π
A technical overview

Application Area Review, 20-22 Oct 2003

Lorenzo Moneta

CERN/EP-SFT

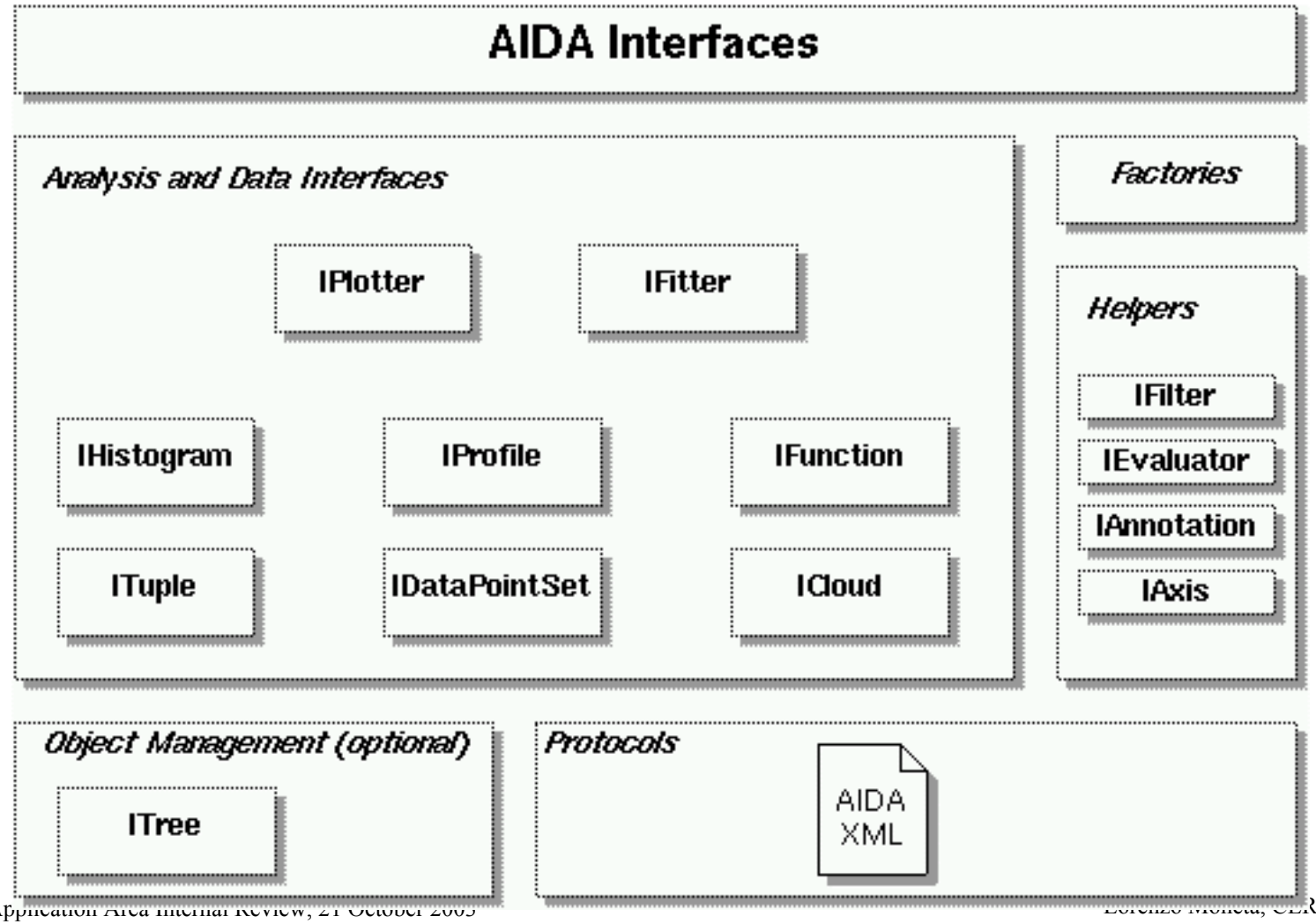
http://lcgapp.cern.ch/project/pi/

Outline

- **PI Analysis Services:**
  - AIDA interfaces
  - PI extension to AIDA: Proxy layer with value semantics
  - ROOT Implementation of AIDA Histogram interfaces
  - Interface to I/O: Root and XML
  - Histogram converter AIDA-ROOT
- **Latest release 1.0.0:**
  - QA : unit and performance tests
- **Integration with external applications:**
  - Prototype of Python binding to AIDA
  - Interface with ROOT (using PyROOT) and HippoDraw
- **User feedback**
- **Summary**

