D5.2 Guidelines for interacting and visualization information in Big Data environments

Lead Author: Ignacio García (TREE)
Reviewer: Iván Díaz (LMDP)

<table>
<thead>
<tr>
<th>Deliverable nature:</th>
<th>Report (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level: (Confidentiality)</td>
<td>Public (PU)</td>
</tr>
<tr>
<td>Contractual delivery date:</td>
<td>M12 (November 2016)</td>
</tr>
<tr>
<td>Actual delivery date:</td>
<td>M12 (November 2016)</td>
</tr>
<tr>
<td>Version:</td>
<td>1.2</td>
</tr>
<tr>
<td>Total number of pages:</td>
<td>37</td>
</tr>
<tr>
<td>Keywords:</td>
<td>Big data, interactive visualization, data visualization</td>
</tr>
</tbody>
</table>
Abstract

This deliverable specifies the technologies, data formats and protocols used to develop an innovative data visualization library focused on big data environments. In this report, we propose an innovative visualization-based solution to deal with the four Vs of big data: volume, velocity, variety and value. We also discuss about theoretical design features of data visualization, such as colour palettes, transition effects, aspect ratios and combinations of them. We finally propose a set of techniques to deal with real-time interactivity in data visualization.

All of the above described features will be developed and included into the next-generation and open-source visualization library developed under this project: Proteic.js\(^1\).

\(^1\) https://github.com/proteus-h2020/proteic
Executive summary

Data visualization is the presentation of data in a pictorial or graphical format. It is viewed by many disciplines as a modern equivalent of visual communication. It involves the creation and study of the visual representation of data. Its main goal is to communicate information clearly and efficiently via statistical graphics, plots and information graphics. These are basic graphical elements that each representation uses such as points, lines, shapes, images, text, and area, and there are attributes associated with these elements such as colour, intensity, size, position, shape and motion [1]. Data visualization is very useful for people to understand data in a graphical manner.

The process of data visualization is becoming an increasingly important component of analytics in the age of big data. In this era, huge amount data are continuously acquired for a variety of purposes. It is a huge challenge to visualize this growing data in static or in dynamic form, since most traditional data visualization tools cannot support at “big” scale [2]. Perceptual scalability, real-time scalability and interactive scalability are the main issues when dealing with big data and data visualization.

In this report, we will discuss about the technologies, data formats, protocols and techniques needed to achieve a real-time and interactive big data visualization when dealing with data streams. The conclusions obtained in this report will be translated and implemented into the next-generation and open-source visualization library developed under this project: Proteic.js.
This deliverable specifies the technologies, data formats and protocols used to develop an innovative data visualization library focused on big data environments. In this report, we propose an innovative visualization-based solution to deal with the four Vs of big data: volume, velocity, variety and value. We also discuss about theoretical design features of data visualization, such as colour palettes, transition effects, aspect ratios and combinations of them. We finally propose a set of techniques to deal with real-time interactivity in data visualization.

All of the above described features will be developed and included into the next-generation and open-source visualization library developed under this project: Proteic.js².

Keywords
Big data, interactive visualization, data visualization

---

² https://github.com/proteus-h2020/proteic
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>October, 15th, 2016</td>
<td>0.0.5</td>
<td>Jorge Yagüe</td>
<td>Protocols</td>
</tr>
<tr>
<td>October, 22th, 2016</td>
<td>0.0.6</td>
<td>Ignacio García</td>
<td>Data formats</td>
</tr>
<tr>
<td>October, 25th, 2016</td>
<td>0.0.7</td>
<td>Jorge Yagü</td>
<td>Conclusions</td>
</tr>
<tr>
<td>October, 31th, 2016</td>
<td>1.0</td>
<td>Ignacio García</td>
<td>Version for review</td>
</tr>
<tr>
<td>November, 21th, 2016</td>
<td>1.1</td>
<td>Iván Díaz</td>
<td>Reviewed version</td>
</tr>
<tr>
<td>November, 23th, 2016</td>
<td>1.2</td>
<td>Ignacio García</td>
<td>Final version</td>
</tr>
</tbody>
</table>
Table of Contents

Executive summary ............................................................................................................. 4
Document Information ........................................................................................................ 5
Table of Contents .................................................................................................................. 7
List of figures and/or list of tables ....................................................................................... 8
Abbreviations ....................................................................................................................... 9
Definitions ........................................................................................................................... 10
1 Introduction ....................................................................................................................... 11
   1.1 Current big data visualization challenges ................................................................. 12
2 ProteicJS: The PROTEUS visualization toolkit ............................................................... 13
   2.1 Data types .................................................................................................................. 13
      2.1.1 1-dimensional ................................................................................................. 13
      2.1.2 2-dimensional ................................................................................................. 14
      2.1.3 3-dimensional ................................................................................................. 14
      2.1.4 Multi-dimensional ......................................................................................... 15
      2.1.5 Temporal ......................................................................................................... 15
      2.1.6 Hierarchical ..................................................................................................... 16
      2.1.7 Network .......................................................................................................... 16
   2.2 Charts ......................................................................................................................... 16
      2.2.1 Specially built for data streams ....................................................................... 17
      2.2.2 General purpose charts .................................................................................. 18
   2.3 Protocols .................................................................................................................... 25
      2.3.1 Websockets ...................................................................................................... 25
      2.3.2 HTTP ............................................................................................................... 26
   2.4 Design ......................................................................................................................... 27
      2.4.1 Colour palettes ............................................................................................... 27
      2.4.2 Smooth transitions .......................................................................................... 28
3 Dealing with the four Vs of big data .............................................................................. 30
   3.1 Velocity ...................................................................................................................... 30
   3.2 Volume ......................................................................................................................... 31
   3.3 Variety ........................................................................................................................ 32
   3.4 Veracity ....................................................................................................................... 33
4 Conclusions ..................................................................................................................... 35
References ............................................................................................................................ 36
List of figures and/or list of tables

