CAD support and new development in DD4hep

Markus Frank[1], *Frank* Gaede[2], *Marko* Petrič[1,*], and *André* Sailer[1]

[1]CERN, CH-1211 Geneva 23, Switzerland
[2]DESY, Notkestraße 85, D-22607 Hamburg, Germany

Abstract. Consistent detector description is an integral part of all modern experiments and also the main motivation behind the creation of DD4hep, which tries to address detector description in a broad sense including geometry and the materials used in the device, additional parameters describing e.g. the detection techniques, constants required for alignment and calibration, description of the readout structures and conditions data. A central component of DD4hep is DDG4, which is a mechanism that converts arbitrary DD4hep detector geometries to Geant4 and provides access to all Geant4 action stages. In addition to that DDG4 also offers a comprehensive plugins suite that includes handling of different IO formats, Monte Carlo truth linking and a large set of segmentation and sensitive detector classes, allowing the simulation of a wide variety of detector technologies. One of the last remaining open issues of detector description was support for drawings from civil engineers for passive detector components. In this proceedings we highlight recent developments in DD4hep/DDG4 that enable support for CAD drawings and generic tessellated shapes and through the help of the library `assimp` enable the import of a wide variety of CAD formats, thus eliminating the need for writing complex re-implementations of CAD drawings in source code. In addition, we present other developments such as support for a new output format called EDM4hep and developments for a more unified and easier handling of units.

1 Introduction

The DD4hep software toolkit [1] targets the unification of detector description at all stages of the experimental lifecycle. The objective is to extract consistent information from a single source and deliver it to simulation, reconstruction and analysis steps in data processing. The advantages of such a consistent, comprehensive description are clearly established from experience in high energy physics experiments.

The design choices and foundations for DD4hep were based on experience from the LHCb experiment [2] and the efforts of the Linear Collider Community [3]. The aim is to deliver on one side the geometrical and material properties, and on the other side also provide parameters such as alignment constants and calibration, description of the readout structures and conditions, and those directly associated with detection techniques. The development of DD4hep was always guided by the desire to provide widely usable generic software to achieve a comprehensive detector description. The two main dependencies of DD4hep are

*e-mail: marko.petric@cern.ch

ROOT, which is used for the construction and visualization of the detector geometry [4], and Geant4 which is used to perform detector simulation via an interface [5].

The working principles of DD4hep [6] and details about simulation [7] have been summarized at previous conferences [8]. The aim of these proceedings is to present Tessellated Shapes and CAD support as well as highlight the most relevant changes that occurred since release DD4hep v01-11, the current realease is v01-16.

2 Main objectives

The objectives which DD4hep aims to cover are:

- **Complete detector description:** A geometry description comprising all structures and the matching materials, the attributes necessary for visualization, the parameters needed for alignment, calibration, and environment description, as well as detector readout information.
- **Coverage of the full life cycle of an experiment:** Providing functionality for all stages in the life-cycle of the experiment, from concept development, through construction to operation, providing a smooth transition between various stages.
- **Single source of information:** Containing all of the required information for interpretation of data in all stages, from simulation to reconstruction and analysis.
- **Ease of use:** Exposes to the final user a simple and intuitive interface, which requires minimal external dependencies.

3 The DD4hep toolkit

At the core of DD4hep lies the Generic Detector Description Model (GDDM) which is built around the ROOT TGeo package. The GDDM is an in-memory model, it is comprised of a set of objects containing geometry and auxiliary information about the detector. Figure 1 shows a diagrammatic depiction of the various components of DD4hep, the interactions between them and the end-user applications, such as alignment, visualisation and reconstruction.

