RespVis:
A Low-Level Component-Based Framework for Creating Responsive SVG Charts

Peter Oberrauner

RespVis:
A Low-Level Component-Based Framework for Creating Responsive SVG Charts

Peter Oberrauner B.Sc.

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 22 Jan 2021

RespVis:
Ein Low-Level Komponenten-Basiertes Framework zum Erstellen von Responsiven SVG Diagrammen

Peter Oberrauner B.Sc.

Masterarbeit

für den akademischen Grad

Diplom-Ingenieur

Masterstudium: Software Engineering and Management

an der

Technischen Universität Graz

Begutachter

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 22 Jan 2021

Diese Arbeit ist in englischer Sprache verfasst.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The document uploaded to TUGRAZonline is identical to the present thesis.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Dokument ist mit der vorliegenden Arbeit identisch.

Date/Datum

Signature/Unterschrift

Abstract

[TODO: Write Abstract]

keywords:

- responsive, visualisation, component-based, low-level, framework
- bar chart, line chart, scatterplot, ... [parcoord]
- JavaScript, TypeScript, D3
- SVG, Canvas, WebGL
- Node, gulp, rollup

Kurzfassung

[TODO: Translate abstract into german]

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

[TODO: Write acknoledgements]

Peter Oberrauner
Graz, Austria, 22 Jan 2021

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews 2019].

[TODO: Add further credits?]

Chapter 1

Introduction

This thesis introduces RespVis, a component-based framework for creating responsive SVG charts which is built on standard browser technologies like HTML, SVG and JavaScript.

[TODO: Outline the various chapters]

Chapter 2

Web Technologies

2.1 HyperText Markup Language (HTML)

HTML is a document markup language for documents that are meant to be displayed in web browsers. The original proposal and implementation in 1989 came from Tim Berners-Lee who was a contractor at CERN at the time [Berners-Lee 1989]. Over the years, the standard has been developed by a range of different entities like the CERN and the Internet Engineering Task Force (IETF). Today, HTML exists as a continuously evolving living standard without specific version releases that is maintained by the Web Hypertext Application Technology Working Group (WHATWG) and the World Wide Web Consortium (W3C) [WHATWG, W3C 2021].

The primary purpose of HTML is to define the content and structure of web pages. This is achieved with the help of HTML elements, which are composed in a hierarchical tree structure and define modular pieces of content that can be interpreted by web browsers. An example of a basic HTML page can be seen in Figure 2.1.

A strong pillar of HTML's design is extensibility. There are multiple mechanisms in place to ensure applicability to a vast range of use cases. These mechanisms include:

- Specifying classes of elements using the `class` attribute. This effectively creates custom elements while still basing them on the most related, already existing elements.
- Using `data-*` attributes to decorate elements with additional data that can be used by scripts. The HTML standard guarantees that these attributes are ignored by browsers.
- Embedding custom data using `<script type="">` elements that can be accessed by scripts.

Figure 2.1: HTML pages are structured as a hierarchical tree of elements which enables the composition of complex structures. [Image drawn by the author of this thesis.]

2.2 Cascading Style Sheets (CSS)

This section is not meant as a comprehensive guide to CSS but to give an overview and reiterate over the concepts that are important in the context of this thesis. In particular, the history of CSS is briefly summarized, a refresher on selectors and cascades is given and the different modules of CSS-based layouting are compared.

Cascading Style Sheets (CSS) is a style sheet language that is used to specify the presentation of HTML documents. It can either be embedded directly in HTML documents using `<style>` elements or it can be defined externally and linked into them using `<link>` elements. This characteristic of being able to externally describe the presentation of documents yields a lot of flexibility because multiple documents with different content can reuse the same presentation by linking to the same CSS file. This solved the problem of having to individually define the presentation of every page with presentation elements like `<strong>` or `<em>`, as was the case in earlier versions of HTML [WHATWG, W3C 1997].

