A DATABASE PERSPECTIVE ON CMS DETECTOR DATA

A.T.M. Aerts, Eindhoven University of Technology, Eindhoven, The Netherlands
F. Glege, M. Liendl, CERN, Geneva, Switzerland

Abstract

Building a state of the art high energy physics detector like CMS (Compact Muon Solenoid) requires strict interoperability and coherency in the design and construction of all sub-systems comprising the detector. This issue is especially critical for the many database components that are planned for storage of the various categories of data related to the construction, operation, and maintenance of the detector like slow control data, conditions data, calibration data, and event meta data. The data structures needed to operate the detector as a whole need to be present in the database before the data is entered. Changing these structures for a database system that already contains a substantial amount of data is a very time and labour consuming exercise that needs to be avoided. Cases where the detector needs to be treated as a whole are detector operation (control, error tracking, conditions monitoring) and the interfacing of the reconstruction and simulation software.

In this paper we propose to use the detector geometry as the structure connecting the various elements. The design and implementation of a relational database that captures the CMS detector geometry and the detector components is discussed. The detector geometry can serve as a core component in several other databases in order to make them interoperable. It also provides a common viewpoint between the physical detector and its image in the reconstruction software. Some of the necessary extensions to the detector description are discussed.

CMS DATABASES

In a recent overview of the CMS databases [1] four types of databases were distinguished that describe aspects of the detector:

- Construction databases
- Equipment Management database
- Configuration database
- Conditions database

This characterization follows the phases in which the knowledge about the detector is built up. It does not imply that the databases will also be implemented in this way.

The construction databases hold the information about the sub-detector construction, which covers the design parameters, the construction process and the concluding tests for each detector component.

The equipment management database (EMDB) [2] details the location of each detector component in the detector assembly, as well as that of the peripheral components in and near the experiment hall, as well as their connections. One of its uses is the tracking of all detector components from the moment of creation to the moment of disposal for the purpose of complying with the INB (Installation Nucleaire de Base) regulations.

The configuration database holds all information required to bring the detector in any running mode. This includes the parameters and other settings that are downloaded at the time the (sub-) detector is put into an operational state.

The conditions database holds all information needed for event reconstruction, except the event data itself. This involves data on slowly varying observables, such as the temperature and gas pressure in the experiment hall, but also on the much more quickly varying statuses of the sensitive detector elements. Note that much of the configuration data that has been downloaded into the detector becomes conditions data, when it is read back to provide the initial or start-up state of the detector.

At present, the construction databases for the sub-detectors are operational, the EMDB is operational but still being extended, and the other databases are mostly still in the design phase. A detailed overview of the current status of these four types of information and their mutual dependencies can be found in [3].

The event data is filtered by the on-line reconstruction software, tagged and then stored for further off-line analysis.

Development Risks

In the CMS collaboration (and also in the other collaborations), the development of the information models and the usage of the information often go hand in hand. Several implementations are being developed on a trial and error basis, guided by local needs. This observation can be illustrated by the case of the Construction databases. Even though a common solution has been proposed that was expressive enough to cover all situations [4], local, sub-detector, and even production site specific solutions were developed that were made optimal for the local situation. The result is a collection of heterogeneous databases, implemented in diverse technologies, often accessible by proprietary interfaces only (see [3]). Although some of the data in these databases are only of local interest, another part of them describes properties of the detector components that is of interest to the EMDB and should be integrated into it. To make these data interoperable with those of the EMDB will require as many conversion programs as there are Construction databases. This statement assumes that all conversion can be done automatically and does not require manual intervention. An alternative is to transfer the relevant data into the EMDB and make them available from there. This can be done only when the construction data has been consolidated and has become read-only, to avoid the risk of divergence of the various copies of the data. Also this option will require the inevitable conversion programs.

When we translate this situation to the context of on-line reconstruction or on-line error tracking, it is clear that one wants to avoid the situation that there will be as many variants of the conditions data as there are sub-detectors, or worse. The timing constraints for processing these data are much more severe and one will want to avoid the performance losses due to unnecessary conversions. Neither does one want copies of the same data in various locations to avoid inevitable inconsistencies and the load of managing (huge quantities of) redundant data.

The problem of technological heterogeneity has been tackled by standardizing on one database technology for the on-line databases, Oracle. This leaves the heterogeneity at the modeling and the syntactic level to be dealt with.

