Discussion with Blueprint RTAG

August 2002

Tony Johnson

SLAC

Topics Covered

- Architectural issues
  - Use of (Abstract) Interfaces to create modular OO software
    - Real Life Examples
  - Comments on Software Bus
  - Comments on Python *vs*. Cint *etc*.
- Many Topics not covered (in any detail)
  - Java (still the best way to write HEP software), JAS, JAS 3, FreeHEP, AIDA, Wired, HepRep, GRID Integration, *etc*.

Abstract Interfaces

- Specifies functionality of software component independent of its implementation
  - Allows multiple implementations
    - Algorthm1/Algorithm2
    - Batch/Interactive
    - Present/Future
- Such an essential element of OO design that newer languages such as Java define special construct “interface” on a par with “class”
  - C++ users must use “pure virtual classes”

Why Use Interfaces?

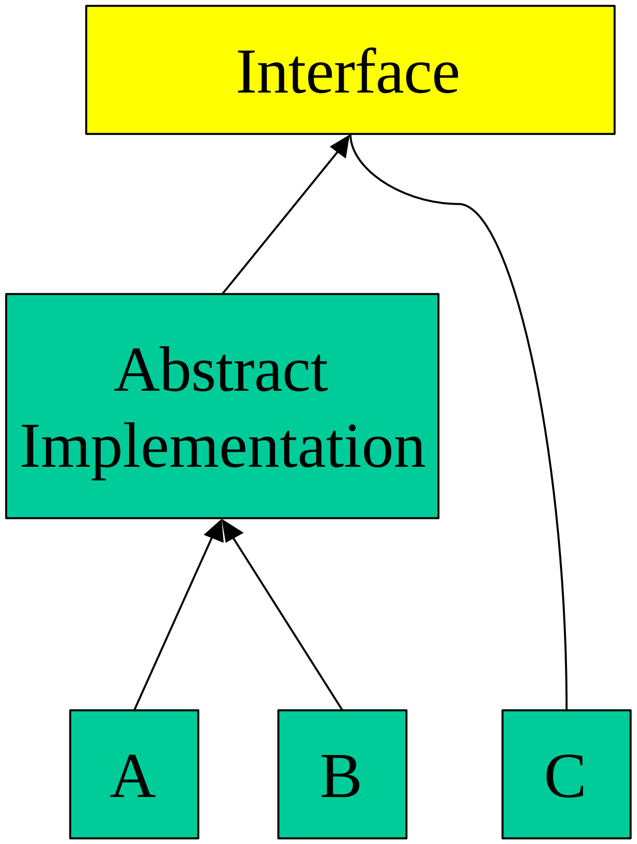
- Clean separation of specification and implementation
- Clean separation of components
  - Interaction only via interfaces ensures components can be dynamically loaded, upgraded, re-implemented, replaced, without effecting usage.
- Runtime selection of appropriate components
  - Especially useful when combined with (fine-grained) dynamic loading.

When to Use Interfaces?

- As a design tool
  - For yourself
    - One class, or set of classes, normally performs multiple roles. Defining separate interfaces for each role clarifies which methods perform which role, avoids confusion.
  - For a collaborative project
    - Separate discussion of goals from implementation
    - Clarify use of terminology
    - Large fraction of effort should go into design of interfaces
      - As opposed to implementation
    - Break implementation into independent components
- As a business model
  - “Collaborate on design, compete on implementation”

Why Abstract?

- Otherwise couples implementation with interface?
  - Can have cake and eat it too ->
  - Why not? There is little overhead.
- Does Abstract --> Factories?
  - Not necessarily
  - I3Vector i3 = new Hep3Vector();
- Downside of Interfaces
  - Adding/Changing methods breaks implementations
    - Abstract implementation mitigates this