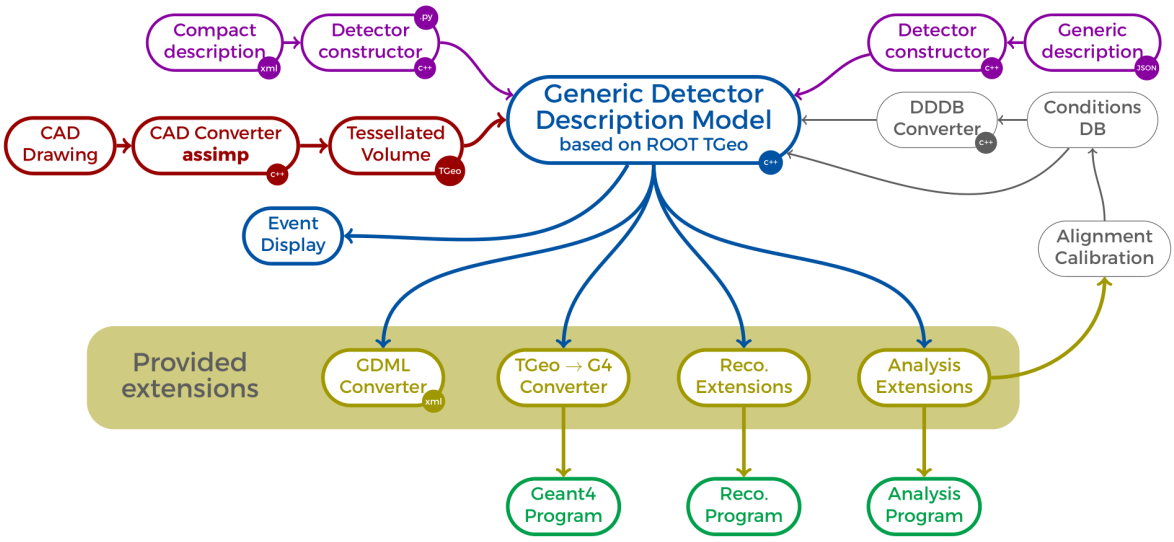

Figure 1. The components of the DD4hep detector geometry toolkit (in red new flow of the CAD interface).

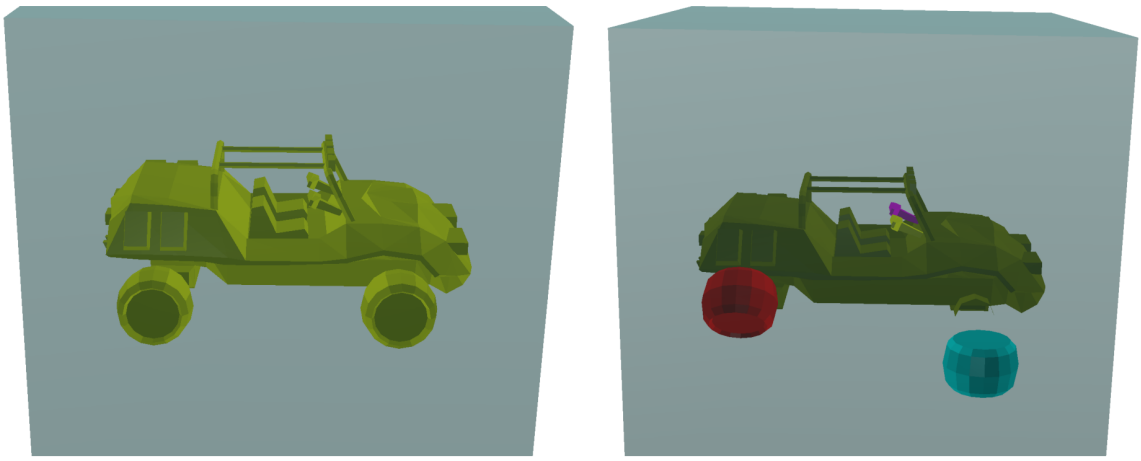

Figure 2. An example for a multi-mesh CAD input file converted to ROOT geometry. To the left the raw converted meshes with their natural placement are shown. To the right the same meshes are shown, but each mesh individually addressed by the plugin. Each mesh can be assigned its own material, visual attributes and corrected placement as supplied to the XML pattern.

