DØ's Experience with Object Oriented Programming

K. W. Merritt [a], D. L. Adams [b], J. Bantly [c], J. F. Bartlett [a], D. Chakraborty [d], B. J. May [e], L. Michelotti [a], L. Paterno [a], M. Paterno [f], K. Streets [g]

[a] *Fermilab, P. O. Box 500, Batavia, IL 60510, USA*

[b] *Rice University, Houston, TX 77251 USA*

[c] *Brown University, Providence, RI 02912 USA*

[d] *State University of New York, Stony Brook, NY 11794 USA*

[e] *Northwestern University, Evanston, IL 60208 USA*

[f] *University of Rochester, Rochester, NY 14627 USA*

[g] *New York University, New York, NY 10003 USA*

The DØ collaboration at the Fermilab Tevatron is in the process of building its offline reconstruction system for Run II in C++, using principles of object-oriented analysis and design to aim at a system with greater maintainability and robustness. This paper presents the history and status of this project.

Key words: C++; object-oriented methods; large systems

1 Introduction

The DØ experiment, like all large HEP detectors, needs an extensive suite of customized software, performing a variety of tasks associated with operating the detector, collecting a large data set, and reconstructing and analyzing the data. This software must be (reasonably) platform- and operating-system-independent, and must be maintained over at least one major data collection/analysis cycle of the experiment (5-10 years). In this paper, we discuss the offline reconstruction part of our software, and in particular the choice we have made for our next running cycle (Run II of the Fermilab Tevatron) to redesign this system using techniques of object-oriented analysis and design (OOAD) and to implement it in C++. This decision was recommended to the collaboration in November, 1995. Serious design for reconstruction algorithms and program infrastructure began in 1996 and is continuing, with the goal of a complete C++ reconstruction program in 1998.

2 Motivation for the OOAD/C++ Decision at DØ

The DØ detector installed for Run I of the Tevatron consisted of 3 major subsystems, including a non-magnetic central tracking volume. For Run II, a solenoid is being installed and the central tracking detectors are being completely replaced. There are also changes to the muon system and to the calorimeter electronics. The large degree of change not only to the physical detection devices but also to the philosophy of the particle identification and calibration implies that the majority of DØ reconstruction code must be rewritten for Run II. We see this as an opportunity to design a new system for greater maintainability while retaining the robustness which made for a successful Run I analysis.

Software design techniques (SASD, in particular) were applied with some success to the Run I online software for DØ, but did not prove a significant aid to the full reconstruction program, especially in the long term. We feel that OOAD techniques hold more promise for a really large software system, for several reasons. (1) In object-oriented programming, the design deals with entities (classes, methods) that translate directly into code. This makes evaluating the design, implementing it, and keeping the code and the design mutually consistent much easier than in some earlier systems. Indeed, it is now possible to use tools which automate the passage from design to code (although we have not yet found any tool to be completely satisfactory in this regard). (2) The concepts of encapsulation and of programming to an interface are extremely useful in designing for the goal of separate testability of the various pieces of the system. The implementation of these concepts is much easier in object-oriented languages than in FORTRAN. (3) Many maintainability problems with the Run I reconstruction arose from the awkwardness of trying to achieve memory management within standard Fortran. This function is provided with the language when we move to C++. It can be said that we trade one set of problems for another, in that now the programs are subject to memory leaks; however, the existence of tools to catch such leaks gives us some confidence that this will be a tractable problem.

C++ was chosen as the implementation language because it is the most widespread of the object-oriented languages, and the choice of several other HEP experiments, thus allowing the maximum flexibility in platform choice and the maximum benefit from shared experience and code.

3 Learning to Design HEP Reconstruction with OOAD

Designing an HEP reconstruction program with OOAD tools, in C++, is still not a routine procedure. Although several experiments have now taken or agreed to take this path, there is not enough public, well documented or just generally familiar code to allow programming by example. The designers must generally begin with a class design, developed with the aid of a formula from one of the various OO methodologies. Ideally, the design can be evaluated to see whether it meets the requirements of the project, *before* implementation begins. (This is one of the advantages of the OOAD approach.) Further changes to the design may still be necessary as implementation and testing proceed.

DØ's earliest effort at an OOAD design for a major reconstruction piece was to design a C++ version of the jet-finding code from Run I. This code used the tranverse energies of calorimeter cells as input and ran several types of clustering algorithms to find jets. A complete design from this effort exists for a prototype reconstruction program, classes to allow generalized parameter input, and classes for jet reconstruction using a variety of different clustering strategies: all those used in Run I, plus a purely geometrical algorithm which was included to test the flexibility of the design.

The experience of this small group in making this design confirmed, in the main, lessons and advice generally available in OOAD circles. However, it is worth making a couple of points from the perspective of HEP programmers. (1) Formal and intensive training in OOAD techniques and principles was found to be very useful in the specific area of evaluating the designs. We had an existing design at the time when most members of the group took a one-week course in OOAD; the formal training and especially the exposure to message trace diagrams led directly to rapid progress and an improved design. Since many HEP programmers have for years used the "program by example" method, and since such examples are rare for C++ HEP code, it is worth stressing that reeducation is not to be scorned and does have beneficial effects. (2) It should also be emphasized that the OOAD method both allows and requires a more iterative process than most HEP FORTRAN programmers are accustomed to. The experience of our group confirmed the general wisdom that it is best to begin with a top-down design generated from a well-thought requirements document, to begin implementation as soon as the design is reasonably advanced, and to be prepared to let the lessons drawn from the implementation trigger a redesign: a complete redesign, if necessary. (3) The design of the classes which contain the event data and the characteristics of the mechanism used to make these classes persistent are crucial for the testability and practicality of the system. It is in this area that a set of tradeoffs particularly unfamiliar to the FORTRAN programmer will need to be made, and here that the advantages of the OO approach will be gained or lost by our choices. Fortunately, this problem is rather generic, and, once an appropriate solution is found, it is likely to be a useful model for most HEP experiments.

4 Making an OOAD/C++ Development Environment

A project such as this one is greatly benefitted by a good development environment; this would be easiest to achieve in that ideal world where only one platform need be supported for both development and production. HEP experiments never live in that ideal world, and our experience is that the biggest obstacle to building our new system for Run II will be defining and acquiring an adequate development, debugging, and configuration management capability on all the supported platforms. Our development projects so far have been carried out on SGI IRIX, IBM AIX, and Windows NT. The major annoyance is the lack of a uniform set of features available reliably in C++ compilers across all these platforms. Programming to the common working subset on these platforms would seem to be extremely restrictive, and in particular makes the use of templates and of the STL containers quite difficult. Since the situation with respect to compiler features is changing rapidly, our current policy is to continue to use templates and container classes with templates, even though these are the source of most difficulties in porting code.

5 Conclusions

We find that the general problems of HEP reconstruction code are indeed amenable to the OOAD approach. The design tools available are up to the task of documentation of a design and to keeping the design current, but not to the complete automation of the translation from design to code. The lack of a standard implementation of C++ across the major platforms in use for HEP is the largest obstacle to building reconstruction systems quickly. We expect that the benefits of being able to share design patterns and solutions through the CLHEP [1] and FPCLTF [2] efforts will help make the Run II reconstruction in C++ a feasible and worthwhile project.

References

[1] http://wwwcn1.cern.ch/asd/lhc++/clhep/index.html

[2] M. Fischler *et al.*, "The Fermilab Physics Class Library", these proceedings.