Analysing CSS using the M^3 model

Nico de Groot
University of Amsterdam
Amsterdam, Netherlands
nico@nicasso.nl

Abstract

Cascading Style Sheets (CSS) is a language that adds presentation semantics to hyper text documents. Even though CSS has a relatively simple syntax, lots of mistakes are made during the development of style sheets, ranging from small syntactic flaws to duplicate code. In this effort we introduce an M^3 front-end for CSS, a simple and extensible model for capturing facts about source code, allowing developers to analyse and transform CSS without much excessive development. During the development of the M^3 front-end for CSS, the proclaimed flexibility of the M^3 model has also been confirmed since it had only been successfully implemented for the Java and PHP programming languages. The unique characteristics of CSS, such as its cascading abilities, or its dependency on HTML, did not result in any major complications for the M^3 model. The M^3 front-end for CSS has also been validated by conducting analysis on real-world CSS, from this we can conclude that the M^3 model is able to successfully support today's CSS standard (level 3). By presenting the implementations for some of the conducted analysis, we support our claim stating that the M^3 model allows developers to conduct analyses, and manipulations on CSS with little ease.

1 Introduction

Cascading Style Sheets (CSS) is a language that adds presentation semantics to hyper text documents and has been around since 1997 [Wal97]. Although most often applied to hyper text documents written in languages such as HyperText Markup Language (HTML) and Extensible Markup Language (XML), CSS can also be applied to other XML based documents such as Scalable Vector Graphics (SVG). CSS is a highly popular language as approximately 95% of all websites use CSS[1], and it is ranked as the 6th most popular language on Github in 2015[2], rising from the 10th place in 2013.

CSS has a relatively simple syntax, but this does not mean that no mistakes are made during the development of style sheets. In fact, Mazinanian et al. demonstrated that there is on average a 60% duplication in style declarations [MTM14] when analysing 38 real-world web application, and Gharachorlu demonstrated a total of 500 websites to find out that 499 of them contained code smells [Gha14].

This is where the M^3 model comes in. The M^3 model is a simple and extensible model for capturing facts about source code for future analysis [BHK+15]. An M^3 front-end for CSS would allow developers to produce

[1] `http://w3techs.com/technologies/details/ce-css/all/all`
[2] `https://github.com/blog/2047-language-trends-on-github`

metrics and perform analysis on their CSS without much excessive development since the M^3 model is part of the Rascal meta-programming language (the "one-stop-shop" [KLP12]). Providing developers with the M^3 model in combination with Rascal's features allows for the analysis, transformation, generation and visualization of CSS without much excessive development.

2 Problem context

The problem studied in this work regards the quality of CSS in real-world web applications, and how it can be improved by presenting a tool that would allow developers to conduct analysis and transformations on CSS more efficient. The success of the implementation of this tool, the M^3 front-end for CSS, is uncertain due to the unique characteristics of CSS in comparison to the already implemented programming languages (Java and PHP). Which are currently the only languages supported by the M^3 model. In comparison to these languages, CSS is very different as it is a DSL, it is used solely to express the presentation of HTML, it is completely dependant of HTML, and other unique features such as its cascading abilities.

Therefore a successful implementation of the CSS front-end for the M^3 model would confirm the claim of the authors of the M^3 model stating that the M^3 model "is not only restricted to handling programming languages", and that it should be able to "model other kinds of formal languages like grammars, schema languages, or even pictorial languages." [BHK+15].

3 Research questions

This study will answer the following research questions:

RQ1: Is the M^3 model able to support CSS without having to alter the M^3 model's core?

RQ2: How does the M^3 model for CSS differ from the other models for Turing complete languages?

RQ3: Can the M^3 model for CSS be used to conduct simple analysis such as volume metrics?

- **RQ3.1:** Can the M^3 model for CSS be used to measure CSS specific metrics?
- **RQ3.2:** Can the M^3 model for CSS be used to validate coding conventions?
- **RQ3.3:** Can the M^3 model for CSS be used to refactor CSS?
- **RQ3.4:** Can the M^3 model for CSS be used to detect code clones?

RQ4: Does the M^3 model allow programmers to be more efficient[3] than other already existing tools?

4 CSS

CSS is a language that adds presentation semantics to hyper text documents written in languages such as HyperText Markup Language (HTML) and Extensible Markup Language (XML), although it can also be applied to other XML based documents such as Scalable Vector Graphics (SVG). It is a web standard and is actively being evolved and maintained by the Wide Web Consortium (W3C)[4]. One of the main benefits of CSS is that it allows developers to separate the styling of a page from its structure. By doing so it encourages the reuse of style sheets, since those can be used to multiple web pages, or even multiple websites.

5 The M^3 model

The goal of the M^3 model is to create a unified form for storing facts about source code, with M^3 models for different languages having predictable shapes. However, the M^3 model gives accuracy a higher priority than reuse, as accuracy is required for conducting insightful analyses. To achieve this, the M^3 model uses two types of intermediate representations to reason about code, these are the abstract syntax tree (AST), and the relational layer. The AST is specific to the language, while the relational layer is more abstract, allowing its structure to be reused for different languages. Therefore the AST is first prescribed as it is specific for the language, and only then the relational layer where it is possible to abstract.

