A new experiment-independent mechanism to persistify and serve the detector geometry of ATLAS

Riccardo Maria Bianchi¹, Joseph Boudreau¹ and Ilija Vukotic²

¹ University of Pittsburgh, Department of Physics and Astronomy, 100 Allen Hall, 3941 O’Hara St, Pittsburgh PA 15260, US
² University of Chicago, High Energy Physics, 5640 S Ellis Ave, Chicago IL 60637, US
E-mail: riccardo.maria.bianchi@cern.ch, boudreau@pitt.edu

Abstract. The complex geometry of the whole detector of the ATLAS experiment at LHC is currently stored only in custom online databases, from which it is built on-the-fly on request. Accessing the online geometry guarantees accessing the latest version of the detector description, but requires the setup of the full ATLAS software framework “Athena”, which provides the online services and the tools to retrieve the data from the database. This operation is cumbersome and slows down the applications that need to access the geometry. Moreover, all applications that need to access the detector geometry need to be built and run on the same platform as the ATLAS framework, preventing the usage of the actual detector geometry in stand-alone applications.

Here we propose a new mechanism to persistify¹ and serve the geometry of HEP experiments. The new mechanism is composed by a new file format and the modules to make use of it. The new file format allows to store the whole detector description locally in a file, and it is especially optimized to describe large complex detectors with the minimum file size, making use of shared instances and storing compressed representations of geometry transformations. Then, the detector description can be read back in, to fully restore the in-memory geometry tree.

Moreover, a dedicated REST API is being designed and developed to serve the geometry in standard exchange formats like JSON, to let users and applications download specific partial geometry information.

With this new geometry persistification a new generation of applications could be developed, which can use the actual detector geometry while being platform-independent and experiment-independent.

1. Anatomy of a GeoModel geometry graph
The complex geometry of the whole detector of the ATLAS experiment [1] at LHC is currently stored only in custom online databases, from which it is built on-the-fly on request. The master copy of all ATLAS detector geometry in the ATLAS software is the GeoModel [2], a C++ package used to build a graph of geometrical primitives resembling a scene graph².

¹ In software development in general, and in HEP computing in particular, persistifying means taking an object which lives in memory only (for example because it was built on-the-fly while processing the experimental data), serializing it and storing it on disk as a persistent object.
² In computer graphics, there are different approaches used to organize the 3D objects in the world: one of them is the scene graph: a structured tree of nodes representing graphical primitives and appearance properties, organized so to give the final image. The GeoModel has a similar structure, with nodes representing geometrical primitives.
The description of a detector geometry consists of a tree of GeoModel nodes pretending to be a tree of volumes. The geometry graph consists of both placed volumes and parameterized volumes, which are used in a combined way to build complex shapes, as shown in Figure 1.

Figure 2a shows the diagram of the implementation of a single PhysicalVolume node (a GeoPhysVol object), representing a geometrical volume. A PhysicalVolume does not have any “geometrical properties”: it only references (or has) a LogicalVolume, as seen in the picture; the constructor signature of its C++ class is GeoPhysVol(const GeoLogVol* LogVol). The LogicalVolume is what defines a real geometrical volume, with a particular shape and a given material; the LogicalVolume, in fact, references two other nodes, a Material and a Shape, as shown in the same figure; it also has a name property to label the volume, specified in its constructor signature GeoLogVol(const std::string &Name, const GeoShape *Shape, const GeoMaterial *Material). Such reference links (displayed as dotted lines in the diagram) are part of the tree description and need to be stored when persistifying the geometry tree.

Two PhysicalVolumes can reference the same LogicalVolume, as shown in Figure 2b; in that case the LogicalVolume is said to be a shared node. Shared nodes not only simplify the geometry graph, but they also reduce the memory footprint of the graph, because only one instance of the shared object is built, acting as a memory optimization mechanism. In order to reflect this into the persistification mechanism and to get a final file with the smallest size possible, shared nodes need to be persistified once when the first node using them is met, and then reused when they are needed by other nodes.

