OO Libraries for Data Analysis using C++ and Python

Andreas Pfeiffer

CERN IT/API

andreas.pfeiffer@cern.ch

Outline

- Motivation
- Anaphe components
- Interactive Data Analysis
- Summary

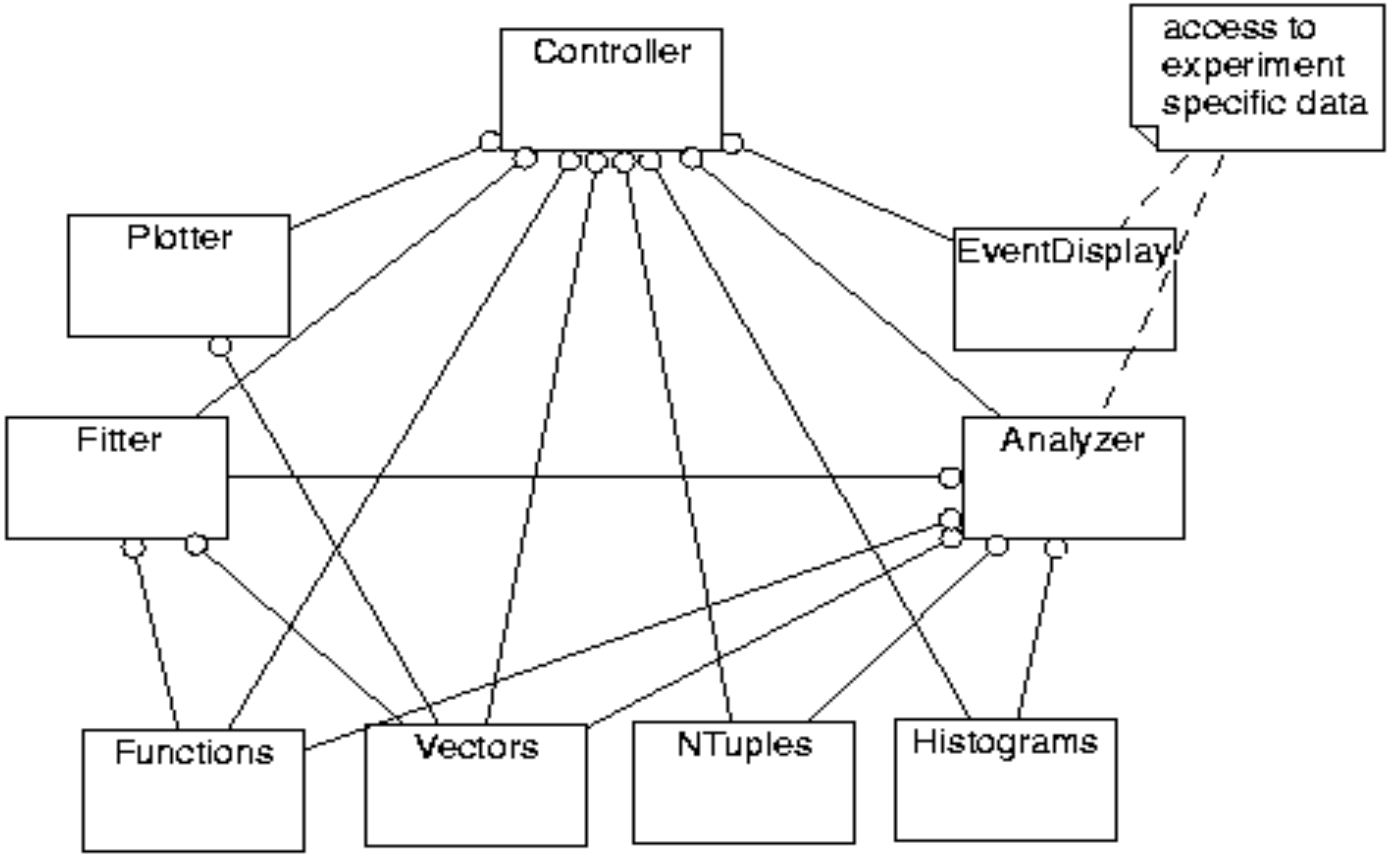
Anaphe: Analysis for Physics Experiments

