Anaphe
status and prospects

Andreas Pfeiffer
CERN IT/API
andreas.pfeiffer@cern.ch
Outline

Introduction to Anaphe
- What, why, how

The AI DA project
- Abstract Interfaces for Data Analysis

Lizard
- Interactive Data Analysis using C++ and Python

Summary
Anaphe

OO libraries for data analysis
Anaphe: what it is

- **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
  - simulation

- Trying to use **standards** wherever possible
- Trying to **re-use** existing class libraries
## Anaphe Components

<table>
<thead>
<tr>
<th>Category</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Analysis</td>
<td>Lizard - AIDA</td>
</tr>
<tr>
<td>Custom graphics (2-D)</td>
<td>Qt - Qplotter</td>
</tr>
<tr>
<td>Basic graphics (3-D)</td>
<td>OpenInventor – OpenGL</td>
</tr>
<tr>
<td>Basic math</td>
<td>NAG C</td>
</tr>
<tr>
<td>HEP foundation</td>
<td>CLHEP</td>
</tr>
<tr>
<td>HEP math</td>
<td>FML - Gemini - CLHEP</td>
</tr>
<tr>
<td>Histograms</td>
<td>HTL</td>
</tr>
<tr>
<td>Database</td>
<td>HepODBMS</td>
</tr>
<tr>
<td>Persistency</td>
<td>ODMG/Objectivity DB</td>
</tr>
<tr>
<td>C++</td>
<td>Standard Libraries</td>
</tr>
</tbody>
</table>
“Structured” 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
  - “wrapper” layer to implement Interfaces in terms of existing libs

- Apply s/w quality control tools
  - code checking, testing

- Consider long lifetime needed for LHC
  - Changes are inevitable
  - “Code for now, design for the future”
Lifetime of LHC software = 25 yrs

Inizio SPS 1976
1975

Osservazione Z e W 1983
1980

Inizio LEP 1989
1985

Fine LEP 2000
2000

K&R C language 1978
1975

Stroustrup C++ Programming Language
1983

Linux V 0.01 Utemli: 1 1991
1990

Intel Pentium 1992
1995

Sun annuncia Java 1995
1995

UNIX V6 Prima versione pubblica 1975
1975

IBM PC 1981
1981

XML 1.0 1997
1997

WWW

Thanks to Dino Ferrero Merlino
Architectural issue: Components

- Identify components by **functionality**
  - not by “historic use”

- 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/...
  - “User’s view” is different from “implementor’s view”
    - separate Abstract Interfaces for both aspects
    - “command-layer” vs. “implementation-layer(s)”

- Identify and **use patterns - avoid anti-patterns**
  - learn from other people’s experiences/failures
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
  - independent components ease maintenance

- AIDA project (Abstract Interfaces for Data Analysis) was initiated at the HepVis’99 workshop
  - aida.freehep.org/
Anaphe Internals:
(Abstract) Interfaces
AIDA
Abstract Interfaces for Data Analysis
The AIDA project

- AIDA project (Abstract Interfaces for Data Analysis) was initiated at the HepVis’99 workshop
  - since then several (mini-) workshops (main ones Paris 2000 and Boston 2001) have been organized

- Presently active mainly developers from existing packages
  - JAS (Tony Johnson)
  - Lizard (Andreas Pfeiffer)
  - OpenScientist (Guy Barrand)
  - Wired (Mark Dönszelmann)
Introduction

 DESIGN Interfaces for Data Analysis (in HEP)

“*The goals of the AIDA project are to define abstract interfaces for common physics analysis tools, such as histograms. 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.*”

Open for contributions of any kind

suggestions, code, implementations, …
History

- started in Sept. 1999 (HepVis’99, Orsay)
- release 1.0 summer 2000
  - concentrated on “developers view”
  - Histograms only
- release 2.0 spring 2001
  - much more complete set -- for feedback on design
  - Concentrated discussions on
    - Management
      - directory like Tree
    - Store
    - Tuple
Status