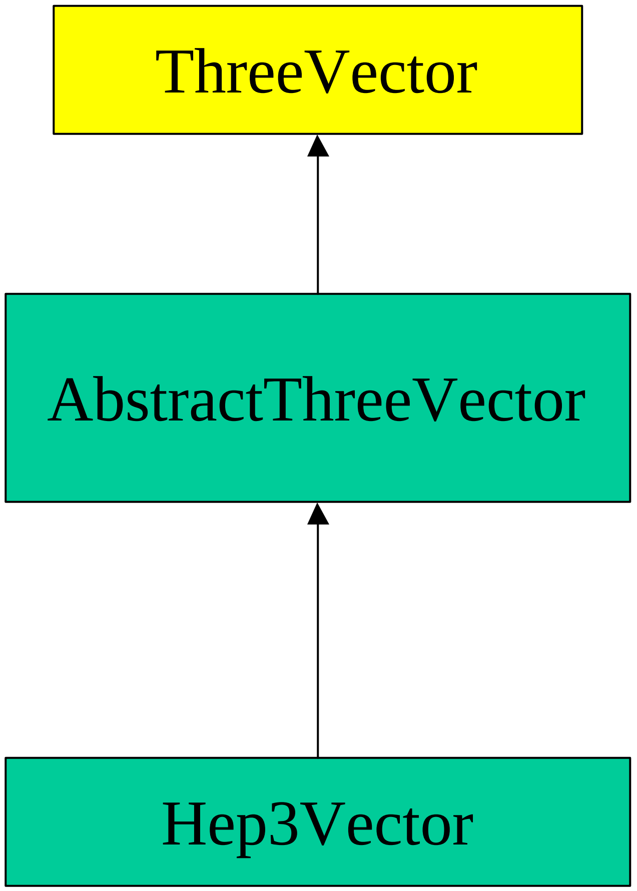


Examples of Interfaces in HEP

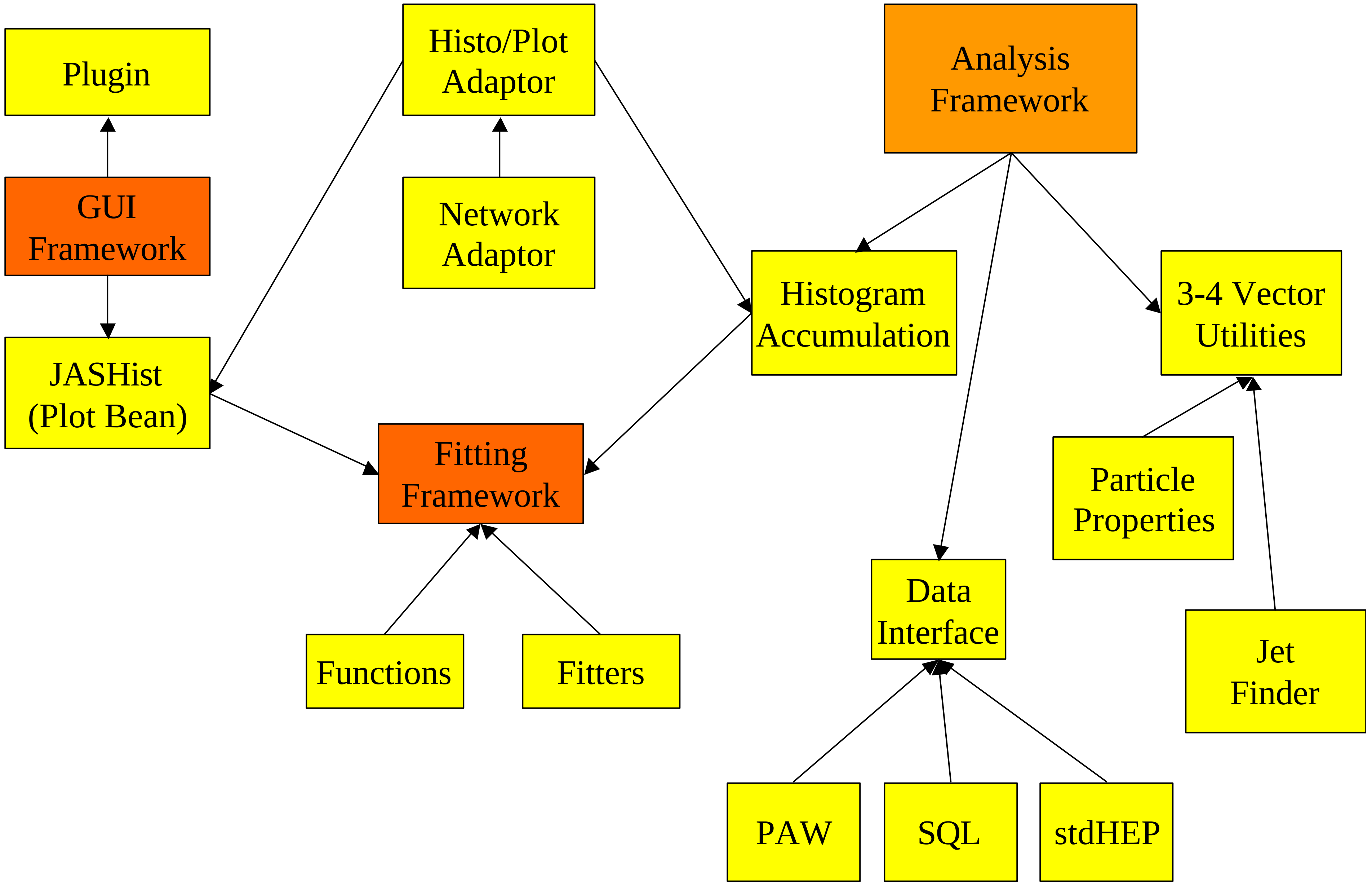
- FreeHEP 3-Vectors/4-Vectors
- JAS Histograming/Plotter
- Data Access (LCD Event Store)
- AIDA
- HepRep