- Modular (OO/C++) replacement of CERNLIB functionality
  - memory management
  - I/O
  - foundation classes
  - histogramming
  - minimizing/fitting
  - visualization
  - simulation
  - interactive data analysis
- Trying to use standards wherever possible
- Trying to re-use existing class libraries
- Detector simulation ==> GEANT-4

Architectural issue: Components (I)

- Identify components by functionality
  - not by “historic use”
- Define this functionality in terms of Abstract Interfaces
- Emphasize separation of different aspects for each component
  - example: Histogram
    - statistical entity (density distribution of a physics quantity)
    - view of a “collection of data points” (which *can* be a density distribution but also a detector efficiency curve)
    - command to manipulate/store/plot/fit/…
- Identify and use patterns - avoid anti-patterns
  - learn from other people’s experiences/failures

Categories and dependencies



Architectural issue: Abstract Interfaces

- Abstract Interfaces
  - only pure virtual methods, inheritance only from other A.I.
  - components use other components only through their A.I.
  - defines a kind of a "protocol" for a component
  - Maximize flexibility and re-use of packages
  - allow each component to develop independently
  - re-use of existing packages to implement components reduces start-up time significantly
- De-couple implementation of a package from its use

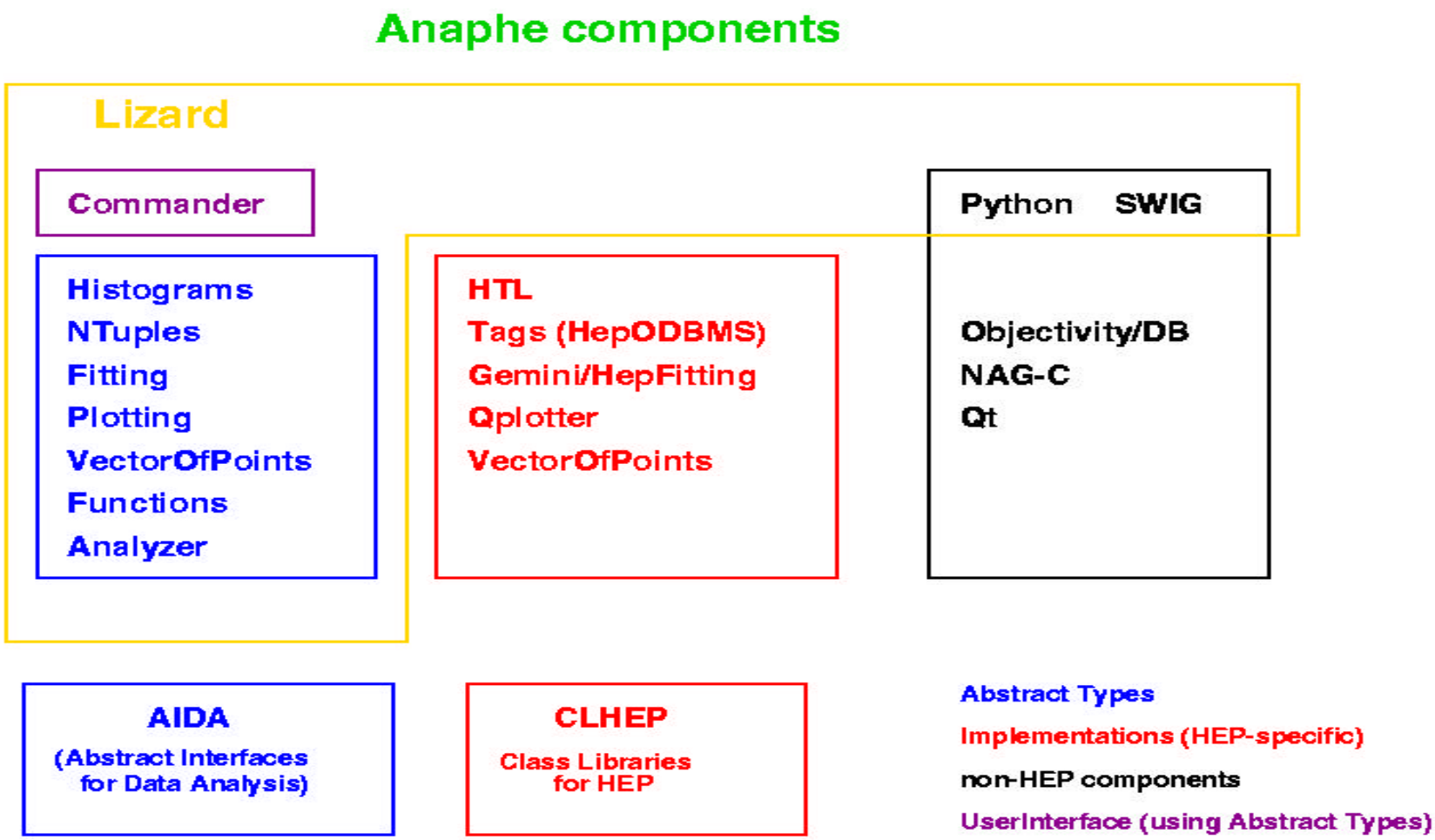
The AIDA project

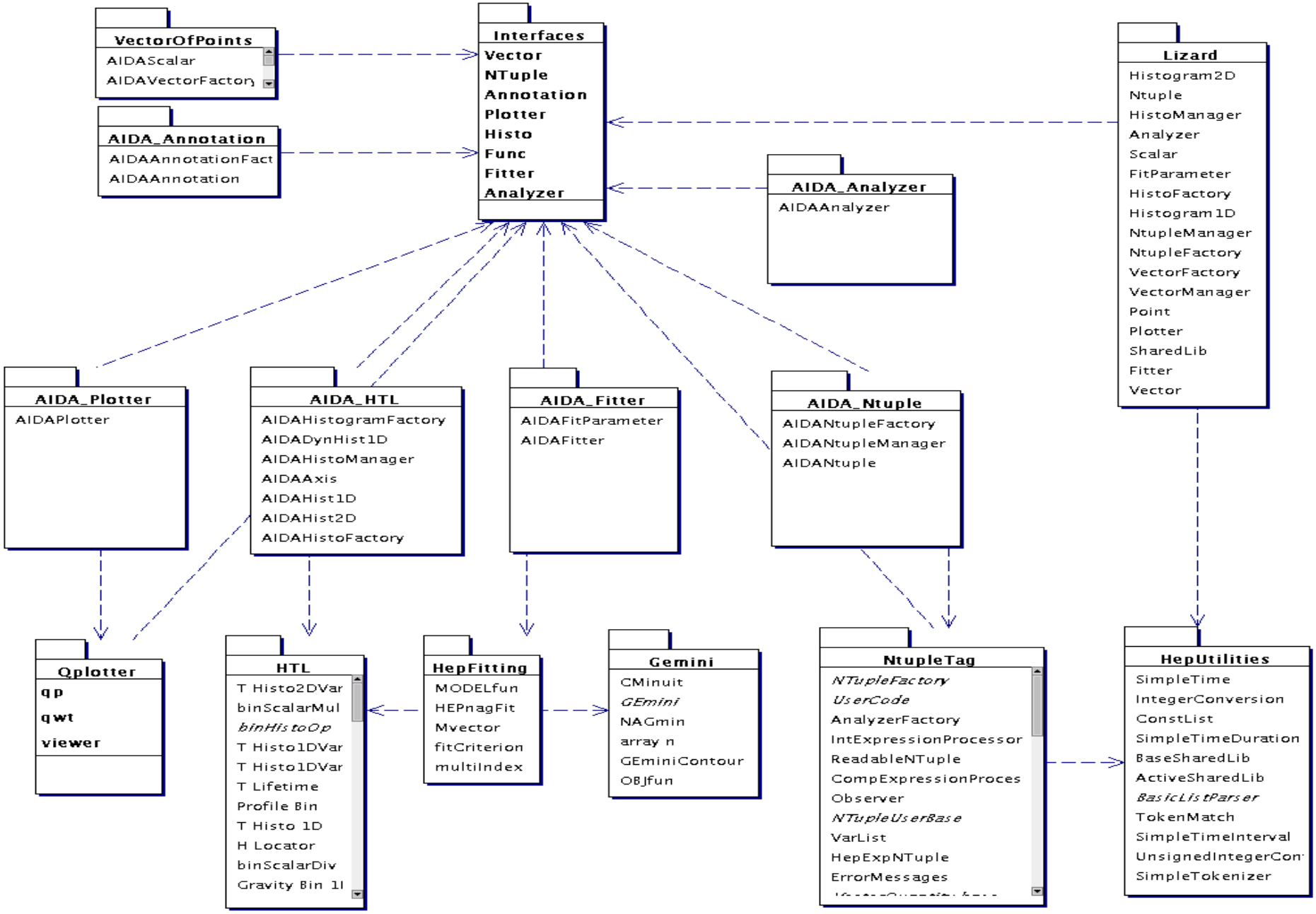
- AIDA project (Abstract Interfaces for Data Analysis) was initiated at the HepVis'99 workshop
  - Aiming at common set of Interfaces to ease use and interchange implementations (even across languages)
- Presently active mainly developers from existing packages
  - JAS (Tony Johnson), Lizard (Andreas Pfeiffer), OpenScientist (Guy Barrand), Wired (Mark Dönszelmann)
- Open to new ideas, projects, people, ...
- **aida.freehep.org/**