DETECTOR GEOMETRY DATABASE

In the previous section it was observed that so far many sub-detector databases have been developed on a trial and error basis, guided by local needs. On the basis of a working version of a model, one can establish on what points the model satisfies the often still tacit requirements and on what points it does not. This can be a good way to elicit hidden requirements and conduct performance studies, and it is a common practice in software engineering. This has also been the way of working for the detector description used in the simulation and reconstruction software (see [5] and [6]). However, it is not the way to proceed for databases containing information that needs to be globally accessible. The data structures needed to operate the detector as a whole need to be clear before the data is entered into the database. Changing these structures for a database system that already contains a substantial amount of data is a very time and labour consuming exercise that needs to be avoided. A better approach is to try to identify the concepts that are common and shared among the various users of the information and use these as a global access point. As has also been noticed by others ([4], [5]), the detector geometry appears to be a good option for this purpose.

The detector geometry is implicitly present in all the detector related databases (all physical components have spatial dimensions), and explicitly in the analysis and reconstruction software, where the detector is regarded as an assembly of shapes with particular material properties. As such, it can serve as an intuitive shared interface between the data and the software world, once it has been made explicit in the databases.

The nominal detector geometry in principle is also a rather stable part of the description. It will only change, when one of the sub-detectors is moved with respect to the others or is replaced by a differently shaped one. This is a rather infrequent occurrence: perhaps once every five years.

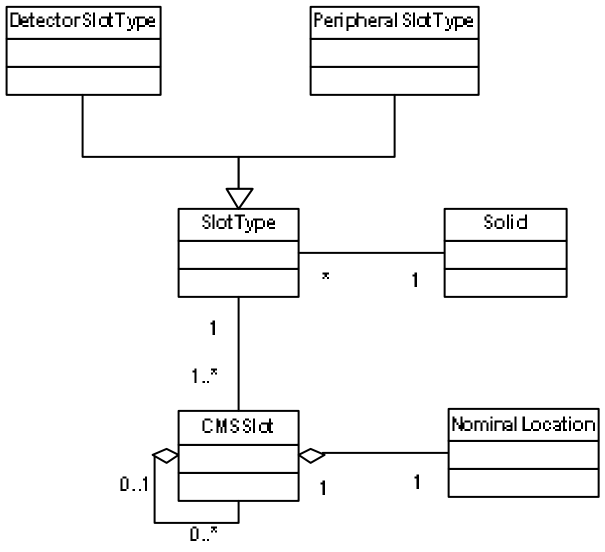

Figure 1 Detector Geometry

The detector and its supporting equipment can be modeled as a hierarchy based on the container-contained relationship between detector parts (see [5]). Many detector parts can be viewed as composed of parts or components that may in turn be composite themselves. In the geometry description the volume or space that each physical detector or peripheral part will occupy is modeled, not these parts themselves. Such a volume is called a CMS Slot. The containment hierarchy is represented in Figure 1 by a Simple Tree pattern ([7], consisting of the CMS Slot and its self referencing aggregation relationship. (We represent the models by means of UML (Unified Modeling Language) class diagrams [8], using database stereotypes.).

The CMS Slot represents a Bill of Materials structure for the detector to which spatial information (Nominal Location) has been attached for the location of the slots in the detector as designed. The latter is given in terms of the absolute positions and orientations of all volumes with respect to the detector frame of reference.

For a complete geometrical description, also descriptive information such as the shape of the slots is needed. Since the detector itself is a fairly symmetrical construct, a Slot Type has been introduced to capture descriptive information which is common to a number of slots, such as a shape, modeled as the Solid, and possibly some other properties. This information, together with the Nominal Location, can be used to construct a 3D-representation of the detector (see [8]). Note that there are some constraints to be satisfied by the descriptions. For instance, the solid of a container should encompass all solids of its contained components.

The data model can handle all information about the detector model that is used in the simulation and reconstruction software, but it is not limited to that. Its structure is suited for capturing both finer details (further decomposition), and information about peripheral structures such as the racks with the measuring equipment and power supply units that are partly co-located with the detector, partly located elsewhere. This is made explicit in Figure 1 by the introduction of special Slot Types, such as the Detector Slot Type and the Peripheral Slot Type that represent volumes that will be occupied by a detector or a peripheral part, respectively. These sub-types will have distinguishing characteristics of their own.

