Object Oriented Programming for Online Systems at H1

J. Coughlan
Rutherford Appleton Laboratory, UK

D. Düllmann
University of Hamburg, Germany

M. Savitsky
M. Zimmer
Deutsches Elektronen Synchrotron, Germany

The H1 experiment at HERA is currently recording collisions between 27 GeV electrons and 820 GeV protons. H1 consists of a dozen large subdetectors, which together provide over a quarter of a million electronic channels. The experiment has a correspondingly complex online system to control and monitor the data acquisition, triggering and slow controls of the entire apparatus. This paper describes the application of Object Oriented Programming (OOP) in several areas of the H1 online system. Emphasis is placed upon the experience acquired with this novel programming technique and the associated commercial software tools. The benefits obtained from the application of OOP concepts, such as data abstraction and class inheritance, in our problem domain are discussed.

1. H1 Online System

The H1 experiment [1] at HERA is composed of a dozen major subdetector components. The online system of a typical subdetector consists of a multi-microprocessor based system embedded in an electronics bus (typically VMEbus) [2]. The tasks of such systems include data acquisition, triggering and the control of apparatus parameters such as high voltages. System steering and monitoring is carried out by modern cost effective personal workstations (typically Macintosh™ computers) interfaced to the bus and exchanging information with the bus based microprocessors via shared memory. The workstations also serve as a platform for the development of most of the software. The desired loose coupling of the microprocessors is demonstrated by the fact that the online system remains fully operational even if the controlling workstations are switched off.

This interprocessor configuration, while enforcing strict adherence to locally defined communication protocols, allows a wide choice of languages and development platforms to be employed by software developers. The software for the bus and workstation based processors can be produced by independent developers using different languages and software development systems. The following discussion is devoted to the workstation based software.

2. Motivation for Object Oriented Programming

The workstation based steering and monitoring software is designed to hide the inherent complexity of the online system from the physicist. User driven applications with clear graphical user interfaces (GUI) exploiting the high resolution graphics of the workstations are employed. Reducing the complexity for the end user implies increasing complexity for the software.

The software must be sufficiently flexible and extendible to cope with apparatus configurations as they develop during the lifetime of the experiment. It should be possible to produce fast prototypes for testing these new arrangements. It must also be reliable, since it is
important not to loose control and monitoring of the online systems whilst the experiment is taking data.

The implementation of a GUI based application in a high level language is a considerable task even for an experienced developer familiar with a given workstation. Therefore, the first prototype applications [3] made use of commercial interface builders, e.g. SuperCard™[4]. These environments allow the layout of the user interface to be designed graphically. Associated simple scripting languages attach behaviour to the GUI elements. This approach yields the benefits of fast prototyping and is ideal for small test configurations. However, it has the disadvantages associated with the poor performance of an interpreted language which provides no data structures or memory management. In order to circumvent these problems, it must be supplemented with high level language add on code segments. The result is that larger applications soon become messy and difficult to manage.

The primary motivation for the adoption of Object Oriented Programming (OOP) by the authors was the recent availability of commercial OOP class libraries [5][6]. These provide frameworks for fully customizable workstation applications. The resulting applications incorporate all the behaviour of a high quality commercially produced GUI. The developer is relieved of the burden of managing the GUI and can concentrate on the application specific part of the project. The generated applications are also guaranteed to be compatible with future workstation hardware and operating systems. The associated development systems also provide many useful tools, such as GUI view editors, code browsers and powerful debuggers.

The OOP approach [7] produces software that is reliable, flexible and reusable by using the techniques of data abstraction, class inheritance and dynamic binding. It also encourages the use of fast prototyping. The software system is viewed as a collection of cooperating objects modelled on the objects existing in the problem domain. Commonality of behaviour is captured by arranging the objects in a hierarchy of classes. The following examples show how some of these ideas and techniques have been employed by the authors.

