Status Report of

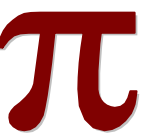
π

Analysis Services

Application Area Meeting, 9 July 2003

Lorenzo Moneta

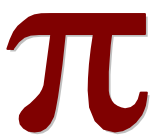
CERN/EP-SFT



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

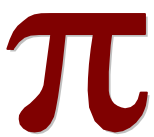
Overview

- Physicist Interface (PI) started in mid Nov'02
  - Review with experiments to define workplan
  - Project proposal to SC2 end January 03
  - Status report at SC2 last week
- Five working areas identified
  - **Analysis Services**
  - **Analysis Environment**
  - **Pool & Grid PI**
  - **Event & Detector Visualization**
  - **Infrastructures & Documentation**
- Not all items have same priority:
  - Analysis Services first
- Resources:
  - Vincenzo Innocente 30%, Andreas Pfeiffer (40%), L.M. (50%)



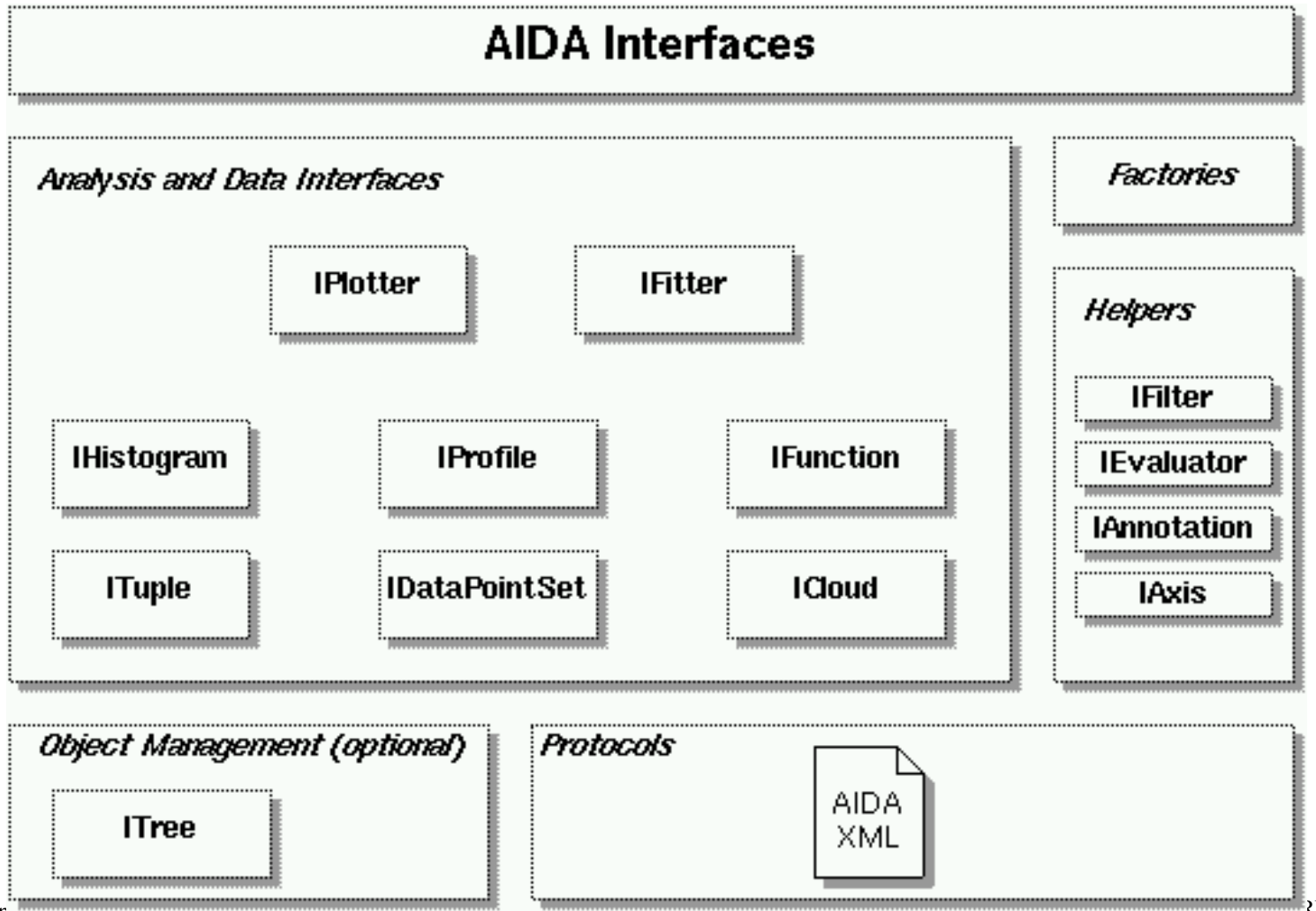
Analysis Services

- AIDA
  - Review, adapt and extend Interfaces to Data Analysis
- Root implementation of AIDA
  - Provide an implementation of the Interfaces to Data Analysis, as defined by the previous work package, based on Root.
- Interface to SEAL and POOL services
  - Use SEAL and POOL to provide PI with services such as plugin management, object whiteboard and persistency services.
- Blueprint compliant Analysis tool set
  - Integration of various component in a consistent analysis tool set based on AIDA interfaces
    - mainly external contributions



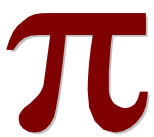
AIDA

- AIDA - Abstract Interfaces for Data Analysis



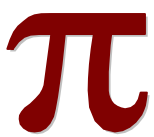
AIDA (2)

- **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
    - Possibility to adopt a different management schema
  - Simplified value-semantic layer with constructors and operators
    - Hiding of factories from end-user
  - Developer interface to ease building generic manipulators and tools
    - Independent analysis components

