Reference Architecture, Design
of
Cascading Style Sheets Processing Model

Mohamadou Nassourou
Department of Computer Philology & Modern German Literature
University of Würzburg
Am Hubland
D - 97074 Würzburg
mohamadou.nassourou@uni-wuerzburg.de

***Abstract**: The technique of using Cascading Style Sheets (CSS) to format and present structured data is called CSS processing model. For instance a CSS processing model for XML documents describes steps involved in formatting and presenting XML documents on screens or papers.*
Many software applications such as browsers and XML editors have their own CSS processing models which are part of their rendering engines. For instance each browser based on its CSS processing model renders CSS layout differently, as a result an inconsistency in the support of CSS features arises. Some browsers support more CSS features than others, and the rendering itself varies. Moreover the W3C standards are not even adhered by some browsers such as Internet Explorer. Test suites and other hacks and filters cannot definitely solve these problems, because these solutions are temporary and fragile.
To palliate this inconsistency and browser compatibility issues with respect to CSS, a reference CSS processing model is needed. By extension it could even allow interoperability across CSS rendering engines.
A reference architecture would provide common software architecture and interfaces, and facilitate refactoring, reuse, and automated unit testing. In [2] a reference architecture for browsers has been proposed. However this reference architecture is a macro reference model which does not consider separately individual components of rendering and layout engines.
In this paper an attempt to develop a reference architecture for CSS processing models is discussed.
In addition the Vex editor [3] rendering and layout engines, as well as an extended version of the editor used in TextGrid project [5] are also presented in order to validate the proposed reference architecture.

Keywords: CSS, XML, Processing Model, Reference Architecture.

Introduction

The visual appearance of texts and graphics on a display area is affected by the operating system and screen size of the machine where it is presented, as well as the applied formatting technique such as CSS rendering engine. Moreover graphics libraries normally included in the layout engines could also affect documents appearance. The operating system and screen size are out of the scope of this paper.
I will mainly talk about the CSS rendering engines, and possible solutions to inconsistencies and incompatibility that exist among them.
Cascading Style Sheets (CSS) is a style sheet language recommended by W3C [1]. It allows authors and users to attach style (e.g. fonts, spacing) to structured documents such as HTML, XML. CSS separates presentation from the content of documents, thereby simplifying Web authoring and site maintenance.
In other words CSS defines how structured data could be displayed independently of their structures.
The technique of using CSS to format and present structured data is called CSS processing model. For instance a CSS processing model for XML documents describes steps involved in formatting and presenting XML documents.

In order to understand the role of a CSS processing model in a rendering engine, a clear distinction between rendering and layout engines must be made. For instance browsers engines do have rendering and layout engine modules.

A rendering engine module transforms graphics elements and texts that constitute a web document into a raster that can be displayed on screens or papers, while a layout engine module has got the role of computing the positions where to display those textual and graphical elements.
A CSS processing model is part of the rendering engine module. However it could be split between the two engine modules as well.

Each browser based on its CSS processing model renders CSS layout differently, as a result inconsistencies in CSS features support occur. Some browsers support more CSS features than others, and the rendering itself varies. Moreover the W3C standards are not even adhered by some browsers such as Internet Explorer. Of course there exist some solutions known as CSS hacks and filters to resolve these inconsistency and incompatibility problems. Test suites cannot definitely solve these problems, because test cases cannot be exhaustive.
Therefore these solutions are temporary, very fragile, and cannot be exhaustive. To solve these problems effectively and durably, as well as facilitate the usage of CSS for effectively and unanimously styling structured data in browsers and WYSIWYG (What You See Is What You Get) editors, a reference CSS processing model is necessary.
W3C hosts a web editor project called Amaya [20]. Amaya possesses a CSS processing model which has not been declared as a reference model for rendering CSS. Therefore no reference architecture has been so far officially proposed for CSS processing models. This paper is the first attempt to propose a reference architecture that W3C could even provide interface classes for implementing it, and thereby eliminating the need to interpret its CSS specification.

CSS is fast evolving and becoming omnipresent in several XML processing technologies, it is therefore important to

define a reference architecture for CSS processing models in order to keep backward and forward compatibility between CSS rendering engines. The reference architecture would provide common software architecture, and facilitate refactoring, reuse, and automated unit testing.