Figure 1. The four Vs of big data: volume, velocity, variety and veracity ........................................... 12
Figure 2. The official logo of Proteic.js .................................................................................................. 13
Figure 3. Streamgraph showing different variables over time .............................................................. 17
Figure 4. Swimlane that shows some events occurred over time ........................................................... 18
Figure 5 Linechart .................................................................................................................................. 18
Figure 6. Barchart .................................................................................................................................... 19
Figure 7. Pie chart ..................................................................................................................................... 19
Figure 8. Scatterplot ................................................................................................................................. 20
Figure 9. Timeline ..................................................................................................................................... 20
Figure 10. Treemap ................................................................................................................................ 21
Figure 11. Heatmap .................................................................................................................................. 21
Figure 12. Windrose ................................................................................................................................. 22
Figure 13. Radar plot ................................................................................................................................. 22
Figure 14. Dendogram .............................................................................................................................. 23
Figure 15. Gantt diagram ......................................................................................................................... 23
Figure 16. Geographic map ...................................................................................................................... 24
Figure 17. Network graph ......................................................................................................................... 24
Figure 18. Sankey diagram ....................................................................................................................... 25
Figure 19 Architecture of the websocket datasource ............................................................................. 25
Figure 20 integrating a websocket server into Apache Flink .................................................................. 26
Figure 21 Pipeline execution of Apache Flink ......................................................................................... 26
Figure 22. HTTP Datasource .................................................................................................................. 27
Figure 23. Example of categorical colour scales .................................................................................... 27
Figure 24. Example of sequential colour scales ....................................................................................... 28
Figure 25. Example of divergent colour scale ......................................................................................... 28
Figure 26. A transition between two different states ............................................................................... 29
Figure 27. HTTP vs Websockets ............................................................................................................ 30
Figure 28. A performance comparison between Canvas and SVG. Horizontal axis shows a set of objects to be render, and the vertical axis show the time needed by the different APIs on different browsers (lower is better) ............................................................................................... 31
Figure 29. Web workers .......................................................................................................................... 32
Figure 30. Common format for hierarchical data .................................................................................... 32
Figure 31. Time-series data format .......................................................................................................... 33
Figure 32. PROTEUS data format ........................................................................................................... 33
Figure 33. Region identification in a Linechart. ....................................................................................... 34
Abbreviations

API: Application programming interface
CSS: Cascading style sheet
ES6: EcmaScript 6
HTML: Hypertext markup language
ICT: Information and Communication Technologies
JS: Javascript
Definitions

**CANVAS**: Canvas (HTML5) allows for dynamic, scriptable rendering of 2D shapes and bitmap images. It is a low level, procedural model that updates a bitmap and does not have a built-in scene graph.

**FULL-DUPLEX**: a feature that allows communication in both directions and allows this to happen simultaneously.

**HTTP**: The Hypertext Transfer Protocol is an application protocol for distributed, collaborative and hypermedia information systems. It is the foundation of data communication for the World Wide Web (W3C).

**SVG**: scalable vector graphics is and XML-based vector image format for two-dimensional graphics with support for interactivity and animation.

**WEBSOCKET**: a computer communications protocol that provides full-duplex, communication channels over single TCP connection.
1 Introduction

