The LCG PI Project: Using Interfaces for Physics Data Analysis

A. Pfeiffer, L. Moneta, V. Innocente, H. C. Lee, and W. L. Ueng

***Abstract*—In the context of the LHC computing grid (LCG) project, the applications area develops and maintains that part of the physics applications software and associated infrastructure that is shared among the LHC experiments. The "physicist interface" (PI) project of the LCG application area encompasses the interfaces and tools by which physicists will directly use the software, providing implementations based on agreed standards like the analysis systems subsystem (AIDA) interfaces for data analysis. In collaboration with users from the experiments, work has started with implementing the AIDA interfaces for (binned and unbinned) histogramming, fitting and minimization as well as manipulation of tuples. These implementations have been developed by re-using existing packages either directly or by using a (thin) layer of wrappers. In addition, bindings of these interfaces to the Python interpreted language have been done using the dictionary subsystem of the LCG applications area/SEAL project. The actual status and the future planning of the project will be presented.**

I. Introduction

WITHIN the application area of the LHC computing grid (LCG) [1] the "architectural blueprint" document [2] establishes the basic architecture of the common software to be developed. Any piece of common software developed in the LCG must conform to this coherent overall architectural vision; make consistent use of an identified set of core tools, libraries and services; integrate and inter-operate well with other LCG software and experiment software; and function in the distributed environment of the LCG.

In particular, the "blueprint" report proposes the adoption of the analysis systems subsystem (AIDA) [3] interfaces to data analysis components. With that, it is expected to facilitate the integration of existing implementations or adaptations and to provide continuity in their current use in the experiment frameworks. As a direct outcome of this, the PI project [4] was created in early 2003 to implement these aspects of data analysis for the physicists as described in the report, mandated to provide a consistent set of interfaces and tools by which physicists will directly use the LCG software. All subsystems designs follow an OO component model, as required by the "blueprint" document for projects in the LCG applications area (LCG AA). The PI project is also expected to deliver new implementations of the AIDA interfaces, or some subset of them; other projects in the LCG AA cover core utilities and basic services (SEAL project [5]) and the persistency services (POOL project [6]).

Manuscript received November 15, 2004; revised June 14, 2005.

A. Pfeiffer L. Moneta, and V. Innocente are with the CERN, CH-1211 Geneva 23, Switzerland (e-mail: Andreas.Pfeiffer@cern.ch).

H. C. Lee and W. L. Ueng are with the Academica Sinica, Nankang Taipei 11,5 Taiwan, R.O.C.

Digital Object Identifier 10.1109/TNS.2005.860150

A. The Analysis Services Subsystem

The initial priorities for the work plan of the PI project concentrated on the Analysis Services subsystem. The subsystem provides an (interactive) analysis environment (in collaboration with the SEAL project) based on the Python interactive language [7]. Work is starting to collaborate with the persistency project POOL on implementing the AIDA Tuple interfaces based on the "Collection" interfaces of POOL.

As part of the work, a review of the AIDA interfaces was done [8], and an implementation of the AIDA interfaces based on the corresponding ROOT (a C++ analysis tool) classes [9], has been developed using wrappers.

The activities of the PI project aim at providing a set of analysis tools which is compliant to the "architectural blueprint," as well as integrating externally contributed tools—like visualization tools and statistical analysis tools—which may or may not be based on the AIDA interfaces.

Emphasis is put on the aspect of (future) maintenance, the intent is to re-use existing software (mainly from sources external to the project), not to re-invent it. This way of not "incorporating" the software (i.e., copy as source into the project's software repository and maintaining it in parallel there) profits from the maintenance work done by the providers of the software, and therefore reduces the amount of resources needed for the much simpler task of providing the integration layer.

1) The AIDA Interfaces: The AIDA project started in autumn 1999, when an open collaboration of teams providing analysis software decided to create a common set of Interfaces for this domain. Presently there are three independent teams working in the collaboration: the JAS [10] group (SLAC), the PI project (CERN) and the OpenScientist [11] group (LAL/Orsay).

The AIDA group has released version 3.0 of the interfaces in October 2002, and has recently (October 2003) published a set of small changes leading to release 3.2.

Fig. 1 shows the various interfaces defined in AIDA. Clearly the basic functionality of providing the service classes for data analysis is visible. Additional functionality is defined for object management (ITree) as well as for object creation ("factories"). A set of "helper classes" deals with issues like fitting/minimization and annotating AIDA objects. Finally a DTD for AIDA has been agreed on to describe the format (or "protocol") of all AIDA objects in XML, allowing for easy interchange of AIDA objects between various implementations.

