Thoughts on LBNE Geometry Description and Model

M.Potekhin

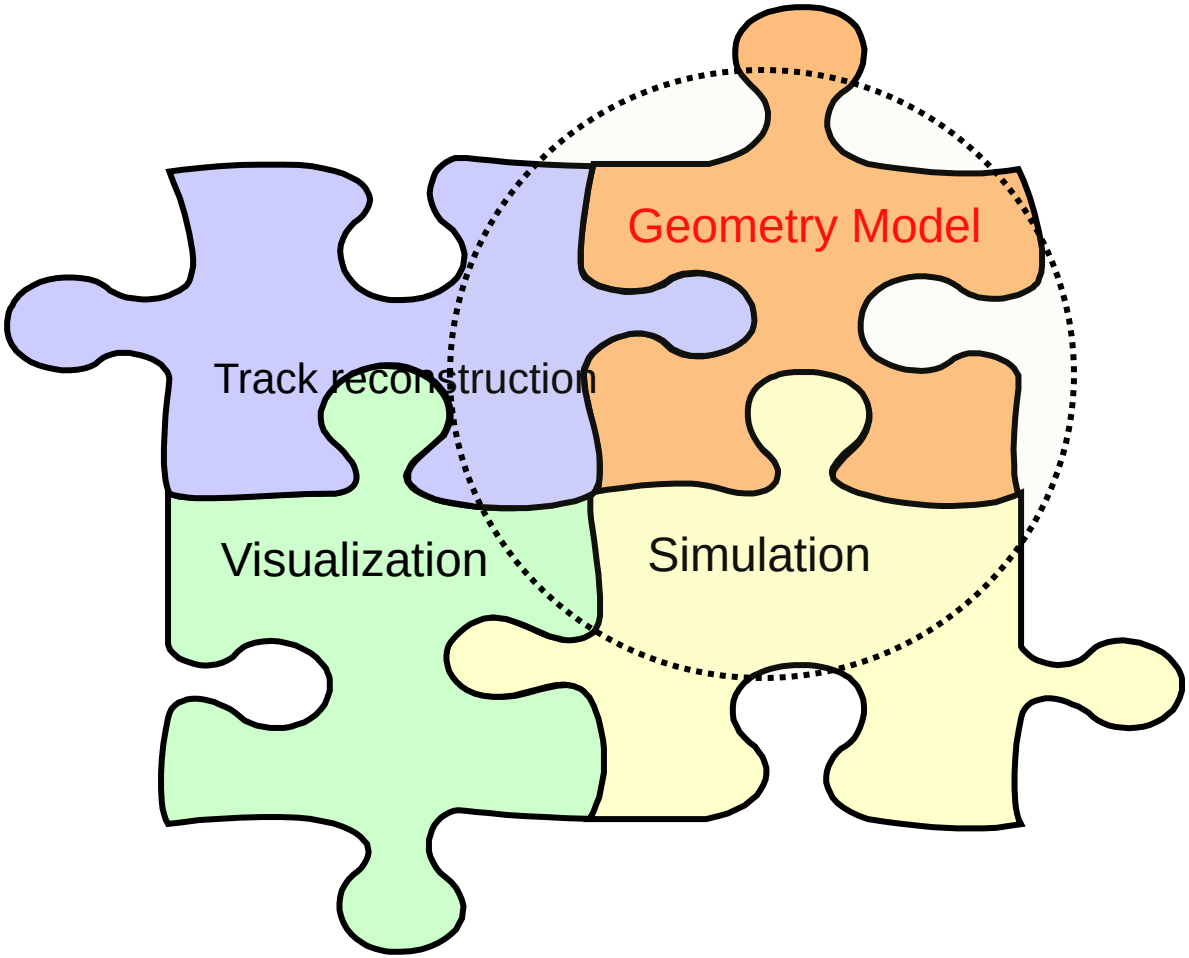
Brookhaven National Laboratory

LBNE Software Workshop

FNAL, November 14th 2013

Geometry Modeling and Description

Ideally, we aim for a single geometry model and/or description (single source) in simulations, event displays and tracking



Geometry Modeling and Description

These definitions are my own, based on prior MC work, and are meant to set the context for the discussion:

Geometry model refers to a memory resident data structures representing the detector's configuration, which are used by an application.

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

These two items are manifestly separate.

Geometry Model and its Platform

The dominant geometry platform in LBNE is G4, but there may be variants of that and other platforms..

There are other systems such as ROOT VMC that might be interesting to consider.