The main objective of data visualization [3] is to represent knowledge more intuitively and effectively by using different graphs. To convey information easily by providing knowledge hidden in the complex and large-scale data sets, both aesthetic form and functionality are necessary. Information that has been abstracted in some schematic forms, in addition to attributes or variables, is also valuable for data analysis. This way is much more intuitive [4] than sophisticated approaches. For Big Data applications, it is particularly difficult to use visualization because of the large size and high dimensionality of data. However, current Big Data visualization tools suffer poor functional performance and lack scalability and efficiency in terms of response time. It is necessary to tackle these problems. Even, successful techniques for data-intensive applications such as history mechanisms proposed in [5] require more efficiency. Big datasets are ubiquitous in many domains, such as finance, discrete manufacturing, monitoring, internet, telecommunication, biology, sports [6]. It is not uncommon that millions of readings from high-frequency sensors are subsequently stored in relational database management systems (RDBMS), to be later accessed using visual data analysis tools. Modern data analysis tools must support a fluent and flexible use of visualizations and still be able to squeeze a billion records into a million pixels [6]. In this regard, one challenge for the scientific community is the development of compact data structures that support algorithms for rapid data filtering, aggregation, and display rendering. These issues are yet unsolved for existing RDBMS-based visual data analytics tools such as Tableau Desktop [7], SAP Lumira [8], QlikView [9], Tibco Spotfire [10] and Datawatch Desktop [11]. While they provide flexible and direct access to relational data sources, they do not consider an automatic, visualization-related data filtering or aggregation and are not able to quickly and easily visualize high-volume historical data. For example, they redundantly store copies of the raw data as tool-internal objects, requiring significant amounts of system memory. This causes long response time for the users and eventually indefinitely in case the system memory is exhausted and gets stuck. Apart of commercial solutions, a number of open-source visual toolkits exist (such as InfoVis Toolkit [12], Prefuse [13], Improvise [14] and D3 [15]); each covers a specific set of functionalities for visualization, analysis and interaction. Using existing toolkits instead of implementing new ones from scratch provides much efficiency [16], although the level of maintenance, development and user community support of open-source code can vary drastically. The major shortcoming of exiting tools, commercial and open-source, lies in the fact that they are dedicated to batch data (data-at-rest), not in data streams (data-in-motion). However, there exist some successful domain-specific tools such as ELVIS that is a highly interactive system to analyze system log data, but cannot be applied to real-time streams. SnortView [17] focuses on the intrusion detection, while Event Visualizer [18] provides real-time visualizations for event data streams for real-time monitoring as well as various exploration mechanisms. On the other hand, authors in [19] propose a real-time visualization system to enhance situational awareness from network traffic data using LiveRAC [20]. Once analysed and aggregated, time-series are displayed in a zoomable tabular interface to enable interactive exploration. Another tool which focuses on monitoring of time series data is VizTree [20], allows to visualize real-time anomaly detection after transforming the time series into symbols.

Compared to existing literature, the approach introduced in the present paper, aims to deal with (i) visualization of data streams and (ii) enabling real-time interaction with big data-in-motion.

To deal with these issues, we propose to build an innovative data visualization library specifically designed for visualizing both batch and streaming data, capable of addressing the previously identified scalability issues. Such a library is designed, implemented and integrated into D3.js [15]. This library will allow both expert as well as users (analysts) to explore big data (both data-at-rest and data-in-motion) faster to make well-informed decisions in time.
1.1 Current big data visualization challenges

Advanced visualization of data analytics in real-time, user experience and usability is still an open issue in the context of big data. The interactivity requirement creates special challenges when it comes to big data [21]. Interaction is a necessary condition for data analysis tasks, especially when using exploratory visual tools. However, most state-of-the-art tools or techniques do not properly accommodate big data.

Specifically, a key challenge of visual analytics is to meet the requirements of big data in supporting real-time interaction while considering the challenges of volume, velocity and variety. Despite the emerging advances to achieve low latency for ad-hoc queries, it is still necessary to rethink efficient software architecture styles to enable real-time interaction:

- **Volume**: refers to the amount of data. Visualizations are not ready to work with an immense number of datasets. Typically, existing visualization libraries and tools do not properly deal with the volume of data, since most of them get overloaded in streaming scenarios.
- **Variety**: data can be stored in multiple formats. Variety refers to the numbers of types of data. It is a challenge to standardize and optimize data formats to properly visualize information. Existing visualization libraries and tools are format-dependent, so they need specific data formats for visualizations. Users need to create a process to transform and adapt original data into the specific library format.
- **Velocity**: refers to the speed of data processing. Visualization library do not properly deal with the velocity of data stream, since many libraries suffer visualization delays.
- **Veracity**: refers to the value of data. Visualizations are commonly attractive enough, but they do not create business value by identifying data patterns or detecting data anomalies.

On the other hand, visualization of data streams is strongly related to its temporal context. Although the data being generated and delivered in the streams has a strong temporal component, in many cases it is not only the temporal component that the analysts are interested in. There are other important data dimensions (e.g. source, space, relevance, etc.) that are equally important and time might be just an additional aspect that they care about. Finally, the use of visualisation paradigms dedicated to machine learning and data analytics methods would help inspect the data as well as to explain the behaviour of the algorithms.

![Figure 1. The four Vs of big data: volume, velocity, variety and veracity.](image)

The rest of this report is structured as follows: Section 2 contains a full description about the PROTEUS visualization toolkit: data formats, catalogue of charts, protocols and connectors. Section 3 highlights those actions necessary to carry out when dealing with data visualization and data streams. Finally, section 4 concludes this report.
2 ProteicJS: The PROTEUS visualization toolkit

Proteic.js is an open-source web-based visualization library that aims to deal with the existing challenges of data visualization on big data [21] [22], by dealing with the volume, variety and velocity of data streams. This library also aims to provide a friendly API for developers by using the latest web standards and novel programming language specifications. It is also focused on good and responsive designs, since it is a key factor for understanding visual information and analytics.

