

UNIVERSITY OF WATERLOO
Faculty of Engineering

**An Analysis of Using Global Scoped Style Sheets and Atomic Scoped
Style Sheets in the Context of Styling Component Based Applications**

Yahoo!
Sunnyvale, CA

Aditya Sridhar
3B Software Engineering
Student ID 20539123
User ID a3sridha
September 8, 2016

Aditya Sridhar
201 Lester Street
Waterloo, ON N2L 3G1

September 11, 2016

Patrick Lam, Director
Software Engineering Program
University of Waterloo
200 University Ave. West
Waterloo, Ontario. N2L 3G1

Dear Patrick Lam:

This report, entitled “*An Analysis of Using Global Scoped Style Sheets and Atomic Scoped Style Sheets in the Context of Styling Component Based Applications*”, was prepared during my 3B work term at Yahoo!, where I worked on Yahoo! Mail in the Front-end team.

The Yahoo! Mail front-end team is currently developing the next generation web client of Yahoo! Mail built using ReactJS. What did you work on ..

Description of report and how it related to co-op position.

I hereby confirm that I have received no help, other than what is mentioned above, in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Yours,

Acknowledgements

Aditya Sridhar, 20539123

Executive Summary

EXECUTIVE SUMMARY

Table of Contents

Executive Summary	iii
Table of Contents	iv
1 Introduction	1
1.1 Background Information	1
1.2 Introduction to the Issue	1
2 Problem Specifications	3
2.1 Problem Description	3
2.2 Design Constraints	4
2.3 Design Criteria	4
2.4 Evaluation Process	5
3 Accepted Solutions	6
3.1 Global Scoped Style Sheets	6
3.1.1 Explanation	6
3.1.2 Advantages	7
3.1.3 Disadvantages	7
3.2 Atomic CSS	8
3.2.1 Explanation	8
3.2.2 Advantages	9
3.2.3 Disadvantages	9
4 Alternative Not Considered	10
4.1 Explanation	10
4.2 Advantages	10
4.3 Disadvantages	10
5 Conclusions	11
6 Recommendations	12
References	13
Acknowledgements	13

1 Introduction

1.1 Background Information

Yahoo! is a technology company and is recognized as one of the pioneers of the early-internet era. Yahoo! Mail is a web-based email service owned by Yahoo! with hundreds of millions of users worldwide. The next generation of Yahoo! Mail web client is currently being developed and is built using the ReactJS framework. ReactJS is a component-based JavaScript library used for building user interfaces. The idea behind a component-based framework is that a complex user interface should be composed of multiple components, and each component should follow the Single Responsibility Principle (SRP). This implies that a particular component should be responsible for a single part of the user interface.

Concern	Resource
Presentation	CSS
Structure	HTML
Behaviour	JavaScript

Table 1.1: Long table caption

Traditionally, the development of web pages was based on the principle of Separation of Concerns (SoC) which advocates breaking a problem into different concerns and using a resource to address a particular concern. In the context of web pages, the structure, presentation and behaviour layers were identified as separate concerns with Hyper Text Markup Language (HTML) defining the structure of a web page, Cascading Style Sheets (CSS) defining the content presentation styles and JavaScript (JS) defining how the web page behaves with user interaction. Table 1.1 summarizes the SoC principle in the case of web page development.

1.2 Introduction to the Issue

CSS is a collection of rules, each rule containing one or more selectors and a block of styles. These rules are written in style sheets which are then packaged in the main project. A selector is used to identify a particular HTML element in the web page content and the styles specified in the rule are applied to that particular HTML element. Figure 1.1 better

elaborates how CSS rules work. The selector *div* targets the `<div>` HTML element and the styles mentioned in the corresponding style block are applied to the targeted HTML element.

```
1  <div>This is my web page</div>
2
3  div {
4      background-color: blue
5  }
```

Figure 1.1: Example of styling using CSS selectors

The scope of a CSS rule is a portion of the web page where that rule can be used to style an element on the web page. The rules contained in a global-scoped style sheet can be used to style any element on the web page. Hence the scope of the rules in a global style sheet is not limited to a particular portion of the web page. Atomic CSS on the other hand is a design principle that advocates restricting the scope of a CSS rule to a component of the web page.

This report compares CSS in global style sheets with Atomic CSS in the context of styling component-based applications. It first provides some additional information about the issue and lists a set of design constraints and the evaluation criteria. It identifies the benefits and pitfalls of each accepted solution. The report uses AHP to perform quantitative analysis of the alternatives against a set of evaluation criteria. The report then concludes by identifying the best solution and provides some recommendations.

