The use of HepRep in GLAST

Joseph Perl
SLAC Computing Services
perl@slac.stanford.edu

Riccardo Giannitrapani
Dipartimento di Fisica, Udine (ITALY)
riccardo@fisica.uniud.it

Marco Frailis
Dipartimento di Fisica, Udine (ITALY)
frailis@fisica.uniud.it

http://www-glast.slac.stanford.edu/software

Contents

- The GLAST mission and instrument
- GLAST event display requirements
- Intro to HepRep
- The GLAST way to HepRep
- Conclusions and outlook

We wish to thank all of the GLAST software group for helpful discussions on event display, graphics, etc.

GLAST Mission

Gamma-ray Large Area Space Telescope measures the direction, energy and arrival time of celestial gamma rays

-**Large Area Telescope (LAT)** measures gamma-rays in the energy range ~20 MeV - >300 GeV

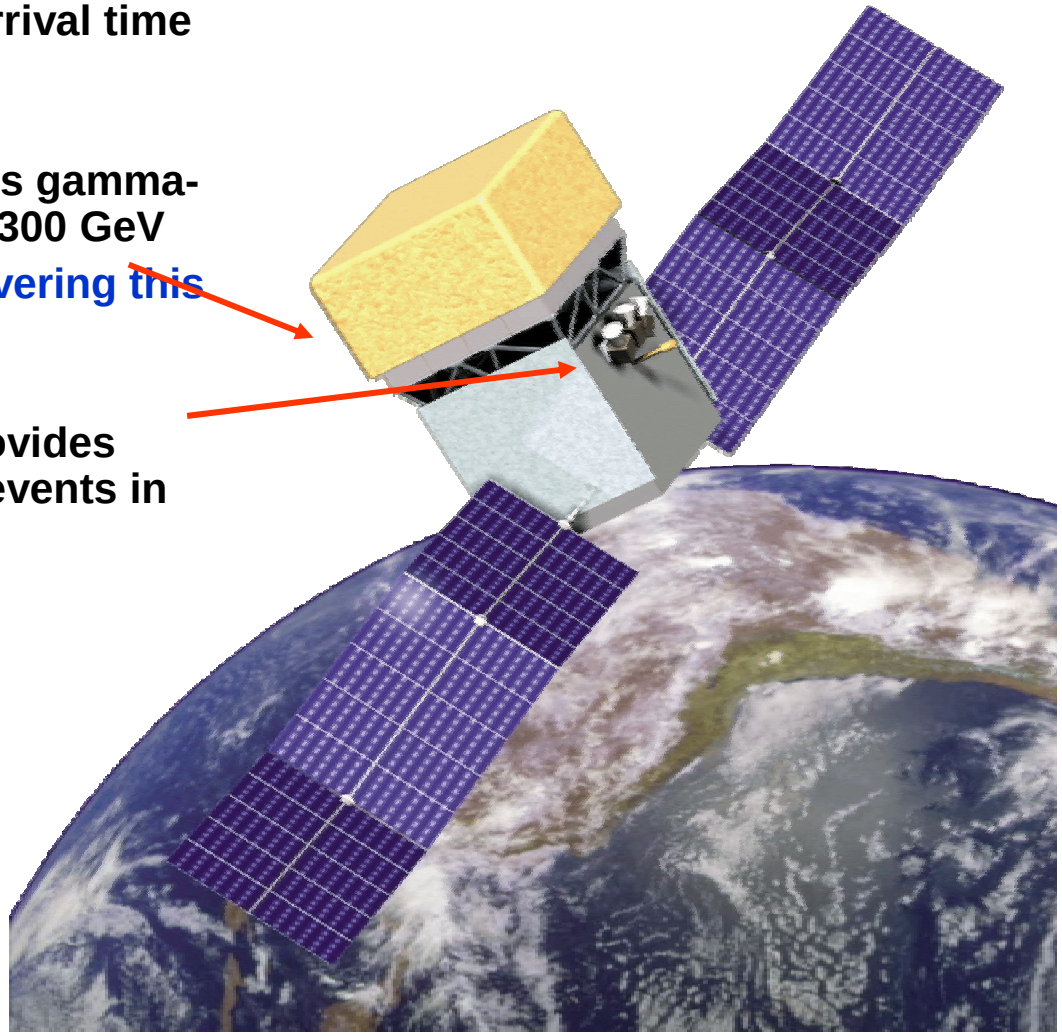
- There is no telescope now covering this range!!

- **Gamma-ray Burst Monitor (GBM)** provides correlative observations of transient events in the energy range ~20 keV – 20 MeV

