Cascading Tree Sheets:
A Templating Language for the Web
6.UAP Final Report

John J. Wang

Supervisors: Edward Benson and David Karger

Department of Electrical Engineering and Computer Science

wangjohn@mit.edu

December 8, 2013

1 Introduction

Modern web development can be quite challenging. A web developer needs to be a jack of all trades, understanding everything from system adminstration to design. In fact, a web developer is typically expected to create, deploy, then maintain a website. This requires a diverse set of skills. Spinning up servers and writing CSS could conceivably both be part of a web developer's job description.

A web developer requires a broad range of skills, and one key challenge of this is that understanding and adapting content to a website is a difficult and cumbersome task on top of everything else a developer faces. Web developers have become the de-facto resource for publishing content, as well as creating an environment in which content can be seen.

In earlier eras, there was always separation between the people creating content and the people creating an environment for the content to live in. For example, book printers took the words that were given to them and transcribed them into a new medium. Other examples of this separation come from newspapers and magazines. Editors designed the layout and made small, superficial changes to the content, while writers actually produced the content.

This separation of concerns made it easier for each type of worker to focus on their specialty. The book printer did not need to study the intricacies of language, he or she only needed to transcribe the words. This separation helped make content publishing more efficient.

However, the internet has unhinged the barrier between content producer and layout creator, shifting extra jobs to the web developer. This extra load has made it difficult to produce websites with both high quality content and a high quality layout. Hypertext Markup Language (HTML) was created to allow a user to structure content on the internet. However, the ability to actually structure content on the web was not fully decoupled from the layout of a webpage. This prevented the full separation between content producers and layout creators.

This fundamental problem in the internet affects more than just web developers– it prevents pure content creators from being able to publish content to the internet easily and without guidance. Content creators now also must learn the basics of web development because their content's HTML structure will provide a basis for a website's layout.

To solve this problem and to decouple content creation from website layout, the Cascading Tree Sheets (CTS) language was created. This paper will describe CTS and show that it can be deployed to completely separate content from layout on the internet.

```
<html>
  <body>
    <div class="surrounding-border">
      <div class="about-me">
        Content for the About Me section
      </div>
      <div class="contact-info">
        Content for the Contact Information section
      </div>
    </div>
  </body>
</html>
```

Figure 1: Example of HTML structure

1.1 The Problem with HTML and CSS

The current idioms for writing modular content in HTML is through using a combination of HTML and Cascading Stylesheets (CSS) files. CSS was created to provide styling to HTML pages. In essence, CSS provides a way to change the display of content on an HTML page. Unfortunately, CSS does require its HTML content to be structured in a particular way. In this senese, CSS does not provide full separation between content and layout.

For example, suppose a user wanted to render two blocks of text on his website, an "About Me" section and a "Contact Information" section. Each section would provide information in an HTML format. The user might style his site purely using CSS. For example, the user's HTML content might look something like this:

In figure 1, there are two CSS classes which hold relevant content, the first is the "about-me" class and the second is the "contact-info" class. However, notice that the

"surrounding-border" element contains no extra information. However, if a developer wants to create a border around both the About Me and the Contact Information sections, then he or she has no choice but to create that class. Unfortunately, CSS does not provide a way to create a layout which such a notion of combined structure without using HTML. This means that is no way to group HTML elements together without defining new groups in the HTML itself.

This is a single obvious example of how CSS and HTML fail to decouple content from layout, but there are many other examples. In essence, these two languages only provided the groundwork for such a decoupling, but never completely finished the implementation. Moreover, it is easy to see how valuable such a separation would be. For example, if one never had to change HTML in order to change the layout and styling of a website, then designers and content producers could work entirely independently, improving efficiency for each party. Moreover, people could specialize to become either a designer or a content producer. CTS attempts to solve this problem by finishing up what HTML and CSS started.