- release 2.2 fall 2001
  - refinements after collecting feedback and starting to implement
  - About 25 Interface classes were defined
    - written down in Java and checked into FreeHep-CVS
- Used by Gaudi/Athena (LHCb, Atlas, Harp)
  - Gaudi people involved in design
- Used by IGUANA project (CMS)
- Adopted and used in Geant-4 examples
# Status of Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram</td>
<td>close to complete, implementations exist</td>
</tr>
<tr>
<td>Tree</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>Cloud</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>Store</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>Tuple</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>Function</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>Fitter</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>MasterFactory</td>
<td>discussed, being implemented</td>
</tr>
<tr>
<td>Plotter/Document</td>
<td>discussion not yet started</td>
</tr>
</tbody>
</table>
Analysis Tools

- **Java Analysis Studio**
  - started on reference implementation
  - JAIDA package to access Java components from C++
    - C++ proxies implement AIDA using Java Interfaces

- **Open Scientist**
  - most advanced in implementation

- **Lizard / Anaphe**
  - will provide parallel implementation for some time for user convenience

- All plan to have full implementation by end 2001
Lizard: a tool for Interactive Data Analysis
Scripting in Lizard

- Typical use of **scripting** is quite different from programming (reconstruction, analysis, ...)
  - **history** “go back to where I was before”
  - **repetition/looping** - with “modifiable parameters”

- **SWIG** to (semi-) automatically create connection to chosen scripting language
  - allows flexibility to choose amongst several scripting languages
  - **Python**, Perl, Tcl, Guile, Ruby, (Java) ...

- **Python** - OO (scripting) language
  - stable language for user-interface
  - independent of language for reconstruction/analysis

- Make use of language specific features as needed
  - templates in C++, interactivity in Python
Example script (ntuple)

# get list of names of all tuples from tuplemanager
ntm.listTuples()
nt1=ntm.findNtuple("Charm1")  # retrieve tuple by name

# create 1D histos to project into
h1=hm.create1D(10, "mass", 100, 0., 5000.)
h2=hm.create1D(20, "mass for pt1>10", 100, 0., 5000.)

# project the attribute "MASS" into histo h1 without cut ("")
nt1.project1D( h1, "" , "MASS")

# project the attribute "MASS" into histo h2 with cut ("PT1>10")
nt1.project1D( h2, "PT1>10" , "MASS")
ntm.listTuples()  # get list of names of all tuples from tuple
ntm.findTuple("Charm1")  # retrieve tuple by name
ntm.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):
    ntm.project1D(h1, "mass", "", start, num)
vtmp=gm.front1D(h1)
pl.plot(vtmp)
del vtmp
start=start+num

# that's it!

Class specific libraries executed:

```
(->) exec("ntup.py")

Explorables present:

Charm1

(->) err()
(->) h1=hm.retrieveHist1D(10)
(->) wfit = 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.
```

---
Lizard: History and Present Status

- Started after CHEP-2000
- Full version out since June 2001
  - "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
- "License free" version since fall 2001
  - uses HBook and Minuit
  - flexible selection of configuration automatically or at startup
    - lizard --noLicense
    - lizard --useHBook --useMinuit
Future Enhancements

- Implementation of AIDA version 2.x Interfaces
  - also: JAS (Tony Johnson) and OpenScientist (Guy Barrand)
- Access to other implementations of components
- Reading of ROOT (> V3.0) files
  - similar to Tony Johnson’s (Java) RootIO package
- AIDA Ntuple/Histo store
  - optimized for Ntuples; Histograms as (compressed) XML
- Communication with Java tools/packages (JAS, Wired)
  - via AIDA
- Adding other scripting languages
- GUI from Iguana?
Software quality control

Using tools for testing/checking has started
- Insure++, CodeWizard

Package dependencies: Igynominy
- Set of perl and shell scripts by Lassi Tuura (CMS)
- Igynominy scans...
  - Make dependency data produced by the compilers (*.d files)
  - Source code for #includes (resolved against the ones actually seen)
  - Shared library dependencies (“ldd” output)
  - Defined and required symbols (“nm” output)
- And maps...
  - Source code and binaries into packages
  - #include dependencies into package dependencies
  - Unresolved/defined symbols into package dependencies

ignominy: dishonour, disgrace, shame; infamy; the condition of being in disgrace, etc.

