



Modellistica Medica

Maria Grazia Pia

INFN Genova

Scuola di Specializzazione in Fisica Sanitaria

Genova

Anno Accademico 2002-2003

Lezione 35

Analysis Tools

AIDA

Anaphe

Collaborating Frameworks

In our applications:

● Simulation

- to model the experimental setup and generate effects from particle interactions

● User Interface

- to control configuration and flow of execution

● Visualisation

- detector geometry, particle tracks, hits etc.

● Analysis

- histograms, ntuples, fitting etc.

Components and Frameworks

- Frameworks are composed of components
- Abstract Interfaces de-couple components and frameworks
- **Frameworks**
 - correlated groups of classes (*components*) together with their interactions
 - re-usable (*generic*) designs of a software system on a very high abstraction level
- **Component**
 - a correlated group of classes together with their interactions
 - reusable design of (part of) a software system on a low or medium abstraction level


Weakly coupled components and frameworks have large advantages

- ease of re-use of component or framework
- flexibility through independence of implementation
- maintainability through independent evolution of components

Interface to external tools in Geant4

Through abstract interfaces

no dependence
minimize coupling of components

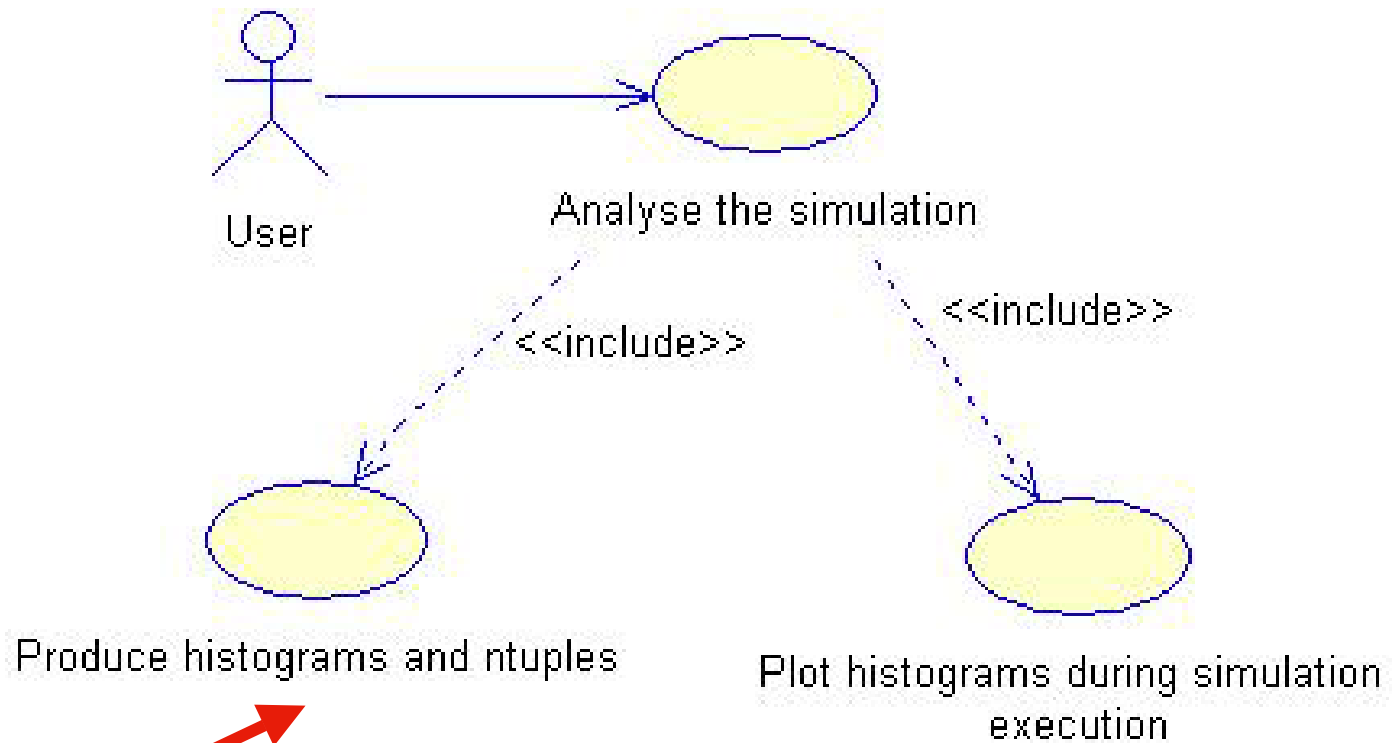


Similar approach

- Visualisation
 - (G)UI
 - Persistency
 - Analysis
- 

The user is free to choose the concrete system he/she prefers for each component

Use cases for analysis in a simulation application





AIDA

Abstract Interfaces

for

Data Analysis

 <http://aida.freehep.org>

Andreas Pfeiffer

CERN IT/API

`andreas.pfeiffer@cern.ch`

Outline



- What is AIDA
- History/Collaboration/Documentation
- Some Details
- Examples
- Ongoing work
- Summary

What is AIDA



- **Abstract Interfaces for Data Analysis (AIDA)**
- *"The goals of the AIDA project are to define abstract interfaces for common physics analysis objects, such as histograms, ntuples, fitters, IO etc.
The adoption of these interfaces should make it **easier** for developers and users to select to use different tools without having to learn new interfaces or change their code.
In addition it should be possible to exchange data (objects) between AIDA compliant applications."*