Anaphe -- 'Layered' Approach

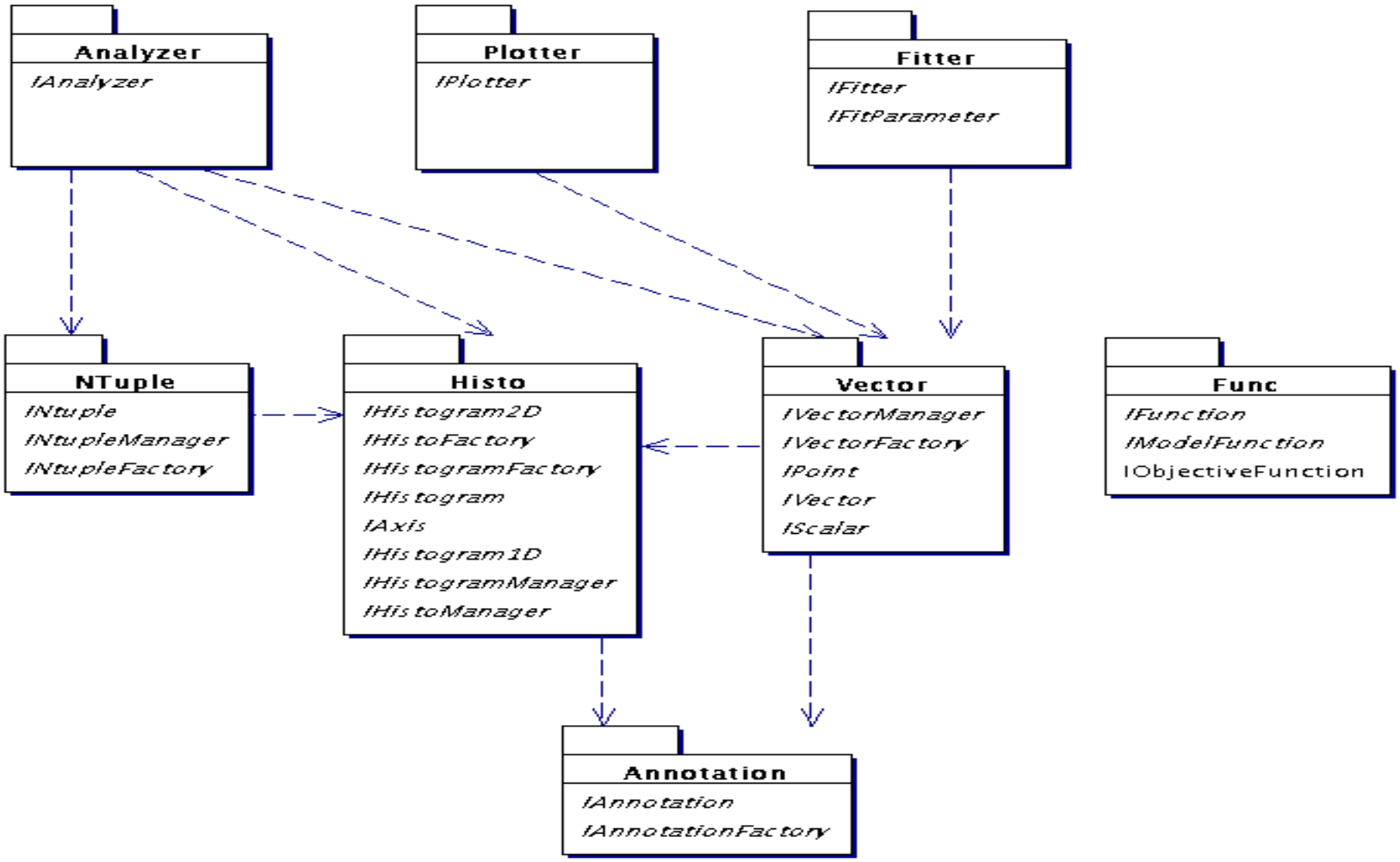
- Basic functionalities (histograms, fitting, etc.) are available as individual C++ class libraries.
- Insulate components through Abstract Interfaces
  - "wrapper" layer to implement Interfaces in terms of existing libs
- Easy replacing one part without throwing away everything
  - Objectivity/DB to provide persistence
  - HepODBMS library ("insulating layer", "tags")
  - Histogram library (HTL)
  - Fitting libraries (Gemini, HepFitting)
  - Graphics libraries (Qt, Qplotter)
  - Basic math for HEP (CLHEP, NAG-C)
  - example for migration: RW --> ObjectSpace --> native STL

