Perspectives for the migration of the LHCb geometry to the DD4hep toolkit

Silvia Borghi[2], *Chris* Burr[2], *Marco* Clemencic[1], *Gloria* Corti[1], *Ben* Couturier[1], *Markus* Frank[1], *Lucia* Grillo[2], and *Dominik* Muller[1]

[1]CERN
[2]The University of Manchester

Abstract. The LHCb experiment uses a custom made C++ detector and geometry description toolkit, integrated with the Gaudi framework, designed when the LHCb software was first implemented. With the LHCb upgrade scheduled for 2021, it is necessary for the experiment to review this choice and adapt to the evolution of software and computing (in terms of e.g multi-threading support or vectorization)
The Detector Description Toolkit for High Energy Physics (DD4hep) is a good candidate for the replacement for LHCb's geometry description framework: it is possible to integrate it with the LHCb core software framework and its features theoretically match the requirements: in terms of geometry and detector description but also concerning the possibility to add detector alignment parameters and the integration with simulation tools.
In this paper we report on detailed studies undertaken to compare the feature set proposed by the DD4hep toolkit, to what is needed by LHCb. We show not only how the main description could be migrated, but also how to integrate the LHCb real-time alignment tools in this toolkit, in order to identify the main obstacles to the migration of the experiment to DD4hep.

1 Introduction

The Large Hadron Collider beauty (LHCb) experiment studies b-quark and c-quark decays, and performs precision measurements in the forward direction at the Large Hadron Collider at CERN. It consist of a single-arm spectrometer with excellent vertexing and momentum resolution capabilities, and will be upgraded in 2019/2020 to improve its capabilities further [1]. A detailed and precise description of the detector is crucial to understand the phenomena occurring within the detector itself, both for simulations of the detector and to evaluate the material crossed by the particles for tracking purposes. The original LHCb detector description framework is a custom development that was tailored to the experiment's needs, in the early days of the LHCb software stack development (early 2000s). While this framework has served its original purpose well since then, it also has shown many limitations. As LHCb is being upgraded for the LHCb Run3 and beyond, it was therefore decided to take this opportunity to review alternative frameworks. This paper presents the process followed for this study and the current LHCb plans.

2 Current framework and alternative

2.1 The LHCb Detector Description Framework

The LHCb Detector Description framework is a C++ framework developed to fulfil the expected needs of the experiment, and was first presented at CHEP 2003 [2]. The geometry is modelled using Constructive Solid Geometry, in a similar way as what is done in the Geant4 Simulation Toolkit [3], or in the ROOT Data analysis framework [4] [5]. The detectors themselves are modelled by Detector Element classes (logical structure), that reference the geometry structure as shown in figure 1.

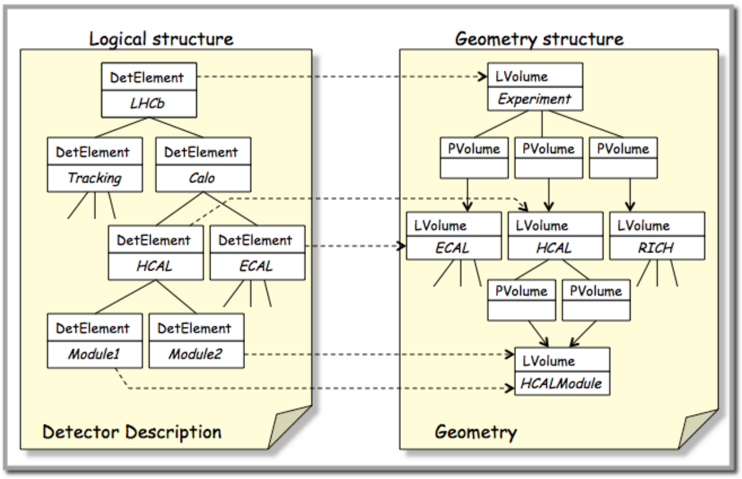

Figure 1. LHCb Detector Description instance diagram

The geometry is persisted as a set of XML files (in LHCb DetDesc XML format) in the LHCb conditions database [6]. Over years of use it was noted that the main drawbacks of the system were:

- The complexity of the description of repeated sensors as everything is described in XML, with no easy looping constructions in the language;
- the difficulties in implementing alternative geometries (e.g. with coarser resolution or with a simplified description of LHCb, as currently used in tracking algorithms) as this implies creating another (unrelated) detector description.
- a design not suitable to multi-threaded architecture, requiring adaptations to run in such an environment.

