INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Diplomarbeit

CSS^{NG}: An Extension of the Cascading Styles Sheets Language (CSS) with Dynamic Document Rendering Features

Christoph Wieser

Aufgabensteller und Betreuer:	Prof. Dr. François Bry,
Abgabetermin:	30. April 2006

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 30. April 2006 Christoph Wieser

Abstract

Styling and formatting of XML documents for various target media is often specified with the *Cascading Style Sheets* (CSS) language. An appealing feature of CSS is that it specifies formatting instructions using rather simple rules. A limitation of CSS is that it focuses on *static* formatting rules. As a consequence scripting languages such as ECMA Script are used in practice for dynamic adaptation of formatting. This leads to rather complex formatting specifications by comparison to CSS style sheets.

CSS^{NG} is a novel extension of CSS 3, the newest version of CSS, introducing just a few rules for a dynamic rendering and for markup visualization. The main goal of CSS^{NG} is to make scripting languages unnecessary for as many applications as possible. This limited extension of CSS 3 turns out to make possible, for instance, a rather advanced visualization of programs. This thesis (1) introduces into the extensions of CSS^{NG} with respect to CSS 3, (2) describes a proof-of-concept prototype implementation of CSS^{NG}, and (3) demonstrates CSS^{NG} by means of sample applications.

Zusammenfassung

Die Aufbereitung zur Präsentation und Formatierung von XML-Dokumenten für verschiedene Ausgabe-Medien wird oft mittels der Sprache *Cascading Style Sheets* (CSS) spezifiziert. Eine günstige Eigenschaft von CSS ist, dass Formatierungs-Anweisungen mit ziemlich einfachen Regeln spezifiziert werden. Eine Einschränkung von CSS ist allerdings die Fokussierung auf *statische* Regeln zur Formatierung. In Folge dessen werden in der Praxis Script-Sprachen wie ECMA Script verwendet, um dynamische Formatierung zu ermöglichen. Damit geht die Einfachheit der Formatierung in CSS verloren.

CSS^{NG} ist eine neuartige Erweiterung von CSS 3, der aktuellen Version von CSS. Diese Erweiterung führt nur wenige neue Regeln für dynamisches Rendering und für die Visualisierung von Markup ein. Dabei ist das Ziel, die Verwendung von Script-Sprachen für möglichst viele Anwendungsarten abdingbar zu machen. Es wird gezeigt, dass diese beschränkte Erweiterung von CSS 3 beispielsweise ziemlich anspruchsvolle Visualisierungen von Programmen ermöglicht. Diese Arbeit (1) führt in die Erweiterungen von CSS^{NG} bezüglich CSS 3 ein, (2) beschreibt eine prototypische Implementierung von CSS^{NG} und (3) demonstriert CSS^{NG} anhand von Beispielanwendungen.

Acknowledgements

Several people have enriched my work on this diploma thesis. I want to express my gratitude explicitly to *all* members of the teaching and research unit *Programming and Modelling Languages* for giving precious advice and for offering a family-style atmosphere.

More than any other person, I want to thank my supervisor Prof. Dr. *François Bry* for his patience, dedication, and thoroughness in reviewing my work on this thesis. His encouraging motivation during the evolution of this thesis and my studies of Informatics was beyond comparison.

Another highly important person during the time working on this thesis as well as during my studies of Informatics was Dr. *Norbert Eisinger*. He read a draft of this thesis and always took the time giving me advice and hints. His dedication to his work made him more than an inspiring example to me.

I am also deeply indebted to *Sacha Berger*. His diploma thesis led to the idea for mine. Many of his visionary ideas enriched the work on this thesis. Without his refreshing and sometimes unconventional views, the thesis would lack many interesting associations.

I am obliged to the Project "Reasoning on the Web with Rules and Semantics" (REWERSE) for giving me the possibility to participate in the workshop on Principles and Practice of Semantic Web Reasoning (PPSWR) in June of 2006, where I will have the chance to present the results of this thesis.

Last but not least I thank my parents and my partner Anne Katrin for loving and supporting me.

Christoph Wieser

CONTENTS

1 Introduction **1**

1.1 What is CSS? 1

1.2 Shortcomings of Styling Semi-Structured Data Today 2

1.3 Objectives of CSS^{NG} 3

1.4 Advantages of CSS^{NG} 4

2 Cascading Style Sheets (CSS): A brief Introduction **9**

2.1 The Origin of CSS 9

2.2 Separating Design from Content using CSS 10

2.3 Structure of CSS 11

2.4 Simple Selectors 12

2.5 Combinators 13

2.6 Grouping 13

2.7 Pseudo-selectors 13

2.7.1 Pseudo-elements 14

2.7.2 CSS Functions 14

2.7.3 Structural Pseudo-classes 15

2.7.4 Dynamic Pseudo-classes 15

2.8 Interpretation of CSS style sheets 16

2.9 Styling at Unknown Depth 17

3 Limitations of CSS 2.1 and CSS 3 **19**

3.1 Transformation vs. Rendering 19

3.1.1 Deleting XML elements 20

3.1.2 Deleting parts of XML text nodes 20

3.1.3 Adding XML text nodes 21

3.1.4 Adding XML elements 21

3.2 Insertion of Markup 24

3.3 Visualization of Markup 24

3.4 Depth-Dependent Styling 26

3.5 Dynamic Rendering 26

3.6 Hyperlinks 27

4 How CSS^{NG} extends CSS 3 **29**
4.1 Framework for the CSS^{NG} extension 29
4.1.1 Downward Compatibility 29
4.1.2 Combined Complexity 29
4.2 Markup Insertion 29
4.2.1 CSS^{NG} Function `element` 30
4.2.2 CSS^{NG} Function `attribute` 31
4.2.3 Prevention of Iterative Insertion of Markup 32
4.2.4 Well-Formedness 32
4.3 Markup Visualization 33
4.3.1 Rendering of XML elements 34
4.3.2 Rendering of XML attributes using CSS^{NG} Attribute Rules 35
4.3.3 Well-Formedness of Insertions using Attribute Rules 37
4.3.4 Open Issues 38
4.4 Depth-dependant Styling 38
4.5 Dynamic Styling Generalized 39
4.5.1 Recurrence of Events 40
4.5.2 Dynamic Styling Combined 42
4.5.3 Extended Input Devices 43
4.6 Structure-independent Selection 44

5 Related Extensions to CSS **47**
5.1 Dynamic HTML (DHTML) 47
5.2 Action Sheets 49
5.3 Behavioral Extension to CSS 50

6 Prototype of a CSS^{NG} engine **51**
6.1 Requirements 51
6.2 Implementation of the CSS^{NG} Prototype 51
6.2.1 Modifying an Existing Rendering Engine toward CSS^{NG} 51
6.2.2 Using DHTML for extending a Web browser toward CSS^{NG} 53
6.2.3 Choice of Technologies for the CSS^{NG} engine 54
6.3 Architecture 55
6.3.1 CSS^{NG} Lexer 55
6.3.2 CSS^{NG} Parser 56
6.3.3 Configurator 58
6.3.4 Styler-Generator 61
6.3.5 Reifier: Representing XML in XHTML 61
6.3.6 Meta-Initializer 64
6.3.7 Dynamic Styler 64

7 Proof-of-Concept Applications of CSS^{NG} **69**
7.1 Rendering of HTML Documents 69
7.1.1 Temporarily Superimposing the Table of Contents on Keypress 69
7.1.2 Superimposed Notes – Adapting Footnotes to Web Browsers 70
7.1.3 Displaying annotations to Documents 71
7.2 Rendering of a FOAF Definition 72

7.2.1 A Serialization of RDF data . . . 73
7.2.2 Rendering of a FOAF Serialization using CSS^{NG} . . . 74
7.3 Program Visualization As Textual Program Rendering . . . 77
7.3.1 Visualization of Xcerpt Data Terms . . . 77
7.3.2 Superimposing of Context Menus . . . 80
7.3.3 Visualization of Xcerpt Query Programs . . . 81
7.3.4 Highlighting Xcerpt Variables . . . 81

8 Conclusion 83
8.1 Summary . . . 83
8.2 Contributions . . . 84
8.3 Further Research Directions . . . 85

A Code of the CSS^{NG} engine 87
A.1 CSS^{NG} Lexer . . . 87
A.2 CSS^{NG} Parser . . . 91
A.3 Configurator . . . 110
A.4 Styler-Generator . . . 124
A.5 Insertion Preprocessor . . . 132
A.6 Reifier . . . 135
A.7 Meta-Initializer . . . 138
A.8 Dynamic Styler . . . 139

B Code of the visXcerpt Viewer in CSS^{NG} 145

Index 149

Bibliography 152

CHAPTER

ONE

Introduction

Style sheet languages such as CSS [BLLJ98] or XSL-FO [Cla01] have considerably gained in importance since the Web has become a mass medium. Such languages are widely applied for a sophisticated rendering of semi-structured data [ABS00] especially expressed using XML [BPSMM00]. Controlling the appearance of Web pages in Web browsers has become the most frequent application of style sheet languages.

This thesis describes CSS^{NG}, which is a rather conservative extension with respect to the style sheet language CSS 3[1] [Bos05]. CSS^{NG} is rooted in a project thesis [Wie05] focusing on extending stylesheet languages with *dynamic* document rendering features. The focus of this work is a proof-of-concept realization of the considered extensions hand in hand with refinements based on experiences gained in practice using CSS^{NG}.