Anaphe Components: Overview




VectorOfPoints
AIDAScalar
AIDAVectorFactory
AIDA_Annotation
AIDAAnnotationFact
AIDAAnnotation
Interfaces
Vector
NTuple
Annotation
Plotter
Histo
Func
Fitter
Analyzer
AIDA_Analyzer
AIDAAnalyzer
Lizard
Histogram2D
Ntuple
HistoManager
Analyzer
Scalar
FitParameter
HistoFactory
Histogram1D
NtupleManager
NtupleFactory
VectorFactory
VectorManager
Point
Plotter
SharedLib
Fitter
Vector
AIDA_Plotter
AIDAPlotter
AIDA_HTL
AIDAHistogramFactory
AIDADynHist1D
AIDAHistoManager
AIDAAxis
AIDAHist1D
AIDAHist2D
AIDAHistoFactory
AIDA_Fitter
AIDAFitParameter
AIDAFitter
AIDA_Ntuple
AIDANtupleFactory
AIDANtupleManager
AIDANtuple
Qplotter
qp
qwt
viewer
HTL
T Histo2DVar
binScalarMul
binHistoOp
T Histo1DVar
T Histo1DVar
T Lifetime
Profile Bin
T Histo 1D
H Locator
binScalarDiv
Gravity Bin 1l
HepFitting
MODELfun
HEPnagFit
Mvector
fitCriterion
multiIndex
Gemini
CMinuit
GEmini
NAGmin
array n
GEminiContour
OBJfun
NtupleTag
NTupleFactory
UserCode
AnalyzerFactory
IntExpressionProcessor
ReadableNTuple
CompExpressionProces
Observer
NTupleUserBase
VarList
HepExpNTuple
ErrorMessages
HepUtilities
SimpleTime
IntegerConversion
ConstList
SimpleTimeDuration
BaseSharedLib
ActiveSharedLib
BasicListParser
TokenMatch
SimpleTimeInterval
UnsignedIntegerCon
SimpleTokenizer