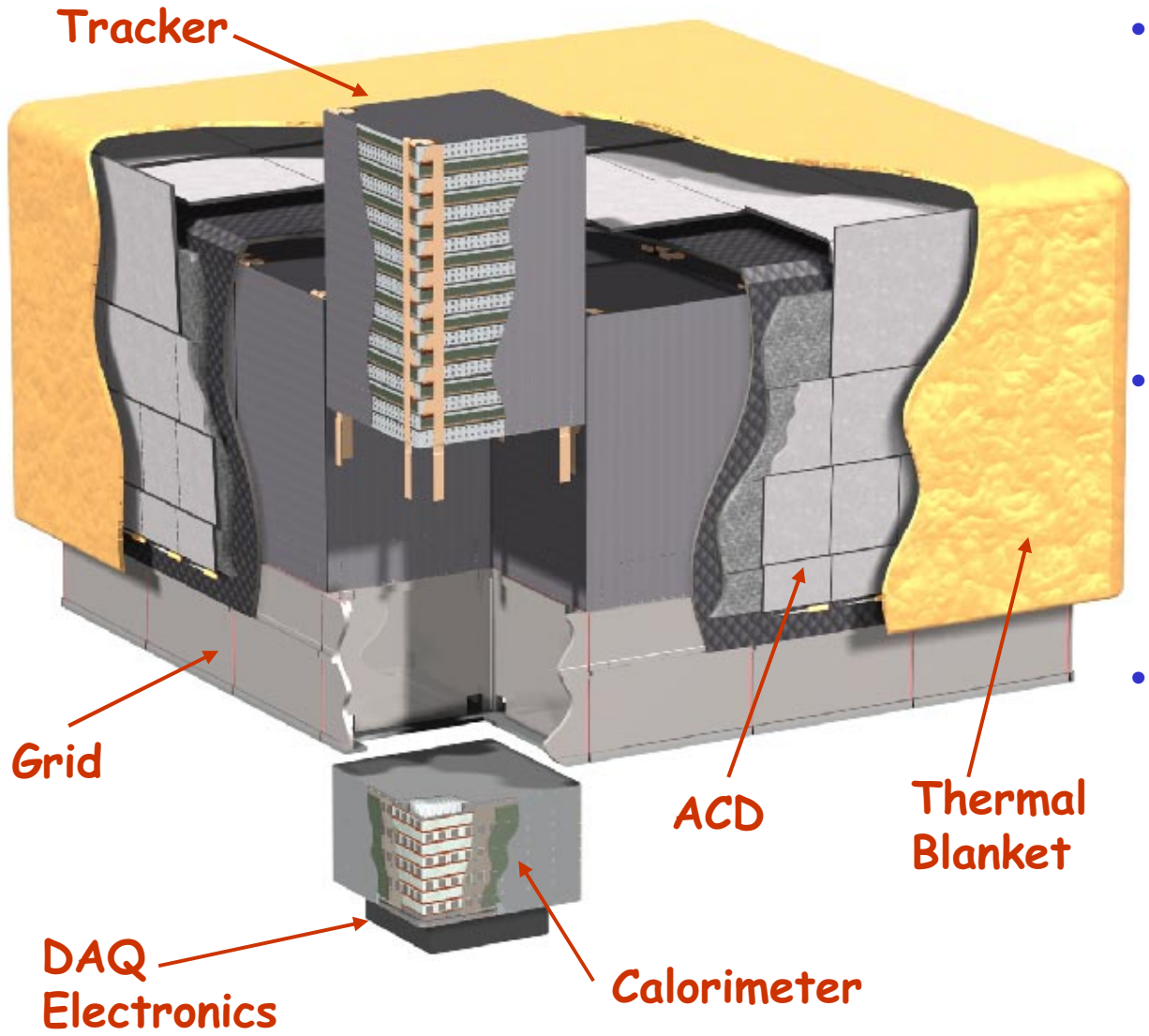
Launch: September 2006 Florida

Orbit: 550 km, 28.5^{o} inclination

Lifetime: 5 years (minimum)



GLAST Instrument: Large Area Telescope (LAT)

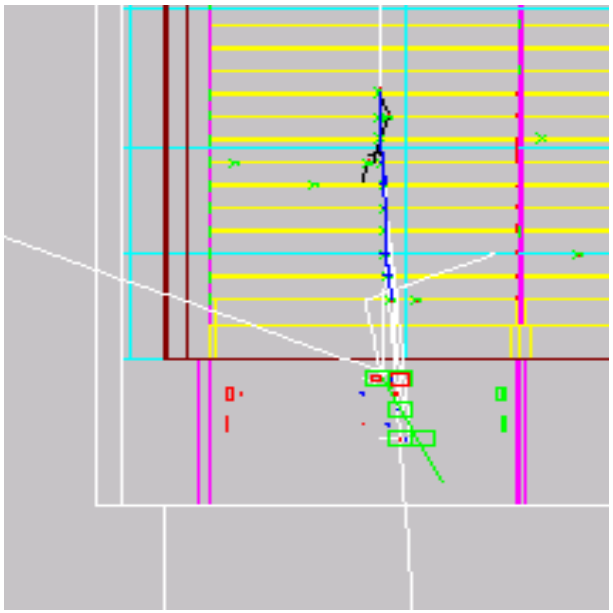


- Array of 16 identical "Tower" Modules, each with a tracker (Si strips) and a calorimeter (CsI with PIN diode readout) and DAQ module.
- Surrounded by finely segmented Anti-Coincidence Detector (plastic scintillator with PMT readout).
- 3 GB data per day
  - 30GB onboard storage
  - download several times per day
  - data makes its way to SLAC

An Event Display for GLAST

- **GLAST already had an integrated Event Display for its offline software:**
  - **Windows and Linux**
  - **simple, fast wireframe**
  - **3D and 2D**
  - **tightly integrated into GAUDI**
  - **drives GLAST main offline event loop (source generation, Monte Carlo, reco, etc.)**

  However:
  - **very limited mouse interaction**
  - **no ability to pick on objects**
  - **no persistency**
- **For the longer term, GLAST wanted something more flexible, extensible and interactive**



Some Key Requirements

- Multiplatform (at least Windows and Linux)
- Easy to install and start
- Fast
- 3D and 2D
- Modern GUI
- Easy navigation and browsing of the event (with incremental download)
- Pick on objects to inquire about them
- Pick on objects to interact with physics algorithms in our GAUDI framework
- Ability to drive the GLAST main offline event loop (source generation, Monte Carlo, reco, etc.)
- Extensible/configurable by the user
  - Requirements dictated by physics should be easily added directly by the physics experts, should NOT require graphics experts
  - New features related to what is represented (for example changing the trajectory color to code for charge, or energy, or any other attribute) should not require significant re-coding

Software Life-Cycle Issues

- Design with the assumption that the life-cycle of the event display may be different from the life-cycle of the infrastructure software.
- Display should not be too tightly coupled with our actual choices of:
  - framework
  - physics algorithms
  - event structure
  - persistency mechanisms
  - Monte Carlo
  - etc.
- We are looking for an event display paradigm rather than a specific event display application.