1.1 What is CSS?

This section introduces informally to basic features of CSS for a better understanding of this introductory chapter. Refer to Chapter 2 for details on CSS features concerning this thesis. The specification of CSS 3 can be found in [Bos05].

CSS 3 and its predecessors have been developed to simplify changes of the content as well as of the presentation of HTML and XML documents by separating content from presentation. Fig. 1.1 below shows the anatomy of a CSS rule :

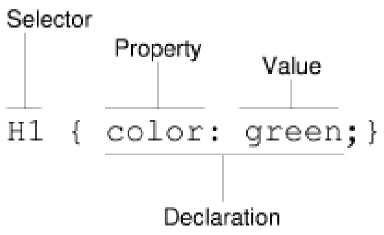


Figure 1.1: Anatomy of a CSS rule.

[1]CSS 3, the newest version of CSS, is about to receive the status of a W3C recommendation, which is in fact a standard.

This schema underlies static and dynamic CSS rules as demonstrated below. The following rule demonstrates a well-known **static styling** feature from Web pages displayed in Web browsers:

CSS Rule:

```
a          { text-decoration: underline; }
```

HTML Code:

```
<a href="http://www.w3.org/Style/CSS/">CSS</a>
```

Rendering in a Web Browser:

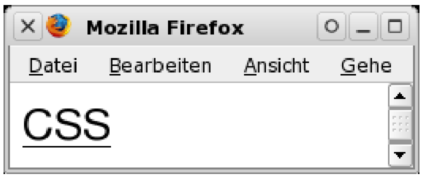


Figure 1.2: Styling specification via CSS for HTML Code and **static** rendering.

The left-hand *selector* of the CSS rule, `a` above, selects HTML anchors. The so-called *declaration* on the right-hand side assigns the styling parameter to XML elements selected by the selector of a CSS rule. In the example above it specifies that anchors are presented underlined as customary in Web pages to mark hyperlinks.

Also **dynamic styling** features are offered in CSS 3. The background color of an HTML anchor can be switched to yellow while the mouse cursor is hovering (`:hover`) over it:

CSS Rule:

```
a:hover   { background-color: yellow; }
```

HTML Code:

```
<a href="http://www.w3.org/Style/CSS/">CSS</a>
```

Rendering in a Web Browser:

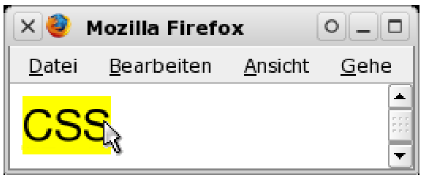
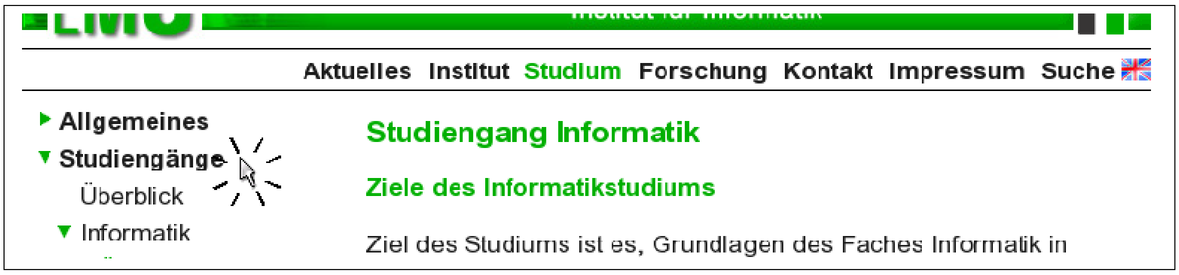


Figure 1.3: Styling specification via CSS for HTML Code and **dynamic** rendering.

1.2 Shortcomings of Styling Semi-Structured Data Today



Ziel des Studiums ist es, Grundlagen des Faches Informatik in

Figure 1.4: Opening a sub-menu on mouse click in a Web browser.

With the emerging trend from static to dynamic Web pages, the expressive power of the dynamic document rendering features in CSS 2.1 and in CSS 3 as informally introduced above are no longer sufficient. Sub-menus, for instance, which can be superimposed on a mouse click, are widespread on Web pages, see Fig. 1.4 above. (In this thesis, mouse clicks are visualized using star-like lines around the mouse cursor.) They cannot be specified in

CSS 3. Furthermore, CSS 2.1 and CSS 3 are often insufficient for a user-friendly rendering of XML documents with complex structures.

In practice scripting languages supporting the DOM [HHW$^+$00] interface to XML documents like ECMA Script [ECM99] are used to obtain dynamic rendering features (see Fig. 1.5). In XHTML documents, for instance, scripts are rather often invoked in the context of an XHTML element by XHTML *intrinsic event* [ABC$^+$99] attributes like `onclick`. As a consequence the styling specification is not separated from content like in CSS. That means that

- dynamic styling via scripting is relatively complicated,
- the maintenance of styling programs is expensive, and
- applying dynamic styling to multiple documents is rather difficult.

HTML Code with ECMA Script Code:

```
<html>
  <body>
     Change my
     <span onclick="this.style.color='red'" >
       color!
     </span>
  </body>
</html>
```

Rendering in a Web browser:

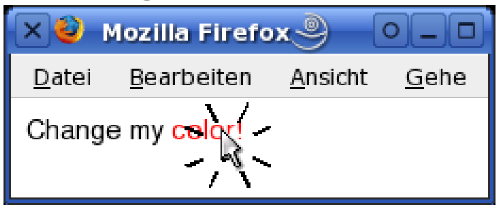


Figure 1.5: Changing the text color in HTML on mouse click to red using ECMA-script.

1.3 Objectives of CSS^{NG}

To overcome the shortcomings above, the main goals of CSS^{NG} are to provide constructs for a *declarative* and, therefore, concise and quite simple specification of dynamic document rendering by comparison to query languages like XSLT [Cla01] or scripting languages like ECMA Script [ECM99].

The main goal of CSS^{NG} is to make scripting languages unnecessary for as many applications as possible such as the visualization of query languages like Xcerpt [SB04] or the visualization of RDF [LS99] graphs such as FOAF [BM05] declaration. At the same time CSS^{NG} is supposed to be a rather limited and conservative extension of CSS 3. Nonetheless CSS^{NG} should make possible

- to specify dynamic styling,
- to generalize markup visualization, and
- to integrate the keyboard as input device.

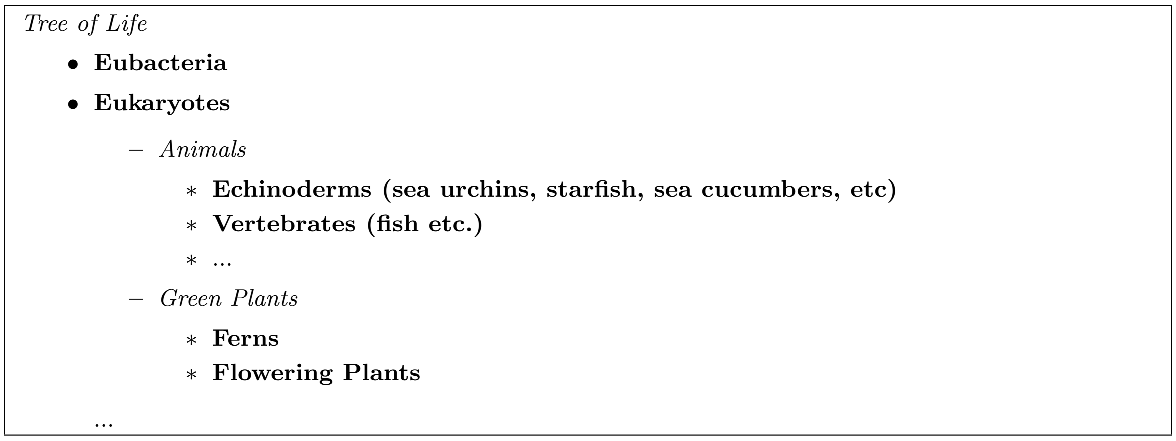
Two types of extensions can be distinguished: extensions of the already existing 'static styling' features and extensions towards additional 'dynamic styling' features. Static styling refers to styling without action of a viewer. For instance, all portions of a structured text that are marked up as headings are rendered using bold letters. In contrast, dynamic styling denotes styling as a consequence of a viewer's interaction. An example of dynamic styling supported by the currently implemented style sheet language CSS 2.1 [BLLJ98] is as follows: A viewer can change the styling of a portion of text in a Web browser by letting the mouse cursor hover over such a text portion. In other words, static styling refers to a styling that remains unchanged during viewing, whereas dynamic styling refers to a styling that might change as a consequence of a viewer's action during viewing.

1.4 Advantages of CSS^{NG}

CSS^{NG} achieves these objectives by static and dynamic styling extensions to CSS 3. Let us briefly discuss four kinds of sample applications of CSS^{NG} that benefit from the extensions proposed later in this thesis (see Chapter 4).

Styling of Subelements at Arbitrary Nesting Depths

The following HTML document shows an excerpt of the highly nested 'Tree of Life'[2], a well known classification of living organisms.



