HepRep forms the central part of a complete generic interface for client server event displays. The HepRep interface supports all of the desirable features of a client server event display, 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.
Connection may be slow or unreliable. Even with a fast connection, requesting data in too many small pieces causes unacceptable accumulation of call overhead. An efficient interface must be designed.
At the same time that a static, installed base of client software is being built up on user desktop computers...
...new Representables are being defined that must be propagated to all of the previously installed clients.

The server evolves, making new Representable Types available, but the user should not be required to update the client software.
Event Display Requirements

Fast:
- Minimize network traffic.
- Minimize accumulation of call overhead.

Maintainable:
- Let a static installed base of client software handle new Representable Types from evolving servers.
- Allow physicists to add Representables by modifying only the server side code.
- Avoid class explosion in the interface definition and client.

Full-Featured:
- Enable picking and physics-based visibility cuts in a simple way for all Representables.
- Provide a flexible, generic solution to the problem of pickable object granularity (give the user fine-grained pick abilities, such as the ability to pick on a single hit on a track, while avoiding having so many separate pickable objects that the graphics engine is overloaded).

But users should only have to pay the speed costs of those full features if they want those features at the given moment.
At first glance, the solution seems easy.
- The server can just give the client a reference to the Physics Object.
- The client can then use remote method invocation solutions (CORBA, RMI) to call methods on the server side.
HepRep: a Generic Interface Definition for HEP Event Display Representables

HepRep Type
- Name: String
- Version: String

HepRep Instance
- HepRep Type

HepRep KnownType
- Name: String
- Version: String

HepRep Primitive
- HepRep Point
  - X, Y, Z: Float

HepRep AttDef
- Name: String
- Desc: String
- Type: String
- Label: boolean
- Extra: String

HepRep AttValue
- Name: String
- Value: Any

ServerThread

HepRep
- Cut
  - Name: String
  - Comparison: String
  - Value: Any

+getKnownTypeList(): HepRepKnownTypeList
The design decision behind HepRep is to serve Representables, not Physics Objects.

A Representable is the 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.
HepRep Attributes

Any Type, Instance, Primitive or Point may have a sequence of Attributes:

- **Draw Attributes** (such as thickness and color) 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** that define special things to do when the user picks on the Representable (such as remove hit and refit track).

- **Association Attributes** that define loose associations between Representables (such as track cluster matching).

**HepRep AttDef**
- Name: String
- Desc: String
- Type: String
- Label: boolean
- Extra: String

**HepRep AttValue**
- Name: String
- Value: Any
HepRep Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Desc</th>
<th>Extra</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt</td>
<td>Transverse Momentum</td>
<td></td>
<td>Physics</td>
<td>1.76</td>
</tr>
<tr>
<td>PID</td>
<td>Particle ID</td>
<td>unused</td>
<td>Physics</td>
<td>Pion</td>
</tr>
<tr>
<td>System</td>
<td>Detector System</td>
<td>unused</td>
<td>Physics</td>
<td>Vertex Detector</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>unused</td>
<td>Draw</td>
<td>Blue</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove Hit &amp; Refit Track</td>
<td>remove</td>
<td>PickAction</td>
<td>Track(5).Hit(8)</td>
</tr>
<tr>
<td>TrkClust</td>
<td>Track’s Associated Cluster</td>
<td>Cluster</td>
<td>Association</td>
<td>Cluster Number</td>
</tr>
<tr>
<td>DrawAs</td>
<td>DrawAs</td>
<td>unused</td>
<td>Draw</td>
<td>Ellipsoid</td>
</tr>
</tbody>
</table>
The Attribute Named DrawAs Encodes How the Primitive Uses the Points

<table>
<thead>
<tr>
<th>DrawAs Value</th>
<th>Additional Attributes</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyLine</td>
<td>xSize, ySize, zSize</td>
<td>one for each of nPoints</td>
</tr>
<tr>
<td>PolyPoint</td>
<td>xSize, ySize, zSize</td>
<td>one for each of nPoints</td>
</tr>
<tr>
<td>Ellipsoid</td>
<td>xSize, ySize, zSize</td>
<td>two foci</td>
</tr>
<tr>
<td>Box</td>
<td></td>
<td>one for each point on inner face,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one for each point on outer face</td>
</tr>
<tr>
<td>HyperCircle</td>
<td>radius</td>
<td>one center</td>
</tr>
<tr>
<td>HyperPolygon</td>
<td>width, nSides</td>
<td>one center</td>
</tr>
<tr>
<td>HelixSegment</td>
<td>helixParms</td>
<td>one start point or end point</td>
</tr>
<tr>
<td>Text</td>
<td>string</td>
<td>one center</td>
</tr>
</tbody>
</table>

As with any Attribute, DrawAs can be specified here or can be inherited from Instance or Type.
Since graphics engines become overloaded if there are too many separately pickable objects, it is best to give the user client-side control of pick granularity.

<table>
<thead>
<tr>
<th>Pick Granularity</th>
<th>What to handle as a single pickable object</th>
<th>Example pick object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>all prims of given Type</td>
<td>all Tracks</td>
</tr>
<tr>
<td>Instance</td>
<td>all prims of given Type and Instance</td>
<td>one Track</td>
</tr>
<tr>
<td>SubType</td>
<td>all prims of given Type, Inst, SubType</td>
<td>all hits on one Track</td>
</tr>
<tr>
<td>SubInstance</td>
<td>all prims of given Type, Inst, SubType, SubInst</td>
<td>one hit on one Track</td>
</tr>
<tr>
<td>Primitive</td>
<td>a single Primitive</td>
<td>one HyperCircle</td>
</tr>
<tr>
<td>Point</td>
<td>a single Point</td>
<td>one Point</td>
</tr>
</tbody>
</table>
HepRep
KnownType
Name:String
Version:String

Recursive structure with which the server can tell the client what Types (or Type.SubTypes) are known (version strings useful for telling when to load new versions of non-event data).

ServerThread
+getKnownTypeList ( ) : HepRepKnownTypeList
+getRepresentables (KnownType,...) : HepRepType

Client can ask server for all Representables of a given Type (or Type.SubType).
Additional Control Over the Download

Client can ask for only those Representables of the given Type that pass a set of cuts on Attributes.

Client can control whether a particular download should include Draw Attributes, NonDraw Attributes and/or Points (even finer control is available via InvertAtts).

- To display without picking or cuts, need Points and Draw Attributes.
- To display with picking and cuts, need Points and all Attributes.
- To enable picking and cuts after the event has already been downloaded, make an additional request only for NonDraw Attributes.
- To display a SubType when the Parent Type is not actually to be displayed, need the Parent Type’s Attributes (for inheritance purposes) but not its Points.
The BaBar Collaboration is converting their existing WIRED Event Display Client and CORBA Event Server to use HepRep.

In BaBar, the client is in Java, the server is in C++ and the communications is by CORBA, but care has been taken to not tie the HepRep interface definition to any particular language specification or communications protocol.

BaBar’s HepRep-capable event display client will be immediately usable by any other collaboration that provides a HepRep-capable event server.

Details: contact Joseph Perl, perl@slac.stanford.edu, or see SLAC-PUB-8332.