CHEF: AN INTERACTIVE PROGRAM FOR ACCELERATOR OPTICS*

Leo Michelotti and Jean-François Ostiguy
Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

Abstract

We report the current status and our plans for the completion of CHEF, an interactive application for performing optics calculations in accelerator physics. CHEF uses high level graphical user interfaces to facilitate the exploitation of lower level tools incorporated into a hierarchy of C++ class libraries, making them usable by those not familiar with C++ programming.

INTRODUCTION

In January, 2004, Fermilab's Accelerator Divison formalized two distinct, but related objectives as a project: (1) develop a framework to access easily information needed for accelerator physics calculations on Fermilab machines, and (2) develop user-friendly software for optics calculations in beam line design and analysis. A component of the first is the Accelerator Division's Lattice Repository, about which a separate paper has been submitted to this conference. [1] The second is being implemented as a GUI-based application called "CHEF," an acronym which stands for:

Collaborative: objects cooperate and share information;
Hierarchical: built on a hierarchy of (mostly pre-existing) C++ class libraries;
Expansible: software is designed for future extensions;
Framework: a set of cooperating object classes that make up a reusable design for a given type of application.

As a framework, CHEF is a collection of largely autonomous components which can cooperate to perform their functions; as a program, CHEF is the name of a GUI-based application that provides an interface for doing optics calculations using these components. The framework provides a hierarchy of libraries to facilitate writing C++ programs; the application presents a user-friendly interface for exploiting the tools in that hierarchy.

In the following sections we shall briefly (a) summarize the underlying software that supports the application, CHEF, (b) describe its current status, and (c) outline our goals for CHEF's first "production" version.

*This manuscript has been authored by Universities Research Association, Inc. under contract No. DE-AC02-76CH03000 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

LIBRARY HIERARCHY

CHEF is supported upon a collection of C++ class libraries that are ordered in a hierarchy of layers. Objects within a layer "know about" each other's existence and that of the objects in lower layers but not higher ones. Presented in order, from highest to lowest, these are called:

widget_toolkit: A collection of autonomous widgets which, although used by the application, CHEF, are fundamentally independent of it and could be instantiated within other C++ programs. This layer contains GUIs that CHEF presents to its users.

physics_toolkit: Helper classes which perform specific computational tasks, such as finding closed orbits or calculating lattice functions. They also relieve a programmer of various administrative responsibilities which tend to be forgotten (e.g. turning off accelerating cavities before attempting to find a closed orbit).

bmlfactory: Contains one C++ class which instantiates to a factory object. The factory uses a LEX-YACC based parser to interpret a restricted MAD v.8 syntax. Upon request, it will build beamline objects corresponding to those described in MAD input files.

beamline: Classes for modeling beamlines and rings as sequential arrangements of neighboring elements. Whatever specifies the elements' physics and spatial configurations is encapsulated within this layer, which has undergone continual evolutionary development since its original version appeared in 1988.

integrator: Contains classes that perform basic numerical integration, including symplectic solvers for Hamiltonian dynamical systems. Combined with mxyzptlk (below), these provide a convenient way of generating maps for arbitrarily specified magnetic field configurations.

mxyzptlk: Classes for doing automatic differentiation and differential algebra: esp. Jet, Mapping, and LieOperator. The operators and functions it provides enable C++ programmers to use source code polymorphically, to calculate numerical answers - e.g. via tracking, numerical integration, or iterative convergence - and to construct maps for analysis, perturbation theory, and tracking. Since its original release in 1989, mxyzptlk has undergone evolutionary development. It's current version takes advantage of templates and other modern C++ concepts. In earlier versions, more attention was paid to accuracy than efficiency. Recent algorithmic and memory management improvements have made its performance comparable to that of software written in Fortran.

basic_toolkit: Its first version was written in 1988, when C++ was new and changing rapidly. It comprises basic algebraic and utility classes that at that time were neither

standardized nor widely available, e.g. lists, vectors, matrices, and header files containing physical and mathematical constants. By now, many of its original classes have been superseded by developments in the language, such as the STL or third party libraries like Boost. This layer's more deprecated classes are being phased out, as gracefully as possible.

Within this short report we can but comment briefly upon this hierarchy's more significant features.

Independence of physics from layout: A beamline element comprises (a) an affine transformation, representing the connection between its upstream and downstream reference frames and (b) a functor encapsulating the physics (i.e. dynamics) that propagates a particle between them. In principle, any element could be bounded by arbitrary frames, but in practice the connection used by most elements is trivially a displacement in the z (longitudinal) direction. The principal exceptions occur in classes like rbend, sbend, their combined-function variants, and Slot - which, like drift, models empty space but possesses arbitrarily positioned and oriented bounding frames.

There is no global coordinate system and no *a priori* "reference orbit," but there is a "registration" process by which a programmer can specify any particle to be "the reference particle." Classes Frame and Slot enable easy, completely general placement of beamline elements by providing a mechanism for describing how neighboring elements are arranged with respect to each other. Physics is expressed by electromagnetic fields written in coordinates local to the element's body. If more physics functors than one are available, they can be "plugged into" beamline elements at runtime. (E.g. not all elements of a particular species need behave exactly the same way.)

This paradigm is not the one familiar to MAD users, in which physics is specified relative to an assumed reference orbit from the beginning and all machine parameters are pre-scaled relative to a magnetic rigidity (i.e. momentum). These fundamental differences sometimes produce tensions when CHEF tries to interpret a MAD input file or when, an unusual geometric arrangement having been constructed, one tries to fit it into MAD's assumptions.

Context: Cooperation - CHEF's "C" - is facilitated by an agent, called BeamlineContext, which keeps track of calculations (i) that have been performed, (ii) that are needed as preliminaries for other calculations, and (iii) that have become invalid because of modifications to the beamline model. It acts as a dispatcher, or overseer, positioned between CHEF's widgets and the worker classes below.

Internal state response. Beamline elements like Lambertsons, pingers, and cavities can have time-dependent behavior or respond to the passage of a particle (or bunch of particles). This capability could be exploited for modeling wakefields, feedback, extraction, or the transfer of beam between rings.

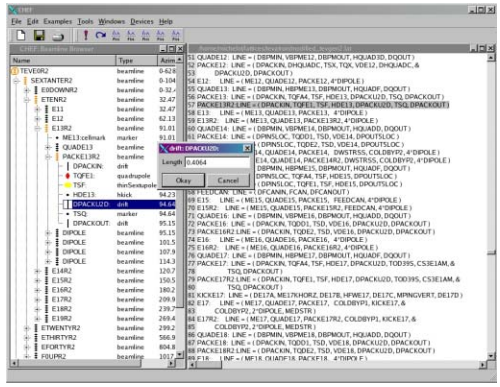

Figure 1: CHEF's browser and text windows.

Persistence. Beamline models can be streamed to a file, retaining their "existence" after a program terminates. Stored in this way, they later can be streamed back into other programs.

Algebraic polymorphism. Employing mxyzptlk's algebraic operators and analytic functions, lines of propagator or integration code can be used interchangeably for tracking and creating maps.

Run-time editing. Because the model is a C++ object, it can be modified at runtime. The possibilities go far beyond merely modifying attributes: e.g. elements can be split, combined, subbeamlines can be replaced with equivalent maps, beambeam and spacecharge lenses can be inserted.

CHEF, THE APPLICATION

While the lower level libraries can be (and are) used within C++ programs, CHEF, the application, provides graphical user interfaces to invoke a collection of standard operations. Described here are its major widgets and menu items.

Browser. A beamline browser provides a hierarchical view of instantiated accelerator models similar to that of familiar file browsers, like Windows Explorer. Instead of folders, subfolders, and files, one sees beamlines, sub-beamlines, and elements. (See Figure 1.) Right-clicking on an element reveals most of its parametric attributes, some of which can then be modified interactively.

If the beamline is instantiated from a MAD input file, the file itself can optionally be viewed and edited in a text window before and after instantiation. All beamlines defined in a file are simultaneously viewable; a user simply selects the ones to be displayed. The hierarchy shown will reflect a beamline's structure as defined within the file.

Edit menu: As mentioned already, some properties of an element can be modified within the Browser. Actions that create new lines from old ones - via operations that duplicate, condense, flatten, merge equivalent elements, convert drifts to slots, or simply rename - can be invoked from this menu.

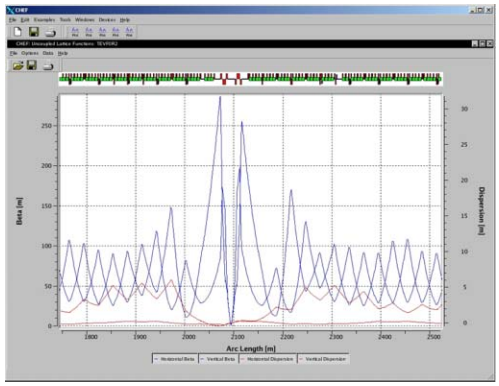

Figure 2: A plotting window, showing lattice functions.

Plotters: Currently used primarily for plotting lattice functions, dispersion, and closed orbits, in principle any data - theoretical or experimental - can be displayed and superposed within these widgets. Horizontal and vertical scrolling is enabled after zooming; plotted data can be saved as tables of numbers for use in other programs.
Phase space views: Multiple views of phase space are available to monitor the behavior of a virtual particle traversing a ring. These widgets use OpenGL to provide both two- and three-dimensional projections that update automatically as the simulation is running. An "action-angle" view is particularly good for exposing KAM tori. [2]
bpm tracer: A circulating particle can also be followed within a window that receives and displays signals from monitors embedded into the accelerator model. The plot has a fading memory of user-specified length, enabling a user to watch an envelope evolve. Also based on OpenGL, it could (in the future) be made three-dimensional, esp. for display with apertures.
Geometry viewer: This "SiteViewer" object provides a global look at a model's geometric layout, essentially integrating the connections between the beamline elements' local frames. Zooming in on and and moving the image are performed with the mouse. Right-clicking on an element provides information about it. Currently two-dimensional, the widget nonetheless is based on OpenGL so that it can be extended to three dimensions in the future.
Selector: Constructs propositions regarding elements' attributes which are then applied to select subsets of elements. When invoked from the Browser, the subbeamlines open to expose and highlight the selected elements. Invoked within the SiteViewer, selected elements are highlighted.
Python interpreter: One window provides an interface for a Python interpreter which utilizes Python bindings that have been created for all lower level C++ classes and for CHEF's components. Within it, a user can extend CHEF without leaving the application by writing and testing python scripts that can be executed, saved, and later recalled.
Device monitor: Using an XMLRPC protocol and procedures provided by Fermilab's Controls Group, this window monitors "devices," a category that encompasses any datum provided about a Fermilab machine. The display can be toggled to update itself automatically or only upon request.
Messages: A message window captures error and warning messages that are streamed during a calculation, either announcing that it has gone wrong or providing information about its progress.

GOALS

In October, 2004, an initial release of CHEF was installed on a UNIX machine. CHEF and its components were themselves almost completely free of dependencies on the operating system, and by January, a Windows XP version was made available. For those who want to explore its possibilities, despite its currently incomplete state and lack of documentation, a Windows version can be downloaded from http://www-ap.fnal.gov/CHEF. An RPM file for Fermi Linux will be provided in the future, and the source code will be included when the product reaches a reasonably stable finished state. Apart from ongoing improvement of lower level libraries, our work on CHEF between now and then will focus on the following priorities.

non-periodic lines: Most of CHEF's current functionality targets storage rings and synchrotrons. Our highest priority is to add features more appropriate for transfer lines and linacs.
unrestricted geometric manipulations: The lower level libraries already contain the tools necessary for arbitrary, even exotic, placement of elements. Our goal now is to lift that capability into the CHEF layer.
operate on selected objects: Having selected a group of elements, a user will be able to edit their properties, if of the same type, or to group them into "circuits," enabling control by specifying values of "knob" parameters.
importing measured properties: To morph a "design" model into a "realistic" one, CHEF will provide a mechanism for importing data from existing databases, including the descriptions contained in the Lattice Repository. [1]
nonlinear analysis: We will add menu entries for doing simple nonlinear analysis on rings, such as calculation of resonance widths, tune spread, and focal properties.

Most of these involve functionalities are already present in the lower level libraries. The task will be to make them available from CHEF's graphical user interfaces.

REFERENCES

[1] Jean-Francois Ostiguy, *et al.*, "The Fermilab Lattice Information Repository." This conference.

[2] This particular tool is a direct descendant of AESOP, which was described in "Exploratory Orbit Analysis," *Proceedings of the 1989 IEEE Particle Accelerator Conference*. IEEE Catalog Number 89CH2669-0.