Journal of Physics: Conference Series **513** (2014) 022026 doi:10.1088/1742-6596/513/2/022026

CMS Geometry Through 2020

I Osborne[1,2], E Brownson[3], G Eulisse[2], C D Jones[2], D J Lange[4] and E Sexton-Kennedy[2]

[2] Fermilab, Batavia, IL 60510-5011, USA
[3] UPR, Mayaguez, 00680, Puerto Rico, USA
[4] LLNL, Livermore, California, USA

E-mail: ianna.osborne@cern.ch

Abstract. CMS faces real challenges with upgrade of the CMS detector through 2020 and beyond. One of the challenges, from the software point of view, is managing upgrade simulations with the same software release as the 2013 scenario. We present the CMS geometry description software model, its integration with the CMS event setup and core software. The CMS geometry configuration and selection is implemented in Python. The tools collect the Python configuration fragments into a script used in CMS workflow. This flexible and automated geometry configuration allows choosing either transient or persistent version of the same scenario and specific version of the same scenario. We describe how the geometries are integrated and validated, and how we define and handle different geometry scenarios in simulation and reconstruction. We discuss how to transparently manage multiple incompatible geometries in the same software release. Several examples are shown based on current implementation assuring consistent choice of scenario conditions. The consequences and implications for multiple/different code algorithms are discussed.

1. Introduction

Anticipating several sub-detector upgrades [1] new geometry designs are being implemented and studied using the CMS software. The CMS geometry configuration software model has been developed to describe multiple geometry scenarios. The compact, or sensitive detector geometry, and the expanded, or Geant4 geometry, version of the scenarios used in simulation and reconstruction have common description of the sensitive detector parts that are physically present in the detector. These parts provide the interface through which calibration and conditions information pass from the physical world into the simulation and reconstruction software. The Detector Description (DD) [2] has been the source for the Simulation and Reconstruction geometry models discussed below. The corresponding in-memory and persistent models of both have been used by the simulation and reconstruction. Presently around 30 geometry scenarios are supported in a single software release. To

[1] To whom any correspondence should be addressed.

guarantee consistent usage of a particular scenario the scenarios have been versioned and labeled. The choice of the scenario is flexible and is an option for a workflow configuration builder.

2. CMS detector description software model

The DD is a software model that represents the description of the 'ideal' CMS detector. Ideal is used to signify that it does not include the detailed alignments of the actual detector. It is the best estimate of the real detectors' shapes as constructed. The model is implemented as a directed acyclic graph with Logical Parts as nodes and Positioning Parts or Algorithms as the edges. An Algorithm in this case is a C++ implementation of a Positioning Part. This provides a hierarchical representation of the detector with parts positioned inside of other parts in a parent-child relationship. This description is used to build the Geometry for both Reconstruction and Simulation within the CMS Software framework.

The time dependent alignment corrections and calibration data of the various sub-detectors are provided by the conditions database. The ideal description is defined using the XML techniques and formats whereas the conditions database format can be different for different sub-detector domains.

A repository for the XML files contains the information needed by the DD to build the in-memory detector description model. An algorithmic capability is implemented in C++ as a set of plug-ins. The Algorithms specific for each sub-detector are implemented deriving from a base DD Algorithm class. Various parameters in the XML file are passed to the Algorithm plug-in for further processing. The balance between how much XML vs. C++ is used to specify the Geometry varies greatly by sub-detector.

3. Simulation and Reconstruction geometries

The Simulation geometry model (Geant4 [3]) requires all parts to be defined in terms of size, shape, material type and position. The Geant4 geometry to be used in simulating the detector is built from the DD compact in-memory model (Fig.1).

The Reconstruction geometry model provides sensitive detector identification numbers as well as extracts from the DD model the relevant coordinate transformations and shape parameters. This part of the Geometry model is needed to reconstruct an event from information provided by the detector. Reconstruction requires the parts of the detector to define hits and tracks. The Geometry model also provides a conversion from local-to-global coordinates of the sensitive detector part in the global CMS coordinate system.

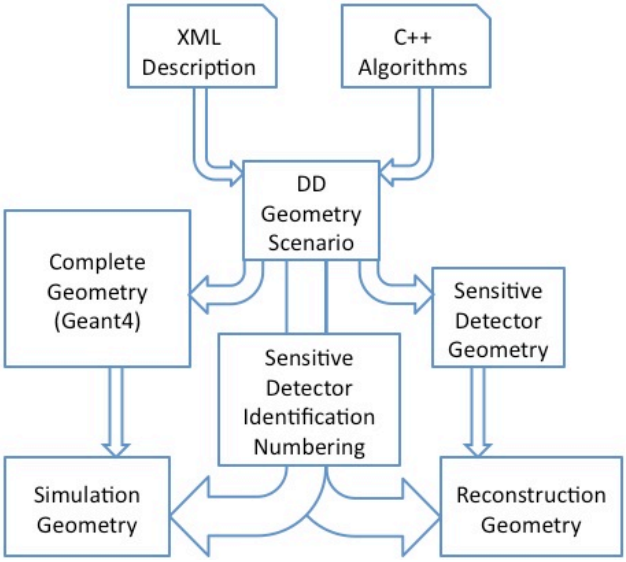

Figure 1. Reconstruction geometry is a subset of a complete geometry for performance reasons. The numbering is shared with Simulation geometry.