Motivation

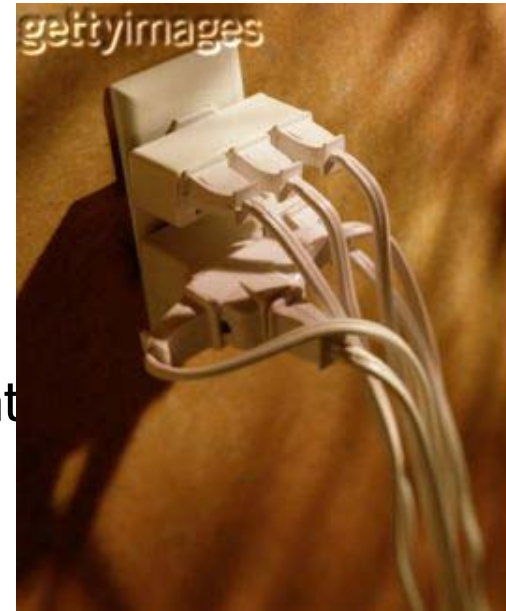


Advantages

- The user needs to learn only one set of interfaces
- Same user code can be used with different AIDA-compliant analysis applications
- Pool experience of different developer teams
 - LHC++, OpenScientist, JAS
- Different analysis tools can exchange analysis objects
 - same storage format, use functionality from other tools

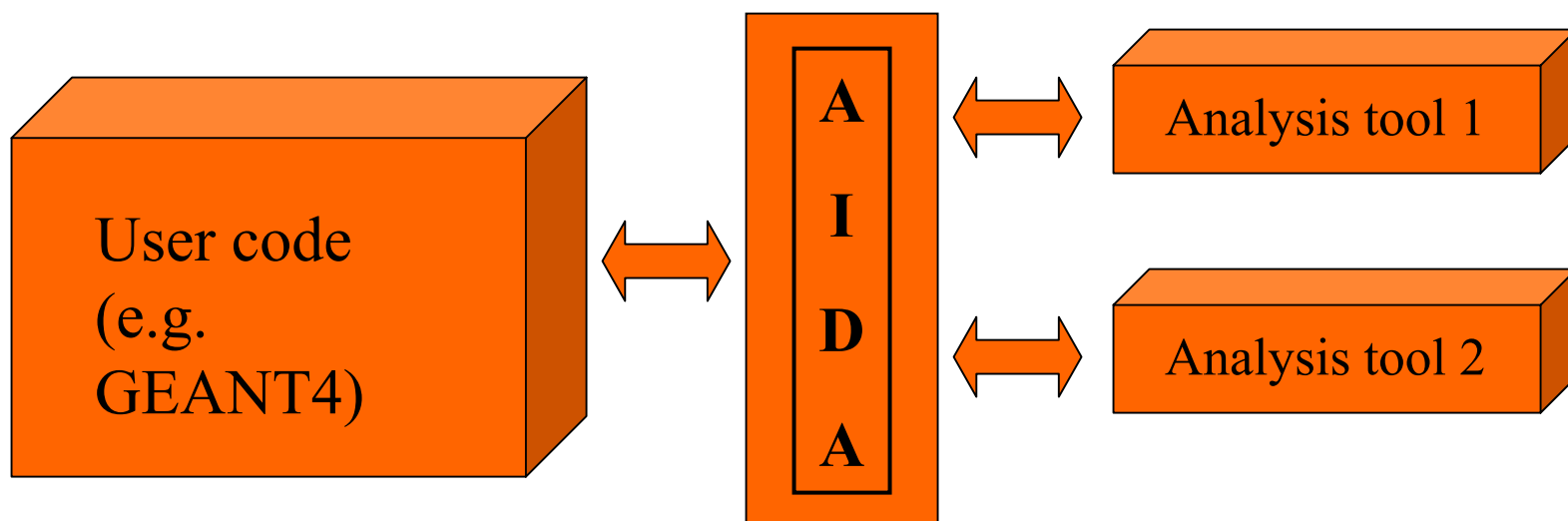
Abstract Interfaces

- **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
- De-couple **implementation** of a package from its **use**