FreeHEP 3-4 Vectors

- Everything that uses 3-4 vectors (jet finders, event shape, utilities) uses ThreeVector
  - Will work with any implementation of ThreeVector
  - The *only* place HepThreeVector is used is with **new**, everywhere else ThreeVector should be used.
- Note, no I in interfaces, the ThreeVector **is the 3 vector**, HepThreeVector is merely an implementation.

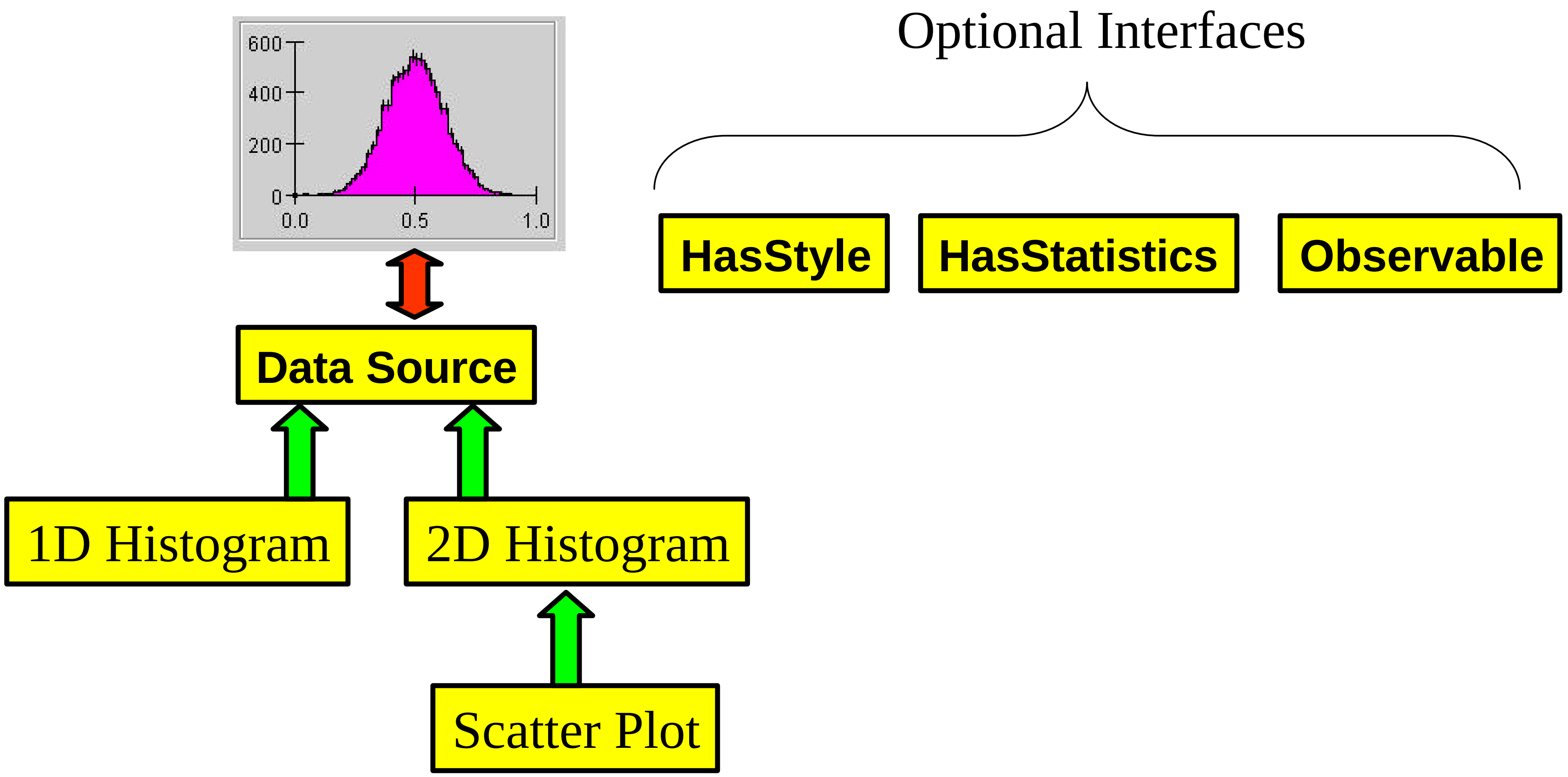


JAS 2 modules



JAS Plotting/Histograming

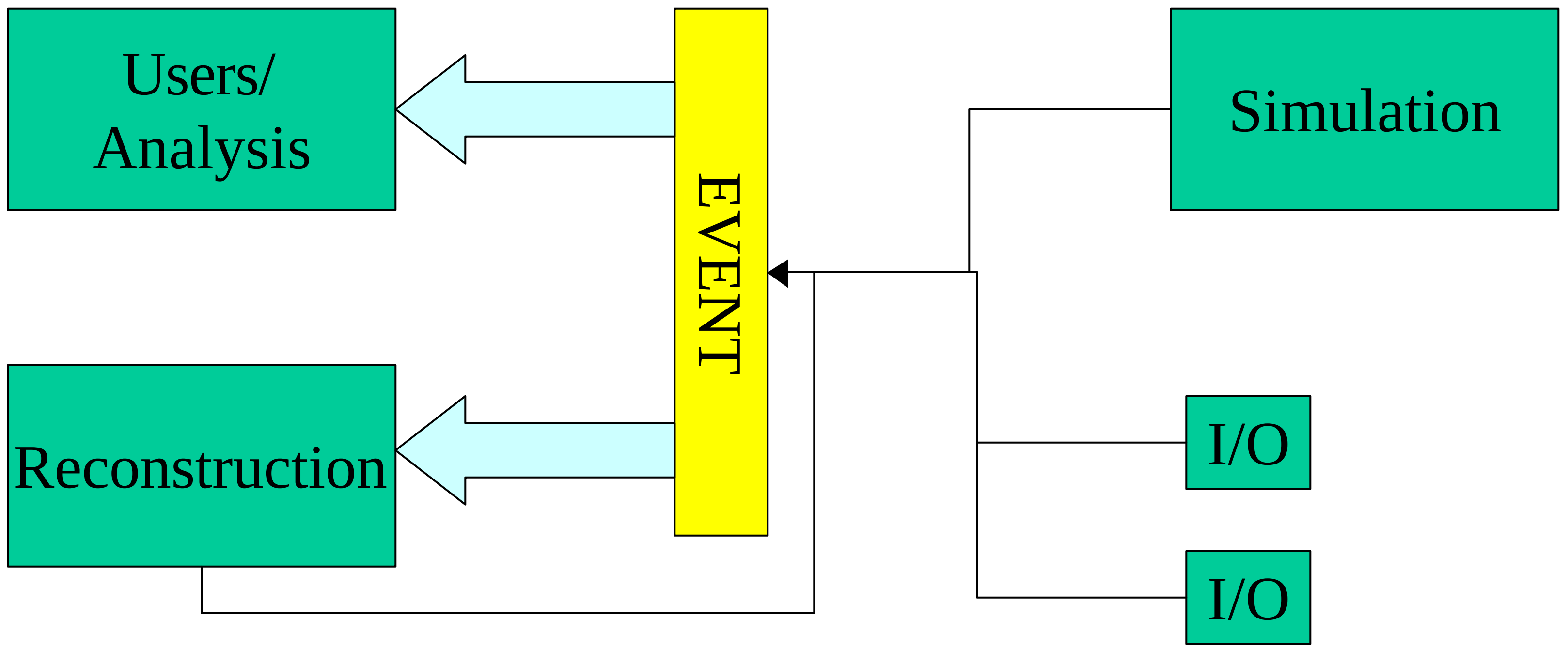
- JAS makes extensive use of abstract interfaces.
- For example
  - No direct coupling between plotter and histogram package.
    - Easy to use histogram package in non-graphical environment
    - Easy to use plotter in many areas unrelated to histograming
    - Easy to switch from existing histograms to AIDA
  - User experience is of tight coupling
    - Plot monitors histogram and updates in real time
    - Plot can be used to rebin and renormalize histograms



