EPJ Web of Conferences **251**, 03007 (2021) https://doi.org/10.1051/epjconf/202125103007
CHEP 2021

The GeoModel tool suite for detector description

Marilena Bandieramonte[1], *Riccardo Maria* Bianchi[1], *Joseph* Boudreau[1], *Andrea* Dell'Acqua[2], and *Vakhtang* Tsulaia[3]

[1]University of Pittsburgh, Pittsburgh, PA 15260, USA
[2]CERN, EP Department, Meyrin, 1211, Switzerland
[3]Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA

Abstract.
The GeoModel class library for detector description has recently been released as an open-source package and extended with a set of tools to allow much of the detector modeling to be carried out in a lightweight development environment, outside of large and complex software frameworks. These tools include the mechanisms for creating persistent representation of the geometry, an interactive 3D visualization tool, various command-line tools, a plugin system, and XML and JSON parsers. The overall goal of the tool suite is a fast geometry development cycle with quick visual feedback. The tool suite can be built on both Linux and Macintosh systems with minimal external dependencies. It includes useful command-line utilities: `gmclash` which runs clash detection, `gmgeantino` which generates geantino maps, and `fullSimLight` which runs GEANT4 simulation on geometry imported from GeoModel description. The GeoModel tool suite is presently in use in both the ATLAS and FASER experiments. In ATLAS it will be the basis of the *LHC Run 4* geometry description.

1 Introduction

A class library for the description of detector geometry, called the GeoModel toolkit, has been used to express the ATLAS [1] detector in software for simulation and reconstruction workflows. The ATLAS geometry was validated by visual debugging tools executing under the ATLAS Athena [2] software framework, then further validated by comparison with testbeam and collision data. The ATLAS experiment now prepares for Run 3 of the Large Hadron Collider (LHC), and simultaneously gears up for Run 4[1], which is the first run of the High Luminosity LHC (HL-LHC) [3]. This phase will involve substantial changes for the ATLAS detector, with the installation of new sub-detector systems like the Inner Tracker (ITk) [4] and the HGTD [5], as well as major upgrades to the muon spectrometer. In view of this future run, the authors are now extending the GeoModel toolkit to a full tool suite [6], which contains powerful lightweight visual and GEANT4[7]-based debugging tools for detector description, and which can be used outside of the highly complex ATLAS computing infrastructure. This tool suite vastly simplifies the detector description workflow, shrinks the development cycle, and is suitable for other HEP experiments.

[1]At the moment, the Run 3 data-taking period of the LHC is scheduled for the period 2022-2025, while Run 4 should begin in 2027.

2 The GeoModel Kernel

The GeoModel geometry kernel is a class library that has been developed within the ATLAS experiment in or around 2005 and used ever since, but recently it has been repackaged as an independent API, together with more recent elements of a tool suite that is described in the following sections.

The kernel allows for the description of a detector geometry with a small memory footprint. Once built, the description services no requests other than those related to geometry. The architecture is designed so as to make this description as compact as possible. It allows to build a directed, acyclic graph (DAG) of *nodes* which is interpreted as a DAG of volumes. The distinction is that certain graph nodes in the library which can appear within the graph represent integer and character string labelling policies that apply to subsequent nodes in the DAG; while other graph nodes represent physical volumes which are to be repeatedly placed according to a transformation function. These nodes are referred to as GeoIdentifierTags and GeoSerialIdentifiers; GeoNameTags and GeoSerialDenominators; and GeoSerialTransformers, respectively.

Nomenclature for the graph nodes follows that of GEANT4 but the concepts are not completely identical. GeoElement nodes representing elements are provided. GeoMaterials agglomerate elements, each with a specific fraction by weight. A large number of shapes, inheriting from GeoShape are available. Logical volumes (GeoLogVols) agglomerate shapes and materials, while physical volumes (GeoPhysVols) hold logical volumes and may contain children. The position of a physical volume is determined from transformation nodes placed upstream of the physical volume in the tree; during tree traversal the position of a physical volume is computed by composing the transformations encountered during tree traversal from the "world" volume to the specific physical volume. This memory-saving trick allows instances of physical volumes to be shared; the same instance can represent multiple volumes in different positions. Also, instances of logical volumes, shapes, and materials may be shared. Memory management is by means of reference counting.