AIDA

- AIDA - Abstract Interfaces for Data Analysis

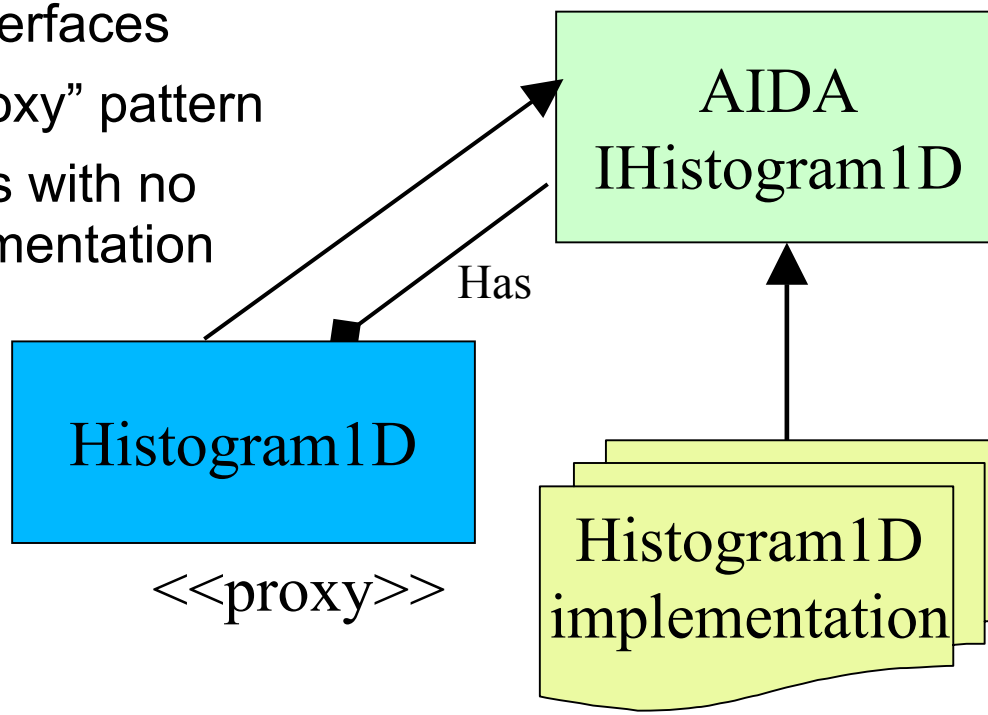


AIDA Interfaces

- **Version 3.0 since Oct. 2002**
  - User level interfaces to analysis objects (histograms, ..), plotter and fitter
  - Expose pointers to objects with factories
  - Management and storage using Tree interface
  - XML protocol for data exchange
- **Missing**
  - Separation between Factories and Tree
    - adopt a different management schema
  - Simplified value-semantic layer with constructors and operators
    - Hiding of factories to end-user
  - Developer interface to ease building generic manipulators and tools
    - Independent analysis components

AIDA Proxy layer

- Extension of AIDA interface:
  - A layer on top of AIDA interfaces with value semantics
  - C++ proxy classes to AIDA interfaces
    - Implemented using the “Proxy” pattern
  - Based only on AIDA Interfaces with no dependency on a given implementation



Advantages of AIDA Proxy

- **Value semantics easier for users**
  - No pointers involved
  - User manages the objects (no magic)
- **Much simpler than using directly AIDA**
  - No need to use factories to create an object
- **Example: creation of an Histogram:**
  - **AIDA:**

```
IAnalysisFactory * af = create_AIDA_AnalysisFactory();
ITreeFactory * tf = createTreateFactory();
ITree * tree = tf->create();
IHistogramFactory * hf = af->createHistogramFactory(*tree);
IHistogram * h = hf->createHistogram1D("myHisto",100,0,10);
```

  - **PI:**

```
Histogram1D  h("myHisto",100,0,10);
```

Features of AIDA Proxies classes

- Keeping the functionality and signatures of AIDA
  - "re-shuffling" of factory methods to object constructors
- Proxy classes can expand functionality of AIDA
  - Additional features can be easily added on user requests
- Hiding of AIDA object management
  - Easier the integration with experiment framework
  - AIDA tree is not exposed to users but hided in the Proxy implementation
  - Tree can be replaced in the future with SEAL whiteboard
- Dynamic loading using SEAL plugin manager
  - load at run time the chosen implementation
    - AIDA ROOT histograms or AIDA Native
  - load store library (Root based files or XML)