(Oxford English Dictionary)
Ignominy Analysis of Anaphe

Distribution of tools and utilities for LHC era physics
- Combination of commercial, free and HEP software
- Claims to be a toolkit

Seems to live up to its toolkit claims
- Good work on modularity
- Clean design is evident in many places
- Dependency diagrams often split naturally into functional units

Thanks to Lassi Tuura (CMS)
Challenge: Distributed Computing

- **Motivation**
  - move code to data
  - parallel analysis

- **Techniques**
  - services via AI
  - late binding
  - plug-in architecture

- **End-user (Lizard)**
  - look-and-feel of local analysis

- **R&D started and first prototype available soon**
  - CORBA based
Summary

The architecture of Anaphe shows some important items for flexible and modular data analysis:

- weak coupling between components through use of Abstract Interfaces
  - standards defined by AIDA project
- basic functionality is covered by individual C++ class libraries
- emphasis on usability and maintainability

Major criteria are flexibility, extensibility and interoperability

- recent example: GEANT-4 examples using analysis components based on AIDA

Lizard is an Interactive Data Analysis Tool based on Anaphe components and the Python scripting language (through SWIG)

- Lizard is young but has very solid base in mature Anaphe libraries
- real plug-in structure
More information

- cern.ch/ Anaphe
- cern.ch/ Anaphe/ Lizard
- aida.freehep.org/
Users and Collaborations

AI DA spoken here!

- IGUANA (CMS visualization)
- GAUDI (LHCb/HARP) framework
- ATHENA (Atlas) framework
- Analyzer modules in Geant 4
- JAS
- Open Scientist
- ...you?
# Package Metrics

<table>
<thead>
<tr>
<th>Project</th>
<th>Release</th>
<th>Packages</th>
<th>Average # of direct dependencies</th>
<th>Cycles (Packages Involved)</th>
<th># of levels</th>
<th>ACD*</th>
<th>CCD*</th>
<th>NCCD*</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaphe</td>
<td>3.6.1</td>
<td>31</td>
<td>2.6</td>
<td>--</td>
<td>8</td>
<td>5.4</td>
<td>167</td>
<td>1.3</td>
<td>630/170k</td>
</tr>
<tr>
<td>ATLAS</td>
<td>1.3.2</td>
<td>230</td>
<td>6.3</td>
<td>2 (92)</td>
<td>96</td>
<td>70</td>
<td>16211</td>
<td>10</td>
<td>1350k</td>
</tr>
<tr>
<td></td>
<td>1.3.7</td>
<td>236</td>
<td>7.0</td>
<td>2 (92)</td>
<td>97</td>
<td>77</td>
<td>18263</td>
<td>11</td>
<td>1350k</td>
</tr>
<tr>
<td>CMS/ORCA</td>
<td>4.6.0</td>
<td>199</td>
<td>7.4</td>
<td>7 (22)</td>
<td>35</td>
<td>24</td>
<td>4815</td>
<td>3.6</td>
<td>420k</td>
</tr>
<tr>
<td>CMS/COBRA</td>
<td>5.2.0</td>
<td>87</td>
<td>6.7</td>
<td>4 (10)</td>
<td>19</td>
<td>15</td>
<td>1312</td>
<td>2.7</td>
<td>180k</td>
</tr>
<tr>
<td>CMS/IGUANA</td>
<td>2.4.2</td>
<td>35</td>
<td>3.9</td>
<td>--</td>
<td>6</td>
<td>5.0</td>
<td>174</td>
<td>1.2</td>
<td>150/38k</td>
</tr>
<tr>
<td>Geant4</td>
<td>4.3.2</td>
<td>108</td>
<td>7.0</td>
<td>3 (12)</td>
<td>21</td>
<td>16</td>
<td>1765</td>
<td>2.8</td>
<td>680k</td>
</tr>
<tr>
<td>ROOT</td>
<td>2.25/05</td>
<td>30</td>
<td>6.4</td>
<td>1 (19)</td>
<td>22</td>
<td>19</td>
<td>580</td>
<td>4.7</td>
<td>660k</td>
</tr>
</tbody>
</table>