The Client-Server Paradigm

- **Server** deals with physics, interaction with reconstruction algorithms, with our data store files etc etc
- **Client** deals only with graphics representations (that may be augmented with additional information that has meaning for the experiment)
- **Client-server does not necessarily imply remote operation**. Client and server may be on the same machine or may be on different machines. The client-server separation is in any case a useful construct to cleanly delineate the two parts of the event display solution
- **Client and server communicate with an interface**; the crucial point is that this interface should be simple, extensible and should accommodate all the needs seen before
- **HepRep** is such an interface:
  "A Generic Interface for Component or Client-Server Event Displays" provides for the correct distribution of computing work between the two parts of the system and effectively addresses the many important maintenance issues involved in such a system

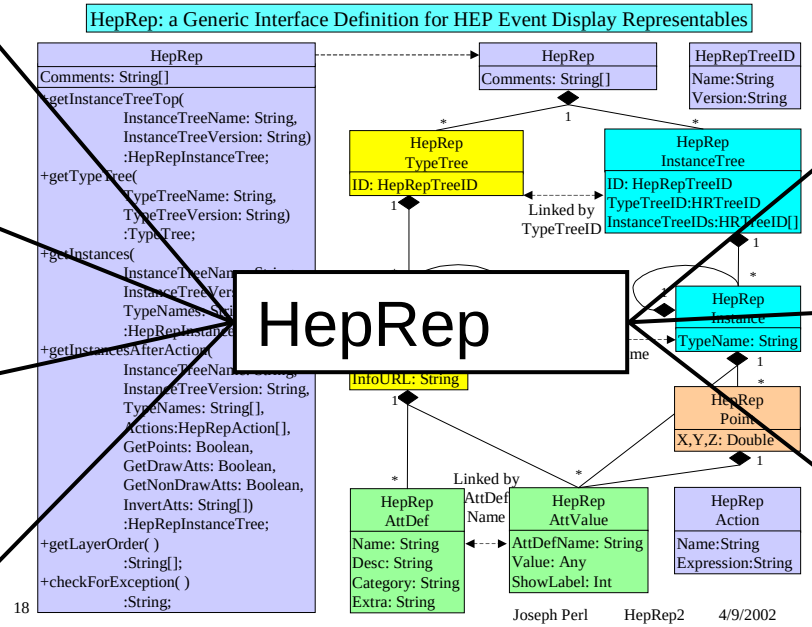
HepRep Purpose

BaBar Server

GLAST Server

LCD Interface

Geant4 Server



HepRep

WIRED Client (Java)

FRED Client (C++/Ruby)

Other HepRep Clients

The HepRep interface breaks the dependency between any particular experiment's event display server and any particular event display client.

The HepRep format is independent of any one particular language or protocol. It can be used from C++ or Java and can be shipped as Corba, RMI, XML, C++, Java or JNI for consumption by WIRED, FRED or any other HepRep-enabled event display client.

The “Rep” in HepRep means Representables

- If one just ships references to the underlying physics objects, there are too many time-consuming callbacks, asking one by one for the points on the tracks, etc. One doesn’t achieve good separation of client-server functionality.
- The design decision behind HepRep is to serve Representables, not Physics Objects.
  - A Representable is the Essential Spatial Information of a Physics Object (track, calorimeter hit, etc.) and can be augmented by that object’s Physics Attributes (momentum, energy, etc.).
  - Serving Representables keeps the detailed reconstruction code, swimmers and detector models on the server side where they belong. Spatial information is assembled and shipped in an efficient manner, avoiding the overhead of too many individual method calls.
  - Rendering decisions are deferred, as much as possible, to the client.

Example HepRep Representable

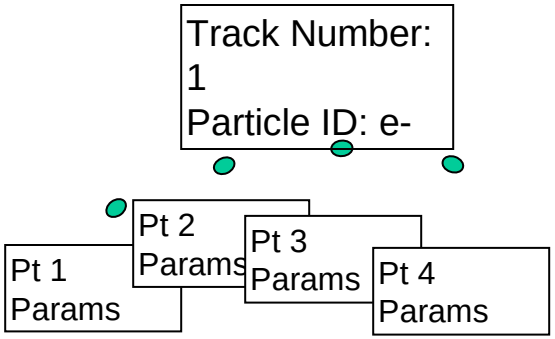
A precise fitted track could be served as a set of swim step points, each augmented by helix parameters and descriptive information (track number, particle id, etc.). Only in the client is the final decision made whether to Represent this Representable as

- **a dotted line,**
- **or as set of individual swim step momentum vectors,**
- **or as a set of helix segments.**

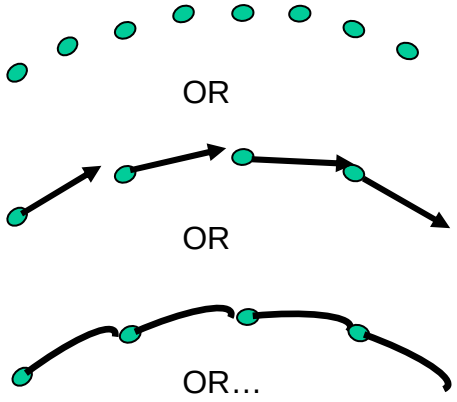
Physics Object

Fitted Track
Track Number Particle ID Points(n) Helix Params(n)