4 Support for Tessellated Shapes and CAD Sources

The support to create real parts of the detector geometry directly from design files provided by engineering departments using Computer Aided Design (CAD) tools was demanded by physicists for a long time. Such an approach obviously offers large benefits, firstly because no work is involved translating design drawings to the geometry description and secondly to minimize potential problems or errors when producing the geometry description. To achieve this goal two basic ingredients were necessary:

- The possibility to model arbitrary tessellated shapes as composites of tri- or quadrilinear facets both in the ROOT geometry description [4] and in Geant4 [5, 9].
- A generic mechanism to convert arbitrary shapes described in CAD files to shapes understood by the ROOT geometry. The Open Asset Import library (`assimp`) [10] is a portable Open-Source library to import 3D-model formats to a common abstraction layer consisting of tessellated shapes built by tri- or quadrilinear facets. The transient, `assimp`-specific tessellated shapes which are created from CAD files when imported are then immediately translated to ROOT tessellated shapes. This immediate conversion minimizes overall memory usage. The Open Asset Import library supports a large variety of CAD formats which can be imported to the ROOT geometry system.

Currently reading CAD input is offered to the clients using the DD4hep plugin mechanism and is designed to be integrated in DD4hep detector constructor plugins. The mechanism is based on XML inputs where the plugin can be triggered with the application of an XML pattern, which supplies additional information. The plan is to collect knowledge of typical client use cases and correspondingly extend the available shape creation mechanism to allow convenient creation of tesselated shapes from detector constructors [6]. The DDG4 [11] extension of DD4hep used to simulate detector geometries using the Geant4 [5] toolkit was extended to detect and automatically translate tessellated shapes to Geant4.

CAD files may contain not only one, but multiple meshes of facet based shapes as it is shown in Figure 2. Each of these meshes corresponds to a separate shape. To properly construct volumes used to build geometries each of these meshes/shapes can be addressed individually and can have its own material, visualization properties and positioning.

Though the inclusion of shapes defined in CAD files into the ROOT geometry description using DD4hep is quite at an infant stage we strongly believe that the potential of this application is very high.

The interface to the `assimp` library is encapsulated in the DDCad package, if the library is found during the `CMake` configuration it will be compiled. Using the `CMake` flag `DD4HEP_LOAD_ASSIMP` it is also possible to compile the `assimp` library on demand.

5 Geant4 Units

For decades, the system of units used in experimental particle physics was based on the three basic units: centimeters, seconds and GigaElectronVolts. ROOT and its geometry description package TGeo [4] adopted this convention. For the LHC era the Geant4 collaboration decided that a basic unit system based on millimeters, nanoseconds and MegaElectronVolts was much more suited for the LHC experiments. Most experiments using Geant4 today effectively adopted this new convention for all areas of data processing: simulation, reconstruction and data analysis. Hence it would be advantageous for experiments using DD4hep (and in turn ROOT) to describe the geometry to use the same system of units both for the application code and the description of the experiment's geometry. The geometry itself is per definition agnostic to units, i.e. all length based quantities simply scale by the appropriate factor (factor 10 when converting `cm` based geometries to `mm` based geometries). However, derived material related quantities such as density, interaction length or radiation length are defined by non-linear relationships of the basic unit-system quantities. These derived quantities need to be properly redefined based on the basic unit-system quantities.

I WOULD HOPE ALL THE NUMBERS REQUIRE AN UNIT OF MEASUREMENT EXPLICITLY

To allow users to have the same system of units in the geometry description and the application code we supplied a patch to the ROOT geometry description, which allows one to switch the ROOT system of units at run-time. This switch can be enabled in DD4hep if required. To avoid confusion between materials described in ROOT units and materials described in Geant4 units, this switch may only be set once, before any element or material is constructed.

To enable Geant4 units in DD4hep, the toolkit must be instructed to use Geant4 units, the `CMake` option `DD4HEP_USE_GEANT4_UNITS` must be enabled. Client projects using the `CMake` configuration file provided by DD4hep automatically inherit the correct compilation settings.

6 Input/Output Plugin Developments

The Geant4 simulation using a DD4hep based geometry can be done using the `ddsim` program [7], which offers a wide range of command line or steering file based configuration options. To further ease the simulation workflow and better integrate with different Monte Carlo Generators or frameworks two new IO modules for DDG4 were implemented.

To support a wider range of MC generators, a new plugin was created, which uses the HepMC3 [12] file reader factory to read all formats supported by HepMC3. The `HepMC3::GenEvent` data is then converted into DDG4's internal representation and forwarded to Geant4.