The PI project conducted a review of the AIDA interfaces involving users and developers from the LHCb, ATLAS, and CMS experiments. The large flexibility of changing the underlying implementations without changes in the user's code (due to the OO component architecture) was pointed out to be

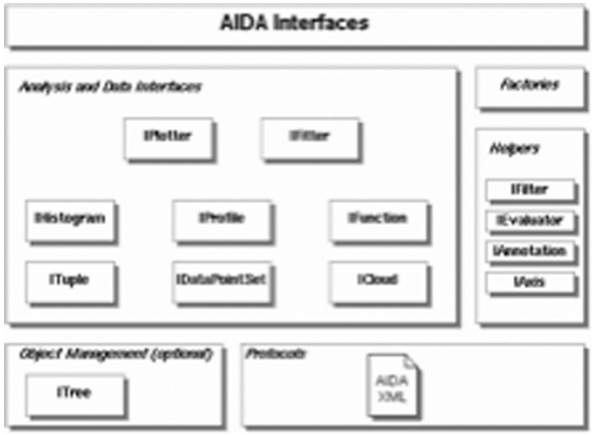

Fig. 1. Set of AIDA interfaces.

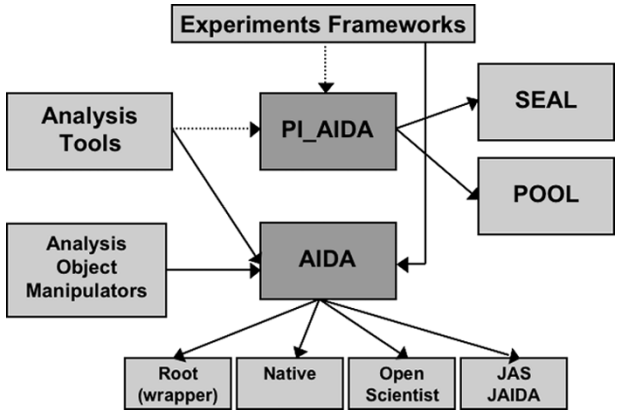

Fig. 2. Components of the PI_AIDA. The user typically accesses the analysis objects through the experimental framework, an analysis tool, or an analysis object manipulator (e.g., tuple or cloud projector). In all cases, the usage is through the AIDA interfaces (or their PI proxy counterpart), so there is no dependency on the specific implementation at compile or link time. The arrows denote run-time dependencies between the components (dotted lines: optional dependencies).

a major advantage of using the AIDA interfaces. Other advantages mentioned were the ease of integration in the experiment's frameworks and the clear separation of the functionalities into the corresponding components. This has lead to the adoption of the use of the AIDA interfaces (together with implementations provided by the PI project) into the software of all three experiments. For example, users in the ATLAS and LHCb experiments operate with AIDA histograms as delivered by the HistogramService of the Gaudi framework, tuple and function/fitting is used in the interactive environments (based on the Python interpreter) used in the experiment. In CMS the use of AIDA histograms and tuples is mainly at the level of the end-user software, there is no equivalent to the HistogramService in the core framework component.

2) The PI AIDA_Proxy Classes: Within the PI project, we have (based on one of the outcomes of the review of the AIDA interfaces) created a "simplified" layer of classes (PI_AIDA) on top of the AIDA interfaces, providing value-semantics but preserving the full functionality of the AIDA interfaces. This has been achieved by employing the Proxy pattern as described for example in [12]. For the implementation of the AIDA interfaces, previous work could be re-used for a direct implementation and for a wrapper to the HBook package of CERNLIB (for some classes). As stated above, a new wrapper to (some) ROOT classes was created in a rather straightforward way.

The Proxy classes exhibit full compatibility with the AIDA interface standard (i.e. they can be used wherever a method expects an AIDA Interface of the corresponding type) and at the same time are completely independent of the actual implementation of the AIDA interfaces used. This way the full flexibility of independence on the actual implementation from the user's perspective is preserved in a fully transparent way for the user.