The LHCb software is evolving to adapt to many core architectures, as explained in he LHCb Upgrade Software and Computing TDR [7]. It is now better to evaluate other geometry frameworks than continue the development of the LHCb one.

2.2 The DD4hep toolkit

The DD4hep toolkit [8] [9] builds on the experience gathered by LHC experiments and aims to bind the existing tools for detector description, simulation and visualisation to produce a consistent toolkit. Its structure is modular (c.f. Figure 2 taken from [10]), with a generic detector description model as its core, and a set of non-mandatory extensions or plugins that can be used when needed. The object model and visualisation are provided by the ROOT framework. For simulations, the Geant4 toolkit is used. As both tools are already used by LHCb, DD4hep is therefore a natural candidate for the replacement of the detector description.

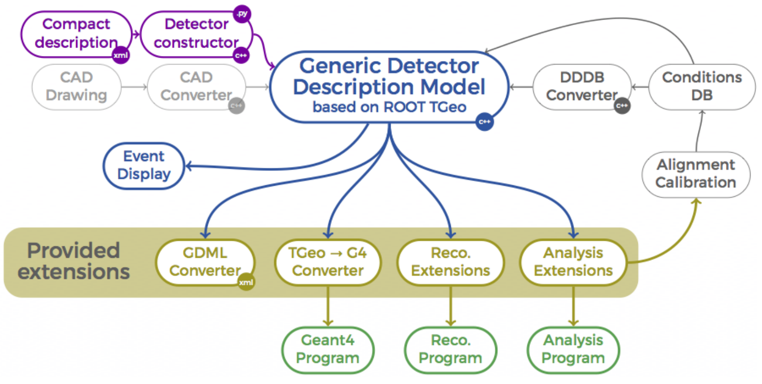

Figure 2. Logical structure of the DD4hep toolkit. This diagram shows the relationships between the core (the generic detector description model) and the extensions that can be used according to the needs, taken from [10]

It was first evaluated by LHCb as presented at CHEP 2015 [11]. That study showed that while the integration is possible, porting the LHCb detector description to DD4hep is nonetheless a substantial endeavour due to the difference in design between the frameworks. Even with this perspective in mind, we decided to keep investigating the possible migration paths.

3 DD4hep integration prototype

3.1 DDDB importer in DD4hep

The LHCb Detector Description is stored in the Detector Description Database (DDDB). As a first step to the migration of this database, the "DDDB" DD4hep component was developed that allows importing the LHCb DDDB in DD4hep. The figure 3 shows a visualization of the current LHCb detector in DD4hep.

It however appeared that the imported model was not completely correct (some volumes were not correctly placed) and needed further validation.

3.2 DDDB Validation prototype

A first prototype was created to allow having in memory both the LHCb representation and the DD4hep one (using the ROOT TGeo classes), but for the LHCb Upgrade detector. A comparison of the two representations was done either by:

- Comparing the volume trees by iterating on all volumes and their placements in parallel in both hierarchies. Thanks to the similarity in design, we could check that all volumes were created, but found some discrepancies in the positioning.
- Comparing the material on a given path: the LHCb framework allows checking which volumes are crossed along a specific trajectory, and returns the list of materials crossed and their width. We implemented the same functionality using the TGeo classes, and found

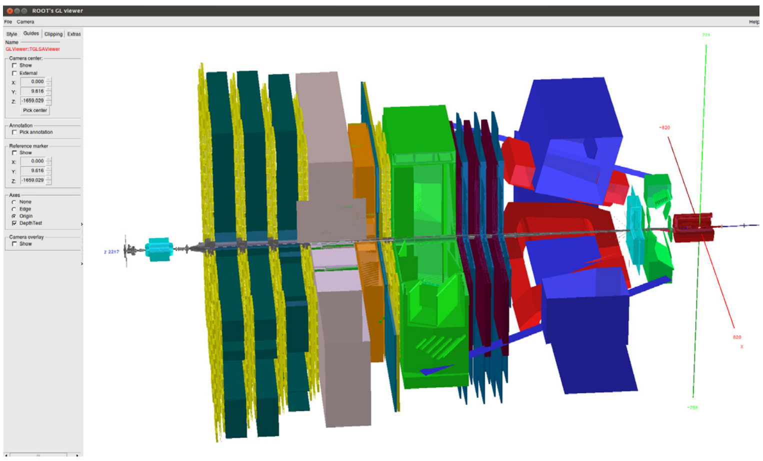

Figure 3. Visualization of the LHCb detector where the description has been imported into DD4hep using the DDDB converter

