An Object-Oriented Approach to Vertex Fitting

Winfried Mitaroff and Peter Stamfest

Institut für Hochenergiephysik der
Österreichischen Akademie der Wissenschaften
Nikolsdorfer Gasse 18, A-1050 Wien, Austria

An existing vertex fitting package is currently being reimplemented using object-oriented design principles in the C++ programming language. The aim of the new library (working name VERA or VTL) is to provide a common interface for different vertex fitting methods, and to be easily extensible using the C++ template mechanism. Currently a Kalman Filter approach is under development, but the framework should allow for a transparent implementation of robust techniques like the M-Estimator or of Neural Networks without changing the interfaces.

Key words: C++; templates; object oriented algorithms; vertex fitting

1 Introduction

The analysis of data from a high-energy collider like LEP, HERA, BaBar or LHC requires sophisticated statistical methods in order to exploit the precision of modern complex detectors. The high spatial resolution of semiconductor vertex detectors has made it possible to identify and fit secondary vertices and their associated tracks from decays as short-lived as b hadrons.

For this task, a package in Fortran-77 had been developed by one author for the DELPHI detector at LEP, based on the robust M-estimator. Its superiority over conventional estimation methods (least squares or Kalman filter) has been shown with simulated and real data in an earlier study [1].

The present project is intended to serve as a testbed for the application of object-oriented (OO) design and programming techniques in the field of high energy physics, and primarily aims at demonstrating (again) the usability of OO design principles in that field.

Preprint submitted to Elsevier Preprint *25 April 1997*

The main problem to be tackled is to identify and to define interfaces between user and library objects and classes, without putting unnecessary constraints on the design of user classes.

2 Templates vs. inheritance

At present, we favour an approach which makes use of template algorithms in the style of the Standard Template Library (STL) [2,3], thus offering the greatest flexibility to the usage of the library.

Moreover, the user of a template algorithm is only obliged to adhere to the requirements of the algorithm itself, which ideally should only require a small number of member functions and operators to be defined for the classes to work with. Because the requirements are checked during compilation and are only based on the naming of member functions, no inheritance requirements have to be met.

As an alternative to the template mechanism, the use of inheritance and polymorphism using virtual member functions was suggested. But that approach would force a library user to inherit from some library-specific superclass, which might require considerable rewriting of code, which in turn could imply to change the design of whole class lattices.

This problem gets even worse if different algorithms with different requirements are defined in the library, but the user only needs a subset of those algorithms. Having to adhere to seemingly unnecessary requirements would not increase the acceptance of the library.

The use of virtual functions could also cause a performance penalty as compared with a template-based implementation due to the necessary runtime lookup of a virtual member function.

3 Current developments

The library currently under development tries to conform to the STL requirements itself, so it is possible to use all the algorithms defined in the library together with STL containers and algorithms. At the time of writing only the first steps of the actual implementation have been completed. This is partly due to seemingly minor problems, like allowing for the use of different mathematical libraries or the still evolving C++ language definition, and the procurement of a working STL.

A more serious problem we have to face is due to existing C++ compilers. Standard-conforming compilers are still rare, particularly when required to support templates. We try to get around these problems without compromising the overall design, which is a non-trivial job in itself. Currently we are using the GNU C++ compiler which still has some major problems with templates, and especially with those features we intend to rely on, like using nested types. But such features are needed in order to obtain a more loosely coupled system of template classes.

Figure 1 shows the very top-level design of the library in a slightly extended form of the Booch notation [4]. It shows the actual algorithms as class-utilities interacting only with an interface class which in turn may be inherited from any class or may use objects of any class, in whatever way it likes.

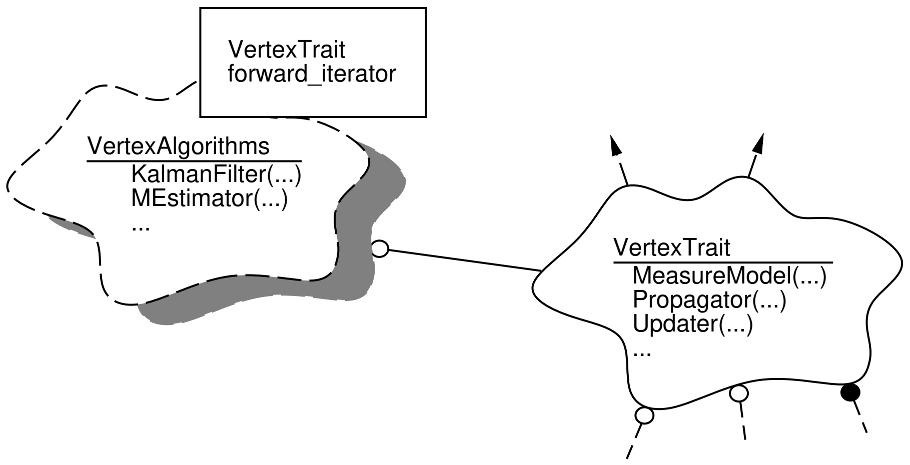

Fig. 1. Top-level structure of VERA/VTL

We try to implement algorithms that work on containers which bundle track data, and using helper classes to implement the communication between the data and the algorithm. This scheme is very flexible and should allow for the use of almost any track class together with the library.

It is even possible to use some of the algorithms for other tasks than vertex fitting: the Kalman filter algorithm may also be used for track fitting, pending the implementation of a suitable interface class, partly playing the role of a trait class.

We intend to finally deliver a field-proven vertex fitting library to be used as a drop-in replacement or an add-on extension for existing and future event reconstruction programs in an OO environment.

References

[1] Frühwirth, Kubinec, Mitaroff, Regler: Comp.Phys.Comm. 96 (1996) 189.

[2] STL, C++ draft standard, ANSI X3J16/95-0087, http://www.cygnus.com/misc/wp

[3] David R. Musser, *STL tutorial and reference guide: C++ programming with the standard template library*, (Addison-Wesley, 1996)

[4] Grady Booch, *Object-Oriented Analysis and Design with Applications, Second Edition* (Benjamin/Cummings, 1994).