A Beamline Matching Application based on Open Source Software

J.-F. Ostiguy*

Beam Physics Department, Fermi National Accelerator Laboratory, Batavia, IL 60510

Abstract. An interactive Beamline Matching application has been developed using beamline and automatic differentiation class libraries. Various freely available components were used; in particular, the user interface is based on FLTK, a C++ toolkit distributed under the terms of the GNU Public License (GPL). The result is an application that compiles without modifications under both X-Windows and Win32 and offers the same look and feel under both operating environments. In this paper, we discuss some of the practical issues that were confronted and the choices that were made. In particular, we discuss object-based event propagation mechanisms, multithreading, language mixing and persistence.

INTRODUCTION

Up until just a few years ago, writing software for scientific applications required the programmer to implement functionality that had little to do with his objectives. Although commercial libraries were available, high costs and licensing hassles made it more practical to reinvent user interfaces, plot widgets, etc. By dramatically improving free software localization and distribution mechanisms, the internet promotes code reuse.

In this paper, we describe an interactive lattice design application assembled with various freely available software components. The objective was not to compete with commercial products, but rather to provide an application that can be modified and adapted to meet our specialized needs. There are currently a limited number of publicly available interactive applications to perform beamline design. Popular lattice design programs like TRANSPORT and MAD have extensive capabilities, but were developed to be run in a batch-oriented environment.

The design goals were the following: (1) given a description of a beamline, allow a user to specify all aspects of a matching problem interactively (2) provide graphical feedback and allow the user to dynamically interrupt a nonlinear iteration and edit the state of variables and constraints (3) make customization as easy as possible. Although this is still a work in progress, basic features have been implemented and are fully functional; the resulting application is called BLIMP (BeamLine Interactive Matching Program). A specialized version, produced to tune the Fermilab Recycler Ring phase trombone, will be described.

* Work supported by DOE Contract DE-AC02-76CH0-3000

THE MATCHING PROBLEM

The matching problem is of fundamental importance for lattice designers. It can be simply stated as follows: given a beamline and a set of lattice functions specified at one extremity, determine the strength and/or longitudinal position of beamline elements necessary for the lattice functions to assume certain specified values at one or more distinct locations.

In most situations of practical importance, horizontal and vertical motion are decoupled and a beamline is to first order, completely characterized by a set of ten quantities: $\beta_{x,y}$, $\alpha_{x,y}$, $\mu_{x,y}$, $\eta_{x,y}$ and $\eta'_{x,y}$, where β and α are the familiar Courant-Snyder lattice functions, μ is the phase advance and η and η' are respectively the dispersion and its derivative with respect to the longitudinal coordinate.

CODE STRUCTURE

BLIMP is written in ANSI standard C++ and takes advantage of the facilities offered by the Standard Template Library. Variables are defined independently of basic beamline elements and can in principle involve arbitrary linear combinations of element strengths. The user can dynamically define both local and global constraints. Typically, local constraints involve equalities while global constraints involve inequalities (e.g. β function smaller than a prescribed maximum). Figure 1, is a screen shot of the user interface. BLIMP has been put together by using both internally developed and freely available software components which are now briefly described.

`MXYZTPLK/Beamline` *Class Libraries*

MXYZTPLK and `Beamline` are class libraries authored by Leo Michelotti (1, 2) at Fermilab. Although considered stable for a few years already, they continue to evolve and are maintained to keep up with the

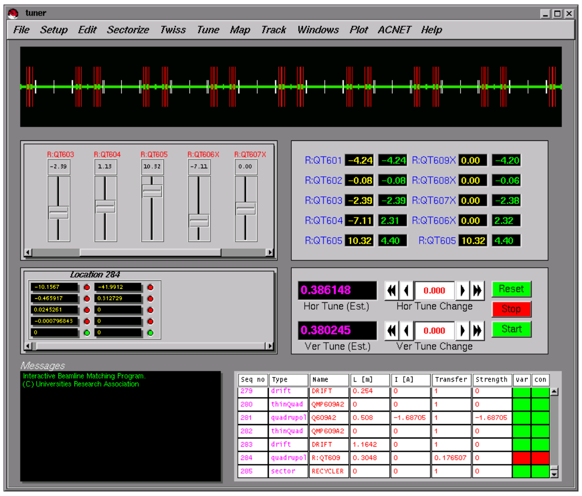

FIGURE 1. *The BLIMP user interface. The top window is a display of the beamline layout. Sliding cursors are displayed for each user-defined variable. Similarly, a custom widget is created for each local constraint.*