3.1. Geometry payloads

The XML is considered the master source of information to build the in-memory model. The sets of XML files provide a scenario for the simulation software. The scenarios are versioned and labeled. For example, the as built CMS detector with the forward detectors is the Extended scenario; without the forward detectors is Ideal.

The in-memory model of the detector description is stored using the CMS conditions database system [4]. The 'master' information from the XML files is loaded into the conditions database as one binary large object (blob), which is a single XML file generated from the component XML files of a scenario. Processing this blob is faster than processing the sets of individual files.

Tools are provided to store the transient geometry XML implementation into persistent objects – records in the database. The records can be accessed via the conditions and calibration framework. Although the persistent record is considered a derivative of the transient one, it can have versions and thus is more flexible than the transient geometry XML files tied to a particular release. One set of loaded objects is used across multiple releases. In theory it means that the XML can change until such a time when a new approved set of payloads or one payload is required to be changed for production or reconstruction.

Intermediate database (DB) objects are created, which are written from Reconstruction objects filled from DD and also read by the geometry builders to directly create Reconstruction geometries. With the intermediate DB objects severing the connection, reconstruction and analysis jobs do not need to link, load, or run DD, thus reducing memory use, CPU time, and increasing flexibility. The Reconstruction geometry records are scenario dependent, and are versioned, however they do not have DB labels associated with them.

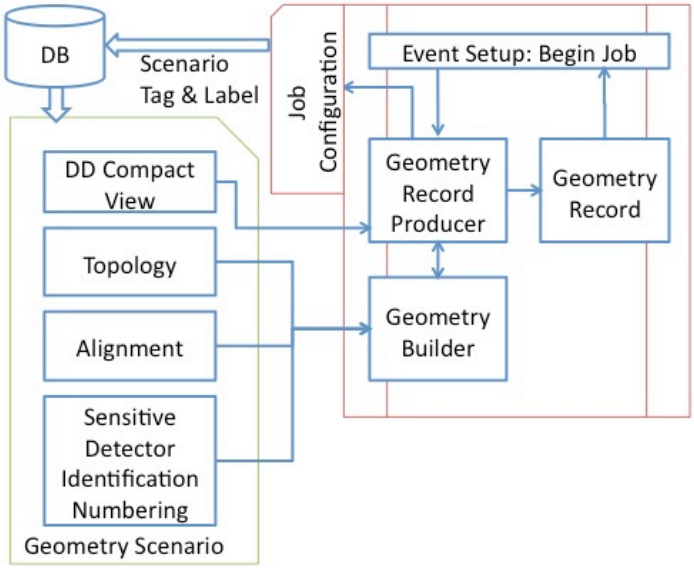

Figure 2. Job configuration defines which scenario is loaded at runtime.

3.2. Integration with the CMS event setup and core software

The compact DD view is loaded to the CMS event setup [5] at the beginning of the job (Fig. 2). The geometry record producer instantiates the geometry builder, retrieves the compact DD view record. The builder creates the geometry record based on current topology and assigns unique identification numbers (DetIds) to the sensitive volumes. The record is returned to the event setup. The event setup provides handles to the records for further access by simulation and reconstruction. The records

register their dependencies with the event setup and are updated during the job if time- and version-dependent descriptions change.

The labels can be assigned to the records, so that multiple records of the same type and the same version can be present in one job. The scenario label and the version are defined by the workflow configuration.

The geometry modules implement the descriptions for its auto-generated Python configuration fragments. The fragments are used in scenario configuration as described in section 4.

3.3. Multiple/different code algorithms

The algorithms describing geometric positioning are implemented as plug-ins. The choice of the positioning algorithm is done in the scenario description. The implications are that the different algorithms should have different names.

The numbering algorithms can be different for different scenarios (sensitive detector identification numbers should allow extra modules, for example). The choice of the numbering algorithm is done based on the scenario topology and the topology is defined by the scenario configuration.

4. Scenario configuration and selection

It is a policy that every scenario is assigned a unique label (Fig. 3). The label is an option for a configuration builder and is passed directly to the DB record loader at runtime. The DB loader chooses the scenario identified by the label and the version aka Global Tag (GT). The GT is applied to a set of records' tags in the DB (see [4] for more details). The GT is also given as an option to the configuration builder and it is used at runtime to load a consistent set of the geometry-related tags.

