Planning for Evolution in a Production Environment: Migration from a Legacy Geometry Code to an Abstract Geometry Modeling Language in STAR

Jason C. Webb
Brookhaven National Laboratory, Upton, NY 11973
E-mail: jwebb@bnl.gov

Jerome Lauret
Brookhaven National Laboratory, Upton, NY 11973
E-mail: jlauret@bnl.gov

Victor Perevoztchikov
Brookhaven National Laboratory, Upton, NY 11973
E-mail: vperev@bnl.gov

Abstract. Increasingly detailed descriptions of complex detector geometries are required for the simulation and analysis of today’s high-energy and nuclear physics experiments. As new tools for the representation of geometry models become available during the course of an experiment, a fundamental challenge arises: how best to migrate from legacy geometry codes developed over many runs to the new technologies, such as the ROOT/TGeo [1] framework, without losing touch with years of development, tuning and validation. One approach, which has been discussed within the community for a number of years, is to represent the geometry model in a higher-level language independent of the concrete implementation of the geometry. The STAR experiment has used this approach to successfully migrate its legacy GEANT 3-era geometry to an Abstract geometry Modelling Language (AgML), which allows us to create both native GEANT 3 and ROOT/TGeo implementations. The language is supported by parsers and a C++ class library which enables the automated conversion of the original source code to AgML, supports export back to the original AgSTAR[5] representation, and creates the concrete ROOT/TGeo geometry implementation used by our track reconstruction software. In this paper we present our approach, design and experience and will demonstrate physical consistency between the original AgSTAR and new AgML geometry representations.

1. Introduction
The STAR experiment has evolved significantly over more than a decade of operation at the Relativistic Heavy Ion Collider (RHIC). The installation of baseline and upgrade detectors was staged over several runs, extending the experimental reach of STAR to take advantage of improvements in collider performance. As a consequence, we maintain several versions of the STAR geometry. Over thirty production-quality geometry releases are required to support...
the reconstruction and simulation of data samples accumulated across twelve RHIC runs. The development, tuning and validation of the STAR geometry model represents significant investments of time and effort on the part of detector experts, students and postdocs as they struggled to reconcile simulations with ever more precise and diverse samples of data in the pursuit of important physics measurements.

From the beginning, the STAR experiment has used the Advanced Geant Interface (AgSTAR) [5] as the framework for representing its geometry model. The AgSTAR framework provides a level of abstraction from the underlying GEANT3 [6] implementation. This is achieved by using the MORtran preprocessor to extend the FORtran programming language with geometry-specific syntax, creating a language which serves as an abstract interface to the GEANT3 geometry library. This framework has served STAR well over the years, providing a robust and flexible development environment for describing the STAR detector. However, the framework’s reliance on MORtran and its tight coupling to the GEANT3 libraries does not permit STAR to leverage more modern technologies such as Geant 4 [7] and Virtual Monte Carlo[8] in the simulation and reconstruction of our data. To support the physics program which STAR anticipates in the coming years, it was clear that STAR needed to migrate away from our legacy geometry model.

2. Design Goals
In migrating the STAR geometry model to a new representation, the constraints of a production environment had to be kept in mind. Support for ongoing experimental operations and data production required maintaining existing infrastructure. This included continued support for our AgSTAR simulation package, as well as the track reconstruction software which utilizes the ROOT/TGeo [1] geometry classes. Furthermore, existing data samples must be reproducible, and changes to physics results must be easily traced. Therefore it must be demonstrated that the new representation will have no demonstrable impact on physics results. Finally, most of our detector experts were heavily involved in data analysis, and some had moved on from STAR to other experiments. Thus, the bulk of the effort had to be carried out by the core Software and Computing team. Based on these considerations, several design goals were developed:

- The geometry model should be translated into the new representation with high accuracy
- Regression tests must be performed to demonstrate the equivalence of the new representation
- Automated tools must be developed to migrate the original code to the new format, and
- There must be a path to create both ROOT/TGeo and GEANT3 implementations.