In [2] a reference architecture for browsers has been proposed. However this reference architecture is a macro reference model which does not consider separately individual components of rendering and layout engines.

This paper discusses the following points:

    a. A reference architecture for CSS processing models
    b. An algorithm for implementing CSS processing model
    c. A flowchart of CSS processing model

Additionally rendering and layout engines of the Vex editor (Visual Editor for XML) [3] and its extended version used in TextGrid project [5] are also presented in order to validate the proposed reference architecture.

Rendering Process

In general a rendering engine is a software program that reads marked up content (such as HTML, XML, etc.), and style sheet (such as CSS, XSL, etc.), and displays the marked up content on a media type (e.g: screen, paper) according to formatting information defined by the style sheet.

In other words a rendering engine requires as input what to paint, how to paint, and where to paint. The inputs are the document to be formatted, the formatting document, and the area or media type where to present the formatted document.

This paper describes the rendering of XML documents with CSS, and the displaying of the result on a canvas.

For reference sake, the basic CSS syntax is as follows: selector {property: value}. This is called a CSS rule.

For instance p {color: black} will apply a value of 'black' to the color property for the text contained in an XML element 'p' representing the selector.

The rendering process is divided into three steps:

a) Mapping between CSS properties and graphics drawing methods

This mapping involves establishing correspondences between CSS properties and graphics (SWT/Swing) drawing functions. For each CSS property, a drawing function is identified, and is used to implement it. Then a table holding CSS properties with their corresponding drawing functions is created.

b) Mapping between CSS selectors and XML elements

An XML document is transformed into a tree structure made of XML elements. Then comparison between the XML elements with CSS selector names is made.

If a match is found, CSS properties of the selector will be applied to the text node of the corresponding XML element.

c) Painting the formatted XML elements on a Canvas

This is where the graphics drawing functions are used to draw the output on the canvas. Depending on how one would like the output to be displayed, it is possible to display an element text node as soon as the matching between a selector and an XML tag is found, or make a temporary storage to hold processed elements, and display the result after having formatted all the elements.

Proposed Reference Architecture

In general software architecture refers to the ways of designing computer software components, modules and the communication between them.

A reference architecture is an agreed architecture for a given domain of knowledge, that provides templates derived from successful solutions for particular problems.

The Reference Architecture is a kind of layered architectural diagrams with each layer depending on the layer above it. Its purpose is to provide common workbenches, and facilitate refactoring, reuse, and automated unit testing.
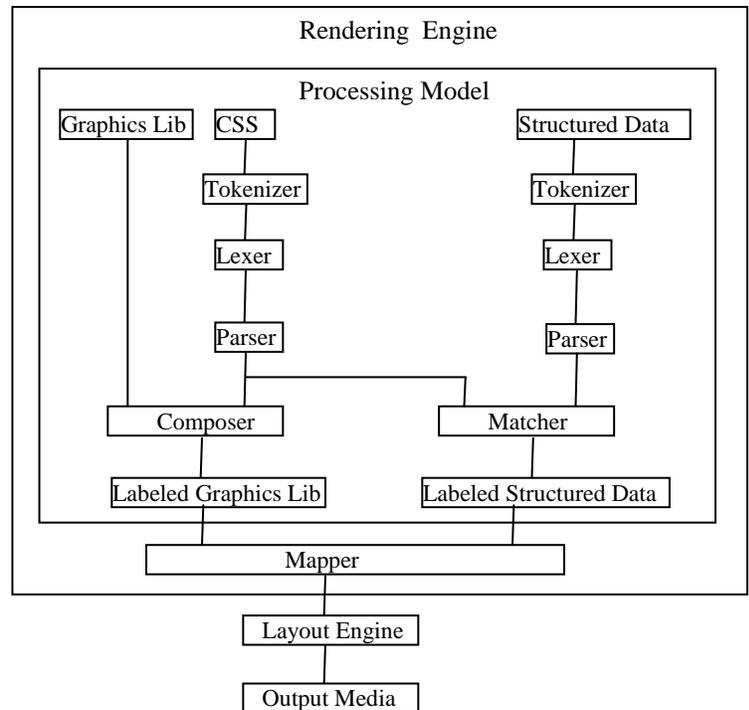
Fig. 1 shows the proposed architecture.

Fig. 1. A reference architecture for CSS processing model

Description of the Architecture

The proposed architecture comprises a rendering engine, a layout engine, and an output media.

