USING XML FOR DETECTOR GEOMETRY DESCRIPTION IN THE VIRTUAL MONTE CARLO FRAMEWORK

V. Fine, J. Lauret, M. Potekhin, Brookhaven National Laboratory, Upton, NY 11973, USA
J. Hrivnac, Laboratoire de l'Accélérateur Linéaire, Orsay

Abstract

The STAR Collaboration [1] is currently migrating its simulation software based on GEANT3 [2], to the ROOT-based Virtual Monte Carlo *(VMC)* Framework [3]. One of the critical components of this framework is the mechanism of the Geometry Description, which comprises both the geometry model as used in the application, and the format that allows the users to define and maintain the detector configuration on the ongoing basis, throughout the lifetime of the experiment.

Having chosen the ROOT [4] geometry library as the platform for the geometry model, we have elected to employ a structured and platform-neutral mechanism of the geometry description, not restricted to just ROOT as a target platform. As a starting point, we followed the path of reusing the already available and proven components such as the AGDD schema and the GraXML application [5]. We have enhanced the initial concept and structure of this application and created a geometry parser based on it.

In our approach, the structured geometry description written in XML [6] in compliance with the enhanced AGDD schema is converted into C++ code, suitable for input into the ROOT system. It is therefore possible to use the thus defined geometry in the Virtual Monte Carlo simulation framework.

We present the features of the enhanced schema, and the ideas and technique behind the XML to C++ parser.

CURRENT STATUS OF THE SIMULATION SOFTWARE IN STAR

Over the years, the STAR Collaboration has used simulation software based on GEANT3 libraries, with a significant amount of infrastructure code added. The geometry of the STAR apparatus is described in a Fortran-based macro language, and can be made persistent in the form of ZEBRA files. The code is structured as approximately 20 source files, each describing a particular subsystem. The versioning is done in the steering module which governs the calling sequence of subroutines which create the model of the detector step by step, and where various flags are set in order to define standard configurations, which are easily referenced by the users via mnemonic names (tags).

Despite being a successful and mature system, the STAR simulation software is nearing the end of its lifecycle, as many of its components are becoming obsolete and difficult to maintain.

In the past few months, a new system based on the Virtual Monte Carlo *(VMC)* platform was created and integrated into the rest of the STAR off-line software. During the transition period, both new and old systems are using the same geometry description as existed before, with automatic translation of GEANT3 geometry structures (ZEBRA) into ROOT ones as needed.

In addition, the introduction of the new tracking software into the STAR framework has brought in a home grown geometry model which is distinct from both GEANT3 and VMC, and somewhat complicates the situation.

MOTIVATIONS FOR THE NEW GEOMETRY MODEL AND DESCRIPTION

The concept of "geometry model" refers to memory resident data structures representing the detector's configuration, which are used by an application such as an event reconstruction routine, a visualization system or a simulation tool.

Geometry description, on the other hand, is a human–readable (and often human-generated) document, which serves as source code for the creation of the geometry model at run-time.

We have the following motivations for evolving the geometry model and description in STAR:

- Retiring GEANT3 is imminent; the set of tools developed within its framework does not provide for the unified geometry model and description, which has been a long standing goal of the STAR collaboration. Resolving this issue will increase the transparency of the analysis, eliminate work duplication and possible (in fact, actual) coding errors and inconsistencies.
- The emergence of the new tracking software used in STAR makes better integration of software components even more important than before. To be successful, such integration should include a single source, unified geometry model as well.
- Since STAR is pursuing a vigorous program of detector upgrades, we need new and more efficient tools for detector development, rapid prototyping of geometrical configurations, simulation and visualization.
- Our goal is a single geometry model fed from a single detector description source, and that would also be used in the tracking software

THE GEOMETRY PLATFORM

The ROOT-based geometry (i.e. an assembly of objects based on the ROOT classes designed for geometry modeling) is a logical choice for any STAR application because of the heavy use of ROOT in most software systems created and run in our experiment. In addition, the ROOT geometry model has been successfully tested in the STAR VMC context.

