Future Plans for the LCD WIRED Event Display

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

• The larger context of WIRED
• Introduction to HepRep
• HepRep Current and Future Architecture
• Evolution from JAS 2 + WIRED 1.x to JAS 3 + WIRED 4
• Features to come in WIRED 4
• Demonstrations
  – WIRED HepRep from:
    • BaBar Offline and Online via Corba
    • GLAST via XML
    • Geant4 via XML
  – FRED HepRep from:
    • GLAST via XML
    • Geant4 via XML
Larger Context of WIRED

Joseph Perl
Limitations of Early WIRED Versions (from Mark’s talk this morning)

- One WIRED Plot per Page
- No Save and Restore
- No Picking Info
- No easy way to extend WIRED
- Memory Consumption

In response to these same issues in BaBar WIRED, we developed HepRep. This also gave us an entirely experiment independent form of WIRED.
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
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.
HepRep is to Event Displays as AIDA is to Analysis

- Both interfaces came out of a desire to cleanly separate the desktop tool from the data source.
- Use well defined interfaces to facilitate a flexible, component architecture.
- Abstract interfaces that can be implemented in a variety of languages.
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.

<table>
<thead>
<tr>
<th>Physics Object</th>
<th>Representable</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitted Track</td>
<td>Track Number: 1</td>
<td>OR</td>
</tr>
<tr>
<td>Track Number</td>
<td>Particle ID: e-</td>
<td>OR</td>
</tr>
<tr>
<td>Particle ID</td>
<td></td>
<td>OR</td>
</tr>
<tr>
<td>Points(n)</td>
<td>Pt 2 Params</td>
<td>OR</td>
</tr>
<tr>
<td>Helix</td>
<td>Pt 3 Params</td>
<td>OR</td>
</tr>
<tr>
<td>Params(n)</td>
<td>Pt 4 Params</td>
<td>OR...</td>
</tr>
</tbody>
</table>
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

- Comments: String[]
- getInstanceTreeTop(InstanceTreeName: String, InstanceTreeVersion: String): HepRepInstanceTree;
- getTypeTree(TypeTreeName: String, TypeTreeVersion: String): TypeTree;
- getInstances(InstanceTreeName: String, InstanceTreeVersion: String, TypeNames: String[]): HepRepInstanceTree;
- getLayerOrder(): String[];
- checkForException(): String;

HepRep Instance

- ID: HepRepTreeID
- Name: String
- Version: String

TypeTree

- ID: HepRepTreeID
- TypeTreeID: HRTreeID
- InstanceTreeIDs: HRTreeID[]

HepRep Instance

- Name: String
- Desc: String
- Category: String
- Extra: String
- InfoURL: String

Type

- Name: String
- Desc: String

HepRep Point

- X, Y, Z: Double

HepRep Action

- Name: String
- Expression: String

HepRep AttDef

- Name: String
- Desc: String
- Category: String
- Extra: String
- AttDefName: String
- Value: Any
- ShowLabel: Int
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)
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)
Evolution from JAS 2 + Wired 1.x to JAS 3 + Wired 4

Up to Last Week

Generic Wired 3.x Application

JAS 2

LCD Wired 1.x
JAS plug-in

LCD Java Not HepRep

Today

Generic Wired 3.11 Application

JAS 3

LCD Wired 3.11
JAS plug-in

LCD Java Not HepRep

Soon

Generic Wired 3.x Application

JAS 3

LCD Wired 3.x
JAS plug-in

LCD Java HepRep2

Eventual Goal

Generic Wired 4
JAS plug-in

JAS 3

LCD Java HepRep2

Got all experiments onto same WIRED base version. Many new features for LCD.

Get all experiments onto HepRep. Allows attribute picking for LCD.

WIRED 4 has HepRep2 as its backbone. Allows us to exploit the full capabilities of HepRep.

Babar, Geant4 and GLAST users can switch from WIRED 3 to 4 whenever they choose, since either WIRED can handle HepRep2.

Joseph Perl
Because Wired 4 will use HepRep2 as its backbone, many advanced features anticipated in the HepRep2 design will become possible:

- Provide a tabular “text” view of the event, showing all attributes and making all attributes editable.
- Allow easy save and restore of user preferences for attribute settings for any particular HepRep Type.
- Support interactive cuts on attributes.
- Allow graphics objects to be labelled with any one or more of their HepRep attributes (e.g., PT=0.4, PID=electron).
- Use association attributes to do things like: “highlight the calorimetry data associated with the selected track” or color all tracks by particle ID in one view and color them by energy in another view.

Wired will pick up various convenient features from JAS 3:

- Save and restore current configuration such as number, position and orientation of graphics windows.
- All text entered into input areas of dialog boxes becomes part of pull down menu of options for rest of that session and future sessions.
Demonstrations

- **WIRED**
  - a HepRep client written in Java based on FreeHEP
    - Full-featured
    - Runs in JAS or as separate app
    - Demos
      - HepRep from BaBar offline and online via Corba
      - HepRep from Geant4 and GLAST via XML
      - HepRep from Geant4 via JNI

- **FRED**
  - a Heprep client written in C++/Ruby based on the Fox toolkit
    - Less features than WIRED, but could be extended
    - Limited functionality, but does include scripting
    - Demo
      - HepRep from GLAST and Geant4 via XML
References

- HepRep: a generic interface definition for HEP event display representables
  http://heprep.freehep.org
- HepRep Complete Presentation (most complete description of HepRep)
  http://heprep.freehep.org/heprep2.Complete.ppt
- 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)
- 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)
- The FreeHEP Java Library
  http://java.freehep.org