Draft1: Private communication on CMS Construction Database

Guillaume Baulieu
IPNL

May, 2006

1. Purpose

   The CMS Tracker is composed of 200000 components from about 20 different species travelling between 20 laboratories around the world. Each component has to be identified, tested, and assembled into other components. The goals of the construction database are :

   - To be able to track components in the different laboratories
   - To store and retrieve the different tests that have been performed on a given object
   - To have a global view of the production
   - To know where each object is in the Tracker
   - To keep a long term track of the tracker built in

2. Content

   (a) Objects description

   Each object is described with 3 fields :

   - The specie of the object (module, sensor, hybrid, AOH...).
   - A type. This field reflect the various shapes of an object.
   - A version. Each type can evolve in time from a prototype to a production model. The DB can follow these evolutions.

   These fields are made of acronyms and numbers, for example :

   ```
   Object : MOD
   Type : 2.11.25.22
   Version : 1
   ```

   To be more understandable for humans, each object/type/version has a corresponding description/type_description/version_description. For the example above we have :

   ```
   Description : Module
   Type_description : EC_R6P.4U
   Version_description : module with HPK sensor
   ```

   To identify a specific instance, each object has a unique object_id in the DB. This ID is also on the physical object as a bar code.

   (b) Assembly

   In order to have a description of the structure of the tracker, each object knows who is its "father" (in which object it is contained). This gives a tree representation of the tracker (see figure 1).

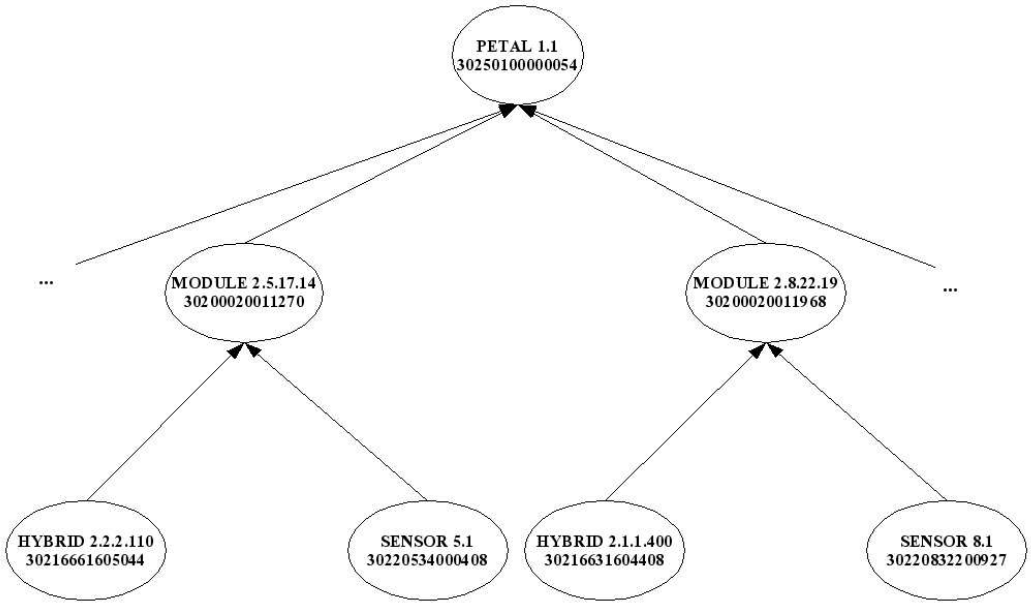

Figure 1: Sample of objects assembly

(c) Assembly rules

To avoid inconsistencies, assembly rules are stored in the DB. For each object/type, the DB has a list of acceptable object/type and the number of required objects. For example we can store that a module type 2.11.25.22 is made of :

- 1 hybrid type 2.1.1.600
- 1 sensor type 11.2
- 1 sensor type 12.2

The database will reject all other associations.

