Web-Probe Architecture

# Main Components:

* Web-Probe GUI/client-side (javascript)
  + Query View
  + Product View
  + Local Justification View
* Web-Probe server-side (Java)
  + PML API
* Visualization Cache
* VisKo
* PML-SPARQL-Services

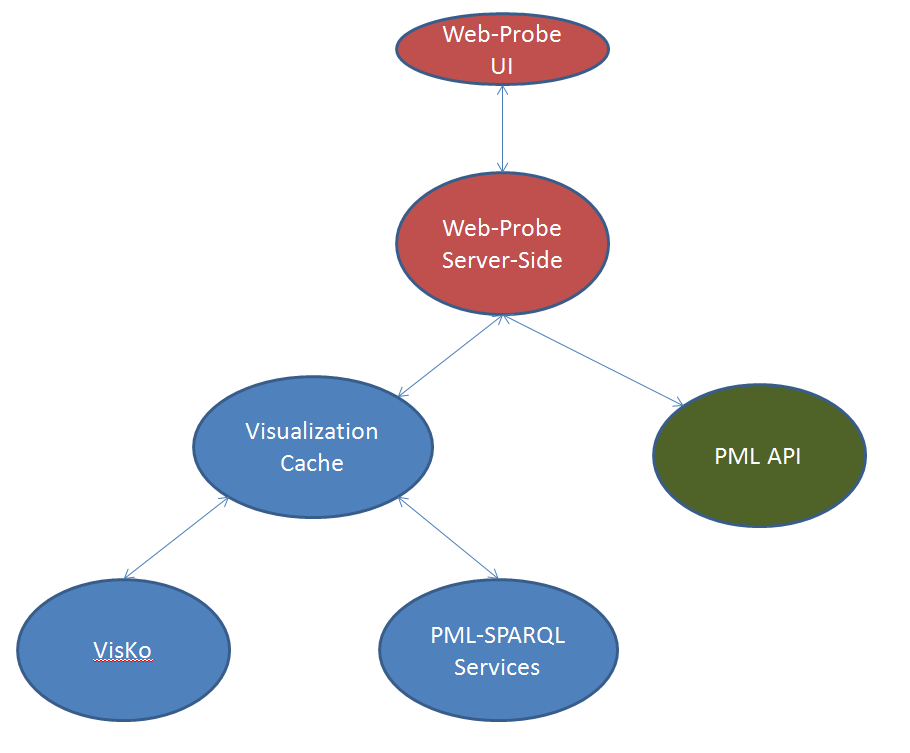


Fig.1 – Web-Probe General Architecture

# Web-Probe GUI

Consists of HTML, CSS, JavaScript/JQuery to present the user interface through a web browser, as well as to capture and react to user input, such as mouse movement/clicks and keyboard input. The GUI includes logic to know how to transition from view to view, and when new information must be queried from the server and how to display it once the response is made.

# Web-Probe Server-Side

Consists of Java code to interact with the PML API and Visualization Cache. Also, includes classes and methods to return JSON (JavaScript Object Notation) representations of Java objects. The Java code is primarily used to act as an intermediary between both the PML API and Vis Cache (on the server-side) and with the Web-Probe GUI on the client-side. There is specific information required from the PML to be displayed on the GUI. This information is gathered by a call to a particular Java service which communicates with the PML API and consolidates this information in a Java object. This Java object can then be translated into JSON to be returned to the client-side JavaScript, where it can use this information and display it accordingly.

# Communication between Client/Server-side

Communication between these two main components is done using Direct Web Remoting (DWR), an approach of AJAX (Asynchronous JavaScript and XML) between Java and JavaScript. DWR enables JavaScript to call any Java class that the developer chooses. The particular Java classes that are accessible are documented in an XML file called “dwr.xml”, in order for DWR to give JavaScript access to any methods/services provided by these classes. The execution of any Java code is then done asynchronously, which is a property of AJAX communication. The response from Java to JavaScript is then packaged into JSON to be returned to the client-side JavaScript.

# Visualization Cache

The Visualization Cache is responsible for caching any visualizations known to be associated with PML documents. In this way, any future necessity to display a visualization while browsing PML can be done with speed; Eliminating the need to generate, map, and/or transform any particular visualization when it’s needed. The Visualization Cache uses the PML-SPARQL-Services to acquire a list of possible PML documents that could have visualizations, and then uses VisKo services to try and generate any possible visualization(s) that in can than cache on the server.

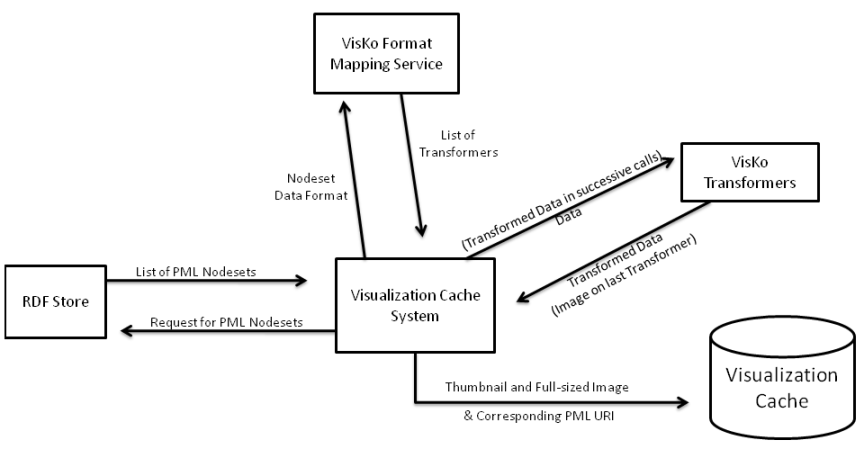


Fig.2 – Visualization Cache Process