Abstract Interfaces for Data Analysis - Component Architecture for Data Analysis Tools

G.Barrand (LAL, Orsay, France)
P.Binko (CERN, Geneva, Switzerland)
M.Dönszelmann (CERN, Geneva, Switzerland)
A.Johnson (SLAC, Stanford, USA)
A.Pfeiffer (CERN, Geneva, Switzerland)

Abstract

The fast turnover of software technologies, in particular in the domain of interactivity (covering user interface and visualisation), makes it difficult for a small group of people to produce complete and polished software-tools before the underlying technologies make them obsolete.

At the HepVis '99 workshop, a working group has been formed to improve the production of software tools for data analysis in HENP. Beside promoting a distributed development organisation, one goal of the group is to systematically design a set of abstract interfaces based on using modern OO analysis and OO design techniques.

An initial domain analysis has come up with several categories (components) found in typical data analysis tools: Histograms, Ntuples, Functions, Vectors, Fitter, Plotter, Analyzer and Controller. Special emphasis was put on reducing the couplings between the categories to a minimum, thus optimising re-use and maintainability of any component individually.

The interfaces have been defined in Java and C++ and implementations exist in the form of libraries and tools using C++ (Anaphe/Lizard, OpenScientist) and Java (Java Analysis Studio). A special implementation aims at accessing the Java libraries (through their Abstract Interfaces) from C++.

This paper gives an overview of the architecture and design of the various components for data analysis as discussed in AIDA.

Keywords: Abstract Interfaces Data Analysis OO Architecture Design

1   Introduction

A characterisation of progress in software development has been the regular increase in levels of abstraction. As the size and complexity of software systems increases, the design problem goes beyond algorithms and data structures: designing and specifying the software architecture, that is the overall system structure, emerges as a new kind of problem.

This is especially true in the case of high energy physics experiments, where the variety of user requirements and the complexity of the problem domain often involves the collaboration of several frameworks, and different components are responsible for providing the functionalities related to each domain.

For the domain of data analysis an attempt is being made by the AIDA [1] group to come to a set of common interfaces for the various categories involved. In this paper we concentrate on the architectural and design aspects of the *Abstract Interfaces* of each of these components.

2 Architectural Overview

Following D. Roberts et al., we define a *framework* as "reusable designs of all or part of a software system described by a set of abstract classes and the way instances of those classes collaborate" [2]. If the frameworks are designed such that they can evolve, and can be easily reused, adapted and configured, they usually create an increased productivity, shorter development times, and higher quality of applications [3].

A *component* is defined as a correlated group of classes together with their interactions. The grouping is usually determined during the domain decomposition phase of OO analysis. Each component (or category) typically depends only very weakly on (classes from) other components. This coupling can even further be reduced by using *Abstract Interfaces*, i.e., classes with only pure virtual methods, as the interfaces for the components. Any remaining references to (parts of) other components are then completely independent of the actual *implementation* of the referred component.

This *weak coupling* between the individual components of a complex system allows independent evolution of each component and therefore a much reduced maintenance overhead. Furthermore it also maximises flexibility of the system as a whole: an implementation of a component can be easily switched with another one, if dynamically loadable libraries are used even at runtime, providing the user of the system with a wide range of possible alternatives and options for customisation.

Special care has to be taken, that further evolution cycles of the system do not introduce new couplings between the components. In case a coupling seems unavoidable, the introduction of helper classes, typically in the form of one or several of the Decorator, Facade or Mediator pattern [4], could help to keep the independence.

The use of Abstract Interfaces for the frameworks used by a specific implementation of the analysis system ensures that the internal details of it are not exposed (or imposed on) the framework using the analysis system. In addition, it also allows to interchange (parts of) the analysis system without the need to recompile the code; if the implementation is done using dynamically loadable libraries, this could be done even at run-time.

In an attempt to avoid the strong coupling of the Interfaces to a specific implementation language, the Interfaces are described in Java (as Java Interfaces) as well as in C++ (as pure abstract classes).

3 Categories for Data Analysis

The components identified in the initial domain analysis and their interconnections (on the level of *using* only the Abstract Interfaces of another category) is shown below and can be classified into different groups:

- Data categories (Histograms (binned), Ntuples/Tags, Vector/Cloud (unbinned histograms), Functions)
- Plotter and Fitter
- Analyzer/Event-Display
- Controller

On the most "basic" level, the Data categories define the behaviour of data entities typically used in high energy physics experiments. These categories do normally not "use" more than one of the other components.

The Plotter and Fitter components are typically "using" more than one of the Data categories, therefore are considered on a "higher" level in the design hierarchy.