Over the past decade, the community has introduced several options for the generic representation of detector models. These include XML-based languages, such as the Atlas Generic Detector Description (AGDD) [2] and the Geometry Description Modeling Language (GDML) [3], and abstract interfaces to geometry frameworks, such as the Virtual Geometry Model (VGM) [4]. We considered utilizing one of these frameworks in updating our geometry model. However, significant effort would have been required to add features used routinely in our geometry model, e.g. flow control, data structures, and support for GEANT 3 style composite shapes (MANY). In the end, we decided that the most efficient path would be to update the AgSTAR language concepts, migrating to an XML approach, and implementing new C++ support libraries to instantiate the concrete ROOT/TGeo geometry.

3. Abstract Geometry Markup Language
The AgSTAR language provided a rich syntax for the representation of a GEANT3 geometry. The MORtran preprocessor was used to extend the FORtran programming language with geometry-specific syntax, replacing calls to the GEANT3 library with simple statements for
the creation and positioning of volumes. AgML retains these and other features of the AgSTAR language, and formally incorporates flow control (IF-statements) and looping (FOR- and WHILE-loops) into the language. These additions enable AgML to support multiple target languages, such as C++ and FORtran. Like its predecessor, AgML provides a compact syntax for the definition of detector elements. This is illustrated in listing 1, a snippet of code from the STAR Beam-Beam Counter (BBC) module. A volume comprises an entire code block in XML, with the material, medium, shape and visualization attributes specified within the same code block. The advantage of this approach is that the developer need not search the document to find the definition of the material or shape. It is right there associated with the volume. Looping and flow control are implemented as markup elements, and AgML supports C-like expressions. These features give the developer a powerful mechanism for the creation and placement of daughter volumes in the parent: algorithms can be specified which take advantage of the symmetries inherent in detector design, avoiding difficult-to-maintain data tables.

Listing 1. AgML syntax describing the triple hexagonal module of the STAR Beam-Beam counters.

Like its predecessor, AgML provides several features which are useful for both the rapid development of detector models and their maintenance. These are:

Inheritance

AgML implements inheritance rules which streamline the development process. When writing a volume block, the developer is only required to specify what type of shape the volume is. All other parameters, the material, medium and even the dimensions of the shape, may be omitted. In this case, the framework will inherit the properties of the volume from the code block which created it.

Parameterization

GEANT 3 allows parameterized volumes, in which definition of the shape of a volume can be deferred until it is positioned. AgML extends this concept. When a volume is referenced
in AgML, the support library tries to determine if any of the properties (material, medium or shape) have changed since the last reference. If they have, a new volume will be created with a new name and the currently specified properties.

C-like data structures
AgML provides C-like data structures. AgML structures enable the developer to group related geometrical constants together, and provides the mechanism for iterators and versioning of data, and steering geometry modules. Structures support the inclusion of three basic types (int, float and character data), as well as 1- and 2-dimensional arrays. An example data structure is illustrated below.

```xml
<Structure name="HEXG">
  <var name="type" type="int" />
  <var name="irad" />
  <var name="clad" />
  <var name="thick" />
  <var name="offset(3)" />
</Structure>
```

Iteration
AgML structures may be filled with multiple data sets. Each time a data set is loaded into a structure, the previous data set is saved in memory. At any point, any of the data sets may be retrieved by a special "Use" operator, which retrieves a data set containing one variable with a specified value. By defining a unique key for each data set, the developer may easily iterate over all of the data sets in the structure.

Versioning
Structures also enable a single module to support multiple configurations of a detector. A typical use case in STAR has been the staged installation of major detector subsystems. For example, the STAR Endcap calorimeter was installed over the course of three runs. All of the relevant geometry information is encoded into three data sets contained in AgML structures. Each data set corresponds to either the full or partial installations of the detector. Creating the different versions of the calorimeter is simply a matter of selecting the appropriate data set. This is an important feature in a geometry framework. As the experiment progresses, and the geometry model is refined, those improvements are immediately available in previous versions of the geometry.