Tree of Life

- **Eubacteria**
- **Eukaryotes**
  - *Animals*
    - **Echinoderms (sea urchins, starfish, sea cucumbers, etc)**
    - **Vertebrates (fish etc.)**
    - ...
  - *Green Plants*
    - **Ferns**
    - **Flowering Plants**

...

Figure 1.6: Tree of Life.

The items of the tree are styled in alternating manner as follows: Items having an odd nesting depth are styled using bold font, and items having an even nesting depth are styled using italic font. Obviously, the alternating styling of fonts depending on the nesting depth helps to recognize the structure of the tree and helps to compare different parts of the document.

Such a styling would also be useful for applications such as threads in a discussion forum, but it is not possible with current CSS. Current CSS allows the styling of items on a given depth only. Hence, for a correct alternating styling in current CSS, the maximal nesting depth of a document must be known, and one rule for each level of nesting depth until the maximum level is necessary as the following CSS program in Fig. 1.7 illustrates.

[2] `http://tolweb.org/tree/home.pages/popular.html`

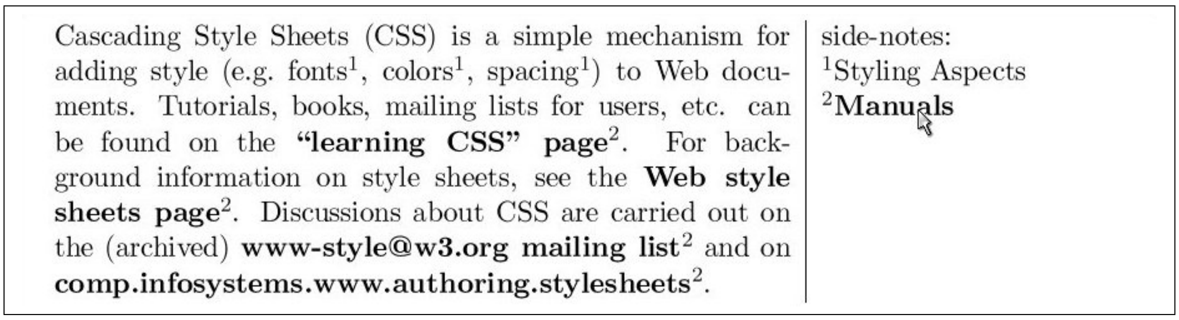
```
*            { font-weight : bold;   }
* *          { font-style  : italic; }

* * *        { font-weight : bold;   }
* * * *      { font-style  : italic; }
...

* * * ... *   { font-weight : bold;   }
* * * * ... * { font-style  : italic; }
```

Figure 1.7: Styling until a certain depth in CSS.

As proposed in this thesis, see Section 4.4, extensions make possible an alternating styling independent of nesting depth. The extensions provide styling rules that can be applied repeatedly depending on the nesting depth. Therefore the nesting depth of a document does not need to be known while writing its style sheet.

Dynamic Styling of Semantically Related Text Portions

Footnotes are often used to annotate text portions. In the Web context, so called 'side-notes' are located beside the text like in the following example.



Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts[1], colors[1], spacing[1]) to Web documents. Tutorials, books, mailing lists for users, etc. can be found on the **"learning CSS" page**[2]. For background information on style sheets, see the **Web style sheets page**[2]. Discussions about CSS are carried out on the (archived) **www-style@w3.org mailing list**[2] and on **comp.infosystems.www.authoring.stylesheets**[2].

side-notes:
[1]Styling Aspects
[2]**Manuals**

Figure 1.8: Highlighting of semantically related text portions.

Hovering with a mouse cursor over a side-note as in the example above can cause a highlighting of every semantically related text passage somewhere else in the document. Such a rendering would provide good support for an easier and better understanding while reading texts.

Current CSS or XSL-FO does not support such highlighting of related text portions. Helper tools beyond CSS and XSL-FO such as scripting languages (for instance ECMA script [ECM99]) are needed to change the styling while viewing a document in a Web browser. A slight extension to CSS introduced in Section 4.6 allows a declarative way to define such dynamic highlighting of semantically related text portions within a style sheet language without additional tools.

Static Visualization of (Parts of) Markup Itself

Viewing XML data such as a timetable of trains in a text editor asks for a more concise representation (see left column of the following example). An extension proposed in this thesis (see Section 4.2) can be applied to render markup. Each XML-tag (such as <train>) can be rendered in a Web browser as demonstrated in Fig. 1.9.

Source	**Rendering**
`<trains>`	• trains
`  <train number="ICE788">`	– train (number ICE788)
`    <departure>`	∗ departure
`      <station>Munich</station>`	· station Munich
`      <time>11:36</time>`	· time 11:36
`    </departure>`	
`    <arrival>`	∗ arrival
`      <station>Hamburg</station>`	· station Hamburg
`      <time>17:45</time> ...`	· time 17:54 ...
`    </arrival>`	

Figure 1.9: Rendering of markup.

While current CSS allows to render known XML tags and known XML attributes only, XSLT allows the visualization also of unknown XML tags. However, such XSLT programs tend to get rather complex, as the following XSLT program transforming the train timetable to HTML illustrates. In this short version of the program, the rendering of attributes is not considered for simplicity reasons.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="*">
    <ul>
      <xsl:for-each select=".">
        <li>
          <xsl:value-of select="name()" />
          <xsl:apply-templates />
        </li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

Figure 1.10: XSLT program rendering the XML elements of Fig. 1.9.

The extensions proposed in this thesis offer a more declarative (hence easier to use) way to render such marked up data as introduced in Section 4.2.

Dynamic Styling of Text Portions by Folding and Unfolding Them

As data like the train timetable of the latter example can become rather long, a better overview of the data is highly welcome. Folding undesired data and unfolding desired data helps to find information more easily.

- train (ICE 788)
  - departure
    - station Munich
    - time 11:36
  - arrival
- train (ICE 586)
- train (ICE 584)
- train (ICE 784)
  - departure
    - station Munich
    - time 15:44
  - arrival
- ...

Figure 1.11: Folded (`train (ICE 586)`) and unfolded (`train (ICE 784)`) data items.

In the example in Fig. 1.11 the viewer has folded sub-items of the already known train timetable by clicking with a mouse cursor on the parent item (the train item). All items except of two train items are folded. Now, the viewer can compare the information about these two trains. Since the viewer is interested in departure times, all other information about the two trains are folded. This rendering makes the comparison of the departure times easy. The extensions allowing this rendering are introduced in Section 4.5.

Note that all four kinds of sample application mentioned afore are highly useful in practice. For instance, in office working, tasks like the orientation in large documents, the analysis of spreadsheets, or the administration of databases should take rather short time. Hence, the rendering of such data should be optimized for human beings to permit a faster recognition of facts and their contexts. The extensions proposed in this thesis offer a declarative way to define such an optimized rendering.

The static and dynamic extensions to style sheet languages proposed in this thesis are in essence compatible with any style sheet language. In this thesis these extensions are worked out in CSS. CSS has been chosen because it is convenient to draw on Web standards to reduce implementation effort. CSS has been chosen instead of XSL, for the following reasons:

- CSS is widespread because of its simple rule-based syntax. Thanks to this rule-based syntax only minor extensions of CSS are needed, whereas for style sheet languages such as XSL-FO, more significant extensions would be needed.
- CSS is implemented in almost every common Web browser and thus it can be easily used from different environments.

CHAPTER
TWO

Cascading Style Sheets (CSS): A brief Introduction

This chapter gives a brief introduction into the W3C's Style Sheet Language *Cascading Style Sheets (CSS)*. Concepts of CSS serving as basis for the extensions proposed in this thesis (such as CSS selectors) are addressed in detail. For details on the remaining concepts of CSS (such as font styling or positioning) refer to the full reference of CSS [BLLJ98]. Major parts of this chapter are taken over from a project thesis [Wie05] for a sound report of the work on CSS^{NG}.

2.1 The Origin of CSS

Separating content from design was already one of the basic principles of HTML. One of the first Web browsers, the NeXT browser of Tim Berners-Lee, already had a built-in style sheet language. This language allowed to determine the appearance of Web pages. Other Web browsers like Viola (1992) or Harmony (1993) implemented their own style sheet languages as well but all of these approaches have in common that only the implementor of the current Web browser could influence the appearance of Web pages via a style sheet language. This fact provoked the resentment of Web page authors as can be derived from the following posting of Marc Andreessen, one of the programmers of the NCSA Mosaic Web browser and founder of Netscape, in the www-talk[1] mailing list in February 1994:

> In fact, it has been a constant source of delight for me over the past year to get to continually tell hordes (literally) of people who want to – strap yourselves in, here it comes – control what their documents look like in ways that would be trivial in TeX, Microsoft Word, and every other common text processing environment: "Sorry, you're screwed."

Eight months later in October 1994, Håkon Wium Lee published the first draft of Cascading HTML Style Sheets [Lie94] to satisfy the needs of Web page authors. The at that time novel principle of cascading styling specifications (see Section 2.8) allowed to provide a specification of the appearance of Web pages by the author, which could be overwritten by the viewer of the Web page where wanted.

[1]`http://ksi.cpsc.ucalgary.ca/archives/WWW-TALK/www-talk-1994q1.messages/643.html`

The first one to join Håkan Wium Liu in the work on CSS was Bert Bos. In 1994 he developed a highly customizable Web browser called Argo with its own styling language. In contrast to Cascading HTML Style Sheets Argo implemented already attribute selectors (see Fig. 2.5) and the generation of text, one of the starting-points for extending CSS in this thesis.

