Lawrence Berkeley National Laboratory
Recent Work

Title
Object-Oriented Software Construction at ALS

Permalink
https://escholarship.org/uc/item/6cc0b9tn

Author
Nishimura, H.

Publication Date
1993-10-01
Object-Oriented Software Construction at ALS

H. Nishimura

October 1993
DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.
Object-Oriented Software Construction at ALS

Hiroshi Nishimura
Accelerator and Fusion Research Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

October 1993

Presented at the International Conference on Accelerator and Large Experimental Physics Control, Berlin, Germany, 18-22 October, 1993.

This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Materials Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF0098.
Object-Oriented Software Construction at ALS*

Hiroshi Nishimura
Advanced Light Source, Lawrence Berkeley Laboratory, University of California, Berkeley, CA 94720, USA

C++ class libraries have been developed and used for accelerator modeling and machine control at the Advanced Light Source. A class library for accelerator modeling is portable and supports multiple model instances dynamically at run time. A class library for machine control covers fields from virtual devices to simulation studies.

1. Introduction

The Advanced Light Source (ALS) [1] is a 1.5-GeV, small-emittance electron storage ring commissioned at Lawrence Berkeley Laboratory recently [2]. It is a third-generation light source with intensive use of eXtension devices, which requires accurate and flexible modeling for machine control. Object-oriented programming (OOP) has played a major role in machine modeling and in construction of the graphical user interface.

At the higher levels of accelerator control and operation, OOP has been used to construct a virtual-device layer on top of the flat 'database' that is a list of channels, and to provide an application framework with graphical user interface for the high-level control programs.

This is a summary of the object-oriented software construction effort by accelerator physicists at the ALS Center.

2. Simulation and Modeling

Our effort has been to make simulation and modeling programs as flexible as possible. In the lattice design phase, we used the Pascal S compiler [3] as our framework. We have extended the Pascal compiler itself and associated simulation and modeling library functions to a set of new P-codes, which enabled us to accept user algorithms as input. The programs created using this approach are: a linear 4x5 code, Tracy [4]; a full 6x6 kick code, Gemini [5]; and a linear 4x5 and full 6x6 code, Tracy2 [6]. They were developed on VAX/VMS using VMS Pascal and are used for off-line studies to evaluate the machine performance in presence of imperfections [7].

For model-based control, we focused not on the application programs but on the simulation and modeling library for reusability. The simulation kernel of Tracy was extracted and rewritten in C to provide a portable library. This library is object based to support dynamic modeling [8]. An accelerator is represented by a variable of the accelerator type, which allows us to manipulate multiple accelerator instances at run time. At that stage, it was not object-oriented in any particular OOP language to conserve portability.

Our first OOP effort was in a pure OOP language, Eiffel [9]. An accelerator object in C was embedded in the Eiffel object [8] on UNIX workstations. It is an OOP taken to a very high level and the scale of the accelerator class is large. The same wrapping was done by using a hybrid language, Objective-C, for the use with NextStep on NeXT.

The ALS control system [10] was optimized for access from the DECpc 480ST PC clones, serving as operator consoles. The accelerator device access over the Ethernet was not fast enough for on-line modeling on UNIX workstations. Therefore, our simulation and modeling library had to be available on PCs. At that stage, we moved to C++. The C library was completely redesigned using objects from a very low level and turned into a C++ class library called Goemon [11]. This is a class library for accelerator modeling and simulation to be the building blocks of high-level physics and control programs, with the following features:

- The numerical integrator of the Hamiltonian is localized. It is the standard 4x5 matrix formalism in the current version of Goemon but is easily exchangeable. That is, Goemon provides a simulation and modeling program framework that is relatively independent from the Hamiltonian and its numerical integrator.
- Isolated from the hardware environment: accelerator device access and graphic user interface are outside of Goemon for better portability and modularity.
- Compact: only 3000 lines of source code for the basic modules.

Additional 1000 lines for ALS-dependent classes.

Figure 1 shows the Goemon class relationship in Object Model Notation [12].

![Goemon Class Relationship](image)

Fig. 1 Goemon Class Relationship

There are two major class categories, Component and Machine. Component includes lattice elements such as drift spaces, magnets, and monitors. Element is the base class for Component objects. Here are some of its member functions:

```cpp
virtual double getK(); // return strength
virtual void setK(double K); // set strength
virtual void pass(Vec5 *x); // propagate a particle through its field.  
virtual void passL(Vec5 *x, const Vec5 &K); // propagate a particle through the linearized field.  
virtual void passM(Vec5 *x, Vec5 *M); // propagate 4 particles through the linearized field.  
virtual double getA(); // return its transfer matrix.  
```

* This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Material Sciences Division of the U. S. Department of Energy under Contract No. DE-AC03-76SF00098.
Element is not a beam transport line but a unit of it. The main function is to propagate a particle through its field. Wiggler is derived from Quad because the "vertical quadrupole" is enough to calculate on-energy functions such as tunes. But a compound object of Drift and Bend is also used for the Hard-Edge model of wigglers and undulators. Machine contains an array of Celement objects to represent a beam transport line. Celement represents an Element objects in a beam transport line and defined as:

class Celement {
  public:
    Element *Elem;
    double S, SizeX, SizeY, Xoff, Yoff;
    // distance, size, misalignment
    Vec5 RefPath, TwissH, TwissV, Eta;
    // reference orbit, orbit, alpha, beta, eta,
    void pass (Vec5 *x);
    void passL(Vec5 *x);
    void passM(Mat55 *M);
    void PassTwiss(Vec5 *twissH, Vec5 *twissV);
    void setOffset(double x, double y);
    double getXoffset();
    double getYoffset();

};

The base class of Machine is BeamLine, whose primary task is linear optics calculation. Ring is derived from BeamLine as a class for circular rings.

