OBJECT ORIENTED C++ SOFTWARE COMPONENTS FOR ACCELERATOR DESIGN*

D. L. Bruhwiler,# J. R. Cary,+ and S. G. Shasharina, Tech-X Corporation, Boulder, CO

Abstract

Object oriented programming techniques make it possible for accelerator designers to independently develop C++ software components that can work together. As an example of this approach, we discuss some of the software components being developed at Tech-X Corporation, including: TxSTD, a library of standard utilities [1]; TxID, a library of data-holding and nonlinear dynamics classes [2]; TxAC, an accelerator modeling class library [3]; an X/Motif library used in the MAPA application [4] for interactive visualisation of dynamical systems such as particle accelerators; TxAN, a library of analysis and simulation classes relevant to dynamical systems; and the LION++ nonlinear optimization library [5].

1 OBJECT-ORIENTED PROGRAMMING

The computer software industry has been rapidly moving towards the use of object oriented programming (OOP) [Ref.'s 6-10]. OOP provides a superior mechanism for rapid development and testing, as well as ease of maintenance and extensibility for large scientific codes.

OOP consists of designing classes, where a class is a format for holding and interacting with data. An object is a particular instance of a class, just as 3.1 is a particular instance of a real number. Each object has its own data, while all objects of a given class share the same functions (called methods). A program is constructed by first defining the classes. The program instantiates objects corresponding to the classes and, by calling the public methods of these objects, manipulates or displays the data. The public methods of a class define its interface.

The three defining properties of OOP are *encapsulation, inheritance* and *polymorphism*. *Encapsulation* refers to the fact that objects are accessed only through a public interface, while their internal data and implementations remain hidden. This feature ensures that the code is safe from unwanted modifications. *Inheritance* allows the programmer to define new classes that inherit most of their coding from existing classes, only modifying or adding data and methods as needed. This allows for flexibility to increase the capability of an application in ways not foreseen by the original programmers, without rewriting the code. *Polymorphism* allows different classes to support the same interface but fulfill requests differently at run time. This makes *extensibility* possible – a new class that implements a new algorithm can inherit from an existing class and be used in its place.

C++ is the best available OOP language for scientific applications. The “expression template” technique [11-13] allows C++ code to perform as well as optimized Fortran77 in vector loop comparisons. Furthermore, the use of "generic programming" and "template meta-programming" methods has yielded C++ linear algebra solvers that are faster than Fortran77. [14] The base of numerical libraries for C++ is rapidly increasing, [15] and, because C++ is a superset of the well-established C language, it has access to a great wealth of legacy C code.

2 PUTTING THE PIECES TOGETHER

Leo Michelotti developed the first C++ class library for modeling beam dynamics in an accelerator [16] and deserves credit for introducing the accelerator physics community to the benefits of C++ and object oriented programming. Other C++ codes are now under development, including MAD-9 [17], LEGO [18], Teapot++ and MAPA [4,19]. Unfortunately, these codes are independently developed, and the C++ libraries used in one code cannot be used in the others.

The goal of the CLASSIC project [20] was to standardize the structure and interface of a C++ accelerator dynamics library in order to promote the sharing and reuse of code among the various developers. We propose an alternative approach – the use of template based “traits” mechanisms [21-23] – to make C++ accelerator class libraries interoperable and, hence, true software components. First, we briefly describe the class libraries under development at Tech-X Corporation, then we elaborate further on the use of traits.

3 ACCELERATOR DYNAMICS LIBRARY

The Tech-X accelerator dynamics library TxAC, now at the first alpha release, has been used to successfully model the Advanced Light Source, finding correct values for tunes and dynamic apertures. TxAC includes an SIF parser (Standard Interchange Format, the MAD-8 input language [24]) that can create an accelerator model (beamlines, elements, and properties of the charged particle) with full support for mathematical formulas.

TxAC comes with a number of built-in element types (e.g. quadrupoles, dipoles, thin RF cavities) which use analytical models for the electric and magnetic fields. Each element type knows how to calculate the phase

* Work supported by Tech-X Corporation and by the U.S. Department of Energy, grant no. DE-FG03-96ER82292.

Email: bruhwile@txcorp.com

+ Also, University of Colorado Physics Department, Boulder, CO

space map for particles passing through it, so the code can be used for tracking, and it allows the user to specify whether tracking should use 2, 4, or 6 dimensions. [19]
TxAC also includes a beamline class, allowing the user to define an arbitrarily deep hierarchy of beamlines within beamlines -- critical for describing large rings, which may consist of only a dozen or so unique elements that are repeated in various combinations for a total of hundreds or thousands. Synchrotron radiation effects are optional.
The TxAC class hierarchy is shown in Fig. 1. A new element type can derive from the most appropriate part of this hierarchy and thus obtain most of the needed features. New features can then be added in the derived C++ class, which must also provide the code that implements tracking through the element. Once a new element class has been defined, it need only be registered with a name in the TxacElements file and, upon recompilation, the new element type will be supported by the parser and the graphical user interface (GUI).