As mentioned above, the geometry description mechanism is distinct from the geometry model per se, and as such should be treated separately.

Consider the choice of the C++ language as the platform for the detector geometry description. Obviously, it has the advantage of being familiar to the users, and its translation into the geometry model is trivial, since it amounts to compilation or interpretation of the C++ code. However, it also has the following disadvantages:

- in a large-scale experiment, there is an ample probability of writing obscure and inaccessible code due to differing coding styles and difficulty of maintaining standards in a free form language such as C++
- no code validation tools exist, i.e. one has to compile and load the code to validate the geometry
- code written as in C++ will remain largely non-portable to other systems, although in case of ROOT this is believed to be mitigated by exporting the geometry to a particular XML schema or GDML and using it as an exchange format
- the choice of Geometry Visualization tools will remain limited by what is created by the ROOT developers or end users

XML for the geometry description

We observe that a real advantage would be in a different and preferably platform-neutral language that can be parsed into the ROOT compatible format and also accessed by other applications and platforms. We have therefore chosen an approach that calls for a structured Geometry Description (as opposed to free-form C++), based on XML. In our view, this has the following advantages:

- XML is an industry standard with a large number of both commercial and user-supported tools, such as highly advanced editors which assist the user in writing and validating the code
- Hierarchical structure of the XML document naturally maps onto the application realm of the geometry model used in simulation
- There is flexibility to create a schema that is best suited for the application
- XML has a potential to facilitate interaction with the parameter database
- Validation: in XML, there are methods of enforcing the rules and verifying that the code is well-formed and complies to a chosen schema
- it opens a possibility to interface other applications (e.g. CAD and visualization), using a suitable XML transform

Having chosen XML as the description mechanism, we had an option of developing a XML schema and other requisite tools from scratch, or using the experience, and hopefully some code, of other groups. We have studied the merits of the following advanced schemas and associated tools:

- AGDD – Atlas Generic Detector Description
- GDML – Generic Detector Modeling Language
- CMS DDD – Detector Description in CMS

Based on factors that included accessibility of the code, possible level of interest and cooperation by the authors, and most importantly, the feature set, we had chosen the AGDD schema as the basis for our development effort.

The AGDD schema features

The features of the AGDD schema make it particularly suitable for the detector geometry description (not surprising as it was designed for that purpose, see ref [5]):

- Full set of classes to describe solid shapes, suitable for any simulation setup
- Hierarchical organization of the geometry code, with objects being grouped and nested, which maps well onto geometry models in most Monte Carlo systems
- Support for variables, one- and two-dimensional arrays for numerical data storage, which is still a fairly unique feature among the candidate schemas
- Support for arithmetic calculations and certain functions (done at parse-time). Issues of numerical accuracy can be addressed, if needed.
- Variety of multiple positioning operators that facilitate the creation of complex geometries
- Support of boolean operations
- Bona fide iterator facility (the “for” loop), which allows the developer to create complex, parameterized structures
- Support for XInclude, which allows for optimal source code sectioning and organization, and aids in versioning.

A few features of the schema are illustrated in a small code sample below (Fig.1)

CONVERTING THE XML CODE

The AGDD schema was developed a while ago in parallel with a visualization tool named GraXML[5]. Since it already included a fairly good AGDD parser, a decision was made to reuse parts of its code and develop additional classes required in order to generate ROOT geometry based on the XML.

