Web based data visualization solutions in quality assurance

Sebastian Wozny, Student, TUM,

Abstract—The growing amount of relevant data in projects demands heavy use of web-based technologies for knowledge discovery and global sharing of information which can be achieved by developing interactive web-based tools in order to process, analyze and visualize scientific data on the web. Modern web-based technologies such as HTML5, CSS3 and powerful JavaScript Libraries offer the prospect of interactive visualization in current browsers. Highly interactive web applications are an important means to research communities as well as companies in order to would like to visualize their work in an accessible manner. In this paper we present an up-to-date study-analysis and compare some of the promising features as well as performance of some freely available JavaScript Libraries that helps interactive visualization of Graphs on the web and compare it to Matplotlib as a baseline for non-interactive visualization.

Index Terms—Data Visualization; Web Technologies; D3.js; Angular.js; Matplotlib

1 INTRODUCTION

Traditionally the choices for visualizing data in applications were non-interactive plotting libraries such as GNUPlot1, matplotlib2 or other libraries provided by the language. The introduction of highly interactive web based tools like Gmail, Asana or Dropbox have raised user expectations on interactivity of web based data visualization. The language of the internet is JavaScript and in recent years several JavaScript libraries have emerged, that aim at delivering a highly interactive data visualization. JavaScript technologies keep on evolving over time, although until recently it had not yet matured and did not have much capability to handle large scientific data sets in order to render and visualize on the web. Web based data visualization has the potential to revolutionize the way we process data because it can be readily accessed without setup in a web browser. In global companies, non functional quality assurance gathers performance data from many measurements around the globe and aids decision making at all levels of technical expertise. Data visualization reduces cognitive load when presenting large datasets, allowing decision makers to extract important information quickly. Interactive elements are important to enable further filtering and extraction of data.

1.1 Prior work

Work done so far on the comparison of JavaScript visualization libraries indicates that modern JavaScript libraries like D3.js, Cytoscape.js, node.js, sigma.js and jsPhyloSVG have evolved far enough to be viable options when visualizing datasets on the web and offer interactive features like mouse over and click features. [1] Especially Data Driven Documents or D3 [2] [3] [4].js has gained a lot of traction lately and a wide array of examples can be found freely available online. It is the base for many other visualization stacks which build abstractions on top of D3 like Alchemy.js or nvd3.js.

2 MOTIVATION

Often, data visualization is not the core problem domain, but rather an information delivery mechanism. Instead of developing the delivery mechanism resources should be should be spent on solving the core problem. The author therefore argues that when comparing JavaScript libraries usability is a major factor, in addition to performance and completeness. When implementing behavior in JavaScript, developers are forced to deal with several challenges posed by the language. JavaScript is an object-oriented multi-paradigm language which supports object-oriented, functional and imperative programming and supports event driven I/O. However its concept of prototype based object-oriented programming is very different from other object oriented languages that are commonly used in science and industry like Java, C++ or scripting languages like Ruby and Python. In addition JavaScript is very permissive, with few restrictions to help and guide developers, which is why JavaScript code often contains anti-patterns and is poorly maintainable. Emergent JavaScript frameworks provide an abstraction layer, implementing a Model-View-Controller architectural pattern. The Model-View-Controller pattern is an established pattern in software design and allows programmers to focus on the business logic of their algorithms, and defer the representation of elements to the framework. On the other hand, open source frameworks such as nvd3.js or rickshaw promise minimal effort to implement a basic set of visualizations. In this paper the author compares the performance, features and usability of several solutions to implement data visualization
3 Open source frameworks based on D3.js

D3.js³ is a JavaScript Library developed by the Stanford Visualization Group from Stanford University. Its main purpose is manipulating documents based on data. It provides tools for efficient manipulation of webpages combining modern web technologies like Cascading Style Sheets (CSS3), HTML5 and SVG (Scalable Vector Graphics). D3.js is meant for implementing interactions, animations and complex and dynamic visualizations in the web. It includes efficient JavaScript functions to select and create HTML and SVG objects, configure their appearance and transitional an behavioral effects. [1] JavaScript libraries like d3.js offer the core tools to implement powerful tools for visualizing data on the web. However they require knowledge of the JavaScript language, which presents an obstacle to programmers unfamiliar with the language. Open source frameworks based on d3.js aim to provide turnkey solutions to data visualization.