There are also translation tools which in some cases allow conversion of one model platform to another.

Geometry Description Options

In general, it is possible to identify the following approaches:

- Algorithmic code, i.e. pure form G4 C++ code – hard to manage the data component (dimensions, positioning etc). Notoriously unwieldy!
- Mixed approach – C++ code with well separated data components in DB – this is a workable approach if the schemas (which are mapped to the geometrical structures) are stable, which is not always the case.
- Structured description languages like GDML – the latter does not scale well to real life complex detectors and usually needs to be augmented by an ad hoc algorithmic part (e.g. Perl afterburner). This is used in certain areas of LBNE. It may or may not be easy to read and maintain.

And further, there is the fourth option, which is a XML schema like GDML, but without its limitations. There is experience in using such for visualization and prototyping the ROOT VMC geometry interface.

XML-based Geometry Description

Note on XML:

- XML being the industry standard with large number of both commercial and user-supported tools, such as highly advanced IDE/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
- Flexibility to create a schema that is best suited for the application
- Facilitates interaction with the database (standard approaches and products exist)
- Validation: standard XML-driven methods of enforcing the rules and verifying that the code is well-formed , before the application is even run
- Opens a possibility to interface with other applications (e.g. CAD and visualization), with a suitable XML transform

One existing solution, as an example

Originated in ATLAS – codename AGDD (not used in ATLAS production now)

- Full set of objects to describe solid shapes, suitable for any simulation setup
- Hierarchical organization of the XML 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
- Support for arithmetic calculations and certain functions (done at parse-time). Issue of numerical accuracy can be addressed, if needed.
- Variety of multiple positioning operators that greatly facilitate the creation of complex geometries
- Bona fide iterator facility (“for” loop), which again allows the developer to create complex, parameterized structures
- Support for Xinclude, which allows for optimal source code sectioning and organization, and aids in versioning.

GraXML

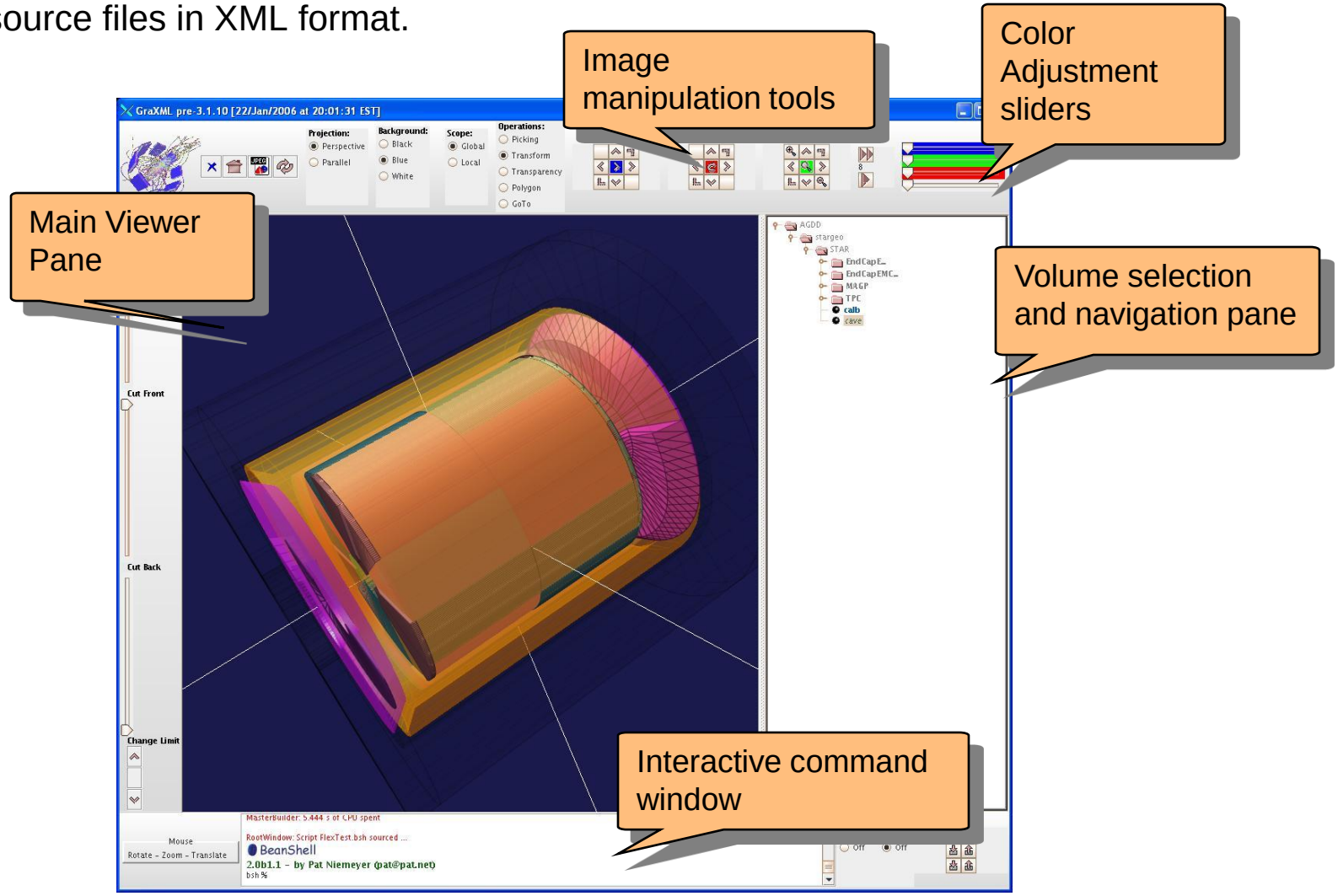
The GraXML is a Java application. It parses the XML input and in doing so creates the so called Generic model, which is a tree of typed objects (not just containers) representing the input data according to the user-supplied schema (in the this case, AGDD).