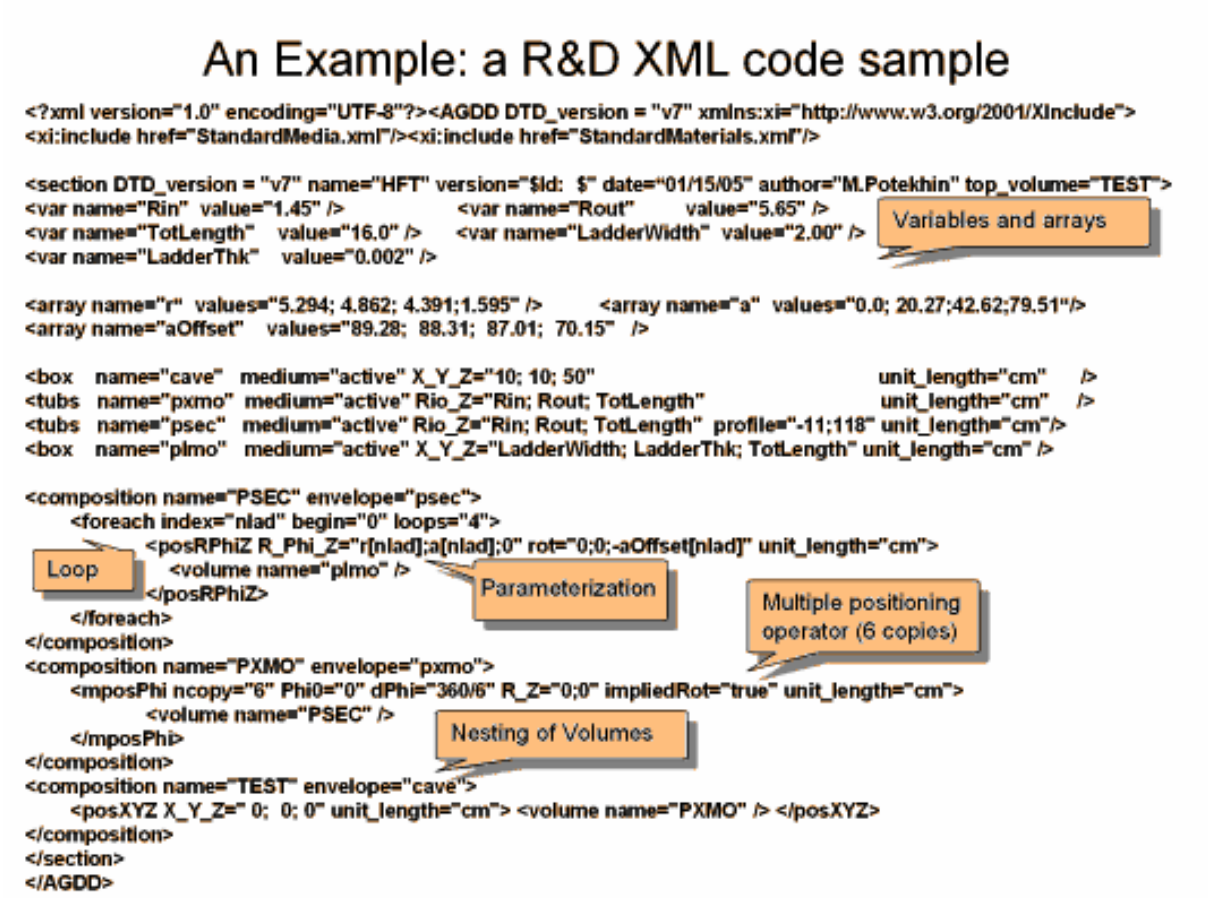

Figure 1: A snippet of AGDD-compliant geometry code: silicon wafers of the proposed STAR Heavy Flavor Tracker. A few useful features of the schema are highlighted. The code can be converted into a ROOT model (visualized in Fig.2)

The GraXML is a Java application. It parses the XML input and in doing so creates the so called *Generic Model*, which is a tree of typed objects (not just containers for string data typical for DOM-based parsers) representing the input data according to the user-supplied schema. As the name suggests, at this point the model is generic and not prepared yet for any application.

It is then traversed in order to inflate a geometric model that can be visualized. The GraXML visualization layer is based on the Java3D graphics library [7]. After the traversal takes place, the Java3D object tree can be readily visualized with a suitable tool. In Fig.3, a view of a few top-level objects of the STAR detector geometry is presented, as a screenshot taken from the GraXML GUI.

However, the traversal process can also be used to construct other types of objects, such as C++ (ROOT or other) tree of graphics objects. This can be done in-memory, or by generating the C++ source code as output. We chose the latter option as more straightforward and providing another layer for debugging. In this approach, one needs to create a set of Java classes representing objects declared in the schema – solids, positioning operators etc. These classes must have the behavior which provides automatic generation of correct C++ code.