The rendering engine consists of a processing model and a Mapper. The processing model produces Labeled Graphics Lib (Libraries) and Labeled Structured Data (XML elements) objects. The Mapper establishes correspondences between the two in order to generate displayable objects, which are passed

to the Layout Engine to perform the actual presentation on a media type (canvas, paper…). Graphics Lib could be an SWT/Swing/AWT library, or any other graphics library. In this case it is an SWT (Standard Widget Toolkit) library.

Structured data could be an XML/HTML document.

The Labeled Graphics Lib component is a table containing SWT printing functions, and corresponding CSS properties and values. The mapping between CSS properties (color, font-size…) and SWT printing functions (drawstring, drawline,…) produces correspondences between each CSS property and an SWT output function.

A table (e.g HashMap in Java) could be used to hold CSS properties as keys and SWT drawing functions as values.

Below is an example of what the table might look like:

CSS properties	SWT functions
color: black	Drawstring
font-size: 12px	Drawstring
text-decoration: underline	drawLine
list-style-type: circle	drawOval
list-style-type: circle	fillOval
border: 1pt solid black	fillRectangle
………..	……….

Table 1. sample of CSS properties SWT functions mapping

Labeled Structured Data represents the modified XML document whereby an attribute style is added to each XML element. The style attribute is made of CSS declarations (property, value) that have to be applied to the XML element.

This step is important because it assigns to each element some defined properties including elements having prior no defined properties, or lower specificity.

In fact rendering CSS is not a trivial task, because it involves some principles such as cascading, inheritance, and specificity. These principles are important for selecting the appropriate CSS properties for a given XML element.

Cascading is concerned with the way styles are assigned to XML documents. Styles could be attached to an XML document through external style sheet, embedded styles, or inline styles.

Inheritance means that properties applied to parent element are also valid for child element.

Specificity implies that the most specific rule must be used.

The Tokenizer

A Tokenizer known as lexical analyzer splits a stream of text into tokens, using for instance whitespace (tabs, spaces, line break, etc). It does not know anything about the meaning or syntax of expressions.

The Lexer

A Lexer is basically a Tokenizer, which is able to identify tokens based on their functions. For example it could find out that one token is a number, another one is a string literal.

The Parser

A parser also called syntactic analyzer takes the stream of tokens from the Lexer and checks whether it is compliant with

a defined grammar or structure. Then it converts the stream into an abstract syntax tree or parse tree.

Because of the difficulty to develop context-sensitive parsers, unifying Lexers and parsers is becoming a common trend. In other words it is possible to include a Lexer as a module within a parser.

CSS Parsers

The function of a CSS parser is to construct a CSS object model for the rules, declarations and selectors contained in the style sheet.

Simple API for CSS (SAC) [9] is a common API for event-based CSS parsing defined by the W3C. It is closely modeled on the SAX API for XML parsers.

SAC 1.3 is implemented in several languages. In java there are many implementations supporting CSS2 among them:

    a. Flute 1.2
    b. Batik SAC 1.2 CSS Parser (Apache).
    c. CSS Parser (David Schweinsberg).

Batik is smaller and faster than Flute. But Flute is easy to update, because Parser.jj (JavaCC file) which describes CSS2 grammar is provided. Unfortunately for almost a decade now the Flute parser has not been officially updated.

The package org.milyn.magger [14] is a CSS Parser that uses Apache Flute for the SAC parsing and Apache Batik for the resulting CSS Selector/Condition model. Batik has problems parsing some CSS rules while Flute does it without difficulty.

CSS parsers such as the above mentioned ones include Tokenizers/Lexers. However for sake of maintenance and reusability, it would be recommendable to clearly separate these components.

As mentioned above SAC is a specification for event-based CSS parsing. Therefore implementations of SAC are also event-based parsing systems. However a tree-based approach specification might be desirable as well, because CSS files are usually not very big.

XML Parsers

Usually an XML parser reads an XML document, identifies all the XML elements and transfers the data for further processing.

There are several XML parsers among them Simple API for XML (SAX) [12] and Document Object Model (DOM) [13]. DOM is a W3C specification system. However there is no formal specification for SAX. Moreover there are other parsers such as Pull-parsers that are similar to SAX, and Data binding parsers that resemble DOM.