Transformations (GeoTransform) are defined in an intuitive way, making use of classes from the Eigen [8] class library for linear algebra. In addition to fixed transformations, a second "alignable" transformation called GeoAlignableTransform is provided. The alignable transform has a default value and a "delta" transformation which may be updated during processing. This allows a piece of the detector, or an entire subtree, to be moved as alignment constants change, say, over the course of a run. For multi-threaded applications it is possible to set multiple deltas for the same GeoAlignableTransform in order to support concurrent processing of events which require different alignment corrections of the detector geometry (e.g. events from multiple run periods).

Certain physical volumes known as "Full" physical volumes contain a cache of their absolute position in space. In single-threaded processing the cache is empty until the first access is made. The absolute position is then calculated from the cascading sequence of reference frames and held in cache. Whenever an alignable transform is modified within a sub-tree, all of the full physical volume position caches in the sub-tree are reset. In multi-threaded processing a single full physical volume can cache multiple absolute positions, which correspond to different sets of alignment corrections. Full physical volumes are mostly to be used for active detector elements, but not for pieces of dead material whose absolute position in space is not requested. Relationships between these entities are show in Figure 1, and an example of a simple geometry tree built from them is shown in Figure 2.

The system allows a complete description of complex detector systems such as ATLAS, for which it has been applied over some sixteen years. Specific shapes not used within ATLAS may still be missing from the library. Recently an additional shape class called a

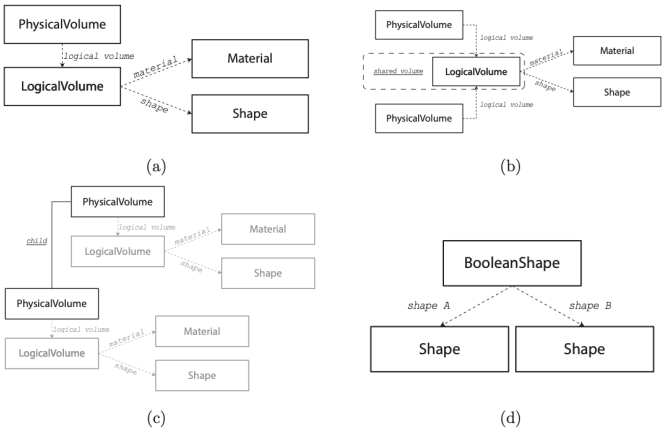


Figure 1: Diagrams showing an example of different GeoModel nodes: a) the implementation of a physical volume; b) a logical volume shared between two physical volumes; c) the parent-child relationship between two physical volumes; d) a boolean shape

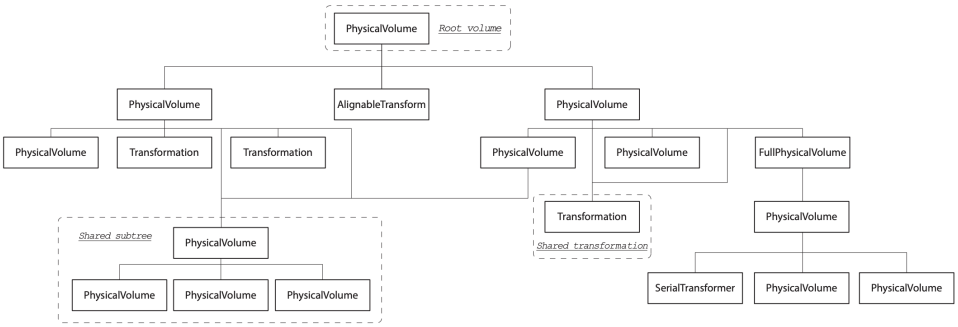


Figure 2: An example of a geometry tree, where different types of nodes are used. The upper node is the root volume, sometimes known as the *world volume*, which contains the whole tree. All connections in this graph represent a parent-child relationship. A shared sub-tree and a shared transformation are used as well

`GeoUnidentifiedShape` has been added. The payload of this shape consists of ASCII data that may be used to record the shapes which are still absent. In all the time during which `GeoModel` has been used in ATLAS, only a very few major changes have been introduced into the code of `GeoModel` Kernel (e.g., a switch from `CLHEP` [9] geometry classes to those provided by the `Eigen` class library). However since 2019 the geometry kernel has now been enhanced with a powerful tool suite for the development of detector description code; the components of this tool suite are described in subsequent sections.