agreement in some parts (e.g. the upgrade vertex locator, VeloPix [12] sub-detector) but found many issues in others (such as the magnet representation). This method is good for validation, but does not allow identifying the reasons for specific errors.

Investigations of discrepancies found that:

- the expression evaluation was designed differently in LHCb DetDesc XML and DD4hep Compact XML: in the former, the expressions defined are evaluated on-demand (i.e. lazily), therefore no explicit order is required when defining the various expressions, provided that they are loaded in the right order. In the latter, the expressions are evaluated at load time (i.e. greedily) thus requiring the variables defined in order. These are design choices that cannot be changed easily but have a significant impact on the actual implementation of the description. One workaround to be able to load the LHCb detector description through the DD4hep DDDB loader, is to predefine the missing expressions in an XML file loaded first, but this approach is dangerous as there is no scoping of the variables and names can be reused. Another approach is to reorder the LHCb definitions. The detector loaded is then different for the one currently used for simulation, which weakens the result of the comparison.
- LHCb DetDesc XML uses the possibility offered by the XercesC XML parser [13] to redefine XML entities. It uses this functionality to import fragments of XML into documents. While this approach eases factorisation of parts of XML, it also makes problem identification difficult as the original source file is lost. Furthermore, many files in the LHCb repository contain XML fragments and not XML documents.

These points show that while this test was necessary, a more robust approach has to be found for the migration of the schema.

3.3 Detector elements and misalignment functionality

A second goal of the prototype was to check the functionality offered by DD4hep to develop the representation of active detectors themselves (called detector elements) and to map them to the constructed geometry. Those classes need to be customizable, and also need to integrate with the database containing the state of the detectors, for example the actual misaligned positions as determined by the alignment procedures.

DD4hep offers equivalent functionality to what is provided with the LHCb detector description. Once the geometry is loaded, individual detector elements can be misaligned. These misalignments will be reflected on the detector element tree (logical structure), but not on the geometry structure (c.f. figure 1). For simulation purposes, it is possible to misalign the geometry structure while converting it to Geant4. In the LHCb code base, this is done by the LHCb Simulation application [14]. DD4hep can use the TGeoPhysicalNode functionality of the ROOT geometry and export the non-ideal geometry to Geant4.

The DDAlign package of DD4hep [15], see figure 4, fulfils this need and can be used by LHCb.

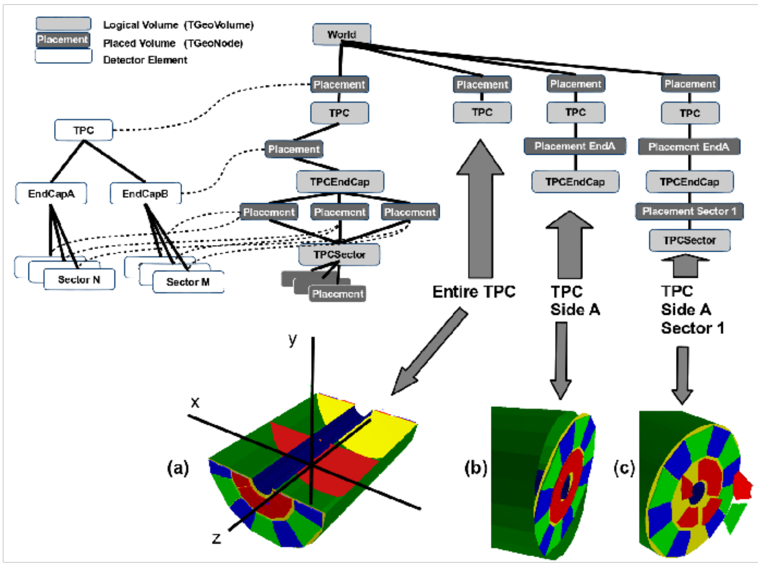

Figure 4. DDAlign functionality: this diagram from the DDAlign manual [16] shows how the toolkit deals with misalignment at different levels, in terms of class instances, for an example TPC detector

3.4 Conclusion

The DDDB converter and the validation prototypes showed that DD4hep fulfils the need of the LHCb experiment. However, it also put in evidence the fact that the automated migration of the geometry representation cannot be fully automated and that a manual conversion will be needed, both to ensure the fidelity of the conversion and to take advantage of the DD4hep features.