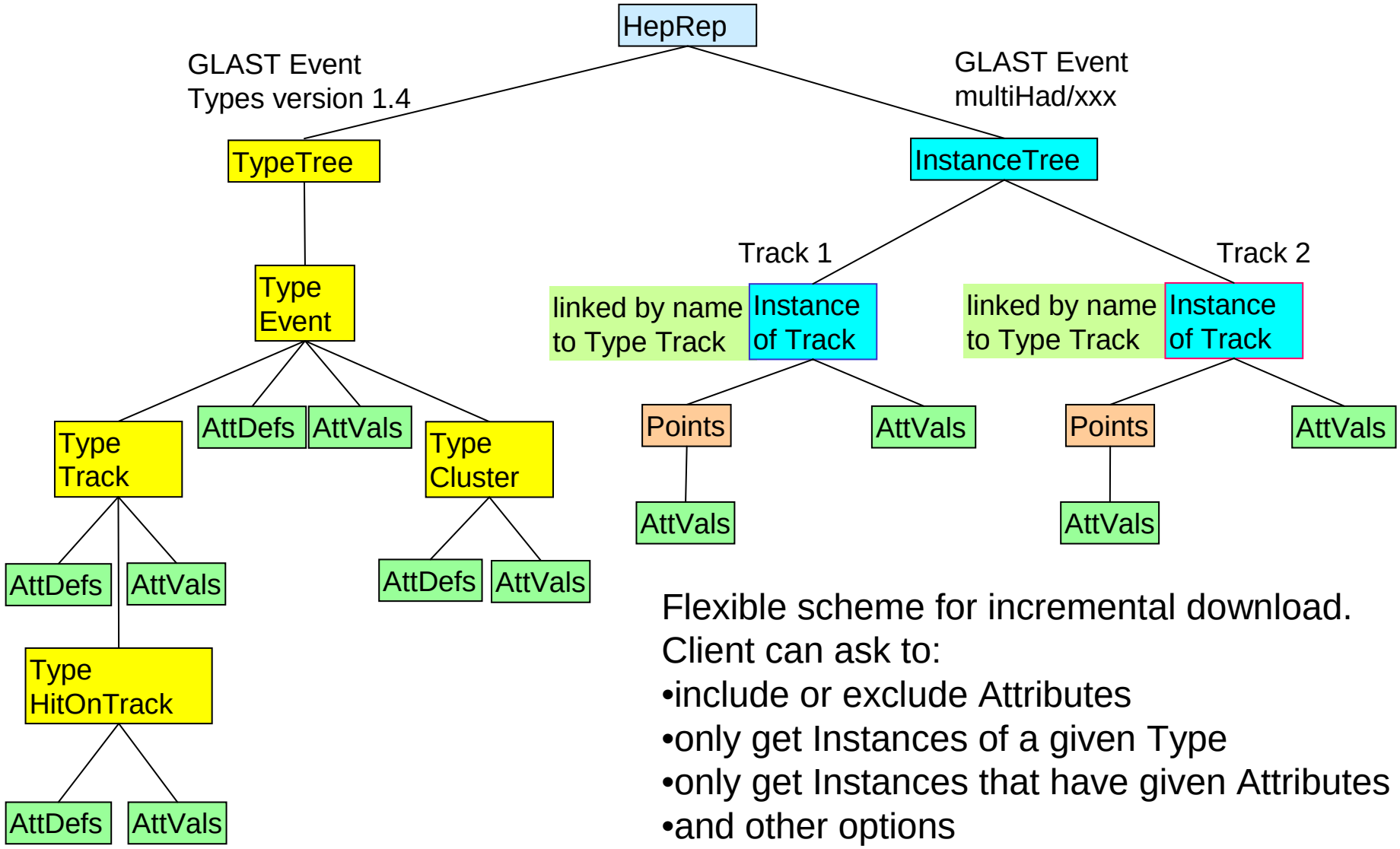
Representable



Representation



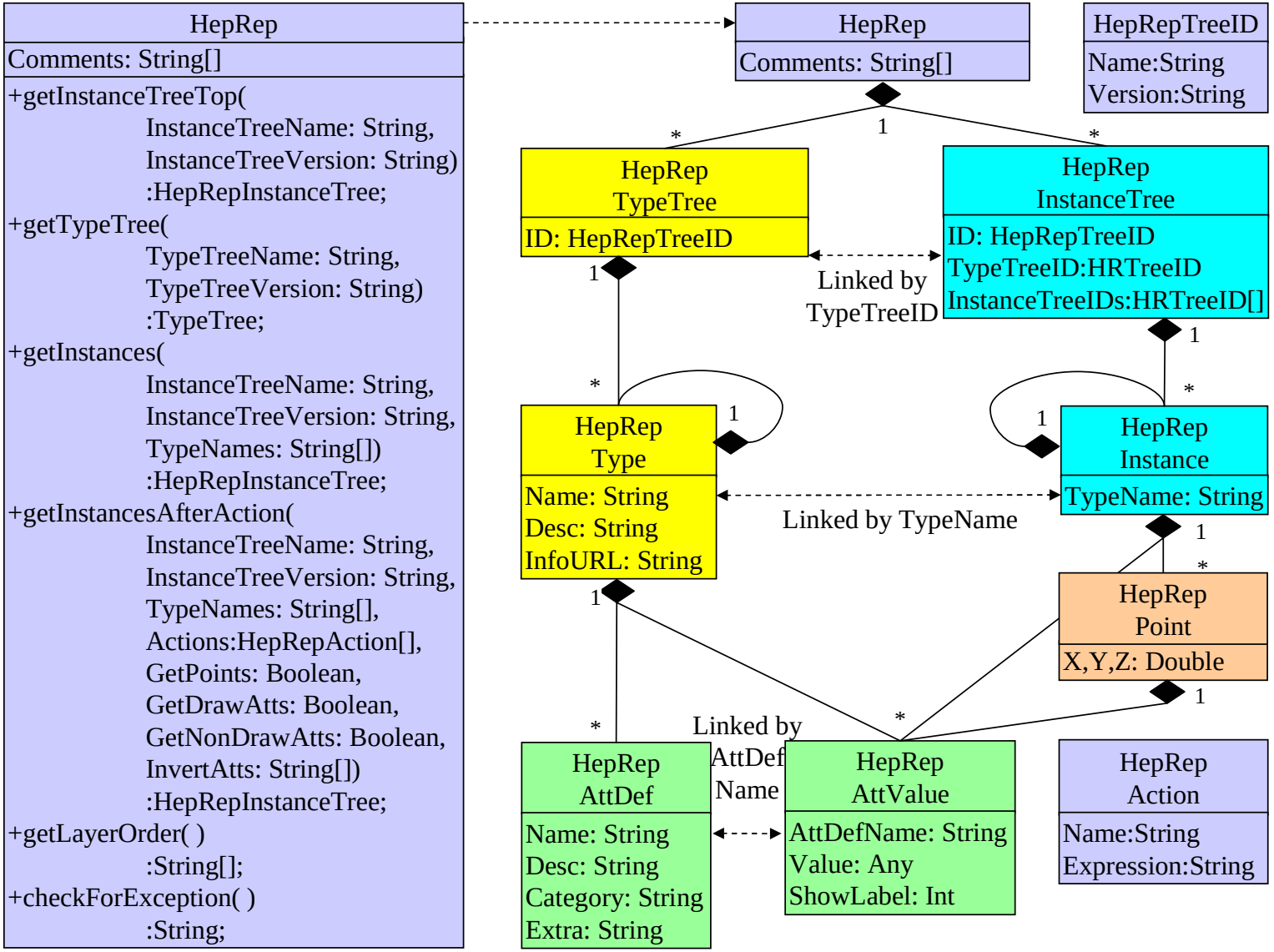
Example HepRep Object Tree



Flexible scheme for incremental download.
Client can ask to:

