking Material Service
- Random Number Generator
- Chrono Service
- (Persistency Services)
- (User Interface & Visualization Services)
- (Geant4 Services)
Algorithm & Transient Store

Transient Store

Algorithm A

Algorithm B

Algorithm C

Data T1

Data T2, T3

Data T2

Data T4

Data T3, T4

Data T5

Data T5

Apparent dataflow

Real dataflow
Data Reside In Data Store

- **Tree** - similar to file system
- **Identification by path**
  
  
  
  
  
  "/Event/MCEvent/MCEcalHit"
  "/dd/Geometry/Ecal/Station1"

- **Objects loaded on demand**
Understanding Transient Store Loading

1. Algorithm
   - retrieveObject(…)
   - Try to access an object data

2. Data Service
   - Search in Store

3. Persistency Service
   - Request load

4. Conversion Service
   - Request creation

5. Converter
   - Register

Data Store

Unsuccessful if requested object is not present

Request dispatcher
Oracle, XML, ROOT,..
Detector Description
Detector Description

- **Logical Structure**
  - Breakdown of detectors
  - Identification

- **Geometry Structure**
  - Hierarchy of geometrical volumes
  - LogicalVolumes (unplaced dimensioned shape)
  - PhysicalVolumes (placed volume)

- **Other detector data**
  - Calibration, Alignment, Readout maps, Slow control, etc.
Logical Structure

- The basic object is a **Detector Element**
  - Identification
  - Navigation (tree-like)
- **DetElement** as information center
  - Be able to answer any detector related question
    » E.g. global position of strip#, temperature of detector, absolute channel gain, etc.
  - Placeholder for specific code
    » The specific answers will be coded by physicists
Algorithm Accessing Detector Data

- Manages store
- Synchronization updates

Request

Detector Data Service

Persistency Service

Conversion Service

Geometry

Conditions DB

Other DBs

Algorithm

DetElement

IDetElement

IGeometryInfo

IDetElement

IGeometryInfo

MuonStation

DetElement

IReadOut

ICalibration

Geometry Info

Calibration

ReadOut

Conversion Service

Conversion Service

Conversion Service

Transient Detector Store
Algorithm Accessing Detector Data

// Algorithm code fragment (initialize() or execute())

SmartDataPtr<MyDetElement> mydet(detSvc(),
                           "Structure/LHCb/MyDet");
if( !mydet ) {
    log << MSG::ERROR << "Can't retrieve MyDet" << endmsg;
    return StatusCode::FAILURE;
}
...

// get the number of sub-DetectorElements
ndet = mydet->childIDetectorElements().size()

// get the material
material = mydet->geometry()->lvolume()->materialName();
Geometry Information

- Constructed using **Logical** and **Physical Volumes** (Geant 4)
  - Logical Volume: Unplaced detector described as a solid of a given material (optional) and a set of daughters (physical volumes).
  - Physical Volume: Placement of a logical volume (rotation & translation).
- Solids
  - A number of basic shapes (boxes, tubes, cones, trds, spheres,...) with dimensions
  - Boolean solids (unions, intersections and subtractions)
Algorithm Accessing Geometry Info

IGeometryInfo* geom = mydetelem->geometry();

<table>
<thead>
<tr>
<th>IGeometryInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>HepTransform3D&amp; matrix() // To Local</td>
</tr>
<tr>
<td>HepTransform3D&amp; matrixInv() // To Global</td>
</tr>
<tr>
<td>HepPoint3D toLocal(HepPoint3D&amp; )</td>
</tr>
<tr>
<td>HepPoint3D toGlobal(HepPoint3D&amp; )</td>
</tr>
<tr>
<td>bool isInside( HepPoint3D&amp; )</td>
</tr>
<tr>
<td>string belongsToPath(HepPoint3D&amp; )</td>
</tr>
<tr>
<td>IGeometryInfo* belongsTo(HepPoint3D&amp; )</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>fullGeoInfoForPoint(HepPoint3D&amp;, ...)</td>
</tr>
<tr>
<td>string lVolumeName()</td>
</tr>
<tr>
<td>ILVolume* lvolume() ...</td>
</tr>
</tbody>
</table>
Two Hierarchies
Transient Store Organization