The tree is then traversed in order to inflate a geometric model that can be readily visualized. The visualization layer is based on the Java3D graphics library. Can create other layers if necessary.

The traversal process can also be used to construct other types of objects, such as C++ graphics objects. This can be done in-memory, or by generating the C++ source code (or any type of code) as output.

GraXML Viewer

Accompanying product for the AGDD schema: an advanced graphics viewer known as GraXML (credit: J.Hrivnac). Allows full visualization of AGDD-compliant geometry source files in XML format.



GraXML Status

- The viewer can be run either as a standalone Java application, or within a browser.
- The code currently rebuilt
- There is a minimal level of support, however this can be ramped up by a formal request (Orsay, France)

Geometry Parsing

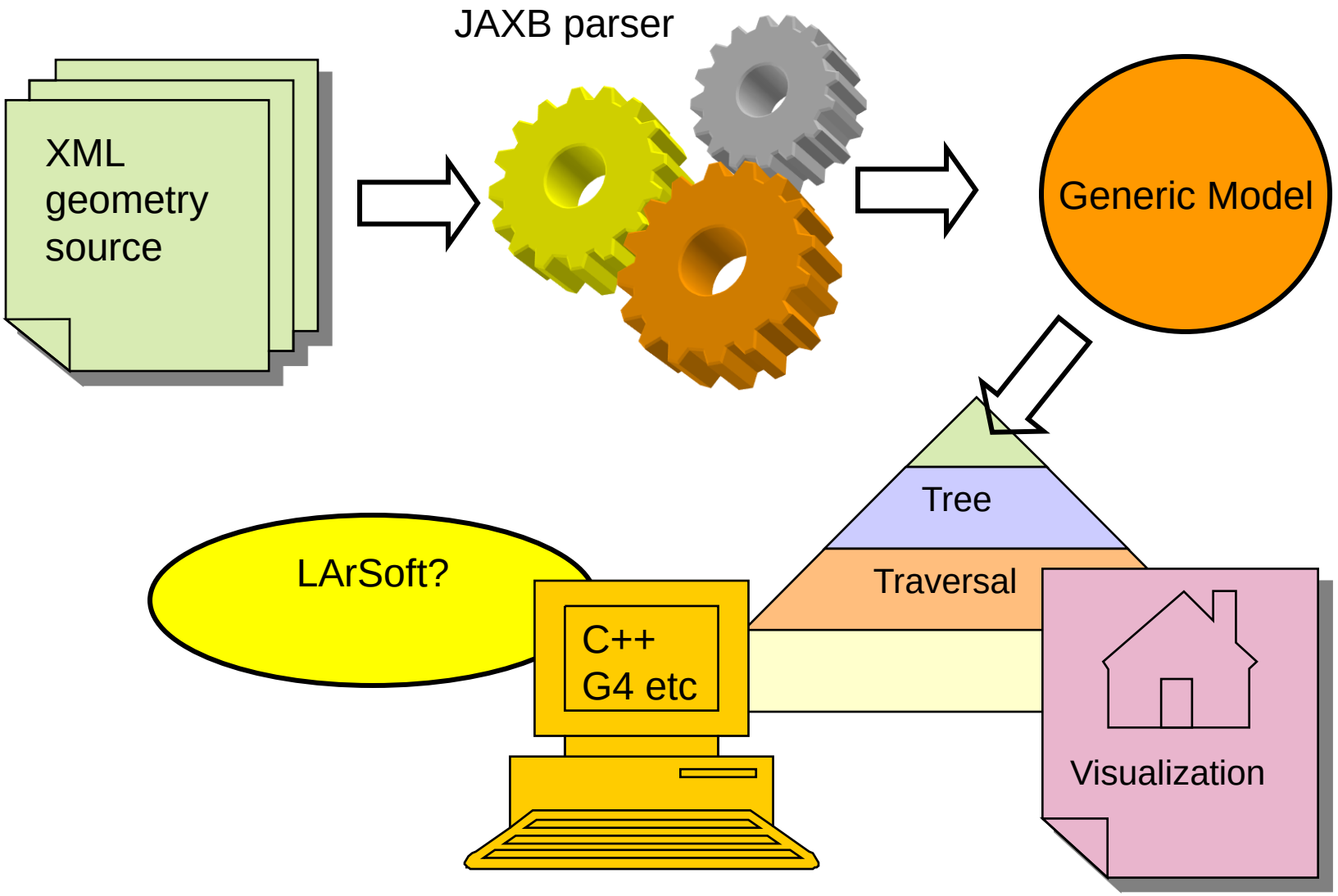
Choices:

- Parse the XML in C++ and create a geometry model inside a Root application – (SAX, DOM?)

- Adopt and reuse the JAXB parsing technology from the GraXML viewer. Advantages:
  - JAXB parser creates a model of the document which consists of typed objects, with the XML schema serving as the object declaration tool. This is in contrast with DOM parsers which simply create hierarchies of containers of string data and thus need significant amount of additional logic to correctly
  - a proven parser with immediate visualization

There is experience with both.

Geometry Parsing and Code Generation



An Example: a R&D XML code sample

```
<?xml version="1.0" encoding="UTF-8"?><AGDD DTD_version = "v7" xmlns:xi="http://www.w3.org/2001/XInclude">
<xi:include href="StandardMedia.xml"/><xi:include href="StandardMaterials.xml"/>

<section DTD_version = "v7" name="HFT" version="$Id:  $" date="01/15/05" author="M.Potekhin" top_volume="TEST">
<var name="Rin"  value="1.45" />                <var name="Rout"        value="5.65" />
<var name="TotLength"    value="16.0" />     <var name="LadderWidth"  value="2.00" />
<var name="LadderThk"    value="0.002" />

<array name="r"  values="5.294; 4.862; 4.391;1.595" />         <array name="a"  values="0.0; 20.27;42.62;79.51"/>
<array name="aOffset"    values="89.28;  88.31;  87.01;  70.15"   />

<box    name="cave"   medium="active" X_Y_Z="10; 10; 50"                                    unit_length="cm"      />
<tubs   name="pxmo"  medium="active" Rio_Z="Rin; Rout; TotLength"                          unit_length="cm"     />
<tubs   name="psec"   medium="active" Rio_Z="Rin; Rout; TotLength"  profile="-11;118" unit_length="cm"/>
<box    name="plmo"   medium="active" X_Y_Z="LadderWidth; LadderThk; TotLength" unit_length="cm" />

<composition name="PSEC" envelope="psec">
      <foreach index="nlad" begin="0" loops="4">
            <posRPhiZ R_Phi_Z="r[nlad];a[nlad];0" rot="0;0;-aOffset[nlad]" unit_length="cm">
              <volume name="plmo" />
            </posRPhiZ>
      </foreach>
</composition>
<composition name="PXMO" envelope="pxmo">
      <mposPhi ncopy="6" Phi0="0" dPhi="360/6" R_Z="0;0" impliedRot="true" unit_length="cm">
            <volume name="PSEC" />
      </mposPhi>
</composition>
<composition name="TEST" envelope="cave">
      <posXYZ X_Y_Z=" 0;  0; 0" unit_length="cm"> <volume name="PXMO" /> </posXYZ>
</composition>
</section>
</AGDD>
```