Other languages like the styling languages of the Web browsers mentioned above as well as DSSSL for styling SGML[2] documents were available at that time. However, none of these languages allowed the cascading feature of CSS providing styling with respect to the wishes of the Web page author, the capabilities of the device for viewing, and the preference of the user/viewer. CSS was not only commended but also criticized for its simplicity at that time. It was not clear that CSS could master future styling requirements without more powerful constructs. In 1996 Netscape proposed JSSS[3], an Java Script-based[4] styling language. JSSS was implemented in the Netscape Navigator 4 but was of little importance in practice[5].

The WWW conference in 1995, held in Darmstadt (Germany), was an important event: CSS level 1 [BW96] (short: CSS 1) was presented and the CSS working group was founded. Since that conference CSS gained in importance. The industry participated in the W3C and implemented its standards in their products and CSS became widespread in Web technology.

CSS 1 focused on basic styling feature like font style, colors or text alignment. Having gained experience with CSS 1 the CSS working group proposed CSS level 2 [BLLJ98] in 1998. Among various styling features listed in [BWLJ96] were now also dynamic styling features and an interface for inserting text. Both features are the main foundations of extensions proposed in this thesis. Currently CSS 3 [Bos05] is about to reach the status of a W3C recommendation. Refer to [LB99] for a detailed history of CSS.

2.2 Separating Design from Content using CSS

CSS has been developed to simplify changes of the content as well as of the design of HTML documents by separating design from content. As a consequence of this separation minor changes on an HTML document (e.g., another styling for headings) require minor changes on a CSS style sheet. Without separation of design from content such minor changes cause a complete revision of an HTML document so as to modify, say, each heading in an HTML document.

According to the W3C recommendation “Associating Style Sheets with XML documents” [Cla99] CSS style sheets can also be used to render arbitrary XML documents and not only HTML documents, using an XML processing instruction (PI) , see Fig. 2.1

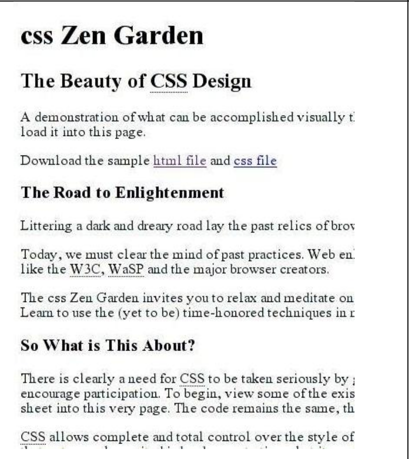
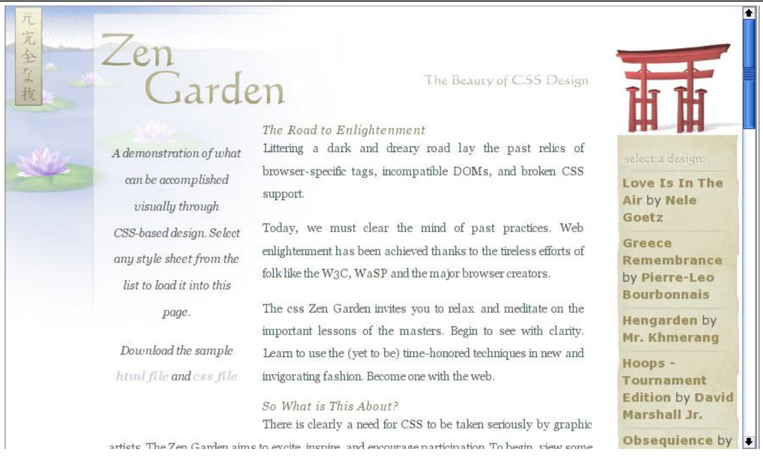
```
<?xml-stylesheet type="text/css" href="stylesheet.css" ?>
```

Figure 2.1: XML Processing Instruction (PI) for using a CSS style sheet in an XML document.

As the Fig. 2.2 illustrates, CSS is capable of styling that is not possible with standard HTML means. On the left side, the HTML page ‘CSS Zen Garden’[6] is rendered without a user defined style sheet according to the guidelines of the HTML specification [PAA$^+$00]. On

[2]ISO 8879:1986
[3]http://www.w3.org/Submission/1996/1/WD-jsss-960822
[4]http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference
[5]http://virtuelvis.com/archives/2005/01/css-history
[6]http://www.csszengarden.com/zengarden-sample.html

the right side, the same HTML document is rendered using a CSS style sheet. (Note that the excerpt of the original rendering on left side shows less text by comparison to the right side only because of layout reasons.) Obviously, one HTML page can be styled by various style sheets. Currently, more than 600 additional style sheets[7] are available for this example. Voluntary designers wrote these style sheets on request of Dave Shea, a web designer from Vancouver (Canada), to demonstrate the advantages of CSS.

HTML Rendering:	**CSS Rendering in a Web browser:**
	

css Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished visually t
load it into this page.

Download the sample html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of brov

Today, we must clear the mind of past practices. Web en
like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on
Learn to use the (yet to be) time-honored techniques in r

So What is This About?

There is clearly a need for CSS to be taken seriously by
encourage participation. To begin, view some of the exis
sheet into this very page. The code remains the same, th

CSS allows complete and total control over the style of

Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.

Download the sample html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This About?

There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some

select a design:

Love Is In The Air by **Nele Goetz**

Greece Remembrance by **Pierre-Leo Bourbonnais**

Hengarden by **Mr. Khmerang**

Hoops - Tournament Edition by **David Marshall Jr.**

Obsequience by

Figure 2.2: Example on separating design from content using CSS.

Currently, the latest CSS version is CSS 2.1. The next version CSS 3 is about to receive the status of a W3C recommendation, which is in fact a standard. Although CSS 3 is not yet a recommendation of the W3C, we will also draw on concepts of CSS 3 to reduce reimplementation efforts after CSS 3 is published.

2.3 Structure of CSS

A Cascading Style Sheet consists of a sequence of CSS rules. Each rule is separated into two parts: The head of a rule is called *selector* and the body of a rule is called *declaration*. The main structure of a CSS rule is given in EBNF syntax as follows:

```
RULE        ::= SELECTOR "{" DECLARATION "}" .
DECLARATION ::= ( TUPLE )* .
TUPLE       ::= ( PROPERTY ":" VALUE ";" ) .
```

Figure 2.3: Main Structure of a CSS Rule in EBNF.

The `SELECTOR` part of a rule matches (zero or more) XML nodes, and the `DECLARATION` part specifies the styling of the currently selected XML nodes. Concrete aspects of styling are addressed in the `DECLARATION` part as so called `TUPLE`s. Each of these `TUPLE`s consists of a `PROPERTY` (such as `font-weight` or `background-color`) combined with adequate `VALUE`s (such as `12pt` or `white`).

[7]`http://www.mezzoblue.com/zengarden/alldesigns/`

Comments in CSS are marked by "`/*`" as opening tag and "`*/`" as closing tag. Comments can appear everywhere in a CSS style sheet.

The following figure demonstrates, how the repository of a library written in XML can be styled using CSS:

Source:

```
<bib>
  <book year="1994">
    <title>
      TCP/IP Illustrated
    </title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>
      Addison-Wesley, 1994
    </publisher>
    <price>65.95</price>
  </book>
  ...
</bib>
```

CSS Style Sheet:

```
1  *             { display         : block; }
2  bib           { list-style-type : decimal;
3                  margin          : 1em 0;
4                  padding-left    : 40px; }
5  book          { display         : list-item; }
6  author        { font-style      : italic; }
7  first, last   { display         : inline; }
8  price::before { content         : "EUR "; }
```

Rendering:

1. TCP/IP Illustrated
   Stevens W.
   Addison-Wesley, 1994
   EUR 65.95

...

Figure 2.4: Example: Rendering of XML data using CSS.

The rule in the **first line** of the CSS style sheet above selects all XML nodes by the wildcard pattern (`*`). Hence, all XML nodes are primarily styled as specified in the declaration part of that rule. In this case, all elements are arranged in blocks (and not as continuous text). In the **second line**, the `bib` element is defined as container for a numbered list with a concrete indentation of `40px` and a margin around the list. This numbered list consists of `books` stated in the **fifth line**. In the **sixth line**, each of the `authors` is rendered in italic style but unlike the other elements the `first` and the `last` name are arranged in a row in **line seven**. The **eighth line** causes the String `"EUR "` in front of each price.

The following sections introduce to selected concepts of CSS in detail. In particular, concepts being extended in Chapter 4 like the CSS selector concept are addressed.

2.4 Simple Selectors

The basis of CSS selectors are so called *simple selectors*. As already applied in Section 2.3, simple selectors can select XML elements by pattern matching: The most general pattern is the star (`*`) which matches every XML element. A selection can be restricted to XML elements bearing the same name. For instance, the selector `author` instead of the star (`*`) selects `author` XML elements only. Further on, selections can be restricted to attribute names and values. The following rule matches all `book` elements having an XML attribute `year` with `1994` as value:

```
book[year="1994"] { ... }
```

Figure 2.5: Example: Rendering of XML data using a Simple Selector.

