MAPA - AN OBJECT-ORIENTED CODE WITH A GRAPHICAL USER INTERFACE FOR ACCELERATOR DESIGN AND ANALYSIS

Svetlana G. Shasharina, Weishi Wan and John R. Cary

Center for Integrated Plasma Studies and Department of Physics
University of Colorado
Boulder, CO 80309-0390 USA

Abstract

We developed a code for accelerator modeling which will allow users to create and analyze accelerators through a graphical user interface (GUI). The GUI can read an accelerator from files or create it by adding, removing and changing elements. It also creates 4D orbits and lifetime plots. The code includes a set of accelerator elements classes, C++ utility and GUI libraries. Due to the GUI, the code is easy to use and expand.

1 OBJECT ORINTED APPROACH

C++ was chosen for the project (mapa) because it is the most commonly used object-oriented programming (OOP) language in scientific and programming communities. It is portable across many platforms and works elegantly with C procedures of X/Motif. This language provides data encapsulation, inheritance (including multiple inheritance) and dynamic binding [1]**.** It allows overriding and overloading of methods. The code, written in OOP language has a better chance to be clear, expandable, flexible and reusable.

Mapa uses many patterns (idioms) of OOP [2]. First of all, we used classical patterns of canonical classes, which emulate behavior of built-in types. Thus, we can treat vectors, Matrices, strings in a most convenient way (i.g. we can add them and perform other "natural" operations). We also used the Composite pattern for building primitive (single) and composite (beamline) accelerator elements, so that we can treat them equally. From behavioral patterns we used the Handle/Body pattern to build the garbage collection for strings, vectors and matrices. The Observer pattern was used for realization of Model/View-Controller structure, which allows to separate model from views and update the views upon changes in models. The Template method, which encapsulates logical parts of algorithms and leaves their definition to derived classes, saved us a lot of coding and made the code more transparent. We also used the Letter/Envelope pattern for providing polymorphic behavior for arithmetic classes (latter will be used for implementation of TPSA).

2 SYSTEM HIERARCHY AND MAPA'S CAPABILITIES

The code has two hierarchy trees describing the models of interest. One tree has to do with general systems, which have names (class System), parameters and options with unified I/O (class SimpleSystem). The map hierarchy describes systems with dynamic features (the Advance methid propagates dynamic variables though time). The two hierarchies meet to create (through multiple inheritance) the SimpleMap class (which is still an abstract class), from which most of mapa systems are derived. Thus, an abstract Element is derived from the SimpleMap, as well as Accelerator, set of classical nonlinear dynamics maps (Henon's, standard map etc.) and Torus (class for studying motion of particles on toroidal fusion devices). Concrete elements and composite element (Beamline) are derived from Element. Accelerator and Element are related through aggregation: Accelerator has a list of Element pointers.

3 GRAPHICAL USER INTERFACE

One of the main efforts in building mapa was to make the code user friendly. To reach the goal, we created a set of C++ classes for encapsulation and convenient use of X/Motif. These classes became the base of the GUI. The use the GUI of makes computing interactive. First of all, it allows the user to select the system of interest through a menu-like widget. Click of the mouse button brings up the controller of the system. The controller can read/write the system from/to files, list parameters and options, change them. It also brings up, start, stops and saves simulations relevant to the system (like now we have Monte Carlo simulations resulting in showing average behavior of dynamic variables of the system versus time). It also allows users to see the orbits in phase or real space (each orbit can be start by a click on the plot) and lifetime plots. The accelerator controller can change the beamline by visual adding and removing of elements from a table, whose parameters can be changed through the same interface. A special widget allows to find fixed points of different order by using various solvers, with the initial guess being being found and input graphically.

The set of controllers mirrors the abstract part of the system hierarchy, so that particular systems do not need a specialized controller. The GUI allows coexistence of many systems and simulations simultaneously, which makes the task of design and analysis more fast easy.

4 DISCUSSION

We are planning to

1. teach mapa to read from diffirent file formats (like MAD and other popular formats);

2. implement TPSA for non-linear map analysis;
3. include possibilities of missalignments;
4. improve fitting;
5. make the set of accelerator elements richer;
6. create survival plots.

REFERENCES

1. B. Stroustrup, *The C++ progamming lanquage, second edition* (Addison-Wesley, Reading, Massachusetts, 1991).

2. E. Gamma, R. Helm et. al, *Design Patterns* (Addison-Wesley, Reading, Massachusetts, 1995).