AIDA Example

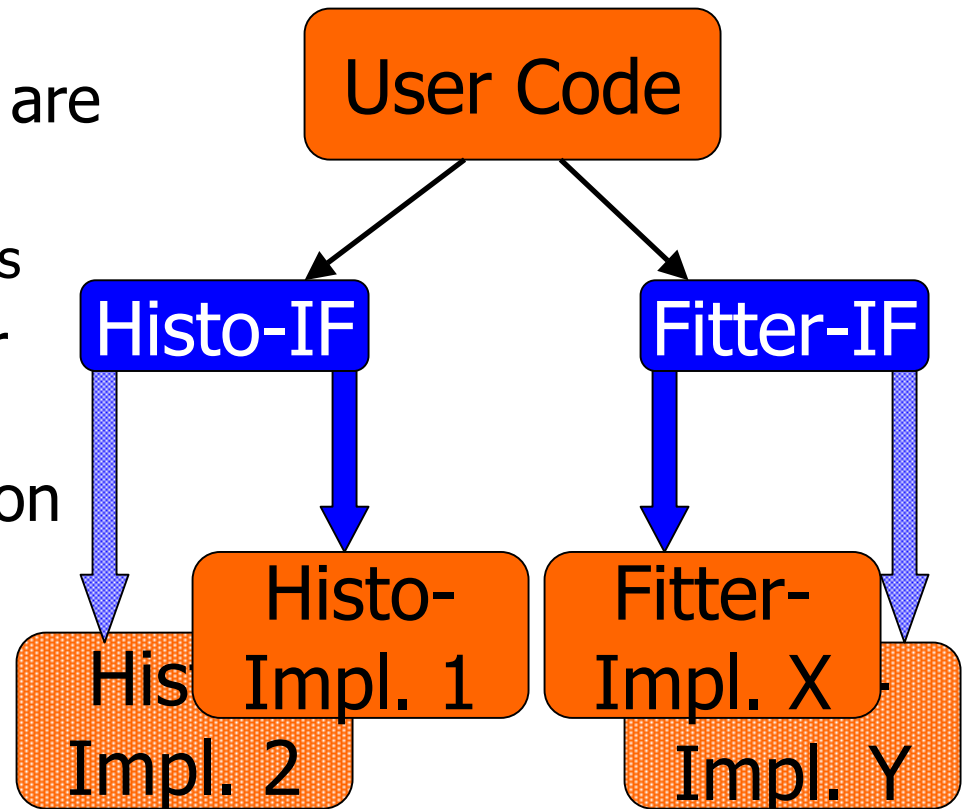
- Use same code with any AIDA-compliant analysis tool



- GEANT4 adopted AIDA for analysis

Use of Components with Abstract Interfaces

- User Code uses only Interface classes
- `IHistogram1D * hist = histoFactory-> create1D('E deposit', 100, 0., 10.)`
- Actual implementations are selected at run-time
 - loading of shared libraries
- No change at all to user code but keep freedom to choose implementation



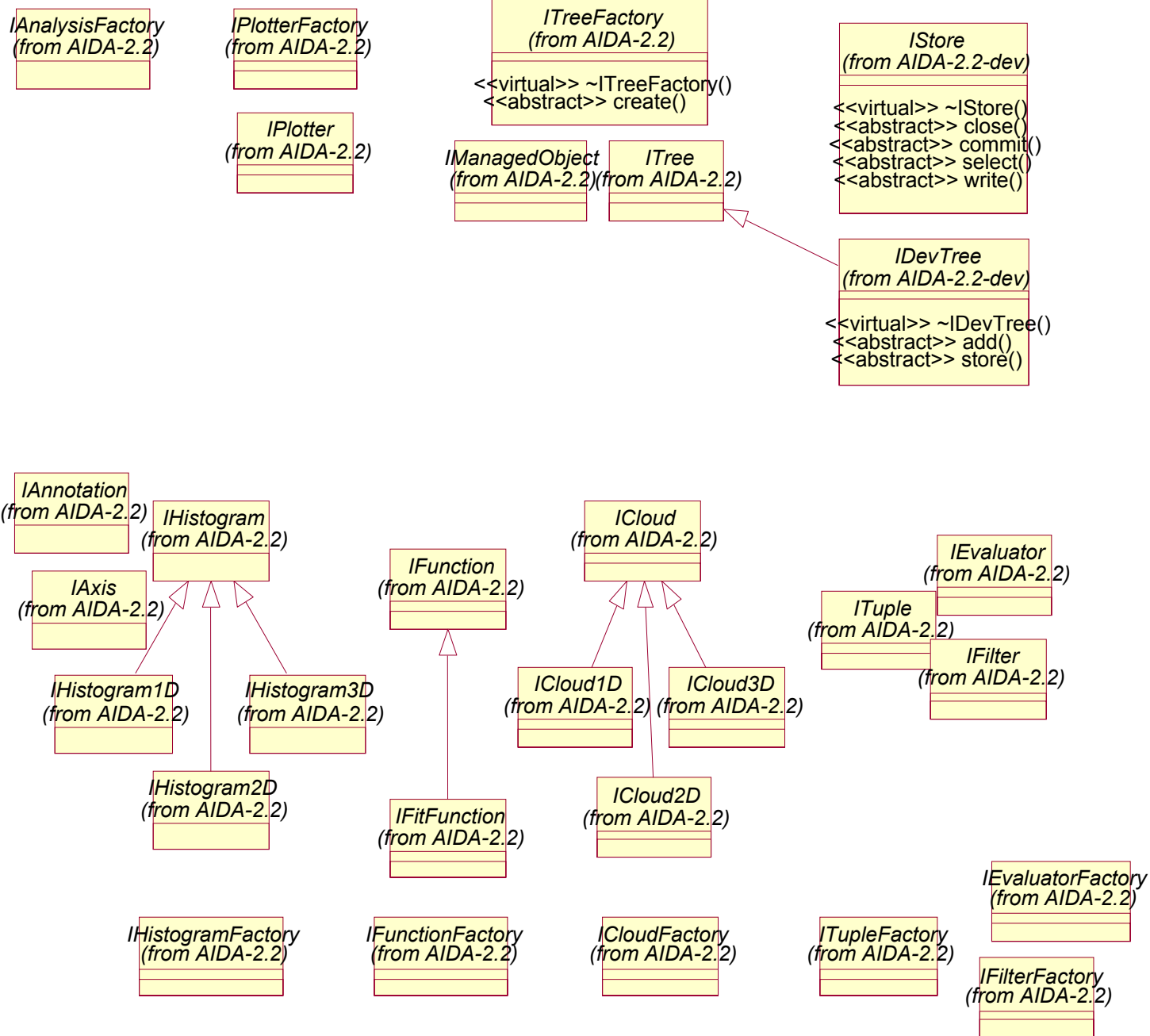
Systems implementing AIDA

- Three implementations of AIDA exist
 - **Anaphe/Lizard (C++)**
 - <http://anaphe.web.cern.ch/anaphe>
 - Open Scientist (C++)
 - <http://www.lal.in2p3.fr/OpenScientist>
 - JAIDA/JAS (Java) + AIDA-JNI 1.0 (C++)
 - <http://java.freehep.org/lib/freehep/doc/aida>