A **DOM** parser is a tree-based parser that creates a literal tree in memory, based on the hierarchical structure of the XML document. It requires loading the entire document before starting parsing, and a complete parse tree is produced, regardless of the size of the document. One can navigate and manipulate the tree until it is cleared from the memory. DOM is simple and easy to understand, but resource intensive which

might be overcome by using disk space as memory (persistent DOM).

A **SAX** parser is an event-based parser, which calls handler functions when certain events such as finding text node, child element are encountered. In other words it parses the document line by line. It doesn't keep the parsed tree in memory instead a virtual tree is generated. Therefore it is generally faster and requires fewer resources. However manipulating, traversing, and serializing XML documents are hard, because the parsed XML tree is not kept in memory. Moreover XSLT and XPath which require accessing XML nodes at any time cannot be used without starting the parsing operation again.

Any way selecting a parser must be application dependent.

The package org.xml.sax provides classes and interfaces for SAX. It is a component API of the Java API for XML Processing.

The package org.w3c.dom provides interfaces for the Document Object Model (DOM), which is a component API of the Java API for XML Processing.

The Graphics Lib (SWT)

The package org.eclipse.swt.graphics contains classes that allow management of graphics resources. The class org.eclipse.swt.graphics.GC encapsulates all of the drawing API, including how to draw lines, shapes, text, images, and filled shapes.

Using a GC one can draw onto a Canvas. A Canvas is a region of the screen where an application can draw things. It has got a default method called paint() that must be overridden in order to perform custom graphics on the canvas. A Canvas component could also be used to catch input events from a user.

The Composer

The composer generates composite SWT drawing functions according to supplied CSS declarations (properties and values).

The Matcher

Its task is to find out which CSS properties have to be applied to which XML element. It does that by comparing CSS selectors with XML tags.

The Mapper

The role of the Mapper is to generate formatted XML objects containing functions of the Labeled Graphics Lib (SWT) and the corresponding XML elements.

The Layout Engine

The purpose of the Layout Engine is to create a visual representation of the formatted XML objects. This visual representation is a nested hierarchy of rectangular boxes, implemented as a tree of objects. A rectangular box is an implementation of the W3C box model.

The Output Media

It is the target medium (e.g., print the results on papers, display them on the screen such as canvas, render them as speech, etc.). In this case a canvas is used for rendering the structured data (XML document).

Design of the Processing Model

Software design involves usually components and algorithms implementation issues as well as the architectural view. Following are an algorithm as well as a flowchart describing the working principle of a CSS processing model.

A Sample Practical Algorithm

Following steps could be implemented:
1. First parse the CSS file using Flute/Batik/… which implements the SAC recommendation, Create a HashMap of rules with each one containing triplet of [Selector][property(ies)][value(s)].
2. Then parse the XML document using SAX/DOM/… and create a tree structure of the document. Each element of the tree is made of [Element][attribute(s)][value(s)]
3. Compare each [Element] with each [Selector]
4. If a match is found
5. Check if it is Pseudo-element, if it is not go to step 10
6. Check if 'content' property is present, if it is there go to step 9
7. Check if more properties are there, if yes then go to step 10
8. Go to step 3 next element
9. Evaluate value of content property. Eval() is a function that must be implemented to evaluate the value of the content property. For instance counter () function requires counting of some elements
10. Then apply corresponding SWT/Swing/AWT drawing methods for each property (including inherited ones) to the text node of the XML element, and pass the drawing functions with the text node to the Layout Engine for display
11. Repeat steps 3 to 10 till all the elements and selectors are covered

The HashMap holding CSS selectors and declarations must be correlated with the SWT graphics. All the CSS declarations (property, value) must be mapped to drawing primitives of the SWT graphics. This is explained with table 1.

The following flowchart provides more details about the processing model.

In the flowchart the Analyzer comprises Tokenizer, Lexer, and Parser. The output of the CSS and XML analyzers are collections of CSS rules [Selector][property][value] and XML DOM [Element][attribute][value] respectively. Pty stands for property, Val for value, and attr for attribute.