A PhysicalVolume can have children: it acts as a container, grouping different nodes in a same sub-tree. Figure 2c shows a PhysicalVolume, which has a parent-child link (shown in the diagram as a solid line) to another PhysicalVolume, which acts as a child. A PhysicalVolume can have any number of children, and the order of insertion of the children is important: many property nodes act only on nodes which follow them. For example, the combination of a PhysicalVolume and a LogicalVolume creates a concrete geometrical object; that object is placed into the space with the usage of a Transformation node, which is inserted as child node before the object to place. Thus, not only the parent-child relationship needs to be persistified, but the order of the children nodes as well.

A reference counting scheme is implemented within GeoModel as well, to automatically delete nodes from memory when they are not used anymore.
GeoModel only provides basic geometry primitives; but more complex objects can be built with the usage of boolean shapes. Those are special nodes which combine two shapes through boolean operations (see Figure 2d), letting the users cut holes in a shape or to add two shapes to make a new one, and so forth (as the shapes shown in Figure 1b). Boolean shapes need to be persistified as well.

The parametrized volumes are created through the usage of the SerialTransformer node. That node references a PhysicalVolume and a mathematical function: new volumes based on the referenced PhysicalVolume are created in memory according to the referenced function. Thus, it is possible to create a complex pattern of detector volumes simply specifying the base module and the parametric function which places them in the space, as the shapes shown in Figure 1c. Persistifying SerialTransformers requires the persistifications of the mathematical functions, which has been implemented and will be discussed in Section 3.4.

Figure 5 shows a more complex example of a geometry tree, where many different nodes, shared nodes and shared sub-trees are used.

2. The GeoModel description of the ATLAS detector: In-memory model vs database

The GeoModel description of the ATLAS detector is the master copy of all geometry. Detector factories running within an on-line geometry service read a database of primary numbers and build a highly detailed description of ATLAS, which is then translated directly to Geant4 [3] for the full simulation of the detector response.

The procedures to build the GeoModel are complex, depend on a large stack of ATLAS software, and lack portability. It is difficult to port the geometry builders into lightweight
applications designed to run without the full ATLAS software stack. And so far it was not possible to import geometry code in ATLAS-neutral or, better, in experiment-independent applications. Moreover, the experiment software is only deployed for very specific platforms.

In this work we designed and developed the mechanism to dump the whole description into a database file and provide a portable mechanism to read it back in, to be used for a generic experiment and within stand-alone applications.

It is worth to notice that the CPU and memory requirements of the GeoModel are modest. Table 1 shows the memory footprint of the different parts of the ATLAS detector, together with the time to build them. The development of the new geometry persistification mechanism presented here is all about portability.

| System | Memory (Mb) | CPU (s) |
|--------|-------------|---------|
| Pixel  | 12.8        | 0.35    |
| SCT    | 13.3        | 0.14    |
| TRT    | 22.2        | 0.18    |
| LAr    | 32.3        | 1.45    |
| Tile   | 14.7        | 0.36    |
| Muon   | 36.1        | 0.79    |

Table 1: Memory footprint and loading time of the different parts of the ATLAS geometry described with the usage of GeoModel

3. Peristify the whole geometry in a database

The purpose of this work has been the design and the development of a new mechanism to persistify the GeoModel description and store it into a standalone file.

3.1. A C++ API to read and write Geometry, independent of ATLAS software

At first, the GeoModel module has been extracted from the ATLAS framework, and converted to a stand-alone package. GeoModel is now an experiment-independent geometry package that depends only on two external libraries: CLHEP [6] for the geometrical and mathematical objects and functions and the Qt [7] library to handle the connection to the database in a convenient way, through the only usage of the QtSql module and of the class QSqlDatabase. That decoupling from ATLAS allows us to use the GeoModel description in cross-platforms or web applications, opening up the possibilities of using the detector description in different ways and for different experiments and purposes.