AIDA Interfaces Summary



- AIDA Factories
- ITuple
- IHistogram
- ICloud
- ITree



IHistogram (1D-3D)

- Binned histogram: **IHistogram1D**, 2D, 3D
 - “fill” methods (with/without weight)
 - Histogram info: entries, mean, rms, axis
 - Bin info: centre, entries, height, error
 - Histogram arithmetic: add, multiply, divide
 - Convenience methods, like coordinate-to-index conversion

ITuple



- **ITuple** - interface to the Data
 - "get/set" methods for double, float, int, ...
 - Information about columns: min, max, mean, rms
 - Navigating: start(), next(), skip(int nRows)
 - Project ITuple into 1D, 2D, 3D histogram
 - New features for AIDA 2.3:
 - Support for complex internal structures (subfolders)
 - Merging and chaining of ITuples under discussion

ICloud



- Unbinned collection of points: **ICloud1D**, 2D, 3D
 - Can represent scatter plot, dynamically rebinnable histogram
 - Can be converted to a binned histogram
 - Standard “get/set” methods for entries
 - Collection info: lower, upper, mean, rms

IFunction and Fitting

■ Fitting: **IFunction**, **IFitFunction**

- **IFunction** – simple interface, allows to set parameters and get function value
- **IFitFunction** – fit function to a histogram
 - Extends IFunction
 - Various fit control methods: step size, bounds, etc.
 - Allows to perform fit and get results
 - AIDA 2.2 fitting functionality fairly limited
- AIDA 2.3 (Under discussion) extended functionality

ITree

■ ITree

- directory-like structure (Unix directory convention)
 - Methods like: cd, ls, mkdir, etc.
- AIDA analysis objects (tuples, histograms, clouds, etc.) exist within ITree directories
- “save/restore” functionality, hides storage details from the user
 - Compatible with database or file storage
 - Can support multiple file formats
 - Mount/Unmount functionality (like unix) allows multiple stores to be seamlessly merged
 - AIDA XML format is defined for data interchange

Summary



- **Abstract Interfaces** de-couple **components** of **frameworks**
- **Weakly coupled** components and frameworks have large advantages
 - User code needs *no change* if changing implementation
 - Even across “language boundaries” (JAIDA)
 - Ease of **re-use** of a component
 - **Flexibility** through independence of implementation
 - **Maintainability** through independent evolution of components
- Geant4 adopted AIDA