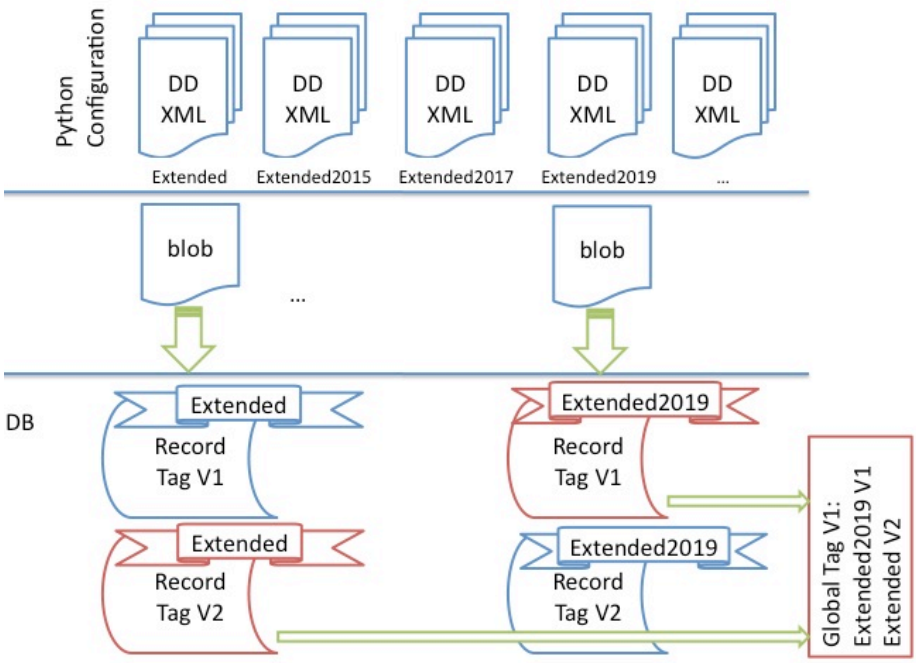

Figure 3. Geometry scenario configuration label corresponds to a year.

The CMS geometry configuration and selection is implemented in Python. The Full geometry configuration is fragmented into a set of modular Python file configurations. The scenario label is described in a Python configuration file fragment parsed by the configuration builder. It is expanded to a set of configuration file fragments describing the modules needed to load the simulation and reconstruction geometries and the numbering. By convention, the label is used in the configuration file

fragment naming, but this is not strictly enforced. The fragments are kept in a standard location - the Standard Sequences Geometry package.

The modular description of the CMS world volume is reflected in the number of the XML files. The file name is used as a namespace. This allows replacing a sub-detector DD in the same parent volume. As the name of the XML file is used as a namespace, it should be unchanged, but the path to the file can vary. Physical separation of the files allows co-existence of multiple incompatible geometries in the same software release. For example several different scenarios for Tracker sub-detectors in Phase 1 and Phase 2 upgrades have been used in one release, but in different workflows. Their payloads to the DB have different tags. The tag of a specific scenario is included in a separate GT that assures a consistent choice of conditions.

The tools collect the Python configuration fragments into a script used in CMS workflow. This flexible and automated geometry configuration allows choosing either transient or persistent version of the same scenario and specific version of the same scenario.

5. Integration and validation

The integration of a new geometry scenario is as follows. The consistency of the XML description is checked where applicable. CMS visualization tools [6] are used to visually inspect the geometry and detect overlaps. The detector identification numbering is checked and the unit tests and the reference tests are run. The standard release validation workflows based on the new in-memory scenario are run. At this point both the scenario and associated C++ code are integrated into an integration build release. The payloads are produced and uploaded to the DB. The new global tag is requested to include the payloads.

Finally, newly defined geometry is integrated to a release and is handled to a validation team which checks data quality and physics based on the new scenarios.

6. Summary

The requirements to maintain multiple Geometry descriptions, flexible access at run-time to a specific Geometry scenario, its conditions and related simulation and reconstruction algorithms have been fulfilled.

The full data workflows based on the geometries have been put in place, data simulated, reconstructed and validated. Flexible configuration to access them has been implemented. Such access has been provided from within one software release. Thus assuring a smooth transition from current CMS geometry description to the subsequent ones planned in the upgrades. In addition, it allows profiting from an ongoing development and improvement of the core, reconstruction and analysis software. It also minimizes manpower needed for maintaining a number of software releases.

References

[1] CMS Collaboration Upgrade of CMS detector through 2020 CERN-LHCC-2011-06
[2] CMS Detector Description: New Developments M Case *et al* 2004 CHEP04 conference proceedings
[3] Geant4 hhp://geant4.cern.ch
[4] Alignment and calibration of CMS detector during collisions at LHC ID 153 this conference proceedings
[5] Access to Non-Event Data for CMS C Jones 2006 CHEP06 conference proceedings
[6] Multiple-view, Multiple-selection Visualization of Simulation Geometry in CMS L A T Bauerdick *et al* 2012 *J. Phys.: Conf. Ser.* **396** 022052