CMS DETECTOR DESCRIPTION: NEW DEVELOPMENTS

A. Aerts, *Eindhoven University of Technology, Eindhoven, The Netherlands*
M. Case, *Univ. of California Davis & European Laboratory for Particle Physics (CERN)*
M. Liendl, *European Laboratory for Particle Physics (CERN)*
Asif Jan Muhamad, *CERN*

Abstract

The Compact Muon Solenoid (CMS) Detector Description Database (DDD) consists of a C++ Application Program Interface (API) and an Extensible Markup Language (XML) based detector description language. DDD is used by the CMS simulation (OSCAR), reconstruction (ORCA), and visualization (IGUANA) software projects as well by test beam software that relies on those systems. The DDD is a sub-system within the COBRA framework of the CMS Core Software. Management of the XML, the source for the DDD, is currently done using a separate Geometry project in CVS.

We give an overview of the DDD integration and report on recent developments concerning detector description in CMS software. These are the algorithmic construction of detector sub-components, the streamer which allows users to write to local binary files, the recent relational database extension effort and the integration of the DDD into the COBRA framework.

INTRODUCTION

The CMS software project requires that offline software for applications such as simulation, reconstruction, and visualization have a coherent common source for the geometrical properties of the detector. These applications use the DDD sources and API as the basis for this consistent view of the detector description [8]. The offline applications have several common requirements which the DDD addresses in the DDD Domain Model. Additionally the DDD provides the ability to include application-specific information to be associated with detector parts.

Requirements

The requirements that continue to guide the development of the DDD are as follows:

1. One Definition Rule
    a. Each identical part in the detector should only be defined once, but may be positioned multiple times.
    b. Each material, solid, rotation and any other DDD object should be constrained by this rule.
2. The DDD must provide a common API for CMS reconstruction (ORCA), simulation (OSCAR) and other CMS software.
3. The DDD should only define the ideal detector and not re-alignment or other conditions.
4. The DDD must provide a generic way for client applications to extend the DDD.
5. The DDD must provide a hierarchical querying and filtering mechanism.
6. The DDD must provide similar ease of access and portability of sources as was provided by CMSIM (the Geant3 based simulation of CMS).
7. The DDD must contain all information necessary to build up application specific internal representations of the detector.
    a. The DDD should not be used as 'internal' representation of the detector within the applications.

A more detailed description of the requirements can be found in [2] [3] [4] [8].

Outline and conventions

We begin this note with an overview of the DDD Domain Model followed by a discussion of the C++ and XML [12] model designs. We then describe new features starting with the extension for algorithmic detector description. We review relational database work and describe the new effort in this area. Lastly we briefly discuss the streamer and the consequences of integration with the CMS COBRA [7] framework.

DDD DOMAIN MODEL

Model overview

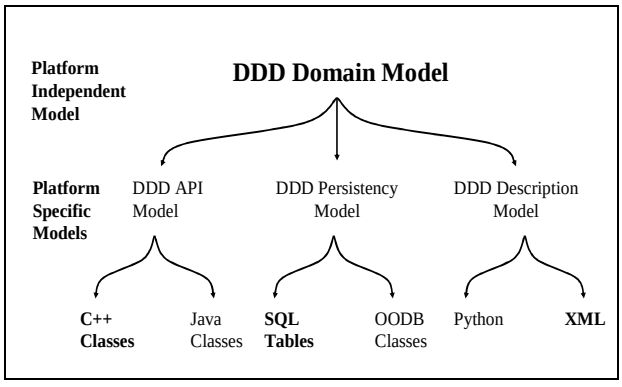

Figure 1: DDD Domain Model as a source from which to derive platform specific models and implementations. [3]

The DDD Domain Model provides a platform independent model which can be used in the Model Driven Architecture [13] approach to develop platform-

and implementation-specific models. The layout of this process and derivations of specific models can be seen in Figure 1.