Anaphe

OO Libraries for Data Analysis using C++ and Python

<http://anaphe.web.cern.ch/anaphe>

Andreas Pfeiffer

CERN IT/API

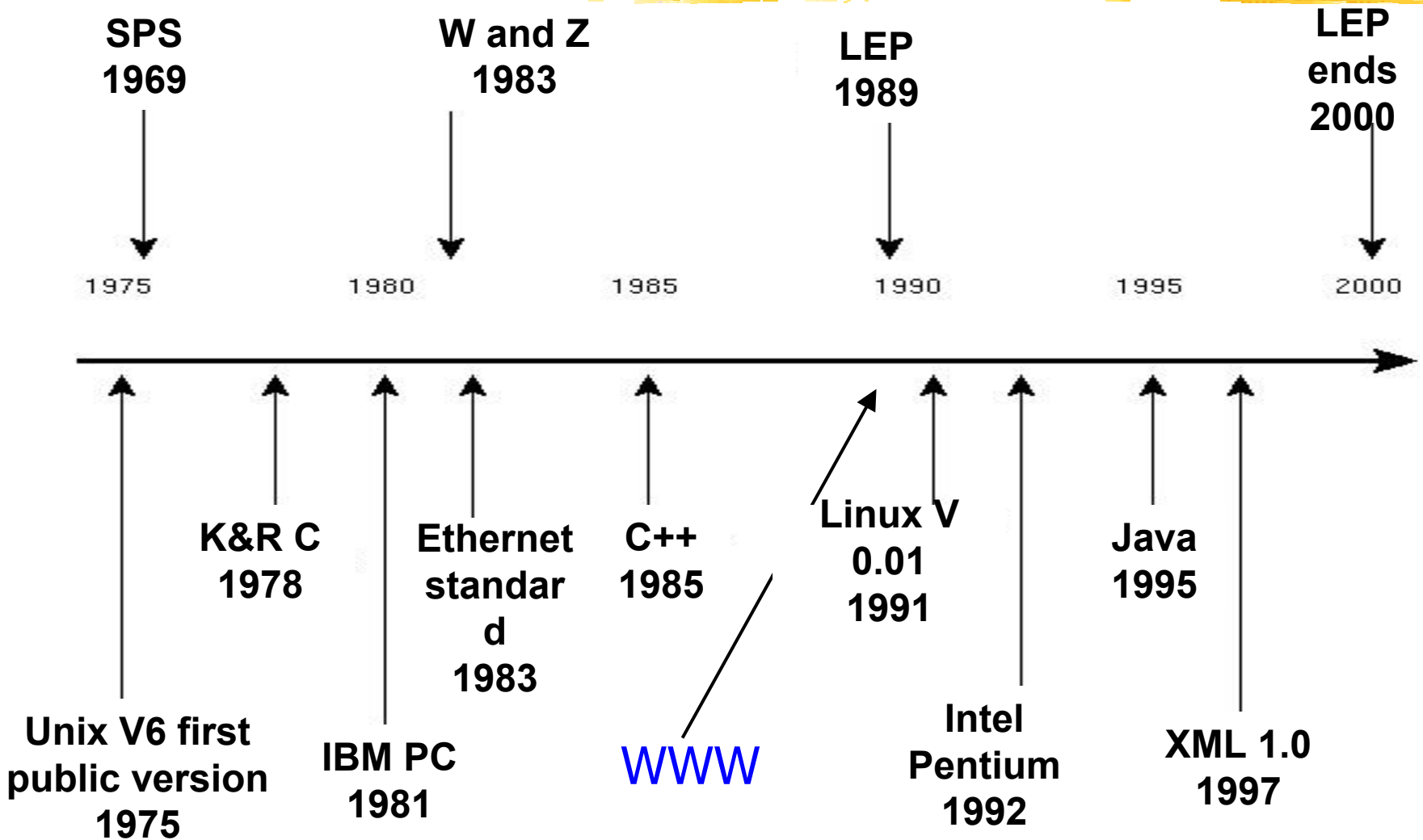
`andreas.pfeiffer@cern.ch`

Outline



- Motivation
- Anaphe Components
 - C++
- Lizard: Interactive Data Analysis
 - Python
- Summary

Lifetime of LHC software = 25 yrs



Anaphe: what it is

- Analysis for physics experiments
- Modular (OO/C++) replacement of *CERNLIB* functionality for use in HEP experiments
 - memory management
 - I/O
 - foundation classes
 - histogramming
 - minimizing/fitting
 - visualization
 - interactive data analysis
- Trying to use standards wherever possible
- Trying to re-use existing class libraries

'Layered' Approach



- Basic functionalities (histograms, fitting, etc.) are available as **individual C++ class libraries**.
- Easy replacing one part without throwing away everything
- Insulate components through **Abstract Interfaces**
- Apply s/w quality control tools
 - code checking, testing

Anaphe Components



Data Analysis	Lizard - AIDA
Custom graphics (2-D)	Qt - Qplotter
Basic graphics (3-D)	OpenInventor – OpenGL
Basic math	NAG C
HEP foundation	CLHEP
Minimization/Fitting	FML - Gemini
Histograms	HTL
Database	HepODBMS
Persistency	ODMG/Objectivity DB
C++	Standard Libraries

ANAPHE Components

Lizard

Interactive Commands

Histograms
NTuples
Fitting
Plotting
VectorOfPoints
Functions
Analyzer

AIDA
(Abstract
Interfaces for
Data Analysis)

HTL
Tags (HepODBMS
Gemini/HepFitting
Qplotter
VectorOfPoints

CLHEP
Class Libraries for
HEP

Python / SWIG

Objectivity/DB | HBook
NAG-C | Minuit
Qt (free edition)

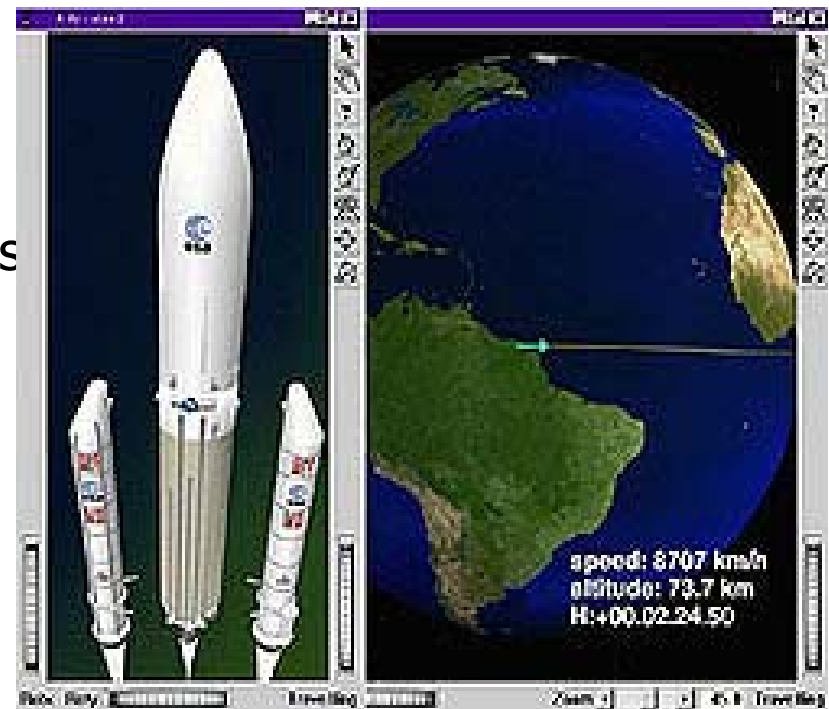
Abstract types

Implementations (HEP-specific)