Proteic.js will contribute to the state-of-the-art of data visualization, by providing novel techniques for visualizing data streams. These techniques are detailed in section 3.

Section 2.1 describes the different data types managed by this library. The full catalogue of data visualizations is included in section 2.2. They are separated into two main categories: general purpose charts and specially aimed ones for data streams. Sections 2.3 describe data protocols and connectors necessary to interactively visualize data streams. Finally, section 2.4 discuss about design features such as responsiveness, colour palettes and transition effects.

Figure 2. The official logo of Proteic.js

2.1 Data types

In this section we identify the most common existing data formats. We summarize and explain each of them and show how they can be easily transformed to the PROTEUS format, in order to achieve interoperability between data formats.

2.1.1 1-dimensional

This is the simplest type of data. It represents a linear sequence of ordered data items like an alphabetical list, a text or a number line. The following is an example of a series of values for a gauge chart, in the PROTEUS format:

```
[  
  { "datum": 34 },  
  { "datum": 35 },  
  { "datum": 36 },  
  { "datum": 35 }  
]
```
2.1.2 2-dimensional

This might be flat data like tables, matrices or planar geographical data. Examples of 2-dimensional data include a set of placemarks for a map, a two dimensional array containing data for a bar chart or a set of points for a scatterplot. The followings are examples of 2-dimensional data in the PROTEUS format:

```json
[
    { "x": 12, "y": 30 },
    { "x": 52, "y": 68 },
    { "x": 45, "y": 23 },
    { "x": 25, "y": 12 }
]
```
Table 1. Scatterplot or bar chart data

```json
[
    { "lon": -5.821, "lat": 43.422, "label": "placemark1" },
    { "lon": -5.820, "lat": 43.423, "label": "placemark2" },
    { "lon": -5.820, "lat": 43.422, "label": "placemark3" },
    { "lon": -5.819, "lat": 43.419, "label": "placemark4" }
]
```
Table 2. Geographical data

2.1.3 3-dimensional

Three-dimensional data can be used to represent real objects or geographical locations, like a 3D model in computer graphics or terrain data, but also data encoded with 3 variables, like a grouped barchart, a 3D scatterplot, or a multi-series linechart. The code below is an example of 3-dimensional data in the PROTEUS format:

```json
[
    { "lon": -5.821, "lat": 43.422, "lon": 36.5, "label": "placemark1" },
    { "lon": -5.820, "lat": 43.423, "lon": 36.3, "label": "placemark2" },
    { "lon": -5.820, "lat": 43.422, "lon": 37.2, "label": "placemark3" },
    { "lon": -5.819, "lat": 43.419, "lon": 36.0, "label": "placemark4" }
]
```
Table 3. Terrain data
2.1.4 Multi-dimensional

This kind of data is often used in statistical and scientific datasets. An example of a visualization using multi-dimensional data can be a 3D categorical scatterplot, using colours to encode the categories.

```json
Table 4. Grouped data

[{
    "key": "adults", "x": "ES", "y": 30,
},
{
    "key": "children", "x": "ES", "y": 49,
},
{
    "key": "adults", "x": "DE", "y": 23,
},
{
    "key": "children", "x": "DE", "y": 68,
}
]
```

2.1.5 Temporal

Temporal data is similar to 1-dimensional data but the items have a start and end date or timestamp, and they can overlap. Essentially, it is a list of events ordered by date.

Temporal data may have more than one dimension, since the events can be grouped into categories. Examples of temporal visualisations are Gantt charts, Marey diagrams, streamgraphs or timelines. Notice that other time-based visualizations may not necessarily use this data format, for example, time series use 2-dimensional data, since the events in this case are instantaneous they don’t need a start and end date.

```json
Table 5. Multi-dimensional data

[{
    "x": 12, "y": 30, "z": 32, "key": "group1",
},
{
    "x": 52, "y": 68, "z": 43, "key": "group1",
},
{
    "x": 45, "y": 23, "z": 97, "key": "group2",
},
{
    "x": 25, "y": 12, "z": 26, "key": "group2",
}
]
```

```json
[{
    "id": "task1", "start": "2016-10-18T09:00:00Z", "end": "2016-10-18T12:00:00Z", "category": "Development" },
{
    "id": "task2", "start": "2016-10-18T12:00:00Z", "end": "2016-10-18T14:00:00Z", "category": "Documentation" },
{
    "id": "task3", "start": "2016-10-18T16:00:00Z", "end": "2016-10-18T21:00:00Z", "category": "Development" }
]
```
Table 6. Temporal data

2.1.6 Hierarchical

In hierarchical data, also known as trees, all the individual items are linked to one parent item except the root node. Each of the items can also be parent of some child nodes. This kind of data is used in visualizations such as tree maps, sunburst diagrams or organizational charts. The following is an example of data from a directory structure:

```json
[
    { "id": "/", "parent": "," , "value": 150, "label": "/ 150Kb" },
    { "id": "home", "parent": "/", "value": 100, "label": "/home 100Kb" },
    { "id": "usr", "parent": "/", "value": 50, "label": "/usr 50Kb" },
    { "id": "bin", "parent": "usr", "value": 50, "label": "/usr/bin 50Kb" }
]
```

