Detector Description and Geometry Frameworks in HEP/NP Experiments

Jason Webb

presented on behalf of the STAR experiment

Introduction

The geometry model is central to the physics programs of HEP/NP experiments, touching on all aspects of the data analysis

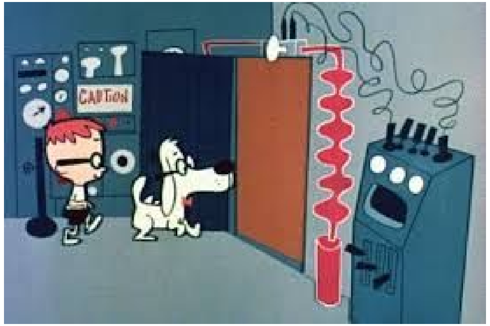
- Developed and supported throughout the entire lifecycle of the experiment
- It represents a *significant* investment in time spent developing, tuning and validating
- While there has been some consolidation, the field has not converged on either a common framework or approach

This talk will

- Present a broad (not exhaustive, possibly biased) overview of requirements on detector description and geometry frameworks
- Make some general remarks about approaches that are typical in the field
- And examine some of the specific frameworks that are available

but first,...

Set the WABAC machine for the year 2000...



The field is undergoing a paradigm shift.

- Old technologies (FORtran, PAW, hbook, GEANT3) are being retired
- New technologies (C++, ROOT, Geant 4) are gaining acceptance

The LHC experiments are planning how to represent complex geometry models consistently over a multi-decade time scale, without certainty of what the underlying technology will be. Two major trends emerge

- Alice develops VMC, adopts ROOT/TGeo. Uses C++ classes as its primary representation of the geometry model. Abstraction at the geometry navigation level.
- ATLAS, CMS and LHCb develop XML-based models for their geometry. Abstraction at the geometry description level.

Requirements (from the viewpoint of the software)

- The detector description *should* provide
  - *Complete definition* of materials, geometry, physics, detector mapping, etc… What gets specified highly dependent on the target application.
  - *Abstraction* from concrete geometry model (e.g. GEANT3, Geant 4, ROOT/TGeo, etc…) which enables path to adopting new models
- The detector description *should* provide the single source of geometry information to applications with conflicting needs
  - Simulation -- requires highly detailed description of everything -- materials and their properties, placement, detector identity, physics and propagation parameters, etc…
  - Event Reconstruction -- may trade off detail in passive volumes for navigation speed / precision alignment of high resolution detectors required
  - Visualization -- material properties relatively unimportant. Level of detail required depends on the intended usage, e.g. event visualization for P.R. versus debugging track reconstruction.
  - Data analysis -- may require detail as complicated as for simulation or as simple as visualization, and everything in between.

Requirements (from the viewpoint of the developer)

Q: Who is the developer? Research scientist? Graduate student? Professor who already has something working in GEANT 3.17 and doesn't understand why he can't just plug his code in?

- Low overhead for the developer -- hard enough to learn good geometry *design* (and more important)
  - Easy to learn and apply / minimize number of languages to be learned / learn it once
  - Minimally the framework shouldn't get in the way
  - Ideally encourages *good organization* of the geometry model.

Q: What stage has the experiment reached? Conceptual design? Evaluating different detector technologies and reconstruction algorithms? Experimental data taking? Long term maintenance / archiving?

- Must support multiple versions of a detector (some form of version control…)
  - Flexible -- R&D needs to be able to easily reconfigure a detector model
  - Stable -- Production needs to be able to select from fixed of known detector models
  - Different experiments may end up different places on a continuum between the two.

GDML -- Geometry Description Markup Language

- Pure XML description of the detector geometry
  - Iteration, constants, variables supports the algorithmic creation of detectors (no branching)
  - Support for large number of shape primitives (G4 compatible)
  - Provides an expression of the geometry, without providing a framework to realize the concrete geometry model. i.e. you'll have to write the code to import GDML yourself, …
    - Or use ROOT and/or G4 to input geometry. But *you* will still need to apply ...
  - Auxiliary information (hints) can be used to pass physics configurations (eg tracking cuts, medium parameters, etc…)
    - But there is no standard for this, and the only two concrete geometry modelers which support (ROOT and G4) do not make use of this feature
- Stable code base
- Primarily useful as an exchange format, would need new development