The Controller finally defines the behaviour of the interactions between the components (e.g., how to show a plot of a fitted histogram) and therefore "uses" most of the components.
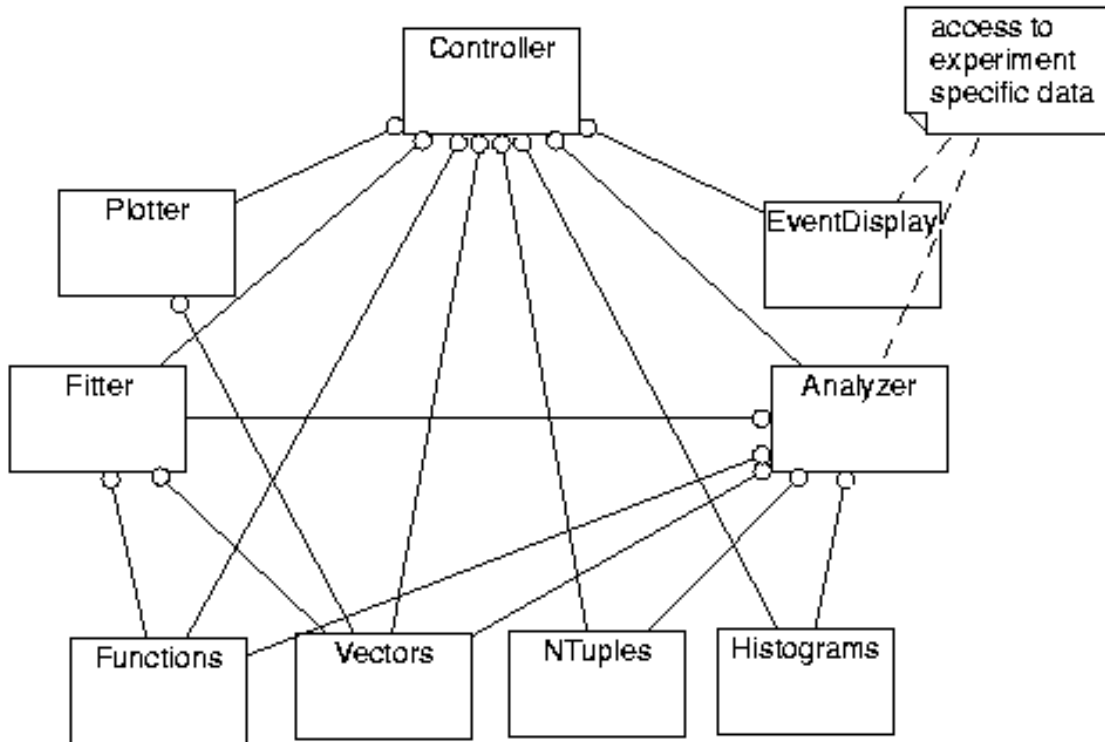
Figure 1: Categories identified for Data Analysis

This behaviour is typically found in the *Facade* Pattern [4].

Analyzer and Event-Display play a special role here, as they are the "glue" categories which allow the user of an implementation of AIDA to make full use of the experiment specific libraries and data model.

4   C++ and Java Implementations

Presently there are three groups of developers working on Analysis Systems implementing the AIDA Interfaces.

Two of the projects, Lizard [5] and OpenScientist [6], are based on C++ for the implementation of the interfaces and are using existing scripting languages (like Python, Tcl and Perl) and/or a graphical user interface for the communication with the user.

The Freehep Java Library [7] contains an implementation of the AIDA interfaces in Java. This implementation can be used in a standalone Java program, and will also be used by Java Analysis Studio (JAS) [8] to give users the ability to use the AIDA interfaces in their JAS analysis modules. A separate project provides access to the Java implementation of AIDA from C++, so that C++ programs using the AIDA interfaces can also choose to use the Java implementation of AIDA.

In a different context the simulation toolkit Geant-4 [9] recently has started to use the

AIDA Interfaces for their analysis category. This allows a developer of a Geant-4 application to write the code dealing with analysis objects *independent* of the actual implementation of the analysis software; therefore allowing to choose any of the three presently existing implementations (JAS, Lizard/Anaphe and OpenScientist) for their work.

5   Summary

In the design of complex software as used by many modern high energy experiments, well defined components with Abstract Interfaces defining their behaviour provide an strong increase of functionality and flexibility at very little additional cost for the developer of an experiment's framework.

The standardisation of a set of interfaces for components commonly used for data analysis is the aim of the AIDA project. The existence of standards for the Interfaces will not only allow for a common "look-and-feel" at the level of the interactive session; it will also allow to interchange existing implementations of these Interfaces, creating a significant push towards larger flexibility and software re-use.

References

[1] The AIDA group; see `http://aida.freehep.org`

[2] Don Roberts, Ralph Johnson, University of Illinois, "Evolving Frameworks", `http://st-www.cs.uiuc.edu/~droberts/evolve.html`

[3] Dirk Bäumer, Guido Gryczan, Rolf Knoll, Carola Lilienthal, Dirk Riehle and Heinz Züllighoven, "Framework Development for Large Systems." Communications of the ACM 40, 10 (October 1997). Page 52-59.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: elements of reusable object-oriented software", Addison-Wesley, 1995

[5] Lizard; see `http://cern.ch/Anaphe/Lizard`

[6] OpenScientist; see `http://www.lal.in2p3.fr/OpenScientist`

[7] The Java Freehep Library; see `http://java.freehep.org`

[8] Java Analysis Studio; see `http://jas.freehep.org`

[9] Geant-4; see `http://cern.ch/Geant4`