Interactive Data Visualizations using JS lib D3

Kantha Cikaiahgari
Pitss GmbH
Stuttgart

Keywords:

Visualizations, Document Object Model, Scalable Vector Graphics(SVG), tool tip

Introduction

Data visualization is the pictorial or graphical representation of the data. It deals in figuring out the ways to translate data into visuals that are more readily understood. It is a key feature in the success of data analysts and businesses. It helps the analysts and decision makers in effectively understanding the difficult concepts and in identifying the new patterns. The specific requirements of the visualization tool are to give an overview of the temporal sequence of processes of a selected execution, reveal and depict dependencies between executed processes, and provide temporal context for the selected execution to show deviations from the schedule observed in other executions.

In modernizing large legacy software systems, it is mandatory to identify all the systems involved and plan future architecture of the modernization projects. For example, it is necessary for a Company to know the number of modules, DB schemas/objects available and the way they are related to each other. So that they can plan or think about changes in the existing architecture or may also adapt new architectural Concepts. Here comes the necessity of Visualizations. Businesses should have a unified view of the data where the details of each Project should be stored in such a way, that it can be fetched to have a single management view, regardless of platform or file structure differences.

The key objective of a business usecase is to illustrate how the business processes are used by its partners and users. However, the density of the data makes it difficult for decision makers to use the data effectively. Here plays the data visualization a key role. It presents large data in an easy to understand and graphical format to provide decision makers perceptive information.
Different types of Data can have different requirements or usecases where each type of data has to be visualized in different ways. For example, a business process flow can be better visualized using BPMN (Business Process Modeling Notation) than any kind of graphs.
The probable primary usecases for the data visualization include

1. Navigation Flow, how a user could run through a system with different applications.
    a. Code Flow, how the algorithm could run through different units of the application. E.g.: Call Stack
    b. Data Lineage, how data could run through a system of different ETL (extract - transform - load) components.
2. The process flow, how user navigate in real for a single process through the system. That could be aggregate for different user and different objects inside the system.

The Code flow can be better described using flowcharts and UML diagrams and the Process flows or business processes can be better visualized using business process flow diagrams. In this document, I

focused on the Navigation flows and Data Lineages which are stated in the below table with the model, respective visualization and the supported layouts of each model.

Model	Visualization	Supported Graphs or Layouts
Network	Node/link Diagram(uses link based layout algorithms)	Dagre, CoSE, cola
Tree / Hierarchal	General tree Visualization	Dendogram, treemap
Temporal	Visualizing Data lineage	Sankey
Arbitrary Graphs	Node-Link Graphs, Matrix	force directed graphs

TABLE 1. GRAPH MODELS AND SUPPORTED LAYOUTS

JavaScript libraries are the best platforms which provide built in functions to be utilized. Specifically for visualizing on the web, D3 libraries have been evolving throughout the implementation of more complex functions which manipulates data, either to visualize statistical or scientific data. A collection of toolkits are in use to help and support the information visualization applications.

D3 JAVASCRIPT LIBRARY

D3 stands for Data Driven Documents. D3.js is a JavaScript library for manipulating documents based on data. D3's emphasis on web standards provides the full capabilities of modern browsers without binding to a fixed framework, combining powerful visualization components and a data-driven approach to DOM (Document Object Model) manipulation. DOM model is explained in the below section. D3.js focuses on binding data to DOM elements. D3.js supports in data exploration by providing control over data's representation using interactivity.

The powerful visualization framework helps to produce dynamic, interactive data visualizations of all kinds using HTML5, SVG, and CSS3. Many charting libraries are built on top of D3 library. Its popularity is due to its flexibility. Since it works seamlessly with the existing web technologies and can influence any part of the document object model, it is as flexible as the Client Side Web Technology Stack (HTML, CSS, and SVG).In addition, it also has a great community support. Instead of hiding underlying view graph within toolkit-specific abstraction, D3 enables direct assessment and operation of a native representation: Immediate evaluation of operators further simplifies debugging and allows iterative development.

D3 Features

D3.js is an open source project. It requires very less code and can be used to produce a variety of visualizations from simple to complex including user interaction and transition effects. Some of its salient features include:

- *Data Driven:* D3 is data driven. It can use static data or fetch it from the remote server in different formats such as Arrays, Objects, CSV, JSON, XML etc. to create different types of charts.
- *DOM Manipulation:* D3 allows manipulation of the Document Object Model (DOM) based on given data.
- *Data Driven Elements:* It authorize data to dynamically generate elements and apply styles to the elements, be it a table, a graph or any other HTML element and/or group of elements.

- *Dynamic Properties:* D3 gives the flexibility to provide dynamic properties to most of its functions. Properties can be specified as functions of data.
- *Types of visualization:* With D3, there are no standard visualization formats. But it enables creations from an HTML table to a Pie chart, from graphs and bar charts to geospatial maps.
- *Custom Visualizations:* Since D3 works with web standards, it offers complete control over the visualization features.
- *Transitions:* D3 provides the transition () function. This is quite powerful because internally, D3 works out the logic to interpolate between values and find the intermittent states.
- *Interaction and animation:* D3 provides great support for animation with functions like duration (), delay () and ease (). Animations from one state to another are fast and responsive to user interactions.