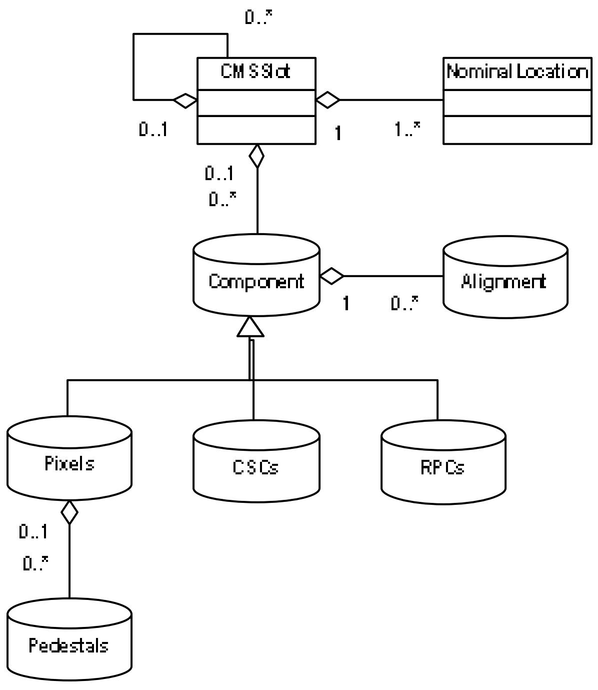

Figure 2 Embedding of the Detector Geometry in the Conditions data

The information in the database can be used for a number of purposes, such as visualization of the detector during construction, integration and operation. The CMS Slot is a core construct for a number of applications. First of all, it serves to integrate the sub-detectors. In Figure 1 it also serves as a point of reference to attach location information. In the same way, alignment (deviations from the nominal location), calibration, and configuration information can be attached to it. This is further illustrated in Figure 2, where the occupancy of a slot by a detector component is modeled. The component may have alignment data associated with it and is sub-typed to represent the various sub-detectors with their own characteristic properties, and associated configuration and conditions data (exemplified by the Pedestals). In Figure 2 some UML class symbols have been replaced by an (non-UML) database symbol to represent the fact that data about components and alignment, and so on, has a much more complex structure than a simple class.

IMPLEMENTATION ISSUES

A prototype implementation of the detector geometry database in Oracle 9i has been made, which is discussed in [9]. The prototype has, including indices, a size of about 600MB. The data for the database was taken from the DDD (Detector Description Database) [5] used in the simulation software. In addition to the slots occupied by the physical detector parts, the DDD also adds a number of intermediary levels to the detector hierarchy that serve to group a number of components. This is both for easy of data entry, and for ease of representing the geometry in the simulation software. Not all of these intermediate levels are needed for the geometry database, which will reduce its size.

The data model prototype was designed from first principles, and not generated as was done in [10]. The reason is that the structure of the DDD is captured in XML Schema, which is quite suited to specify document structures but lacks expressiveness when it comes to specifying useful database structures.

Depending on the implementation of the various detector databases, the geometry database can be a central part of one large database which encompasses the four database types mentioned above, it can be a (read-only) component in each of a number of separate implementations, and it can be implemented as an independent database from which the other databases are accessible.

An important implementation point is that a common geometry also implies a common identification scheme. Since the identification of slots (slot-id) is kept independent of that of the components, the slots can be taken as an independent point of reference to provide geometry based global access. From the slot-id the nominal position in the detector should be deducible and vice versa. The slot ids should have a two level structure. The higher level would point to volumes that are interesting to the off-line software. The second level would point to positions taken by components (chips, or boards) that are relevant for configuration settings but too detailed for the simulation and reconstruction software.

Note that once the ids have been assigned and the database has become operational, the ids can only be changed at a very high cost, since they will propagate across all data.