JAS Plotter has been used in many applications beyond JAS, from online monitoring, to web servlets, to financial applets.

LCD Event Interfaces

- All access to event data via Event Interfaces
  - Everything from raw data to reconstructed particles, vertices and jets are described by (read-only) abstract interfaces.



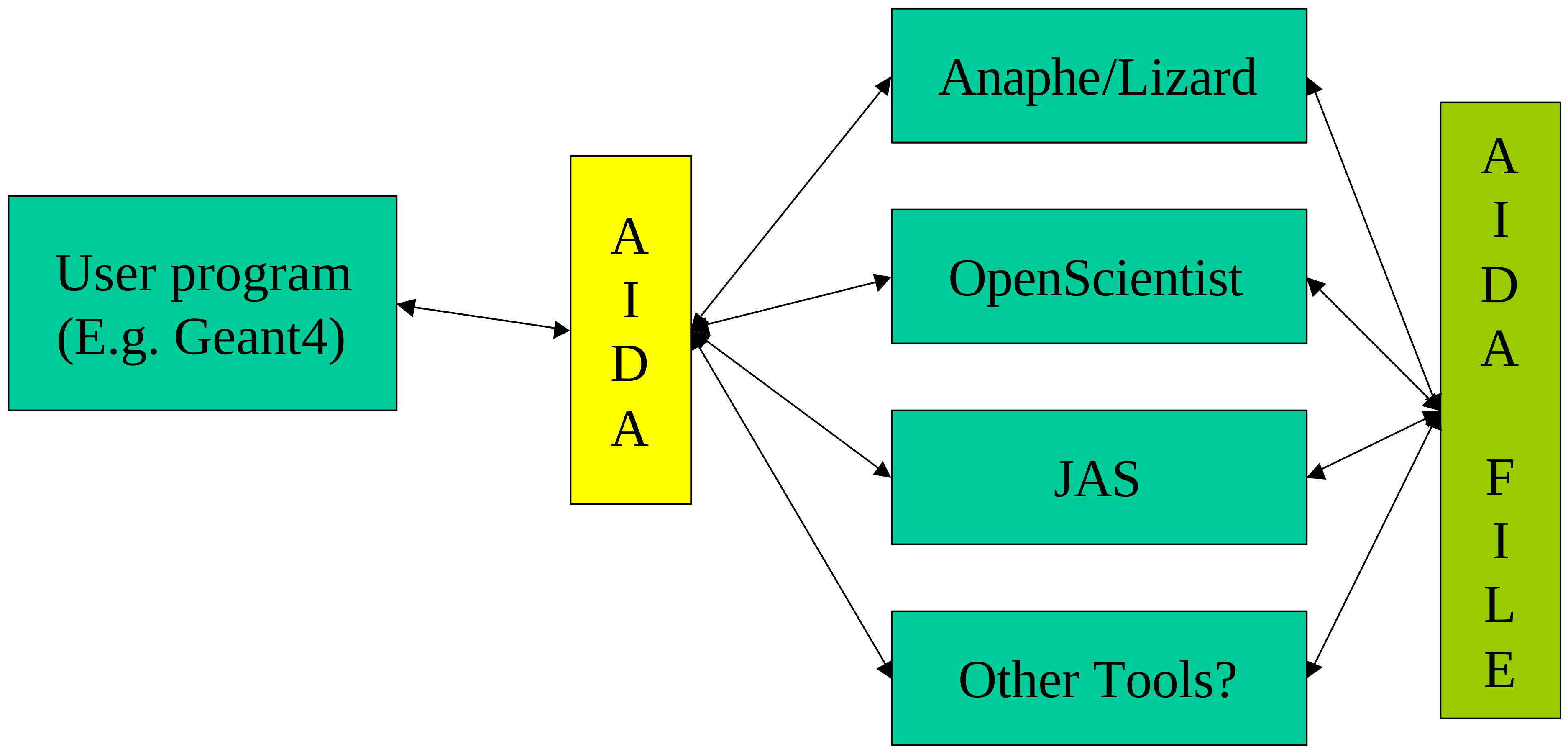
LCD Event Store

- Advantages of Abstract Interfaces
  - No need for “converters” or copying of data
  - If IO objects already implement event interfaces nothing to do, otherwise lightweight “adaptors” are all that are needed.
  - Leaves plenty of flexibility to implementation
    - For example read-on-demand, or reconstruct-on-demand
    - no need for user to say in advance what he intends to read.
  - **Warning**: Transient/Persistent mappings are often developed when reconstruction/simulation is main goal. They do not necessary provide the speed or modularity needed for interactive data analysis (c.f. Babar)