Variables and arrays

Loop

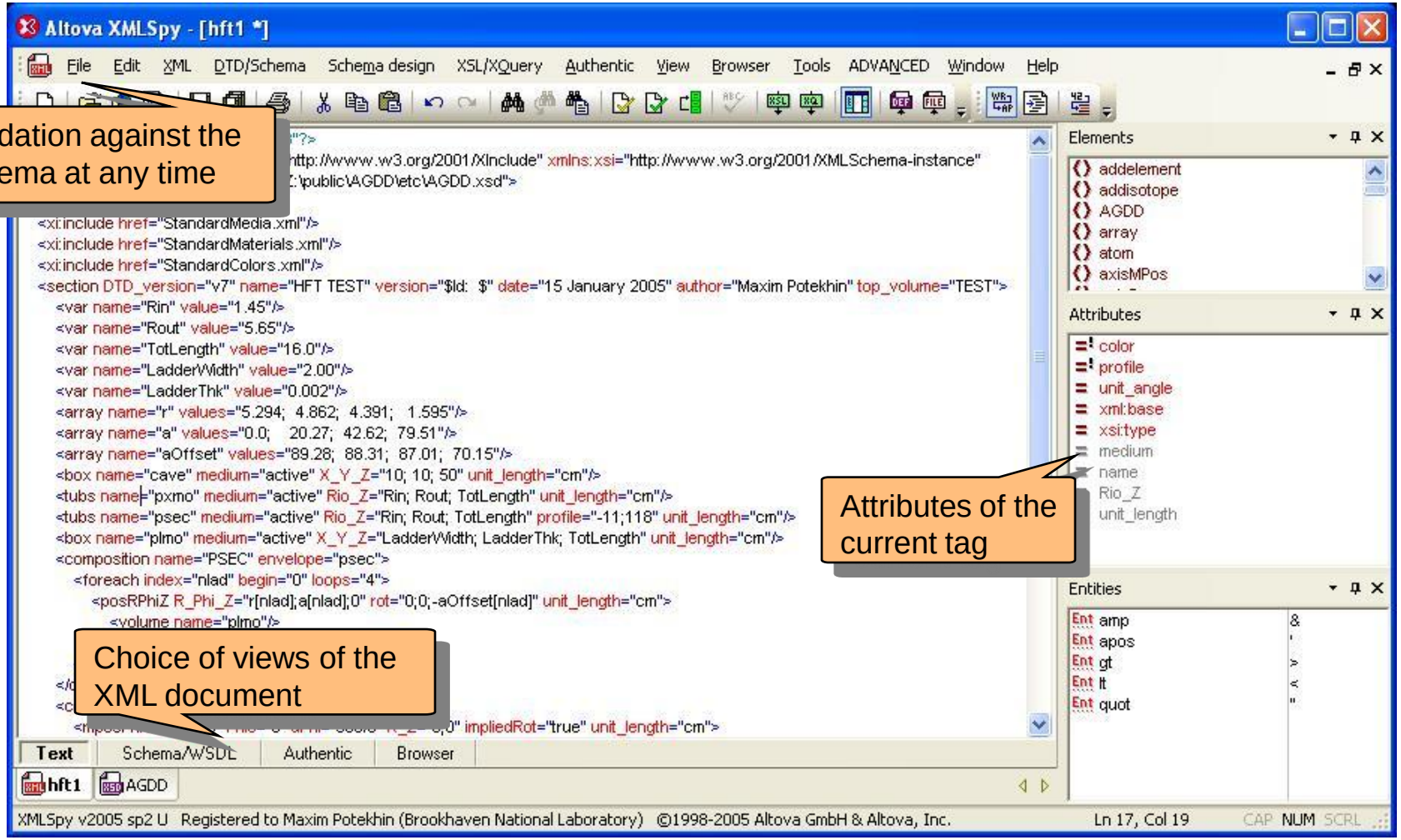
Parameterization

Multiple positioning operator (6 copies)

