CERN-IT-2001-012
September 21, 2001

Anaphe – OO Libraries and Tools for Data Analysis

O.Couet (CERN, Geneva, Switzerland)
B.Ferrero-Merlino (CERN, Geneva, Switzerland)
Z.Molnár (CERN, Geneva, Switzerland)
J.T.Mościcki (CERN, Geneva, Switzerland)
A.Pfeiffer (CERN, Geneva, Switzerland)
M.Sang (CERN, Geneva, Switzerland)

Abstract

The Anaphe project is an ongoing effort to provide an Object Oriented software environment for data analysis in HENP experiments. A range of commercial and public domain libraries is used to cover basic functionalities; on top of these libraries a set of HENP-specific C++ class libraries for histogram management, fitting, plotting and ntuple-like data analysis has been developed. In order to comply with the user requirements for a command-line driven tool, we have chosen to use a scripting language (Python) as the front-end for a data analysis tool.

The loose coupling provided by the consequent use of (AIDA compliant) Abstract Interfaces for each component in combination with the use of shared libraries for their implementation provides an easy integration of existing libraries into modern scripting languages thus allowing for rapid application development. This integration is simplified even further using a specialised toolkit (SWIG) to create "shadow classes" for the Python language, which map the definitions of the Abstract Interfaces almost at a one-to-one level.

This paper will give an overview of the architecture and design choices and will present the current status and future developments of the project.

Keywords: OO Architecture Design HEP Data Analysis Interfaces Components Tools

1 Introduction

The Anaphe project was set up (with the old name of LHC++) to replace the full suite of functionality formerly provided by the FORTRAN based CERNLIB [1] in the new object-oriented computing environment required by the experiments.

The aim of the project is to provide a flexible, inter-operable, customisable set of interfaces, libraries and tools, re-using existing (public domain or commercial) packages wherever applicable and possible. HEP specific adaptions are written where needed. In close collaboration with the experts from the experiments and other common HEP projects (like Geant-4 [2] and CLHEP [3]), these packages are designed considering the huge data volume expected for LHC, the distributed computing necessary to analyse the data as well as long-term evolution and maintenance.

The Anaphe [4] software suite consists of a set of individual class libraries for data storage, fundamental numerical mathematics and physics data analysis (histogramming, ntuples and minimisation/fitting) which exist since several years in their present form.

More recently, the development of packages for visualisation and a command-line driven interactive data analysis tool (Lizard) has started. Based on a user requirement document (URD) the initial functionality of the interactive data analysis tool was implemented and the first release with full functionality is available since end May 2001.

Using object oriented methodologies and techniques, the domain of data analysis has been analysed and a set of categories (components) have been identified for which *Abstract*

Interfaces have be defined. These components contain the basic data structures used in physics data analysis like Histograms, Ntuples, Functions, Vectors as well as higher level components like Fitter, Plotter, Analyzer, Event-Display and Controller/Hub. Together with developers of similar tools and users from experiments, the AIDA [5] project has been initiated to standardise the *Abstract Interfaces* used.

2 Architectural overview

The architecture of Anaphe concentrates on a set of well-defined categories (components), each of them having a set of *Abstract Interfaces* describing the functionality of the category for use from "outside" classes. The *Abstract Interfaces* used here are classes which have *only* pure virtual methods (and an in-line dummy destructor) and are allowed to only inherit from other *Abstract Interfaces*. This allows the development of each category to be completely independent of other components, therefore de-coupling the components. In addition, it allows to speed up the development, as existing class libraries can be re-used (possibly through a wrapper).

2.1 Components and Collaborating Frameworks

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.

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

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

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 (or Hub) 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. This structure is typically found in the *Facade* Pattern [6].

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.

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.

This *weak coupling* between the individual components of a complex system allows for independent evolution of each component and therefore a much reduced maintenance overhead. Furthermore it also maximises flexibility of the system as a whole: implementations of components can be easily switched, 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.

3 Libraries for Data Analysis

Each of the components of the Anaphe suite described above is implemented in the form of one or more individual libraries. As these libraries are typically created in the form of shared libraries, an application can change the actual implementation by loading – at run-time – a different library implementing a given component.

Presently, for some categories *two* independent implementations exist, e.g., for the Fitter/Minimiser package (one based on algorithms from the Nag-C [7] library, the other based on the CERNLIB-Minuit package) as well as the Histogram and Ntuple packages (where the persistent store is either based on Objectivity/DB [8] or CERNLIB-HBook).

In the future it is planned to extend the variety of implementations of the categories.