Table 7. Hierarchical data

2.1.7 Network

Network or graph data is used to encode arbitrary relationships between items. This kind of data is mostly used to generate network graphs, although the look of the diagram can vary heavily between distinct libraries. The following code is an example of network data, mixing directed and undirected links.

```json
[
    { "id": "a", "label": "A" },
    { "id": "b", "label": "B" },
    { "id": "c", "label": "C" },
    { "id": "ab", "source": "a", "target": "b", "label": "AB" },
    { "id": "bc", "source": "b", "target": "c", "label": "BC", "type": "directed" }
]
```

Table 8. Network data

2.2 Charts

Charts are a graphical representation of data, in which data is represented by symbols such as bars, lines, slices, points, etc. Charts can represent tabular numeric data, functions or any kind of
qualitative structure or unstructured data. Next subsections describe a catalogue of charts to be implemented and included in Proteic.js. They all will be available for both data streams and batch data.

2.2.1 Specially built for data streams

Streamgraph
A streamgraph is a type of stacked area graph which results in a flowing and organic shape. They were popularized by Lee Byron [23], when he introduced them in an article on the New York Times journal.

Streamgraphs are always displaced around a temporal axis, representing how data evolves over time. A streamgraph is shaped by grouping and stacking values that belong to a specific category. Categories are stacked one above the others.

![Streamgraph showing different variables over time.](image)

**Figure 3.** Streamgraph showing different variables over time.

Swimlane Timeline
A swimlane timeline is a type of visualization that can display one or more data series of ordered data over time. The position of each datum in the chart is given by its time interval and the category that the datum belongs to. Every datum is represented as a box, where its width is defined by the duration of the event.
2.2.2 General purpose charts

Line chart
Linecharts can display one or more data series of ordered data points linked by straight or curved line segments. The position of the data points is determined by the value of its X and Y variables, representing its cartesian coordinates. The Y axis of the chart will always have a continuous, quantitative scale, while the X axis can also use a categorical scale like a timeline.

For easier identification of the variables, each data series is identified with a different colour and data points can be represented as markers that trigger events when the user interacts with them, like showing a tooltip or highlighting the marker. The chart can also display an optional legend.

Linecharts are often used to show relationships and trends between different variables.
A bar chart uses bars of different heights to represent numerical variables grouped in different categories to show comparisons between categories. Barcharts have two axis with different scales: a categorical axis representing the different categories and a quantitative scale that can be either continuous or discrete. The bars and the categorical axis can either be horizontal or vertical (column chart), and the length of the bar represents the value of the variable.

Each of the categories may have one or more variables that can be represented as different bars per category, or as a single bar with stacked segments. The segments or bars of the same category will have the same colour, and the library provides a legend for its identification. These elements can also trigger events with user interactions.

![Figure 6. Barchart](image)

**Pie chart**

Pie charts are circular charts divided into sectors that represent various categories in the data. The proportion of the circle taken by each sector is equivalent to the value of each category, and each sector is assigned a different colour that identifies its category.

These charts are useful to represent proportional categories and give a quick overlook of the data distribution. However, it is not the proper chart to represent lots of categories, categories with very small proportions, or changes in proportions over time.

![Figure 7. Pie chart](image)
Scatter plots are very similar to Linecharts, since they use horizontal and vertical axes to plot data points. However, they have a very specific purpose: scatter plots show how much one variable is affected by another. The relationship between two variables is called their correlation.

Scatter plots usually consist of a large body of data. The closer the data points come when plotted to making a straight line, the higher the correlation between the two variables, or the stronger the relationship. If the data points make a straight line going from the origin out to high x- and y-values, then the variables are said to have a positive correlation. If the line goes from a high-value on the y-axis down to a high-value on the x-axis, the variables have a negative correlation.

**Figure 8. Scatterplot**

**Timeline**

A timeline comprises a temporal axis, displayed as a long bar, where a sequence of events in chronological order are shown as markers over the axis. The temporal axis can use different types of scales, such as linear or logarithmic to suit the distribution of the events.

These charts can be combined with other visualization methods to highlight changes in the data over time.

**Figure 9. Timeline**

**Treemap**

Treemaps can display hierarchical data, specifically a tree structure, as a set of nested rectangles with its area proportional to the sum of the values of its correspondent node and its children. The size and position of the rectangles is determined by a tiling algorithm and the color can represent categories, its level in the hierarchy or any other variable.
Treemaps are useful to display hierarchies in a more compact way than a tree and quickly assess the size of each node.

![Figure 10. Treemap](image)

**Heatmap**
A heatmap colorizes the values contained in a 2-dimensional matrix using a scale. The result is a table in which the header row and columns display the names of the variables and the rest of the cells have a colour that represents the value of the original numeric matrix.

Heatmaps are useful to identify correlations and perform cluster analysis.