Nesting of Volumes

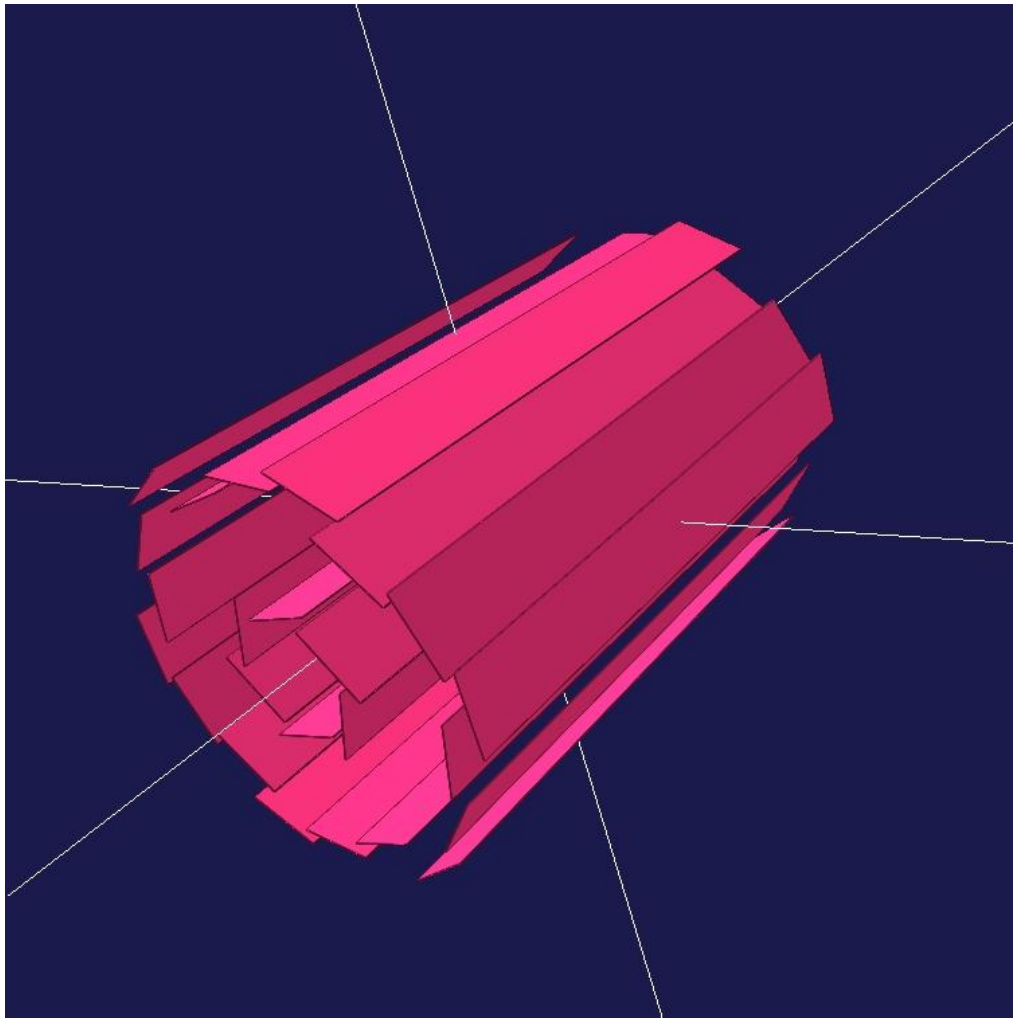
Using IDE tools in XML development

Some helpful features of the editor, Altova XMLSpy (same code sample as above)



Am example: a R&D XML code sample

Continued from the previous page: visualization of the XML geometry code for the central tracker in STAR, with complex positioning of the silicon sensors



An Example: generated C++ code sample

Continued:

the ROOT C++ code produced by the parser. A small snippet presented due to screen size limitations.

```
TGeoCombiTrans* ct_plmo21 = new TGeoCombiTrans();
psec->AddNode(plmo,21,ct_plmo21);
ct_plmo21->RotateX(0.0);
ct_plmo21->RotateY(0.0);
ct_plmo21->RotateZ(-89.28);
ct_plmo21->SetTranslation(0.53,0.0,0.0);


TGeoCombiTrans* ct_plmo22 = new TGeoCombiTrans();
psec->AddNode(plmo,22,ct_plmo22);
ct_plmo22->RotateX(0.0);
ct_plmo22->RotateY(0.0);
ct_plmo22->RotateZ(-68.04);
ct_plmo22->SetTranslation(0.46,0.17,0.0);


TGeoCombiTrans* ct_plmo23 = new TGeoCombiTrans();
psec->AddNode(plmo,23,ct_plmo23);
ct_plmo23->RotateX(0.0);
ct_plmo23->RotateY(0.0);
ct_plmo23->RotateZ(-44.39);
ct_plmo23->SetTranslation(0.32,0.3,0.0);

TGeoCombiTrans* ct_plmo24 = new TGeoCombiTrans();
psec->AddNode(plmo,24,ct_plmo24);
ct_plmo24->RotateX(0.0);
ct_plmo24->RotateY(0.0);
ct_plmo24->RotateZ(9.36);
ct_plmo24->SetTranslation(0.03,0.16,0.0);
```