2.1 XML Interpreter

A text-based front-end which allows users to code in their geometries in a simple and intuitive way, without being exposed to the C++ implementation of `GeoModel`, is now included. Based on `XML` (eXtensible Markup Language) [10], this layer can handle generic data structure, and also `GeoModel`-aware data structures, automatically interpreting them to produce a

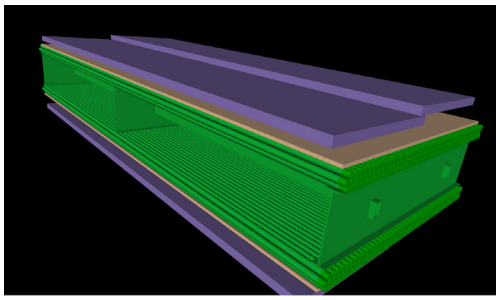


Figure 3: An example of ATLAS barrel muon station as developed in GeoModelXML.

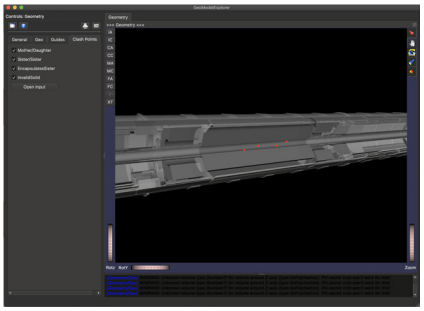


Figure 4: Geometry clash points (red dots) generated with `gmclash`, visualized in `gmex` (`GeoModelExplorer`).

nested geometry. It uses the Xerces-C [11] libraries for `XML` parsing and extends the functionality with new tags. A thin software layer is then used to access the XML contents, to create all needed `GeoModel` primitives and arrange them in the desired tree structure. This procedure, when supported by appropriate graphics tools, allows for very fast development of a geometry and much improved turn-around times. As an example, Figure 3 shows an actual ATLAS muon chamber, whose geometry is fully described with `GeoModelXML`; the chamber is then visualized with `GeoModelExplorer` (see Section 4). `GeoModelXML` is currently under further development aimed at generating efficient geometries with smaller memory footprint by taking advantage of more memory-saving features of the kernel.

3 Input and Output

The recently added `GeoModelIO` component extends the core library with the ability to save and restore a persistent copy of the full GeoModel geometry tree [12]. It provides high-level classes to dump the full geometry description to file and restore from it. `GeoModelWrite` traverses the full tree and serializes all GeoModel nodes, their attributes, and the parent-child relationships to be saved to a local SQLite [13] file. The `TFPersistification` package introduces tools to dump and restore functions and mathematical expressions; those are used, for instance, when placing volumes in a parametric way, by the use of the `GeoSerialTransformer` class. `GeoModelRead` de-serializes the information stored in the geometry file to restore all the nodes, the shapes, and the full GeoModel tree. The interface to the underlying SQLite database engine is furnished by the `GeoModelDBManager` package, which handles all the low level I/O operations.

The data model has been designed to optimize the disk footprint, to minimize the storage size of multiple versions of the geometry and to ease the sharing of geometry files through the network and their use in standalone applications. For example, GeoModel nodes which are shared between different sub-trees are saved only once, and then referenced by their ID wherever used. The optimized architecture of the `GeoModelKernel` classes and the carefully designed data model lets a large detector description as that of the ATLAS experiment, containing more than 500,000 nodes (of which about 80,000 are volumes), to be stored in about 40 megabytes.

`GeoModelRead` can build GeoModel nodes from a geometry file in parallel, relying on multi-threaded code. In that way, a geometry file containing a large number of nodes, such as that which contains a description of the ATLAS detector, can be read and the full geometry restored in about 7 seconds on a standard macOS laptop.

The ability to save and restore geometry opens the door to many opportunities which rely upon the exchange of geometry data, some of which are exploited in the GeoModel tool suite. In addition to taking input from SQLite files, those tools are also designed to take input from *plugins*. Plugins are shared libraries with .so or .dylib extensions containing object code that builds geometry. The object code is compiled from subclasses of the base class GeoVGeometryPlugin, from GeoModel kernel. They build and publish a geometry hierarchy and, where applicable, an index of GeoFullPhysVols and GeoAlignableTransforms.