3. Examples of OOP Applications

The following four H1 online systems made use of OOP techniques for their workstation based software projects:

a) Central Data Acquisition Run Control
b) Central Trigger System
c) Drift Chamber Slow Controls
d) Backup Data Logging System

Project a) was implemented using the THINK Pascal™ class library [6] together with the THINK Pascal™ language. Projects b), c) and d) used the MacApp™ class library [5] with the "C++" language within the MPW™ development environment. All four projects have made use of their chosen application frameworks to provide the generic Macintosh™ application behaviour. Each project has then been customized to provide its own unique behaviour, again using OOP techniques.

The aim of project a) is to provide a user friendly interface, suitable for non-experts, to the experiment's run control. The emphasis is placed upon hiding the complexity of the data acquisition system by guiding the user through a series of panels which allow her to easily configure the running conditions of the experiment. This project therefore makes heavy use of inheritance from the provided GUI library classes to customize the appearance of the application.

The other three projects interact more directly with the electronics of their respective systems. The choice of the "C++" language was dictated by a need to share data structures with software written in "C" existing on the bus based microprocessors.

Project b) has extended the application of OOP techniques beyond the GUI to describe its system hardware. The central trigger system comprises of several custom VME boards managed via a VME based 68K processor. The project has introduced a class hierarchy to
describe the hardware boards. An abstract base “VME module class” provides the common behaviour expected from all VME boards. Boards with specialized functionality are then represented as subclasses of this base class. The software is designed to adapt to an evolving hardware system. Extensive use is made of collections of hardware modules implemented using the library list classes. The use of dynamic binding means the actual type of a given module in a list does not need to be known until run time.

Project c) provides for detailed monitoring of the hardware parameters and calibration constants of the H1 central drift chambers. It uses a hierarchy of classes representing the hardware modules of its system. It incorporates histogramming abilities in its status display package.

Project d) caters for the control and monitoring of cartridge drive systems with automatic loaders. A class was designed which gives a graphic representation of the state of any given cartridge in the system. This class also inherits control behaviour so that the user may click on screen to select a cartridge for a given operation. A collection of such objects is managed by another class which provides the functionality to read the hardware status and inform the cartridges to update their status displays accordingly.

4. Comments and Conclusions

For each of the four authors this has been their first experience with OOP. They have found the OOP approach to be a natural one for software development. The effort invested in learning the new techniques is more than repaid in the production of reliable and flexible software. Reuse was clearly made of the code in the application framework class libraries, but it was found that designing one’s own code for reuse requires a considerable amount of care and experience at the design stage.

It should be emphasized that although the Macintosh™ was used in the examples presented the discussion is applicable to any workstation platform. Platform independent class libraries also exist [8].

All four authors chose OOP languages which include strong type checking at compile and link times in order to produce reliable run time systems. In several cases the choice of language was determined by a desire to be compatible with existing software. The authors observed little or no performance reduction due to the invocation of class methods compared to normal function calls.

OOP demands good support from tools, such as class browsers and object level debuggers. The OOP approach leads to many small class based program modules and the authors using “C++” would have benefited from faster incremental compilers.

The use of application frameworks also entails a different attitude to programming in which the framework is in charge and calls the developers code (in the form of inherited classes) when needed and not vice-versa. Sufficient time must be invested to acquire a good working knowledge of ones chosen class library.

In conclusion, it has been shown that the techniques of OOP are applicable to the problem domain of workstation based control and monitoring applications for online systems. In particular, the use of commercial OOP class libraries to provide frameworks is ideally suited for producing customized GUI based applications.

References

[1] H1 Collaboration, Technical proposal for the H1 Detector, DESY, Hamburg, Germany, March 1986


W.J. Haynes, Experience at HERA with the H1 Data Acquisition System, this conference proceedings
   STAR DIVISION GmbH, *StarView - A Portable C++ Class Library for Graphical User Interfaces*, Lüneburg, Germany