Review of the Abstract Interfaces for Data Analysis (AIDA) from a developer's perspective

Vincenzo Innocente, Lorenzo Moneta, Andreas Pfeiffer

CERN

Version (1.0) – 08 Oct. 2003

Editing History

Draft 0.0, July 23, A. Pfeiffer: First draft

Draft 0.1 July 27, V.Innocente: small changes

Draft 0.2 Oct. 08, A. Pfeiffer: final draft in preparation for internal review

Version 1.0 Oct. 08, A. Pfeiffer: finalize in preparation for internal review

Table of Contents

1 Introduction

This document will give a review of the present version (3.0) of the AIDA interfaces from the point of view of a developer of external frameworks. A recent AIDA workshop at CERN provided a significant input to this discussion and is a major resource for this review.

1.1 What is AIDA

The Abstract Interfaces for Data Analysis (AIDA) project defines today *user level* interfaces for some common analysis data objects: binned and unbinned histograms (IHistogram, IProfile, ICloud), free format data points (IDataPointSet), and tuples (ITuple), together with interfaces to some common facilities as Fitting and Plotting of these objects. In addition, there is a set of interfaces defined for the management of these objects (IAnalysisFactory, IFactories, ITree (IManagedObject)). This management has been modelled along the metaphor of the Unix file system.

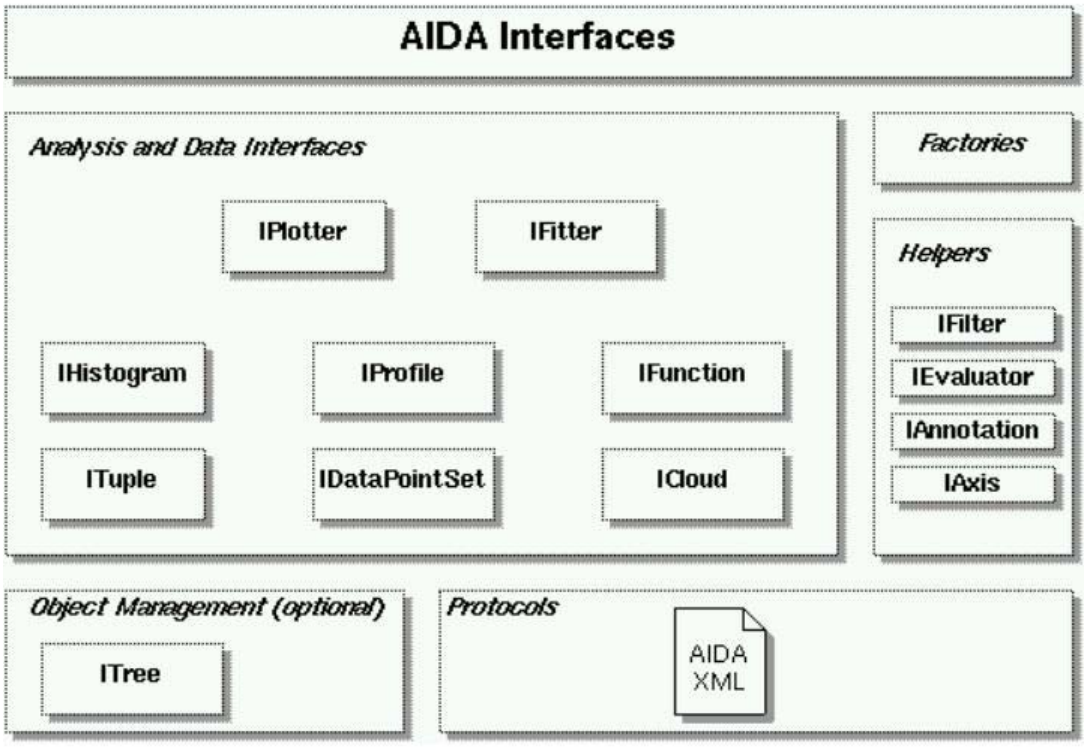

Figure 1: The AIDA interfaces and protocols

The adoption of these interfaces should make it easier for physicists to use different tools without having to learn new interfaces or change all of their code. Additional benefits will be interoperability of AIDA compliant applications by making it possible for applications to load different implementations of a given component at runtime and to exchange analysis objects via XML between applications.