Lizard Internals: Interfaces



Interactive Data Analysis

- Aim: OO replacement “for PAW”
  - analysis of “ntuple-like data” (“Tags”, “Ntuples”, ...)
  - visualisation of data (Histograms, scatter-plot, “Vectors”)
  - fitting of histograms (and other data)
  - access to experiment specific data/code
- Maximize flexibility and re-use
- Foresee customization/integration
  - allow use from within experiment’s s/w
- Plan for extensions
  - “code for now, design for the future”

Architectural issue: Scripting

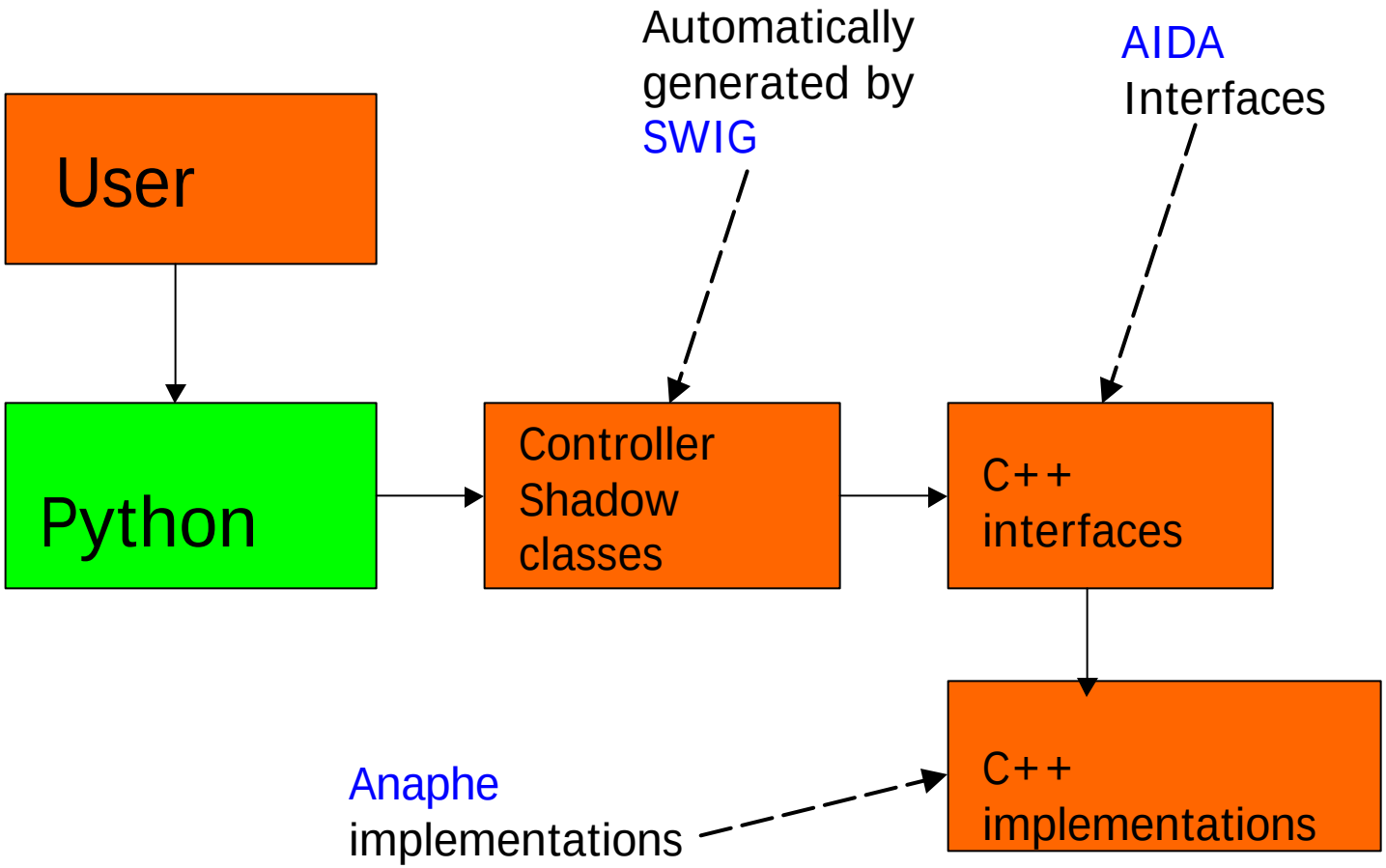
- Typical use of scripting is quite different from programming (reconstruction, analysis, ...)
  - history “go back to where I was before”
  - repetition/looping - with “modifiable parameters”
- Scripting language is an interface to the UserInterface component
  - Can be enhanced and/or replaced by a GUI
  - scripting window within GUI application
- Scripting language is “framework” to use “toolkit”
  - plug-and-play like loading of shared libraries
  - rapid development cycles
    - interactive working with classes