1.2 The CTS Language

The way CTS aims to solve this separation problem is essentially by defining a way to annotate HTML. Instead of having content living directly inside of HTML, with the CTS abstraction, one can think of content as living in its own separate world, then being mapped to particular elements of HTML. This annotation scheme provides a couple of interesting ideas that can make web development much simpler.

First, it allows HTML to become a language of structure. Without CTS, HTML serves as a means to simultaneously store content and define the structure which used by CSS. The dual nature of HTML makes it brittle and hard to adapt. This dual nature is one of the fundamental reasons why the internet, as it stands today, is unable to separate content from layout. However, CTS adds a third abstraction to this set of languages. CTS provides a storage layer for pure content so that HTML

can be used solely for structuring the layout.

The second advantage of the CTS annotation scheme is that it provides a mapping between content and structure. Currently, there is an implicit mpping between content and structure. This exists through the interleaving of content and structure that is seen in HTML. Thus, one can find the mapping by looking at a piece of content on a webpage and finding the corresponding HTML element to which that content belongs. This implicit mapping makes it harder for developers to reason about structure. Moreover, it prevents developers from being able to easily switch exchange layouts with different websites. A developer would first have to find the implicit mapping if he wanted to change all of the content of the website at once. The CTS annotation scheme makes this mapping explicit and easy to think about. A consequence of this mapping is the ability to dynamically restructure a webpage, which will be addressed later in the paper.

Finally, a third advantage is that CTS enables easy content creation and retrieval. One common problem among developers is retrieving content from a webpage. Unfortunately, since content and structure is interleaved inside of HTML elements, one must find the implicit mapping between content and structure, pruning a webpage for content belonging to some category. Instead of having a nicely categorized set of content, HTML pages make it very difficult to actually identify what categories a particular piece of content belongs to. CTS solves this problem by keeping the content in a single place, agnostic of structure, and providing categorization of that content. In CTS, one can define a particular category for content and place all relevant content in that category. For example, if a user wanted to write about cups, then he could create a cup category and confine his discussion of cups to that category. Each of these categories would be defined in a CTS file and serve to organize content. Not only does this make content easier to retrieve, but it also makes it easier to create and edit, since one no longer has to traverse the structure of HTML.

1.3 Summary of CTS Syntax

This subsection will provide a brief overview of how CTS operates. Each CTS file contains rules which are defined as follows:

```
TargetSelector { Relation: SourceSelector; }
```

Here, the TargetSelector refers to a CSS selector which refers to some HTML element. SourceSelector refers to the contents of a CTS node. Finally, relation can be one of the following possible CTS relationships, as given by Benson 2013:

- IS - Copies node from source to target
- ARE - Repeats target nodes for each in source
- IF-EXIST - Conditions target existence on source
- IF - Conditions target nonexistence on source

For example, a possible CTS rule would map content between the title on an HTML page to the content's title. An example of a possible CTS rule for doing this would be the following:

```
div.main-title { is: #title }
```

This relation serves as a part of the mapping between content and structure. Here, the structure would lie in the HTML element div.main-title, which should contain the content #title. The content for the #title category is stored in what is termed a mockup: an abstract document which consists of a category and the content corresponding to that category.

One can perform operations to either populate HTML using content and categories from a mockup, or to populate the mockup using content from the HTML. Both of these types of operations can be performed after the CTS rules have been defined. With multiple CTS rules spanning all different content categories on a web-page, one can complete the mapping between content and structure.

```
default | .site-title          :is      #site-title;
default | .site-description    :is      #site-description;
mockup  | .site-description    :is      #site-description;
default | .menu > ul           :is      #nav-main > ul;
```

Figure 2: An example of a CTS file providing a mapping between content and structure.

Figure 2 provides an excerpt from an example CTS file. This mapping was used when annotating Wordpress blogs using CTS. The additional component used before the CSS selector for the TargetSelector is specified is the document one wishes to map to. The default document is the canonical HTML webpage that one sees when visiting a particular url. The mockup document contains the content that has been mapped fromt he default document.