2.5 Combinators

Being able to select individual types of XML elements using simple selectors is a first step. The next step is to select XML elements being in a relationship to other XML elements. Two simple selectors can by related by the following CSS combinators:

`E F`	Matches any F element that is a descendant of an element E.
`E > F`	Matches any F element that is a child of an element E.
`E + F`	Matches any F element immediately following an element E in the so-called document order.
`E ~ F`	Matches any F element following an element E in the so-called document order.

Figure 2.6: CSS Combinators.

The following rule selects all `title` elements within `book` elements. Other `title` elements having no ancestor node `book` are not selected:

```
book title        { ... }
```

Figure 2.7: Example: Rendering of XML data using a Combinator.

2.6 Grouping

Obviously, XML elements of different types can be rendered using the same CSS `declarations`. Hence, each type of an XML element such as `first` or `last` can be rendered using its own rule like in the first and the second line of the following example:

```
first         { display : inline; }
last          { display : inline; }

/* Grouping of the previous rules */
first, last   { display : inline; }
```

Figure 2.8: Grouping of CSS rules.

To simplify writing of CSS style sheets and to simplify changes on a set of elements, CSS allows grouping of elements. In the fifth line of the example above (see library example 2.4) the CSS rules of the lines one and two are grouped together.

Note, that the grouping operator "`,`" can be easily confused with CSS combinators introduced in Section 2.5.

2.7 Pseudo-selectors

Selecting the first (or n-th) child of an XML element cannot be expressed by applying simple selectors and combinators only. Therefore, CSS offers pseudo-selectors for selections that go beyond the expressive power of simple selectors and combinators.

In this section both types of pseudo-selectors are introduced: so called ‘pseudo-elements’ and ‘pseudo-classes’ . Pseudo-elements address selections concerning the “surroundings of one XML element” such as the first letter of a text node. Pseudo-classes refer to the context of an XML element that cannot be expressed by simple selectors and combinators only, such as selecting XML elements being the third child in XML document order [BPSMM00].

2.7.1 Pseudo-elements

According to the specification of CSS [BLLJ98], pseudo-elements allow ‘language specific’ (e.g., HTML specific) selections such as the selection of the first letter or the first line of a paragraph. For instance, texts can be rendered using majuscules, where the first character of a paragraph is rendered differently from the other characters. (This was common in medieval texts.) Since the first character of a paragraph is usually not surrounded by XML tags, it cannot be selected by a simple selector. The pseudo-element `::first-letter` makes such a styling possible.

Beside selecting text portions inside an XML element like the first letter or the first line, pseudo-elements can also select ‘virtual’ XML text nodes before opening XML tags or after a closing XML tags. The following example renders the XML element `price` with additional text defined in the style sheet.

Source (extract):

```
  </publisher>
           <!-- virtual XML text node -->
  <price>65.95</price>
           <!-- virtual XML text node -->
</book>
```

CSS Style Sheet (extract):

```
price::before { content : "EUR "; }
```

Rendering:

EUR 65.95

Figure 2.9: Selecting ‘virtual’ XML text nodes.

The CSS rule on the right side (taken from the library style sheet given in Figure 2.4) demonstrates the insertion of text: The currency `EUR` is inserted in front of the price of a book using the pseudo-element `::before`. Besides `::before`, the pseudo-element `::after` allows insertions of text behind a rendered XML element. In both cases, the inserted content is part of the style sheet and not part of the source document (unlike all other content such as `65.95`). Hence, another style sheet could insert the currency symbol € instead of `EUR`.

2.7.2 CSS Functions

Besides the insertion of fixed text, CSS functions grant *limited access* to the markup of a source document. For instance, the rendering of HTML ordered lists (`<ol>...</ol>`) demands to insert the current number of each list item. CSS offers the following functions to afford insertions depending on the source document:

- `counter(name)`, `counter(name, style)`, and `counter(name, string, style)` insert the numbers of enumerations.
- `attr(X)` returns the value of an XML attribute `X` to the subject of the CSS selector as string. If the subject of the selector does not have an attribute X, an empty string is returned.

These CSS function are used within the `content property` of a CSS rule in combination with the pseudo-Elements `::before` or `::after`. Referring to the library example (see Section 2.4) the XML attribute `year` of a `book` could be rendered using the following additional CSS rule:

Source (extract):

```
<bib>
  <book year="1994">
    <title>
      TCP/IP Illustrated
    </title>
  <!-- ... -->
```

CSS Style Sheet (extract):

```
book::before{content:"published "attr(year)": ";}
```

Rendering:

1. published 1994: TCP/IP Illustrated
   ...

Figure 2.10: Example: Rendering of XML data using CSS.

Note, that neither of the offered CSS functions cover computational capabilities.

2.7.3 Structural Pseudo-classes

CSS introduces the concept of structural pseudo-classes to permit *"selections that are based on information lying in the document tree but cannot be represented by other simple selectors or combinators"* [BLLJ98].

The lines of a rendered list such as in the library example (see Section 2.4) can be rendered using alternating colors to support a better orientation. Since the (even or odd) position of a list item cannot not be derived using simple selectors, Web designers use XML attributes to mark up even and odd items in CSS 2.1. CSS 3 offers structural pseudo-classes that can be used for a rendering depending on the relative position of a list item. The following CSS rule renders the first item in a bold font (line one) and turns the background color of every second book item into gray beginning with the second one (line two):

```
1 bib:nth-child(2n+1)   { font-weight      : bold; }
2 bib:nth-child(2n+2)   { background-color : gray; }
```

Figure 2.11: Structural Pseudo-classes.

The selector `:nth-child(An+B)` matches an XML element that has "$A \cdot n + B - 1$ siblings before it" [Bos05] in the XML document tree. In other words, the selector matches every `A`-th XML element beginning with the `B`-th sibling. While `A` and `B` stand for integers, `n+` is a reserved word. This thesis will refer to the expression `An+B` as *recurrence patterns*. Beside the `:nth-child(An+B)` selector, other selectors of this category allow to match the XML document root and siblings beginning with the last XML element. We refer to the CSS reference [Bos05] of the W3C for a detailed reference on the remaining structural pseudo-classes.

2.7.4 Dynamic Pseudo-classes

In contrast to pseudo-elements and structural pseudo-classes, dynamic pseudo-classes denote all selections that cannot be deduced from the document tree, e.g. selections that depend on interaction of the viewer.

In a Web browser window dynamic rendering of hyperlinks can be observed on many Web pages[8]. Hovering with the mouse cursor[9] over a hyperlink anchor can change its styling such as underlining the link. Removing the mouse cursor from the hyperlink anchor again activates the former styling. The following CSS rule causes such a dynamic rendering:

```
a:hover       { text-decoration : underline; }
```

Figure 2.12: Dynamic Rendering.

Besides the `:hover` dynamic pseudo-class CSS offers the dynamic pseudo-classes `:active` and `:focus`. The pseudo-class `:active` selects the hyperlink that points to the current Web page itself. Finally, the `:focus` pseudo-class selects the area where the text cursor is focused on. In XHTML this could be a text field of a form.

2.8 Interpretation of CSS style sheets

The basic interpretation concept of CSS programs is based on the so called 'cascading styling' . Similar to the concept of inheritance in object oriented languages (such as Java or C++) the rendering of the parent XML element is 'inherited' by each child XML element. Referring to the following example (see library example of Fig. 2.4) the `first` name and the `last` name of each `author` are rendered using an italic font-style, although no CSS rule declaring the font style matches these XML elements.

Source (extract):

```
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
```

CSS Style Sheet (extract):

```
author   { font-style : italic; }
```

Rendering:

Stevens W.

Figure 2.13: Cascading Styling.

Further on, cascading styling allows to modify the scope of a CSS rule. To modify, e.g., the rendering of the `last` name in the example above, an additional CSS rule matching the `last` name can be introduced:

Source (extract):

```
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
```

CSS Style Sheet (extract):

```
author  { font-style : italic; }

/* Additional CSS-Rule: */
last    { font-style : normal; }
```

Rendering:

Stevens *W.*

Figure 2.14: Overwriting CSS rules in depth.

[8]see `http://www.ifi.lmu.de`

[9]A mouse cursor is controlled by a *mouse device* like a 'mouse', a 'touch pad', or a 'track ball'.

According to the figure above the `last` name of the `author` is styled in 'normal' font style and not in italic font style as defined in line one of the style sheet. Hence, the declaration of the parent XML element `author` is shaded.

CSS rules can be shaded not only in depth such as in the latter paragraph but also in breadth. Referring the example in 2.15, a third rule could overwrite the styling of the `last` name as follows:

Source (extract):

```
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
```

CSS Style Sheet (extract):

```
author  { font-style : italic; }
/* Additional CSS-Rules: */
last    { font-style : normal; }
last    { font-style : italic; }
```

Rendering:

Stevens W.

Figure 2.15: Shading CSS rules.

The rule in line four of the style sheet frame above, shades the previous rule. As a consequence, the `last` name of the `author` is rendered in italic font style.

In CSS, the most specific rule is used for the rendering. Obviously, the most specific rule in depth is the rule whose selector is the tag name of the XML element itself or of the nearest ancestor XML element. In breadth, the most specific rule is the last one matching the same XML element.

2.9 Styling at Unknown Depth

CSS allows styling at unknown depth. Simple selectors allow to select XML elements independent of depth (see Section 2.4). Furthermore, according to the cascading styling paradigm (see Section 2.8) each XML element is styled depending on the most specific rule. For instance, the `author` rule of the library example (see Fig. 2.4) defines a default styling of all `author` XML elements independent of depth. This styling is inherited by all descendant XML elements independently of depth such as `fist` and `last`.