- include or exclude Attributes
- only get Instances of a given Type
- only get Instances that have given Attributes
- and other options

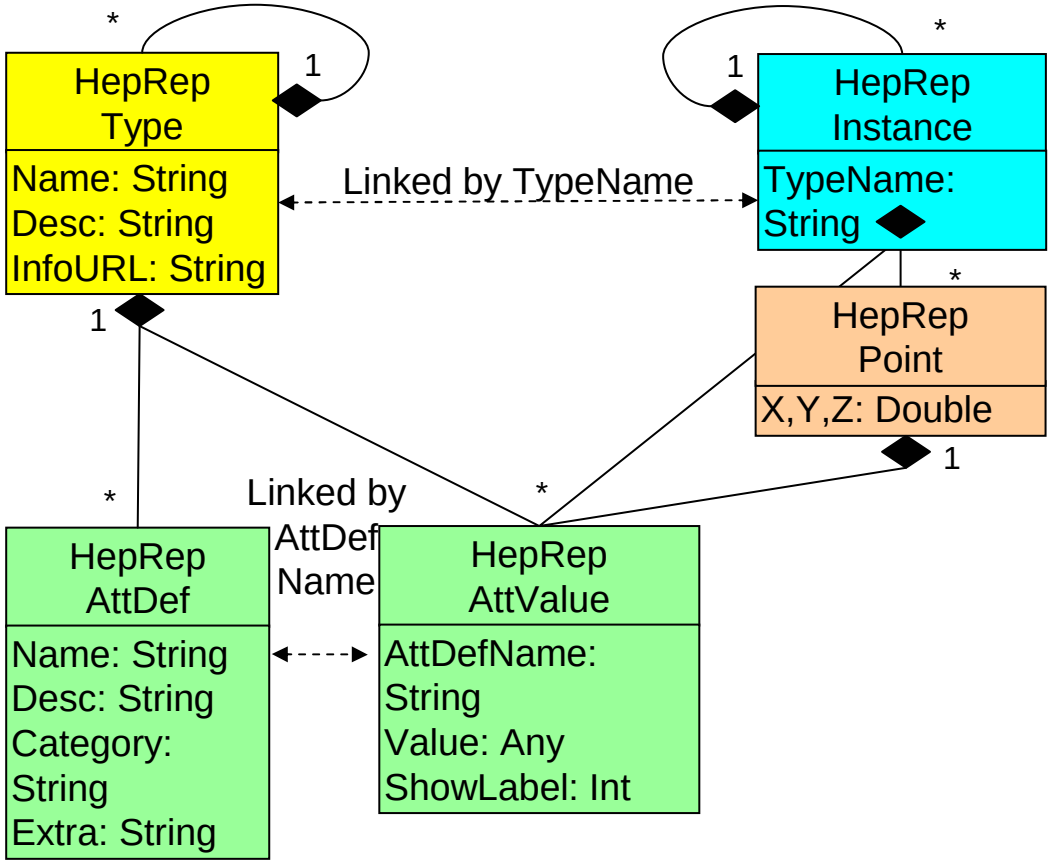
The HepRep Interface



HepRep Attributes

Any number of Attributes can be hung from a Type, Instance or Point.
There are four Categories of Attributes:

- **Draw Attributes** (such as thickness, color and what shape to draw from the points) can be modified in the client through a draw attribute editor
- **Physics Attributes** (such as track momentum or hit error) can be used for visibility cuts (client side or server side)
- **PickAction Attributes** define special things to do when the user picks on the Representable (such as remove hit and refit track)
- **Association Attributes** define loose associations between Representables (such as track cluster matching)