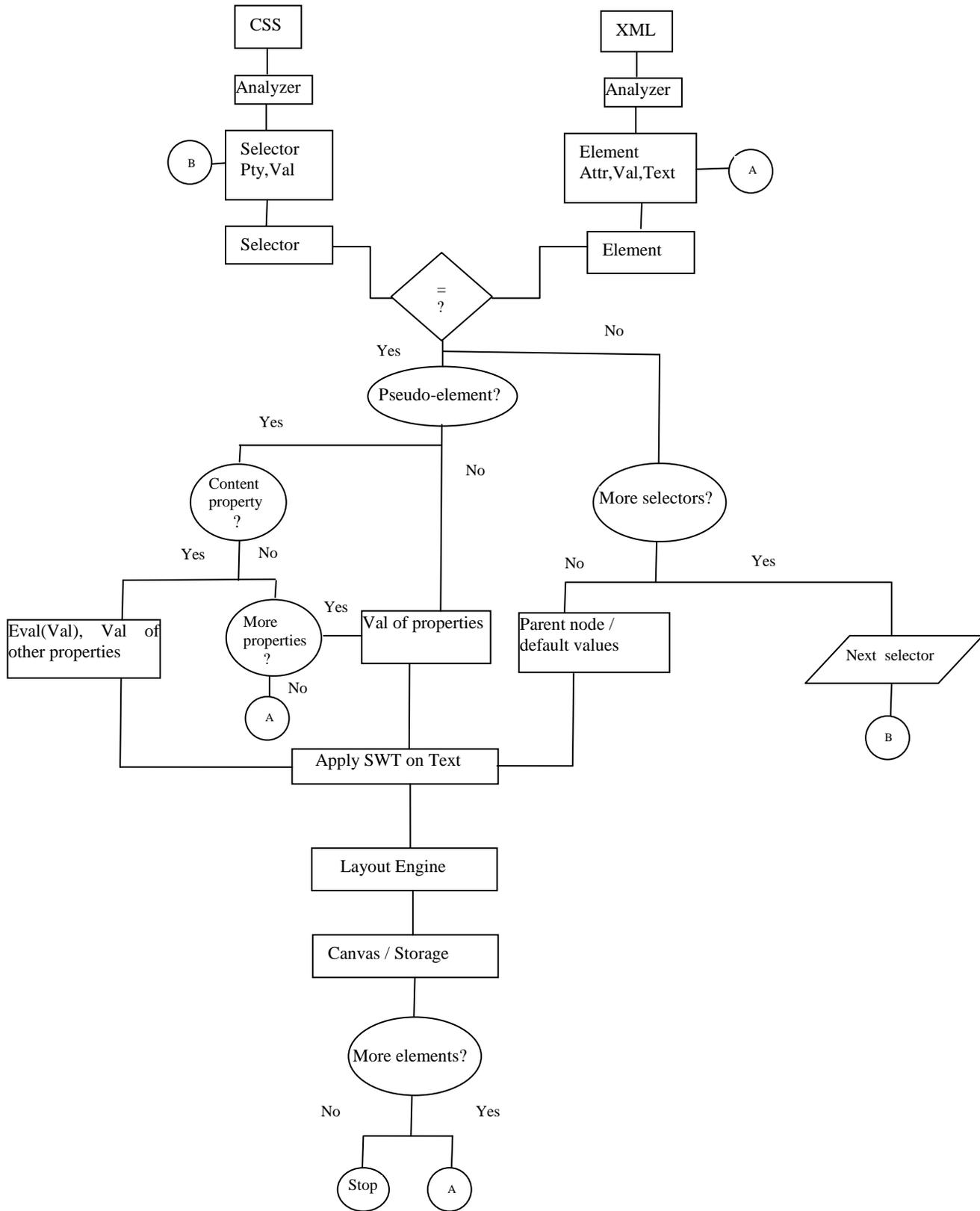The following **Flowchart of the Processing Model** shows stepwise how the model is executed.

Fig. 2. Flowchart of the processing model

Validation of the Reference Architecture

To recover the architecture of the CSS processing model used in the Vex editor [3] and its extended version implemented in TextGrid project, I applied pattern-based techniques. I first built the above described conceptual reference architecture, then the source code of the Vex editor was searched to find instances of those patterns in a top-down manner. It might be important to mention that I could have used a bottom-up strategy, which involves a systematic analysis of CSS rendering engines in order to build the reference architecture.

Vex is an editor for XML documents based on the Eclipse platform [4]. It hides the raw XML tags from the user, providing instead a wordprocessor-like interface.