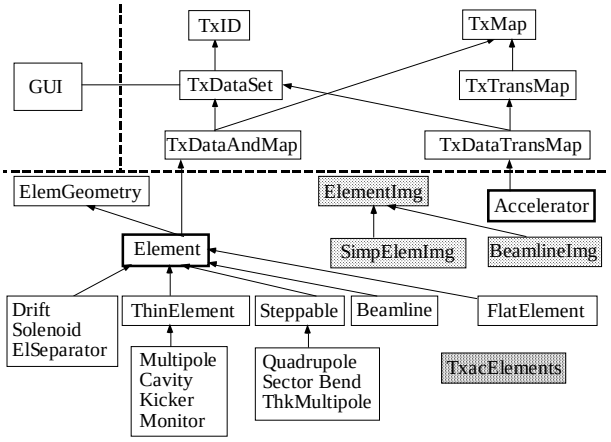

Figure 1: Schematic showing TxID object hierarchy (upper right), the TxAC object hierarchy (bottom) and one example of interface with the X/Motif graphical user interface (upper left).

4 ACCELERATOR ANALYSIS LIBRARY

The accelerator analysis library TxAN uses TxAC to calculate matched Twiss parameters (or the linear and 2nd-order dispersion) for any closed orbit or to propagate specified initial values element by element through a beamline. TxAN can also calculate the position and orientation of all beamline elements in a global Cartesian coordinate system, or use Monte Carlo methods to propagate the RMS moments of a particle distribution.
The classes of TxAN inherit from the data holding classes of TxID and define a new data holding class for storing the analysis results. Developers can define a new C++ class for accelerator simulation, inheriting appropriately from the TxAN hierarchy. The data resulting from the new simulation is stored in the plot data class, while any parameters relevant to the simulation are stored in the TxDataSet class of TxID. The X/Motif GUI can extract data from these classes, thus allowing the user interactive control over the simulation with immediate rendering of the resulting data.

5 GRAPHICAL USER INTERFACE

The X/Motif GUI supports on-line tracking and renders the resulting surface-of-section (SOS) plots. Initial conditions for the next particle can be specified simply by clicking in the windows. These SOS plots can be resized interactively, and the phase space variables can be paired up in any order. Fixed points can also be found.
The GUI also allows one to browse the local file system for an appropriate input file, and then to save changes in the same file or in a new file. The GUI can plot a schematic layout of the accelerator, warning the user if, for example, a ring does not close on itself.
The GUI provides appropriate windows for interactively changing any relevant parameters for the accelerator, for individual elements or for the analyses. For each type of analysis, the GUI provides a menu item allowing the user to activate the simulation, and then renders line plots of the resulting data. Developers can define new element classes or analysis classes through inheritance, and these new types will be directly supported by the GUI after compiling the new classes and relinking the application.

6 LION++ OPTIMIZATION LIBRARY

LION++ [5] is a suite of flexible and extensible C++ software components for numerical computing. Still being actively developed, the present release features TxOptSlv, a library for the optimization of nonlinear user-specified functions, and TxBase, a library of unary functors and other general utilities. LION++ takes full advantage of sophisticated templating techniques and object oriented design in order to provide users with maximum flexibility in the choice of argument type and return type for the merit function that needs to be optimized and in the configuration of options for the built-in algorithms.
Three multidimensional algorithms have been implemented, including nonlinear *simplex* and *Powell*, which do not need the gradient of the function, and one due to *Fletcher, Reeves, Polak and Rebiere* (FRPR), which does require access to the gradient. The Powell and FRPR algorithms require access to 1-D line optimization algorithms. Three 1-D algorithms have been implemented, including *golden section* and *Brent*, which do not need the function derivative, and a *modified secant* algorithm, which does require access to the derivative.
At present, all of the optimization algorithms are unconstrained although we are currently implementing algorithms that constrain the arguments in various ways. We

are also now implementing the Levenberg-Marquardt algorithm, which is effective for nonlinear least squares fitting and, more generally, for simultaneous optimization of many nonlinear functions. LION++ is readily extensible so developers can implement their favorite algorithms or create a thin interface to other C and C++ algorithms, all with the same convenient user interface.

7 THE TRAITS MECHANISM

"Traits" are defined through a template class or struct. For example, Fig. 2 shows the definition of two traits for a 1-D array class: the type of the argument held by the array and a public method Resize() that will change the length of the array. It is assumed that the 1-D array supports the [] operator for accessing the elements. The TxArrayWrap class in Fig. 2 wraps a simple C-style pointer, which does not have built-in resizing capability.

In LION++, the argument list for multidimensional functions is declared to be of type VecType, and all optimization classes are templated over VecType. Users must instantiate a templated optimization object, where the template parameter specifies the argument type of the function to be optimized. Thus, LION++ is "container-free" as defined in Ref. [23].