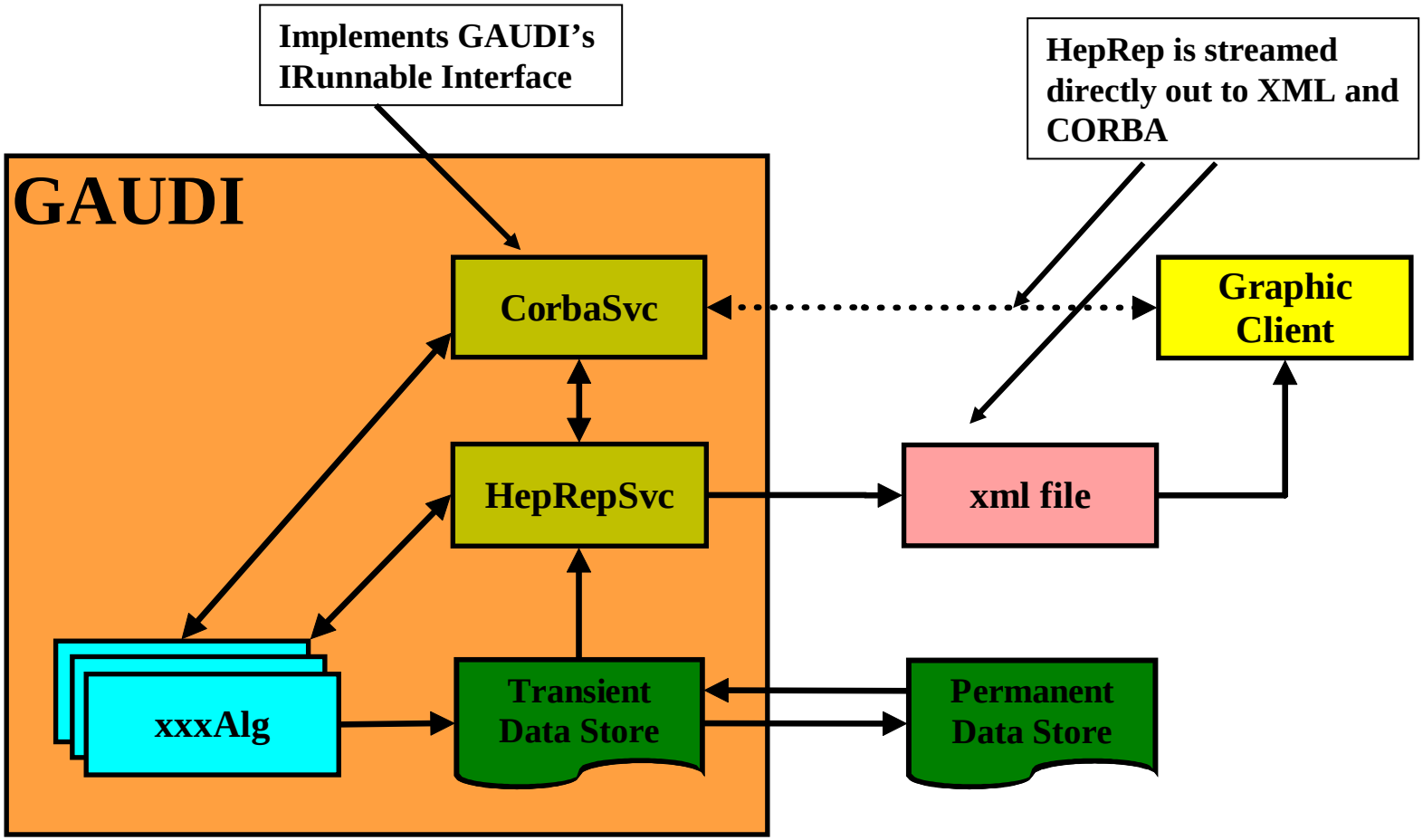
The GLAST way to HepRep

- GLAST uses GAUDI
  - An object oriented (C++) framework
  - Separation between data and the algorithms on that data
    - Data are stored in Transient Data Store (TDS) and/or Permanent Data Store (PDS)
    - Algorithms can act on the TDS, filling it or retrieving things from it
    - Services provides common functionalities on algorithms
  - Has its own event loop
  - Can be customized at runtime through initialization files (jobOptions files)
- We need to develop our client-server HepRep framework inside GAUDI
- We want to be able to drive the event loop from the external graphics client

HepRep within GAUDI

- We have implemented a HepRep based event display architecture to be usable within GAUDI
  - Server lives in the GAUDI world, having access to all Algorithms, Services and the TDS. Server has full knowledge of the physics contents of each event and we have full control on our tools
  - Client lives outside GAUDI. Client implements GUI and has access to HepRep attributes, but does not have direct access to tools within GAUDI.
  - HepRep interface brings information from the server to the client and commands from the client back to the server.
    - HepRep can be implemented in various way; we currently have implementations in XML (persistent) and CORBA (live)

HepRep GAUDI Architecture



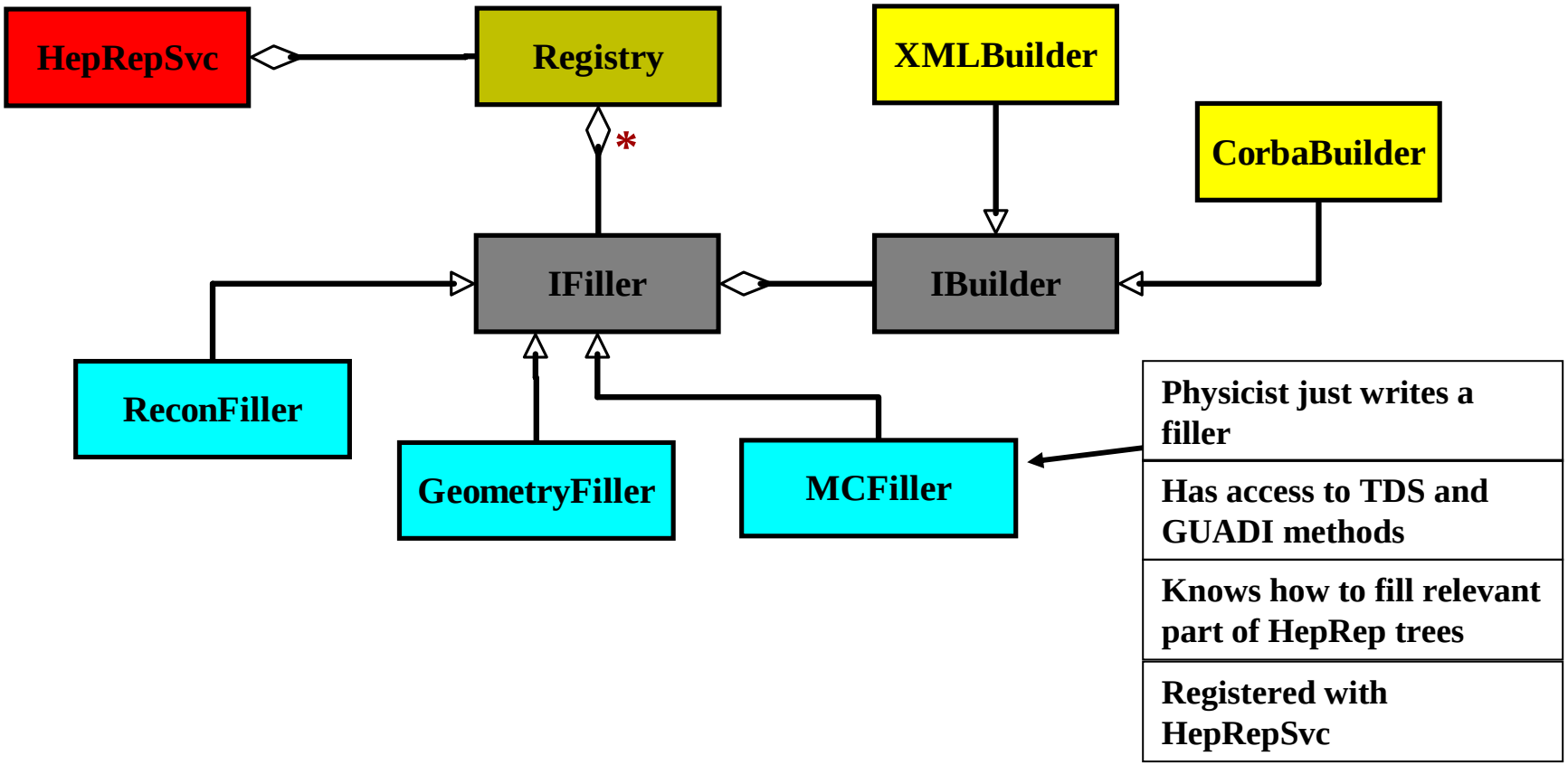
HepRepSvc and CorbaSvc

- **HepRepSvc produces a HepRep representation of the event from the transient data store at the end of each event; this representation can be "published" either as a persistent XML file (compressed) or as a CORBA object**
  - **To minimize unnecessary memory costs, we took care that the entire HepRep is never actually held in C++ memory but is instead streamed directly out to XML and CORBA**
- **CorbaSvc implements a GAUDI IRunnable interface and so can drive the event loop. It publishes a CORBA object that can be than used by the graphics client to retrieve the event.**
  - **Since the CorbaSvc lives inside GAUDI, it is possible (at least in principle) to call all the collaboration algorithms on the TDS data event. This means that this framework will allow a complete interaction of the graphics client with the physics software.**
  - **Since it is an IRunnable it can stop the event loop and wait for remote method invocation from the external client**