non-HEP components

Basic 3D Graphic Libraries

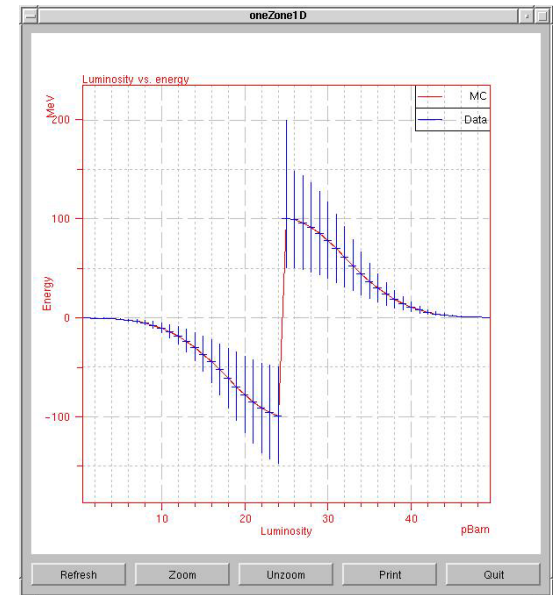
- **OpenGL** (basic graphics)
 - De-facto industry standard for basic 3D graphics
 - Used in CAD/CAE, games, VR, medical imaging
- **OpenInventor** (scene mgmt.)
 - OO 3D toolkit for graphics
 - Cubes, polygons, text, materials
 - Cameras, lights, picking
 - 3D viewers/editors, animation
 - Based on OpenGL/MesaGL



2D Graphics libraries

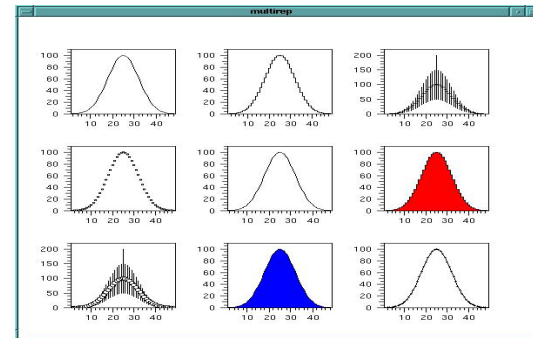
Qt

- multi-platform C++ GUI toolkit
 - C++ class library, not wrapper around C libs
 - superset of Motif and MFC
 - available on Unix and MS Windows
 - no change for developer
- commercial but with public domain version
- www.troll.no



Qplotter

- "add-on" functionality for HEP
 - "HIGZ/HPLOT"



Mathematical Libraries



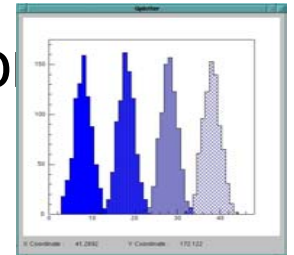
- **NAG** (Numerical Algorithms Group) **C Library**
 - Covers a broad range of functionality
 - Linear algebra
 - differential equations
 - quadrature, etc.
 - Special functions of CERNLIB added to Mark-6 release
 - mostly for theory and accelerator
 - Quality assurance
 - extensive testing done by NAG
 - www.nag.com

CLHEP - foundation classes

- HEP foundation class library
 - Random number generators
 - Physics vectors
 - 3- and 4- vectors
 - Geometry
 - Linear algebra
 - System of units
 - more packages recently added
 - will continue to evolve
- wwwinfo.cern.ch/asd/lhc++/clhep/

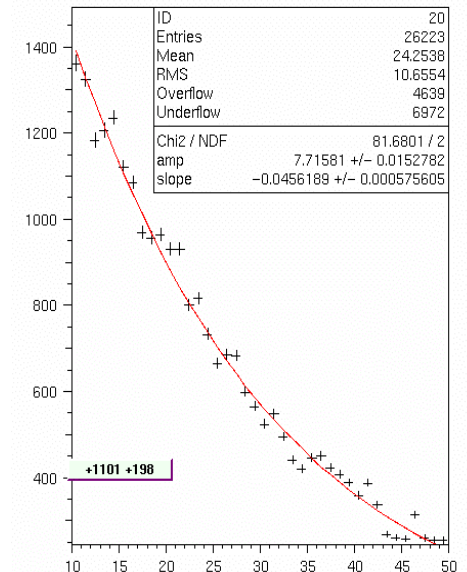
Histograms: the HTL package

- Histograms are the basic tool for physics analysis
 - **Statistical** information of density distribution
- **Histogram Template Library (HTL)**
 - design based on C++ templates
 - Modular : separation between sampling and display
 - Extensible : open for user defined binning systems
 - Flexible: support transient/persistent at the same time
 - Open: large use of abstract interfaces



Fitting and Minimization

- Fitting and Minimization Library (FML)
 - common OO interface
 - **NAG-C, MINUIT**
 - based on Abstract Interfaces
 - `IVector`, `IModelFunction`, ...
 - fitting as a special case of minimization
 - minimize “distance” between **data** and **model**
 - replacement for HepFitting (and Gemini)
- Gemini
 - common interface to minimizer engine
 - very thin layer