Finally by bringing together the authors of a number of different applications in the AIDA context, it is expected to gain deeper insight into the strengths and weaknesses of different applications, and to improve collaboration between different authors.

1.2 AIDA development process

The development in the AIDA project started by defining Abstract Interfaces to allow interoperability on the *user-level*, i.e., users can interchange implementations of a given version of the AIDA interfaces without recompiling their code.

A common set of functionality as an agreed superset of the existing analysis tools was identified and discussed. This discussion started from existing implementations, some of which were already based on a (different) set of abstract interfaces. The resulting common set of AIDA Interfaces therefore has a significantly larger functionality than simply using the "least common denominator" of the analysis tools.

From the beginning the AIDA team was aiming to be able to read/write various different storage formats such as AIDA-XML, Objectivity/DB, HBOOK, ROOT, SQL. In order to hide the complexity of the various storage systems from the user, it was decided to employ a simple string based interface where the user simply specifies strings for the name and the type of the storage and the implementation then handles the complex interactions with the store. This functionality has been "co-located" with the object management component of AIDA (ITree).

In the next step it is foreseen to improve the interoperability at *component* level , allowing the users to mix components from different implementations at runtime. This way, a user can choose which implementations to use for any given component and developers can implement *individual components* using the technology of their choice without any constraint nor dependency on the implementation of other components.

1.3 AIDA Implementations

So far, there were three independent groups implementing the AIDA interfaces. The Anaphe project at CERN (which is no longer in active development) and a group at LAL provided direct implementations in C++ while a team at SLAC implemented the interfaces in Java and provided a "glue layer" to use the Java implementation from C++.

All teams provide a rather complete implementation of the AIDA 3 interfaces although some limited functionality may still be present in some classes.

Bindings to Python have been developed in all three groups; however, the necessary extensions to the interfaces have not yet been standardized.

Recently the LCG PI team as released an implementation of IHistogram and IProfile based on the corresponding ROOT objects. In addition, the capability to store to and retrieve from the AIDA histograms and profiles from ROOT files has been implemented in the PI project. The extension of this work to a full AIDA implementation based on ROOT is under discussion in the application area of LCG.

2 Developer's view

This section deals with the AIDA interfaces from the viewpoint of a developer of a framework or application using the interfaces. It does not cover the issues related to the implementation of one (or several) of the AIDA components.

2.1 Proxy classes for value semantics

As a study to improve acceptance of the AIDA interfaces, the LCG PI project has started to create proxy classes to (initially a subset of) the AIDA interfaces, providing value semantics to the user and hiding the object management (ITree). The package is based on the AIDA interfaces only, without any dependency on a specific implementation. The classes "implement" the corresponding AIDA interfaces and add the constructors taken from the create-methods of the factories.

In addition, the package allows to load different implementations of the component at run-time in a fully transparent way using a plug-in mechanism. Early feedback resulted in creating an additional Proxy_Store class handling the storage and retrievals of objects (copy in/out semantics). This package is planned to be used in the evaluation of the AIDA interfaces during the foreseen user review. Examples on how to use the proxy layer in C++ code are available from the PI web pages (http://lcgapp.cern.ch/project/pi/) or directly at the following URL: http://lcgapp.cern.ch/project/pi/Examples/PI_0_4_1/index.html

2.2 Allow for creation of unmanaged objects

In the actual version of AIDA (3.0), objects created by a factory are automatically connected to an instance of ITree, which is managing the objects. While this is certainly a good design for stand-alone analysis tools and needed in an GUI environment to allow a coherent view and handling of the objects, it makes the integration of AIDA into other frameworks which have their own object management system more difficult. It is therefore desirable to have a way in AIDA to create unmanaged objects of any type; this could either be done at the developer-level interfaces or through a (transparent to the user) change of semantics of the existing user-level interfaces (e.g., through accepting a null-pointer as input for the ITree).

2.3 Component-level interoperability

One of the initial aims of AIDA was to provide a modular toolkit of components with a set of alternative implementations from which the user can select at run-time according to the functionality required for the task (e.g., a specific implementation of a fitter or plotter together with a histogramming package optimised for memory usage). In order to provide this flexibility, an additional set of Abstract Interfaces needs to be defined which will then be used by the existing user-level AIDA interfaces and decouple the components further. As this layer will be used by developers implementing a given component, these interfaces are called "developer-level interfaces".