HepRep Fillers

- How is the HepRep built out of the Transient Data Store?
- We must make it easy for physics experts to add whatever they want into the HepRep hierarchy without requiring expert assistance or requiring them to know what other physics experts are adding.
- We provide a Filler mechanism:
  - The physics expert implements an abstract interface (IFiller)
  - In this implementation he has access to the TDS and to all relevant GAUDI methods and tools to retrieve information from it
  - In the filler he decides what to put in the HepRep tree, both in terms of types and of instances
  - Filler uses an abstract HepRep Builder (HepRep factory), so that the physicist does not need to be concerned with the eventual HepRep implementation (XML, CORBA or other)
  - Each filler is listed in a register (held by HepRepSvc). At each event, all needed fillers are called back to create the HepRep representation of the event

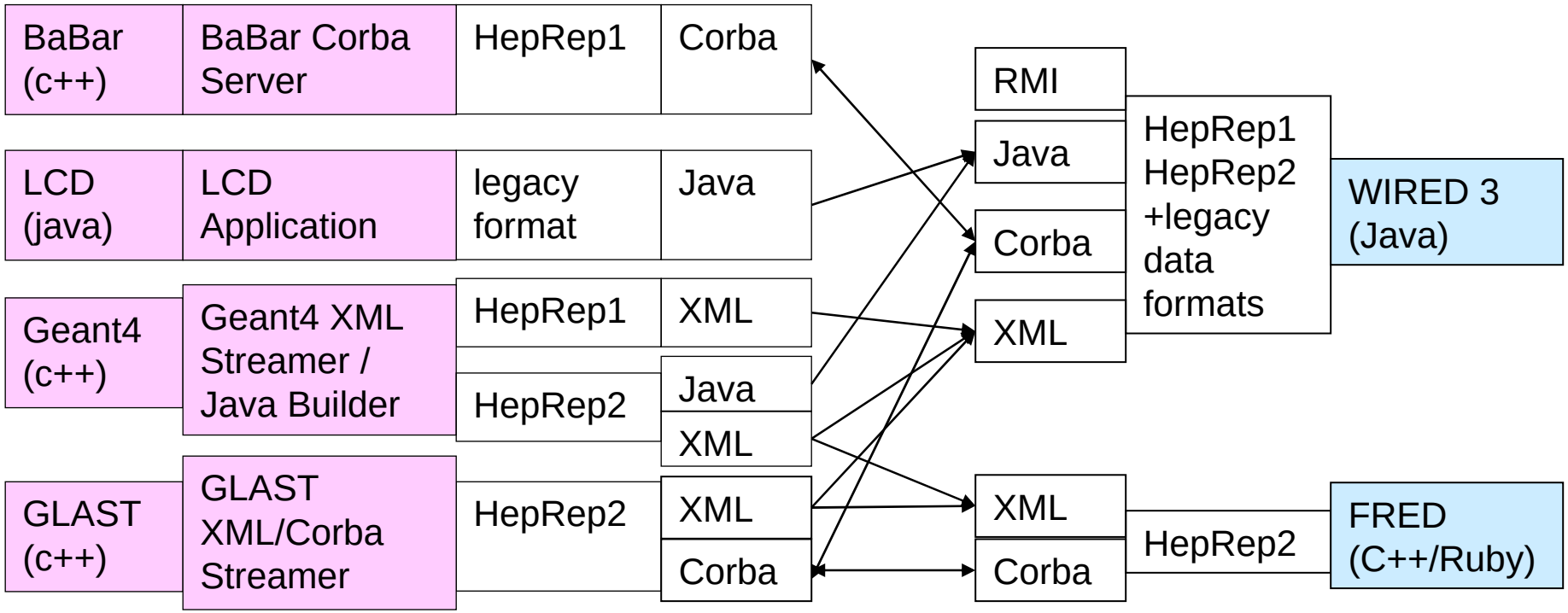
HepRep Filler and Builder Architecture



Value of the Filler and Builder Architecture

- Good separation between the physics experts and the graphics experts
  - Physicist just looks at a few example fillers, then makes a new one based on those.
  - Physicist does not need to learn anything else about the event display server or client (no need to learn CORBA, no need to learn XML, etc.)
  - Similar filler mechanism has been used by BaBar to good effect.
- Different physicists can work on different fillers
  - Good separation between GLAST subsystems, either physical (for example ACD from TKR) or conceptual (for example reconstruction from Monte Carlo)
- Good abstraction from the runtime implementation of the HepRep
  - One filler is used for all the possible HepRep formats (CORBA, XML, etc.)
- Very flexible
  - New fillers can be added at any time

