Agile Visualizations Design Document

AMS-102

**Thomas Sgouros & Julio Carrera, v0.0.2 on 2018-03-07**

# Overview

This is the design document for the Agile Visualizations project jointly developed with Brown University. As the project is still in the development phase this document will be updated as needed.

The project is based on the ParaViewWeb architecture. As such it consists of a Python interface to a ParaView server and a Javascript client. The library dependencies for both will be listed in this document.

# Goal

The goal is to make a visualization pipeline for hydrodynamic model data that is easy to use on a portable device like a tablet. The idea is to make the visualizations available on the shop floor to users relatively unsophisticated in graphics and visualization.

Paraview supports a wide range of visualizations and supports use in a distributed environment, as well as supporting parallel processing for big jobs. That is, as the engine that creates the visualization, it appears to be flexible enough to be a visualization server, with many performance enhancements available at the server end. This choice was made before the project began, so we will not revisit it here, but will discuss the choices made based on that one.

# Requirements

The requirements are to support several basic visualizations:

1. Mean age using iso-surfaces. To see iso-surfaces of mean age, and how they interact with the vector velocity field.
2. Velocity field streamlines, shaded with scalar values, such as the mean age.
3. Simple volume renderings, of such data as mass transfer, or oxygenation.
4. Permutations and combinations of the above.

All these visualizations are relatively straightforward to a sophisticated Paraview user, but they require many steps. At least as important are the many options in the Paraview user interface that are irrelevant to the desired purpose and thus must be ignored. These options make for a powerful interface, but a confusing experience. A useful application should not merely show data, but should make it easy for a novice to get satisfying results quickly.

The UI requirements are thus:

1. Should run on lightweight processors, like laptops or tablets, maybe even phones.
2. Should guide the user to making interesting views of the data they wish to examine.
3. Should preserve screen space in order to be used on small devices.
4. Should allow the user to save the choices made in generating a view—the variables, the colors, the scale, and the view position—and be able to apply those choices to a new dataset.

# User Interaction

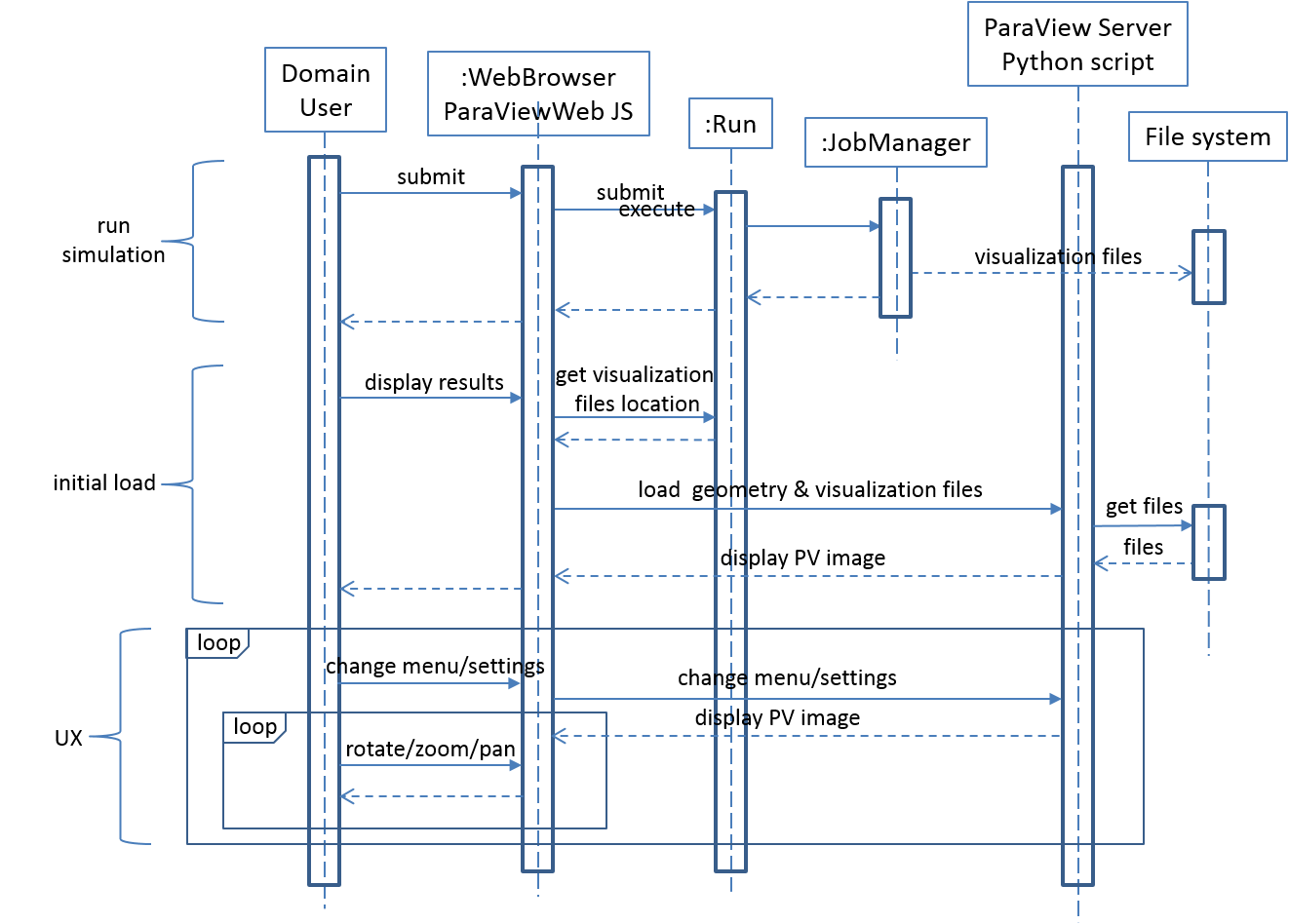
The basic Paraview user interface is heavy and confusing and takes up a lot of screen space better spent on visualization. The vast majority of the options available at the top level are not relevant to the proposed Amgen application. There are ways to avoid displaying all the menus, but the subsets that remain when you turn off this menubar or that one do not match the subset we wish to use.