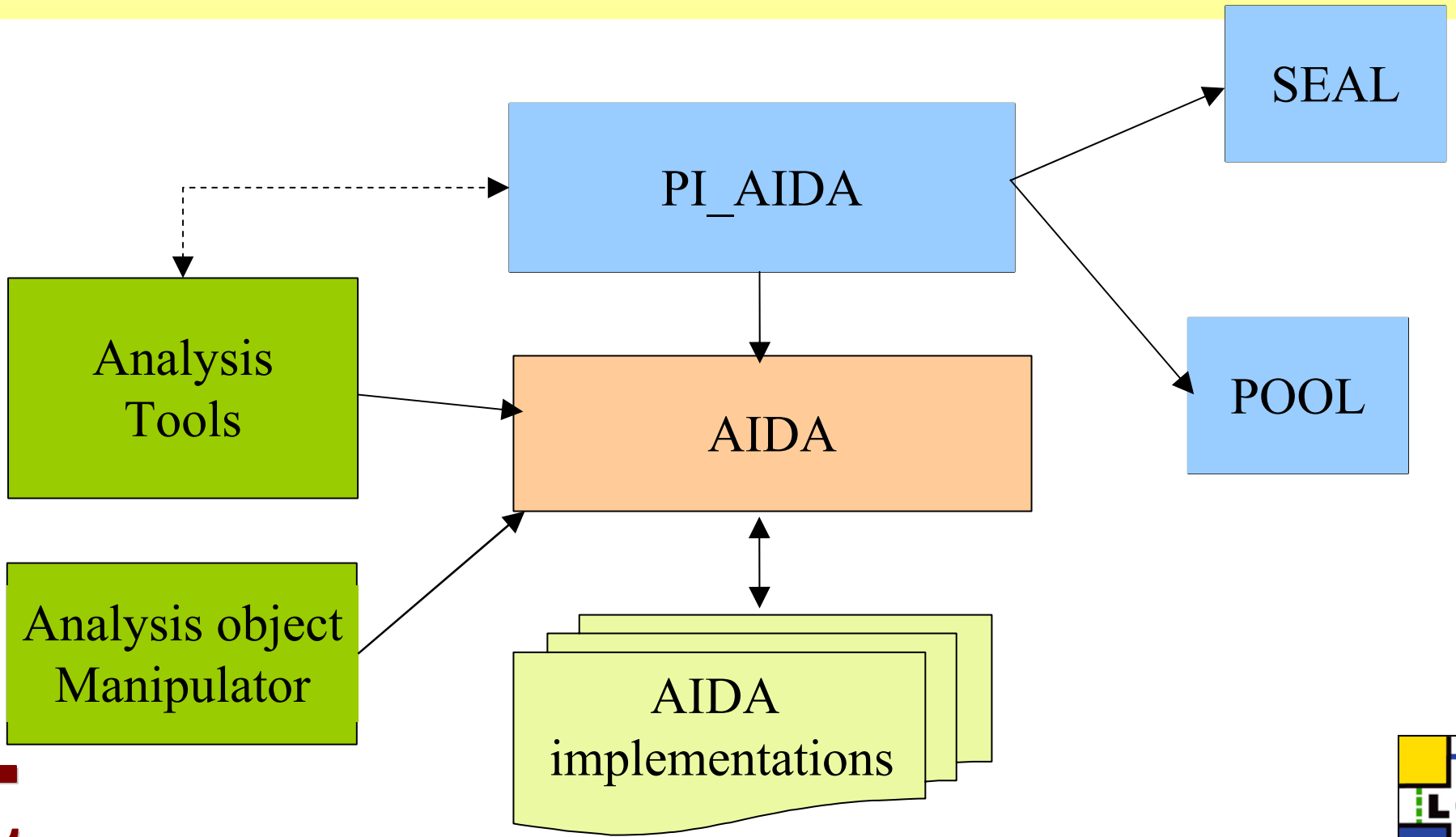


The Goals

- **Interoperability**
  - Plug in different implementations
  - Mix components from the existing implementations
    - Use OpenScientist plotter with AIDA ROOT implementation
- **Extensions**
  - Generic manipulator such as projectors, rebinners, etc
  - Build and manipulate aggregate of objects (CANs)
- **Interface to external applications**
  - Store AIDA histograms in POOL (connected to experiment specific data)
  - Display AIDA histograms using HippoDraw or EXCEL
- **Framework to develop complex analysis tools**
  - Statistical comparison of data-sets
  - Modelling parametric fit problems using a MonteCarlo approach

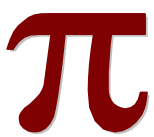


PI Analysis Services



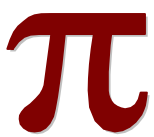
Milestone 1 : AIDA Proxy layer

- "Value semantics" for AIDA objects
  - C++ proxy classes to AIDA interfaces
    - Implemented using the "Proxy" pattern
    - 80% done using a script
  - Based only on AIDA Interfaces
    - ➔ ***no dependency on a given implementation***
- Initially "hiding" of AIDA object management
  - AIDA Tree is not exposed to users but hided in the Proxy implementation
- Keeping the functionality and signatures of AIDA
  - "re-shuffling" of factory methods to object constructors
- Dynamic implementation loading
  - Use SEAL plugin manager to load the chosen implementation