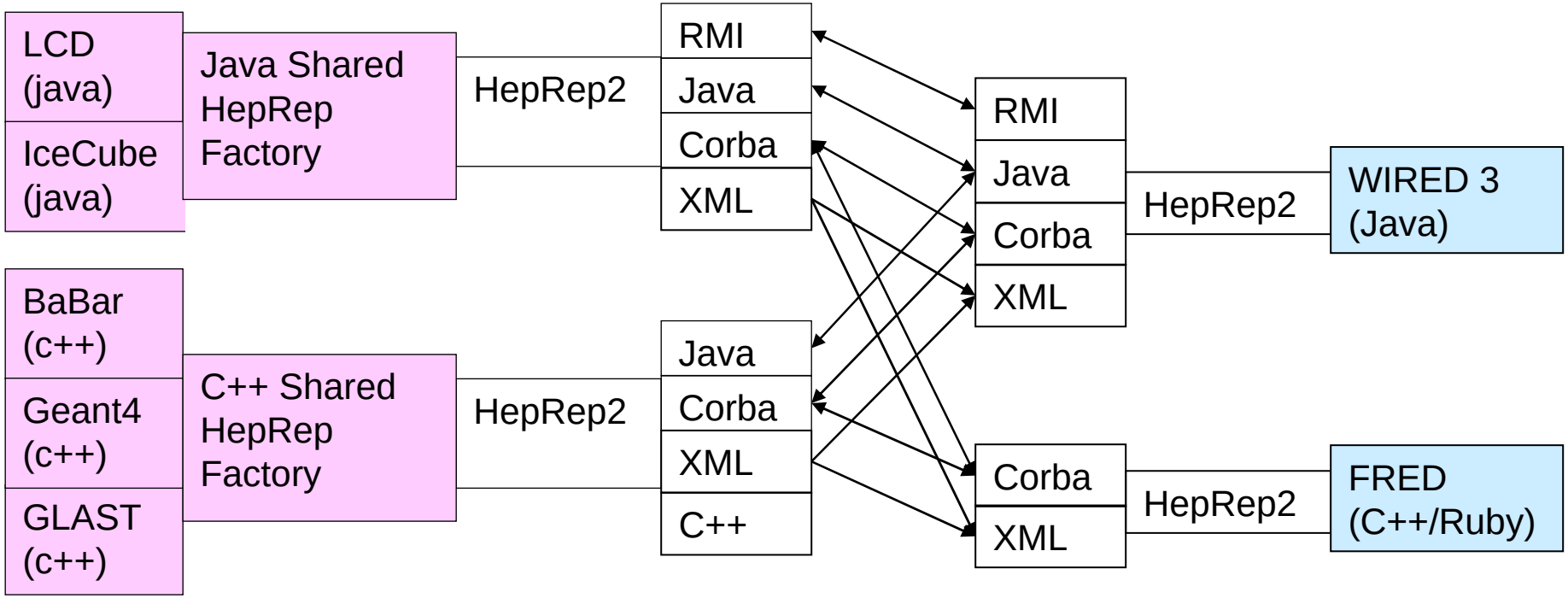
HepRep Current Use Architecture



While all four experiments are now using WIRED, and two can use FRED, they use a variety of HepRep and legacy implementations:

- BaBar has a HepRep1 Corba server, dependent on BaBar code.
- LCD passes WIRED java objects using a legacy data format (pre-HepRep).
- Geant4 has abstract HepRep1 and HepRep2 implementations to XML and Java.
- GLAST has an abstract HepRep2 implementation to XML and Corba.

HepRep Near-Term Future Architecture

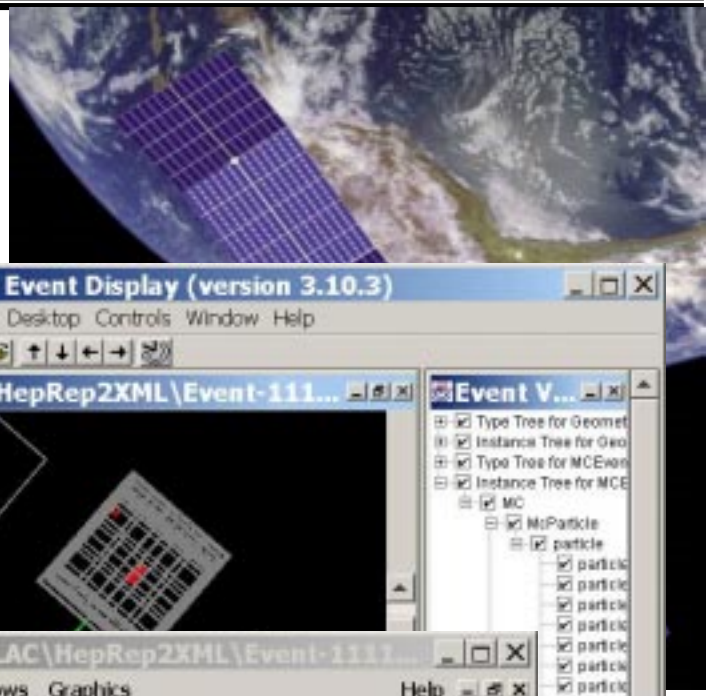


All data sources speak HepRep2 to an abstract HepRep factory (from FreeHEP). By instantiation of one or another concrete implementation of HepRep:

- a C++ program can change from creating HepRep in C++ memory
- to creating HepRep as an XML streamer (a pure C++ solution with no external library dependencies and no creation of the HepRep in memory)
- to creating HepRep as Corba streamer (depends on Corba libraries)
- or creating HepRep as Java (via Java Native Interface)

Conclusions and outlook

- GLAST wanted
  - A flexible, extensible and maintainable framework for event display
  - Without committing to any one graphics application
- GLAST now has
  - A HepRep based client-server framework integrated into our GAUDI application
  - A filler and builder mechanism to abstract event description from event representation
  - User has choice of client application:
    - WIRED (Java)
    - FRED (C++/Ruby)
- Outlook
  - GLAST can use any event display application that implements the HepRep interface



References

- HepRep: a generic interface definition for HEP event display representables
  http://heprep.freehep.org
- Fred: oh no, another event display (a HepRep client)
  http://www.fisica.uniud.it/~riccardo/research/fred
- WIRED: world wide web interactive remote event display (a HepRep Client)
  http://www.slac.stanford.edu/BFROOT/www/Computing/Graphics/Wired
- SLAC HepRep WIRED Work Plan
  http://www.slac.stanford.edu/~perl/wired
- A Component Approach to HEP Event Displays
  http://www.slac.stanford.edu/~perl/component
- Requirements for a New BaBar Event Display (most parts apply to any exp)
  http://www-sldnt.slac.stanford.edu/hepvis/paper/paper.asp?id=37
- WIRED for GLAST
  http://www.slac.stanford.edu/~perl/GLAST/talk_20010118.ppt
- The FreeHEP Java Library
  http://java.freehep.org
- GLAST Software
  http://www-glast.slac.stanford.edu/software