- **Standard Gaudi Transient Store**
  - “Catalogs” of Logical Volumes and Materials
  - “Structure” as a tree
  - All elements identified with names of the form: /xxx/yyy/zzzz
Persistency Based on XML Files

- XML is used as persistent representation of the Structure, Geometry and Materials
- Why XML?
  - Instead of inventing our own format use a standard one (extendible)
  - Many available Parsers and Tools
  - Strategic technology
The LHCb Detector DTD

- **Divided into 3 main parts**
  - structure
  - geometry
  - material

- **External DTDs, to be referenced in every LHCb XML files**
Some Specificities

• Expressions evaluator – units & functions known

\[ 12.2\text{mm} + .17\text{m} / \tan(34\text{degree}) \]

• parameter : a kind of macro

\[
\begin{align*}
\text{<parameter name="InCell" value="40.6667*mm"/>} \\
\text{<parameter name="MidCell" value="1.5*InCell"/>}
\end{align*}
\]

• References : element + “ref”

\[
\text{<delemref href="LHCb/structure.xml#LHCb"/>}
\]

protocol://hostname/path/file.xml#ObjectID
Structure Elements

- DDDB : the root
- catalog : a list
- detelem : a detector element
- geometryInfo : connection to the geometry
- userParameter(Vector) : hook for adding parameters
- specific : hook for extending the DTD

```xml
<DDDB>
  <catalog name="...">
    <detelem name="...">
      <geometryInfo
        lfname="...
        npath="...
        support="..."/>
      <userParameter
        comment="...
        name="...
        type="string">
        ...
      </userParameter>
    </detelem>
  </catalog>
</DDDB>
```
Geometry Elements (1)

- DDDB: the root
- catalog: a list
- logvol: logical volume
- physvol: physical volume
- paramphysvol(2D)(3D): replication of physical volumes
- tabproperty: tabulated properties

```
<DDDB>
  <catalog name="...">
    <logvol material="..."
      name="...">
      <physvol logvol="..."
        name="..."/>
    </logvol>
  </catalog>
</DDDB>
```
Geometry Elements(2)

- posXYZ, posRPhiZ, posRThPhi: translations
- rotXYZ, rotAxis: rotations
- transformation: composition of transformations
- box, trd, trap, cons, tub, sphere, polycon
- union, intersection, subtraction: boolean solids
- surface

```
<subtraction name="sub2">
  <box name="box3"
    sizeX="1*m"
    sizeY="1*m"
    sizeZ="15*cm"/>
  <tubs name="tub2"
    outerRadius="15*cm"
    sizeZ="25*cm"/>
</subtraction>
<posXYZ z="-40*cm"/>
<rotXYZ rotX="90*degree"/>
```
Material Elements

- materials: the root
- catalog: a list
- tabproperty: tabulated properties
- atom
- isotope
- element: a mixture of isotopes
- material: mixtures of elements or materials

```xml
<isotope A="11*g/mole"
   name="Bor_11" />
<element name="Boron"
   symbol="B" >
   <isotoperef href="#Bor_10"
      fractionmass="0.20"/>
   <isotoperef href="#Bor_11"
      fractionmass="0.80"/>
</element>
<element name="Oxygen"
   symbol="O" >
   <atom A="16*g/mole"
      Zeff="8.0000"/>
</element>
<material name="CO2" >
   <component name="Carbon"
      natoms="1"/>
   <component name="Oxygen"
      natoms="2"/>
</material>
```
XmlEditor

- Explorer-like XML viewer
- No need to know XML syntax
- Checks the DTD when opening a file
- Allows copy, paste and drag and drop of nodes
- Allows view of several files at the same time
- Hide references across files