Scripting in Lizard

- Python - OO scripting, no “strange $!%-variables”
  - lots of extension modules available in public domain
    - e.g., plotting: bibbles
    - e.g., vector/matrix arithmetics: NumPy
    - ... and lots more
  - kind of an “intelligent shell”

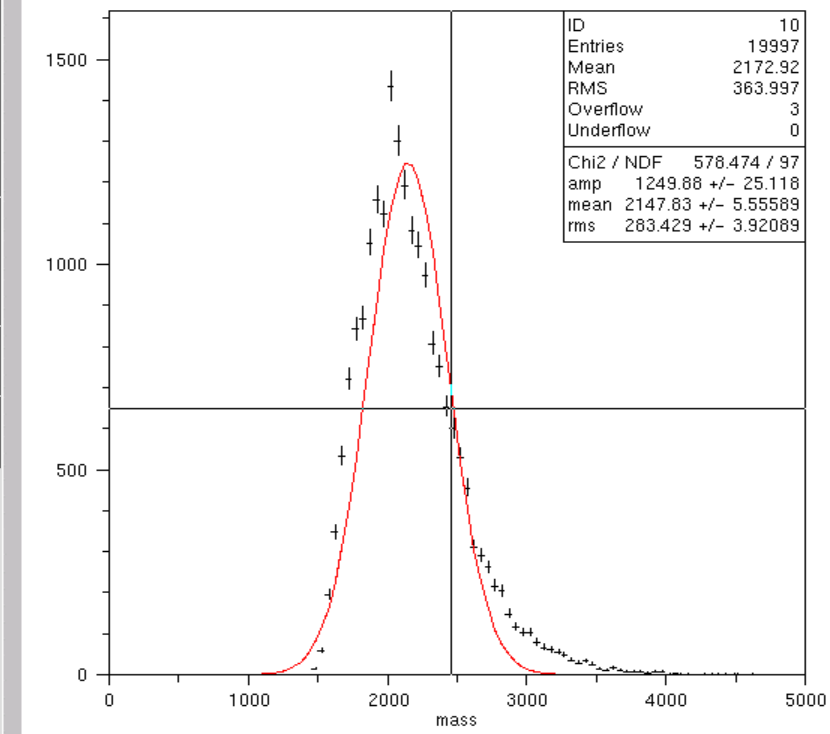
- Use of SWIG to create python “shadow classes”
  - possible change of public interface
  - can be used as normal python classes (extended by inheritance)
  - pointers are pointers to the actual C++ object
    - speed where it’s needed
  - allows flexibility to choose amongst several scripting languages

Scripting in Lizard



(Control Client) | xterm | xterm <2> | xterm <4> | emacs: ntup.py | Qplotter | (xterm <3>)