Model Detail

The DDD contains the geometric properties of the detector. This includes shapes, materials, positions and application-specific parameters [Figure 2]. In the model a *LogicalPart* defines a part of the detector and comprises a *Solid* and a *Material*. The Solid describes the shape and size of a part and the Material describes the material properties of the part. To build the detector hierarchy, each part is placed inside its parent part. There is a root LogicalPart for the whole detector with appropriate dimensions.

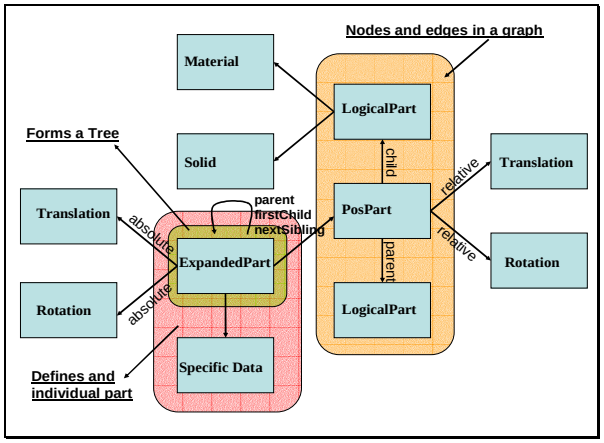

Figure 2: Overview of the DDD Domain Model (for details see [1])

A *PosPart* object positions a child LogicalPart inside a parent LogicalPart by specifying a *Rotation* and *Translation* with respect to the parent. Together the LogicalParts and PosParts form a graph structure with LogicalParts being the nodes and PosParts being the edges of the graph. We call this graph the *CompactView*.

Multiple LogicalParts (children) can of course be placed inside the same LogicalPart (parent). Such a graph is directional (parent to child) and allows multiple edges between two nodes. Of course there are no cycles in the graph because a part can neither contain itself nor any of its predecessors in the real world.

The *ExpandedPart* represents the actual part once it has been positioned in the parent. It knows its absolute position and parent. By traversing all paths through the graph, a tree is generated such that each copy of the part is a node on the tree. The result is what we call the *ExpandedView* of the detector.

A client application can attach to ExpandedParts application-specific data in order to build a customized internal representation of the detector.

This model fulfils the necessary requirements and forms the basis for the DDD.

C++ AND XML DATA MODEL

The designs of the C++ and XML Models were derived and developed together from the DDD Domain. These models are thus very similar and are described here [Figure 3].

Client software accesses the DDD via an API. The API allows access to the CompactView and ExpandedView representations of the detector. Nodes of the ExpandedView are built on the fly, transparently to clients, by traversing paths in the CompactView for reasons of memory optimization. The ExpandedView can be further filtered to allow the user to traverse a selected set of nodes from the tree.

Processing

The processing works as follows: the Document Management portion reads a configuration file or otherwise acquires a list of XML files which constitute the current geometry configuration of the detector. All files are then processed in the Document Processing section which makes DDD C++ objects using the DDD services. This all happens via the DD Language interface which is defined using an XML Schema. Once in the runtime system, the information is available to the client software and can be stored in other repositories as well.

Further details on the XML can be found in [4][2]. There is a logical if not exactly one-to-one mapping

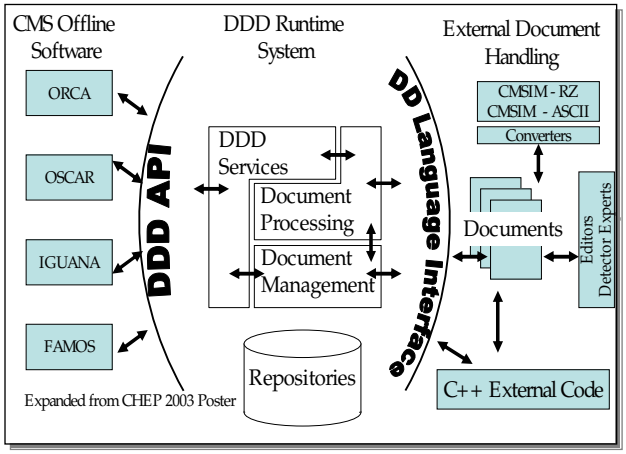

Figure 3: DDD API Model [3]

between the XML and the C++ models.