One can see that specifying CTS rules is relatively simple and straightforward, as long as one has a good understanding of CSS. By supplying these CTS rules, a user can build up a mockup with categorized content, or graft that mockup onto a webpage.

2 Testing and Evaluation of CTS

In order to evaluate how well CTS has enabled the separation of content and layout, I developed a platform for hosting, editing, and copying web content for my 6.UAP final project. This platform will serve as a product for testing the viability of using CTS in production. The platform essentially builds a foundation to experiment how Content Management System (CMS) technologies can be improved in the presence of web relationships, where CTS serves as an implementation of web relationships.

I have worked alongside Edward Benson, Oliver Song, and Jessica Anderson to create a testing system which consists of the following components:

- CTS-UI: The front-end component of the platform. Provides UI components for editing, copying, and saving a website.
- CTS-Server: The back-end component of the platform. Provides an API to the front-end component and stores and serves hosted websites.
- CTS-js: A javascript library for implentating CTS annotations, this is the main library behind powering CTS.
- Mockups: A collection of CTS files which annotate common Wordpress themes.

2.1 Function of the CTS System

The CTS System is implemented as a bookmarklet–a web browser bookmark used for extending functionality on a web page. The use of a bookmarklet allows us to create a system which provides extra functionality in all types of web browsers. For example, a Chrome or Firefox extension would only provide functionality for a specific web browser, whereas a bookmarklet is web-browser agnostic.

The CTS System allows you to navigate to a webpage and perform common operations on that webpage. These operations allow a user to examine a webpage and conceivably copy the theme or layout of the webpage. Some of the specific operations which exist are the following:

- Copy: Parse the HTML, javascript, css, and other assets on the current webpage, and stores the resulting data on either the CTS-Server or allows the user to download the data as a zip file.
- Edit: Make changes to the HTML content or structure.
- Save: Persist any changes or edits made to a webpage on the CTS-Server or download the resulting changes as a zip file.

Each of these operations allows a user to change webpages and persist those changes. Once these operations have been performed, CTS annotations would allow a user to switch content across different webpages. For example, the bookmarklet would allow you to annotate your own webpage with CTS annotations, and switch in the layout for a different website using the CTS system.

In particular, the CTS system enables easy changes to a website layout and theme using CTS annotations. Thus, the system will hopefully test how well CTS enables separation between layout and content, since its primary use case would be to switch around themes or content for different webpages.

2.2 The CTS Server

My part in the project was to develop parts of the CTS-Server. I worked on the following parts of the CTS-Server:

- Url storage and viewing: I created the ability for users to persist a website and access that website through a unique url on the CTS-Server. Through this feature, a user can navigate to a website, use the CTS systems' bookmarklet to persist a website to the CTS-Server, and then view that persisted website later. The url storage feature gives the user a unique url whenever a user persists a website to the CTS-Server. This means that a user can host and edit their own website on the CTS system. Thus, users of the CTS system who do not know or are unwilling to host their own website can skip over the system administration aspect of web development. This means it is much easier for users to observe a website's theme and provide their own content for that website.

- Tree adapters: These adapters provide operations for editing a webpage. In particular, tree adapters encapsulate the changes that occur when a user makes edits to a particular webpage and serve as an endpoint for processing edit and save events. I created a number of tree adapters which provided functions which

allowed edit operations to be processed more easily.

- Cache persistence prototype: I created a prototype for how websites and edits would be stored on the CTS-Server. This prototype involved using an in-memory cache to store website data. The persistence prototype had a very simple API consisting of saving, fetching, and removing website data from the cache. The cache was implemented through associative arrays in javascript.

- MongoDB persistence: After the cache persistence prototype was created, the CTS team realized that in-memory persistence was unacceptable for longer lasting applications. To design a longer-term system of persistence, I designed and implemented a persistence layer in MongoDB. This persistence layer used MongoDB as a means to persist data across multiple sessions. This meant that the data stored on the server would exist whether or not the CTS-Server was up and running. Additionally, this new persistence layer meant that the CTS system would be much more reliable even throughout server crashes.

- Zip controller: