EPJ Web of Conferences **245**, 02032 (2020) https://doi.org/10.1051/epjconf/202024502032
CHEP 2019
FERMILAB-CONF-20-736-V

CMS Experience with Adoption of the Community-supported DD4hep Toolkit

Carl Vuosalo[1,*], *Sunanda* Banerjee[2], *Markus* Frank[3], *Vladimir* Ivanchenko[3,4], *Sergio* Lo Meo[5,6], *Ianna* Osborne[7], and *Andres* Vargas Hernandez[8]
on behalf of the CMS Collaboration

[1]Dept. of Physics, University of Wisconsin-Madison, USA
[2]Fermi National Accelerator Laboratory, USA
[3]CERN, Geneva, Switzerland
[4]Tomsk State University, Russia
[5]ENEA, Bologna, Italy
[6]INFN Sezione di Bologna, Italy
[7]Princeton Institute for Computational Science & Engineering, Princeton University, New Jersey, USA
[8]The Catholic University of America, Washington D.C., USA

Abstract. DD4hep is an open-source software toolkit that provides comprehensive and complete generic detector descriptions for high energy physics (HEP) detectors. The Compact Muon Solenoid collaboration (CMS) has recently evaluated and adopted DD4hep to replace its custom detector description software. CMS has demanding software requirements as a very large, long-running experiment that must support legacy geometries and study many possible upgraded detector designs of a constantly evolving detector that will be taking data for many years to come. CMS has chosen DD4hep since it is a high-quality, community-supported solution that will benefit from continuing modernization and maintenance. This presentation will discuss the issues of DD4hep adoption, the advantages and disadvantages of the various design choices, performance results, and the integration of the plugin systems from CMS and Gaudi, another open-source software framework. Recommendations about DD4hep based upon the CMS use cases will also be presented.

1 Introduction

The design and simulation of a complex particle detector requires advanced software. In particular, software that describes and represents the detector is needed. The Compact Muon Solenoid (CMS) detector [1] is one of the most complex particle detectors ever built. In coming years most of the detector will be upgraded. These planned upgrades go through many design iterations and versions. Each upgrade design must be simulated and assessed, which means the detector description software must support not just one complete description of the complex CMS detector, but many different versions of it. In addition, the CMS experiment and the software to support it is expected to continue for the next twenty years or so, and the software must keep up with advances in technology over this long period.

*e-mail: covuosalo@wisc.edu

This document was prepared by CMS Collaboration using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359.

In years past, the CMS collaboration developed its own custom detector description (DD) software, but this code has fallen behind other advances in CMSSW, the CMS software system. In particular, CMSSW adopted multi-threaded processing to enhance performance, but the CMS DD is a singleton that does not support multi-threading. The ability to build components of the CMS detector geometry in parallel would help improve performance, but it is not possible with the existing CMS DD. In addition, the CMS DD software has not kept pace with advances in C++ standards and has accumulated obsolete sections over the years, which would only add to the cost and difficulty of enhancing it. Rather than update its own DD software, which would also entail continuing maintenance costs, the CMS collaboration looked for an external solution.

The high energy physics (HEP) community has long promoted the development of shared software toolkits to support HEP experiments. Rather than each HEP collaboration independently developing very similar software, shared toolkits allow the collaborations to save on software development and maintenance costs while benefiting from community expertise.

DD4hep [2] is one such shared toolkit. It is a full-featured detector description toolkit that supports multi-threading. Its detector geometry implementation is based upon ROOT [3], another toolkit widely used in HEP. It has been used by the Compact Linear Collider, Future Circular Collider, and the International Large Detector collaborations for studies of proposed particle detectors. DD4hep benefits from innovations and contributions from across the HEP community. It has dedicated funding that will ensure it will continue to develop and evolve with advancing technology. The development team is centrally located at CERN.

DD4hep was a natural fit for CMS's need for DD utilities, but an evaluation process was necessary to ensure that DD4hep would meet all of CMS's requirements for performance, memory usage, multi-threading, navigation, and functionality. The evaluation period started in January 2018. A test migration of a small package in CMSSW was performed, and validation and performance tests were performed to ensure that DD4hep could reproduce the original behavior of the code with acceptable performance. The evaluation was completed in December 2018 with the conclusion that DD4hep would meet all of the requirements of CMS.

2 Benefits of Migration

DD4hep's support for multi-threading advances a CMS goal of making CMSSW more fully multi-threaded. A single-threaded step in an otherwise multi-threaded program creates slowdowns and complicates thread scheduling as threads have to wait at the single-threaded bottleneck [4]. With DD4hep, the geometry building step of event simulation jobs will avoid these problems and may even gain a slight performance boost from parallel building of subdetectors.

Another benefit of migration is relieving CMS of the maintenance burden of DD software. Also, the migration gives the opportunity to improve the CMSSW code base, to drop obsolete features, and to refine the geometry description of the detector. Lingering overlaps of geometric volumes can be fixed, and testing and validation processes can be enhanced. In addition, the migration helps build the expertise of the developers involved.

On the other side, the adoption of DD4hep by CMS has prompted the improvement of the DD4hep and the underlying ROOT toolkits as the development teams of the two projects endeavored to meet CMS needs.

This migration shows the value of community-supported software. CMS benefits, DD4hep has been improved, and the HEP community can take advantage of a powerful software toolkit that is getting better and better. The HEP community faces big computing challenges. It can only hope to meet these challenges by working together and pooling

efforts. The community is turning the page on past practices, when each HEP experiment would develop all their software themselves.

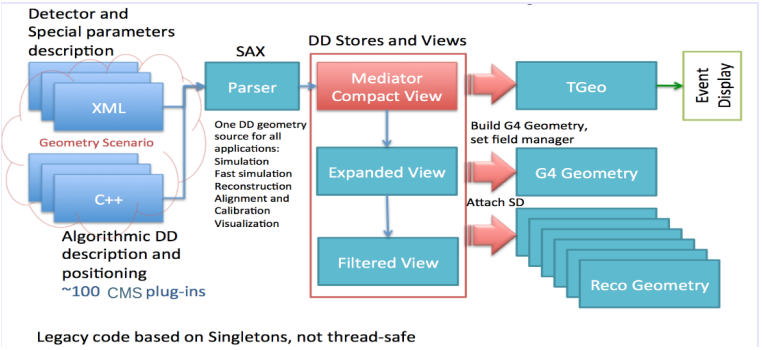

Figure 1. Diagram showing the existing CMS detector description. The old DD uses a SAX (Simple API for XML) parser, which parses XML step-by-step in a state-independent manner. In this diagram, "G4" refers to the Geant4 simulation toolkit, "Reco" refers to particle reconstruction, and "TGeo" refers to the ROOT geometry classes.

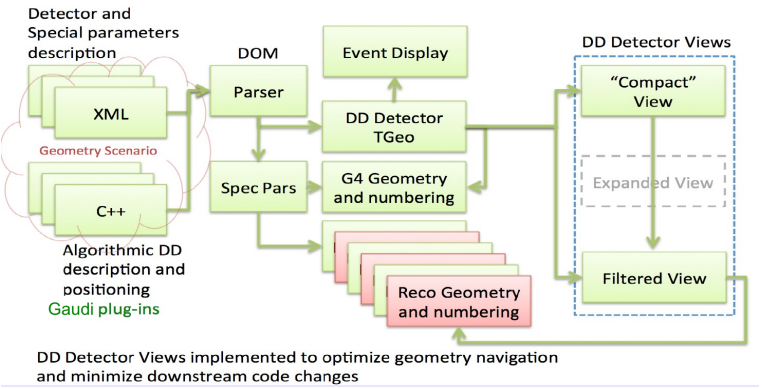

Figure 2. Diagram showing the CMS detector description using DD4hep. DD4hep uses a DOM (Document Object Model) parser that parses the XML document all together in a state-dependent manner. In this diagram, "G4" refers to the Geant4 simulation toolkit, "Reco" refers to particle reconstruction, and "TGeo" refers to the ROOT geometry classes.

Figures 1 and 2 compare features of the old DD and the new DD with DD4hep.

3 Migration Scope and Timeline

The DD4hep migration by CMS has been underway since January 2019. As 2020 begins, 80% of the migration is complete. As the final packages are being migrated, validation to ensure that DD4hep reproduces the original geometry and results is underway. In early 2020 optimization will commence to bring performance of the DD4hep DD up to the level of the old DD in terms of speed and memory use and to remove unneeded legacy features. In mid-2020, the plan is for full event simulation and reconstruction to be performed solely with DD4hep. At the end of the year, the goal will be the elimination of the old DD software.

The scope of the migration encompasses only a fraction of the 6.5 million lines of code in CMSSW, which is mostly C++ but also includes Python and XML. It is the DD code that must be migrated, comprising about 150,000 lines of C++ and Python code and several hundred files. Fortunately, not all this code needs to be changed, but all of it needs at least to be reviewed. In addition, there are 1.5 million lines of XML detector geometry description, but only the most limited fixes are needed here. Tied to the XML are 61 C++ algorithms that must be migrated.

One simplifying factor for the migration is that CMS preserves older reference versions of CMSSW for processing old data and generating simulated events with previous iterations of the CMS detector. This fact allowed the migration to focus on current and future versions of the detector, though DD4hep should be able to reproduce old detector geometries if desired.

The migration effort involves developers working part-time from eight groups of sub-detector and other specialists from across CMS over the 2019-2020 period. This effort has been distributed and cooperative, instead of being done by a directly managed team dedicated exclusively to the task.

4 Migration Strategies

Several strategies have been employed in the migration process. During the evaluation phase, migrated code was kept in a separate package, and old code was left untouched. With the commencement of the actual migration, the migrated code is being put into the mainline branch, known as the integration build. This approach enables easy access to migrated code among developers, and it also subjects the migrated code to daily testing and validation. The migration process has been very incremental, with the preference for small, step-wise changes that are easier to review and test.

For migrating code files, three techniques are used. For some code, migrated versions of files are put into "dd4hep" directories or given file names starting with "DD4hep_". Then a Python script is configured to load either the old version or the DD4hep version. Another technique is to have sections of a file activated by a "fromDD4hep" boolean flag and thereby combine old and migrated code in the same file. Finally, template classes provide another method to represent old and migrated code.

During migration two goals need to be balanced. On one hand, it is important to preserve the original behavior of the code by minimizing changes so reliable validation can be performed. On the other hand, code duplication needs to be minimized.

5 Integration of DD4hep

DD4hep is handled as an external tool in CMSSW. CMS keeps up with the latest DD4hep releases, and DD4hep is built by the CMSSW build system. As a consequence, when DD4hep recently changed its build configuration, CMS needed to make fixes in order to build the new version.

Plug-ins provide a defined mechanism for programs to load libraries at run-time. DD4hep uses the Gaudi format for plug-ins, which utilizes a particular template class, while CMS has its own plug-in format, which relies upon a C++ macro. For the migration, CMS enhanced the CMSSW build system to support the Gaudi plug-in format.

Figure 3 shows the complex geometry of the full CMS detector model for 2021, as built with DD4hep.

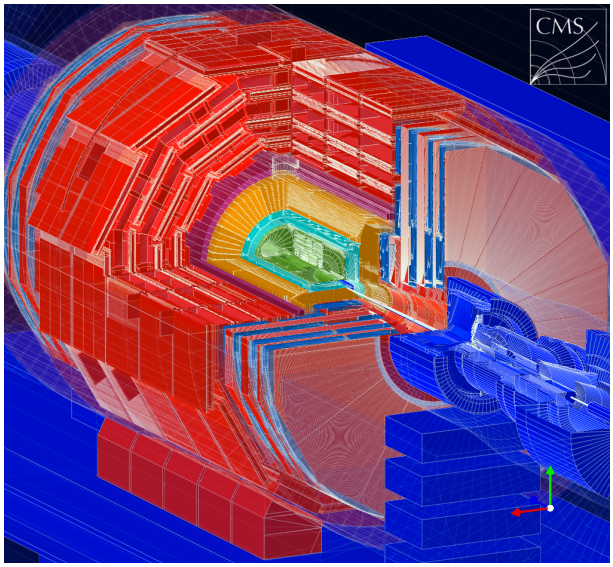

Figure 3. CMS detector model for 2021 built with DD4hep

6 Migration Challenges

The migration presented many challenges. In the evaluation phase, CMS developed a comprehensive requirements document that was discussed with the DD4hep and ROOT development teams to identify missing features and capabilities. One conclusion of these discussions was that DD4hep lacked several key features required by CMS. First, CMS required support for three special shapes: two versions of truncated tubes and a trapezoid combined with a cylinder. Second, CMSSW required support for Geant4 [5] version 10.4. Third, support for two different unit conventions was needed: the Geant4 convention where a millimeter equals 1, and the ROOT convention where a centimeter equals 1. Fourth, the ROOT "TGeo" geometry classes used by DD4hep were not thread-safe. Fifth, CMS uses left-handed coordinate systems for representing sub-detectors that have two mirror-image sides, but DD4hep did not support left-handed coordinate systems. Sixth, CMSSW needs to build both dynamic and static versions of DD4hep libraries, but DD4hep could not be built as a static library.

The solutions in all these cases were that DD4hep was enhanced to support the features needed by CMS. The DD4hep developers made several of the enhancements, but in some cases, the issue lay in the underlying ROOT architecture, and in these cases the DD4hep and ROOT development teams worked together to implement the enhancements. The result was that both DD4hep and ROOT were improved.

Further challenges were overcome by CMS developers. CMS uses DD4hep as a mediator between its DD primitives stored in XML files and the simulation and reconstruction applications. CMS needed to develop a thin layer to handle communication between DD4hep and CMSSW. CMS required a hierarchical querying and filtering mechanism for navigating through detector volumes based upon special parameters defined in the XML files, and this code had to be developed as an extension to the basic DD4hep detector representation. Additionally, CMS code was needed to define geometric volumes that act as sensitive detectors and to set CMS-specific restricted values used in simulating particle interactions with materials. The old CMS DD allowed reference to undefined geometric objects in XML files as

long they were eventually defined, but DD4hep requires all objects to be defined before being referenced. CMS had to develop code to safely allow processing of legacy XML files with undefined references. Another problem was that in old CMS XML files certain conical shapes called polycones were incorrectly defined, and these definitions had to be fixed. Finally, the CMS developers faced the general problem of migrating, testing, and validating lots of old, partially documented code.

7 Good Practices for Migration

The process of CMS adopting DD4hep demonstrates several good practices for migrating to the use of any software toolkit. Before a toolkit is adopted, it must be carefully evaluated, and a test migration should be performed to ensure that it will meet all requirements. Second, in the migration process, particular attention must be paid to special exceptions and peculiar cases in the legacy code. It is these special exceptions that will take most time to migrate, compared with standardized code. Dropping the special features is quickest option, but if the special feature must be retained, then sufficient resources must be allocated to handle its migration. Third, developers should be provided with migration examples and instructions to help them. Fourth, active engagement with the toolkit developers to enhance the toolkit itself is very beneficial. And fifth, the migration should be viewed as an opportunity for overall software improvement. While code is being migrated, it can also be checked and modernized, and obsolete features can be eliminated.

Another important practice for successful migration is thorough validation of migrated code. CMS uses a multi-step process of testing and validation. Any new or revised code for CMSSW must be reviewed and approved by domain experts and pass a series of automated tests that enforce CMS coding standards and ensure that there are no unintended changes to previous behavior and results from the software. New code is required to come with unit tests that check its functionality, and these tests then become part of the automated testing process. Nightly builds of the entirety of CMSSW are performed along with the entire suite of tests. Benchmark releases of CMSSW are made at regular intervals and carefully checked by experts for consistency with past physics results.

8 Conclusion

DD4hep is a full-featured, powerful toolkit for detector description. It is widely used in the HEP community, and it will continue to benefit from innovations, enhancements, and maintenance for many years to come. Its development team is very responsive to user needs.

The adoption of DD4hep by CMS is a success story for HEP community-supported software. DD4hep shows it can support the highly complex geometry of the CMS detector, and the migration process has improved CMS software and DD4hep itself. Community-supported software provides big benefits for the HEP community and is a key tool for meeting upcoming major computing challenges.

9 Acknowledgments

The DD4hep project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

References

[1] The CMS Collaboration, Journal of Instrumentation **3**, S08004 (2008)

[2] M. Frank, F. Gaede, M. Petric, A. Sailer, AIDASoft/DD4hep (2018), webpage: http://dd4hep.cern.ch/, https://doi.org/10.5281/zenodo.592244

[3] R. Brun, F. Rademakers, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A **389**, 81 (1997)

[4] C.D. Jones on behalf of the CMS Collaboration, Journal of Physics: Conference Series **898**, 042008 (2017)

[5] S. Agostinelli et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)