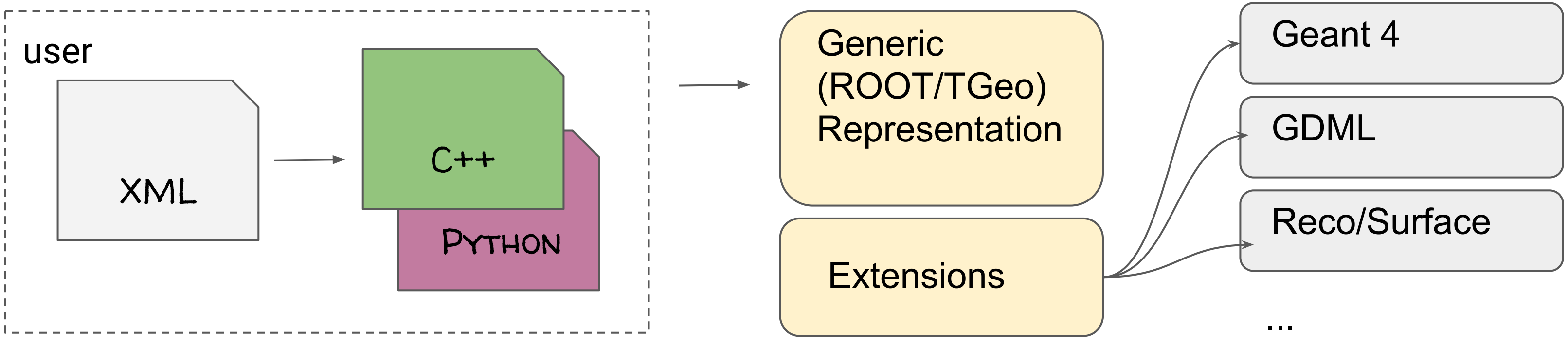
GEMC -- GEant4 MC

- Builds a Geant 4 geometry model from many possible sources
  - GDML, CAD files, ASCII format, perl...
- Used at JLAB for CLAS12 and EIC studies

I’ve only scraped the surface of the documentation / presentations. Mention for completeness.

DD4Hep -- Detector Description for HEP

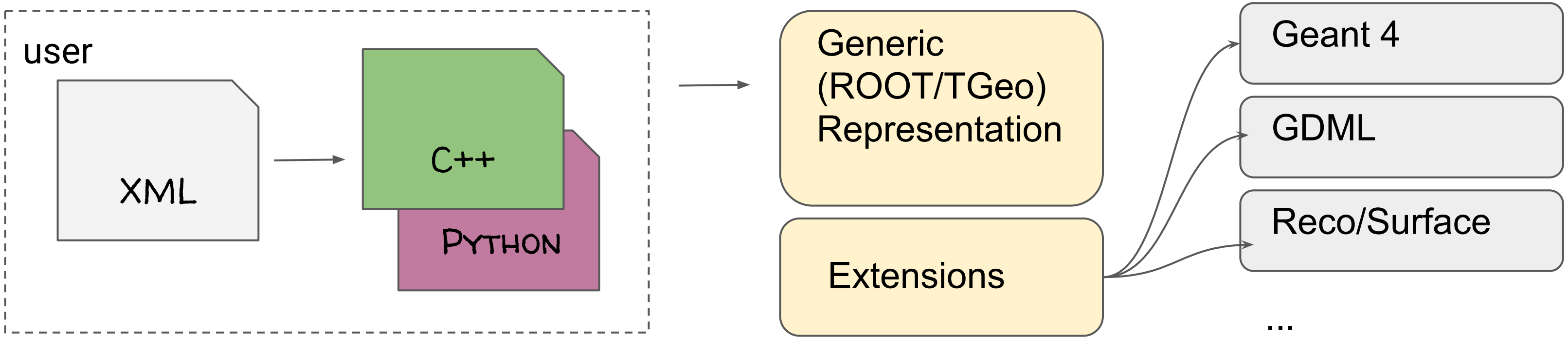
Emerged from LHCb and International Linear Collider efforts. ATLAS and CMS frameworks similar in approach.



- Compact XML description input into C++ (python) constructors
  - Adds the concept of “repetition”, “layers”, and “envelopes” to geometry model
- Persistent in-memory geometry model based on ROOT/TGeo plus extensions
- Large toolkit for detector modeling
  - Includes bindings for G4 hit scoring / ships with a G4 application
  - Simplified surface representation for reco, produced from full model
  - Alignment support and event display under development
  - CAD files can be used as input

DD4Hep -- Detector Description for HEP

Emerged from LHCb and International Linear Collider efforts. ATLAS and CMS frameworks similar in approach.