Auxiliary data can also be stored in the output geometry file, on request. Auxiliary data consist of all data that are not strictly necessary to build the geometry tree itself, but they are related to the geometry and the user might want to store for them to be used at a later stage in the overall detector description workflow. For example, indexes of specific geometry nodes, or data to subsequently cluster nodes in specific ways. Auxiliary data can be either directly defined in the C++ code, or fetched from XML files.

Command-line tools in the tool suite are designed to accept either an SQLite file or a plugin as an argument. Several examples will be encountered below.

4 Geometry Visualization

Geometry can be visualized in an application program called the GeometryExplorer, hereafter referred to as gmex. This program was not developed from scratch, but from one of the principal visualization programs, VP1 [14] which has been in use in ATLAS since 2007. gmex is essentially VP1 running standalone rather than as a module within the ATLAS software framework, and from which all display of event data (i.e. hits, tracks, and the like) is removed, so that the program functions as a pure geometry interactive browser. These steps are taken in order to sever dependencies on the complex ATLAS environment, to insure portability, and to facilitate packaging and distributions.

gmex reads input from geometry files or from plugins. In the graphical user interface, a check box is created for each top-level physical volume allowing to toggle its visibility. Users can interactively set the viewpoint and viewing angle. gmex supports multiple viewing windows, cutaway views, and interactive control over the expansion, contraction, and iconization of detector geometry–features which are inherited from its predecessor VP1. Additional features specifically for debugging geometry have been added. Individual elements may be selected, and saved in an output file for further examination. Clash reports (see Section 6) can be read in and co-displayed with geometry. Local coordinate systems may be visualized. Plugins can be reloaded without restarting gmex, so that the effect of a modification to input data can be visualized immediately.

gmex relies upon the Coin graphics libraries [15], the Qt5 class library for graphical user interfaces [16], and the SoQt "glue" layer [17] which allows for the display of Coin graphics within Qt windows. Qt5 is a cross-platform library, which uses native windowing libraries on both Mac and Linux platforms. Coin uses OpenGL for rendering.

5 Utilities

The GeoModel tool suite provides a number of tools to handle files with different formats of the geometry description.

The gmcat is a command line tool whose arguments may include one or several SQLite inputs, one or several plugins, or a mixture of input files and plugins. It builds a geometry from all of these sources, concatenates the geometry, and writes it to an output file.

The gdml2gm utility converts GDML input into a GeoModel SQLite file. GDML is a de facto standard for the description of detector geometry and is widely used in the HEP community.

By converting GDML to GeoModel one can use the functionality of the gmex visualization tool, as well as other utilities of the GeoModel tool suite, on the detector systems originally not described in GeoModel. At the time of writing this paper gdml2gm utility is not yet able to interpret the full set of GDML entities. We plan to add the missing conversion routines over the next months.

The gm2gdml command line tool converts GeoModel geometry descriptions into the GDML format. The input geometry description can be constructed either by reading an SQLite file, or by loading a GeoModel C++ plugin (.so/.dylib).

6 FullSimLight and related utilities

The FullSimLight tool-suite depends on the GEANT4 simulation toolkit, whose libraries are used to perform some of the tasks, and it includes different components, starting with a standalone Geant4-based light simulation (fullSimLight), a geometry clash detection tool (gmclash), a tool to generate geantino maps from an input geometry (gmgeantino) and the aforementioned tool to convert geometries and dump them in GDML format (gm2gdml). Another tool that calculates inclusive and exclusive mass of an input geometry (gmmasscalculator), is under development. This tool-suite is experiment agnostic and is compatible with different inputs, in particular GeoModel SQLite files (.db) and C++ plugins (.dylib/.so), and GDML (.gdml). Furthermore, the FullSimLight tool-suite is designed to work independently of any specific experiment infrastructures, in a purely standalone way and it can be easily installed on personal computers, even laptops (Linux and macOS systems are supported).

