3rd Int. Conf. Accel. Large Exp. Phys. Control Syst. ICALEPCS1991, Tsukuba, Japan JACoW Publishing
ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P01

OBJECT-ORIENTED PROGRAMMING TECHNIQUES FOR THE AGS BOOSTER*

Joseph F. Skelly
AGS Department, Brookhaven National Laboratory
Upton, New York 11973

Abstract

The applications software developed for the control system of the AGS Booster Project was written in the object-oriented language, C++. At the start of the Booster Project, the programming staff of the AGS Controls Section comprised some dozen programmer/analysts, all highly fluent in C but novices in C++. During the course of this project, nearly the entire staff converted to using C++ for a large fraction of their assignments. Over 100 C++ software modules are now available both for Booster and general AGS use, of which a large fraction are broadly applicable tools. The transition from C to C++ from a managerial perspective is discussed and an overview is provided of the ways in which object classes have been applied in Booster software development.

Introduction

At the outset of the Booster Project,[1] management decided to promote the use of object-oriented techniques among the programming staff. Our hope was to achieve improved programming efficiency and greater maintainability of code through increased modularity. The C++ language was chosen because of its accessibility to a staff fluent in C, and because it was well supported on the computing system already in place. Whereas prior efforts at in-house self-education in C++ had yielded only very limited success, our staff now is very comfortable using C++, and we consider that our goals in promoting C++ have been satisfactorily achieved. During the past two years, our programmers have accumulated nearly 200 staff-months of experience with C++, and produced some 160 source-code modules totaling more than 100,000 lines; of these, more than 80 are tool modules which define more than 300 object classes. The Booster was commissioned in June of this year; during this period our software was exercised vigorously, and software performance and user reaction were favorable. The reasons for this success will be discussed below.

Environment

The AGS Distributed Control system (AGSDCS) comprises a network of approximately 50 Hewlett-Packard/Apollo workstation nodes on a Domain token-ring network which spans the AGS accelerator complex. Ten workstations provide the operator interface at five consoles in the AGS Main Control Room. About 15 workstations are used for programmer or physicist development nodes, and the remainder are used as control system consoles by engineering and technician work groups among the accelerator staff, or as data-collection servers in the accelerator complex. The workstations run a Unix-like operating system and provide a high-resolution display, for which an internal Graphics User Interface (GUI) standard for the programs has been established.

The AGSDCS is interfaced to some 5800 accelerator devices via more than 100 so-called "device-controllers" in more than 50 locations. The device-controller layer is currently implemented with Intel Single-Board Computers (SBCs) in Multibus packaging. Device-controllers are connected to so-called "stations" via the GPIB (IEEE-488) bus; stations are implemented either in Multibus SBCs (the older AGS version) or in Apollo workstations (the new Booster version). Access by high-level programs to the network of accelerator devices is supported by a library of toolkit routines which permit a device to be referenced by just its name. The library routines resolve the device address in the network by reference to descriptor tables constructed from a relational database which describes the entire control system.

Transition to C++

A number of factors are discussed here which contributed to the successful transition of the staff to C++. Experience with this process suggests that each factor is important, and that the absence of any one of them would have had a very negative impact on its success.

Assignment Profile

Staff members were given independent software assignments for the Booster Project, and permitted to develop them individually. The opportunity to nurture a new project from its inception without undue burden of prior development encouraged the staff to apply new techniques. In addition, it was recognized early that many of the assignments required common tools, and management fostered cooperative efforts

*Work performed under the auspices of the U.S. Department of Energy.

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P01

among the staff to define and develop generic object-oriented tool packages.

Staff Experience

The programmers had already mastered the details of the control system infrastructure. New staff hired for the Booster Project were given adequate time to become familiar with the control system before learning C++ and addressing the new Booster-specific programming assignments.

C++ Lead Programmer

The staff was seeded with one experienced C++ programmer to serve as in-house consultant and mentor. During the succeeding year, the staff members' primary responsibility became their Booster assignment, and as they addressed this assignment they adopted C++ as their design language. A C++ culture was established within one year, and a class library rapidly accumulated which functioned as a peer-developed resource of programming models.

Classification of Classes

The class library was recently examined to acquire a snapshot of its contents (which are still expanding). The contents have been categorized according to the type of services which the classes offer.

Class Category	# Modules	# Classes
Operating System Services	14	36
GUI Services	4	22
Control System Services	22	60
Data Acquisition, Display	17	53
Device Tools	12	28
Accelerator Tools	16	87
Accelerator Physics Tools	7	44

Although the class library contains a large number of classes, many of these are intended only for internal use by the tool modules; a programmer wishing to use these tools need become familiar with only a few classes at a time.

Class Examples

Some samples are offered of the classes in each category of the class library, along with some methods (function members) defined for them, in order to exhibit the ways in which these classes are applied. The format in this table is similar to the C++ code from which these examples were derived: a class-definition line ("class ClassName") is followed by a number of lines defining methods for the class ("ClassName::method"). The formal parameters (arguments) for the methods are not displayed, for the sake of simplicity; likewise, most of the syntax of the C++ language is suppressed, although class derivation is exhibited.

Accelerator Tools - A Special Niche

An object-oriented approach to design of the "accelerator tools" category seems to offer a special opportunity for programmers in an accelerator controls environment. It is often the case that the architecture of the control system imposes constraints on the hardware designers, constraints which cause the elements of the accelerator to be artificially fragmented into multiple "devices", or "control system primitives". In the AGS control system, the control system primitive is called a "logical device". As an example, the engineer designing an interface for a multi-wire profile monitor or "harp", was obliged to implement the timing control as one logical device, the gain control as a second logical device, positin control (insert/retract) as a third logical device, and acquisition of the profile as a fourth logical device. Moreover, gain and timing control were shared among a collection of several harps in the same beam line. This complexity is by no means unusual, and is a common consequence of the necessity to standardize control system architecture and to solve difficult accelerator design problems.