Pros

- Actively maintained and developed, widely used
- Simple detectors can be implemented in just the XML
- Syntax encourages good organization of the geometry model (hierarchy)
- Supports reconstruction w/ surface model and alignment

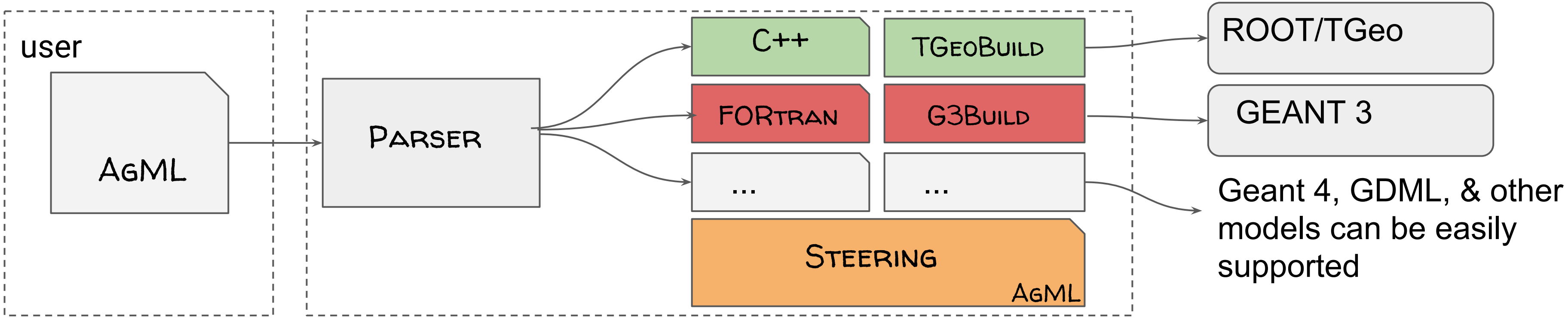
Cons

- Users need to learn the XML syntax and C++/python
- Geometry cannot be expressed in XML alone
  - Developers *must* keep the XML and constructors synchronized… requires discipline
- No branching in XML, so detector versioning requires many XML files and/or logic in ctors

- CAD model import useful for rapidly integrating new detector models and *creating cpu bottlenecks* during R&D

AgML -- Abstract Geometry Modeling Language

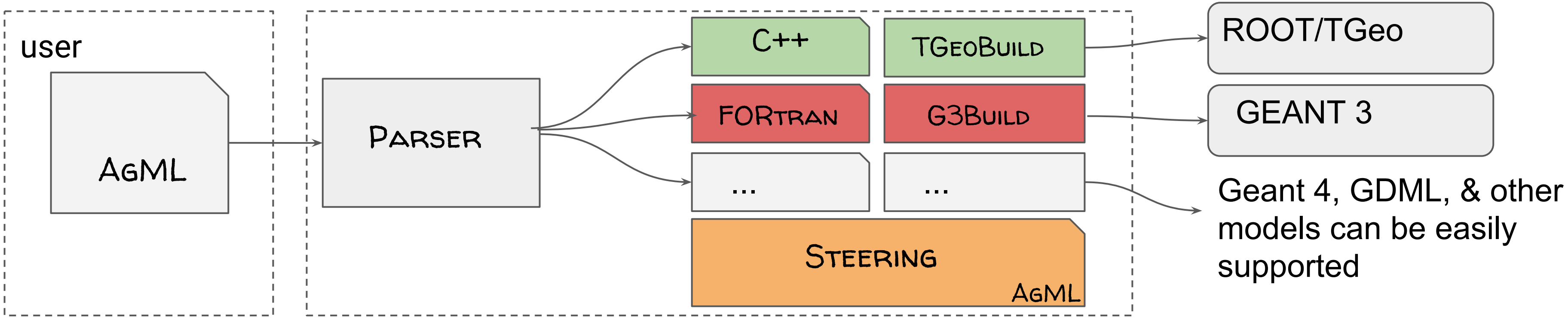
Developed from the Advanced GEANT Interface (Agi) used in early ATLAS development and STAR production until 2011.