3.1 rickshaw

Rickshaw⁴ is a JavaScript toolkit for creating interactive time series graphs, developed at Shuttlestock, and builds on top of d3.js, but requires the jQuery library for certain extensions. [5] It includes by default different types of graphs such as area, bar, stacked, line and scatter. Additional components other than the actual graph or chart can be added with extensions and offer functionality like scaled axes, showing details on hover, legends and drag and drop reordering. It promotes an object oriented approach, so different extensions are instanciated and then connected to the fundamental chart object. The project is stable and has not been modified in ten months.

3.2 nvd3.js

novus d3 or nvd3.js⁵ is a JavaScript data visualization library built on top of d3.js often used in research. [7] [8] aimed at providing encapsulating d3 behavior and reusable charts, while giving programmers the option to leverage the full functionality of d3 if required. It offers over twenty pre-made charting types, including pie charts, candlesticks, bar charts and line charts. To configure extensions like Axes and labels the chart object exposes methods which can be chained to allow for heavy customizability. There are a lot of examples available online, however the documentation is lacking Novus d3 underwent drastic breaking changes in the past, but has not been updated in over half a year.

4 Data binding frameworks and libraries

Implementing complex data visualization in JavaScript is difficult for developers otherwise not used to the language. Turn key solutions as described above implement lots of common use cases, but individual requirements may take development effort. Aimed at reducing development effort, especially for complex concepts, The data binding pattern is an established technique to couple user interface (UI) elements and data objects. [6] It is referred to as the ability of a framework to track changes in the model and reflect these changes in the view without interaction of the programmer. Dijkstra’s statement “Our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.” [11] is still true today and data-binding frameworks enable developers to substitute dynamic event-driven interaction with declarative state-based logic and representation. Arguably this makes it possible to implement much more complex logic, allowing for more interactivity from the user. Data binding is currently implemented as JavaScript code by frameworks and libraries performing so called dirty-checking when models change. This incurs a considerable loss of performance.

4.1 Backbone.js

Backbone.js⁶ gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to existing API over a RESTful JSON interface. [9] It is a minimal library based solution developed by Jeremy Ashkenas, the inventor of CoffeeScript, to implementing the Model View Controller architectural pattern on JavaScript. It only provides the separation of concerns, while the programmer is responsible for connecting the different parts and making it work together. The only hard dependency of backbone.js is underscore.js, which makes the framework very lightweight. It is under development but no breaking changes are being introduced.

4.2 Ember.js

Ember.js⁷ is an open-source JavaScript application framework, implementing a full Model View Controller pattern in JavaScript on the client side. [10] Its design focuses to provide a wholesale MVC solution to the client-side application problem, unlike Backbone which requires connecting code. It relies on one way data flow, separating changes in the data which travel “down”, from user interactions which travel “up”. It provides a set of tools that promise to implement best practices, such as Ember data which implements Object-Relational-Mapping(ORM) , Ember Inspector, an extension for contemporary browsers like Mozilla Firefox and Google Chrome and Liquid Fire for animations between transitions. It is under development, following a six week release cycle and may introduce breaking changes on major version updates.

4.3 Angular.js

AngularJS⁸ is an open source web application framework supported by Google Inc. It aims to enhance the functional-
ity of JavaScript by supporting modularization on multiple levels. Functionality of applications is broken into parts that can be tested and reused independently. AngularJS is built on a monolithic design and makes heavy use of the dependency injection pattern. It promotes two-way data binding, meaning that there is no separation between modification of data through user interaction, or programmatically. It is currently under development with the 1.3.4 version being in its maintenance phase and the 2.0 version introducing breaking changes.

5 PERFORMANCE COMPARISON BETWEEN NVD3.JS AND ANGULAR.JS

For determining the viability of different technical data visualization solutions in the context of large data sets, a solution for a line graph obtained from novus d3 was compared to a custom implementation in AngularJS. Both solutions took roughly the same amount of time to implement.