With an object-oriented tool to support program interaction with a harp, the complexities resulting from the multi-device interface can be hidden inside the class design. The high-level programmer can then interact with a single entity--the harp object--and function much the same way the physicist does when he views the harp as a single component of the

Table I. Class Samples - Operating System Services

```
Class  SharedMemory
    SharedMemory::GetLock
    SharedMemory::ReleaseLock
Class  MbxMessage                          //    Mailbox Message
Class  ApolloMail  :  MbxMessage           //    derived class from MbxMessage
    ApolloMail::ServerGet                  //    server access to mailbox
    ApolloMail::ServerPut
    ApolloMail::ClientGet                  //    client access to mailbox
    ApolloMail::ClientPut
```

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P01

Table II. Class Samples - GUI Services

Class PopupMessage : GenericPopup	//	derived class from Generic Popup
PopupMessage::display	//	display in proportional font
PopupMessage::display_mono	//	display in monospaced font
PopupMessage::display_ok	//	display and await confirmation
PopupMessage::ask_yn	//	ask question, get yes/no reply
Class PopuMenu : GenericPopup	//	derived class from Generic Popup
PopupMenu::getchoice	//	get a single choice
PopupMenu::getchoices	//	get multiple choices

Table III. Class Samples - Control System Services

Class Alarm		
Alarm::Log	//	Log in database
Alarm::DeLog	//	DeLog from database
Alarm::Priority		
Class Sld : Alarm	//	alarm for Sld (Simple Logical Device)
Class Controller : Alarm	//	alarm for Controller

Table IV. Class Samples - Data Acquisition, Display

Class SldRequest : DataRequest	//	derived class from DataRequest
This class is not exported to the public; it is used by DataCollector		
Class DataCollector		
DataCollector::settimeout	//	set timeout period
DataCollector::setup	//	set up list
DataCollector::get	//	request data, wait until it arrives
Datacollector::getimmediate		
Datacollector::getsynchronized		
Class GraphMonitor : Monitor		
GraphMonitor:resize	//	resize the graph
GraphMonitor::title	//	display routines
GraphMonitor::writelabel		
GraphMonitor::writecycle		
GraphMonitor::hardcopy	//	hardcopy to printer

Table V. Class Samples - Device Tools

Class FunctionGenerator		
FunctionGenerator::menu_edit	//	edit function
FunctionGenerator::load	//	load it to devices
FunctionGenerator::readback	//	read the devices
FunctionGenerator::set_cld_names	//	names of complex-logical-devices
FunctionGenerator::set_default_value		
FunctionGenerator::set_start		
FunctionGenerator::set_end		
FunctionGenerator::set_timing_cld_names		
FunctionGenerator::set_tolerance		

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P01

Table VI. Class Samples - Accelerator Tools

```
Class  Instrument
    Instrument::calibrate
    Instrument::acquire_data
    Instrument::display_data
    Instrument::save_data
    Instrument::read_data
Class  HARP  :  Instrument                      //    multi-wire profile monitor
    HARP::insert
    HARP::retract
Class BPM  :  Instrument                        //    Booster Position Monitor
Class XF  :  Instrument                         //    Transformer
Class  MagnetCalibration
    MagnetCalibration::ReadCalibrationDataFile
    MagnetCalibration::Interpolate
    MagnetCalibration::ReadIvalues
    MagnetCalibration::ReadBvalues
Class  Transient Recorder
    TransientRecorder::GetLiveReadback
    TransientRecorder::SaveLiveReadback
    TransientRecorder::GetSavedReadback
    TransientRecorder::DisplayReadback
```

Table VII. Class Samples - Accelerator Physics Tools

```
Class  ManualHarmonicsCorrector  :  OrbitCorrector
    ManualHarmonicsCorrector::set_harmonic
    ManualHarmonicsCorrector::set_pue_display
    ManualHarmonicsCorrector::display_setpoint_harmonics
    ManualHarmonicsCorrector::display_readback_harmonics
    ManualHarmonicsCorrector::increment_coefficient
    ManualHarmonicsCorrector::execute_correction
Class  BoosterOrbitBump
    BoosterOrbitBump::magnet_device_list
    BoosterOrbitBump::pue_device_list
    BoosterOrbitBump::what_bump_order
    BoosterOrbitBump::what_bump_type
    BoosterOrbitBump::magnet_readbacks
    BoosterOrbitBump::magnet_measurements
Class  TuneModel
    TuneModel::WriteTuneIntoSetpoints           //    Send setpoints to devices
    TuneModel::ReadSetpointsIntoTune            //    Read setpoints from devices
    TuneModel::StartMad                         //    Run modeling program MAD
    TuneModel::TestMadDone
    TuneModel::GetTwissAtElement                //    Get Twiss params from model
    TuneModel::DisplayTwissAtElement            //    Popup Twiss param display
    TuneModel::DrawBeamLine                     //    Iconic display of beam line
    TuneModel::DrawEnvelope                     //    Draw beam envelope
    TuneModel::DrawAperture                     //    Overlay magnet apertures
    TuneModel::DrawPhaseEllipseAtElement
```

ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S1400P01

accelerator. This opportunity for class design to offer a clean interface to accelerator components is characteristic of these accelerator tools. With proper class design, a high-level program can be coded to read as cleanly as the designer's statement of the program function.

Acknowledgments

The work discussed here was developed over the last two years by the entire staff of the Controls Section of the AGS; their contributions made this report possible, and their cooperation made the work a pleasure.

Reference

1. W.T. Weng, Construction and Early Commissioning Results of the AGS Booster, 1991 Particle Accelerator Conference (in press).