29 March - Toward a common HEP vision

Game engines:

Matevz, Ilija: discussed the possibility of using game engines in our tools
Sergey: yes, graphical tools offered by game engines are very nice, but we should pay attention to all the other features and infrastructure we need, which is not only graphics. For example UI
Ilija: game engines have powerful UI tools, very modular and customizable
Ric: I think game engines have an issue if we want to use them in our integrated tools
Ben: Not entirely happy with relying entirely on business model of commercial company not changing. OK if this is one of several platforms, but not as only one. (Check open-source licence and policy etc.)
Tom: Being open source only mitigates part of the risk if game engine changes terms or goes out of business. Great, we have the code but then do we have to then support evolving APIs for different VR environments and SDKs?

Modular software:

Ric: the main point I think is to build our HEP software in a modular way, to let us switch the base packages (the open-source ones), if they will become outdated or not maintained anymore.
Ben: agree strongly! Not easy, but important.

Qt and GUI:

Sergey, Ric, Matevz, Barth: Qt is a good example of common packages which can be used; in the past it had problems with Mac Os X11 which was needed by ROOT, but probably now it's fine; Qt now is open-sourec since some years, and we could even contribute to it as HEP community, so we can develop the features we need without maintaining teh whole package ourselves.
Barth: Qt for Mac now seems OK. Qt now is multi-platform: one code base, many platforms.

Qt3D:

Ric: coul;s be interesting testing what presented yeseterday by KDAB. Having 3D and GUI into one package could be fine. It's well maintained because used in automotive and industry; and we could use that for our tools.
Ric volunteers to build a small demostrator of using Qt 3D.

Common data format:

Sergey, Matevz: if we could agree on a common exchange data format, that would let us easily change our base packages and clients, plugins, etc…

Barth: Do you mean a common protocol or a common content?

Sergey: we want common functionalities of the clients exchangable.
Barth: at least we should define a common format; is there any constraint/request from the experiments?
Matevz: common data formats for what? For event data? For metadata? For geometry? For visual objects?
Matevz: We could have a common part of common data, plus experiments can add their own part of data in the stream.
Sergey, Matevz: let's start with a test of common information among experiments: which might be geometry volumes, or common objects like tracks and jets.
Sergey: TGeo classes can be exported to JSON, transported and open in the browser. We don't need to develop another JSON
Ric: is that usable as module in other code?
Sergey: There's a function in JSROOT.
Ric: can I use that in an exporter lets say from ATLAS GeoModel to have a JSON stream of data? From C++ or something else?
Sergey: Yes, this can be used out of the box.
Tom: Is the function a core thing in JSROOT or is it only needed for special cases? Is it easy to include in custom code?
Sergey: code is very modular, functionalities are used when needed.
Ric: could you provide a simple demostrator how to use this ROOT JSON streamer to send and receive a JSON stream of data?
Sergey: of course!

Sergey volunteers to build a ROOT JSON usage demonstrator
Tom volunteers to build a demontrsator of a client of Sergey's tool

Generalized / "templated" GUI

There are several parts of GUI that allow inspection of or changing values of a given object. Those parts can be generated automatically, given instructions on allowed ranges / precision. The hard part are the callbacks … how user actions in GUI translate into actions. Obviously, each GUI frontend would require its own GUI generators from the common template.

HSF repository:

Alex: collect functionalities and functions we are HEP developers we can share with the outside world

Matevz's attempt at domain decomposition

1. Data representation / formats
    a. For transport between server / client.
    b. As server side representation (like Eve classes, but can be more general, i.e. include more details).
    c. format for exposing the low-level experiment data format and available methods (what client can ask about). This is effectively a dictionary.
2. Data request / interaction protocol
    a. How to identify objects, containers

        i. Set of top-level, standard “entry” points, e.g. Configuration, Event, EventDB.
        ii. Sort of object hash / unique id is probably an easy optimization over URI
    b. Format of object detail request and response
    c. Format of method invocation request
3. Server / client state updates & synchronization
   Method invocation can have significant side effects. How do server and client figure out what needs to be transferred? One possibility:
    a. server notifies client with a list of changed objects
    b. client responds with a list of objects it is actually interested in
    c. server sends requested details
4. GUI / 3D graphics frontends
    a. It seems there is interest in supporting the following:
        i. JSROOT
        ii. some other combination of three.js / web-gl / web-gui
        iii. Qt
        iv. Unreal and/or Unity
    b.