*) John Lakos, Large-Scale C++ Programming

- **Size** = total amount of source code (not normalised across projects!)
- **ACD** = average component dependency (~ libraries linked in)
- **CCD** = sum of single-package component dependencies over whole release
  - Indicates testing/integration cost
- **NCCD** = Measure of CCD compared to a balanced binary tree
  - A good toolkit’s NCCD will be close to 1.0
  - < 1.0: structure is flatter than a binary tree (= independent packages)
  - > 1.0: structure is more strongly coupled (vertical or cyclic)

**Aim:** NCCD ~ 1 for given software/ functionality

Thanks to Lassi Tuura (CMS)
Metrics: NCCD vs ACD

NCCD vs Av. Component Deps (Fraction of Packages)

Toolkits & Frameworks

ATLAS
ORCA
G4
COBRA
IGUANA
Anaphe
ROOT

Thanks to Lassi Tuura (CMS)
Metrics: NCCD vs Cycles

Fraction of Packages in Cycles

NCCD

Toolkits & Frameworks

ATLAS
ORCA
G4
COBRA
Anaphe
IGUANA
ROOT

Thanks to Lassi Tuura (CMS)

15-Nov-2001
Andreas Pfeiffer, CERN/IT-API

0% 10% 20% 30% 40% 50% 60% 70%
Analysis of Geant4

- Fairly large C++ project
  - Very fine-grained (and multi-level) package structuring
  - Seems quite clean from the preliminary analysis

- Fine package subdivision helps in many ways but makes analysis and code understanding more complicated

- One subsystem seems strongly coupled and needs attention

- Need to study the use of the internal command system

Thanks to Lassi Tuura (CMS)
Analysis of ROOT

ROOT developers have done a formidable job of breaking binary (shared library) dependencies, but…

For example: By static analysis, nothing seems to use the postscript package directly (no incoming dependencies), but there is this code:

```c
void TPad::Print (const char *filename, Option_t *option) { […]
    TVirtualPS *psave = gVirtualPS;
    if (gROOT->LoadClass("TPostScript","Postscript")) return;
    gROOT->ProcessLineFast("new TPostScript()");
    gVirtualPS->Open(psname,pstype);
    gVirtualPS->SetBit(kPrintingPS); […] }
```

Taking these and global objects into account makes the dependency diagrams very different

Sign of fast growth? Need a “next evolutionary step”?

So “coherent” that replacing parts could get painful…

Thanks to Lassi Tuura (CMS)
Analysis of ROOT...
Metrics: NCCD vs Size

Size (k-lines of source [files])

NCCD

Tools & Frameworks

ATLAS
ORCA
ROOT
G4
COBRA
Anaphe
IGUANA

Thanks to Lassi Tuura (CMS)
Andreas Pfeiffer, CERN/IT-API

15-Nov-2001
Metrics: NCCD vs AID

- Toolkits & Frameworks
  - ATLAS
  - ORCA
  - Anaphe
  - IGUANA
  - COBRA
  - G4
  - ROOT

Thanks to Lassi Tuura (CMS)
Metrics: Packages vs Size

- ATLAS
- ORCA
- COBRA
- G4
- IGUANA
- Anaphe
- ROOT

Tools & Frameworks

Thanks to Lassi Tuura (CMS)

Andreas Pfeiffer, CERN/IT-API
Metrics: Packages vs Size

Toolkits & Frameworks

Thanks to Lassi Tuura (CMS)

Andreas Pfeiffer, CERN/IT-API