Sankey Diagram

Sankey diagram is a dataflow diagram, in which the size of the node is proportionally to the flow quantity. These diagrams put a visual emphasis on the major transfers or flows within a system. They are helpful in identifying the foremost contributions in an overall flow. Below is the example of Sankey diagram:

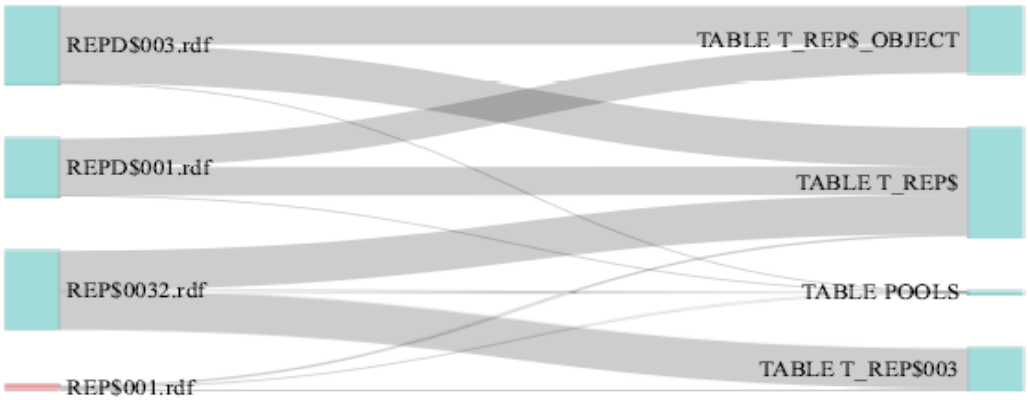

Figure 1. Sample Sankey Diagram

It gives overall view of the complete system and at the same time allows interactions to know how the nodes are connected in the visualization. Sankey diagrams can be drawn using D3 visualization.

D3 Sankey Implementation

In this section, I would like to explain firstly the frontend code part then the database or server part from where I fetched the data in the JSON format from the Oracle Database.

Frontend part mainly covers the HTML, CSS and Oracle APEX web interface. Oracle APEX stores the files in the static application files. The CSS part covers the stylesheets for the graph including the nodes and links and the d3-tip

API Structure

D3.js API contains several hundred functions, and they can be grouped into following logical units:

- Selections
- Transitions
- Arrays
- Math
- Color
- Scales
- SVG
- Time
- Layouts
- Geography
- Geometry
- Behaviors

D3 Sankey Plug-in

The concept of a plug-in is that it is a separate piece of code that allows further functionalities to a core block (in this case, d3.js). I used the minified version of the D3 version 3 library in the Oracle Apex environment ie., d3.v3.min.js

I defined two more JavaScript files, one for using the D3 Sankey plug-in and the other for doing customizations to modify the graph according to the requirements. I used the tool tip and zoom functionalities of D3 library in customizing the required Sankey graph.

The first part in the JavaScript code is to set the dimensions for the graph by setting canvas size and margins.

```
var margin = {top: 100, right: 100, bottom: 100, left: 100},
width =6000 - margin.left - margin.right,
height = 3500 - margin.top - margin.bottom;
```

before setting some formatting

```
var formatNumber = d3.format(",.0f"),
format = function(d) { return formatNumber(d) + " "+units; },
```

Our next block sites our canvas onto our page in relation to the size and margins we have already defined;

```
.svg = d3.select("#sankey").append("svg")
              .attr("viewBox", "0 0 " + width + " " + height)
              .attr("preserveAspectRatio", "none")
              .call(zoom).call(tipLinks).call(tipNodes)
              .append("g")
              .attr("transform", "translate(" + margin.left + "," +
              margin.top + ")");
```

Then we set the variables for our Sankey diagram

```
var sankey = d3sankey()
              .nodeWidth(25)
              .nodePadding(15)
              .size([width, height]);
```

Without trying to state the obvious, this sets the width of the nodes (.nodeWidth(25)), the padding between the nodes (.nodePadding(15)) and the size of the diagram(.size([width, height]);).

var path = sankey.link();

The above line defines the path variable as a pointer to the sankey function that make the links between the nodes to their clever thing of bending into the right places.

Once a dataset is bound to a document, use of D3.js typically follows a pattern wherein an explicit .enter() function, an implicit "update," and an explicit .exit() function is invoked for each item in the bound dataset. Any methods chained after the .enter () command will be called for each item in the dataset not already represented by a DOM node in the selection (the previous selectAll()). Likewise, the implicit update function is called on all existing selected nodes for which there is a corresponding item in the dataset, and .exit() is called on all existing selected nodes that do not have an item in the dataset to bind to them. To make the visualizations interactive, we use tools like zoom and tool tip or q-tip which helps in viewing the information related to the nodes and links.

Contact address:

Kantha Cikaiahgari
Pitss GmbH
Industrie Straße 3
70565 Stuttgart

Phone: +49 (0) 711/ 91 40 12-25
Email kcikaiahgari@pitss.com
Internet: www.pitss.de