5.1 Technology use

The system is based on a Ubuntu 14.04 operating system\(^9\), with Cherokee\(^10\) serving as a reverse proxy for the underlying wsgi web server. The server backend was written in Python\(^11\) using the django\(^12\) web framework to access a postgresql\(^13\) 9.3 database. In the frontend jQuery\(^14\) is employed alongside with AngularJS and nvd3.js respectively. Two line charts were implemented to measure their respective performance. In nvd3 a simple line chart, visible in Figure 2, was configured, which featured interactive mouse over and click functionality. To demonstrate more complex interactive features, a custom selection mechanism was implemented in AngularJS, which enabled viewers of the chart to filter out irrelevant data by raising a sliding difference filter by means of a slider. The chart in Figure 3 was implemented in a AngularJS html template, only specifying declarative relationships, only the filtering logic is expressed in JavaScript code.

6 METHODOLOGY

To measure the performance of the respective solutions several metrics are possible, such as measuring the time to render or processor load. In this paper the total time from sending the HTTP GET request, which requests the chart until the finished rendering of the chart is the primary indicator of performance. This is justified because it has the biggest impact on user experience, which is critical for user acceptance. Additionally the composition of the total load time will also be analyzed in three parts: 1) Time until the page is loaded, which is a first HTTP request to retrieve static data. 2) Time until data is loaded, which is a second HTTP request transmitting the data to be displayed. 3) Time until the chart is visible and loaded. The nvd3 chart was profiled using the standard chrome developer tools, while the profiling for the AngularJS chart was done with batarang\(^15\). [13] The test data consisted of 5 datasets containing 100, 1000, 3000, 6000 and 10000 elements respectively. The result was calculated as the average timing over 5 repetitions. The Cherokee server ran on a dual core virtual machine, and the Google Chrome client was run on a quad core Intel Core I7-3740QM.

15. https://github.com/angular/batarang

---

7 Results

As expected the time until page load and time until data load were near constant, with the data loading phase exhibiting a slight linear growth. The results in Figure 4 and 5 show the average loading times of the two solutions in seconds. The nvd3 based solution performs well until a set size above 3000 elements to be visualized. The total load time did not exceed 5 seconds up to this limit. The visualization only took a fraction of the total time. The time to visualize data grows super linearly with set size and the maximum total time was 19.5 seconds. Set sizes below three thousand produce no noticeable delay and the interactive features remain operable. Above three thousand items the initial delay appears to grow quadratically, however no impact on the operability of the interactive features such as mouse hover and mouse click were noticed. The AngularJS based solution performs well with up to 300 elements, causing no noticeable delays. The custom filtering logic, as well as mouse over and mouse click features work reasonably well until this limit. Set sizes above 300 cause the framework to freeze and become unresponsive. With a set size of 1000 the initial delay was intolerable and the interactive features broke to the point that it was impossible to reconfigure the chart. Set size 3000 only completed after excessive waiting and not every attempt to display the chart succeeded. The tests with 6000 and 10000 elements could not successfully be completed because the script became unresponsive.

8 Conclusion

In this paper it was shown that data visualization using modern web technologies can be performed without becoming a JavaScript expert, both because of turnkey solutions like novus d3 and because of the emergence of data-binding frameworks like AngularJS. Basic visualizations of data like line charts, pie charts, stacked charts and scatter charts are efficiently implemented by novus d3 and also serve to visualize large sets of data up to ten thousand elements. AngularJS allow the user to apply data selection algorithms easily and enable developers to quickly develop custom data visualization solutions. One advantage of using custom data visualization solutions is that no reshaping of the data to fit the requirements of the used framework needs to take place. Furthermore, custom requirements for interactive elements can easily be met and connected to the graph. It runs about an order of magnitude slower than novus d3 charts. It has been shown that the performance of AngularJS based solutions degrades severely for large datasets, which has been attributed to the superlinear algorithmic complexity with regards to the size of a dataset of the performed dirty checking.

8.1 Further work

The proposed ECMAScript 7 wc3 standard “Object.observe()” specifies an efficient means to detect changes that occur to a JavaScript variable natively within the language, which eliminates the need for dirty checking in frameworks and libraries. Preliminary research [?] suggests that the cost of data binding might implies that it has a 20-40 times improved performance over dirty checking. Further work should investigate the performance gains for data visualization libraries that can be obtained by leveraging the new standard. Particularly, the next major version 2.0 will take advantage of Object.observe(), when it is supported by major browser vendors. Currently Object.observe() is only implemented in Google Chrome.

Acknowledgments

I would like to thank Dolby Labs Germany for letting me carry out this research during my internship.

References