AIDA Proxy classes

- **Generated Proxies for all AIDA data objects**
  - Histograms, Profiles, Clouds, DataPointSets, Tuples
- **Proxies exist also for Functions and Fitter**
  - Plotter can be done later (if requested)
- **AIDA_ProxyManager class**
  - Not exposed to users
  - Implemented using the Loki singleton
  - Use AIDA Tree to manage the objects
  - Load AIDA factories using SEAL plugin manager
    - Implementations can be chosen by the user
  - No dependency on any AIDA implementation
    - Only interfaces and SEAL plugin manager

AIDA Proxy classes (2)

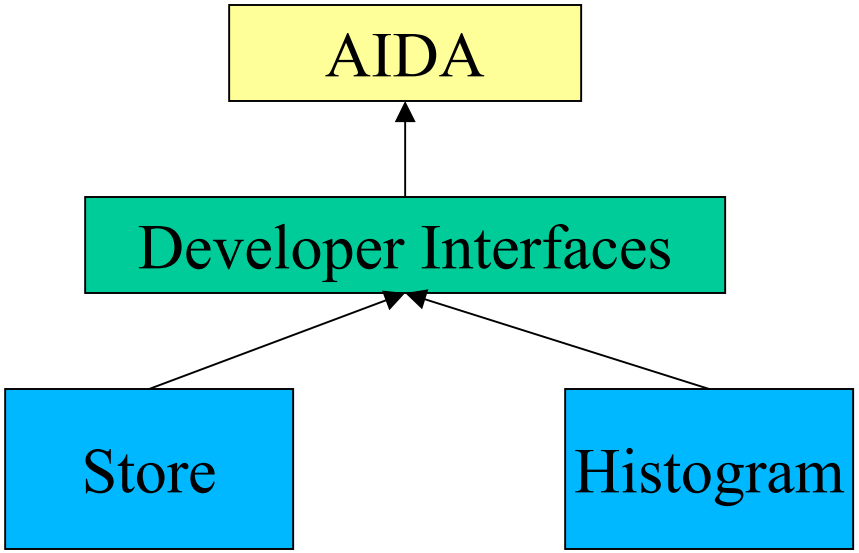
- **Proxy_Selector**
  - Use to select default implementations of AIDA objects to be used by in the application
- **Proxy_Store**
  - Class for storing and retrieving objects from I/O
  - Requested by users for evaluation of interfaces
  - Simple interface
    - Only *open( ), write( ), retrieve() and close( )* methods
  - Copy objects from a AIDA memory Tree to a Tree mapped to a store
  - Support for XML and Root I/O
- **HistoProjector**
  - Helper class for projections
  - Avoid using factories

Summary of AIDA_Proxy

- **All AIDA functionality is available** (excluding ITree)
- **Easy to use**
  - Hide factories from users
- **Value semantics**
  - Implemented operator “+” and “=“
  - Conversion (with copy constructors and operator “=“) from AIDA interf.
- **Copy between implementations**
  - AIDA native to Root and vice versa
- **Choose implementation at runtime**
  - User can decide implementation when constructing the objects
- **Objects are managed by the user** (not by AIDA Tree)
  - Easy integration with other frameworks

AIDA Developer interfaces

- **Abstract interface layer extending AIDA**
  - Allows interoperability between different implementations
  - De-couple AIDA implementation components
- **Preliminary interfaces exist in PI**
  - to be discussed in AIDA team and converge on a common set



AIDA ROOT Implementation

- **AIDA Histograms implementation using ROOT**
  - Default implementation used by Proxy layer
  - Support now for all histograms (1D, 2D and 3D) and profiles (1D and 2D
  - Implemented as a wrapper around the root objects
  - Use AIDA developer interface layer
- **AIDA_RootConverter**
  - For conversions from AIDA histograms to ROOT and vice versa
  - Based on AIDA developer interface
- **AIDA_RootStore**
  - provide storing and retrieving of histograms and profiles using a root file
  - Use AIDA_RootConverter

AIDA implementation Plug-ins

- **Use SEAL plugin manager to load implementations of AIDA interfaces**
- **Plugins exist now in PI for :**
  - Histograms:
    - AIDA_ROOT histograms (default implementation)
    - AIDA Native
  - Tuples, DataPointSets libraries
  - Functions and Fitting library (based on old Minuit or NagC)
  - AIDA Tree library
  - XML store library (based on expat)
  - Root store library