latest developments in the C++ language. MXYZTPLK is a stand-alone set of C++ classes for performing automatic differentiation and differential algebra. The `Beamline` class library –built on top of MXYZTPLK– is a rich set of classes supporting lattice related calculations. `Beamlines` are represented by doubly linked lists whose nodes can either point to other beamlines or to basic elements. `Beamlines` can be edited, concatenated, cloned, flattened (i.e. no hierarchical structure). Most quantities of interest to accelerator physicists can be computed and both field and alignment errors can be included if necessary. Maps of arbitrary order can in principle be computed to machine precision in either four-dimensional (i.e., transverse) or full six-dimensional phase space.

Nonlinear Optimizer

Numerical nonlinear optimization problems can be classified according to (1) whether or not the objective function is expressible as a continuous, differentiable function (2) the nature of the external constraints that need to be enforced, if any. For matching problems, the objective function is usually a differentiable function of the elements' strengths and positions. In that case, Newton method has the advantage of quadratic convergence if the extremum is sufficiently close. In practice, it is expensive to compute a Hessian matrix since it involves second order derivatives. To avoid this, a standard strategy is the Davidon-Fletcher-Powell (DFP) algorithm (3) which progressively constructs the inverse of the Hessian matrix at each step of an iteration involving a sequence of one dimensional conjugate gradient searches.

BLIMP uses the DFP method as implemented in the MINUIT library from CERN (6), a good general purpose optimization library freely available for research institutions. Unfortunately MINUIT suffers from various limitations associated with its Fortran heritage. Among the most problematic issues: Fortran I/O cannot be mixed with the C/C++ I/O in a portable way; the objective function must be passed to the library as a static function. The optimization code is encapsulated into an `Optimizer` class; this should allow an alternative to MINUIT to be substituted with minimal side effects. A full C++ implementation would also allow the use of functor objects to completely decouple the optimizer from the rest of the code.

Graphical User Interface

The choice of a user interface toolkit has been driven by two requirements: (1) object orientation and (2) portability between various flavors of UNIX and Windows NT. The Fast Light Toolkit (FLTK) (7) satisfies both requirements and is available under the terms of the GNU Public Library License. FLTK also provides support for OpenGL (or MESA, a free compatible alternative) and BLIMP takes advantage of the facilities offered by OpenGL to efficiently display the beamline at different scales.

Events

In addition to the primitive event propagation mechanism provided by FLTK for GUI related events, BLIMP uses a generic event propagation scheme. It relies on relatively recent, but nevertheless, standard features of ANSI C++ templates. This allows the code to be structured as a collection of independent "components" in a completely generic and type-safe manner. Events can be arbitrarily complex objects and generic lists are created for each distinct published event type. Subscribers are added to each list by interested objects. To propagate an event, a publisher object need only to instantiate it and invoke its virtual `propagate` method. In practice, to publish and/or subscribe to a certain event type, a class need only to be derived from a `Publisher` or `Subscriber` base class (or both).

At this point, a few comments are in order. In certain commercial frameworks such as MFC, event declaration and propagation mechanisms are enforced by the framework. MFC does not rely on templates, but rather on a compex collection of macros. Some commercial compilers, like Borland C++ Builder for example, provide language level support forevent propagation by introducing special additional keywords for that purpose. An intermediate approach, used by Qt (a popular C++ framework) is to extend the C++ language and use a special preprocessor to convert the original source code into standard ANSI C++.

A number of similar free generic template-based “call-back” libraries can be found on the internet (4).

Persistence

A valuable feature for any interactive application is the ability to save its current state. This state is completely defined by a certain number of objects and their relation to each other, as defined by pointers and references. Unfortunately, simply saving objects in binary form is not sufficient to save the state of an application, since pointers and references are process specific. To correctly restore the state of an application, it is necessary to keep track of relations between objects and to fix all memory references a posteriori. Although it is a straightforward matter to fix explicitly declared pointers and references, correctly restoring virtual functions presents a special challenge, since virtual function pointers are not directly accessible. Fortunately ANSI C++ provides a solution. In a nutshell, the technique consists in reading a raw binary object into a block of (allocated) memory of correct size, and subsequently invoke the operator `new` using the ANSI C++ placement syntax. This syntax allows one to place an object in pre-allocated memory. Provided the constructor does not perform any explicit initialization, all data fields are left untouched; however, *all virtual pointers are correctly initialized*. Persistent versions of the MXYZTPLK and `Beamline` libraries have been developed based on the above described scheme.

Multithreading

Although the FLTK code is not reentrant and therefore does not directly support threads, an application can still take advantage of multithreading provided all GUI activity remains confined to a single thread. Since the `Optimizer` consumes a fair amount of CPU it is useful to dedicate a separate thread to it. An added benefit of using separate threads is that the Optimizer no longer needs to contain any specific GUI code. A threaded version of BLIMP is currently under consideration. One difficulty arises from the fact that the UNIX and WIN32 thread models are somewhat different.

APPLICATIONS

We now describe two applications that motivated the development of BLIMP.

Phase Trombone

The Fermilab Recycler ring is a new machine for antiproton accumulation and recycling which has the distinction of being the first machine to make large scale utilization of permanent magnet technology. The machine operates at a fixed energy of 8 GeV with a lattice based on fixed-field combined function magnets. The tune of the machine is adjusted (± 0.5) by varying nine electromagnetic quadrupoles grouped in five symmetric families within a region where $\eta_{x,y} = \eta'_{x,y} = 0$. Four hard constraints must be met, i.e. at the symmetry point $\alpha_{x,y} = 0$ and the two phase advances set to the desired values; an additional softer requirement is to prevent the beta functions from exceeding a maximum value.

Low Beta Insertion

In a low-beta insertion, the objective is to use a pair of quadrupole doublets or triplets to focus counter-circulating beams into a very small size interaction region. In general, the insertion has to match the lattice functions of the ring at both extremities; the phase advance is unconstrained. At the interaction point, $\beta_{x,y}$ must assume specified values and the beam envelope must go through a minimum i.e. $\alpha'_{x,y} = 0$. It is also often required for the dispersion to be as small as possible and one usually demands $\eta_{x,y} = 0$. Constraining η' may also be desirable.

Low beta insertions are notoriously nonlinear. Without experience, it is difficult for a novice to find a satisfactory solution and interactivity is certainly no substitute for experience. However, the ability to quickly experiment with different strategies and stop the iterations dynamically can be a significant advantage.

CONCLUSION

BLIMP is still a work in progress, although it is certainly already useful as it stands. The current priority is to make the application multithreaded. Also under consideration is a port from FLTK to Qt, a commercial framework much more mature than FLTK, which has only recently (Sept 2000) been released under the terms of the GPL license.

REFERENCES

1. L. Michelotti, “MXYZPLTK Version 3.1 User’s Guide: A C++ Library for Differential Algebra”, Fermilab Publication FN-535-REV, October 1995
2. L. Michelotti, “MXYZPLTK and `Beamline`: C++ Objects for Beam Physics”, Advanced Beam Dynamics Workshop on Effects of Errors in Accelerators, their Diagnosis and Correction, AIP Conf. Proceedings No 225, 1992
3. R. Fletcher and M.J.D Powell, *“A rapidly convergent Descent Method for Minimization”*, The Computer Journal, **6**, 163-168, 1963
4. For example, see `http://libsigc.sourceforge.net`.
5. J. Hesse, *“EZSave for C++”*, C++ Report 12(2): 21-26, 40-41, Feb. 2000
6. F. James, “MINUIT Minimization Package Reference Manual Version 94.1”, CERN Program Library D506, Computing and Networks Division CERN Geneva, Switzerland
7. Information about FLTK is available at the following URL: `www.fltk.org`
8. Fermilab Recycler Ring Technical Design Report, Fermilab Publication TM-1936, July 1995