An example: a R&D XML code sample

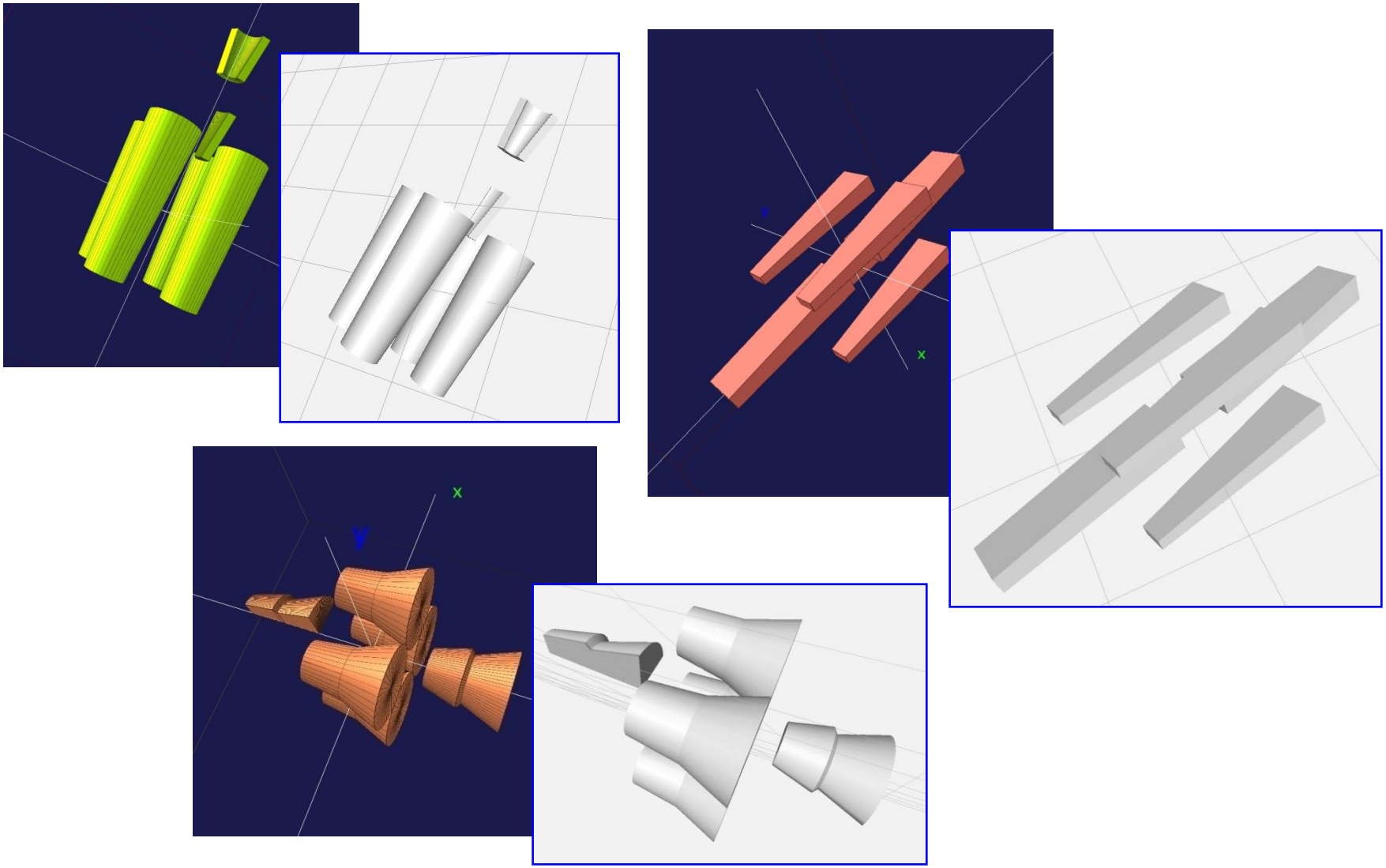
Question: How do we verify the C++ code generated by our parser is valid?
Answer: We use the STAR-developed ROOT geometry browser

Below: See the visualization of the generated C++ code for the same assembly of silicon sensors. *(There was only one light source used in this rendering)*