“Tags”, Ntuples and Events

- Tags - a special kind of Ntuple
 - Always associated with an underlying **persistent store**
 - Tags may be used to store “ntuple-like” data
 - extracted from all over the event
 - minPt, maxEmiss, nJets, nMuon, trigger, ...
- Main use: **speedup** data selection for analysis ...
 - Tag simplifies selection without losing complexity
- Association from the Tag to the Event may be used to **navigate** to any other part of the Event
 - even from an interactive visualization program



Lizard: a tool for Interactive Data Analysis

Interactive Data Analysis

- Aim: “OO replacement for PAW” (at least)
 - **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”
- Ensure **maintainability**
 - use of s/w quality control tools

Scripting - why

- Typical use of **scripting** is quite different from programming (reconstruction, analysis, ...)
 - **history** "go back to where I was before"
 - **repetition/looping** - with "modifiable parameters"
- avoid "one size fits all" or "using power-tool as hammer"
 - **rapid prototyping** in "scripting language"
 - quick turn-around times
 - performance **critical code** in "core language"
 - exploit richer set of features/functionality (e.g. templates in C++)
- scripting languages usually less susceptible to changes than "mainstream languages"
 - potentially longer lives

Python - why

- Python - OO (scripting) language
 - ✦ no "strange \$!%-variables"
 - ⇕ sensitive to indentation
- More **easy for users**
 - as Java
- **Lots of user supplied modules** available and ready for use
 - scientific, numerics, graphics, GUI, network, OS, games, DBs, ...
 - example: <http://www.vex.net/parnassus/>
 - Parnassus Totals: 1173 items in 49 categories.
- Also usable in Java (Jython)
 - used in JAS for scripting
 - minimize changes needed within AIDA compliant environments

Python - how

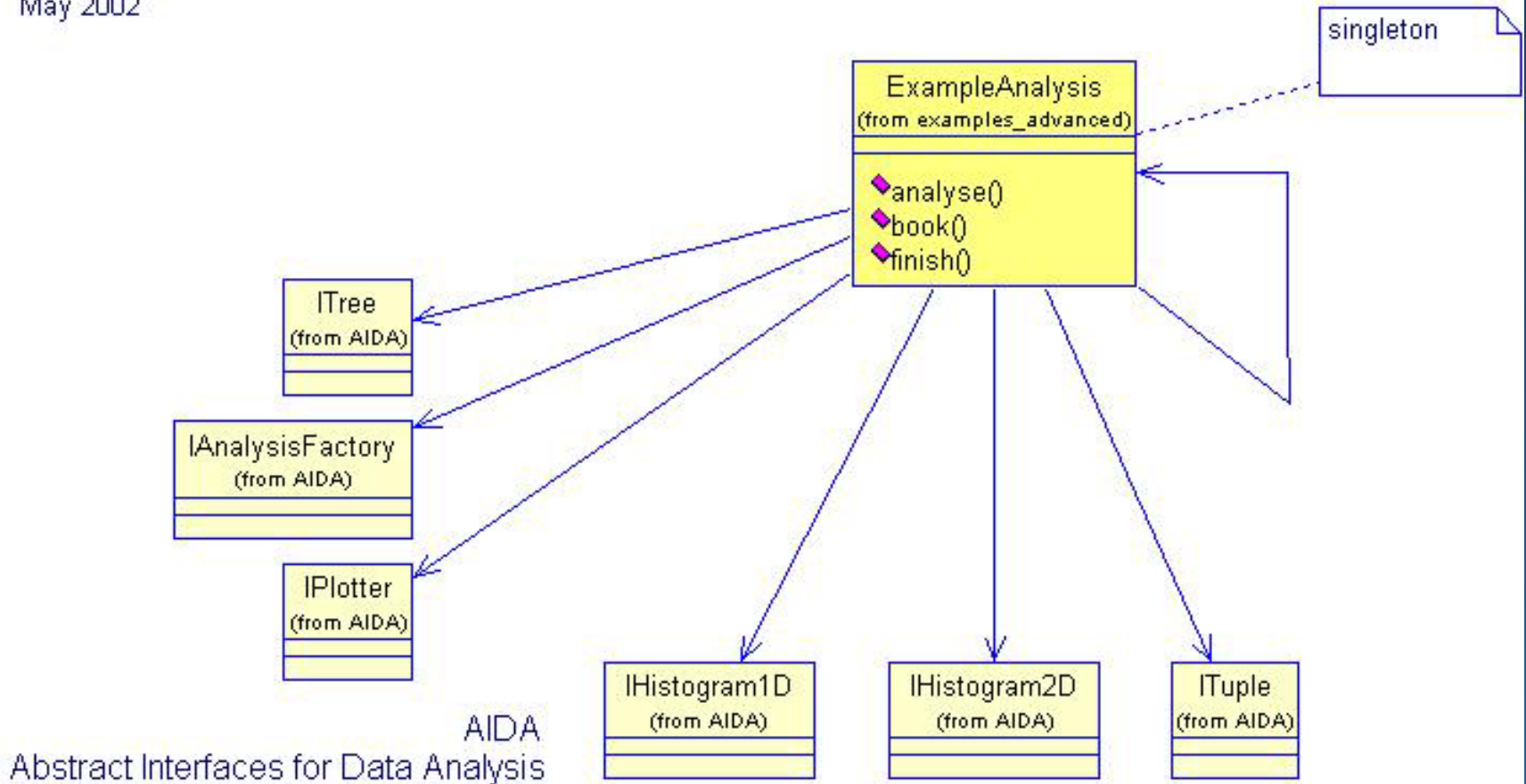
- **SWIG** to (semi-) automatically create connection to chosen scripting language
 - allows flexibility to choose amongst several scripting languages
 - **Python**, Perl, Tcl, Guile, Ruby, (Java) ...
- Very easy to **use**
 - **swig -c++ -python -shadow -c myClass.h**
 - create shared lib from **myClass.cpp** and **myClass_wrap.c**
 - start python and **import myClass.h** to use it
- Very easy to **extend**
 - simply inherit from "swiggified" class in python
 - modifications can later be fed back into C++
 - performance, type safety, special language features (templates), ...