CSS was initially proposed by Håkon Wium Lie in 1994 [Lie 1994] and standardized into CSS1 by the W3C in 1996 [W3C 1996]. Throughout its history, the adoption of CSS by browser vendors was fraught with complications and even though most major browsers soon supported almost the full CSS standard, their implementations sometimes behaved vastly different than their competitor's. This meant that authors of web pages usually had to resort to workarounds and provide different style sheets for different browsers. In recent years, CSS specifications have become much more detailed [W3C 2011] and browser implementations have become more stable with less inconsistencies. It has therefore become much rarer that browser-specific workarounds have to be applied, which drastically improves the development experience.

A CSS style sheet contains a collection of rules and each rule consists of a selector and a block of style declarations. Selectors are defined in a custom syntax and are used to match HTML elements. All elements that are matched by the selector of a rule will have the rule's style declarations applied on them. The selector syntax is fairly simple when merely selecting elements of a certain type but it also provides

Pattern	Matches
*	Any element.
E	Elements of type E.
E F	Any element of type F that is a descendant of elements of type E.
E > F	Any element of type F that is a direct descendant of elements of type E.
E + F	Any element of type F that is a directly preceded by a sibling element of type E.
E:P	Elements of type E that also have the pseudo class P.
.C	Elements that have the class C.
#I	Elements that have the id I.
[A]	Elements that have an attribute A.
[A=V]	Elements that have an attribute A with a value of V.
S1, S2	Elements that match either the selector S1 or the selector S2.

Table 2.1: A summary of the CSS 2.1 selector syntax. [Table created by the author of this thesis with data from [W3C 2011]]

the means for selecting elements based on their contexts or attributes. For a summary of the CSS 2.1 selector syntax, see Table 2.1.

Another important characteristic of CSS is the cascading of styles. There's a lot of depth to how the final style of an element is calculated and W3C 2011 should be consulted for detailed notes on this topic. The most important thing to state in the context of this work, is that styles can be overwritten. When multiple rules match an element and define different values for the same style property, the values of the rule with higher specificity will be applied. If multiple rules have the same specificity, the one defined last in the document tree will overwrite all previous ones.

2.2.1 Box Layout

All elements in a HTML document are laid out as boxes, which is defined as the CSS box model. The box model states that every element is wrapped in a rectangular box and every box is described by its content and the optional surrounding margin, border and padding areas. Margins are used to specify the invisible spacing between boxes, whereas the border is meant as a visible containment around the content of a box and the padding describes the invisible spacing between the content and the border. A visual representation of these concepts can be seen in Figure 2.2.

In early versions of CSS, before the introduction of the Flexible Box Layout (Flexbox) Module [W3C 2009], the box model was the only way to lay out elements. Style sheet authors had to meticulously define margins of elements and their relative (or absolute) positions in the document tree. The responsive capabilities of this kind of layouting are very limited because different configurations for varying screen sizes have to be done manually using media queries and more complex features, like filling the remaining space available, had to be implemented via scripting.

2.2.2 Flexible Box Layout (Flexbox)

Even though the first draft of the Flexbox Layout Module was already published in 2009 [W3C 2009], browser implementations have been a slow and bug ridden process [Use 2021] that held back adoption by users for the first couple of years after its inception. Over the past few years, partly through the deprecation of Internet Explorer [Microsoft 2020], all major browser implementations of current Flexbox standards [W3C 2018] have become stable and, in most cases, fallback styling is not necessary anymore.

Flexbox is a mechanism for one-dimensionally laying out elements in either rows or columns. This one-dimensionality is what separates it from grid-based layouting, which is inherently two-dimensional.