6.1 FullSimLight

FullSimLight [18] is a standalone lightweight GEANT4 simulation that allows to run particle transport Monte Carlo simulation on different detectors that can be described in different geometry formats and that can be plugged-in simply using a command line option. There are two ways of generating primary particles in FullSimLight. The first is to use the standard GEANT4 particle gun (that can shoot single primary particles between e^-/e^+ or γ or electromagnetic showers with particles randomly chosen). The second one is to use the Pythia event generator [19] choosing between three different standard configurations available (i.e. *ttbar*, *higgs* or *minbias*) or using directly a Pythia input configuration file to determine the kind of events that the user wants to generate and simulate. For what concerns the transport in magnetic field, the user has two options: either to set a constant magnetic field or plug a magnetic field map. At the moment this mechanism is customized to support a specific ATLAS magnetic field map, but it can be extended in order to support any magnetic field map that is compliant with a given format. FullSimLight provides a basic scoring mechanism with a minimal set of observables that is collected during the simulation per-primary particle type: mean energy deposit, mean charged and neutral step lengths, mean number of steps made by charged and neutral particles, mean number of secondary e^+, e^+ and γ particles. The scoring function is being extended at the time of writing this paper, with the creation of different histograms to track stepping quantities, such as step kinetic energy, step pseudorapidity, step length and other quantities. The histograms are merged automatically by each worker thread at the end of the run and saved in a ROOT [20] file using g4root mechanism from GEANT4, which allows for writing out ROOT files without introducing a dependency on ROOT.

6.2 GeoModelClash

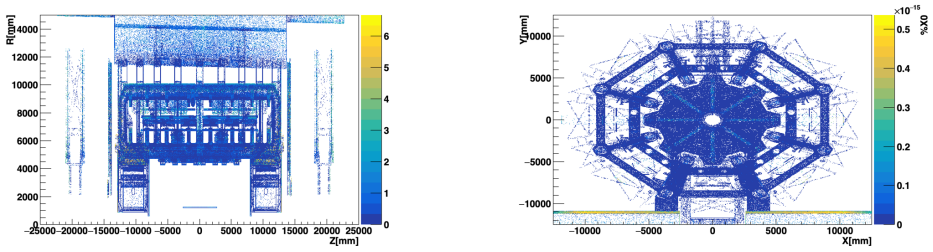
The `GeoModelClash` application, hereafter `gmclash`, is a command line tool that allows for detecting clashes in an input geometry description. A transient detector geometry description is usually organized into a hierarchical structure, which consists of mother-child relationships between volumes. The clash detection checks that the volumes on the same hierarchy level do not overlap, and also that all children volumes of a specific mother volume are fully contained in it. Volume overlaps must be avoided because they can produce an inconsistent geometry description (e.g., wrong material description in the overlapping regions), and they can also lead to unpredictable behaviour of the particle transport in `Geant4` (e.g., stuck tracks, wrong energy depositions).

The `gmclash` tool first converts the input `GeoModel` geometry into the `Geant4` geometry format, and then runs the `Geant4` geometry checker on it. The report produced by the checker is saved into a `json` file that contains a detailed information for each clash, i.e., the minimum overlapping distance, the type of the clash (mother-child, sibling-sibling, fully encapsulated sibling, invalid solid definition), some information about the two intersecting volumes (name, entity type, copy number), and the global coordinates of the point of overlap.

The report file generated by `gmclash` can be plugged into `gmex` and visualized together with the detector geometry. The points of clashes will be displayed in different colours according to the 4 different categories of clashes, as it can be seen on Figure 4.

6.3 GeoModelGeantino

The `GeoModelGeantino` tool (`gmgeantino`) can be used to generate geantino scans of a specific input geometry. A *geantino* is a virtual `Geant4` particle that does not interact with matter, and it is mainly used for testing tracking capabilities and the transport in the magnetic field. Geantino maps are generated shooting neutral geantino particles into the detector, by default with isotropic distribution, and collecting and integrating the radiation lengths traversed inside the detector per each step or event in the simulation. For their characteristics, geantino maps represent an useful instrument to check the validity of a geometry description. By default the total radiation length and the integrated radiation length for η and R-Z and X-Y are collected and saved in 1D and 2D profile histograms that are then stored in a `ROOT` file using `g4root` mechanism.



(a) (b)

Figure 5: Total R-Z radiation length profile (a) and X-Y radiation length profile for the element Aluminium (b) in the ATLAS Muon Spectrometer geometry.

The tool provides different flags and options that can be used to restrict the area of interest in the detector and to customize the creation of η, R-Z and X-Y profile histograms per each

sub-detector component, material or element in the geometry description. As an example, in Fig. 5 are shown two radiation length profiles that were generated running `gmgeantino` on the ATLAS Muon Spectrometer geometry. In (a) is the total R-Z radiation length profile, while (b) shows the X-Y radiation length profile of a specific element: aluminium.