- AgML (XML) description is the single and complete source of geometry information
  - Complete language, supporting loops, variables, constants, data structures, branching, hits, and construction
  - AgML sources are parsed and translated into compilable code, linked into shared libraries
  - ROOT and/or G3 geometry created at run time from the shared libraries
- Simulations use G3 geometry
- Reconstruction code takes ROOT/TGeo as input, and translates/simplifies into native tracker format
- Support for misalignment in development

AgML -- Abstract Geometry Modeling Language

Developed from the Advanced GEANT Interface (Agi) used in early ATLAS development and STAR production until 2011.



Pros

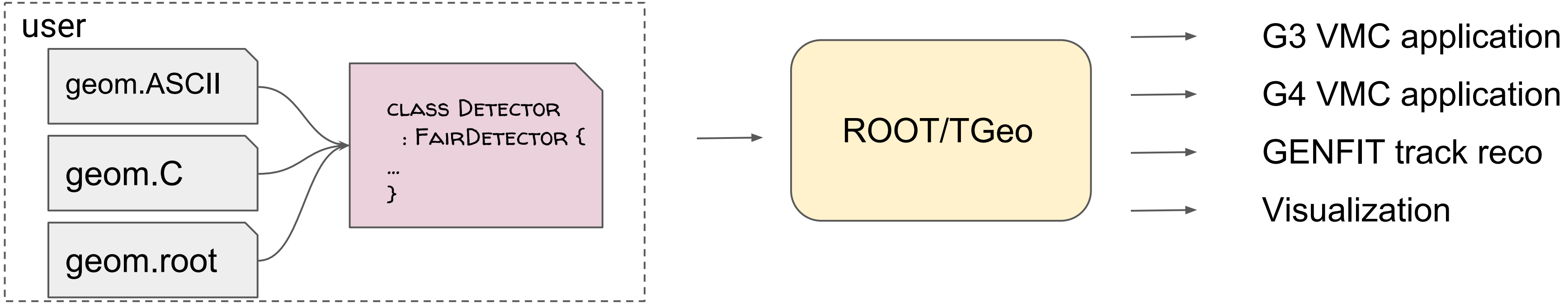
- Actively developed and maintained
- Demonstrated track record in production environment
- The full geometry model, including versioning of the STAR detector from run-to-run, is defined in AgML
- Geometry versions fixed at compile time -- tagged and released with our software libraries -- ensuring consistency across 17 years and 83 distinct versions
- Language syntax captures the problem domain: Materials, shapes defined within volumes, which are also responsible for creation / placement of daughters

Cons

- Changes to the geometry model require compilation
- Lacks support for input of other formats (*however…*)

FairROOT

Common development coming out of GSI for Fair experiments, and widely used beyond. Very similar to the Alice approach, and collaborating with them. Used in the eRHIC / EIC studies.



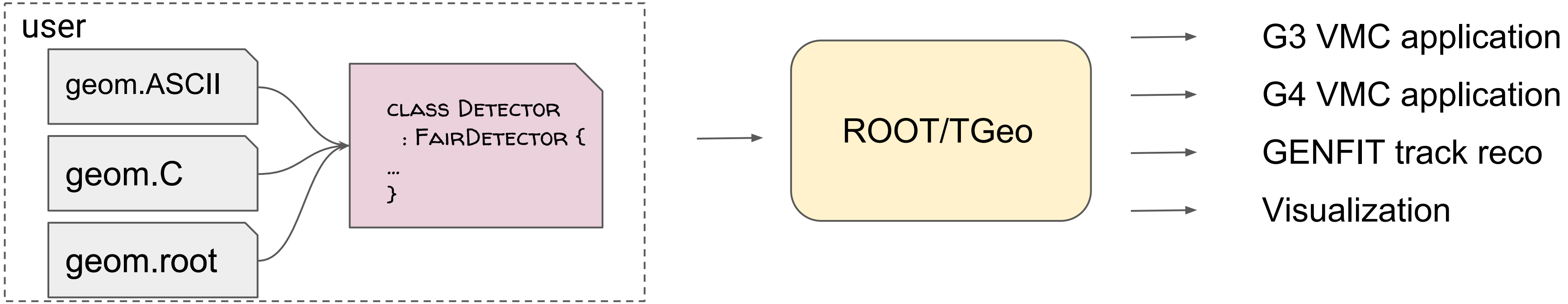
Fair geometry is part of a comprehensive framework supporting simulation, reconstruction and analysis. User defines detectors and modules inheriting from FairGeo base classes. Multiple paths to defining geometry

- Read in from ASCII text format (HADES), ROOT macro or ROOT file containing the geometry
- Implement C++ classes inheriting from FairGeo base classes (volume, material, etc…)

Simulation, reconstruction, etc… support by ROOT/TGeo model

FairROOT

Common development coming out of GSI for Fair experiments, and widely used beyond. Very similar to the Alice approach, and collaborating with them. Used in the eRHIC / EIC studies.