AIDA_Proxy classes

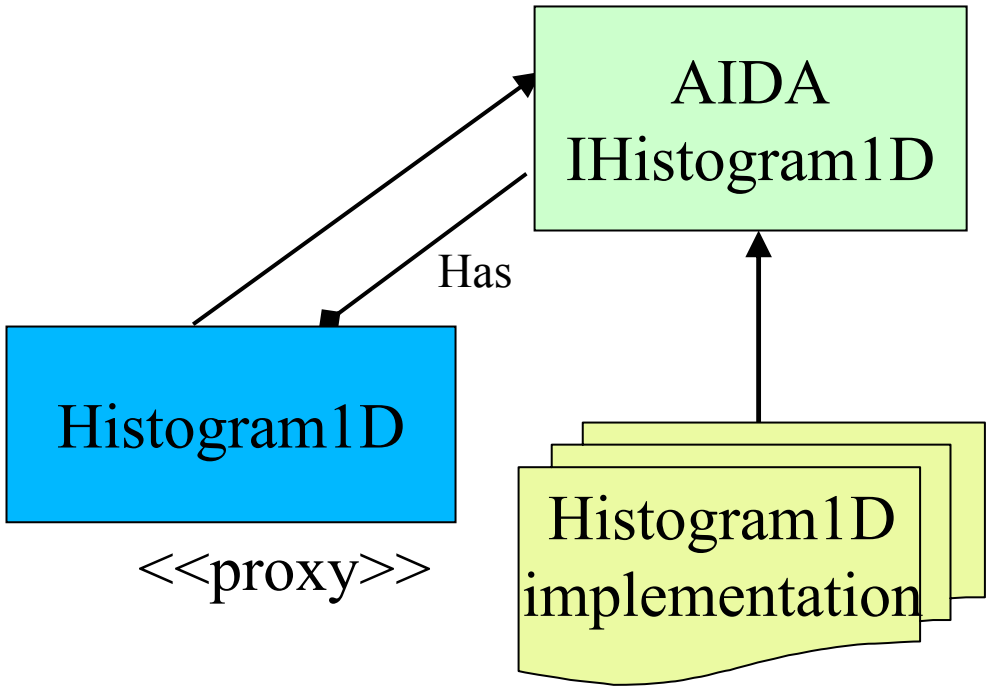
- **Generated Proxies for all AIDA data objects**
  - Histograms, Profiles, Clouds, DataPointSets, Tuples
- **Proxies exist also for Functions and Fitter**
  - Plotter will be done later
- **AIDA_ProxyManager class**
  - Not exposed to users
  - Load factories (implementations) using SEAL plugin manager
- **Proxy_Store**
  - Prototype class for storing objects in a **XML** and/or a **Root** file
    - Only *open( ), write( ) and close( )* methods
  - Requested by users for evaluation of interfaces
- **HistoProjector**
  - Helper class for projections
  - Avoid using factories



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;
}; }
```

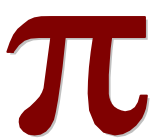


Example: Create and Fill a 1D Histogram

```
// 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;
```



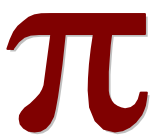
Example: Fitting the Histogram

```
// 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: Storing Histograms

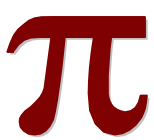
- **Support writing in :**
  - **ROOT** (only binned Histograms as TH* objects)
  - **XML** (all the AIDA analysis objects)

```
// 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();
```

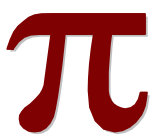
Features of AIDA_Proxy

- **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 decides implementation when constructing the objets
- **Objects are managed by the user** (not by AIDA Tree)
  - Easy integration with other frameworks



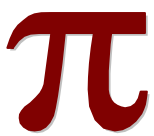
AIDA Proxy

- **Examples on how to use with web-docs**
  - Available from PI web page since first release (0.1.0 at the end of May)
- **Latest release :**
  - **0.2.1** with support for storage and projections
- **Started integration with CMS SW**
  - Examples using PI_AIDA in ORCA
- **Will be basis for a user-review and further evaluation**
  - **Need feedback on the existing interfaces from users in LHC experiments**
  - Any feedback will be propagated back to AIDA team



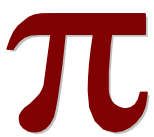
AIDA ROOT Implementation

- **AIDA_ROOT provides an implementation of AIDA Histograms**
  - Support now for 1D Histograms and Profiles.
  - Complete with implementations for all binned histograms (for 0.3.0)
- **AIDA_Root::Histogram1D is a wrapper around a TH1D**
  - Use a developer interface layer
    - Creation by generic template factories
- **Storage of Histogram using a Root file**
- **Package integrated with AIDA_Proxy**
  - Plugin exists to load AIDA_ROOT implementation
  - Integrated in the application with other AIDA implementations
- **Future Evolution:**
  - Root Store with full read/write support



Future evolution

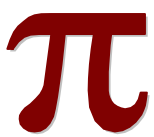
- **Use AIDA developer interfaces**
  - Develop common utilities based only on developer interfaces
    - Copying objects, manipulators, projectors,…..
  - Interoperability between components from different implementations
- **Plotter Integration**
  - Use OpenScientist and/or HippoDraw
  - Integrate JAS plotter through Java JNI interface
- **Integration with experiment frameworks**
  - using SEAL component model and object whiteboard
- **Integration with persistency services from POOL**
  - Implement AIDA tuples using POOL collections
- **Implement fitter using new Minuit C++ (from SEAL)**