4 Migration plan and future work

4.1 DD4hep Compact XML and GDML as bridge

DD4hep does not mandate a specific representation, but features by default the Compact XML format:

- Detectors are represented as XML files, as in the LHCb case, and can be constructed from XML files only.
- A C++ constructor can be associated to each XML file thus adding the possibility to have complex logic dictating the creation of a detector. This can be used to build geometries with different resolutions depending on the purpose: the DD4hep detector features a "build_type" enum which can be used to specify profiles. Furthermore, variables can be added to the evaluator and checked by the various constructors to determine whether to build a full or simplified geometry.

Direct migration of the detector is difficult, but can be eased by the Geant4 XML persistency format (GDML):

- The LHCb simulation application converts the LHCb memory model to Geant4 objects, and this object tree can be exported directly to GDML. This export is the reference as it is used for Monte Carlo simulations.
- GDML can be imported back to Geant4 and to ROOT (modulo little issues concerning the "opticalsurface" objects used by the LHCb RICH detectors, that will be fixed in ROOT).

A Gaudi [17] service has been written to load the DD4hep geometry into the LHCb event processing application. This code will be merged with the LHCb upgrade main code branch as soon as it is ready. The plan is to:

- Split the GDML into each sub-detector, and write DD4hep constructors to load them into the geometry tree. This will allow working independently on the migration of each sub-detector, while having a full view of the detector. Furthermore, the validation of the migrated geometry will be eased by the fact that we will have to compare two TGeo object trees.
- Rewrite some of the structures like the beam-pipe, magnet, and BCMs: as they are defined as "assemblies" (i.e. group of shapes without external envelope), they are attached directly to the world volume by Geant4, thus making them difficult to import back.
- Rewrite one detector to identify best practices and recipes, in order to guide future developments which will have to be done by the sub-detector themselves.

4.2 Run1 and 2 geometry

Porting the LHCb geometry to DD4hep is a considerable effort, and it needs to be done for the representation of the LHCb Upgrade detector for Run3.

Migrating the representation of the Run1 and Run2 detector would be the same amount of work, and would also imply extra risks in case of migration problems. In order to continue with the analysis of Run1/Run2 data, LHCb needs to keep on running simulations (with updated simulation software), while filtering and reconstructing with the software used at the time of data taking (therefore using the old geometry framework anyway).

A convenient solution is to export the Run1/Run2 geometry to GDML for each needed set of conditions (in practice we use one set of conditions per year of data taking) and modify Gauss, the LHCb simulation, to load directly from a GDML file instead of converting a geometry. Loading from GDML allows simulating the detector, but does not provide detector element objects that allow interacting with the sub-detectors. This is however not an issue as the reconstruction and filtering would be done by the triggering software used in Run1/Run2 that uses the LHCb DetDesc.

4.3 Benefits of the migration

While a huge endeavour, the migration will have a number of benefits for the LHCb experiment:

- LHCb currently uses two geometries: a detailed geometry and a simplified one for tracking in the high level trigger (tracking in the full resolution geometry is much too slow). Simulation can be done on part of the detector by configuring the simulation application to avoid converting some sub-detectors to the Geant4 model used for simulation. Properly used, Compact XML C++ constructors can be used to produce alternative views of the geometry (with some parts simplified) to allow for faster simulations. As Monte Carlo simulations represent more than two-thirds of the CPU usage on the grid, this can have a large impact for the experiment in terms of required offline resources.
- DD4hep is a common framework, under development. The LHCb experiment will therefore be able to share developments and make use of new features brought by the toolkit: e.g. it will be possible to evaluate easily the navigation using the VecGeom vectorized geometry [18].
- LHCb currently uses its own visualisation tools based on the OpenScientist [19] framework. This framework is no longer actively developed and using the ROOT Geometry framework would also allow using the latest features, e.g. the visualisation in a browser using jsroot [20] and three.js [21].