CHAPTER
THREE

Limitations of CSS 2.1 and CSS 3

CSS 2.1 and CSS 3 both offer many facilities for a sophisticated rendering of semi-structured data (see Section 2: 'CSS Zen Garden'). This section shows limitations of CSS, which the extensions proposed later in this thesis try to overcome.

3.1 Transformation vs. Rendering

A widespread argument against CSS and in favor of XSLT [Cla01] is that CSS, in its current versions including CSS 3, does not allow significantly to *transform* the structure of a document and instead only offers primitives for adding style to structured data while, besides limited changes, keeping the document's original structure unchanged.

The author believes that this limitation of CSS is one of its appealing features. The reason is that *transforming* up to significantly restructuring a (structured) document is a task that

1. basically is not part of *rendering* and

2. is not only needed for *rendering.*

It is the author's conviction that if a rendering requires a significant restructuring, then it is preferable to specify this restructuring independently from rendering. Obviously, this leads to clearer programs and therefore to programs that are easier to maintain. (This is often called *separation of concerns* and is a key objective of 'good' programming.)

If significant restructuring should, in the authors' opinion, be kept separated from rendering, *slight transformations* of a document's structure are surely desirable within rendering specifications, e.g.

1. **"deleting"**, or merely "hiding" an **XML element** (and its sub-elements)

2. **"deleting"**, or merely "hiding" parts of **XML text nodes**

3. **adding** XML **text nodes** without iteration , and

4. **adding** XML **elements** without iteration.

Adding without iteration means that recursively specified addition of XML elements should be precluded. This restriction prevents the specification of infinite documents in a CSS style sheet. Refer to Section 3.1.4 for more details.

Note that deleting an XML element does not affect the *source document* but the *visualization* of the XML element only. However, the view of deleting is applicable to a (virtual) *intermediary step* of the rendering process. This intermediary step is the result of slight transformations caused by adding and deleting of markup. For instance, an XML element that is added by a CSS style sheet would appear in the intermediary step while the source document does not change. Consequently, the XML document of the intermediary step is rendered (instead of rendering the source XML document directly). In the following we will use 'deleting' in this sense.

As demonstrated in the table below, the rendering specifications 'deleting XML elements' and 'adding XML text nodes' are already possible in current CSS. However, deleting XML text nodes and adding XML elements is not possible in current CSS, and there seems to be no reason for this asymmetry.

	Adding	Deleting
XML text nodes	yes	no
XML elements	no	yes

One would expect that adding and deleting are permitted or prohibited line by line. The following sections address each value of the table above in detail:

3.1.1 Deleting XML elements

Deleting XML elements is supported in CSS 2.1 and CSS 3: The CSS declaration `display:none` prevents the rendering of all XML elements that are matched by the corresponding CSS selector of a CSS rule, as demonstrated in the following figure:

Source (extract):

```
1     <author>
2       <last>Stevens</last>
3       <first>W.</first>
4     </author>
```

CSS Style Sheet (extract):

```
1  first   { display : none; }
```

Rendering:

```
Stevens
```

Figure 3.1: Deleting an XML element.

The `first` name 'W.' of the `author` in the Figure above is not displayed because of the rule in line 1 of the style sheet above. (Since CSS inherits styling declarations to sub-elements of a XML element, see Fig. 2.13, the sub-elements inherit the declaration `display:none`. Consequently, these sub-elements are not rendered except another CSS rule overwrites this declaration, see Fig. 2.14.)

3.1.2 Deleting parts of XML text nodes

Deleting parts of XML text nodes is not possible in current CSS. Although text nodes can be selected using pseudo-elements (see Section 2.7.1) such as `::first-letter`, XML text nodes cannot be deleted because the CSS `display` property cannot be applied to *'pseudo-selections'* [BLLJ98]. Hence the CSS rule `author::first-letter { display: none; }`

is not possible in current CSS. The author is not aware of any reason for this restriction because this transformation is more limited than *deleting XML elements*, which is supported by current CSS as discussed above.

Deleting XML text nodes would be an appealing feature of CSS. For instance, all text portions surrounded by parentheses could be deleted to get a compact version of a text, for instance, for rendering on a portable device with small screen. Such a selection would be possible using extended pseudo-elements to select arbitrary XML text nodes (instead of only selecting the first letter or the first line like in current CSS). A new pseudo-selector based on *Regular Expressions* (e.g., expressed in the POSIX[1] syntax) can afford such selections. This extension goes beyond the scope of this thesis and is therefore not addressed here.

3.1.3 Adding XML text nodes

Adding XML text nodes is possible in current CSS. The pseudo-elements `::before` and `::after` allow to specify the insertion of context. XML text nodes can be inserted before or after an XML element as illustrated with the currency symbol `Euro` in Section 2.7.1.

3.1.4 Adding XML elements

Adding XML elements is not possible in current CSS (see Section 2.7.1). However, adding XML elements would be a winning feature of CSS because styling would become more flexible. For instance, tabs could be added to data items (such as `person`, `name`, etc., see Fig. 3.2). If realized with XML elements (instead of XML text nodes), CSS rules could be used to render the tabs. In particular, dynamic rendering could enrich the facilities of such tabs: For instance, a mouse click on a tab could fold the data item (and its subtree) and a second mouse click could reopen the tab.

```
<tab>person</tab>
<person>
   <tab>name</tab>
   <name>Igor</name>
   <tab>sharedMusic</tab>
   <sharedMusic>
      <tab>music</tab>
      <music>
         <tab>title</tab>
         <title>carry</title>
```

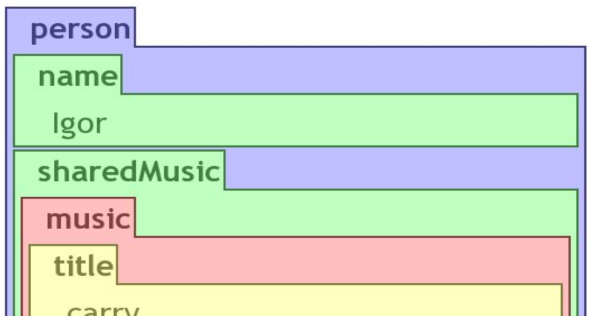


Figure 3.2: Adding XML Elements as Tabs.

Arguably, adding XML elements is not allowed in current CSS because of the following reasons:

As first reason, the *well-formedness of the rendering* of an XML document would be dependant on the style sheet. (The rendering does not affect the source document, see Section 3.1.) For instance, an opening XML tag is inserted by a CSS rule but the corresponding closing XML tag is not inserted because there is no CSS rule inserting it:

[1]`http://www.pasc.org/plato/`

Source (extract):

```
  </publisher>
            <!-- virtual XML text node -->
  <price>65.95</price>
            <!-- virtual XML text node -->
</book>
```

CSS Style Sheet (extract):

```
price::before {content:"<strong>";}

/* Not applied:
price::after {content:"</strong>";}
*/
```

Intermediary Step (extract):

```
  </publisher>
  <strong><price>65.95</price>
  <!-- missing closing tag -->
</book>
```

Rendering:

```
not defined
```

Figure 3.3: Invalid insertion of XML elements.

As a consequence, the source document might not be rendered like in the (hypothetic) example above. The intermediary step in the rendering process (including the added XML element) cannot be rendered because rendering of invalid XML documents is not defined in current CSS.

As second reason, an XML element could be *added iteratively* using recursive CSS rules. The following (hypothetic) example demonstrates the declaration of such repetitively applicable CSS rules:

Source (extract):

```
1   </publisher>
2             <!-- virtual XML text node -->
3   <price>65.95</price>
4             <!-- virtual XML text node -->
5 </book>
```

CSS Style Sheet (extract):

```
1 price::before {content:"<price />";}
```

Intermediary Step (extract):

```
1   </publisher>
2   <price />...<price /><price>65.95</price>
3 </book>
```

Rendering:

```
not defined
```

Figure 3.4: Recursive Insertion of XML elements.

The XML element <`price />` is added recursively because the CSS rule in line 1 of the CSS style sheet adds an XML element, that is styled by the rule itself. Obviously, <`price />` is inserted infinitely. Therefore, the intermediary step would result in an infinite XML document, and thus the rendering would be undefined. It is assumed that added XML elements are styled using the CSS rules of the same CSS style sheet[2].

As third reason, iterative rendering processes should not be allowed (even if terminating) because primitives like *adding* and *deleting* of XML elements allow significant restructuring (as discussed above). For instance, XML elements can be grouped by criteria such as the content of an XML attribute or an XML element as demonstrated in Fig. 3.5. The XML

[2]Obviously, adding XML elements that are not being styled would make no sense.

document on the left side can be transformed to the XML document on the right side (and vice versa) using adding and deleting XML elements iteratively.