Paraview supports a ``ParaviewWeb'' interface, and this seems like a way to address these issues, as well as move the data processing off the small portable processor. There are two full interfaces already built and available: Visualizer and LightViz. These were evaluated, and though their design is more modern than the basic Paraview interface, they are probably not the appropriate solution. That is, they seek to preserve all of the flexibility of the original application, albeit in a friendlier package, and thus provide no guidance to the user. Though their interfaces are not as obviously intimidating as the basic Paraview screen, the options are as numerous, and there is no guidance available to the user through them. Learning to use one of these interfaces will take as long as learning to use the original Paraview interface.

The Paraview server has an optional Python scripting interface available through the pvpython binary, and this can serve to ``channel'' the user's choices. That is, we can create a small number of Python functions to generate each of the desired visualizations and control them from the remote web interface that displays the result. A modest set of menus can be used to guide the user through the choices necessary for the visualization.

# Sequence Diagram

Julio shared a graphic about how the visualization sequence would work, and I have edited it slightly to reflect how I think it would go. Time runs down in the image, so the process begins with the user request (RQ) for a model run in the top left. The shaded boxes represent the parts of the system that Brown/CCV is responsible for.



In the above, the user uses a web browser to request a model run, and the request is forwarded to some run system and then to a job manager, per Julio's original description. This part is Amgen's responsibility.