Note that in the DDD no explicit id-scheme is proposed: volumes are distinguished by their position in the detector hierarchy, obtained by following the path from the full detector volume (the root of the hierarchy). This comes from the fact that the DDD is stored in a compact form where information that is common to a number of components is stored only once [5]. This compact description can be read in quickly and is expanded in memory to generate all individual shapes. The matching of detector components in the on-line databases to the volumes in the software detector model is therefore a major concern and has to be solved to support the usage of, e.g., conditions data in the simulation and reconstruction processes. The match between the slots in the database and those in the software based model will have to be done at the level of sensitive and support (e.g., yokes) parts which correspond to the stable part of the software model. However, one cannot use the software model to generate ids, because the model as a whole is still subject to change, as intermediate grouping levels are added or omitted. These changes will affect the ids that are generated. One possible strategy to establish correspondence is by embedding the database slot ids into the software detector description. Since the detector model for the simulation and reconstruction software is composed by hand, this will be a manual task. Great care will have to be taken to leave the thus augmented part of the software model invariant under subsequent optimizations of the software detector model. On the other hand, then also the database geometry will be fixed, because of this dependency. It is clear that a solution should satisfy the needs of both sides as well as possible.

A similar remark holds for the conditions data. The globally supported matching [11] between event data and conditions data on the basis of Interval-of-Validity, TAG and Version is a high level matching that shield off all (sub-)detector dependent conditions data structures. These structures have to be known and agreed upon by both the database and the reconstruction software. In the latter case, this will mean that the conditions data structures will be hard coded into the software (and thus will be highly resistant to change). This imposes a big dependency on the databases containing the source for these data. Fortunately, in the case of relational database implementations, one can use the view mechanism to shield off some of these dependencies.

What is lacking at the moment is the implementation of a uniform naming scheme for the physical detector parts. It is not possible at this point to track a detector part from inception to decommissioning. A uniform naming scheme for part-IDs would include one data type for all databases concerned. At present only a prescription exists for a global format for the identification string (19 characters format) [12], but this is not adhered to by all production groups. The ID-string would allow the encoding of the major sub-component that the part belongs to, and its unique ID inside this component. Also versioning information of the part should be included. A good place to introduce this naming scheme would be at the point where the construction database information is copied into the other databases. A reference copy of this can be kept in the geometry database. A complicating factor here is that for some sensitive components an ID has already been hard-coded into the hardware. The incorporation of these kinds of IDs has to be studied.

CONCLUSION

The detector metaphor can support detector monitoring and error tracking. It has served as the leading concept for a database that makes it possible to access regions or components in the detector on the basis of positional information. This is useful for adjacency queries (such as: give me all temperatures for a given period in the neighborhood of this specific component) needed in the process of tracking down sources of errors.

It can also serve as a skeleton, as a structure to connect the various data elements such as slow control, conditions and configuration data and make them efficiently and uniformly accessible. This characteristic will make it a suitable tool for the integration of the various databases.

A lot of work still has to be done to come to an acceptable integration of the database and the software worlds.

ACKNOWLEDGEMENTS

It is a pleasure to thank Ian Willers, Igor Vorobiev, Martti Pimia, Frank van Lingen, Michael Case, Boby Gomez-Reino, Pedro Arce, Asif Jahn Muhammad, and Stefan Wynhoff for many interesting discussions.

REFERENCES

[1] F. Glege, “Databases in CMS”, Presentation Database Meeting, June 25, 2003.

[2] F.Glege, “Integration database requirements evaluation”, Presentation TCM 63, April 28, 2003.

[3] A.T.M. Aerts, F. Glege, M. Liendl, I. Vorobiev, I.M. Willers, S. Wynhoff “A database perspective on CMS data requirements”, CMS Note 2004/xxx

[4] J.-M. Le Goff et al., “C.R.I.S.T.A.L. Concurrent Repository and Information System for Tracking Assembly and production Lifecycles”, CMS Note 1996/003

[5] M. Case, M. Liendl, F. van Lingen, “Detector Description Domain Architecture and Data Model”, CMS Note 2001/057

[6] A.T.M. Aerts et al., “CMS Detector Description: New Developments”, CHEP04, these proceedings.

[7] Gamma Erich, Richard Helm, Ralph Johnson, and John Vlissides. "Design Patterns:Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, MA, 1995

[8] Unified Modeling Language, http://www.uml.org

[9] Available from the authors on request.

[10] Asif Jan Muhammad et al., "Migration of the XML Detector Description Data and Schema to a Relational Database", CMS Note 2003/031.

[11] The ConditionsDB project home page: http://lcgapp.cern.ch/project/CondDB

[12] A.H. Ball, “CMS numbering and naming scheme”, http://cmsdoc.cern.ch/~cmstc/naming_and_labelling/