Qplotter



X Coordinate : 2456.9 Y Coordinate : 648.406

emacs: ntup.py

File Edit Apps Options Buffers Tools

```
ntm.listNtuples()                        # get list of names of all tuples from tupl

nt1=ntm.findNtuple("Charm1")             # retrieve tuple by name
# nt1.listAttributes()                   # prints names and types of attributes

h1=hm.create1D(10,"mass",100,0.,5000.)   # create 1D histo for one of the attribute
h2=hm.create1D(20,"mass for pt1>10",100,0.,5000.)

# for the updating plot, reset the plotter to a single zone
pl.zone(1,1)
h1.reset()
start = 0
num = 1000
for i in range(0.,20.):
    nt1.project1D(h1,"MASS","",start,num)
    vTemp=vm.from1D(h1)
    pl.plot(vTemp)
    del vTemp
    start=start+num

# that's it !
```

```
d classes initialised

:-) exe("ntup.py")

Explorables present:

        Charm1

:-) err()
:-) h1=hm.retrieveHisto1D(10)
:-) vFit = hfit(h1,"G")
+++Some of the bins have zero errors. They will be ignored in the chi-square fit.
+++Use setDefaultError(double) to redefine zero errors.

+++ Performing a chi-square fit.
:-)
```

One | Two | Three | Four | Five | Six | 15:05

Lizard -- development so far

- First prototype (with limited functionality) available since CHEP-2000
  - feedback from users on Python scripting IF
  - prototype not based on Abstract Interfaces
- Re-design started in April 2000
  - Beta version October 2000
- Full version out since June 2001 (Anaphe-3.6.0)
  - “PAW like” analysis functionality plus
  - on-demand loading of compiled code using shared libraries
    - gives full access to experiment’s analysis code and data
  - based on Abstract Interfaces
    - flexible and extensible

Present status and near future

- Working on “lite” version
  - limited functionality for Ntuples/Tags (through Python objects)
    - in memory, mainly user-written
  - using Minuit (CERNLIB)
  - no direct object persistency for Histograms
    - XML format to file or through VectorOfPoints

- Working on “license-free” version
  - based on Minuit (CERNLIB)
  - HBook-RWN as “alternative persistency” for Ntuples/Tags and Histograms
  - HBook CWN ?

Near to medium future

- Access to other implementations of components
  - OpenScientist (easy through AIDA)
  - direct reading of ROOT (>3.0) files (for histos and “trees”)
    - re-creation of “struct-like” “objects” (public attributes, no methods)
      - similar to “reading files with lost library” of ROOT
- Implement new AIDA interfaces
  - makes it easy to interchange implementation
    - can use AIDA compliant packages from other providers
- Improve “plug-and-play” mechanism
  - define and implement “plugin-manager”
    - use of more than one implementation of a category in the same session in parallel

Longer term future

- Communication with Java tools/packages
  - through AIDA “mapping” (JACO)
    - JAS, WIRED
- Optimized Ntuple file format (AIDA) ?
  - minimize overhead, allow complex structures
  - de-couple logical and physical representations
- Adding other “scripting” languages
  - Perl, Tcl, cint ?
- More Features
  - e.g., “picking” on plots
  - e.g., “life” histogram displays
  - ...

Architectural issue: Distributed Computing

- Very complex field where several basic questions are not yet answered
  - data to processes (classical “farm” approach) ?
  - processes to data (CORBA, JAVA RMI) ?
  - How to assess the “cost” of a “query”
    - *“I would like to have a brontosaurus steak, medium”*
      “Ok, it will be ready in about 4.2 million years ...”
- Easy if “hidden” in implementation of component(s)
  - e.g., distributed filling of Histograms in *Analyzer*
  - Abstract Interface hides the complexity, no change in tool(s) or user code needed

Summary

- The architecture of Anaphe shows some important items for flexible and modular data analysis:
  - weak coupling between components through use of Abstract Interfaces
  - Basic functionality is covered by C++ class libraries
- Major criteria are flexibility, extensibility and interoperability
  - example: GEANT-4 space examples using G4Analysis component (based on AIDA)
- Lizard is based on Anaphe components and the Python scripting language (through SWIG)
  - maximizes re-use of existing components
  - allows for easy extensions through modules from public domain
  - easy creation of “wrappers” using SWIG

More information

- **cern.ch/Anaphe**
- **cern.ch/Anaphe/Lizard**
- **aida.freehep.org/**
- **cern.ch/DB**
- **wwwinfo.cern.ch/asd/lhc++/clhep/**