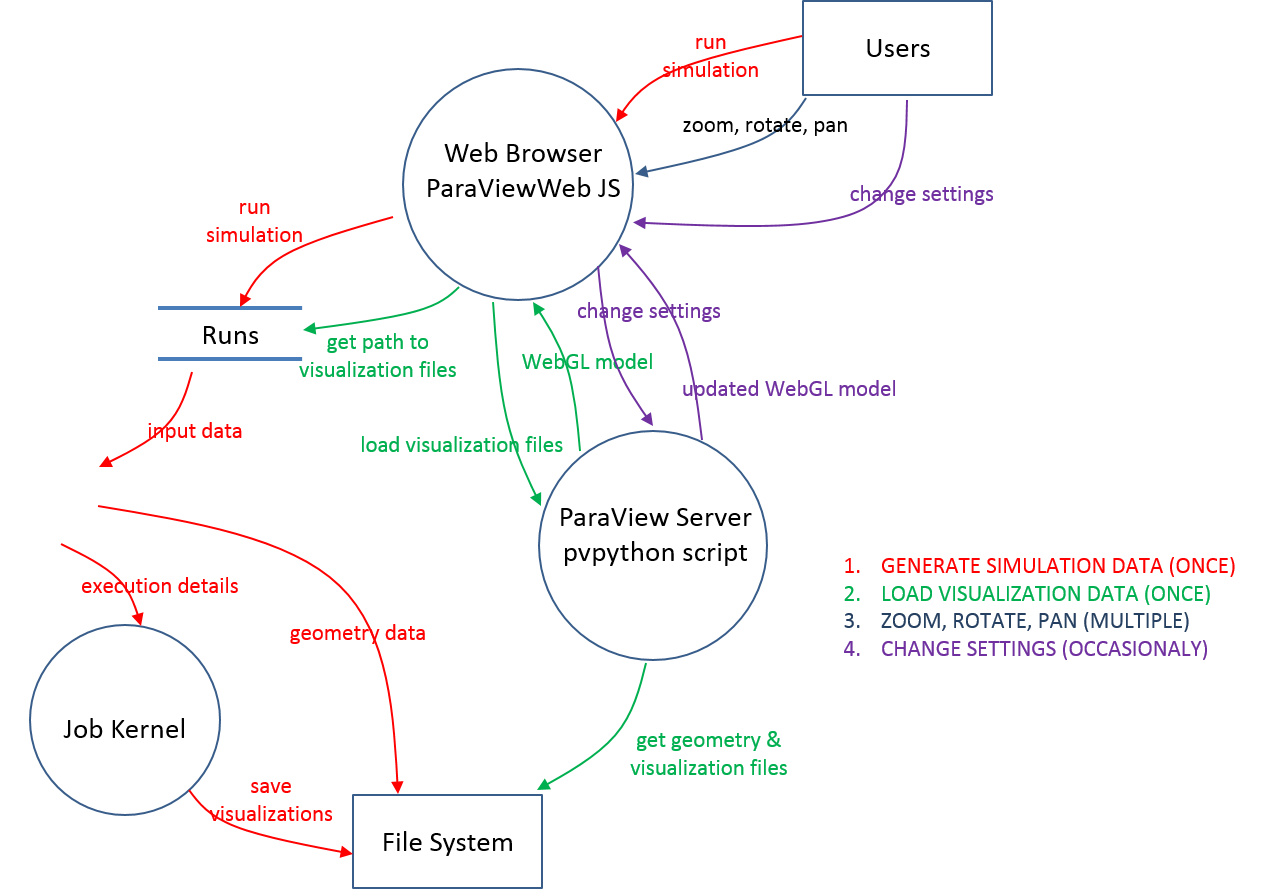
Subsequent to the generation of the model data, the user initiates the next step by requesting a visualization of the simulation results. That RQ is forwarded to the Paraview server, activating one of a small arsenal of Python functions pre-loaded into the pvpython server. That function retrieves the relevant simulation results data from the Amgen file system, wherever it is, and makes the data available to another function, still within the pvpython server. That function in turn invokes the ParaView server to generate a 3D visualization representation of the results data. The 3D visualization is forwarded to the ParaviewWeb Javascript infrastructure running in the web browser, where it is displayed to the user.

The user's manipulation of the image data while the user rotates and zooms in and out is performed entirely on the web browser. That is, no exchange of data with the ParaView server is needed until the user changes one or more visualization parameters (e.g. color). The user will likely iterate this process, refining the visualization parameters, until the desired result is achieved. Only the first RQ for a Paraview model will result in a request for simulation results data from the Amgen file system.

# Loading Model Geometry

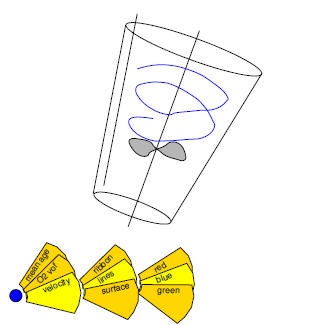
Loading two datasets for comparison side by side only makes sense when they use the same model geometry. Thus, it makes sense to import the geometry separately from the simulation results datasets.

The data flow diagram below shows the Run Kernel saving the geometry to the file system as a run is submitted for execution. The geometry will then be imported into the ParaView server at the same time as the simulation results.



# User Interface Mockup

The other important interest in developing this application is to make the loops shown in the diagram operate as efficiently as possible. The typical usage of a visualization application involves an iterative approach to creation: make something, edit one parameter, change another, and so on. In the process, it is often challenging to remember what the options are, as well as remembering what was actually done to create the view on the screen.



Above is a crude mockup of a user interface meant to maximize the screen area used for viewing the model, and an indication of how the menu system will work. The idea of the menu structure is to present the choices to be made across the bottom of the view, in such a way that when the menus are displayed, they will act as a visible record of the choices that were actually made to create the view. The goal would be to make the choices not only easy to read, but easy to edit as well. The interface visibility will toggle with the small blue button to the left and the menus will ``unfold'' from that button.

We are currently evaluating a range of Javascript UI library frameworks to see which is best suited for the purpose. Bootstrap, Webix, and Blueprint are popular. There is one called OpenUI 5 that might be worth examination; it is freeware, but supported by SAP.

# Support for Visualization of Two or More Datasets

TBD

# Process for Starting the ParaView Server

TBD

# Process for Packaging the ParaViewWeb Javascript

TBD