Abstract Interfaces for Data Analysis AIDA

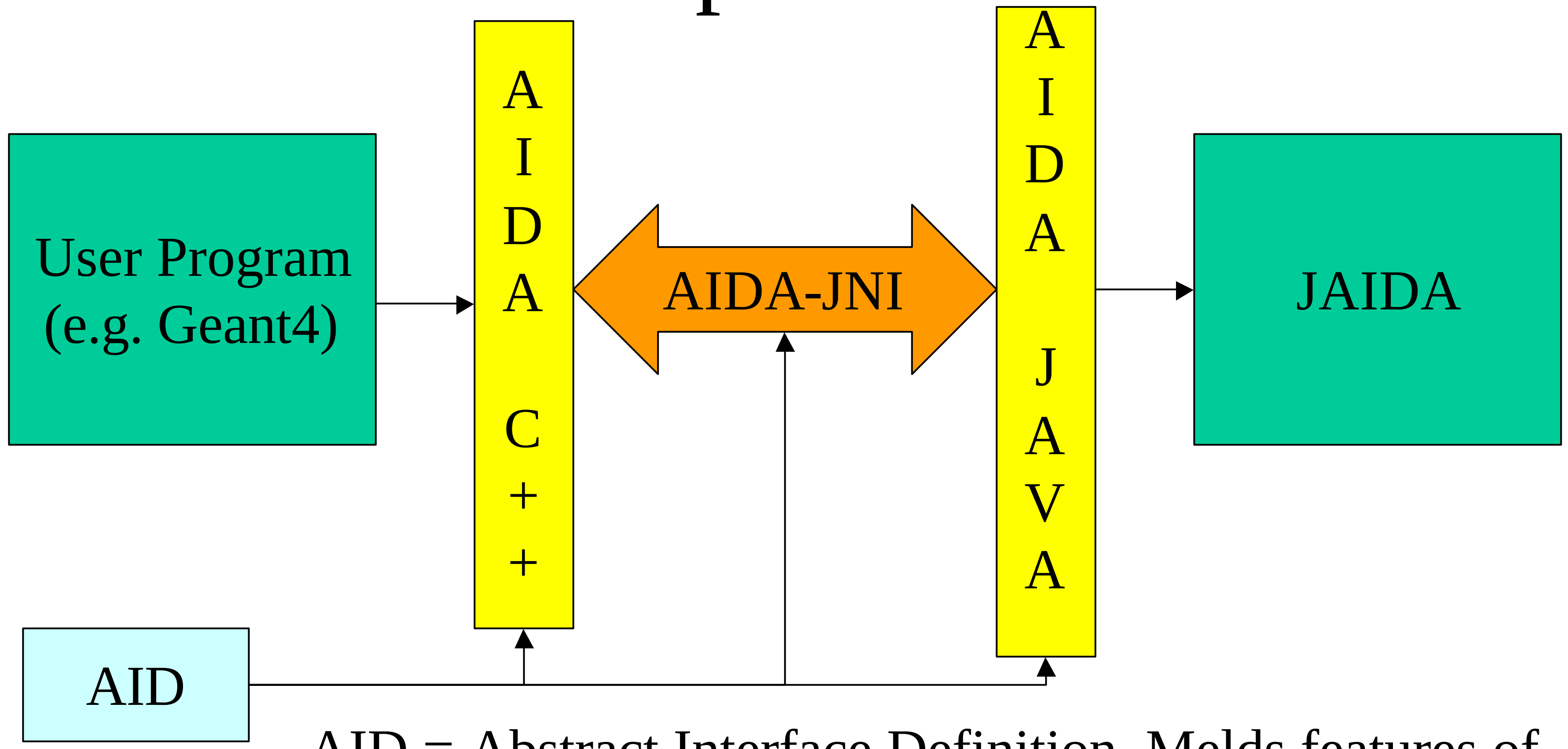
- Goals
  - Provide set of abstract interfaces for data analysis
    - Histograms, Clouds, Tuples, Fitting, Plotting, Trees, Storage.
  - Allow for multiple implementations
    - OpenScientist, Anaphe/Lizard, JAS
  - Allow for flexibility in implementations
    - Interpreter, GUI, 3D GUI, batch/interactive.
  - Language Independence
    - C++, Java, (Python, …)
  - Meta Goals (Abstract Interfaces as Collaborative Tool)
    - Encourage teams with different existing tools to collaborate.
    - Initially develop common terminology, make interfaces represent best feature of all existing tools.
    - Interoperability via common file format.
    - Interoperability via modular components.