Steering
Since each geometry module supports multiple versions of a subdetector in STAR, a mechanism is needed to steer the creation of a specific run’s geometry. AgML provides facilities to manipulate a module's data structures from a main steering routine. Thus, we pass configuration information from the main geometry module to the modules which create each subdetector through data structures.

4. Language Framework
AgML is an XML-based language. It is supported by a parser framework which translates the language into compilable source code, and a C++ library which is responsible for generating the concrete geometry model. The parser framework is implemented in python, and utilizes a plugin model. A single python module “Syntax Handler” (SH) establishes the interface between the front-end language parsers and user-defined backend export modules. A backend exporter implements a set of classes, one for every markup element defined in AgML (e.g. Volume, Material, etc...). Each of the backend classes is responsible for accepting the geometry constructs and/or compilable code to a target programming language (e.g. C++, MORtran). The resulting
code is compiled and linked with a support library which is responsible for steering the creation of the concrete geometry model at run time. AgML is currently capable of representing all of the functionality of GEANT 3 geometry models, with support for new ROOT/TGeo features — such as composite shapes, volume assemblies and misalignments — planned in the near future. This geometry representation and the associated language tools offer an interesting path for experiments to update their legacy geometry codes. It is conceivable that, with additional integration work, AgML could be integrated into the ROOT framework.

In order to support the automated translation of our detector model to the new language, we have developed three backends for the AgML parser:

(i) An AgSTAR backend capable of writing the original AgSTAR representation
(ii) An AgML backend capable of writing the new AgML representation
(iii) A C++/ROOT backend, capable of writing a set of C++ classes which create the concrete ROOT/TGeo implementation through calls to an AgML support library

We have also implemented two frontends for parsing the source files. The first is an XML parser for reading in the AgML files. The second is an AgSTAR parser, capable of recognizing the original representation of the geometry model. This allows us to translate the geometry from the original to the new format.

![AgML parser framework diagram](image)

**Figure 1.** The AgML parser framework. Frontend modules parse files in either the AgML or AgSTAR formats. A common “syntax handler” passes identified syntax elements to user-defined backends, capable of exporting the code to various formats. Support for the original AgSTAR format, the new AgML format and C++ classes capable of generating ROOT/TGeo geometries have been implemented.
5. Validation
Four sets of validation tests were applied, to ensure that the AgML representation accurately reproduces the STAR geometry model.

5.1. Self Consistency
The first test sought to ensure that the parser was able to consistently read and write both the AgML and AgSTAR representations of the geometry, and was useful in quickly debugging the parser framework. The AgSTAR representation was first translated to AgML. The AgML files were then parsed and exported back to the same AgML representation. The consistency check required that the exported AgML file was identical to the original AgML file\(^1\). A second consistency check was applied, whereby the AgML files were exported to the AgSTAR representation, the resulting AgSTAR files were parsed and exported back to the AgML representation, and the process repeated. Consistency between the AgML (AgSTAR) files from successive iterations was required\(^2\).

5.2. Monte Carlo Hits
The next test performed ensured that Monte Carlo simulations using the new geometry representation would be identical to those using the original. A common set of particles (muons) was propagated through each production release of the STAR geometry starting with year 2005. The muons were propagated through the concrete GEANT3 geometry, created from both the original representation and the new AgML representation. Figure 2 shows a hit-by-hit comparison of the simulation for the $Y2010C$ geometry. The results are shown for the Time Projection Chamber (TPC), the main tracking detector in STAR. As shown by the panel in the right, the energy deposited in each hit perfectly correlates between the AgML geometry (y-axis) and AgSTAR geometry (x-axis).

5.3. Material Budget
The amount of material in the detector was determined using a custom Virtual Monte Carlo application StarBASE \(^9\) which was developed primarily for this purpose. StarBASE tracked single geantinos through a ROOT implementation of the STAR geometry model using the full navigation machinery of ROOT. Volume-by-volume comparisons were made between the material encountered using the original representation of the detector\(^3\) and the AgML representation. Figure 3 shows a comparison of the material budget in the STAR Time Projection Chamber, the main tracking detector in STAR, for the Y2010c geometry release. Differences are small, at or below the level of 0.1%, and considered acceptable for both reconstruction and simulation.

