Arve: An Object-oriented Analysis Framework Design for ATLAS

T. H. Burnett [1]

Physics Department, Box 351560, University of Washington, Seattle WA 98195

I discuss the design of an object-oriented framework intended to facilitate development of detector reconstruction and physics analysis code. To satisfy this, we have developed a modular and flexible scheme. The modularity is primarily with regard to how easy it is to create applications containing only the subdetectors relevant to a developer. The control model follows this scheme, in that subdetectors are responsible to defining their commands, via a common interface. Rapid testing and development is made much easier with two other capabilities: the ability to generate and simulate simple events, perhaps single particles; and a simple implementation of the ATLAS event display model that allows flexible display of any geometric object. A display is shown of particle tracks, and responses in, the ATLAS LHC detector.

Key words: analysis, application programming, C++, class libraries, object-oriented methods; event display;

1 Introduction

Physicists interested in developing reconstruction algorithms for LHC experiments are currently in a quandary. Both major experiments have declared their intention to switch from the current Fortran-based environment to using C++, and object-based design techniques. But over-all designs do not yet exist for essential components like geometry, particles and their physics, and visualisation. Thus, to test design ideas with real code, they must invent *ad hoc* frameworks.

[1] On leave, 1996-7 at Stanford University; partially supported by the National Science Foundation

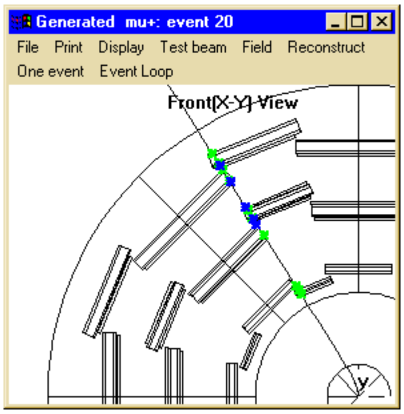

Fig. 1. Event display, showing detector, a muon track, and response

The Arve [2] framework grew out of the Gismo[1] package, but has been adapted to isolate a developer from dependence on ad hoc solutions; rather, to provide a consistent set of classes in each of these areas.

It is further intended to adapt the Arve solutions to match decisions made by developers in the common areas. This has already happened for the event display, for example. There is clearly a risk that this could impose some burdens on developers of subdetector code. Also, it has to be adapted to be consistent with the Atlas Software Process.[3]

An example is shown in Figure 1. It is a conventional window, containing a title bar, menu bar, and graphics display area. A PC running Windows NT was used to generate the display, but the graphical user interface definition allows other implementations: the appearance of the X-Motif version is identical. The display is of a projective r-phi view of the ATLAS detector, with some of the barrel muon chambers, and the solenoid shown. Also shown is a single 20-GeV muon, (under control of an object that responds to commands from the "Test Beam" menu) which bends in the central field, then intersects five muon chambers. The marks where the track intersects the chambers are generated independently, by objects that record hits.

2 Domains

The elements in the display of Figure 1 all have associated objects. These are allocated to domains, a partitioning of the problem into distinct pieces to aid in the design, and allocation of effort. An initial list of top level domains associated with the framework is:

[2] the name could be an acronym for ATLAS Reconstruction and Visualisation Environment, but I prefer the actual inspiration for the name, the swiftly-moving, but dirty, river that flows from the Alpine glaciers to the meet the Rhone in Geneva.

- *Subsystems:* Separation of different parts of the project, but allowing dependencies;
- *Control:* The user interface for control of the application; the command loop;
- *Display:* 3-D display of detector elements;
- *Simulation:* Generation of events.

I discuss each of these in subsequent Sections.

3 Subsystems

We distinguish logical and physical subsystems. The Atlas detector naturally divides into different physical subsystems, for example central tracking, muon barrel, etc. Each is geometrically distinct, and mostly autonomous. However, all must respond to requests to accumulate data (either from an event source, or generated on the fly), and especially to reconstruct. During reconstruction, moreover, several systems may need to cooperate. And here is a need for subsystems not connected with specific detectors, namely tracking that combines information from different systems.

4 Control and Servers

All programs designed to be interactive require a means for the user to be aware of, and specify options. This need is fulfilled by the class Menu. Also, all such programs require services, in the form of print and perhaps graphics. We have encapsulated this idea, using the classes Server and Services. Our model of a program is that it is composed of a list of Servers, which need to be instantiated, then configured for a particular application. Then the program is "run", meaning that a list of commands is processed, or user input begins. Before this happens, some servers may require notification to finish setup.

5 Display

The importance of good 3-D graphics to allow the user and developer to understand relationships among complex 3-D entities cannot be over emphasized. Event displays are universally used to allow visualization of physics processes. However, it has never been easy for developers to observe the 3-D entities that they deal with in the early, debugging stages.

Arve address this need by integrating graphical display with the development environment and the control system, and requiring no external display packages. This presumes a situation where the developer has a temporary need to see what is going on with his/her new objects, but has no desire to contribute this particular visualization to the formal physics-oriented event display package. (But it should also fit into the event display framework, to allow an eventual migration.) In other words, such code would be similar to that written for debug printout, only intended for the development process. Such visualization would not be useful without a context: the rest of the event, the detector, other detector responses. And, of course, it must be easy to control.

Recently the interface has been made consistent with the proposed design for the Atlas Event Display; now developers can generate displays with confidence that such code will continue to be applicable.

6 Simulation

A key part of the framework for reconstruction is the possibility to generate simple events on the fly. Thus we identify simulation as a domain, and expect that the full functionality of the standard simulation package can be integrated into the framework. This will soon be the Geant4[2] package: but until it is available, we use parts of an older system, Gismo[1].

References

[1] T. H. Burnett, GISMO: An Object-oriented Approach to Particle Transport and Detector Modelling, in *Proceedings of the Internation Conference of Monte Carlo Simulation in High Energy and Nuclear Physics*, Tallahassee, Florida 22-26 February 1993; GISMO: an Object Oriented Program for High-energy Physics Event Simulation and Reconstruction, W.B. Atwood, T. Burnett, R. Cailliau, D.R. Myers, K.M. Storr. *Int. J. Mod. Phys.* C3 (1992) 459-478.

[2] S. Giani, GEANT4: a world-wide collaboration to build Object Oriented HEP simulation software, *this conference.*

[3] S. Fisher, The ATLAS Software Process, *this conference.*