Easy XML edition

$LHCBSOFT/Det/XmlEditor/v*/scripts/xmlEditor(.bat)
XMLEditor
Conversion From XML to C++

- Converters used to build C++ objects from XML
- One converter per object type
  - `XmlDetectorElementCnv`
  - `XmlLVolumeCnv`
  - `XmlMixtureCnv`
  - `XmlMuonStationCnv`
  - ...
  - `XmlMySubDetCnv`
- Almost 1 to 1 mapping between XML elements and C++ objects
- Uses the xerces-C parser - Could use any DOM parser
First Summary

- We are able to reach the geometry description from the C++ transient world.
- Everything is transparent for the C++ user, there is no need to know it comes from XML.

- At this point, we have no way to extend the schema and especially to add specific parameters to a detector element.
Specializing Detector Elements

1. **adding userParameter(vector)s to default DetectorElements**
2. **extending and specializing the DetectorElement object in C++**, using userParameters in XML
3. **extending XML DTD and writing a dedicated converter**
Specializing by using UserParameter[Vector]

- Two elements:
  - `<userParameter>` and `<userParameterVector>`
- 3 string attributes: name, type and comment
- One value given as text

```
<userParameter
  comment="blablabla"
  name="description"
  type="string">
  Calibration channels
</userParameter>

<userParameterVector
  name="NbChannels"
  type="int"
  comment="blabla">
  530 230
  570 270
</userParameterVector>
```
C++ API for userParameters

- Methods on DetectorElement for userParameters:
  - string userParameterAsString (string name)
  - double userParameterAsDouble (string name)
  - int userParameterAsInt (string name)

- The equivalent exist for userParameterVectors

```cpp
std::string description = elem->userParameterAsString ("description");
std::vector<int> channelNbs = elem->userParameterVectorAsInt ("NbChannels");

log << MSG::INFO << description << "": ";
for (std::vector<int>::iterator it = channelNbs.begin();
    it != channelNb.end();
    it++)
    log << *it;
log << endreq;
```
Extending Detector Elements

- Free extension of the DetectorElement class
- Specific initialization using initialize()
  - called after conversion
  - access to userParameters
- A converter is needed but very simple (4 lines)

```c
#include "DetDesc/XmlUserDetElemCnv.h"
#include "MyDetElem.h"

static CnvFactory
  <XmlUserDetElemCnv<MyDetElem> > s_factory;
const ICnvFactory& XmlMyDetElemCnvFactory = s_factory;
```
Full Customization

- extension of the DTD to define new XML elements
- parsing of the new XML code using the xerces parser
- "real" converters to initialize C++ objects according to XML
The <Specific> Element

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DDDB SYSTEM "extendedDtd.dtd">
<DDDB>
  <detelem classID="7294" name="Head">
    <geometryinfo .../>
    <specific>
      <channelSet description="..." name="Controls">
        <channels description="Inputs" nb="20"/>
        <channels description="Outputs" nb="150"/>
      </channelSet>
      <channelSet description="..." name="Data">
        <channels description="head" nb="2000"/>
      </channelSet>
    </specific>
  </detelem>
</DDDB>
```
Writing a Converter

- One needs:
  - to get a C++ representation of the XML (DOM tree)
  - to deal with expressions and parameters
  - to reuse existing code (only convert specific XML elements !!!)
Implementing the Converter

- Real converter =
  1. extension of XmlUserDetElemCnv<DeType>
  2. implementation of method
     StatusCode i_fillSpecificObj (DOM_Element, DeType*)

  - \texttt{i\_fillSpecificObj} is called once per direct child of tag \texttt{<specific>}
  - the DOM\_Element is given, the DeType object was created and must be populated
  - all other elements (not inside \texttt{<specific>}) are automatically converted
class XmlMyDetElemCnv :
  public XmlUserDetElemCnv<MyDetElem> {

public:
  XmlMyDetElemCnv (ISvcLocator* svc);
~XmlMyDetElemCnv() {}

protected:
  virtual StatusCode i_fillSpecificObj
       (DOM_Element childElement,
        MyDetElem* dataObj);
};

static CnvFactory<XmlMyDetElemCnv> s_Factory;
const ICnvFactory& XmlMyDetElemCnvFactory = s_Factory;

XmlMyDetElemCnv::XmlMyDetElemCnv(ISvcLocator* svc) :
  XmlUserDetElemCnv<MyDetElem> (svc) {}
Converter Example (2)

```cpp
StatusCode XmlMyDetElemCnv::i_fillSpecificObj
    (DOM_Element childElement, MyDetElem* dataObj) {

    std::string elementName =
        dom2Std (childElement.getNodeName());

    if ("channelSet" == elementName) {
        const std::string name = dom2Std
            (childElement.getAttribute ("name"));
        const std::string description = dom2Std
            (childElement.getAttribute ("description"));
        dataObj->addChannelSet(name, description);
        ...
    } else {
        ...
    }
```
Panoramix
Geometry Visualization

- Visualization is essential for developing the geometry
  - Applicable at the different data representations
- Generic geometry information conversion to 3D graphics data
- Panoramix (OnX)
Panoramix

- Events and Geometry viewer
- Takes LHCb specificities into account
  - references
  - logical volumes hierarchy
  - subDetectors
- Interactive move inside the geometry

$\text{LHCBSOFT}/\text{Vis}/\text{Panoramix}/v*/\text{scripts}/\text{panoramix}(\text{.bat})$
http://www.lal.in2p3.fr/SI/Panoramix/tutorial/tutorial.html
Panoramix GUI
Event Visualization
Zoom on Ecal
The VisualizationSvc

- A Gaudi service
- Used by Panoramix/Geant4 converters
- Allows independent customisation of visualization, shared by all visualization softwares
- Takes into account:
  - colors (with alpha channel)
  - visibility
  - open status
  - display mode (wire Frame, Plain)
Geant4
Interfacing With Geant4

- We integrate Gaudi with Geant4 by providing a number of “Gaudi Services” (GiGa)
- The GiGaGeomCnvSvc is able to convert transient objects (DetElem, LVolume, Surfaces, etc.) into G4 geometry objects
  - The conversion does not require “user” code
  - Flexibility in mapping Gaudi model to Geant4 model
- Single source of Geometry information
GiGa Geometry Conversion

- Unidirectional
- Conversion of transient detector description (common) into Geant4 representation
- Gaudi Conversion Service and Converters
  - Volumes & Surfaces
  - Materials
- Instantiation of Sensitive Detector and Magnetic Field objects through Abstract Factory pattern
Condition Database

Production version:
- VDET: v3 for T ≤3, v2 for t3 < T ≤5, v3 for t5 < T ≤9, v1 for T > 9
- HCAL: v1 for T ≤2, v2 for t2 < T ≤8, v1 for T > 8
- RICH: v1 everywhere
- ECAL: v1 everywhere
Conditions DB

- Detector conditions data (calibration, slow control, alignment, etc.) are characterized by:
  - Time validity period
  - Version

- The conditions data objects will also appear in the Detector Transient Store

- The persistency of conditions data is done with the Conditions DB (IT product)
Condition Data Object

- “Block” of data belonging to some detector element
  - coded in XML
  - seen as a BLOB by the database
- Time (CondDBKey) validity range
  - [since, till]
  - CondDBKey is a 64 bit integer number. Sufficient flexibility (absolute time in ns, run number, etc.)
- Version
  - Sequence version number
- Extra information
  - Textual description, insertion time, etc.
ConditionsDB: Integration in Gaudi

- Manages store
- Synchronization updates
Conditions Conversion Service
Conditions DB Implementation

- The database used is ORACLE through the IT implementation of the interface already used for objectivity.
- XML references are used to select between plain XML and condition DB:
  - `<conditionref href="../Ecal/condition.xml#caEcal"/>` → XML
  - `<conditionref href="cond://dd/Calibration/Ecal/caEcal"/>` → DataBase