AIDA



- AIDA 2.2 since December 2001
- AIDA 3.0 to be released September 30.

AIDA
Abstract Interfaces for Language Independence

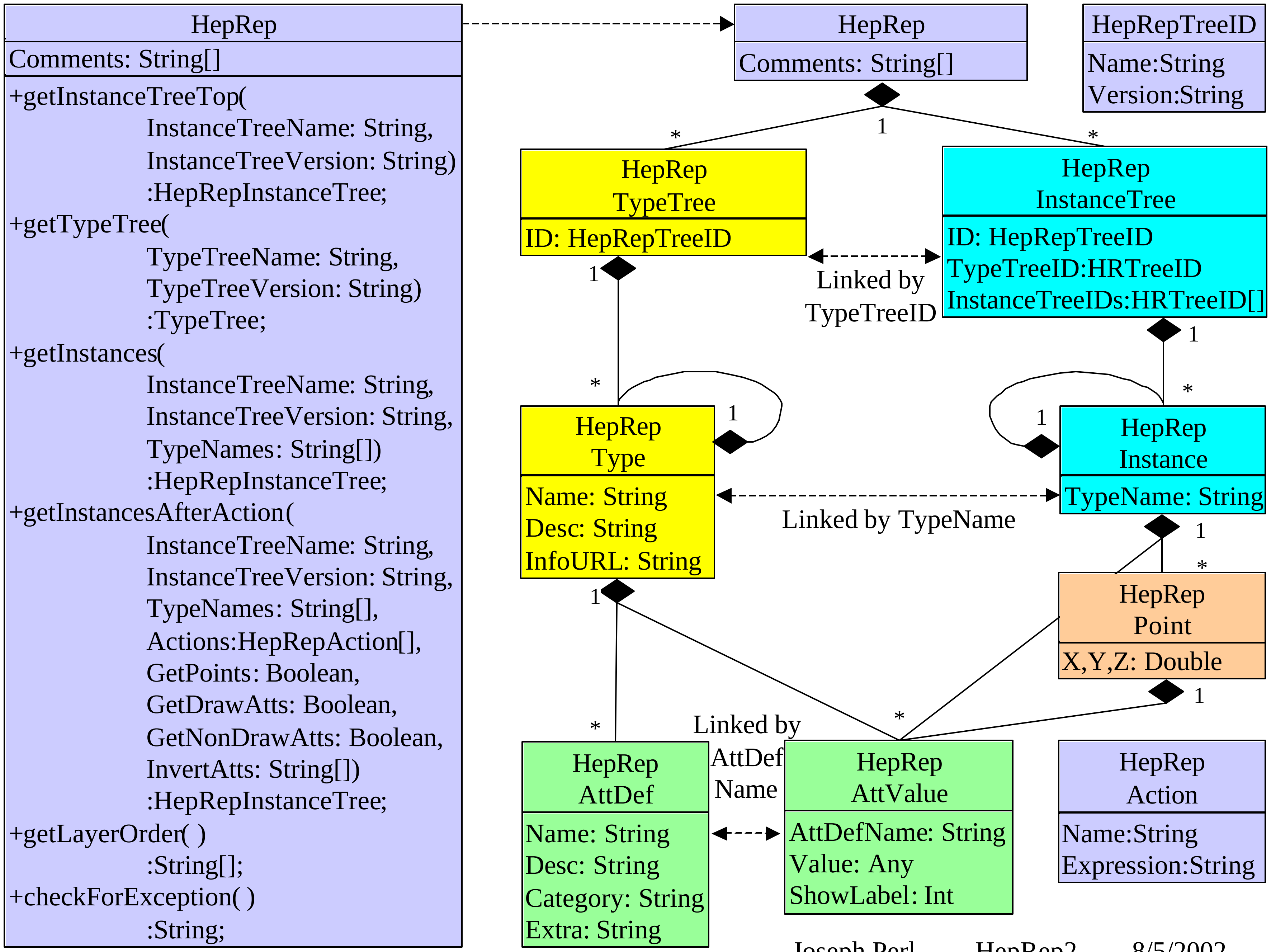


AID = Abstract Interface Definition. Melds features of C++, Java, in natural way. Generates C++ and Java (and Python in future). C.f. Swig, Boost, Jace *etc.*

HepRep

- Abstract interface used to decouple experiment framework/data from Event Display.
- WIRED event display (HepRep client) used with
  - BaBar, LCD, Glast
- Possible to introduce new Event Display clients in future with no change to experiment code.
- Language independent
  - Interfaces easily mapped to CORBA, RMI, XML, Web Services, JNI etc.

HepRep: a Generic Interface Definition for HEP Event Display Representables



Other Architectural Issues

- Software Bus
- Python *vs*. Cint
- Abstract Interfaces for LCG?

Software Bus

- High “Pointy-Haired-Quotient” in various references and diagrams referring to “Software Bus”
- “scripting (Python) interpreter tool as software bus”

I THINK WE SHOULD BUILD A SOFTWARE BUS.
UH-OH
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
WHAT COLOR DO YOU WANT THAT BUS?
I THINK MAUVE HAS THE MOST RAM.
E-mail: SCOTTADAMS@AOL.COM
© 1995 United Feature Syndicate, Inc.(NYC)
S. Adams

- Define “software Bus” and where/how it will be used.
- Publish/subscribe, messages, services
- Don’t treat it as a panacea
- Don’t confuse scripting language and software bus.

Scripting

- Why tie framework to single (scripting) language?
  - (Remember we want modular software)
- If you have OO object model + data dictionary + software bus
  - Should be drivable from any OO scripting language (c.f. Java)
    - Cint, Python, NetRexx, Java, JavaScript etc.
  - Maybe favour one or more,
    - Should be no reason to exclude others
    - Should be no need to tightly couple to any
    - Should support GUI on equal level as scripting language

Where to Use (Abstract) Interfaces

- Data Dictionary
  - Should be possible to tie to Root (Cint) or any other data source
- Data catalog
- Detector Geometry
- Analysis Tools (AIDA)
  - Root should be included
- Scripting
- Services on Software Bus
- GRID services

Conclusions

- Short-term benefits of good design not always obvious
  - To users
  - Or sometimes even developers
- LHC will have a very long life span and will have a profound impact on HEP software well beyond CERN
  - Be serious about modular software to ensure that HEP software can evolve over LHC lifetime.