HepRep: a Generic Interface Definition for HEP Event Display Representables

J. Perl[1]

[1] SLAC, Stanford University, California

Abstract

HepRep, the Generic Interface Definition for HEP Event Display Representables, 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.

keywords event display, visualization, interface

1. Introduction

This document describes HepRep, a generic interface definition by which event display servers can serve all different types of HEP Representables. A display client that implements this interface (plus the server management extensions covered in a more detailed version of this document[1]) could then work immediately with any experiment's event server. The HepRep interface definition addresses the following issues:

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

2. Definition of Terms

A Representable is the Spatial Information of a Physics Object (track, calorimeter hit, etc.) perhaps augmented by that object's Physics Attributes (momentum, energy, etc.).

- A Representable has Type and Instance information, for example, Type=Track, Instance=7.
- Some Representables are further defined by a SubType and SubInstance, for example, Type=Track, Instance=7, SubType=HitOnTrack, Instance=5.

The SubType is a separate Representable, distinct but related to (and inheriting some Attributes from) the Parent Type and Instance. This sub-structure issue is an important one. Consider that a user may want to see any one or more of the following three track-related Representables in the same display:

- Simple Parameterized Track: denoted as Track, defined by one endpoint and a 4-vector. It can be drawn as a perfect helix in detector space or as a 4-vector in momentum space.
- Found Track: denoted as Track.HitOnTrack, defined by all the hits found on the track. It can be drawn as a set of dots or line segments, and it also has access to the parameters of the Parent Track.

- Precise Fitted Track, denoted as Track.KalManPoint, defined by a different set of points, the steps of the KalMan fitter, with fit parameters at each step. It can be drawn as a set of dots, line or helix segments, and it also has access to the parameters of the Parent Track.

Each of these three track-related Representables contains different spatial information. Therefore a physics object like an abstract track is not termed Representable until this further distinction has been made of what exactly to represent.

3. The Client Server Event Display Problem

In a client server event display, the client is designed to exploit as much as possible the available desktop CPU and to provide a "runs anywhere" solution. The server retains the code that must live in the collaboration's standard analysis environment (reconstruction code) and the associated complete data (event data and detector constants). The client should be able to do many useful things, not just draw but also let the user:

- Pick on the display to inquire about Physics Attributes (momentum, energy, etc.)
- Pick on the display to perform various actions back on the server side (such as the often mentioned though maybe not that useful "remove hit and refit track")
- Turn on and off what is visible based physics cuts (show only tracks over 100MeV)

The connection between client and server may be slow or unreliable. Speed and smoothness of response therefore require a careful plan of how to assemble the data served. This is no surprise; the interface is the heart of any client server system. The right solution for this interface is impacted by the following maintenance problem.

As an event display program becomes popular, there are requests to represent new kinds of physics objects. At the same time that new Representable Types are being added, a large, static installed base of client software is being built up on user desktop machines. The user must not be asked to continually update their desktop client software (and automated desktop update tools such as SMS will be of limited use since important event display desktops include user's laptop and personal home machines). Instead, the existing client software (and interface definition) must be inherently able to accept new Representable Types presented by the server.

4. The HepRep Data Structure

The fundamental decision in HepRep is that what is served are not Physics Objects (tracks, colorimeter hits, etc.) but Representables (the spatial information of a physics object perhaps augmented by that object's Physics Attributes such as momentum and energy). Consider the serving of a precise fitted track.

- If the server ships the basic Physics Object, the track, then before the client can draw anything it must either go back to the server to ask for fit points, or it must itself reproduce the fitting process (requiring parts of the magnetic field map, the detector model and the latest swimmer code.
- If the server instead ships the Representable, the piecewise helix parameters, augmented by Physics Attributes such as track momentum and errors, then the magnetic field map, the detector model and the swimmer code stay on the server side where they belong.

The latter solution is the correct one.

Note that the client still retains some flexibility in how it represents this precise fitted track Representable. The Representation could be:

- “piecewise helix”, drawing actual helix pieces.
- “dot to dot”, just drawing lines between fit points.
- “fit points”, just drawing fit points with error bars.
- “simple parameters”, just drawing a simple helix.