2 Problem Specifications

2.1 Problem Description

CSS is an important component of the front-end toolkit. Since the internet is composed of billions of web pages, CSS is being used on a regular basis to govern the look and feel of web pages. One of the main concerns of using CSS is maintainability. CSS is easy to maintain when the web pages have less content, but as the size of the project increases, the maintainability of style sheets decreases. High maintainability implies that a developer can modify a particular rule without worrying about it adversely impacting other rules. Low maintainability hampers the scalability of a project from a design standpoint and significantly affects the developers in the team.

As mentioned earlier, a CSS selector targets a particular HTML element in the web page content and the styles linked to the selector is applied to element it targets. As the content of the web page grows, CSS selectors are more specific and descriptive to target a particular element in the web page. This is termed as selector specificity. If two selectors target the same element in the web page content, the selector with higher specificity tends to override the selector with lower specificity. The style sheet containing selectors with high specificity makes it harder for an existing style to be overridden by a new style. Hence low specificity leads to high maintainability.

```
1  <!-- HTML code -->
2  <div class="parent">
3      <div class="child1"></div>
4      <div class="child2"></div>
5  </div>
6
7  <!-- CSS code -->
8  .parent .child1 {
9      background-color: red
10 }
```

Figure 2.1: Example of styling using context-based selectors

At times, a developer might want to match a web page element that is a descendent of another web page element. For example in Figure 2.1 the `<div>` element whose class name is `child1` is a descendent of the `<div>` element whose class name is `parent`. The CSS selector `.parent .child1` is an example of a context-based selector. It is targeting the element with class

name as *child1* that is a descendent of the element with class name as *parent*. This means that the descendent element is being styled in the context of its parent, and that the rule is redundant outside of this context. This will ultimately lead to more rules being added to the style sheet if the requirement to style the same element outside of the context arises.

2.2 Design Constraints

A solution should adhere to the following design constraints, in order to be deemed as an accepted solution:

1. The styles should be able to be overridden by other styles if required. That is, the selector must not have the highest specificity.
2. The styles should be able to be applied using class selectors.

2.3 Design Criteria

Reusability and maintainability are important features in the context of component based applications. The following evaluation criteria were designed keeping these features in mind and are used to compare and analyze the two accepted solutions.

1. **Minimize the usage of context-based styling:** The accepted solution should limit the usage of contextual styling as this demotes reusability and portability of styles. This ultimately makes style sheets hard to maintain.
2. **Low specificity of styles:** Low CSS specificity enables the developer to override a CSS rule when requirements change. This makes the style sheets more maintainable.
3. **Intuitive style changes:** The accepted solution should be intuitive and predictable enough for the developer to visualize any changes made to the style sheet in terms of web page presentation. This includes adding, removing and modifying rules.
4. **Style reusability** The accepted solution should promote reusability of styles. New style requirements being added to the specification should sparingly lead to new rules being added to the style sheet.

5. **Decouple markup and styles** The accepted solution should disallow coupling of web page content (markup) and styles. This enables the developer to make changes to the presentation without making changes to the markup.

2.4 Evaluation Process

3 Accepted Solutions

3.1 Global Scoped Style Sheets

3.1.1 Explanation

Before CSS was invented, styles were written directly on the HTML markup, resulting in web page content and styles being tightly coupled with each other. As the use of the internet evolved, styles were isolated in style sheets by virtue of the SoC principle. These style sheets had a global scope and the rules present in these style sheet can be used to style any element in the web page content. Since the scope is not limited to a particular portion of the web page, the usage of context-based selectors is inevitable as the size of the project grows. As an example consider the case in Figure x.x. The CSS selector is targetting the `<p>` element and setting its font size to be 10 pixels.

```
1 <p>Here is a paragraph with small font</p>
2
3 p {
4     font-size: 10px;
5 }
```

Figure 3.1: Plain selector targetting `<p>` element

```
1 <p>Here is a paragraph with small font</p>
2
3 <div class="enlarged">
4     <p>Here is a paragraph with large font</p>
5 </div>
6
7 p {
8     font-size: 10px;
9 }
10
11 .enlarged p {
12     font-size: 30px
13 }
```

Figure 3.2: Context-based selector targetting `<p>` element

If a new requirements demands all `<p>` elements contained inside a `<div>` element with class name as *enlarged* to have a font size of 30 pixels. To override the existing style targetting all `<p>` elements, the usage of context-based selector as illustrated in Figure x.y cannot be avoided. It can noted that the specificity of the new selector has also increased, making it harder to be overridden in the future.