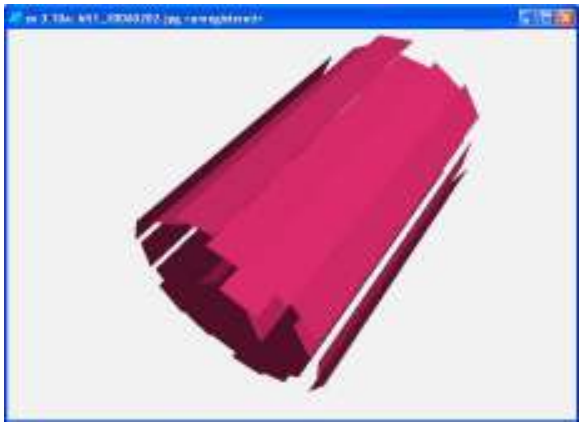

Figure 2: Visualization of the detector geometry described in the XML example in Fig.1, in the STAR ROOT geometry viewer

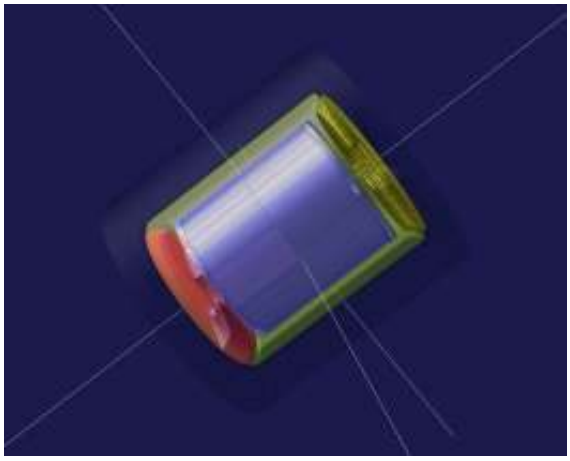

Figure 3: GraXML view of the simplified STAR detector geometry

In addition, one has to address a few issues related to the specifics of the target platform (ROOT) which are not reflected in the AGDD schema. For example, when creating instances of the same solid, with same dimensions), we must implement a referencing mechanism similar to what was used in GEANT and what is now also a feature of ROOT geometry – a single object is positioned multiple times with different "copy numbers". In order to achieve that, we implemented a hashing scheme whereby a database of existing solids is checked for existing instances of a solid with equivalent parameters, and no extraneous objects are created.

The AGDD schema

The development effort that was needed to implement a working geometry solution included not only the parser component, but also additions to the schema, in order to better tune it to a realistic and large scale application such as a complete STAR detector geometry description.

Examples of such additions and changes include:

- The "BuildingBlock" XML tag, which allows grouped descriptions of individual parts that can be organized as separate XML files referenced in various ways, such as via the XInclude mechanism. This is fairly crucial for proper code organization
- a full complement of new tags corresponding to the media and materials that, while being quite generic, maps onto the realm of similar ROOT classes
- new useful features of the colouring scheme

Mode of operation

In it's current implementation, the user starts the GraXML application in conjunction with additional Java libraries, and the code generation process happens in parallel with the parsing which is normally done for visualization purposes. It is also possible to create a command-line driven application that does not have a GUI.

The C++ code produced in this manner can be loaded into ROOT system to be used in simulations (work in progress) or into a ROOT based tool, such the one used for a screenshot in Fig.2.

Validation and consistency checks

Checking the validity and consistency of the geometry source code is an important part of the geometry development process.

At the most basic level, the XML code is validated against the supplied schema in the IDE used by the geometry developer. This is done automatically, for example in the Altova XMLSpy suite [8] and many others. Of course, there are other kinds of validation which can include automatic detection of protrusions and overlaps in the geometry. For that purpose, a full target geometry build is still necessary.