3.2. packages

Figure 4 shows all the new packages developed within this work and all the interaction between them. The new GeoWrite package contains a class inheriting from the GeoModel::GeoNodeAction, which visits all the nodes of the in-memory geometry tree and dump their properties to the database. The package TFPersistification provides tools to dump and restore parametric mathematical functions used with a SerialTransformer. More details will be given in Section 3.4. All the connections to the new database and all operations on it, are handled by another new package, GeoModelDBManager, which abstracts low level SQL database calls providing higher level methods to store and retrieve GeoModel objects to and from the database. The package GeoRead features classes and methods to restore GeoModel objects from the database file and to build the geometry tree.

---

4 Scientific Linux CERN 6 (SLC6) only, at the time of writing.
3.3. the database

We wanted to store the GeoModel objects and their mutual connections in the smallest possible file, to be able to easily distribute it through the network.

With that in mind, we have designed an efficient database schema where elements of the geometry tree are stored in the appropriate tables, where only the arguments of the constructor of the corresponding C++ class of each object are stored, which are the most basic piece of information needed to fully characterize a GeoModel object. The objects are then restored taking the parameters and invoking the constructor of the corresponding GeoModel class with them as arguments. Moreover, shared instances appear only once in the database tables, to keep the database size small.

Concerning the connection between objects, as explained before in Section 1, there are two type of links to be stored. The first type is the link between a node and another node used in its definition, or a referenced node. For example, let us consider a logical volume: this references a Material and a Shape within its definition; when a GeoModel tree is built programatically, the pointer to a GeoMaterial and to a GeoShape are passed as argument to the GeoLogVol constructor.

1  GeoMaterial* mat = new GeoMaterial("mat", 0.1);
2  GeoBox* shape = new GeoBox(15, 15, 15);
3  GeoLogVol logVol = GeoLogVol("name", shape, mat);

To persistify this type of link, we keep track of the unique IDs of all nodes stored in the database. Then, the ID of the referenced node is persistified as a numerical parameter within the arguments list of the nodes that reference it, and in the corresponding database record. Examples of such nodes are physical volumes, logical volumes, serial transformers and so forth.

Another type of link to be stored is the parent-child relationship between two nodes (see Section 1). To store this type of information, a link table has been added to the database: the ChildrenPositions table shown in Figure 3. The first parameter is the unique ID of the relationship; the second and the third one identify the parent node (unique ID of the node and the ID of the table which stores the record). The fourth parameter stores the position of the child within the children list of the parent node; as we have said, order is important because the effect of certain nodes on the other nodes depends on their position within the tree. The fifth and sixth parameters identify the child node (table and unique ID).

This link table is then used by the client application to restore the complete geometry tree: new parents and children are created as soon as they are met, using the parameters stored in the relevant tables, together with all referenced nodes and all parent-child connections.

SQLite [5] is the chosen database technology. The resulting database schema we have developed let us store the full ATLAS GeoModel tree, which has a memory footprint of \(\approx 140\) Mb, in a very small file of about 36 Mb, small enough to be distributed through the network and to be used in stand-alone applications.

3.4. Persistification of parametric mathematical functions

As said, mathematical functions are used by the GeoModel SerialTransformer to programatically create and place copies of a shared master physical volumes on-the-fly.

In CLHEP mathematical expressions are binary expression tree. The interface is natural. When instanciating a SerialTransformer object, the pointer to the function is passed to its constructor.

To store and restore the mathematical functions we have developed a new mechanism to persistify generic mathematical functions. The class `TransfunctionPersistifier` stores the transfunction expression tree as a nearly human readable string. In the end, `Function` objects are dumped by the Recorder modules into text strings and stored in the database as text. Then,
Figure 3: All the tables used in the new database that stores all information of the GeoModel tree.
the TransfunctionReader class interprets the text strings and build the mathematical function in memory, ready to be used with the GeoSerialTransformer.