3.1.2 Advantages

The following are the advantages of using global-scoped style sheets:

1. Web page styles are decoupled from the markup allowing for a more flexible development with less refactoring. Changes made to the markup did not often lead to changing of styles and vice-versa.
2. This approach introduced the usage of selectors to target a particular element in the web page markup. This enabled matching markup elements with similar styles and styling them all at once, improving reusability considerably.

3.1.3 Disadvantages

The following are the disadvantages of using global-scoped style sheets:

1. Since scope of CSS rules is not restricted to a particular section of the web page, the usage of context based selectors is unavoidable as size of the project grows.
2. CSS selectors tend to be more descriptive as size of the project grows. Hence CSS selector specificity is high.
3. CSS rules are not intuitive as the effect made by a context-based CSS rule on the presentation of a web page element is hard to visualize.

3.2 Atomic CSS

3.2.1 Explanation

Atomic CSS advocates restricting the scope of a CSS rule to a component of the application. It was invented by engineers at Yahoo! primarily to adopt a more modular approach to maintaining style sheets. This approach was found to thrive in the context of component-based technologies like ReactJS. This approach violates the SoC principle since a component's presentation is mixed with its structure. However this is logically appropriate in the context of component-based applications as a component is considered an independent view that handles its own behaviour and application state. Hence this approach enables a developer to maintain components instead of maintaining style sheets.

```
1  <p class="small-font">Here is a paragraph with small font</p>
2
3  <div class="enlarged">
4    <p class="large-font">Here is a paragraph with large font</p>
5  </div>
6
7  .small-font {
8    font-size: 10px
9  }
10
11 .large-font {
12   font-size: 30px
13 }
```

Figure 3.3: Atomic CSS method

This method promotes the use of unsemantic class names. There is a one-to-one relationship between a style and a class name. Hence the styles are context agnostic* improving their reusability dramatically. Figure 3.3 is an example of how one can style web pages using the Atomic CSS method. The specifications of this example is the same as that in Figure 3.2. Here, a class name is allocated for each style and the styles are not contextual. The specificity of the styles is also low. Further, since the styles are devoid of a context, they can be reused in other parts of the component.

3.2.2 Advantages

The following are the advantages of using the Atomic CSS method:

1. Since CSS styles are context agnostic, they can be used to style a particular HTML element in the component without the usage of context-based selectors.
2. Owing to the one-to-one relationship between styles and class names, a CSS selector need not be descriptive when targetting a particular HTML element in the component. Hence specificity of a selector is low, allowing for easier style overrides in the future.
3. The effect made by a CSS rule on a component's presentation is easy to visualize. This makes it easier to identify and clean up rules that are obsolete.

3.2.3 Disadvantages

The following are the advantages of using the Atomic CSS method:

1. This method does not follow the SoC principle to some extent as a component's presentation is linked with its behaviour and structure. Hence a change made to a style often leads to changes being made to the component's markup
2. HTML5 documentation[2] promotes the use of semantic class names and states that a class name should be reflective of the element's behaviour instead of describing an element's presentation. This is being violated by the Atomic CSS approach as class names are used to describe an element's style.

4 Alternative Not Considered

4.1 Explanation

This method is used to apply styles directly to an HTML element[1] and is named the Inline CSS method. Hence styles and markup are tightly coupled together. Figure 4.1 illustrates how to style a particular HTML element using the Inline CSS method. The style attribute is used to record an element's styles. Inline CSS have the highest specificity and will have precedence over other methods used to style web page content. This was one of the earliest methods to style web pages and was soon replaced by external style sheets. Today, it is considered a bad practice to use this method to style web pages.

```
1 <p style="font-size:10px">Here is a small paragraph</p>  
2
```

Figure 4.1: Inline CSS method

4.2 Advantages

There are limited advantages to the Inline CSS method.

1. Since they have the highest precedence, they can be used to try out a particular style on an element to determine if it fits or not.

4.3 Disadvantages

The following are the disadvantages of the inline CSS method

1. Inline styles have the highest specificity, making them hard to be overridden by other styles.
2. This method leads to poor reusability of styles. Styles must be specified using the style attribute for every occurrence of the target element in the markup.

5 Conclusions

CONCLUSIONS

6 Recommendations

RECOMMENDATIONS

Acknowledgements

ACKNOWLEDGEMENTS