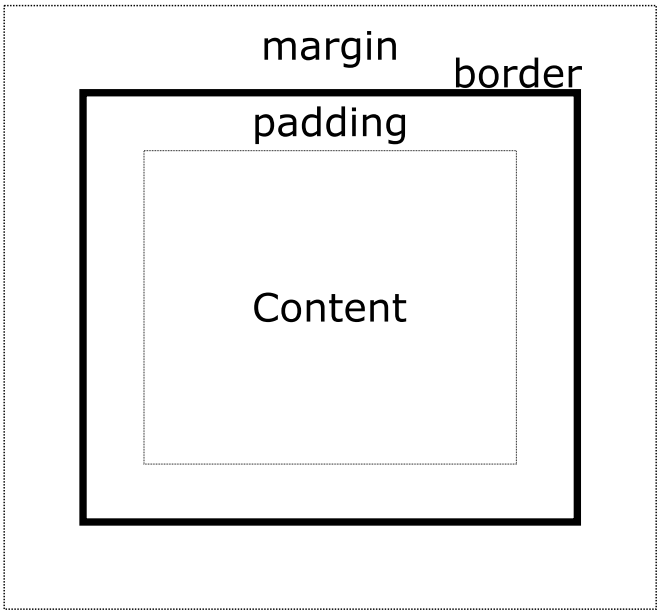

Figure 2.2: The CSS box model is used to define the properties of boxes that wrap around HTML elements. [Image drawn by the author of this thesis.]

Figure 2.3: The `justify-content` property is used to distribute items along the main axis of a Flexbox container. [Image created by the author of this thesis.]

Flexbox layouting can be enabled for child elements via setting the `display: flex` property on a container element. The direction of the layout can then be specified using the `flex-direction` property which can be set to either `row` or `column`.

The items inside a Flexbox container can have either a fixed or a relative size. When items should be sized relative to the size of their containers, the proportions of how the available space should be divided can be controlled using ratios. These ratios can be set on item elements via the `flex` property.

Another important feature of Flexbox layouting is the controllable spacing of items which can be specified separately for both the main and the cross axis of the layout. Spacing along the main axis can be configured with the `justify-content` property which can take a number of different values that are demonstrated in Figure 2.3. The property that controls alignment of items on the cross axis is either the `align-items` property on container elements or the `align-self` property on the items themselves.

This section only graced the surface of what is possible with the Flexbox Layout Module. Those properties that were discussed, were only discussed in a very broad sense and there are many useful

properties like `flex-grow`, `flex-shrink` or `flex-wrap` that weren't included in this overview. For a more detailed look at this topic it is recommended to review W3C 2018.

2.2.3 Grid Layout

2.3 JavaScript (JS)

2.4 TypeScript (TS)

2.5 Web Graphics

2.5.1 Raster Images

[TODO: Describe raster images]

[TODO: Mention JPEG]

[TODO: Mention PNG]

2.5.2 Scalable Vector Graphics (SVG)

[TODO: Define detail in which to write about SVG]

[TODO: Describe SVG]

[TODO: Describe filters]

[TODO: Talk about issues with using CSS for styling]

SVG elements can be styled with CSS which is highly convenient as it brings all the benefits of CSS like allowing users to override parts of the styling in their own style sheets. SVG styles defined as CSS can also be animated using CSS animations and transitions. This is recommendable to manual animations using JavaScript because external configuration is inherently supported and the declarative syntax of CSS animations is powerful enough to define complex animations.

2.5.3 Canvas

2.5.4 WebGL

2.6 Layout Engines

2.6.1 Yoga Layout

2.6.2 FaberJS

2.7 Visualization Libraries

2.7.1 Chartist

2.7.2 Highcharts

2.7.3 ECharts

2.7.4 ...?