Present Status of PI

- **Latest release :**
  - 1.0.0 available on /afs release area containing:
    - AIDA developer interfaces
    - Proxy Layer
    - Complete AIDA ROOT Histograms
    - AIDA Tree implementation with interface to store
    - ROOT and XML format stores
    - ROOT ↔AIDA Converters
    - SEAL Plugins to AIDA implementations
- **Integration with CMS**
  - Examples using AIDA Proxy from PI exist in ORCA
- **Started integration with LHCb**
  - Integrate in Gaudi the ROOT Histogram implementation of PI
- **ATLAS will follow LHCb soon**

QA in PI

- **Documentation**
  - Examples on how to use PI are available on the WEB since first PI release
    - http://lcgapp.cern.ch/project/pi/Examples/PI_0_4_1
  - Reference documentation obtained from Doxygen (thanks to SPI)
- **Tests**
  - Extensive test suite (order of 1000) in CPP unit of AIDA histograms
    - Test all functionality of interfaces
    - Test I/O and copying between implementations
    - Integrated in Oval and Qmtest
    - *Thanks to Hurng-Chun Lee*
  - Unit tests exist also for the other components
    - Plan to migrate them to Cpp unit

Performance Tests

- **Mesure for Histograms (1D, 2D and 3D) and Profiles (1D and 2D) in the**
  - AIDA ROOT implementation
  - AIDA Native
  - ROOT
- **CPU time for booking and filling 10^6 events**
- **Memory size occupied by the histograms**
- **File size in I/O format**
  - ROOT (compressed/uncompressed)
  - XML (compressed/uncompressed)
- **Results available on the Web**
  - CPU, Memory, File Size

Possible Future evolution

- **Integration with persistency services from POOL**
  - Implement AIDA tuples using POOL collections
- **Fitting and Minimization**
  - Develop interfaces for minimization
  - Pluggable minimization engine
  - Make an implementation using new C++ Minuit from SEAL
- **Integrate more with SEAL services**
  - Use SEAL whiteboard
- **Work on Interoperability between components from different implementations**
  - Push in AIDA for developer interfaces
  - Plotter Integration
    - Use OpenScientist and/or HippoDraw
    - Integrate JAS plotter through Java JNI interface

Integration with External Tools

- **Use Python for a prototype integration in an interactive environment**
- **Integration of AIDA with ROOT (using PyROOT) and HippoDraw**
  - Use Python bindings to AIDA interfaces
  - Simple Python program to copy the AIDA histograms in ROOT or HippoDraw compatible objects
  - use the Boost-Python interface to copy in and plot them in HippoDraw
  - Or use PyROOT to plot in a Root canvas
- **Demo:**
  - Create AIDA histograms and plot them in Root canvas
  - Create AIDA clouds and plot them in HippoDraw

User feedback

- **Got a very positive feedback on AIDA (and PI)**
- **AIDA Histograms are widely used through Gaudi in LHCb and ATLAS**
- **Received from users (LHCb and CMS)**
  - Interest in clouds (unbinned histograms)
  - Request for more work towards interactivity (Python)
  - **LHCb :**
    - Specific requests for AIDA ROOT histograms
  - **CMS :**
    - Gravity bin histograms
    - Online requirements on histograms
    - Interest in an interface for Tuples
      - Use POOL collection ?
    - Interest in fitting (replace NagC with new Minuit C++)