The main advantage for the user is the ability to change the underlying implementation without having to change any part of the user's code. As an example, the same code will run in an on-line monitoring environment with an implementation of the AIDA interfaces optimized for memory usage and as well (even without recompilation!) in an offline environment where the implementation of the AIDA interfaces is optimized for high-precision statistics information. It is also possible to "mix" the various implementations allowing the user to select, for example, the implementation of the fitting component from a different "provider" than the histograms. This possibility of easily mixing various implementations is achieved through the use of the plugin-manager provided by the SEAL project. It is used in a singleton-like object ("Proxy_Manager"), which checks if the (user-) selected specific implementation is available (in the form of a plug-in module) and loads (and manages) the various implementations.

Fig. 2 shows the various building blocks of the proxy-layer provided by PI. The user interacts with it only through the use of AIDA interfaces, either directly or through their experiment's framework or an analysis framework or analysis object manipulators. The respective framework can then use either the AIDA interfaces directly or through the AIDA_Proxy layer provided by PI the framework or tool then selects (loads) one (or several) of the available implementations of AIDA for the execution of the user's code. The arrows indicate the run-time dependencies of the components. Each component builds completely independent, so there is no dependency at source level at all.

3) Unit Testing: An extensive test suite has been created based on the CppUnit testing framework [13] provided by the SPI project [14] of the LCG application area. Various test cases have been implemented checking not only the functionalities of the AIDA Proxy classes, but also the consistencies between the different histogram implementations. The framework has been designed such that new test cases can easily be added.

Presently a total of 1164 individual tests are implemented, only 104 of these tests show differences in the various implementations of the interfaces and are therefore (not quite correctly) reported as "failed." These differenced come from the fact that some implementations of the interfaces treat their internal data structures differently (caching versus noncaching) and therefore create slightly different results for some of the methods. The full test suite is run for every new release in order to ensure there was no regression introduced.

II. Integration With Other Tools

Another aspect of the work done in the PI project dealt with integration of AIDA based analysis objects with other tools, which are not "AIDA-aware."

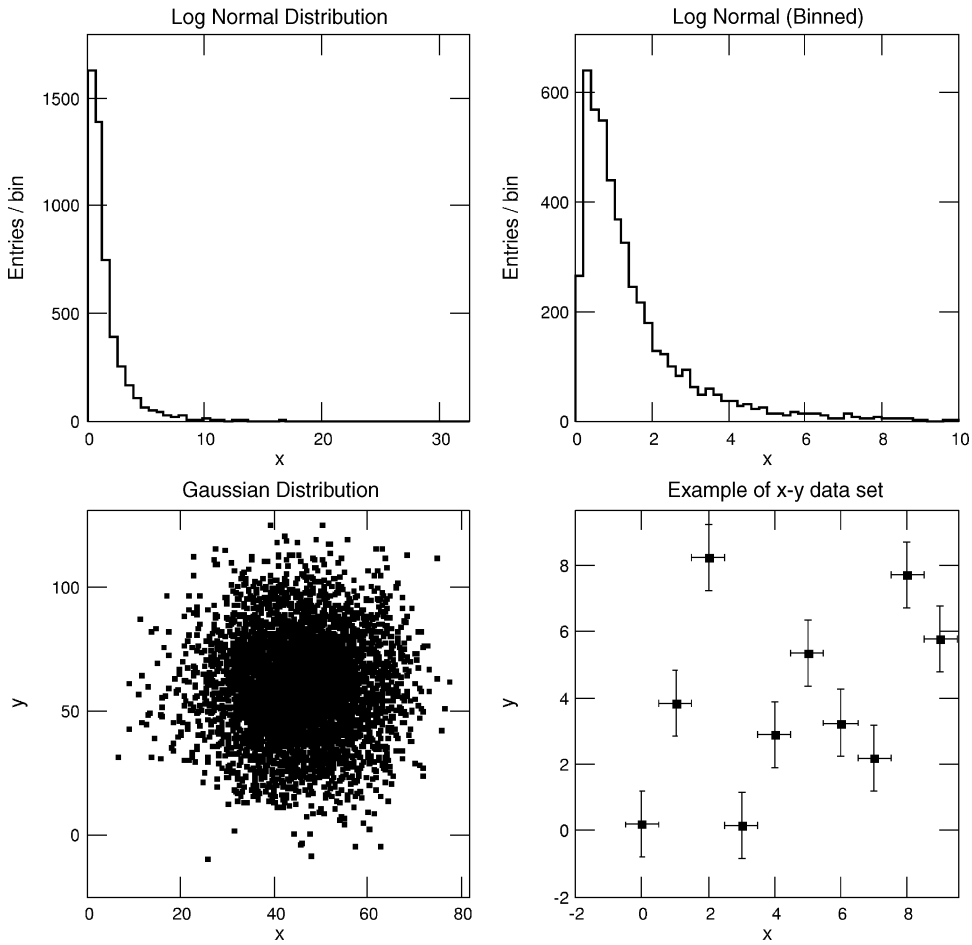

Fig. 3. Example of visualization of AIDA objects (binned and unbinned histograms and dataset) in HippoDraw.