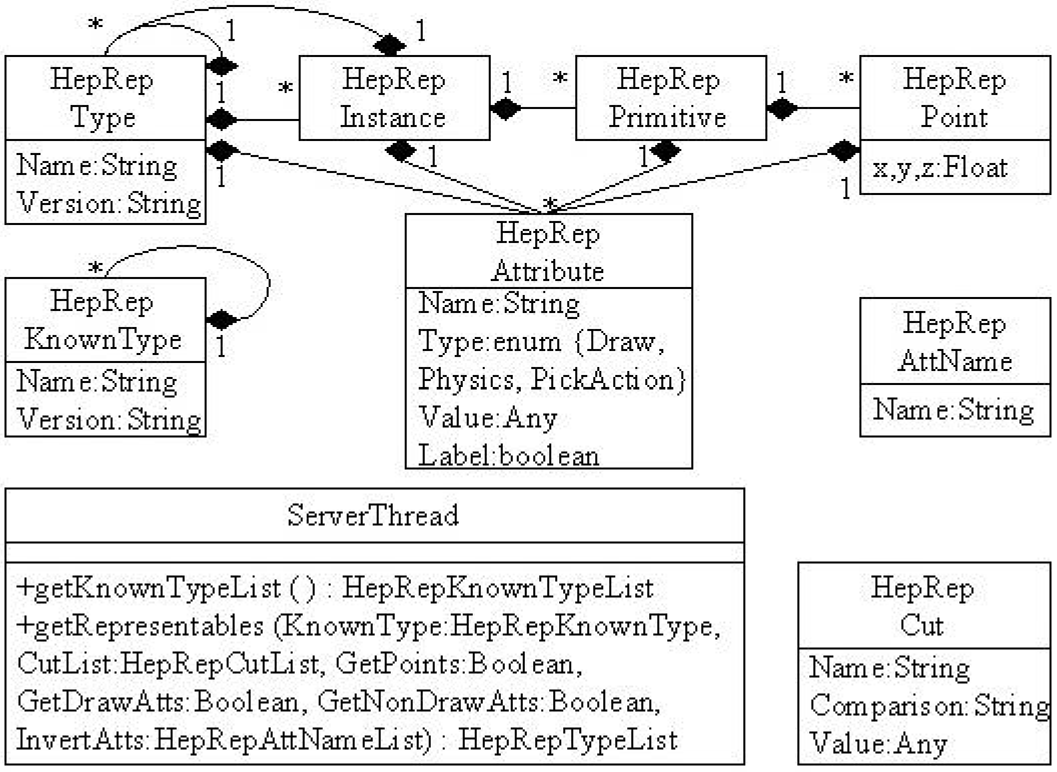

Figure 1: UML Class Diagram: The Complete HepRep Interface for Managing Representables

Ignore the lines of recursion for a moment (the lines from Type back to itself and from Instance back to Type).

- Each Type (such as "Track") has a sequence of zero or more Instances.
- Each Instance (such as a particular Track) then has a sequence of zero or more Primitives.
- Each Primitive (a basic shape such as "PolyLine" or "Helix Segment") has a sequence of one or more Points.
- Each Point contains three-dimensional spatial information (x,y,z).

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

- Draw Attributes (such as thickness and color)
- Physics Attributes (such as track momentum or hit error)
- PickAction Attributes that define special things to do when the user picks on the Representable. (Much more about Attributes later).

The kind of Primitive is specified by a special one of the Primitive’s Draw Attributes named "DrawAs". DrawAs determines how the Primitive's Points will be used (they might be the points of a PolyLine, the center of a Circle, the foci of an Ellipsoid).

Lines of recursion provide the necessary support for Representables that have not only Type and Instance but also SubType and SubInstance (for example Track.HitOnTrack).

- The recursion from Type to Type allows the Type.SubType structure to be defined in the absence of any actual Instances and provides a place to attach default attributes for all Instances of a given Type.SubType (as in making all Track.HitOnTrack default to blue).
- The recursion from Instance to Type provides a place to attach default attributes to a specific Instance of a given Type.SubType (as in making all of Track 5's HitOnTrack default to blue).
- Notice that all of these recursive relationships are 1 to 0,1 or more (so a given Instance of the Type "Track" might have two SubTypes "HitOnTrack" and "KalManPoint").

The interface thus far described is for event data, but it is easy to extend it to accommodate non-event data such as detector descriptions or other notations that are meant to last longer than one event (such as a ruler superimposed on the display). The Type just needs an additional string, Version. At each new event, the client asks the server what Version the server has for each Type. The client then requests the actual Representables only if the server has a new Version.

- For event data, Version will always be null, meaning the client should request new data for every event.
- For geometry data, an actual geometry version will be indicated, and the client will ask for new data only if the version has changed (the client may also choose to cache versions).

The terms SubType or SubInstance are just used here for convenience. In practice these are always just recursive calls to Type and Instance. The use of recursion allows for arbitrarily deep nesting. While event information does not generally require more than two level nesting (such as Track.HitOnTrack), detector geometry descriptions often involve deeper recursion.

5. The HepRep Download Method

The server thread's getKnownTypeList method tells the client what Representable Type Names and Versions (and SubTypes) the server thread knows how to serve. This does not necessarily imply that any Representables of that Type are present in the current event.

When the client wants to download all Representables of a given Type, the client calls the server thread's getRepresentables method for the appropriate KnownType.

The client also has the option to request service of just a specific Instance of the given Type. For example, the client might ask only for the Track.HitsOnTrack for Track Number 7. The download filter is implemented in a generic way so that what is downloaded is the set of all Representables of the given Type that pass a given set of cuts on Attributes.

- Each cut is specified by Attribute name, comparison operator (">", "=", "<") and value.
- Examples of cuts might be: "Particle Type=pion", "Cluster Number=5", "Momentum>1."