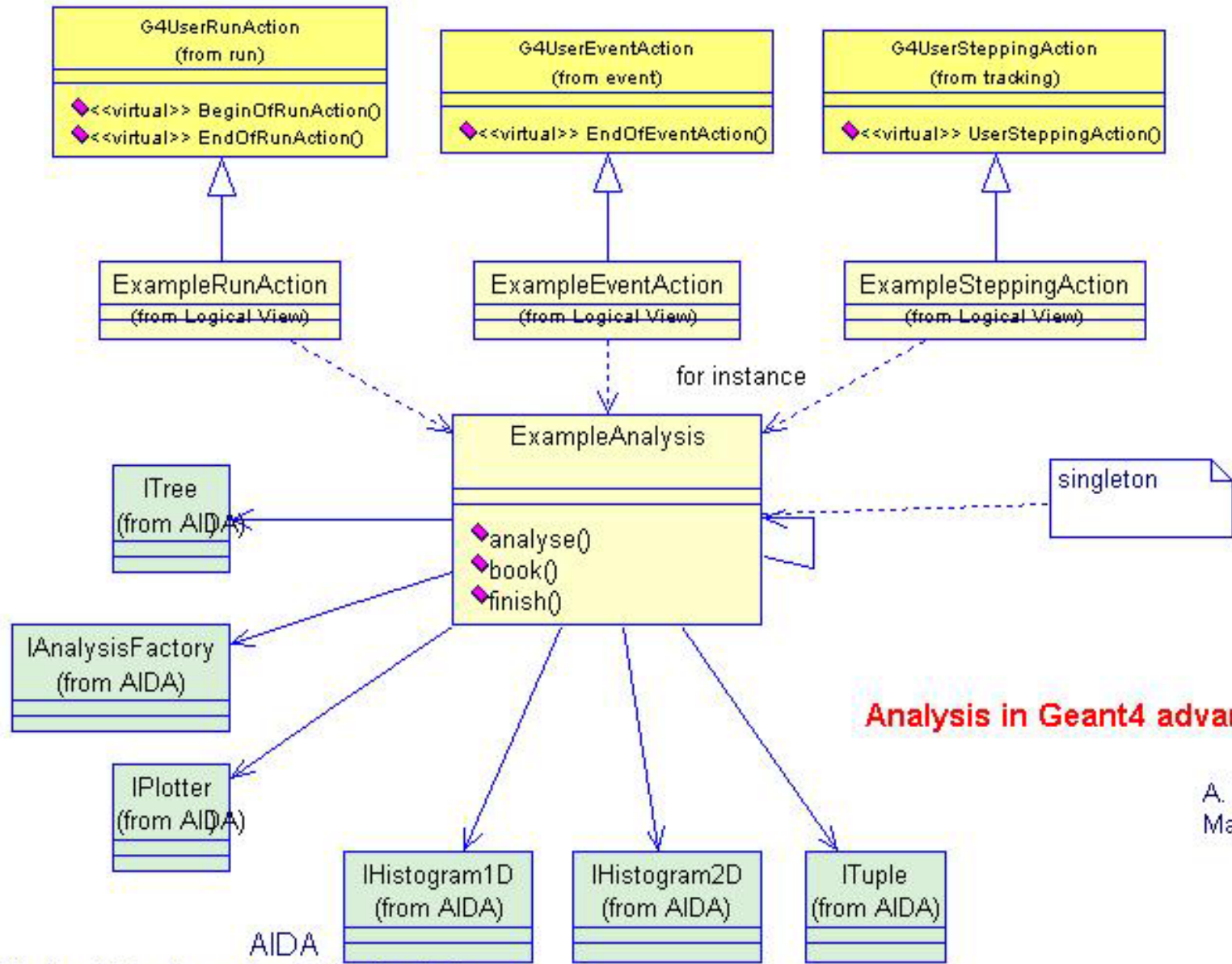
Design choices in Geant4 applications

- **Analysis is based on AIDA**
 - Independence of implementation: Anaphe, JAS, OpenScientist can be used interchangeably without changing code
- **All analysis is concentrated in a Singleton**
 - Eases access from several different classes
- **Analysis is done in the User* classes, the Singleton only provides access to the analysis objects**
 - Histograms, tuple

Analysis in Geant4 advanced examples

A. Pfeiffer, M.G. Pia
May 2002





Analysis in Geant4 advanced examples!

A. Pfeiffer, M.G. Pia
May 2002

Abstract Interfaces for Data Analysis

Analysis in Geant4 advanced examples

A. Pfeiffer, M.G. Pia
May 2002