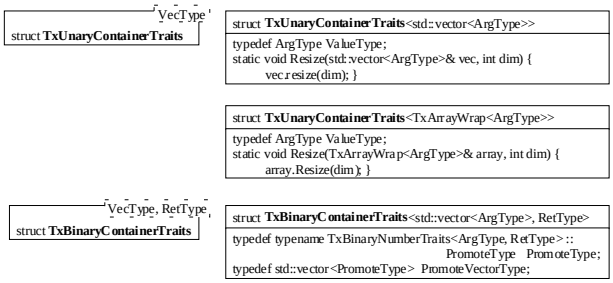

Figure 2: Unary (above) and binary (below) traits for containers (user-defined 1-D arrays).

The method for tracking in an accelerator library (called Advance() in TxAC) could be templated over the type of the class that is to be tracked. Then the traits mechanism could be used to allow this class to be any of: a) a single particle of precision float, double, etc.; b) a 1-D array of particles, with the array type completely arbitrary; c) a generalized user-defined particle class; or d) any C++ differential algebra (DA) library. Thus, the traits mechanism immediately allows for some interchange of software components between different C++ code developers (particle classes and DA libraries).

Taking this idea a step further, one could template the controlling class of the accelerator dynamics library (called Accelerator in TxAC) over the element type. In this case, the "traits" would include methods for reading and writing the physical data associated with an element, tracking a particle (or array, or DA vector) through an element, etc. With this approach, the developer of one class library could use the element classes from another accelerator class library, allowing for direct model intercomparisons.

There are two prices to be paid for this approach. The first is that such heavy use of templates can lead to very long compilation times. The second is that the use of traits requires the use of rather awkward syntax in the source code for the accelerator class library.

8 CONCLUSIONS

A "container free" approach to the development of C++ class libraries for the modeling and design of accelerators is an elegant and relatively straightforward way to make the many existing class libraries interoperable. This approach places some burdens on the library developer, but it avoids the difficulty of trying to convince developers that they should base their code on some standard class library design.

The authors thank Julian Cummings for discussions regarding the use of traits and John Verboncoeur for discussions regarding object-oriented programming.

9 REFERENCES

[1] TxSTD page, URL http://www.techxhome.com/freestuff/txstd
[2] TxID page, URL http://www.techxhome.com/freestuff/txid
[3] TxAC page, URL http://www.techxhome.com/freestuff/txac
[4] MAPA page, URL http://www.techxhome.com/products/mapa
[5] LION++ page, URL http://www.techxhome.com/products/lion
[6] G. Booch, "Object Oriented Development," *IEEE Transactions on Software Engineering* **12**, 211 (1986).
[7] B. Stroustrup, *The C++ Programming Language*, Third Ed. (Addison-Wesley, Reading, Massachusetts, 1998).
[8] S. B. Lippman and J. Lajoie, *C++ Primer*, Third Ed. (Addison-Wesley, Reading, Massachusetts, 1998).
[9] E. Gamma, R. Helm et. al, *Design Patterns* (Addison-Wesley, Reading, Massachusetts, 1995).
[10] A. Elins, *Principles of Object Oriented Software Development* (Addison-Wesley, Reading, Massachusetts, 1994).
[11] A. D. Robison, "C++ Gets Faster for Scientific Computing", *Computers in Physics* **10**, 458 (1996).
[12] T. Veldhuizen, "Expression Templates," *C++ Report* **7** (1995).
[13] S. Haney, "Beating the Abstraction Penalty in C++ Using Expression Templates", *Computers in Physics* **10**, 552 (1996).
[14] The Matrix Template Library (MTL) home page at URL http://www.lsc.nd.edu/research/mtl/
[15] URL http://monet.uwaterloo.ca/blitz/oon.html#libraries
[16] L. Michelotti, "MXYZPTLK and Beamline: C++ Objects for Beam Physics," AIP Conf. Proc. **255** (Corpus Christi, 1992).
[17] MAD-9 web site, URL http://wwwslap.cern.ch/~fci/mad/mad9
[18] Y. Cai, M. Donald, J. Irwin, Y. Yan, "Lego: A Modular Accelerator Designer Code,"SLAC-7642, August 1997.
[19] D. L. Bruhwiler, J. R. Cary and S. G. Shasharina., Proc. Sixth European Particle Accelerator Conf., (Stockholm, June, 1998).
[20] CLASSIC web site, URL http://wwwslap.cern.ch/classic
[21] N. C. Meyers, "Traits: a New and Useful Template Technique," *C++ Report* **7** (1995); URL http://www.cantrip.org/traits.html
[22] T. Veldhuizen, "Using C++ Trait Classes for Scientific Computing;" URL http://monet.uwaterloo.ca/~tveldhui/papers/traits.html
[23] G. Furnish, "Container-Free Numerical Algorithms in C++," *Computers in Physics* **12** (3) (May, 1998).
[24] MAD-8 web site, URL http://wwwslap.cern.ch/~fci/mad/mad8