![Figure 11. Heatmap](image)

**Wind rose**
The wind rose is a kind of chart used in meteorology to display how wind speed and direction are distributed at a certain location. This diagrams show bars of different sizes and colours around a central axis. Where the size and colours of the bars represents the frequency and speed of the wind and the direction of the wind is shown by the position of the bar in the circle.
Figure 12. Windrose

**Radar chart**
Radar charts are circular charts that display three or more quantitative variables in axes that radiate from a central point. The values of each variable in the axis are connected by straight line segments, forming a closed shape that can be used to quickly discover certain qualities of the data.

![Radar chart](image)

Figure 13. Radar plot

**Dendrogram**
Dendrograms are tree diagrams used to visualize the arrangement of clusters produced by hierarchical clustering.
Gantt charts are used to visualize the schedule of a project as bars in a grid with two axis. The horizontal axis is temporal and represents the time span of the project, while the vertical axis is categorical, where the categories are the tasks of the project. Bars placed on the grid represent the duration and timing of each task.

This kind of charts helps the user to visualize the sequence of task and its dependencies to ease the project planning and management.
Geomap
A geomap is a map of a country, continent, or region map, with colours and values assigned to specific regions. We can differentiate two different classes of geomap:

- Colour scale based maps: values are displayed as a colour scale. Depending on a specific value, the colour displayed will be more or less intense.
- Icon-based map: values are displayed through icons. Depending on a specific data attribute, the displayed icon could different.

![Geomap Example](image)

**Figure 16. Geographic map**

Network diagram
A network diagram is a representation of the nodes and edges of a graph. The nodes of the graph are displayed as circles, rectangles or any other geometric shape, while the edges are shown as lines linking the nodes (or arrows if the graph is directed). The layout of this elements in the graph is determined by an algorithm.

![Network Diagram Example](image)

**Figure 17. Network graph**

Sankey
Sankey diagrams are a specific type of flow diagram, in which the width of the arrows is shown proportionally to the flow quantity. They are typically used to visualize energy or material or cost transfers between processes.
In this kind of diagrams, the things being connected are called nodes, and the connections are called links. This visualization fits very well when you need to show a many to many mapping between two entities, by representing the entities as nodes and the flow as links.

![Sankey diagram](image)

Figure 18. Sankey diagram

### 2.3 Protocols

Proteic.js uses different protocols in order to achieve system interoperability. ProteicJS is a visualization library able to run over both big data environment and small data ones.

#### 2.3.1 Websockets

Websocket is a technology, based on the websocket protocol ws [24], that makes it possible to establish a continuous full-duplex connection stream between a client and a server. Although the ws protocol is platform independent, clients are typically based on web browsers.

![WebSocket architecture](image)

Figure 19 Architecture of the websocket datasource.

Proteic.js provides different ways to retrieve data: this library takes advantage of the concept of datasource. It enables developers to easily access data from different sources. In particular, Proteic.js provides a datasource that retrieve data from a websocket endpoint, by simply specifying a host and a specific port where this server is running. After connecting to this server, a full-duplex communication is established between a client (typically a web-browser) and the server. At this point, the server is able to constantly send new messages without needing a previous request from the client. Every time a new message is received, Proteis.js automatically calls to the `keepDrawing()` method that is in charge of replotting the visualizations with the new data that is been received. Figure 19 summarizes this continuous process.
In order to provide a full-duplex communication between Apache Flink and web-browsers, two steps are needed:

- **Include a Websocket server inside Apache Flink**: Flink is a platform for processing distributed stream and batch data. It lacks the presence of an HTTP server, since it is not needed at all. However, since we need a bidirectional communication between Flink and web browsers to be able to real-time transmit data across the network, we have included a lightweight websocket server inside Apache Flink.

  ![Figure 20 integrating a websocket server into Apache Flink](image)

- **Develop a new Apache Flink Sink**: Sinks consume data and are used to store or return them. They are used in the last stage of the pipeline execution of Apache Flink (see Figure 21), when data have already been processed and it needs to be sent somewhere. Apache Flink provides a wide range of sinks, which have many different goals, such as writing data as csv or text.

  ![Figure 21 Pipeline execution of Apache Flink](image)

2.3.2 **HTTP**

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Proteic.js provides a HTTP datasource that makes it possible to retrieve information from any http endpoint. This datasource is designed for easing data access, since the most of the existing web-based visualization libraries do not provide any data access connector. Furthermore, this connector provides an API that allows developers to transform incoming data into the PROTEUS format, as Figure 22 depicts.
2.4  Design

2.4.1  Colour palettes

One of the most important points to make data readable and understandable for humans is the colour used to represent information. If we are dealing with a categorical (also known as “keyed”) visualization, the fact of representing each data record with a different colour help us to easily identify the category that this record belongs to. The same occurs when we are visualizing points with different quantitative values. By representing each point with more or less intensity, we can quickly identify the value of records.

This section summarizes the colour palettes that are available in Proteic.js.

Categorical

These kinds of colour palettes are appropriate to represent qualitative data such as category names without an obvious order. Categorical palettes map a set of category names as the domain values to a range of colours that identify each of the categories. The categorical palettes provided in ProteicJS have been designed so that colours are easy to distinguish, allowing a quick identification of the categories by the user, while providing a consistent look for its inclusion into existing designs.