4. Visualization
The new persistification mechanism is already used by the new prototype of the stand-alone version of the ATLAS 3D event display VP1 [4]: VP1-Light, which is the first client of the new geometry persistification mechanism. Figure 4a shows it in action with a very simple fake geometry. VP1-Light is also be used to debug the geometry persistification.

![Figure 4a](image)

(a)

![Figure 4b](image)

(b)

Figure 4: a) the VP1-Light window showing the rendering of a sample detector geometry loaded from its SQLite file; b) the packages developed within this work

5. Serving the geometry through a REST API
Other stand-alone desktop or web-based applications could make use of the restored detector geometry. An example is the new ATLASRift Virtual Reality application [8] which could take great advantage from the serving of the detector geometry through a web-based API.

A system to deliver the geomodel data has to fulfill the following requirements: support large number of users, quickly and efficiently deliver potentially large amount of data, provide simple but powerful query syntax via a REST interface, and deliver data in JSON format. We find all these requirements well fulfilled by Elasticsearch [9], a highly scalable open-source search and analytics engine based on Apache Lucene. As most other NoSQL databases, Elasticsearch accepts and stores data as JSON documents and does not provide relations, or join functionality, so all the data have to be fully denormalized. There are two main things to develop: a code that constructs appropriately structured JSON documents and a mapping document that the Elasticsearch will use to map the variables in incoming documents to its own variable types. By fully indexing all the document content, it is possible to search, filter, aggregate against any of the properties. As all the aggregations are done on the Elasticsearch cluster itself even complex queries (eg. find a total volume of all the objects made of Aluminum in a slice of detector given by min and max Theta and Phi) would take order of tens of milliseconds to compute. While the denormalization significantly increases size of documents delivered, Elasticsearch is capable of delivering compressed responses, so loading even full detector geometry should not take more than few seconds. The two features that are important for any application that does not have enough memory and computing power to simply preload full geometry in advance are: capability to deliver different Level Of Detail (LOD) geometries, and delivery of fully tessellated meshes. As the service is still under development, detailed description with performance measurements and application examples will be described in a dedicated paper in the near future.
6. Summary
The GeoModel geometry system has been in service in ATLAS for more than a decade and is a fully mature system. It is now a stand-alone, experiment-independent and cross-platform package which can be used for the detector description of different experiments. Also, the possibility of saving and restoring the geometry through the newly developed mechanism opens up new possibilities for stand-alone applications, like lightweight Visualization APIs. A new REST API will allow web-based applications to use chunks detector geometries on-demand, through the network.

Figure 5: An example of a geometry tree, where different types of nodes are used. The upper node is the root volume, sometimes known as world volume, which contains the whole tree. All connections in the graph represent a parent-child relationship. A shared sub-tree and a shared transformation are used as well.

References
[1] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, JINST 3 (2008) S08003.
[2] J Boudreau and V Tsulaia 2004 The GeoModel Toolkit for Detector Description, CHEP ‘04, Book of Abstracts, https://indico.cern.ch/event/0/contributions/1294152/
[3] S Agostinelli et al. 2003 Geant4, a simulation toolkit, Nuclear Instruments and Methods in Physics Research A 506 250–303, http://dx.doi.org/10.1016/S0168-9002(03)01368-8
[4] T Kittelmann et al. 2010 The Virtual Point 1 event display for the ATLAS experiment, J. Phys.: Conf. Ser. 219 032012; http://dx.doi.org/10.1088/1742-6596/219/3/032012
[5] D R Hipp et al., SQLite [computer software], https://www.sqlite.org
[6] CLHEP - A Class Library for High Energy Physics, http://proj-clhep.web.cern.ch/proj-clhep/
[7] The Qt Company, Qt 5 [Computer software], https://www.qt.io/
[8] ATLASRift - First person science, https://atlasrift.web.cern.ch/
[9] Kibana, ElasticSearch [Computer software], https://www.elastic.co/products/kibana