2.7.5 D3

[TODO: Mention that D3 is successor of Protovis]

2.8 Tools

2.8.1 Node

2.8.2 Rollup

2.8.3 Gulp

Chapter 3

Responsive Information Visualization

[TODO: Find literature]

3.1 Information Visualization

four basic types of data

- Binary
- Qualitative
- Diverging
- Sequential

3.2 Responsive Design

[TODO: Define 'Responsiveness']

3.3 Responsive Visualization Patterns

Chapter 4

Software Architecture

[TODO: Add software architecture diagram]

[TODO: Describe relationship to D3]

[TODO: Describe storing data on elements]

[TODO: Describe using DOM events for callbacks]

[TODO: Describe components]

4.1 Primitives

4.1.1 Text

4.1.2 Rectangle

4.1.3 Circle

4.2 Series

[TODO: Describe series extension mechanism (enter/update/exit events)]

4.2.1 Bar Series

4.2.2 Grouped Bar Series

4.2.3 Stacked Bar Series

4.2.4 Point Series

4.2.5 Line Series

4.3 Charts

4.3.1 Bar Chart

4.3.2 Grouped Bar Chart

4.3.3 Stacked Bar Chart

4.3.4 Point Chart

4.3.5 Line Chart

4.4 Chart Windows

4.4.1 Bar Chart Window

4.4.2 Grouped Bar Chart Window

4.4.3 Stacked Bar Chart Window

4.4.4 Point Chart Window

4.4.5 Line Chart Window

4.5 Components

4.5.1 Lifecycle

events

- updating on data change
- updating on bounds change

4.6 Layouter

Chapter 5

Layouter

5.1 CSS Layouting

Chapter 6

Giving a Presentation

Chapter 7

Technical Realisation

Chapter 8

Selected Details of the Implementation

8.1 D3 Select Function Data Modification

Chapter 9

Outlook and Future Work

9.1 Outlook

9.2 Ideas for Future Work

9.2.1 Relative Positioning of Series Items

[TODO: Write about plans to use relative units (%) to position series items which would most likely get rid of the need to update components on bound changes]

Chapter 10

Concluding Remarks

Appendix A

User Guide

Appendix B

Developer Guide

Bibliography

Andrews, Keith [2019]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 24 Jan 2019. http://ftp.iicm.edu/pub/keith/thesis/ (cited on page xiii).

Berners-Lee, Tim [1989]. *Information management: A Proposal*. 1989. http://www.w3.org/History/1989/proposal.html (cited on page 3).

Lie, Håkon Wium [1994]. *Cacading HTML Style Sheets: A Proposal*. 1994. https://www.w3.org/People/howcome/p/cascade.html (cited on page 4).

Microsoft [2020]. *Microsoft 365 apps say farewell to Internet Explorer 11 and Windows 10 sunsets Microsoft Edge Legacy*. 17 Aug 2020. https://techcommunity.microsoft.com/t5/microsoft-365-blog/microsoft-365-apps-say-farewell-to-internet-explorer-11-and/ba-p/1591666 (cited on page 5).

Use, Can I [2021]. *Can I use CSS Flexible Box Layout Module*. 13 Aug 2021. https://caniuse.com/flexbox (cited on page 5).

W3C [1996]. *Cascading Style Sheets Level 1 (CSS 1) Specification*. 17 Dec 1996. https://www.w3.org/TR/REC-CSS1 (cited on page 4).

W3C [2009]. *CSS Flexible Box Layout Module*. 23 Jul 2009. https://www.w3.org/TR/2009/WD-css3-flexbox-20090723 (cited on page 5).

W3C [2011]. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. 07 Jun 2011. https://www.w3.org/TR/CSS2 (cited on pages 4–5).

W3C [2018]. *CSS Flexible Box Layout Module Level 1*. 19 Nov 2018. https://www.w3.org/TR/css-flexbox-1 (cited on pages 5, 7).

WHATWG, W3C [1997]. *HTML 3.2 Reference Specification*. 14 Jan 1997. https://www.w3.org/TR/2018/SPSD-html32-20180315 (cited on page 4).

WHATWG, W3C [2021]. *HTML Standard*. 11 Aug 2021. https://html.spec.whatwg.org (cited on page 3).