Sequential

Sequential colour palettes are designed to represent quantitative data by dividing the domain values in different ranges and mapping each range to a different colour. The set of colours are ordered by one of the appearance parameters of the colour, like hue, brightness or saturation, meaning that the colours of the palette progress from low to high according to these parameters so that low data values correspond to low colour values.
Divergent palettes are a kind of sequential palettes that are appropriate to encode data values ranging from negative to positive values. In this palettes, the sets of colours grade from one saturated colour to another, passing by an unsaturated colour in the middle. These palettes can emphasize the extreme values of the data while showing the intermediate values.

Accessible palettes

Up to 8% of males and 0.5% of females, depending on the country, are affected by some form of colour vision deficiency. This condition could lead some users of ProteicJS to erroneous conclusions about the data they are visualising. To prevent this, we provide several versions of the colour palettes that are suitable for colour blind users with any form of colour vision deficiency (deuteranopia, protanopia and tritanopia ³).

2.4.2 Smooth transitions

Transitions occur when data change. They are defined as movements, passages or changes from one position to another. Transitions are in charge of reflecting changes in data.

When dealing with the four Vs of big data (volume, velocity and variety), existing visualization libraries lack a good transition procedure. Generally, if we are dealing with high frequency data, transition duration overcomes data generation ratio, which will produce delays in data visualization.

To deal with such an issue, Proteic.js provides a set of transitions that are smooth and fast enough to deal with the high frequency of data generation.

Figure 26. A transition between two different states.
3 Dealing with the four Vs of big data

As stated before, one of the main challenges related to data visualization is to deal with the four Vs of big data: volume, velocity, variety and veracity. Next subsections summarize a set of techniques to efficiently deal with these problems. These techniques and procedures are available in Proteic.js.

3.1 Velocity

Velocity is an issue when dealing with big data and real time processing. It refers to the speed of data processing. Data is often available in real-time, generated in a continuous and “infinity” process. Sometimes occurs that time required by visualization libraries to draw (render) data is higher than the time taken to generate the datum. In a real-time and continuous process, it could result in a delay in visualization. The list below provides useful techniques to deal with this problem:

- **Websockets**: Websocket is a computer communications protocol that provides full-duplex communication channels over a single TCP connection and is designed to be implemented in web browsers and web servers. Before Websocket all communication between web clients and servers relied only on HTTP. Now, dynamic data can flow over Websocket connections that are persistent. Also, they are faster than HTTP connections, since opening and closing connections for every request is very slow. This protocol is currently supported in most major browsers, such as Google Chrome, Microsoft Edge, Internet Explorer, Firefox, Safari and Opera. Figure 27 shows that the HTTP overhead increases with the number of messages. Websocket keeps a linear scalability when the number of messages increases.

![Figure 27. HTTP vs Websockets](image)

Proteic.js allows developers to easily configure a Websocket Datasource, instead of obtaining data from other less efficient protocols like http endpoints.

- **Decreasing rendering time**: a very simple but strategic measure to decrease the rendering time is to minimize the transition time. Transitions typically occur when new data is appended to a specific chart. In Proteic.js, transition times are totally configurable. By decreasing the transition time, it will make rendering faster. Section 2.4.2 provides more information about transitions.
3.2 Volume

Another important issue to tackle is the volume of data. Since Proteic.js uses D3.js as a dependency to low-level manipulate the DOM, it renders data in SVG format. For each data point, D3.js adds an element into the SVG container (commonly circles, rectangles or paths). Each of these elements takes resources in the web browsers. If we drastically increase the number of data records, it could result in a memory leak: data points are typically binded to the DOM\(^4\). Thus, the more elements are rendered, the more is the memory taken by the browser.

For that reason, we have defined a set of actions that help Proteic.js to render data without leaking memory, even if we are dealing with a huge volume of data streams:

- **Remove non-essential elements**: By removing non-essential elements (legends, decorative components, useless event handlers, etc.) it drastically decreases the size of the SVG document and the memory used by the web browser.

- **Isomorphism**: The isomorphic concept allows to use a library both in the client and the server side. Proteic.js is an isomorphic library. The use of the server to render charts is very useful when dealing with a huge amount of data. Hardware resources are typically better in servers than clients (web browsers).

- **Use the Canvas API**: Canvas is an HTML element which can be used to draw graphics using scripting (in this case, Javascript). This can be used to draw graphics, make photo composition or simple animations. There are some performance differences about using SVG or Canvas. Since SVG is a vector-based and scalable format that relies on the DOM to draw components, Canvas manipulates pixels and is considered as a simple graphic API. Thus, there is not way to alter existing drawings or react to events. In consequence, Canvas is faster than SVG and requires less time to render graphics, as depicts Figure 28.

![Figure 28](image.png)

Figure 28. A performance comparison between Canvas and SVG. Horizontal axis shows a set of objects to be renders, and the vertical axis show the time needed by the different APIs on different browsers (lower is better).