Trains Grouped by Time of Day:

```
<trains timeOfDay="AM">
  <train id="ICE111">
    <departure>
      <station>Munich</station>
      <time timeOfDay="AM">11:36</time>
    </departure>
  </train>
</trains>

<trains timeOfDay="PM">
  <train id="ICE333">
    <departure>
      <station>Hamburg</station>
      <time timeOfDay="PM">19:47</time>
    </departure>
  </train>
  <train id="ICE888">
    <departure>
      <station>Munich</station>
      <time timeOfDay="PM">21:48</time>
    </departure>
  </train>
</trains>
```

Trains Grouped by City of Departure:

```
<trains departure="Munich">
  <train id="ICE111">
    <departure>
      <station>Munich</station>
      <time timeOfDay="AM">11:36</time>
    </departure>
  </train>
  <train id="ICE888">
    <departure>
      <station>Munich</station>
      <time timeOfDay="PM">21:48</time>
    </departure>
  </train>
</trains>

<trains departure="Hamburg">
  <train id="ICE333">
    <departure>
      <station>Hamburg</station>
      <time timeOfDay="PM">19:47</time>
    </departure>
  </train>
</trains>
```

Figure 3.5: Grouping of XML Data (Intermediary Step).

In the left window the trains are grouped according to their time of departure. Hence, `ICE111` belongs to the first group (because it departs before noon) and the remaining trains `ICE333` and `ICE888` belong to the second group (because they depart after noon). Obviously, the XML document in the right window is significantly restructured by comparison to the left window. The restructuring is the result of a grouping by the city of departure instead of a grouping by the time of day. Therefore, `ICE111` and `ICE888` are grouped together on the right window instead of `ICE333` and `ICE888`.

If CSS allowed recursion, such a significant restructuring could be implemented using adding and deleting XML elements as follows: For each group iterate over all XML elements of the source document and add each matching XML element (including its subelements) to a 'container XML element' of the group. As a consequence, if iterative rendering processes admitted such restructuring, CSS would be extended to the capabilities of Turing-complete transformation languages such as XSLT or XQuery [Kep04]. Hence, the posited *separation of concerns* (see Section 3.1) would be abandoned.

The restriction of forbidding iterative adding of XML elements complies with the author's conviction concerning *slight transformations* of semi-structured data instead of a significant restructuring (see Section 3.1). However, adding XML elements not iteratively should be allowed in CSS (as motivated by additive XML elements serving as tabs). Refer to Section 4.2 for concrete extensions.

3.2 Insertion of Markup

The insertion of XML elements using CSS pseudo-elements as demonstrated in Section 3.1.4 (Fig. 3.4) seems to be natural. Indeed, as explained in Section 2.7.1, CSS 2.1 allows to insert arbitrary characters (see encoding schemes UTF-8, ISO-8859-15, etc.) using pseudo-elements. Since markup is composed of XML 'control characters' such as < or >, XML tags can be inserted, too, as demonstrated in the following example:

Source (extract):

```
  </publisher>
    <!-- virtual XML text node -->
  <price>65.95</price>
    <!-- virtual XML text node -->
</book>
```

CSS Style Sheet (extract):

```
price::before { content : "<new>&euro; "; }
price::after  { content : "</new>"; }
new           { font-weight : bold; }
```

Expected Rendering:

€ 65.95

Actual Rendering:

<new>&euro; 65.95</new>

Figure 3.6: Insertion of XML Control Characters.

The rendering frame on the right side of the figure above illustrates the effect of the insertion of <`new`> and </`new`> on the rendering. Obviously, the characters < and > are not interpreted as expected (see rendering frame on the left side). Even the XML entity `&euro;` is not rendered as expected. Hence, XML elements cannot be added by this construct and rendered by CSS 2.1 or CSS 3 like elements in XML source documents. Other constructs to insert XML elements to an XML document are not available in CSS.

3.3 Visualization of Markup

In current CSS, markup being inserted (as discussed in the latter section) as well as markup of a source XML document can be visualized. Pseudo-elements (see Section 2.7.1) and the CSS property `content` can define such a rendering using the selectors `::after` and `::before`. For instance, the markup of the train example (see Chapter 1) can be implemented in current CSS as follows:

Source:	**Rendering:**
`<trains>`	● trains
`<train id="ICE788">`	– train (id ICE788)
`<departure>`	∗ departure
`<station>Munich</station>`	· station Munich
`<time>11:36</time>`	· time 11:36
`</departure>`	
`<arrival>`	∗ arrival
`<station>Hamburg</station>`	· station Hamburg
`<time>17:45</time>`	· time 17:54
`</arrival>`	
`...`	...

Figure 3.7: Rendering of markup.

In Fig. 3.7 the markup of the source document on the left side (e.g. <`trains`> or <`train id="ICE788"`>) is rendered on the right side (e.g. trains or train (id ICE 788)). The specification of the rendering is given by the style sheet below (see Fig. 3.8):

```
1  trains::before    { content : "trains"; }
2  train::before     { content : "train (id " attr(id) ")"; }
3  departure::before { content : "departure"; }
4  station::before   { content : "station "; }
5  time::before      { content : "time "; }
```

Figure 3.8: CSS style sheet rendering the markup of the XML file of Fig. 3.7.

Obviously in the style sheet (see Fig. 3.8), each XML element such as `train` as well as each XML attribute such as `id` in line 2 is addressed explicitly. Due to the lack of generic constructs to access the markup of the source document in the style sheet, the rendering of each markup construct (such as XML elements) needs to be defined separately. A generic construct like in XSLT (see Fig. 3.9) to insert the name of a currently selected XML element in a CSS rule is not available.

```
<xsl:value-of select="name()" />
```

Figure 3.9: Generic Access to the name of XML elements in XSLT.

CSS functions such as `attr(X)` (see Section 2.7.1) offer a similar functionality by comparison to the XPath function `name()`. Contrary to expectations, the CSS function `attr(X)` does not offer generic rendering capabilities because the name of the parameter (selecting an attribute) must be known in advance while writing the style sheet. Note that the wildcard pattern `*` (see Section 2.3) is not applicable as parameter to the CSS function `attr(X)`.

As a consequence, CSS 2.1 and CSS 3 offer constructs to render XML elements and XML attributes. However, the rendering capabilities of CSS are restricted to XML elements and XML attributes that are known in advance. Styling of unknown XML elements and XML attributes such as covered by XSLT is not possible. An extension introduced in Section 4.3 overcomes this restriction.

3.4 Depth-Dependent Styling

According to Chapter 1 an alternating styling of highly nested trees such as the 'Tree of Life' or threads in a discussion forum is highly welcome in practice. In current CSS, an XML element on a certain depth can be addressed using CSS combinators (see Section 2.5) as follows:

```
*             { /* DEFINITION A */ }                /* level 1                */
* *           { /* DEFINITION B */ }                /* level 2                */
* * *         { /* DEFINITION A */ }                /* level 3                */
* * * *       { /* DEFINITION B */ }                /* level 4                */
.
.
.
* * * ... *   { /* DEFINITION A */ }                /* level n - 1            */
* * * * ... * { /* DEFINITION B */ }                /* level n (maximum level) */
```

Figure 3.10: Styling on a certain depth in CSS.

Each nesting level in the XML tree needs its own CSS rule. Since CSS style sheets must be finite, the styling can only be written until a certain depth. Hence, the number of CSS rules defining an alternating styling in depth must be equal to the maximum depth of a source document. XML elements exceeding the maximum level of depth are styled depending on the most specific matching rule (see Section 2.8). Therefore, the styling stays the same for all depths exceeding the maximum specified in the style sheet. Hence, an alternating styling of XML elements on unknown depth is not possible using CSS. Note that this restriction is similar to the restriction of CSS concerning visualization of markup (see Section 3.3).

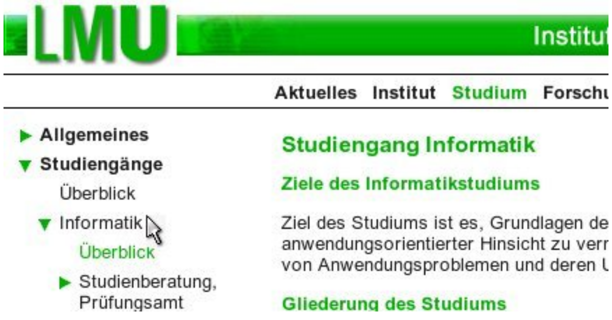
The parameterization of CSS selectors is available for direct child XML elements in CSS 3 (see Section 2.7.3). The author is not aware of any reason, why a parameterized selection depending on depth is not offered by CSS 3. Although CSS 3 offers the parameterized selection for sibling XML elements, it does not offer the symmetric case of parameterized selection of nested child elements. For instance the `:nth-child(An+B)` structural pseudo-class allows an alternating styling in breadth. A corresponding construct for alternating styling in depth is not offered in current CSS. An extension introduced in section (see Section 4.4) will overcome this restriction.

3.5 Dynamic Rendering

Obviously, input devices are essential to dynamic rendering (see Chapter 1) because such devices serve as interface between the viewer of a document and the rendering engine (e.g., of a Web browser). In CSS, the mouse (see Section 2.7.4) is the only input device that can change the rendering of a source document directly. (Indirectly, the history of visited Web pages of a Web browser is an 'input device', too, see Section 2.7.4). Input provided by other devices such as the keyboard cannot be selected.