Table 1 shows some examples of this mapping. For example a DDMaterial is created from <Material… > element descriptions in the XML. Similarly, a DDSolid is made from such entries as a <Box …> or <Trapezoid…> in the XML. A DDLogicalPart is created from the XML element <LogicalPart … > which includes a reference to a material <rMaterial…> and to a solid <rSolid…> or a solid element inside the LogicalPart element and so forth.

Table 1: Examples of correspondence between XML Elements and C++ objects

XML	C++
<Material name="copper"…/>	DDMaterial
<Trapezoid name="t1" dx="30*cm" …/>	DDSolid sub-class DDTrapezoid, DDBox, DDTube, etc.
<LogicalPart name="lp1"> <rSolid name="t1"/> <rMaterial name= "copper"/> </LogicalPart>	DDLogicalPart (with reference to DDSolid and DDMaterial)
<PosPart name="pp1" copyNo="3"> <rParent name="lp1"/> <rChild name="lp2"/> <Translation x="3*cm" y="10*cm" z="10*cm"/> <rRotation name="30DegreesX"/> </PosPart>	Ddpos(DDLogicalPart parent, DDLogicalPart child, DDTranslation t, DDRotation r)

ALGORITHMIC DETECTOR DESCRIPTION

Requirements

During the past year the sub-detector groups began rewriting their generated XML files taking advantage of features of the DDD that were not utilized in the software that generated the current XML files in the Geometry project. This has lead to some new requirements for the DDD.

1. Sub-detector software experts want to algorithmically create and position as needed, DDD objects such as DDLogicalParts and DDSolids.
2. We do not want to store such sub-detector specific code with the main DDD.
3. We need to allow DDD access to these algorithms with a clearly defined C++ interface and its corresponding XML.

Implementation

As a result, the XML Schema has been extended to include an Algorithm element and the C++ code has been extended with a SEAL Plugin [10] in the Runtime System (implemented as the DDAlgorithm class). The XML Algorithm element can contain other elements such as Numeric, String, Vector, Map and StringVector. These elements are passed as arguments to the initialization of the SEAL Plugin.

The SEAL-Core Libraries and Services Project is an LHC Computing Grid Application project [9]. External software is interfaced to the DDD Services via the SEAL Plugin Manager. This mechanism provides algorithmic creation and positioning of LogicalParts and other DDD objects within one LogicalPart parent.

An XML snippet is given in Table 2 to illustrate the structure of the Algorithm element. All DDD expressions, such as those in the Vector element in the example, are evaluated by the usual DDD expression processing and all such expressions and constants are available to the user in the XML as well as to the C++ external code.

Table 2: XML Fragment

```
<Algorithm name="DDHCalBarrelAlgo">
   <rParent name="TBHcal:HCal"/>
   <String name="MaterialName"
           value="materials:Air"/>
   <Numeric name="NSector" value="[numSectors]"/>
   <Vector name="DetWidth1"
           type="numeric" nEntries="[numDets]">
    154.0*mm, 161.8*mm, …
  </Vector>
  <Vector name="AbsorbMat"
           type="string" nEntries="2">
   materials:Aluminium,  materials:Iron
  </Vector>
  <Map name="D1TypeToWidth" type="numeric"
    nEntries="[numTypes]">
    normal=20.5*mm,
    missing=0.0*mm,
    narrow=20.0*mm,
    midsize=23.5*mm
    largest=75.0*mm
  </Map>
</Algorithm>
```

RELATIONAL DATABASE

Existing work

Our initial approach to using a relational database as an alternative repository for the DDD was implemented by syntactically mapping the XML Schema to a MySQL Schema and migrating data from the XML documents to the MySQL database [6]. This was successful in that it was relatively automatic and the functionality was enough for the DDD runtime to be instantiated from the relational database. However, the mapping was not very rigorous and has been superseded by a new scheme better suited to the latest requirements.