[3]The lesser amount of lines of code required to implement analyses and manipulations in relation to other third-party CSS analysis tools.

[4]https://www.w3.org

The relational layer consists of binary relations, these binary relations are almost all between source code locations, with source code locations being hyperlinks that refer to pieces of source code. Allowing developers to navigate directly to the related source code, allowing them to conduct very precise analysis that require the original source code. The M^3 model's core provides 8 binary relations by default of which only 3 are "necessary core relations" [5], these 3 are `containment`, `declarations`, and `uses`), mapping what is logically contained in what else, mapping declarations to where they are declared, and mapping source code locations of usages to the related declarations respectively.

The AST in the M^3 core consists of 5 predefined node types, being Statement, Declaration, Expression, Modifier, and Type. Some of these node types also have specific annotations bound to them. To give some examples: Declarations may have a `decl` annotation pointing to their logical location identifier (a unique source code location represented as a fully qualified name), and Expressions may have a `type` annotation, containing the type produced by the expression. However, there is one annotation that node types have, which is the `src` annotation that points to the source code location.

6 Implementation

The core M^3 model has to be extended for supporting additional languages. Extending the AST for CSS did not result in any issues as the AST predefined in the core M^3 model was flexible enough. For CSS not all predefined annotations where relevant, and thus not all were used. However, since there are optional this was not a real issue. The relational layer was also flexible enough to support CSS, the 3 predefined relations which where mandatory all applied to CSS. Of all 8 predefined relations, only the `types` relation was not applicable for CSS, which is intended to map types to declared source code artefacts. This is because all types of source code artefacts can be resolved directly. The `types` relation will only be relevant when the M^3 model will also support CSS variables[6]. However, since the CSS variables specification is not a part of the 2015 CSS Snapshot[7] (as it is still a draft), which contains all specifications that the M^3 model currently supports, the `types` will be ignored for this version of the CSS front-end.

7 Validation

The sample-set, used for validation, consists of 50 style sheets are a selection of the Alexa top 500 most popular websites on the web[8]. The validation of the M^3 model using these style sheets starts relatively simple with some classic analysis; volume metrics. With the M^3 model, each of these metrics only required a single lines of code (LOC) to implement (Listing 1), making it very simple to obtain these metrics. Running these metrics on the sample-set provided us with the following information, the sample-set contains a total of 543,311 LOC, with the average for a website being 10,866. Furthermore Pinterest.com stands out due to its size with a total of 57,266 LOC.

Listing 1: Volume metrics

```
int calculateBlankLines(loc stylesheet) = size([0 | line <- readFileLines(stylesheet), trim
    (line) == ""]);
int calculateLinesOfComments(M3 stylesheetM3) = sum([0]+[calculateAllLines(d[1]) | d <-
    stylesheetM3@documentation]);
int calculateLinesOfCode(loc stylesheet, M3 stylesheetM3) = (size(readFileLines(stylesheet)
    )-calculateBlankLines(stylesheet)-calculateLinesOfComments(stylesheetM3));
```

For the CSS specific metrics, a search had been conducted resulting in 3 papers. However, 1 of the 3 papers required additional information regarding the related HTML besides the CSS, and as no M^3 model for HTML exists yet, this paper was dropped for the validation process. However, all metrics proposed by the other 2 papers have been successfully implemented. The implementations of two metrics selected from the paper by Adewumi et al. [AMIO12] are shown in Listing 2 and Listing 3. By using the containment relation and some predefined methods that the M^3 provides, such as `declarations`, `ruleSets`, and `isDeclaration`, both metrics could be implemented in a single line.

[5] http://stackoverflow.com/questions/37116715/generic-core-binary-relations-of-m3/37116935#37116935
[6] https://drafts.csswg.org/css-variables/
[7] https://www.w3.org/TR/css-2015
[8] http://www.alexa.com/topsites/global

Listing 2: Number of Attributes Defined per Rule Block

```
real NADPB(M3 stylesheetM3) = toReal(size(declarations(stylesheetM3)))/toReal(size(ruleSets
    (stylesheetM3)));
```

Listing 3: Number of Cohesive Rule Blocks

```
int NOCRB(M3 stylesheetM3) = size([0 | rule <- ruleSets(stylesheetM3), size({a | <e,a> <-
    stylesheetM3@containment, e == rule, isDeclaration(a)}) == 1]);
```

Besides volume metrics, and CSS specific metrics, the M^3 model could also be used to validate coding conventions with, such as the disallowing of ID selectors. Using a single line containing a list comprehension, this coding convention was easily implemented as can be seen in Listing 4. Other tools also validating this coding conventions, such as CSSLint[9] and Styleint[10], required 44 and 37 LOC respectively, thereby showing the efficiency of the M^3 model.