The first version of Goemon is portable on UNIX workstations, IBM PCs clones running Windows 3.1 and Windows NT, and Macintosh with System 7.x.

There is another OOP effort at very high level for model-based control and machine physics studies. It uses the GLISH software bus architecture [13] to assemble a high-level application program by plugging together modular, single-function programs [14]. Goemon will be one of the suppliers of the libraries to the software bus.

4. Device Control

The ALS Control System [10] is a simple but highly distributed system. At the lowest level, there are more than 600 Intelligent Local Controllers (ILCs) providing a direct interface to target devices. Most of the devices are controlled by one ILC but some are controlled by several ILCs. Each ILC is mapped onto the 'database' on the shared memory in the Collector Micro Modules (CMMs). This database is shared by Display Micro Modules (DMMs). Each DMM is independently connected to a PC clone that serves as an operator's console. This architecture has provided very efficient device access during the commissioning, and we plan to adopt an EPICS system [15]. The OOP effort described in this section was undertaken by the accelerator physicists who used the control system during commissioning.

In the ILC-CMM-DMM architecture, the PC control consoles have logically transparent access to all the device entries in the database. A class library DEVICE provides a virtual device layer on top of the database. As required for PC clones running MS Windows, we implemented the DEVICE class by using Turbo Pascal with ObjectWindows that is a class library for Windows application programming. When C++ caught up with it, the DEVICE class library was ported to C++ compilers, Borland C++ 3.1 and Visual C++ 1.0.

Here is the interface of DBobj [16] that is the base of DEVICE. It encapsulates channels associated with the same database device name.

class DBobj {
  public:
    float getAM(); // get analog monitor value
    float getSP(void); // get analog control value
    void setSP(float X); // set analog control value
    int getBM(void); // get boolean monitor value
    int getBC(void); // get boolean control value
    void setBC(int B); // set boolean control value
    void getBytes(int Offset, char *s, int N); // get N bytes of data
    void setBytes(int Offset, char *s, int N); // set N bytes of data

};

It is a wrapper of the basic call library for DMM access in C. Based on this base class, device classes are derived as follows:

DBobj // base class
MAGobj // magnet
QUADobj // quadrupole magnet
BENDobj // bending magnet
SEXTobj // sextupole magnet
STEERobj // steering magnet
VACobj // vacuum pumps and gauges
BPMobj // beam position monitors
DCCTobj // beam intensity monitor

Each object of these classes corresponds to a virtual device that handles its local operational context. MAGobj performs turn on/off, ramp up/down, and controlled cycling, status watching. BPMobj supports normal XY reading, fast turn-by-turn reading and its Fourier analysis, XY averaging and referencing, gain setting, and calibration. Some of them are simple C++ wrappings of existing C library functions [17].

There are three major classes for ALS storage ring control: SRMAGS for quadrupole, bend, and sextupole magnets, SRSTRS for steering magnets, and SRBPMs for beam position monitors. They are collections of the DEVICE objects:

SRMAGS QUADobj(49), BENDobj(1), SEXTobj(2)
SRSTRS STEERobj(160)
SRBPMs BPMobj(96)

The numbers in the parentheses are the number of each DEVICE object used in the programs for daily operations.

4. Ongoing Effort

We are in the transition phase to the normal operation mode. A more systematic effort is required in a limited amount of machine time for the development. The role of OOP keeps increasing in the field of physics application programs. Goemon has been continuously
enhanced for the use at ALS to construct application programs. There are several new C++ classes that are under development. GPIB devices are going to be supported over the network by using an object [18]. RF and RF-related beam physics is going to be supported by an object [19].

**OOP** for database construction and management is becoming an urgent issue. As the relational database scheme cannot map the object relationships efficiently, we will be using RAIMA Object Manager [20].

Several efforts are in process for networking. Some are Windows NT based including NetDDE [21] and others are UNIX based such as RPC. It is desirable to have a standard class library to encapsulate the detail of the network programming.

5. Conclusion

A class library approach has been proven to be efficient for modeling and simulation studies. It is also applied to machine control mainly by the physicists, which will be compatible with future plans of the control systems design and development. **DEVICE** will be released from the low level tasks that can be done by **EPICS**.

**Acknowledgments**

The author would like to express his appreciation to all the ALS commissioning team members, particularly the controls engineers led by S. Magyary.

**References**

[20] Raima Object Manager, Raima Corporation, WA.