- **Webworkers**: Web workers are a mechanism by which a script operation can be made to run in a background thread separate from the main execution thread of a

\(^4\) https://www.w3.org/TR/WD-DOM/introduction.html
web application. The advantage of that laborious processing can be performed in a separate thread, allowing the main (usually UI) to run without being blocked or slowed down. Webworkers are very useful in those scenarios in which we need to render a huge amount of data. Typically, all rendering operation are executed in the main and unique Javascript thread. By allowing the library to render data using a separate thread, it could result in a more efficient way to visualize data without blocking the UI until the rendering process ends. Figure 29 depicts how Webworkers work: a task is sent from the main thread to a separate one. After finalizing the task execution, the thread returns a message (usually containing data) with the results of the task execution. By using Webworkers it could decrease the overload in the main execution thread.

![Figure 29. Web workers](image)

### 3.3 Variety

Variety refers to the many sources and types of data, both structured and unstructured. Data comes in the form of emails, photos, videos, PDFs, sensors, etc. This variety of unstructured data creates problems for storage, mining, analysing and visualizing data.

Each data type has an own data format by default. For example, Figure 30 shows a common format for representing hierarchical data. It is a tedious, prone to error and heavy format.

![Figure 30. Common format for hierarchical data](image)

The time-series format is another example of a complex format. It consists on an array, that contains elements, that contains arrays. Figure 31 shows an example of this format.
In Protec.js, all data formats are reduced to a single array containing elements. This format is very simple, but also very powerful for both batch and data streams. Also, the complexity for looping data with this format (O(n)) is lower than others that contains arrays into arrays. Each of the elements contained into the array can has different attributes (depending on the chart) and are represented as a data point into a specific chart.

Figure 32 shows an example of the PROTEUS format for some types of data: time-series, hierarchical and temporal.

3.4 Veracity

Generating significant value from data is critical in a business. All that available data will create a lot of value for organizations, societies and consumers. Big data means business and every industry will reap the benefits from big data. Data in itself is not valuable at all. The value is in the analyses done on that data and how the data is turned into information and eventually turning it into knowledge- The value is in how organizations will use that data and turn their organizations into an information centric company that relies on insights derived from data analyses for their decision making. Existing data visualization toolkits do not allow detecting patterns or regions in data.
Proteic.js will provide an API to easily allow users to show particularities in data (regions, data anomalies, etc.). Figure 33 shows an example of a region of interest in a line chart.

![Region identification in a Linechart](image)

Figure 33. Region identification in a Linechart.
4 Conclusions

Visualization of high-volume data, both static (data-at-rest) and in real time (data-in-motion), still presents several challenges related to the performance and usability of the tools [2]. Generating a visualization to display billions of records requires the use of filtering and aggregation techniques that may require large amounts of memory and processing [25]. This problem is even more relevant when the visualization is displayed in a web browser with limited resources. Aside from the performance issues, the purpose of visualizations is to provide the user with insights into the data, but the display of large amounts of information or rapidly changing data can hinder its comprehension by the users.

Another defect found in available visualization tools is extensibility and maintainability. The chosen tool would need to cope with the problems derived from big data visualization while providing enough extensibility for us to develop new visualizations and maintainability to ensure future developments. This means that the tool should be maintained and open-sourced.

Some of the current visualization tools are capable to cope with some of the considered issues. However, there is not any open-source tool flexible enough to be able to work with both data-at-rest and data-in-motion that perform well in web environments while providing the needed usability and exploration techniques for high-volume data.

The development of a novel open-source visualization library for the web is therefore needed to assess all of the issues mentioned previously. The developed library enables the creation of multiple kinds of visualizations for the web, for both static and real-time data, and provides a series of features to aid the exploration of high volumes of data and its compatibility with Big Data environments.

Proteic.js offers a wide catalogue of visualizations that we have classified in seven categories according to the nature of the data: 1-dimensional or linear, 2-dimensional, 3-dimensional, multidimensional, temporal, hierarchical and network. We have already developed several charts and will provide more in the future.

To ensure extensibility and maintainability of future visualizations, Proteic.js is being developed with a modular architecture using ECMAScript 6 modules [26], allowing us to reuse common components between charts.

We have also studied the most common data formats used in other tools for each of the data categories and defined the PROTEUS data format: a common data format suitable for all of the seven categories of data. This format has been designed to be efficient for both static and streaming data with extensibility in mind.

To achieve integration in Big Data environments, Proteic.js provides a series of native data connectors. These are modules used to receive streams of data from the backend in various streaming protocols. These connectors have also been designed to be easily extended in the future.

As stated before, usability is a key concern in data visualization software that improves insight into the data. Proteic.js will implement smooth transitions and interactions into the visualizations, so the user can follow the stream of data more easily, and a set of colour palettes designed to improve differentiation between variables; some of them suitable for colour-blind users. Another usability feature included in Proteic.js is the ability to describe events or alerts in the data, like a variable exceeding certain value, which will be displayed in the visualization as highlighted regions or other methods defined by the user.
References