Listing 4: Function for checking if ID selectors are using

```
list[str] avoidIds(Statement stylesheetAST) = ["ID selector used at: <i@src>" | /i:id(str
    name) := stylesheetAST];
```

Another coding convention, which requires a bit more complexity than the previous example, is that declarations containing vendor (browser) specific properties, should be followed up by another declaration containing the related standard property (providing a fallback for other browsers). Implementing this coding convention using the AST took only 22 LOC as can be seen in Listing 5.

Listing 5: Validating the margin shorthand convention convention

```
void vendorPrefixFallback(Statement stylesheetAST) {
  list[str] prefixes = ["-moz-", "-webkit-", "-o-", "-ms-"];
  set[str] required = {};
  visit (stylesheetAST) {
    case rs:ruleSet(list[Expression] selector, list[Declaration] declarations): {
      if (size(required) > 0) {
        println("One or more vendor specific declarations are used without a fallback of
            the standard property at: <rs@src>");
        required = {};
      }
    }
    case d:declaration(str property, list[Type] values): {
      if (property in required) {
        required -= property;
      } else {
        for (pre <- prefixes, startsWith(property, pre)) {
          required += replaceFirst(property, pre, "");
          break;
        }
      }
    }
  };
}
```

While detecting a coding convention is one thing, the M^3 model also allows developers to refactor their CSS using its AST in combination with the pretty printer. By rewriting the implementation shown in Listing 5 a bit, the coding convention is now not only detected, but also fixed, by appending the standard property with the correct value to the rule set, as shown in Listing 6.

Listing 6: Function for adding a fallback for vendor specific properties

```
Statement vendorPrefixFallbackRefactor(Statement stylesheetAST) {
  return visit (stylesheetAST) {
    case ruleSet(list[Expression] selector, list[Declaration] declarations) =>
        vendorPrefixFallbackRefactorHelper(selector, declarations)
  };
}
```

[9] https://github.com/CSSLint/csslint/blob/master/src/rules/ids.js

[10] https://github.com/stylelint/stylelint/blob/master/src/rules/selector-no-id/index.js

```

Statement vendorPrefixFallbackRefactorHelper(list[Expression] selector, list[Declaration]
    declarations) {
  list[str] prefixes = ["-moz-","-webkit-","-o-","-ms-"];
  set[tuple[str,list[Type]]] required = {};
  for (d:declaration(str property, list[Type] values) <- declarations) {
    if (<property,values> in required) {
      required -= <property, values>;
    } else {
      for (pre <- prefixes, startsWith(property, pre)) {
        required += <replaceFirst(property, pre, ""), values>;
      }
    }
  }
  for (r <- required) {
    declarations += declaration(r[0], r[1]);
  }
  return ruleSet(selector, declarations);
}
```

The conducted analysis on the sample set containing real-world CSS prove that the M^3 model is fully capable of supporting the current state of CSS as it is defined in the CSS Snapshot 2015[11]. The implementations presented for the analysis of volume metrics, CSS specific metrics, coding conventions, and solving a coding convention by refactoring, show that the M^3 is able to facilitate an environment that is capable of conducting all kinds of diverse analyses and manipulations on CSS. The presented implementations also support our claim that the M^3 model allows developers to conduct analysis without excessive development, as there is less code required than already existing tools, and the code tends to be relatively simple due to the well structured AST and relational layer. As for future work, additional M^3 could be developed for HTML and Javascript to allow more complex analysis that require a combination of data from these languages.

References

[AMIO12] Adewole Adewumi, Sanjay Misra, and Nicholas Ikhu-Omoregbe. *Computational Science and Its Applications – ICCSA 2012: 12th International Conference, Salvador de Bahia, Brazil, June 18-21, 2012, Proceedings, Part IV*, chapter Complexity Metrics for Cascading Style Sheets, pages 248–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[BHK+15] B. Basten, M. Hills, P. Klint, D. Landman, A. Shahi, M. Steindorfer, and J. Vinju. M3: A general model for code analytics in rascal. In *Software Analytics (SWAN), 2015 IEEE 1st International Workshop on*, pages 25–28, March 2015.

[Gha14] Golnaz Gharachorlu. *Code smells in cascading style sheets: an empirical study and a predictive model.* PhD thesis, University of British Columbia, 2014.

[KLP12] Paul Klint, Bert Lisser, and Atze Ploeg. *Software Language Engineering: 4th International Conference, SLE 2011, Braga, Portugal, July 3-4, 2011, Revised Selected Papers*, chapter Towards a One-Stop-Shop for Analysis, Transformation and Visualization of Software, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[MTM14] Davood Mazinanian, Nikolaos Tsantalis, and Ali Mesbah. Discovering refactoring opportunities in cascading style sheets. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 496–506, New York, NY, USA, 2014. ACM.

[Wal97] Norman Walsh. An introduction to cascading style sheets. *World Wide Web J.*, 2(1):147–156, April 1997.

[11]https://www.w3.org/TR/css-2015/