It is possible to run `gmgeantino` on different parts of the given detector, and then combine the results to obtain plots like the one on Figure 6. This plot represents the material distribution expressed in units of radiation length X_0 as a function of η in the ATLAS Inner Detector.

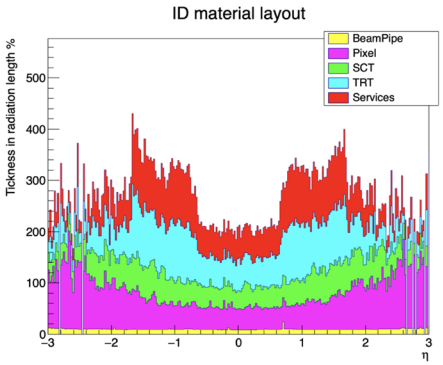


Figure 6: Material distribution expressed in units of X_0 as a function of η for the ATLAS Inner Detector.

This figure was generated running `gmgeantino` separately on every sub-detector component of the ATLAS Inner Detector and results were combined in a `ROOT` stacked histogram. It reproduces a similar figure that can be found in [21], that was produced to study the material budget of the ATLAS Inner Detector. The breakdown indicates the contributions of external services and of individual sub-detectors, including services in their active volume.

7 Code, License, Build system, Distribution kits, Documentation

The GeoModel tool-suite is open source, distributed under a permissive free software Apache 2.0 license [22], and currently hosted on the CERN GitLab servers [23].

The tool-suite makes use of CMake [24] as build system generator, to let users build the code with their preferred build system, and exports CMake targets, to let users easily use the GeoModel libraries in CMake-based client code. The tool suite can be easily compiled on macOS and Linux, with minimal third-party dependencies. Distribution kits are also built for the most common platforms among the current GeoModel users, namely macOS and Ubuntu; on those platforms, the full GeoModel tool-suite can be installed through the standard package managers (`brew` and `apt`) or with a simple shell command. GeoModel can rely on a dedicated Continuous Integration (CI) infrastructure, which continuously tests changes to the code-base on different platforms. A dedicated website [6] hosts the GeoModel tool-suite documentation and the reference guide.

8 Use of `GeoModel` tools by CERN experiments

As of this writing, the new `GeoModel` toolkit is being adopted by detector groups of the ATLAS experiment (Figure 7). The thrust of the current effort is to develop an infras-

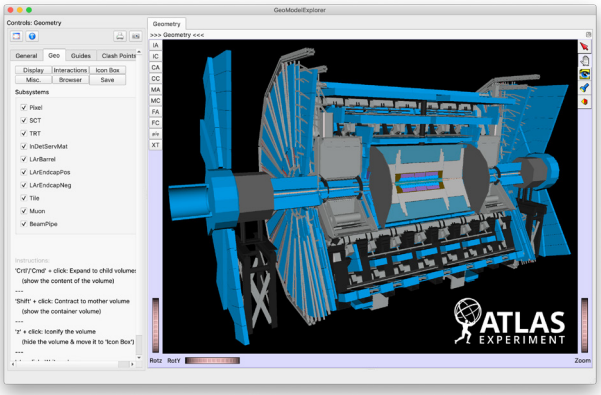


Figure 7: The ATLAS detector visualized in `GeoModelExplorer`.

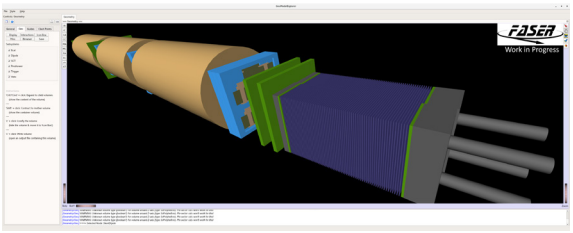


Figure 8: The LHCb and ATLAS components used in the FASER experiment, visualized in `GeoModelExplorer`.

tructure, in conjunction with subsystems experts, that can produce an `SQLite` file expressing the full geometry of ATLAS, in addition to an index of `GeoFullPhysVol`s and `GeoAlignableTransform`s, the former being required for connection to readout geometry and the latter being required for the ATLAS alignment system. This "master" file is to capture and preserve information about the detector geometry and its composition for posterity, and also shall be introduced into `Athena` based simulation and reconstruction workflows. Validation of the full system is to take place well in advance of LHC Run 4 data taking.