The TextGrid [5] project provides a multipage editor environment to users. A source editor which uses the Structured Source Editor (SSE) document object model (DOM) of the Web Standard Tools (WST) [15], and the Vex editor whose DOM is derived from the WST's DOM. This implies that the Vex editor depends on the source editor to perform its task.

Vex uses CSS to style the text in the editor. However its documentation does not include a description of the architecture of its CSS processing model. Its layout engine has been introduced as an implementation of the W3C CSS box model [7].

Following is a description of the Vex Engine made of rendering and layout engines.

The Vex Engine

The Vex engine comprises a rendering and a layout engine. The rendering engine is exactly the processing model. The Matcher component produces Labeled XML objects by comparing CSS selectors with XML tags. Contrary to the proposed reference architecture, its Mapper is made part of the Layout Engine. The Layout Engine draws XML elements on the Canvas by mapping the Labeled XML objects to SWT graphics primitives.
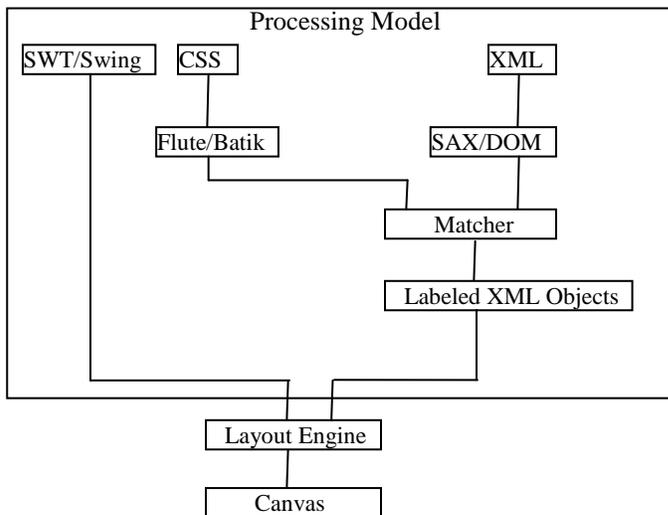
Following is a diagram of the Vex engine.

Fig. 3. Architecture of Vex Engine

Batik [11] was first used to parse CSS files, and then replaced with Flute [9], because Batik seems to have problems parsing some CSS where Flute does it without difficulty.

Flute and Batik parsers do not present Tokenizers or Lexers separately, but rather consider them as modules of the parser. As mentioned previously this approach is undesirable as far as maintenance and reuse are concerned.

Moreover making the Mapper part of the Layout Engine increases the difficulty of refactoring and reuse. Therefore I would suggest a modification of the Vex Layout Engine, which does not even comply with the previous definition of layout engine.

If you are interested in understanding Vex CSS rendering and layout engines, please have a look at [17].

The standard Vex editor uses SAX to generate an object model for XML documents.

The TextGrid's Vex Editor

The data model of the extended Vex in TextGrid project uses the SSE's DOM of the WST. In fact the sse.core package provides a method called IModelManager that can be used to share structured document between many clients at runtime. This is how the Vex editor is synchronized with the source editor to retrieve its DOM.

Obviously this is a nice approach because there is no need to parse and reparse XML documents.

However as I pointed it out before, this might not be the best approach to do it for the following reasons:

1. The Vex editor does not function without the source editor, thereby limiting the use of the editor.

2. It is not clear to me so far which type of parser the SSE DOM uses. Is it a tree-based or an event-based one?

   This is important because of memory and parsing time issues. From my own investigations it appears that it uses a tree-based approach, which I need to confirm from the SSE source code. Simply try opening a very big XML file (e.g > 3 Megabytes), the editor will hang.

3. The SSE data model might not be adequate for Vex because SSE DOM is a hierarchical one, while Vex looks at XML documents as sequence of words. CSS3 properties (e.g calc(),…etc) might not be easily implementable.

The tree-based approach could be improved by making the DOM persistent, that is storing the nodes of the DOM as objects in an object database. Because disk storage access is slower than memory access, it is a challenge to define a method for efficient persistence. Usually binary representation of the document is used.

The Vex Layout Engine

The purpose of the Vex Layout Engine is to create a visual representation of a document given a CSS stylesheet.

This visual representation is a nested hierarchy of rectangular boxes, implemented as a tree of objects.