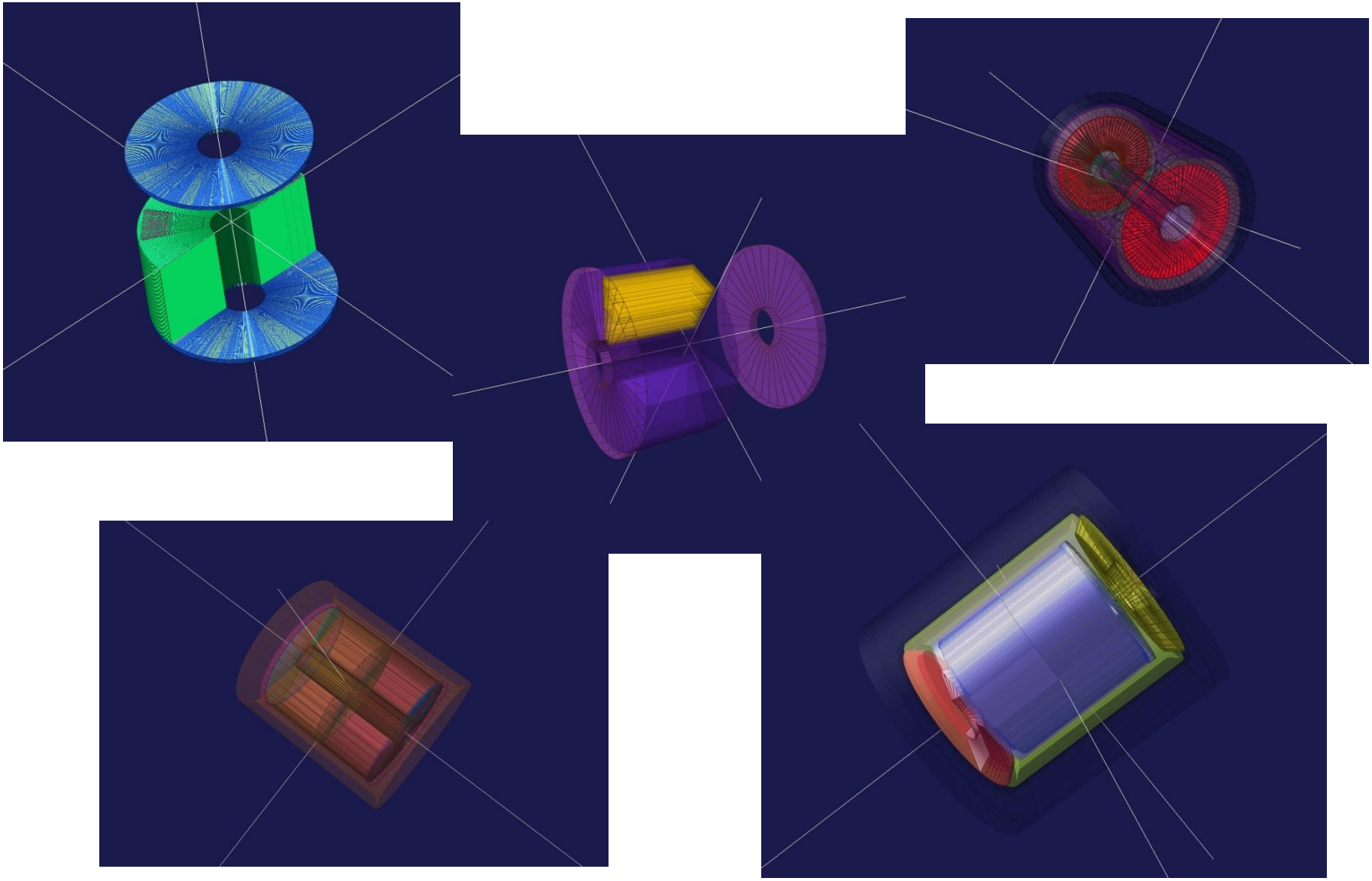
Example: validation of the XML to C++ conversion

The TGeo browser developed in STAR is a valuable geometry development and visualization tool, which is also useful in validation of the XML to ROOT C++ transformation

Compare: visualization of XML in GraXML (left) vs the generated C++ code in TGeo browser (right)



Elements of the STAR geometry coded in XML



Conclusions

- Geometry model and description are an important part of the LBNE S&C planning process, and recently the S&C Requirements
- We are still in early stages of formulating the requirements and these will need to be well understood when it comes to geometry
- I’m not proposing any solution at this point but just want to share experience accumulated in a previous project
- This is one of the most fun parts of the software suite and we should be able to find willing participants to work on these components.