Pros

- Actively maintained and developed, widely used
- Flexible, able to import multiple formats. Simple detectors can be implemented in ASCII, more complicated w/ ROOT macros or C++.
- Part of a tightly integrated system

Cons

- No abstraction of the detector description
- Detector description does not encourage good design
- Diverse input files complicates the task of the maintainers, and…
- Geometry model *can* be split between the input file and the detector class.

Concluding Remarks (*remember the disclaimer*)

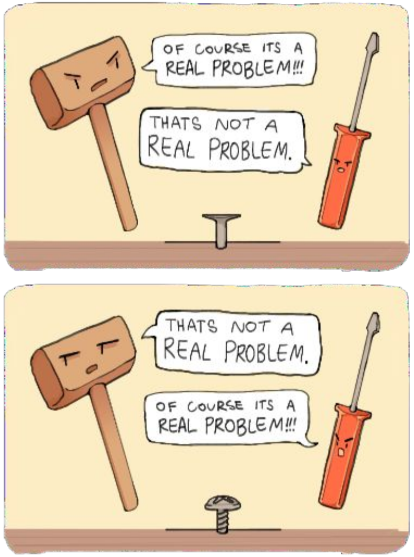
- Different frameworks approach *usability* issues differently. Support for multiple input formats and/or CAD, or a single *feature-enriched* detector description language
  - My take: *experiment* benefits from large user base using a common description -- larger pool of developers to maintain, extend, debug. Also simplifies support of retired detector models.

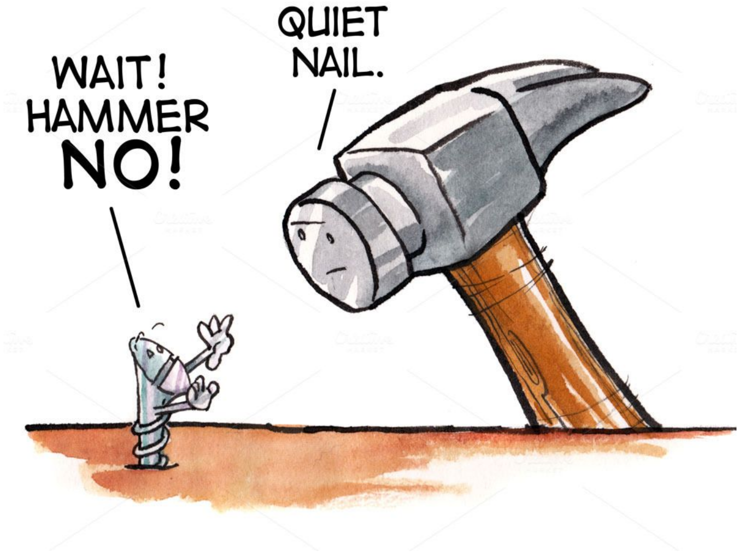
- Each framework has the capability leverage new technologies which become available
  - Abstraction layer can be changed to adapt to the underlying geometry library
  - My take: Agi/AgML has *made* such a transition without disrupting support for data production .
- R&D greatly benefits from and production absolutely requires a reproducible *versioning* scheme. This requires disciplined procedures on the part of the code maintainers.
  - My take: this can *and should* be supported by the geometry framework. DD4hep provides some support. AgML provides a workable solution.

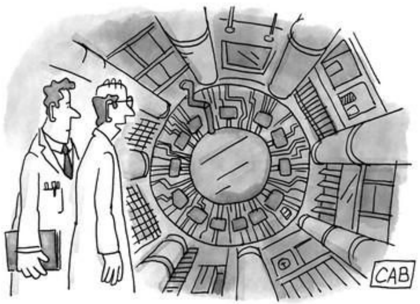
Concluding Remarks (*remember the disclaimer*)

There are several detector description and geometry frameworks available, each capable of supporting a detector R&D program followed by experimental data taking and production.

Coalescing around one of these approaches would have the benefit of building a community of developers who could work efficiently, together, to advance the scientific vision of an EIC detector.







"Once you have a collider, every problem starts to look like a particle."