4 Data Analysis Tool

Based on the components described above, an interactive tool for data analysis (Lizard) has been developed. We have chosen the approach to use a scripting language (Python) for the user interface, as scripting languages can easily be used for "gluing" together components into applications. The use of type-less approaches in the scripting language to achieve a higher level of programming and more rapid application development than programming languages. Increases in computer speed and changes in the application mix are making scripting languages more and more important for applications of the future [9].

4.1 Python and SWIG

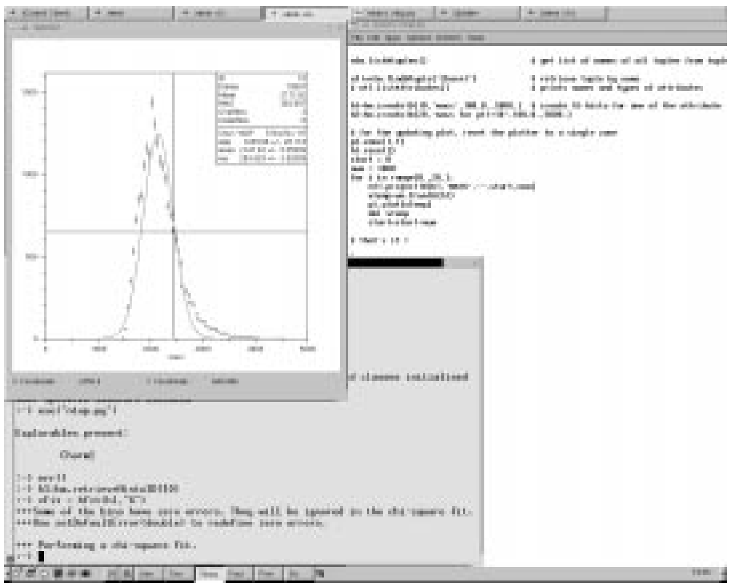
Python [10] is a real OO programming language, offering much more structure and support for large programs. On the other hand, it also offers much more error checking than for example C, and, being a very-high-level language, it has high-level data types built in, such as flexible arrays and dictionaries, similar to the STL in C++. In addition, Python is an object-oriented language, therefore it is easy to enhance functionality of a given class by using well known OO techniques such as inheritance.

The ease and flexibility of Python is complemented by another tool (SWIG) [11] which intends to provide a mechanism for building interactive programs that is extremely easy to use. SWIG is a simple tool for building interactive C, C++, or Objective-C programs using common scripting languages such as Tcl, Perl, and Python. While SWIG was originally developed for scientific applications, it has become a general purpose tool that is being used in an increasing variety of other computing applications.

4.2 Lizard

An interactive tool for data analysis has been created using Python as the interactive "language". To implement the required functionality, basically all interfaces from the components described above have been "converted" into Python classes using SWIG (the Python classes are mainly "proxies" to the C++ objects they handle). At startup time, the corresponding (shared) libraries implementing the various components are dynamically loaded and the user has full access to objects of the components. This way, Python acts as a "glue" or "framework", the user can select (e.g., via command-line options) the required specific implementation and/or functionality, e.g., choice of Minimiser/Fitter or persistence scheme for Histogram and Ntuple (or decide to not use a component at all). Future versions will allow to have several independent implementations of a component accessible at the same time ("plug-in-like" functionality).

A screen shot of an analysis session using Lizard is shown in the figure.



```
[illegible]
```

```
[illegible] classes initialized
[illegible]
>>> [illegible]

Explorables present:

        [illegible]

>>> [illegible]
>>> [illegible]
>>> [illegible]
***Some of the bins have zero errors. They will be ignored in the chi-square fit.
***Use setDefaultError(double) to redefine zero errors.

*** Performing a chi-square fit.
>>>
```

Figure 1: Screen shot of a Lizard session

5 Summary

A set of well-defined categories (components), each of them having a set of *Abstract Interfaces* describing the functionality of the category, has been developed in the context of providing libraries and tools for physics data analysis. Each of these components has a (shared) library providing an implementation; for some components multiple independent implementations exist already. A project has been started (AIDA) to standardise these Interfaces.

Using the Python scripting language – in combination with the "conversion" tool SWIG – a very powerful and extremely flexible data analysis tool has been created allowing users to select a specific implementation for a given component at startup-time.

References

[1] CERN Program Library (CERNLIB); see `http://wwwinfo.cern.ch/asd/index.html`

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

[3] CLHEP; see `http://wwwinfo.cern.ch/asd/lhc++/clhep/index.html`

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

[5] See these proceedings and the AIDA project web page at `http://aida.freehep.org`

[6] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995

[7] Numeric Algorithms Group; see `http://www.nag.co.uk`

[8] Objectivity/DB; see `http://www.objectivity.com`

[9] See `http://tcl.activestate.com/doc/scripting.html`

[10] See http://www.python.org
[11] See http://www.swig.org