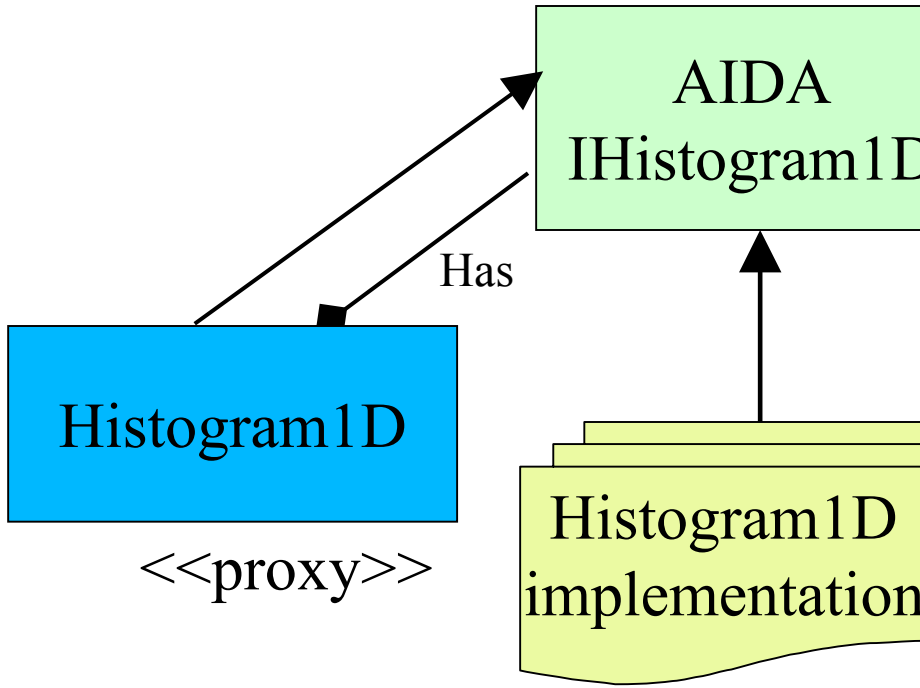
Summary

- **Development on PI Analysis Services according to the project plan is almost completed**
  - A ROOT implementation (wrapper) for AIDA binned histograms
  - Value semantic layer for AIDA objects for end-users
  - I/O support in XML and Root
  - Started integration with experiments
    - CMS provides examples to use AIDA Proxy
    - LHCb is integrating AIDA ROOT in Gaudi
- **Review of AIDA completed**
  - AIDA review document available (link)
  - Very positive feedback received from users (LHCb and CMS)
    - Need have integration completed with experiment framework for more user feedback

AIDA_Proxy in more detail

Histogram1D

```
namespace pi_aida {
class Histogram1D : public AIDA::IHistogram1D {
  public:
 // Constructor following the factory-create method
    Histogram1D(std::string title,
        int nBins, double xMin, double xMax);
    // as an example the fill method:
    bool  fill ( double x, double weight = 1. )
         { if (rep == 0) return 0;
           else return rep->fill ( x , weight ); }
    // other methods are also mostly inlined …
  private:
    AIDA::IHistogram1D * rep;
}; }
```



Example: Histogram1D

```
/ Creating a histogram
 pi_aida::Histogram1D h1( "Example histogram.", 50, 0, 50 );
/ Filling the histogram with random data
 std::srand( 0 );
 for ( int i = 0; i < 1000; ++i )
   h1.fill( 50 * static_cast<double>( std::rand() ) / RAND_MAX );

// Printing some statistical values of the histogram
 std::cout << "Mean:" << h1.mean() << "   RMS:" << h1.rms() << std::endl;

/ Printing the contents of the histogram
const AIDA::IAxis& xAxis = h1.axis();
for ( int iBin = 0; iBin < xAxis.bins(); ++iBin )
   std::cout << h1.binMean( iBin )    << "       "
             << h1.binEntries( iBin)   << "       "
             << h1.binHeight( iBin ) << std::endl;
```

Example: Fitting a Histogram

```
// create and fill the histogram
//…….
// Creating the fitter (ChiSquare by default)
pi_aida::Fitter fitter;   // or:   fitter("BinnedML")
// Perform a Gaussian fit, use shortcut with strings
// fitter.fit(h1,function) to pass a user defined function
AIDA::IFitResult& fitResult = *( fitter.fit( h1, "G" ) );
// Print the fit results
std::cout << "Fit result : chi2 / ndf : " << fitResult.quality() << " / " <<
    fitResult.ndf() << std::endl;
for ( unsigned int i = 0; i < par.size(); ++i ) {
  std::cout << fitResult.fittedParameterNames()[i]
            << " = " << fitResult.fittedParameters()[i]
            << " +/- " << fitResult.errors()[i]
            << std::endl;
}
```

Example: Operations on Histograms

```
/ Creating a histogram in the native AIDA implementation
pi_aida::Histogram1D h1( "Example h1", 50, 0, 50, "AIDA_Histogram_Native" );
/ fill h1
std::srand( 0 );
 for ( int i = 0; i < 1000; ++i )
   h1.fill( 50 * static_cast<double>( std::rand() ) / RAND_MAX );

/ Creating a histogram using Root implementation
pi_aida::Histogram1D h2( "Example h2", 50, 0, 50, "AIDA_Histogram_Root" );
/Copying
h2 = h1;
/adding  (default type is used when creating h3)
pi_aida::Histogram1D h3 = h1 + h2;
```

