

Analysis Services in the PI project

AppArea, 9 April 2003

Andreas Pfeiffer

CERN/EP



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

April 9, 2003

Andreas Pfeiffer CERN/EP

Work packages in PI

❖ PI started in mid Nov'02 (Vincenzo Innocente)

- ❑ Review with experiments to define workplan
- ❑ Project proposal to SC2 end Jan'03

❖ Analysis Services

- ❑ Status: active

❖ Analysis Environment

- ❑ Status: on hold (mainly) by PEB

❖ Pool & Grid PI

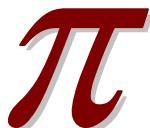
- ❑ Status: on hold pending RTAG by SC2

❖ Event & Detector Visualization

- ❑ Status: on hold until May by SC2

❖ Infrastructures & Documentation

- ❑ Status: active



Analysis Services

❖ AIDA

- ❑ Review, adapt, extend Interface 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.

❖ AIDA interface to SEAL and POOL services

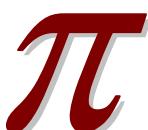
- ❑ Use SEAL and POOL to provide AIDA with services such as object whiteboard and persistency.

❖ Blueprint compliant Analysis tool set

- ❑ On Hold by SC2 till May 03

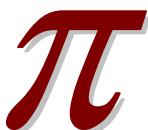
❖ Resources

- ❑ Lorenzo Moneta (also SEAL)
- ❑ Andreas Pfeiffer (also CMS)



Milestone 1: AIDA Proxy layer

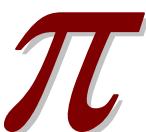
- ❖ “Value semantics” for AIDA objects
 - Implemented using the “Proxy” pattern, very easy !
 - Based only on AIDA Interfaces
→ ***no dependency on a given implementation***
 - Initially “hiding” of AIDA object management, later: use of SEAL whiteboard
- ❖ Keeping the functionality and signatures of AIDA
 - “re-shuffling” of factory methods to object ctors
- ❖ Examples on how to use with web-docs
 - Exist since March 24, as ***PL_0_0_7*** release tag
- ❖ Will be basis for user-review and further evaluation
 - Any feedback will be propagated to AIDA team



AIDA_Proxy in more detail

❖ Proxy to IHistogram1D

```
class Histogram1D : public IHistogram1D {  
public:  
    // Constructor following the factory-create method  
    // (example)  
    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:  
    IHistogram1D * rep;  
};
```



Example: Histogram

```
#include <iostream>
#include <cstdlib>
#include <memory>

#include "AIDA_Proxy/AIDA_Proxy.h"
#include "AIDA/AIDA.h"

int main( int, char** ) {
    // 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 );
    }
}

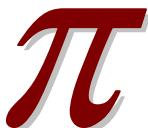
// Printing some statistical values of the histogram
std::cout << "Mean:" << h1.mean() << std::endl;
std::cout << "RMS:" << h1.rms() << std::endl;

// 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;
}
return 0;
}
```



Example: Fitting a histogram

```
// create and fill the histogram ...
// Creating the function which is going to be fitted with the histogram data
AIDA::Function gaussFun("G");
// set parameters to starting values
gaussFun.setParameter("mean" , 50.);
gaussFun.setParameter("sigma", 10.);
gaussFun.setParameter("amp" , 10.);
// Creating the fitter (ChiSquare by default)
AIDA::Fitter fitter; // or: fitter("UnbinnedML")
// Perform the fit
AIDA::IFitResult& fitResult = *( fitter.fit( h1, gaussFun ) );
// 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;
}
```

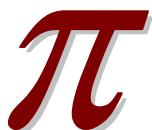


Example: Tuple (I)

```
// Defining the description of the tuple columns
std::string description = "int nTracks; double beamEnergy; bool
    goodTrigger;";

// Define a “tuple-in-a-tuple”
description += "Tuple{ double px, py, pz, mass } p;";

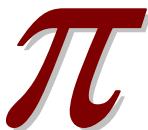
// Creating the actual tuple in memory
AIDA::Tuple tuple( "id", "example tuple", description );
```