There are two main types of box. Block boxes containing other boxes and stack their children vertically. Inline boxes whose children are stacked horizontally and are splittable to wrap content into series of lines.

After having examined the CSS rendering process of the Vex editor in order to validate the proposed reference architecture, some browsers' CSS Rendering Engines such as Gecko, Webkit, KHTML were briefly investigated. However I have planned to thoroughly analyze their CSS processing models in the near future.

CSS design principles

In order to make CSS achieve its goals, W3C recommends the use of some design principles such as:

- Forward and backward compatibility between all CSS levels
- Complementary to structured documents such as HTML and XML applications
- Vendor, platform, and device independence
- Maintainability
- Simplicity
- Network performance
- Flexibility
- Richness
- Alternative language bindings
- Accessibility

Discussion and Conclusion

The lack of reference architecture and design for CSS processing models has led to a strong competition between Mozilla's Gecko layout engine used in Firefox, the WebKit layout engine used in Apple Safari and Google Chrome, the similar KHTML engine used in KDE's Konqueror browser, and Opera's Presto layout engine. Unfortunately none of them has perfectly implemented the W3C CSS 2.1 specification as well as the upcoming CSS 3 level.

It is therefore very important to agree on a common architecture and design for CSS processing models.

Of course there are already some solutions known as CSS hacks and filters to inconsistency and incompatibility problems. Test suites for guarantying interoperability among browsers' rendering engines have also been introduced. However these solutions are temporary, very fragile, and cannot be exhaustive.

A reference architecture for CSS processing model as a solution to browsers and XML editors inconsistencies and compatibility issues has been discussed. An algorithm and a flowchart describing how basically CSS rendering engines work have been presented. The described working principle is extensible.

The proposed reference architecture has been validated with the help of CSS rendering engine of the Vex editor, as well as briefly using KHTML, Gecko and Webkit rendering engines.

Additionally the Vex editor used in the TextGrid project has also been explained.

The aim of this research was to find out whether a single CSS rendering engine for all browsers was possible. The results of this paper show that it is in fact doable, if the W3C specification is followed and a reference architecture for CSS rendering engine is agreed upon.

Future Work

Implementation of the proposed architecture as well as the design patterns will be the next steps of this research. Additionally, thorough examination of some browsers' CSS Rendering Engines such as Gecko, Webkit, KHTML, and Presto in order to further validate the architecture will also be performed. A basic standalone CSS debugger would also be designed and implemented.

Finally knowing that CSS is constantly evolving, an interactive tool for automatically updating the implemented architecture would be developed. Even though it might be a separate research on its own, it would however facilitate comprehension and extension of the software.

References

[1] http://www.w3.org/Style/CSS/
[2] Alan Grosskurth and Michael W. Godfrey. A Reference Architecture for Web Browsers. Proceedings of the IEEE International Conference on Software Maintenance, 2005
[3] http://vex.sourceforge.net/index-old.html
[4] http://www.eclipse.org/
[5] http://www.textgrid.de/
[6] http://www.tei-c.org/Guidelines/P5/
[7] http://www.w3.org/TR/CSS2/box.html
[8] http://en.wikipedia.org/wiki/Web_browser_engine
[9] http://www.w3.org/Style/CSS/SAC/
[10] http://www.w3.org/TR/CSS21/intro.html
[11] http://xmlgraphics.apache.org/batik/javadoc/org/apache/batik/css/parser/package-summary.html
[12] http://en.wikipedia.org/wiki/Simple_API_for_XML
[13] http://en.wikipedia.org/wiki/Document_Object_Model
[14] http://www.milyn.org/javadoc/v1.0/magger/org/milyn/magger/CSSParser.html
[15] http://www.eclipse.org/webtools/wst/components.html
[16] http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(CSS)
[17] To appear: Mohamadou Nassourou (2010), Understanding the Vex Engine, in (http://www.opus-bayern.de/uni-wuerzburg/abfrage_suchen.php?la=de), University of Würzburg
[18] http://www.wiki.gis.com/wiki/index.php/Cascading_Style_Sheets
[19] Architecture and evolution of the modern web browser, Alan Grosskurth, Michael W. Godfrey David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada, http://grosskurth.ca/papers/browser-archevol-20060619.pdf
[20] http://www.w3.org/Amaya/