(d) Transfers

To follow the objects moving from one laboratory to an other, the notion of transfer has been created. A transfer contains :

- The sender laboratory
- The receiver laboratory
- The sending date
- The receiving date
- The carrier (which can be 'courier')
- The list of objects in the transfer

A transfer is identified by a unique transfer_id which is also a bar code on the box containing the objects. When a laboratory has to send objects, it creates a new transfer and scan all the objects to get the list of travelling objects. The objects are then known as 'shipping' and no data concerning these objects can be entered. When the destination is reached, the receiver scans the bar code on the box and acknowledges the reception. All objects are then moved to the new centre in the DB.

(e) Actions

One of the goal of the database is to store the different tests results that have been performed on a given object. These results are stored in the actions tables. One action is defined by a name, a version and the kind of object it deals with. All actions are described in the action_description table.

There are two different classes of actions.

i. Base actions

This kind of action stores the results of a single test performed on one single object. A base action always contains the following fields :

- the DB Id of the object (the barcode).
- the tool used to perform the test
- the name of the upper level composite (see below).
- An input parameter representing the input conditions of the test. For example a range of hygrometry, an analysis cut,....
- the date of the action.
- the name of the operator.
- a quality flag representing the summary of the test. This flag is a list of column separated subflags. If a subflag is positive or equals to zero the object is in conformity with the specifications, if at least one subflag is negative the object has failed the test and should be considered as faulty. All possibles values of the subflags are documented for each action in the diagnostic_description table.
- a status flag describing the status of the action. The status is not related to the quality of the object, but to the stage of the action. The possible values are described in table 1

status flag	description
reference	The result of the action is correct and should be considered as the reference for this object. Most of the time, this is the last result
valid	The result of the action is correct.
notvalid	The result of the action is not correct.
unknown	Should not be used (only in case of DB pb)

Table 1: possible status flag value for base actions

- All the fields needed to describe the test results (integer, float, vector or strings).

ii. Composite actions

This kind of actions do not really store datas but organizes actions in an action tree. Base actions are at the base and the composite actions contains pointers to other actions (base or composite). In most of the case a composite action contains all the actions needed to validate an object or an action. Among the composite actions, one distinguish

- composite of actions : the composite action and the base actions linked together are refering to the same object. A simple example is shown on figure 2.
- composite of objects : used when a an object is composite and when the action acts on all of its subcomponents. (see figure 3)

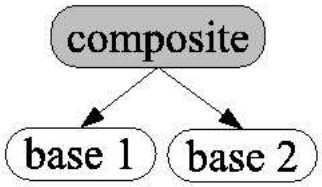

Figure 2: Example of composite of actions with 2 base actions. Both base and composite actions are related to the same type of object

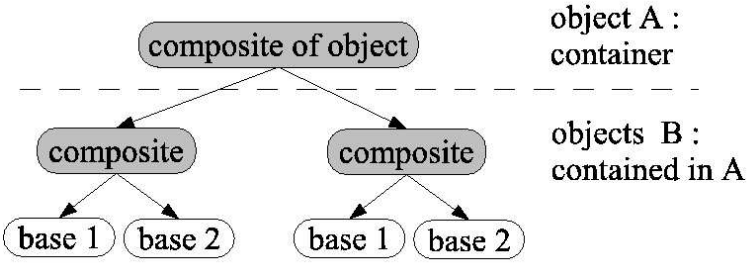

Figure 3: Example of an action composite of objects. The composite of object is related to the container object A, while the composite and the base actions are related to all subcomponents B of A.

A composite action always contains the following fields :

- the DB Id of the object.
- the name of the upper level composite (if it exists).
- a quality flag representing the summary of the test. This flag is calculated with the one provided by the base actions. Possible values are -1(faulty), 0 (correct and all subactions status equal 0) and 1 (correct but at least one subaction's status positive)
- a status flag describing the status of the action. Possible values are identical to those described in table 1. In addition, a 'running' status is available when all the base actions are not fully completed.

(f) History

The database keeps a track of all activities on each object. The following events of the life of an object are recorded :

- Registration
- Transfers
- Composite actions
- Assembly
- De-assembly
- Type/version change
- Bar code change

Each record in the history contains the date of the event, the centre where the object was, the result of this event and the state of the object.

(g) Problems

In order to be able to manage a problem concerning many objects, there is the possibility to create lists of objects concerned by a given problem. For example if a defect is discovered on a specific production system, all objects produced with this system can be flagged.

3. Technical aspects

(a) Database

The database is running on Oracle 9i and hosted by the IN2P3 computing centre in Lyon (France), which provides a 24/24, 7/7 long term database service.The figure 4 describes the database structure.

(b) Data insertion

The unique way of inserting or updating data into the database is to use XML-files to describe the actions performed. Insertion is decomposed into four main categories : registration, assembly, transfer and actions. The three first items are separated from the last because describing and storing results of each action on each object is a rather complicated thing which need a particular treatment. Registration, Assembly and Transfers must be performed in a coherent way ; for example registration of all sub-objects is obviously done before any assembly operation on them and this must be ensured by a kind of workflow. To preserve some flexibility, a real workflow hasn't been implemented but XML files format and pure database constraints has been designed to prevent from incoherences.

i. register a new object in the database:

```
<DBFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="TrackerDB.xsd">

   <registration>
      <registrationItem>
          <commonData>
             <name>ROD</name>
             <date>2002-09-25T10:00:00</date>
             <type>1.2.3.1.2</type>
             <version/>
             <center>CERN</center>
          </commonData>
          <objects>
             <object id="30240000000097" faulty="false"/>
          </objects>
       </registrationItem>
      <registrationItem>
          <commonData>
             <name>MOD</name>
             <date>2002-09-25T10:10:00</date>
             <type>3.4.12.12</type>
             <version/>
             <center>STRASBOURG</center>
          </commonData>
          <objects>
             <object id="30200020007810" faulty="false"/>
             <object id="30200020007811" faulty="false"/>
             <object id="30200020007812" faulty="false"/>
```

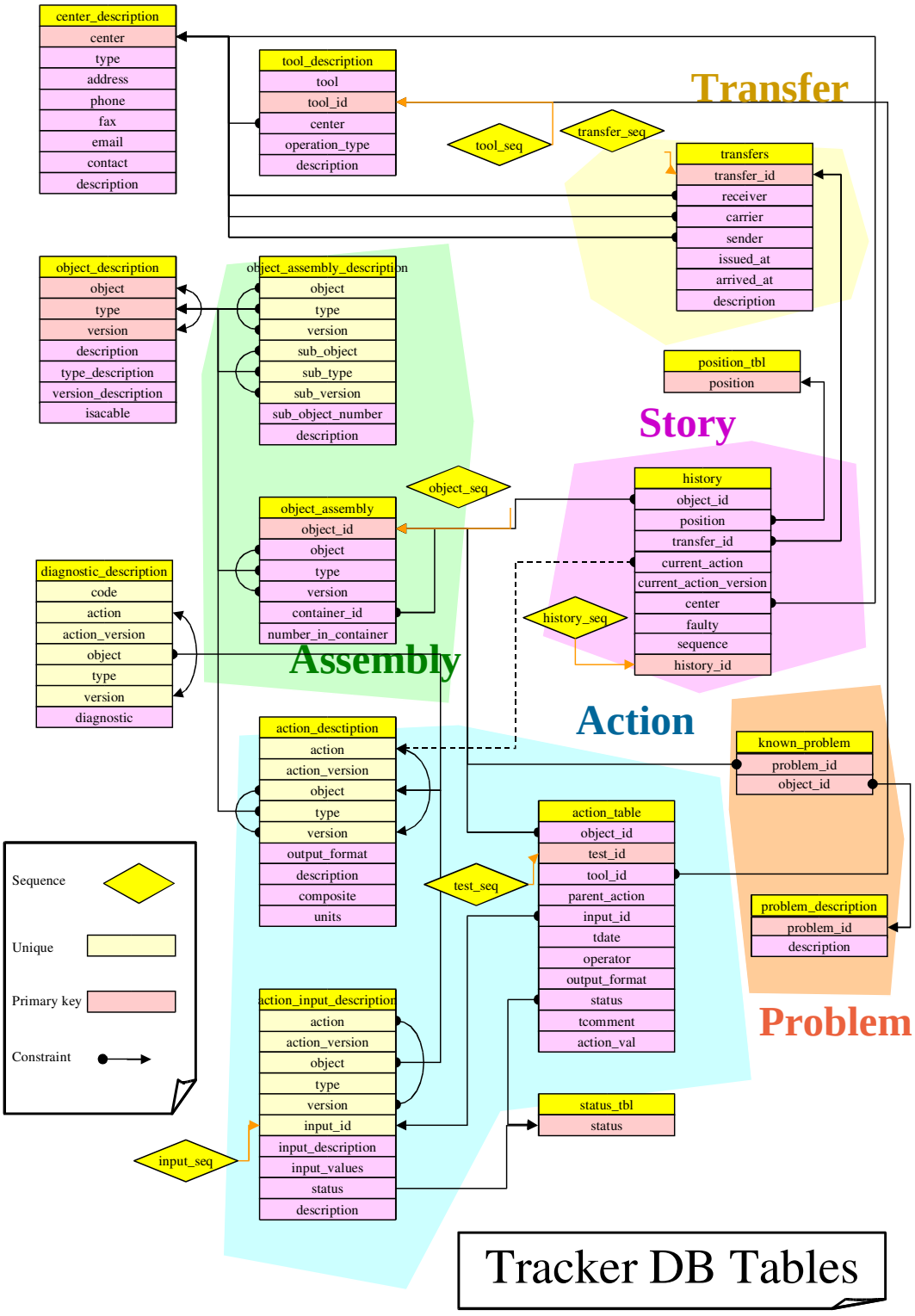

Figure 4: Database structure

```
          </objects>
        </registrationItem>
    </registration>
</DBFile>
```

ii. assembly
Once objects have been registered, they may be assembled with a file like the following :

```
<DBFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="TrackerDB.xsd">

   <assembly>
      <assemblyItem>
         <parent id="30240000000097"/>
         <subobject id="30200020007810" position="1" action="add"
                    date="2002-03-20T09:00:00" faulty="false"/>
         <subobject id="30200020007811" position="2" action="add"
                    date="2002-03-20T09:01:00" faulty="false"/>
         <subobject id="30200020007812" position="3" action="add"
                    date="2002-03-20T09:02:00" faulty="false"/>
      </assemblyItem>
   </assembly>

</DBFile>
```

Here, a list of `<subobject>` is indicated to be placed in the `<parent>` object and a given `date` at a given `position`. The position is just a number and people doing such a job need to choose (and remember !) which real position correspond to this number. if `action="remove"` flag is here, this subobject will be remove from this parent object. If something is going wrong during the assembly, the subobject become faulty by setting the `faulty` flag to true.

iii. Trace transfer of objects between production centers
One need to know where are each objects stored in the database. Each time someone is sending some objects to another production center, it is necessary to start a `<transfer>` indicating which objects are concerned, where they are sended at which date. The file for that is this kind :

```
<?xml version="1.0" encoding="UTF-8"?>
<DBFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="TrackerDB.xsd">
    <transfer>
        <transferItem>
            <info>
                <id>30299991000001</id>
                <sender>LYON</sender>
                <receiver>CERN</receiver>
                <carrier>COURIER</carrier>
                <issued>2002-04-22T17:33:18</issued>
                <arrived> </arrived>
                <description> object is sent not received. </description>
            </info>
            <batch>
                <object faulty="false" id="30221133500239"/>
```

```
            </batch>
          </transferItem>
        </transfer>
      </DBFile>
```

when the object is arrived at the `receiver` center, the tag `<arrived>` is filled with the current date. This action closes the transfer.