Reason for new effort

The online community will be writing conditions, calibration and slow control data to relational databases. These databases need to have associations to objects in the reconstruction software (ORCA) and potentially other DDD client software. Therefore a mapping from the DDD to the relational database world needs to be made.

Description of new effort

As a first step, a redesign of the detector geometry database model [5] was undertaken using a more rigorous

database design approach instead of an automatically generated schema and database. The resulting relational detector geometry can provide a persistent link between selected individual components in the expanded view detector model of the offline software world and the physical detector components as monitored in the online database environment. This will allow us to maintain the mapping between the two representations.

Additionally we need to consider new procedures and tools for executing schema migration.

The relational database model we now have is implemented in Oracle 9i and is considered a prototype. It is a skeleton on which conditions, calibration, slow controls and other relational databases can be attached or associated. We are working on refining the model for performance, on its integration with the software and with other CMS detector related databases, and on the creation of a portable, read-only version.

OBJECT STREAMER

The runtime system provides a streaming mechanism to write and read binary files. These files can be used as a local cache for the applications, so that the first run (or a pre-run application) can load the files and subsequent jobs can read from this cache. The streamer is faster than the XML parsing. It can be used instead of or simultaneously with the standard mechanism, which allows users to alter the XML and run jobs using the sources.

COBRA INTEGRATION

The DDD is now a component of the CMS software framework. This integration allows greater code re-use and optimization. The repackaging also facilitates configuration management, release, testing, and maintenance, without compromising the architectural integrity and modularity of the DDD.

SUMMARY

The DDD has been integrated into the CMS software framework and continues to evolve as required by the collaboration. Algorithmic positioning provides a new way to produce DDD objects in the core graph/tree model. We are starting to provide a few different means of storage with an eye to helping the conditions and calibration project to attach or associate their data with the geometry model as well as for caching and speed improvements for batch processing. The DDD provides the necessary geometry services to the CMS software project and evolves with user needs while remaining true to the requirements in the original project.

REFERENCES

[1] M. Liendl, *"Design and Implementation of an XML based object-oriented Detector Description Database for CMS."* Thesis, 2003, Vienna University of Technology

[2] M. Case, M. Liendl, F. van Lingen, *"Detector Description Domain Architecture & Data Model"*, CMS Note 2001/57

[3] M. Case, A. Muhammad, M. Liendl, F. van Lingen, *"CMS XML-based Detector Description Database System"*, CHEP 2003 Poster

[4] M. Liendl, F. van Lingen, M. Case, T. Todorov, P. Arce, A. Furtjes, V. Innocente, A. de Roeck, *"The Role of XML in the CMS Detector Description"*, CHEP 2001

[5] A.T.M. Aerts, F. Glege, M. Liendl, *"A database perspective on CMS Detector data"*, CHEP 2004 Poster P225

[6] A. Muhammad, M. Liendl, F. van Lingen, A. Ali, I. Willers, *"Migration of XML Detector Description Data and Schema to a Relational Database"*, CMS Note 2003/31

[7] COBRA: http://cobra.web.cern.ch/cobra/

[8] DDD: http://cmsdoc.cern.ch/cms/software/ddd/www/index.html

[9] LCG Applications: http://lcgapp.cern.ch/project/

[10] SEAL Plugin documentation:

[11] http://seal.web.cern.ch/seal/

[12] XML: http://www.w3c.org/

[13] Object Management Group, Inc. (OMG) Model Driven Architecture(MDA): http://www.omg.org/mda/