Example: Histogram Projections

```
// Creating a 2D histogram
  pi_aida::Histogram2D h( "Example 2D  hist.", 50, 0, 50, 50, 0, 50 );
// Filling the histogram…..
  …..
 // projections
pi_aida::HistoProjector hp;
// project:  created histogram is of default type
pi_aida::Histogram1D hX = hp.projectionX(h);
// project on a Root histogram
pi_aida::Histogram1D hY= hp.projectionY(h,0,50,"AIDA_Histogram_Root");
```

- Implement projections on Histograms ?
  - **hX = h.projectionX()**

Example: Store and Read Histograms

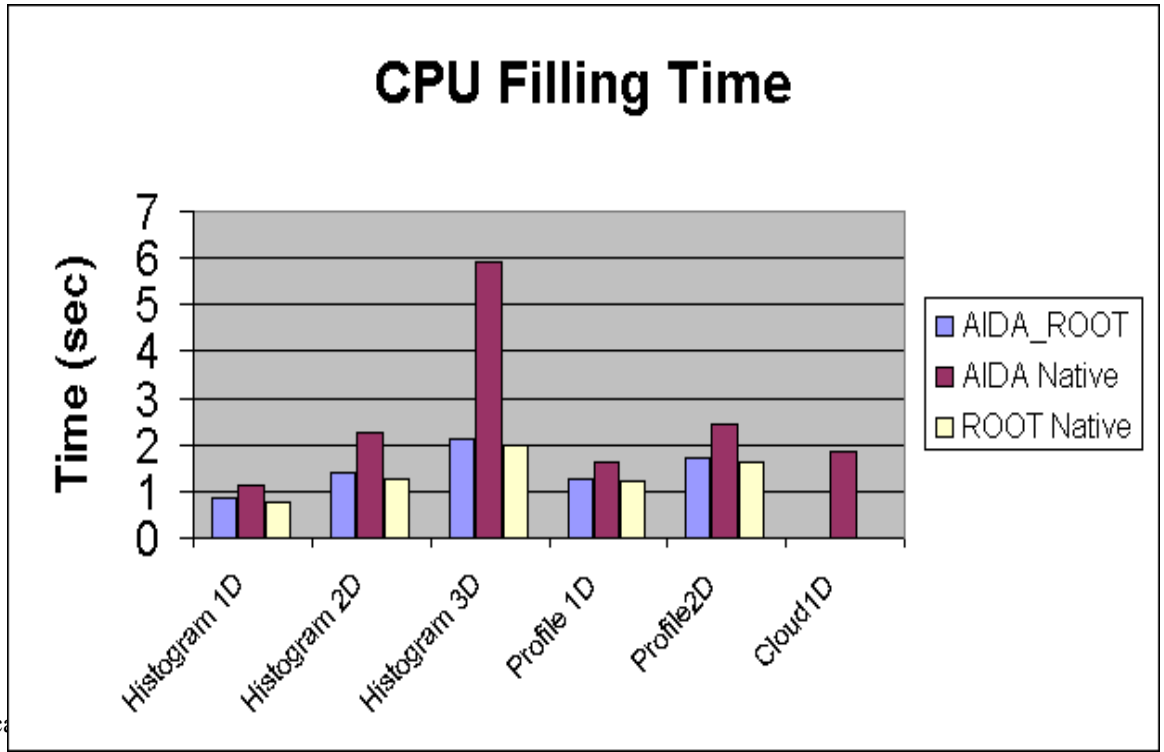
```
// after created and filled the histograms
  ……

// create a ROOT  Proxy_Store
 pi_aida::Proxy_Store s1("hist.root","Root");
s1.write(h1);
s1.close();
// create a XML Proxy_Store
pi_aida::Proxy_Store s2("hist.xml","XML");
s2.write(h1);
s2.close();

// read histogram from the Root store
Histogram1D h1 = s1.retrieve<Histogram1D> ("h1 name");
```

CPU Time test results

- **Time to book and fill an histogram with 10^6 events**
  - Example: results for Histogram1D:
    - **ROOT: 0.75 s**
    - **AIDA ROOT: 0.85 s**
    - **AIDA Native: 1.11 s**



Memory and File Size test results

- **Size occupied for 10^6 equivalent bin histograms:**
  - 1000 Histogram1D with 100 bins
  - 100 Histogram2D with 100x100 bins