iv. Actions
The description of the test results in the database is done via xml files. The content of these files is sent to the database either via BigBrowser or either via a command line parser. During the update of the DB, the flags (status flag and quality flag) of the composite action are automatically calculted starting from the individual flags of the base actions. The history table is also updated. Technical details on the xml file format and on the rules of flags calculations can be found on the DB website : http://cmsdoc.cern.ch/∼cmstrkdb.

v. Special cases
To be able to deal with events that are not foreseen (for example an object fell down and was broken), a SPECIAL action has been created (with its composite FREEACTION) to allow an operator to enter data outside the normal workflow.
It is also possible to have a different behaviour for some specific cases. For example the validation of the sensors can be done through a sampling procedure : only some sensors are really tested to validate a whole batch.

(c) Cabling
The construction database also store cabling informations for the Tracker End Cap (TEC) and the Tracker Outer Barrel (TOB) up to the patch panel level 1.

(d) Data access

i. The Graphical User Interface : BigBrowser
The standard way to communicate with the database is to use the JAVA graphical user interface called `BigBrowser`. It has been designed to perform all of the standard tasks one may want to do with the database : inserting objects, making some assemblies, transfering objects between two production centers and inserting some results of an action realized on one particular object. `BigBrowser` offers the possibility of inserting data "in live" or saving the session in one XML file which can be inserted in the database later.
Apart from that, `BigBrowser` can be used for retrieving different informations :

A. Tracker Description
This menu shows the trees of all the components in the database.

B. Production Status
This menu is very usefull. It shows a kind of snapshot of the production at a given time. An inventory of all the objects and a view of all current transfers are avaible.

C. Quality control
This menu allows people to make some statistics on the production. For instance, one can knows the number of petals that have passed the thermal test, etc ... There is also a

special menuitem allowing someone at ease with SQL syntax to query directly the database with SQL language.

D. Tables
This menu is rather technical ; it shows the database tables and their description. This information can be usefull for people that want to know a little more on this aspect to develop their own database interface.

`BigBrowser` ensures that data are correctly inserted in the database. The technique used by `BigBrowser` is to produce XML files that are treated to insert data. This approach allows people that need a special control to produce their own XML files with their own application and then inserting data using either `BigBrowser` or the command line version provided with the installation package.

ii. Java Interface Plug-ins
To extend the possibilities of `BigBrowser`, the concept of plug-in has been implemented. It is therefore possible to write small programs in Java which will be able to query/insert data from/to the database. Those programs will have a graphical interface that will be available from BigBrowser. A guide to develop plug-ins is available on the website.

iii. Relay application
The relay application is used to query the database from anywhere, using any language. The application is waiting on a socket, so you just need to open a TCP connection and send you query using a very simple XML format. The answer will be send also in XML format. For example, if you want the list of bad strips of a sensor you send :

```
<?xml version="1.0"?>
<select db="prod">
  POSITION_OF_BAD_STRIPS
  from stripscansummary_1_sen_
  where object_id=3022111605326
        and status='reference'
</select>
```

and you will receive :

```
<?xml version="1.0" encoding="UTF-8"?>
<answer>
<status>200 DBQuery: OK</status>
<row>
<column>POSITION_OF_BAD_STRIPS</column>
<value>256 494</value>
</row>
</answer>
```

The relay application is waiting on port 3615 of cmstrkdb.in2p3.fr. There is also a backup server on port 3615 of lyopc72.in2p3.fr.

4. Status and evolutions

(a) Statistics

The database is in production since the summer 2001. The current number of registered objects (May 2006) is around 190 000. During the last year (from may 2005 to may 2006) the average data insertion rate was about 2000 records/day in the history table. During the

last 6 months, the 2 relay applications answered to an average of 10 queries per minute.

For more informations on the Construction Database, you can go to the website : http://cmsdoc.cern.ch/∼cmstrkdb. A mailing list for users has also been created with CERN's mailing list system (SIMBA) : cms-trackerdb-users@cern.ch.