5 Conclusion

The detector description is a key component of any High Energy Physics experiment software framework. Porting the geometry to a new framework is an endeavour, but staying stuck on a very old framework is also problematic as it prevents adapting to new tools, and results in many missed opportunities. For LHCb, moving to the DD4hep framework represents a significant amount of work, but will also add desired features to the framework. Furthermore, as the Geant4 simulation toolkit is already used by LHCb, its XML persistency format (GDML) is a good intermediate format to help with the migration.

References

[1] I. Bediaga, J.M. De Miranda, F. Ferreira Rodrigues, J. Magnin, A. Massafferri, I. Nasteva, A.C. dos Reis, S. Amato, K. Carvalho Akiba, L. De Paula et al. (LHCb collaboration), Tech. Rep. CERN-LHCC-2012-007. LHCb-TDR-12 (2012), `http://cds.cern.ch/record/1443882`

[2] S. Ponce, Tech. Rep. LHCb-PROC-2003-004. CERN-LHCb-PROC-2003-004, CERN, Geneva (2003), `https://cds.cern.ch/record/1496875`

[3] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

[4] R. Brun, F. Rademakers, S. Panacek, Conf. Proc. **C000917**, 11 (2000)

[5] F. Rademakers, P. Canal, A. Naumann, O. Couet, L. Moneta, V. Vassilev, D. Piparo, G. GANIS, B. Bellenot, wverkerke et al., *root-project/root: v6.16/02* (2018), `https://doi.org/10.5281/zenodo.1292566`

[6] M. Clemencic, *A Git-based Conditions Database backend for LHCb* (2018), `http://cds.cern.ch/record/2631775`

[7] C. The LHCb Collaboration, Tech. Rep. LHCb Upgrade Software and Computing TDR. CERN-LHCC-2018-007. LHCB-TDR-017, CERN, Geneva (2018), `http://cds.cern.ch/record/2310827`

[8] N. Nikiforou, M. Frank, F. Gaede, S. Lu, M. Petric, A. Sailer (2015)

[9] M. Frank, F. Gaede, M. Petric, A. Sailer, *Aidasoft/dd4hep: v01-09* (2018), this project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168., `https://doi.org/10.5281/zenodo.1464634`

[10] M. Petrič, M. Frank, F. Gaede, S. Lu, N. Nikiforou, A. Sailer, Journal of Physics: Conference Series **898**, 042015 (2017)

[11] M. Clemencic, A. Karachaliou, J. Phys.: Conf. Ser. **664**, 072012. 5 p (2015)

[12] L. Collaboration, Tech. Rep. LHCb VELO Upgrade Technical Design Report CERN-LHCC-2013-021. LHCB-TDR-013 (2013), `https://cds.cern.ch/record/1624070`

[13] *The apache foundation, xercesc xml parser*, `http://xerces.apache.org/xerces-c/`

[14] S. Roiser, Tech. Rep. LHCb-PROC-2003-005. CERN-LHCb-PROC-2003-005, CERN, Geneva (2003), `http://cds.cern.ch/record/1498606`

[15] M. Frank, Tech. Rep. AIDA-2020-NOTE-2016-003, CERN, Geneva (2016), `http://cds.cern.ch/record/2212533`

[16] AIDA2020, *Ddalign manual* (2019), `https://dd4hep.web.cern.ch/dd4hep/usermanuals/DDAlignManual/DDAlignManual.html`

[17] G. Barrand, I. Belyaev, P. Binko, M. Cattaneo, R. Chytracek, G. Corti, M. Frank, G. Gracia, J. Harvey, E. Van Herwijnen et al. (2000)

[18] S. Wenzel, Y. Zhang (VecGeom Developers), J. Phys. : Conf. Ser. **898**, 072032. 8 p (2017)

[19] G. Barrand, *OpenScientist. Status of the project* (2005), `http://cds.cern.ch/record/865623`

[20] *The jsroot javascripts library*, `https://root.cern.ch/js/`

[21] *The three.js javascript library*, `https://threejs.org/`