In addition to this set of developer-level interfaces, a mechanism to identify at run-time which library to load and how to instantiate the corresponding factories (the "loading mechanism") needs to be defined. Initially this could be done using a file with an XML description, using a common naming scheme for libraries and factories, or a combination of both.

The AIDA team has set end October as a milestone for a concrete demonstration of this component-level interoperability. By then a prototype should demonstrate that (at least) histograms, plotting and fitting can be used from different implementations (OpenScientist, Anaphe and JAS). Once this is achieved, the developer-level interfaces will be reviewed and published.

2.4 Object Management and Store

Discussions with the developers of the LCG SEAL and POOL projects will start with the aim to standardise the interfaces (again at developer-level) to object management (whiteboard) and object store (persistency). These discussions should give additional feedback on the requirements from framework developers.

In conjunction with the object management issues, a potential sub-framework handling issues such as observing/notification between objects will be discussed. This functionality is needed in various places. For example a Histogram might notify all observers once it's content has been updated; a Plotter then would update the view of the histogram, a streaming store might "push" the new data through a network channel to another recipient.

2.5 The ITuple interface

The interfaces to the tuple component, while considered adequate for the user-level, is deemed too wide for the developers. A proxy layer here could allow for a narrower developer interface used by implementers of the tuple while keeping the full functionality for the user. This layer could be provided by the PI project.

2.6 Users of AIDA

So far, users of AIDA are mainly developers as for example the Geant-4 advanced examples and the LCG PI project. As some of the interfaces are also used in various frameworks of experiments, such as the Gaudi and Athena offline framework of LHCB, Harp and ATLAS at CERN, the online environment of BaBar and the Linear Collider users at SLAC, the number of "end-users" is not easy to estimate. It is certainly felt that feedback from the users in the experiments is important and an effort will be made to encourage this feedback. In a first round of interviews with users in the LHCb and CMS experiments, the users were satisfied with the overall functionality and completeness of the interfaces. The flexibility, both concerning the choice of implementations and the ability to read various storage formats – without changes in the code – was felt to be very important.

3 Summary

The Abstract Interfaces for Data Analysis (AIDA) project defines today *user level* interfaces for some common analysis data objects, together with interfaces to some common facilities as Fitting and Plotting of these objects. These interfaces are aiming to allow interoperability on the *user-level*, i.e., users can interchange implementations of a given version of the AIDA interfaces without recompiling their code.

A common set of functionality as an agreed superset of the existing analysis tools was identified and discussed, starting from existing implementations. The resulting common set of AIDA Interfaces therefore has a significantly larger functionality than simply using the "least common denominator" of the analysis tools. From the beginning the AIDA team was

aiming to be able to read/write various different storage formats, hiding the complexity of the various storage systems from the user through a simple string interface.

Today three implementations exist, two implemented in C++, one in Java (with a "glue layer" to use the Java implementation from C++). Bindings to the Python language have been developed as well, the standardisation of the necessary extensions to the interfaces is in progress. The LCG PI team at CERN has released a partial implementation of the interfaces based on wrappers to the corresponding ROOT objects (IHistogram and IProfile), including the capability to store and retrieve these objects from/to ROOT files.

As part of a study to improve acceptance, the LCG PI project has created a "Proxy layer" on top of the existing AIDA interfaces. This layer is independent of the actual implementation; in fact, various implementations of the AIDA interfaces can be used – fully transparent to the user – simultaneously in the same application through employing a plug-in mechanism.

Discussions with the developers of other LCG projects will start to improve and standardise the developer level interfaces towards the functionality of object management and object store. In this context, also additional functionality concerning sub-framework object handling issues (observing/notification) will be discussed.

A proxy layer for the tuple component, narrowing the interface as seen by a provider of the component, is under discussion in the PI project.

Initial feedback from users of the AIDA interfaces is being collected. In general, the users are satisfied with the functionality and completeness of the interfaces and positively acknowledged the flexibility provided by the use of the interfaces.