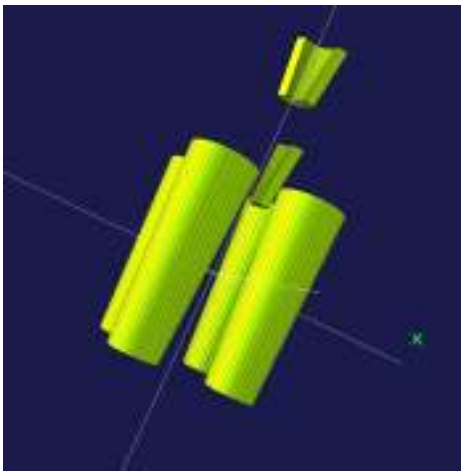

Figure 4: A GraXML view of an assembly of Polycone objects described in XML

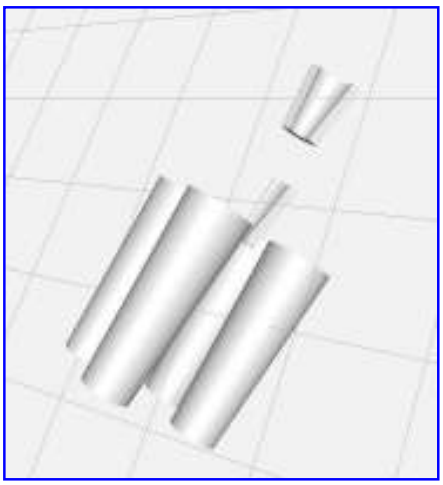

Figure 5: A ROOT view of the same assembly of Polycone objects after the C++ conversion (cf. Fig.5)

Then, there is a question of validity of the conversion of the Generic Model into C++. This can be addressed, for example, by visualizing the geometry in GraXML and then, its C++ counterpart, in a ROOT-based viewer. This indeed has been routinely done, and a very simple test example is illustrated in Figs. 4 and 5 above.

DEFINING THE GEOMETRY FOR THE STAR VMC

The STAR Virtual Monte Carlo system, which is under development, employs the ROOT classes for geometry navigation. As such, it can immediately use the following forms of input for the geometry description:

- a ROOT format file
- a piece of C++ code (cint macro)

Since we don't have a detailed XML model of the STAR detector yet, an automatic conversion procedure is currently in place, which builds the ROOT geometry based on the ZEBRA data structures saved in GEANT3, thus allowing us to continue development and testing. This means that the primary geometry description source if still in FORtran and this is obviously a temporary solution.

A permanent solution will include a complete XML model of the STAR geometry, with appropriate configuration and version control, which is parsed into a ROOT-compliant C++ code.

CONCLUSIONS

We have chosen ROOT as the integration platform for the geometry model used by various applications in STAR. We identified XML as the proper tool for the geometry description and maintenance.

We have developed a XML parser that generates C++ code that can be utilized by ROOT-based applications, be it simulation or visualization tools. To this end, we augmented the previously developed schema that originated in the Atlas Collaboration (AGDD) and reused parser elements of a Java-based visualization tool, the GraXML. The desired functionality was been achieved by developing a specialized Java library.

The functionality of the parser was validated using a number in test cases, by comparing the results of GraXML parsing and visualization, to the corresponding ROOT geometry model studied in a separate geometry browser.

We have not addressed yet the issue of the database interface, which will be necessary if one decides to feed the numerical values into the XML tree, from an external source, potentially shared with other applications (such as conditions database). This will be resolved at a later point.

REFERENCES

[1] The STAR Experiment at BNL: http://www.star.bnl.gov/
[2] Geant 3 documentation: http://wwwinfo.cern.ch/asdoc/geant_html3/geantall
[3] Virtual Monte Carlo: http://root.cern.ch/root/vmc/VirtualMC.html
[4] The ROOT system documentation: http://root.cern.ch/root/
[5] The AGDD schema: http://root.cern.ch/http://hrivnac.home.cern.ch/hrivnac/Activities/Packages/AGDD/
[6] The Extended Markup Language: http://www.xml.org/
[7] Java3D: http://www.j3d.org/
[8] Altova XML Spy Integrated Development Environment: http://www.altova.com/