A. *Visualization of AIDA Objects With HippoDraw and ROOT*

Two prototypes were created at the Python layer to validate the ease of integration of other data analysis tools with AIDA. Using the (Py)LCGDictionary from the SEAL project, bindings of the AIDA_Proxy classes (and the AIDA interfaces) to Python were performed, allowing the creation and manipulation of any of these objects in the Python language.

In a second step, simple Python classes were created to copy the AIDA objects into objects of the two chosen external analysis tools: HippoDraw [15] and ROOT. While HippoDraw already provides a binding to Python which is exploited here, the binding to ROOT was done using both the PyROOT package from SEAL (and now in ROOT) and the PyLCGDictionary binding to ROOT classes.

As HippoDraw can deal with unbinned histograms as well and since the corresponding classes are already available through the (Py)LCGDictionary, the extension to also visualize objects of these classes with HippoDraw was trivial. Fig. 3 shows examples of this.

The total effort needed to develop these integration prototypes was remarkably low: about a day for one developer in total. This is certainly due to the component-based architecture of AIDA and the ease of use of Python as an interpreted OO language.

B. *Cross Language (C++/Java) Interoperability*

Using the JAIDA/AIDAJNI [16] packages from the SLAC AIDA team it is also possible to use Java implementations of the AIDA interfaces from C++ programs. Again, the use of abstract interfaces allows this without any change in the user code by selecting the appropriate environment at run-time. An example of this is shown in Fig. 4.

C. *The Statistical Analysis Toolkit*

As an external contribution to the PI project, a toolkit to perform statistical comparison of binned and unbinned AIDA histograms is distributed with the PI software.

This package [17] provides tools for easy comparison of binned and unbinned histograms using a large set of algorithms, covering the well-known Chi-square test for binned histograms and Kolmogorov–Smirnov test for unbinned histograms as well as a number of other algorithms (Anderson–Darling, Cramer–von Mises, Lilliefors, Kuipers, and others), from which the user can choose in an easy and intuitive way. The package has a user-layer which is build on top of the layer implementing the actual algorithms, this way possible changes in the implementations can be done in a way which is completely transparent to the user.

Also a package for modeling parametric fit problems using a Monte Carlo approach is part of the contributed package.

III. Summary

The PI project provides Interfaces (and their implementations) for physicists in the form of a consistent set of tools, which