Integration with External Tools (1)

❖ Integration of AIDA and HippoDraw

- Prototype integration performed at the Python layer level
  - AIDA Histograms are created and filled using AIDA Python bindings
    - Bindings to AIDA generated with SWIG
  - Simple Python program to copy the AIDA objects in HippoDraw compatible objects
    - Create an HippoDraw tuple from AIDA analysis objects
    - Can plot also AIDA Clouds and DataPointSets (see demo)
  - use the Boost-Python interface to copy in and plot objects in HippoDraw
  - *Thanks to Paul Kunz for helping*

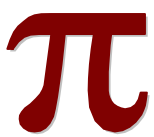


Integration with External Tools (2)

Integration with ROOT

- Bridge to Root from Python:
- Use Python bindings to Root (PyROOT) from SEAL
  - Done using the Root dictionary
- AIDA objects are copied in Root objects at the Python level
- Example:
  - display AIDA Histograms in a Root canvas from Python

    (see demo)

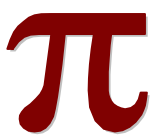


PI releases

- **PI latest release : 0.2.1**
  - Available on afs at
    - **`/afs/cern.ch/sw/lcg/app/releases/PI/PI_0_2_1`**
  - configuration
    - based on SEAL 0.3.2
- **PI 0.3.0** will be released in a few days
  - Complete AIDA_ROOT implementation for all AIDA binned Histograms
- Public release PI 1.0.0 when SEAL has released 1.0.0
- Examples available on the Web

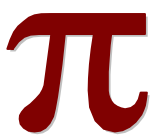
  http://lcgapp.cern.ch/project/pi/Examples/PI_0_2_1
- More information and documentation at the PI homepage:

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



Conclusions

- **Work started on Analysis Services**
  - A ROOT implementation (wrapper) for AIDA binned histograms
  - Value semantic layer for AIDA objects for end-users
- **Review of AIDA in progress**
  - AIDA workshop last week here at CERN (see Andreas summary)
  - **We need input from users working in the experiments !**
- **Goal:**
  - Provide a fully consistent interface and a set of low level tools that satisfy the requirements of both end-users and developers of high-level tools
- **Prototype work in integration and interoperability**
  - Integration with other frameworks/tools at the interactive level
    - Example of visualization using HippoDraw and Root
  - Use python bindings from SEAL (PyROOT)

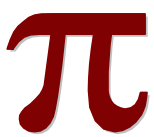


Other Work Packages

- WP 2 – Analysis Environment
  - Basic interactive analysis application: based on SEAL with python binding, plugin manager, distributed interfaces
  - Visualization services: interactive canvas, Model-Document-View
  - Bridge to and from ROOT: interoperability at cint prompt, etc.
  - in collaboration with SEAL, POOL
- WP 3 – POOL & GRID Interfaces
  - Collections & Metadata
  - File Catalog and Replicas (both local and remote)
  - Job Wizard (preparation, submission, validation)
  - Job Helpers (monitoring, error recovery, resource discovery)
  - In the Requirements & Analysis Phase (RTAG 11)
- WP 4 – Event Display and Detector Visualization
  - HepVis: review, adjust, extend (to cover LCG and Geant4 needs)
  - Integrate into interactive analysis environment
  - Geant4 visualization application
  - In collaboration with the experiments (aim for common product at a later stage)
- WP 5 – Infrastructure and Documentation
  - In collaboration with SPI

AIDA_ProxyManager

- **Implemented as a Loki singleton**
- **Objects are created using AIDA Factories**
- **No dependency on a particular implementation**
- **Factories (and relative implementations) are loaded using a plugin manager (from SEAL)**
  - Possible to choose implementation a run-time
  - Plugins exist now for all Anaphe implementations and for ROOT implementation of AIDA Histograms
- **Objects are managed by a memory tree**
  - Tree implementation can also be chosen using the plugin manager
- **Store objects using AIDA tree types (XML based or Root)**
  - Users interact only with the Proxy_Store



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()**