Example: Tuple (intermediate)

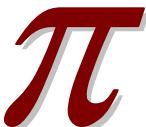
```
// these will be used in the filling ... so they are listed for completeness
DRand48Engine randomEngine;
RandGauss rBeamEnergy( randomEngine, 90, 5 );
RandGauss rTracksSpread( randomEngine, 0, 2 );
RandGauss rMomentum( randomEngine, 0, 3 );
RandGauss rMass( randomEngine, 1, 0.1 );

// Filling the tuple, for performance get the indices first ...
int i_nTracks = tuple.findColumn( "nTracks" );
int i_beamEnergy = tuple.findColumn( "beamEnergy" );
int i_goodTrigger = tuple.findColumn( "goodTrigger" );
int i_p = tuple.findColumn( "p" );
```



Example: Tuple (II)

```
const double tracksPerEnergy = 0.5;
for ( unsigned int i = 0; i < 1000; ++i ) {
    double beamEnergy = rBeamEnergy.fire();
    int numberOfTracks = static_cast<int>( tracksPerEnergy * beamEnergy + rTracksSpread.fire() );
    tuple.fill( i_nTracks, numberOfTracks );
    tuple.fill( i_beamEnergy, beamEnergy );
    tuple.fill( i_goodTrigger, ( (rMomentum.fire() > 0)?true:false) );
    AIDA::ITuple* tp = tuple.getTuple( i_p );
    for ( int iTrack = 0; iTrack < numberOfTracks; ++iTrack ) {
        tp->fill( 0, rMomentum.fire() );
        tp->fill( 1, rMomentum.fire() );
        tp->fill( 2, rMomentum.fire() );
        tp->fill( 3, rMass.fire() );
        tp->addRow();
    }
    tuple.addRow();
}
```



Near term plans

❖ Various small changes

- ❑ Namespace PI_AIDA for Proxies
- ❑ Use of SEALS plugin manager to allow parallel use of several implementations

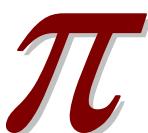
❖ First *public* release (V-0.1.0, planned for April 11)

❖ User review (see later)

- ❑ Need feedback on what exists now !

❖ Next steps

- ❑ Continue to work on Root implementation
- ❑ Add remaining functionality (according to user review) to Proxy layer
 - Projections, ...
- ❑ Integrate tests with CppUnit and Oval



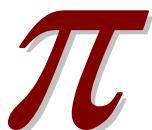
Further plans (I)

❖ Prototyping on integration with other frameworks

- ❑ SEAL – object whiteboard
- ❑ POOL – object persistency
- ❑ hippoDraw – visualisation
- ❑ Expt frameworks – volunteers ?

❖ Enhance AIDA

- ❑ User review (see later)
- ❑ ErrorPropagation Functor to allow correct (and/or user specified) treatment of error propagation in profile histos and DataPointSets



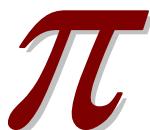
Further plans (II)

❖ Review “Developer level” Interfaces

- ❑ Adapt to use SEALS object whiteboard and POOL persistency
- ❑ Re-implement Proxies using this
 - Transparent to the user !
- ❑ Propose to AIDA team

❖ Design “Canned ANalysis” objects (CANs)

- ❑ Container to hold various “related” AIDA objects
 - Histo(s) for data, Histo(s) for MC, Fit(s) to either ...
- ❑ Gather user-requirements first



User Review of AIDA

- ❖ **Review of AIDA (Proxies) by users in the experiments**
 - ❑ Need feedback on what exists now !
 - Concentrate on the **Interfaces**, *not* on the implementations (or on performance (yet))
- ❖ **Questions to be answered:**
 - ❑ Are the interfaces complete ?
 - ❑ What features would you like to add/change ?
 - ❑ Are the methods and signatures clear enough ?
 - ❑ What are the specific needs (use-cases) for the CANs ?
 - ❑ ...
- ❖ **Identify contact people in the experiments !**