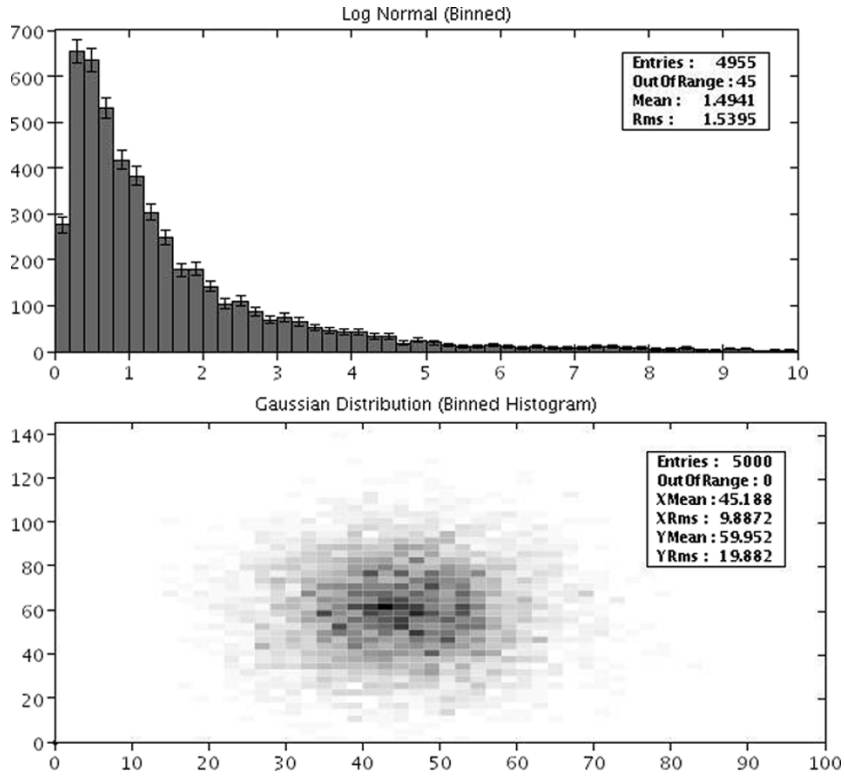

Fig. 4. AIDA objects plotted from a C++ program using the Java implementation of the IPlotter interfaces through the JAIDA/AIDAJNI package.

is compliant to the "architectural blueprint" of the LCG application area.

The Analysis Services subsystem has been developed based on the AIDA interfaces and following a component model, resulting in a flexible, plug-in controlled set of implementations. A large number of unit-tests reveal the differences in the underlying implementations.

A review of the AIDA interfaces has been conducted in conjunction with users from the LHC experiments; feedback is given to the AIDA collaboration.

Integration with external tools has been shown to be easily feasible; prototypes in Python have been developed showing that visualization of AIDA objects is possible from the same application inside ROOT, HippoDraw or JAIDA.

The project has developed its deliverables within the planned scheduling and released a first release in May 2003, followed by a first production version (1.0.0) in early October 2003, well on schedule with respect to the milestone planning. The latest release in late summer 2004 includes the change to the latest AIDA version (3.2) and the prototypes for the integration with external frameworks and tools.

Work is continuing mainly in the area of user support and maintenance; some development will be done in the area of interoperability with other AIDA implementations and integration with other (external) frameworks. Apart from this, the further evolution of PI will follow the overall planning as required by the LCG Applications Area.

REFERENCES

[1] *http://cern.ch/lcg* [Online]
[2] *http://lcgapp.cern.ch/project/blueprint/BlueprintReport-final.doc* [Online]
[3] *http://aida.freehep.org* [Online]
[4] A. Pfeiffer, V. Innocente, L. Moneta, and H.-C. Lee, "Status of the LCG Physicist Interface (PI) project," in *Proc. ACAT Workshop*. Tsukuba, Japan: KEK, Dec. 2003, vol. 534, Nuclear Instruments and Methods, p. 106.
[5] R. Chytracek, J. Generowicz, W. Lavrijsen, M. Marino, P. Mato, and L. Moneta *et al.*, "Status of the SEAL project," in *Proc. ACAT Workshop*. Tsukuba, Japan: KEK, Dec. 2003, vol. A 534, Nuclear Instruments and Methods, p. 115.
[6] I. Papadopoulos, R. Chytracek, D. Duellmann, M. Frank, M. Girone, and G. Govi *et al.*, "POOL development and plans," in *Proc. CHEP'04*, Interlaken, Switzerland, Sep. 2004, p. 475.
[7] G. van Rossum and F. L. Drake Jr., Eds., *An Introduction to Python*. Bristol, U.K.: Network Theory Ltd., 2003.
[8] *http://cern.ch/pi/ireview03/aidareview.pdf* [Online]
[9] R. Brun and F. Rademakers, "ROOT—;an object oriented data analysis framework," in *Proc. AIHEPN-96*, Lausanne, Switzerland, Aug. 1996.
[10] *http://jas.freehep.org* [Online]
[11] *http://www.lal.in2p3.fr/OpenScientist/* [Online]
[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Reading, MA: Addison-Wesley, 1995.
[13] *http://sourceforge.net/projects/cppunit* [Online]
[14] *http://spi.cern.ch* [Online]
[15] *http://www.slac.stanford.edu/grp/ek/hippodraw/* [Online]
[16] M. Donszelmann, T. Johnson, V. Serbo, and M. Turri, "AIDA, JAIDA and AIDAJNI: data analysis using interfaces," in *Proc. CHEP'04*, Interlaken, Switzerland, Sep. 2004, p. 445.
[17] G. A. P. Cirrone, S. Donadio, S. Guatelli, A. Mantero, B. Mascialino, and S. Parlati *et al.*, "A goodness-of-fit statistical toolkit," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 5, 2004.