To enable simulations for the Key4hep project [13], an output plugin for the EDM4hep event data model was created. The EDM4hep output plugin converts DD4heps internal information about Monte Carlo particles and hits into the respective output collections of EDM4hep and lets them be written to the output file.

Both plugins are optional and can be enabled with the `CMake` options `DD4HEP_USE_HEPMC3` or `DD4HEP_USE_EDM4HEP`. If the plugins are enabled one can

```
// obtain event information from HepMC::GenEvent
template <class T=HepMC3::GenEvent>
    void EventParameters::ingestParameters(T const& genEvent) { ... }

// obtain event information from LCIO event
template <class T=EVENT::LCParameters> void
   EventParameters::ingestParameters(T const& source) { ... }

// add the event information to the lcio::LCEventImpl
template <class T=lcio::LCEventImpl>
    void EventParameters::extractParameters(T& event) { ... }

// add the event information to the podio EventStore
template <class T=podio::EventStore>
   void EventParameters::extractParameters(T& event) { ... }
```

Listing 1: Template instantiation examples for passing event metadata between different inputs and outputs.

simply use them with the `ddsim` executable by using the input- and outputfile options with files with the respective extension. For example, `ddsim -inputFiles events.hepmc -outputFile simulated_edm4hep.root` Because DD4hep already contains a plugin to write out the information from simulated events in a ROOT file, the string `edm4hep` before the `.root` extension is mandatory to identify the correct output plugin.

To simplify the exchange of meta information between different input and output handlers a `dd4hep::EventParameters` class was developed to encapsulate this information. Each input or output module only has only to instantiate and implement a templated function with on the object containing the required information. Listing 1 shows some example signatures of the instantiated functions. The `dd4hep::EventParameters` are kept attached to DD4hep's internal event by its extension mechanism.

7 Conclusion

The DD4hep Detector Description Toolkit has been providing a fully functional toolset for detector description for many years. The user community of DD4hep is growing steadily, which has triggered a myriad of activities, including new features such as described in these proceedings and, last but not least, also bug fixes. One of the most important new features is the support for tessellated shapes and CAD sources, which adds an important mechanism of importing partial geometries to a comprehensive detector description. Though the inclusion of CAD sources into ROOT geometry description using DD4hep is at a pilot stage, we foresee that the potential for widespread application of these feature is very high. A further improvement in terms of unification of standards was the development of the ability to uniformly use the same system of units throughout DD4hep geometry description as well as in the application code bridging the ROOT vs. Geant4 units gap. It is also noteworthy to mention the extension of the input/output palette that DD4hep supports with the inclusion of broader HepMC support as well as the inclusion of EDM4hep. These developments are a true testimony of the continued strive of the DD4hep developers to improve and evolve DD4hep as a community tool.

Acknowledgements

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

References

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

[2] S. Ponce, P. Mato Vila, A. Valassi, I. Belyaev, eConf **C0303241**, THJT007 (2003), physics/0306089

[3] LCC, *The Linear Collider Collaboration*, https://www.linearcollider.org/

[4] R. Brun, A. Gheata, M. Gheata, Nucl. Instrum. Meth. **A502**, 676 (2003)

[5] S. Agostinelli et al. (GEANT4), Nucl. Instrum. Meth. **A506**, 250 (2003)

[6] M. Frank, F. Gaede, C. Grefe, P. Mato, J. Phys. Conf. Ser. **513**, 022010 (2014)

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

[8] Gaede, Frank, Frank, Markus, Petric, Marko, Sailer, Andre, EPJ Web Conf. **245**, 02004 (2020)

[9] J.A. et al., IEEE TNS, 53 **1**, 270 (2006)

[10] K.K. et al., *Open Asset Import Library (assimp)*, http://www.assimp.org

[11] M. Frank, F. Gaede, N. Nikiforou, M. Petric, A. Sailer, J. Phys. Conf. Ser. **664**, 072017 (2015)

[12] A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski, A. Verbytskyi, Comput. Phys. Commun. **260**, 107310 (2021), 1912.08005

[13] A. Sailer, G. Ganis, P. Mato, M. Petrič, G.A. Stewart, EPJ Web Conf. **245**, 10002 (2020)