To provide a rich set of features in the client, the interface has been loaded down with many types of Attributes. But if the user has no desire to do picking or cuts, not all of these Attributes will be needed. The download method therefore provides a flexible way to specify what part of a given Representable Instance should be downloaded. Three Boolean arguments allow downloading only what is actually needed:

- GetPoints controls getting Points.
- GetDrawAtts controls getting Draw Attributes.
- GetNonDrawAtts controls getting Physics and PickAction Attributes.

If all are set true, the complete Representable is downloaded.

- To display without picking or cuts, the client just requests points and Draw Attributes.
- To display with picking and cuts, the client requests points and all Attributes.

- To enable picking and cuts after the event has already been downloaded, the client makes an additional request only for Physics and PickAction Attributes.
- To display a SubType when the Parent Type is not actually to be displayed, the client needs the Type's Attributes (for inheritance purposes) but not its Points.

To allow more flexibility, a specific list of Attribute names can be provided in InvertAtts. Attributes named in this list have their normal download decision inverted (for InvertAtts details see the more detailed version of this document).

The download method may look complex, but a particular client is under no obligation to support all of this complexity. It can always just ask for the complete Type, as in:

- getRepresentables (someKnownType, null, TRUE, TRUE, TRUE, null);

6. HepRep Attributes

Since not all Attributes need to be downloaded at once, the developer is free to implement as many Attributes as may be useful.

6.1. Draw Attributes

Draw Attributes relate to how the object is displayed, such as: Thickness, Color or DrawAs (the Draw Attribute that decodes how the Primitive uses the Points).

The client might include a Draw Attribute editor, allowing the user to change Draw Attributes (for example to change the color of Tracks).

- An interesting example is that if the Primitive has two points and one were to change DrawAs from Ellipsoid to PolyPoint, the representation would change from showing the ellipsoid to showing the two foci.
- The editor might allow a new Primitive to be added to an existing Representable, for example adding an arbitrary text label to a HitCrystal.
- The editor might also allow the creation of whole new Representables, for example adding an overall title to the plot.

6.2. Physics Attributes

Physics Attributes relate to qualities of the original physics object, such as: P, Pt, Pz, E, Q, Particle Type, Track Number, Subsystem (such as Drift Chamber or Vertex Detector), EndCapOrBarrel (such as +EC, Barrel, -EC), Data Source (such as MC or Real).

Visibility cuts can be made on these Attributes. The client just needs to provide a slider for every Physics Attribute. The user can then cut out displayed objects by manipulating these sliders. Visibility cuts can be made on any Physics Attribute even if this Attribute is found below the current pick granularity (the level at which groups of Primitives pick together).

- If the visibility cut is on an Attribute that exists at the current pick granularity, the client software can handle the cut quickly (since it is just toggling separate pickable objects).
- If the visibility cut is on a Attribute that exists at a finer level than the current pick granularity, the cut can still be done, but the client software will handle it more slowly (since it will need to go back to rebuild the pickable objects to leave out what was cut).

If the user has established some cuts that they do not intend to widen very often, the event download can be sped up by doing these cuts on the server side simply by adding the relevant cut to the Download method cut list. Server side cuts explain why physics indices such as track number must be supplied as Attributes rather than being inferred from an Instance count. With cuts in place, such an Instance count may or may not match the original physics index.

6.3. PickAction Attributes

PickAction Attributes define things to do when the user picks on a Representable. Two such actions have already been discussed, picking to inquire about a Representable's Attributes ("pick to inquire") and picking to bring a Representable into a Draw Attribute editor ("pick to edit"). Additional PickAction Attributes may be freely defined.

Some PickActions (such as two just mentioned) do not require any work on the server side. Others (such as “pick to remove hit and refit track”) do require work back on the server. In such cases, the PickAction would identify the original physics object to the server by relevant indices or object references stored as additional Attributes.

Ideally, the client would allow the user to pick in a fine-grained way, able to separately pick every Primitive. But 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.

Different levels of pick granularity are best explained by example. Here is what pick to inquire would report if the user were to pick on a Track.HitOnTrack at various granularities:

Pick Granularity	Picking on a Track.HitOnTrack reports
Type	only that a Track or its descendant has been picked
Instance	above plus details about the particular Track (such as momentum)
SubType	above plus that a HitOnTrack has been picked
SubInstance	above plus details about the particular Track.Hit (such as hit num)
Primitive	above plus details about the Primitive (Primitive type, color, etc)

The server always ships the Representables in the same detailed structure; therefore a change in pick granularity only affects the client. No new information need be requested from the server.

7. Conclusion:

The interface shown in this document assumes that the client has already attached to a server thread and that a server thread has already been set to the desired event. It is a generic interface for HEP event display Representables, not a generic interface for the complete HEP client server event display. But the missing parts of this larger interface are actually quite minimal:

- some methods for the client to initially obtain a server thread
- some methods for the client to set the server thread to the desired event.

When augmented with these server management extensions (described in a more detailed version of this document), the HepRep Generic Interface for Event Display Representables provides a complete generic interface for a HEP client server event display.

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.

References

[1]J. Perl, ``HepRep: a Generic Interface Definition for HEP Event Display Representables", SLAC-PUB-8332, Stanford, California, January 2000.