5.4. Tracking
The results of the tracking algorithms applied in STAR depend on the accurate representation of the material in front of the tracking detectors. In order to determine the impact of AgML in a realistic tracking environment, the AgSTAR and AgML geometries were used in the reconstruction of minbias AuAu collisions at $\sqrt{s} = 200$ GeV from the 2010 run. Figure 4 compares the number of reconstructed vertices per event, and the $p_T$ distribution of the resulting tracks. The two distributions are a perfect match.

\(^1\) Up to meaningless whitespace.
\(^2\) Again up to meaningless whitespace.
\(^3\) Translated to ROOT using g2root.
Figure 2. Hit-by-hit comparison of single muons propagating through the TPC. The left panel shows the number of hits registered for the AgML geometry (y-axis) and the original AgSTAR geometry (x-axis). The Right panel compares energy deposition of the hit in the AgML geometry (y-axis) and AgSTAR geometry (x-axis).

Figure 3. Material budget as a function of pseudorapidity for the STAR TPC. The left panel shows the material budget for the TPC envelope and its daughter volumes (top) and the percent difference between AgML and AgSTAR (bottom). The right panel shows the material budget for the TPC gas volume, and the sensitive volumes (virtual cells in gas volume) of the TPC.

6. Conclusion
The STAR experiment has been carrying out an important and leading edge research program for well over a decade. During this time the field has moved away from technologies such as
Figure 4. $\phi$-distribution of tracks (left) and number of primary vertices per event (right) reconstruction using AgML and AgSTAR geometries. The bottom panels show the ratio of the distributions obtained using the AgML geometry to those of the AgSTAR geometry.

FORtran and GEANT 3, in favor of C++, ROOT and Geant 4. With a rich experimental program ahead of us in the coming years, anticipating major detector upgrades, and planning for operations in the eRHIC era, it became clear that we must migrate away from our legacy GEANT 3 geometry model. To that end we have developed the Abstract Geometry Modelling Language framework, as an extension to the AgSTAR language which has served us well over the years. Detailed regression tests were performed to ensure that the new representation reproduces the results of the original, and we can begin using AgML in a production environment without any expected impact on physics results. The AgML framework retains the best features of the original. It is supported by a parser framework and support libraries which allow us to create multiple concrete implementations of our geometry model, and rapidly take advantages of new technologies if and when they emerge, such as Geant 5 [10] and the new solids library proposed in this conference [11].

References
[1] R. Brun et al., The ROOT Geometry Package, Nucl. Instrum. Meth. A502:676-680,2003.
[2] L. Chevalier, et al., An XML generic detector description system and geometry editor for the ATLAS detector at the LHC, this proceedings.
[3] R. Chytracek, The Geometry Description Markup Language, Conf. Proc. C 0109031 (2001).
[4] I. Hřivnáčová, B. Viren, The Virtual Geometry Model, J. Phys. Conf. Ser. 119 042016 (2008).
[5] A. Artamonov et al., DICE-95, internal note ATLAS-SOFT/95-14, CERN, 1995.
[6] GEANT - Detector Description and Simulation Tool, CERN, Geneva 1993.
[7] S. Agostinelli et al., Geant4 - a simulation toolkit, Nucl. Instrum. Meth. A506 (2003) 250-303.
[8] I. Hrivnacova et al. [ALICE Collaboration], eConf C 0303241, THJJT006 (2003) [cs/0306005 [cs-se]].
[9] J. Webb et al., StarBASE: Fighting and Tracing Geometry Changes by Applying Differential Studies, J. Phys. Conf. Ser. 331 032033 (2011)
[10] GEANT project, http://geant.cern.ch
[11] M. Gayer et al., New software library of geometrical primitives for modelling of solids used in Monte Carlo detector simulations, this proceedings.