The GeoModel tools are also used by the FASER experiment [25] at CERN to describe, visualize, and debug the geometry of their detector. Figure 8 shows both the LHCb [26] and ATLAS components used in the FASER experiment, each constructed from their native representations (plain `GeoModel` for ATLAS and `GDML` for LHCb).

9 Summary

The ATLAS experiment has been using the `GeoModel` tool suite as a main detector description engine since 2004. In 2019 `GeoModel` was repackaged into an independent and experiment-agnostic API. Since then its core functionality has been enhanced by introducing the mechanisms for creating persistent representations of transient geometry descriptions, automatic building of geometry descriptions from XML files, visual inspecting and debugging of detector geometries, running lightweight simulation tasks with the geometry converted from `GeoModel` to `GEANT4`, as well as other useful features. The new `GeoModel` toolkit is

currently being adopted by the ATLAS and FASER experiments. In ATLAS it will become the basis of the LHC Run 4 geometry description

References

[1] ATLAS Collaboration, Journal of Instrumentation **3**, S08003 (2008)
[2] P. Calafiura, W. Lavrijsen, C. Leggett, M. Marino, D. Quarrie, *The Athena control framework in production, new developments and lessons learned*, in *Computing in high energy physics and nuclear physics. CHEP'04, Interlaken, Switzerland,* (2005), pp. 456–458
[3] G.Apollinari, I.B. Alonso, O. Brüning, P. Fessia, M. Lamont, L. Rossi, L. Tavian, *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1*, CERN Yellow Reports: Monographs (CERN, Geneva, 2017), https://cds.cern.ch/record/2284929
[4] A. Macchiolo, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **962**, 162261 (2020)
[5] S. Mazza, Journal of Instrumentation **14**, C10028 (2019)
[6] *GeoModel Tool-suite Documentation Website*, http://cern.ch/geomodel
[7] S. Agostinelli et al. (GEANT4), Nuclear Instruments and Methods in Physics Research Section A **A506**, 250 (2003)
[8] *Eigen, a C++ template library for linear algebra*, http://eigen.tuxfamily.org
[9] L. Lönnblad, Computer Physics Communications **84**, 307 (1994)
[10] *XML, Extensible Markup Language*, https://www.w3.org/XML/
[11] *Xerces-C, a C++ XML parser*, https://xerces.apache.org/xerces-c/
[12] R.M. Bianchi, J. Boudreau, I. Vukotic, Journal of Physics: Conference Series **898**, 072015 (2017)
[13] *SQLite, a C-based SQL database engine* (2009), https://www.sqlite.org
[14] T. Kittelmann, V. Tsulaia, J. Boudreau, E. Moyse, Journal of Physics: Conference Series **219**, 032012 (2010)
[15] *Coin, an OpenGL-based, 3D graphics library*, https://github.com/coin3d/coin
[16] *Qt, the cross-platform software development framework*, https://doc.qt.io/
[17] *SoQt, a Qt GUI toolkit library for Coin*, https://github.com/coin3d/soqt
[18] M. Bandieramonte, R.M. Bianchi, J. Boudreau, *FullSimLight: ATLAS standalone Geant4 simulation*, in *24th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2019)* (2020), Vol. 245 of *EPJ Web Conf.*, https://doi.org/10.1051/epjconf/202024502029
[19] T. Sjöstrand, Computer Physics Communications **246**, 106910 (2020)
[20] R. Brun, F. Rademakers, P. Canal, A. Naumann, O. Couet, L. Moneta, V. Vassilev, S. Linev, D. Piparo, G. GANIS et al., *root-project/root: v6.18/02* (2019), https://doi.org/10.5281/zenodo.3895860
[21] V. Kartvelishvili, E. Bouhova-Thacker (ATLAS), PoS **ACAT**, 046 (2007)
[22] *Apache 2.0 License*, https://www.apache.org/licenses/LICENSE-2.0
[23] *GeoModel — A user-friendly C++ Toolkit for HEP Detector Description*, https://gitlab.cern.ch/GeoModelDev/GeoModel
[24] *CMake, a cross-platform build system generator*, https://cmake.org/
[25] FASER Collaboration, Tech. rep., CERN (2018), http://cds.cern.ch/record/2651328
[26] LHCb Collaboration, Journal of Instrumentation **3** (2008)