A mouse can perform various actions like clicks or movements that can be received by the rendering engine of a Web browser as so called events. For instance, if the mouse is moved on the desk such that the mouse cursor enters the rendering of an XML element, the rendering engine receives a corresponding event message. In the case of the dynamic pseudo-class `:hover`, the rendering can be derived directly from the position of the mouse cursor because `:hover` affects only the rendering of XML elements the mouse cursor is hovering above. For instance, tree structures can be rendered dynamically using the following style sheet in Fig. 3.11:

```
1  *:root       { display:block; }
2  *:root *     { display:none;  }
3  *:hover > *  { display:block; }
4  *:active     { display:block; }
```



Ziel des Studiums ist es, Grundlagen de
anwendungsorientierter Hinsicht zu verr
von Anwendungsproblemen und deren L

Figure 3.11: Dynamic Rendering of Tree Structures.

The CSS rule in line 1 of the CSS style sheet above renders the root XML element of a source document as block. Since the descendant XML elements of the root element should not be visible by default, the rule in line two defines that every child element of the root element is not displayed. Dynamic rendering in this example is defined in the rule of line 3: Whenever the mouse cursor hovers above an XML element, its children become visible. Note that this rule works transitively because the rendering of each 'unfolded' XML element can be unfolded again, if the mouse cursor hovers the way down to it.

The CSS style sheet above can be used to define the 'navigation tree' of the Web-site on the right side in Fig. 3.11: If a hyperlink is active, it is displayed by default as defined by the rule in line 4. Otherwise an item that was dynamically unfolded by the viewer would be folded again as soon as the mouse cursor has left the rendering of an item.

However, a generic folding and unfolding like in Fig. 3.7 is not possible because the rendering state of each XML element (being folded or not) cannot be realized in CSS. Extensions of current CSS allowing extended dynamic rendering are introduced in Section 4.5.

3.6 Hyperlinks

In an HTML document a hyperlink is a reference to an Internet resource such as another HTML document or an e-mail address. Obviously, the declaration of the functionality of such hyperlinks is not subject of a style sheet language. For instance, CSS does not offer means for declaring XML elements as hyperlinks. The W3C offers XLink [DMO01] and HLink [PI02] for such issues.

However, the *styling* of hyperlinks is one of the killer applications of CSS (see Section 1.1) and, of course, hyperlinks can be styled using dynamic style sheets. Thus a hyperlink-like behavior of the visualization of hyperlinks can be simulated **within** Web pages. For instance, menus that allow to change the *visible* content of a Web page can be provided using CSS but the Web resource stays the same.

In any case, such simulations have their limits, and CSS is not a language for the declaration of hyperlinks. On this account the declaration of hyperlinks is not covered in this thesis.

CHAPTER
FOUR

How CSS^{NG} extends CSS 3

This chapter addresses the concrete extensions proposed in this thesis, which are motivated in Chapter 1. Each extension proposed in the following sections tries to overcome restrictions of current CSS (see Chapter 3) respecting principles as introduced in the Section 4.1.

The Sections 4.2 to 4.4 briefly introduce novel static CSS^{NG} rules mainly aiming at visualizing XML markup. Section 4.5 introduces the rule-based interface for dynamic document styling. Finally, Section 4.6 discusses a novel CSS^{NG} combinator, which is especially apt for the dynamic but also for static document rendering. These extensions are based on proposals on extending style sheet languages in a project thesis [Wie05].

4.1 Framework for the CSS^{NG} extension

CSS^{NG} is a rather expressive extension but actually it is a rather conservative extension, too. All new features of CSS^{NG} are designed with respect to the following guidelines.

4.1.1 Downward Compatibility

CSS^{NG} does not interfere with style sheets written in prior CSS versions CSS 2.1 and CSS 3 because the semantics of existing constructs in current CSS were not changed. Hence, every style sheet of version 2.1 or 3 can be re-used.

4.1.2 Combined Complexity

One of the main guidelines of the CSS^{NG} extension is to preserve the computational complexity of styling semi-structured data using CSS 3. Hence, the combined complexity of the CSS^{NG} styling process is linear in the size of the style sheet and the input document.

4.2 Markup Insertion

Markup especially in XML documents often conveys application relevant information. Therefore, it might be useful to visualize it. However, CSS 2.1 and CSS 3 offer quite limited means for markup visualization. CSS 3 allows the insertion of plain text specified in a CSS style sheet. The *pseudo-elements* `::before` and `::after` cause insertion of text before and after a selected XML or HTML element (see Section 2.7.1) as demonstrated below:

```
price::before { content : "EUR "; }
```

Figure 4.1: Insertion of XML text nodes.

CSS^{NG} extends the value set of the CSS property `content`, which is associated with the CSS pseudo-elements of CSS 3 `::before` and `::after`. CSS 3 can only insert plain text before and after XML elements. CSS^{NG} can also insert elements using the CSS^{NG} function `element(NAME, VALUE)`. In order to insert elements that have attributes, the function can take additional arguments between `NAME` and `VALUE`. For these additional arguments CSS^{NG} offers another function `attribute(NAME, VALUE)`. The following sections describe these new features for markup insertion in detail.

4.2.1 CSS^{NG} Function `element`

The CSS^{NG} function `element(NAME, ATTRIBUTES, VALUE)` inserts XML elements to the 'intermediary step' as demonstrated in Fig. 4.2.

```
<span>elem</span>
                                           is inserted before each XML element by the rule
*::before { content: element("span", "elem"); }
```

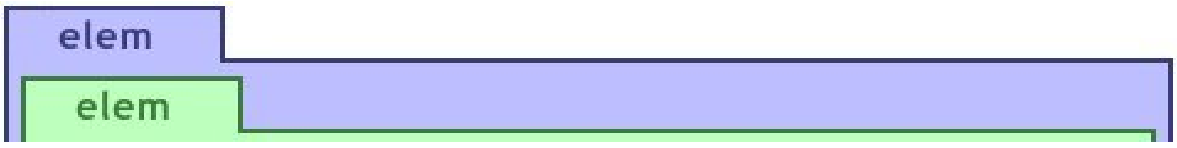


Figure 4.2: Insertion of a `span` element representing a tab.

The parameter `NAME` specifies the name of the XML element (such as `span`). The type of the parameter `NAME` is String.

The parameter `ATTRIBUTES` denotes the possibility to add XML attributes that belong to the current XML element to be inserted. Since an XML element can have zero or more XML attributes [BPSMM00], we substitute `ATTRIBUTES` by a variable number of arguments (varargs) making `element` a variadic function[1]. The only allowed type of input parameters is an XML attribute. XML attributes can be constructed by the CSS^{NG} function `attribute` that is addressed in Section 4.2.2.

The last parameter `VALUE` of the CSS^{NG} function `element(NAME, ATTRIBUTES, VALUE)` has the type string expression.

Like in CSS 3, a string expression can be a string literal such as `"elem"` or a concatenation such as `"e" "le" "m"`. The fact that CSS 3 concatenates strings by simple juxtaposition without explicit operator can be confusing in large string expressions (especially if the concatenated string literals start or end with whitespace). Nevertheless, this is a CSS convention, so CSS^{NG} adopts it for the sake of downward compatibility.

The extension now is that the `VALUE` parameter may be a concatenation not only of string literals, but also of string expressions according to the table in Fig. 4.3.

[1]A variadic function is a function of variable arity, see ISO/IEC 9899:1999

VALUE	**expression in** `content` **declaration**	**description**	**element inscribed in intermediary step**
no value	`element("em")`	empty element	`<em />`
empty string	`element("em", "")`	empty element	`<em />`
string	`element("em","Hello")`	element with content	`<em>Hello</em>`
`element(...)`	`element("em", element("br"))`	nested element	`<em><br /></em>`
String `element(...)`	`element("em", "Hello" element("br"))`	mixed content	`<em> Hello<br /> </em>`

Figure 4.3: Alternatives of the parameter `VALUE` of the CSS^{NG} function `element`.

4.2.2 CSS^{NG} Function `attribute`

The CSS^{NG} function `attribute(NAME, VALUE)` offers to construct XML attributes that can be used in the CSS^{NG} function `element` (see Section 4.2.1). The first parameter `NAME` specifies the name of an XML attribute such as `class`. The second parameter `VALUE` specifies the value of the XML attribute such as `tab` (see Fig. 4.4).

```
*::before { content: element("span", attribute("class", "tab"), "elem"); }

                                   This rule inserts the following markup:

<span class="tab">elem</span>
```

Figure 4.4: Demonstration of the CSS^{NG} `attribute` function.

Note that in both CSS^{NG} functions, `element` and `attribute`, double quotes within String parameters must be paraphrased by XML entities such as `&quot;` or `&#34;` to avoid ambiguous expressions such as:

```
element("em", "The function "element("br")" inserts the element "br"".)

                                   This expression must be paraphrased by

element("em", "The function &quot;element(&quot;br&quot;)&quot;
              inserts the element &quot;br&quot;".)
```

Figure 4.5: Escaping of nested command chars.

4.2.3 Prevention of Iterative Insertion of Markup

As discussed in Section 3.1.4 iterative adding must be avoided. Otherwise recursive non-terminating CSS rules could be defined (see Fig. 3.4).

To avoid recursive insertion of XML elements, the pseudo-elements `::before` and `::after` can be applied only to XML elements of the source document and not to XML elements of the ‘intermediary step’. As a consequence, inserted XML elements are not styled by CSS^{NG} rules that use one of the pseudo-elements `::before` or `::after` as selector, as demonstrated in the following example:

Source (extract):

```
  </publisher>
            <!-- virtual XML text node -->
  <price>65.95</price>
            <!-- virtual XML text node -->
</book>
```

CSS Style Sheet (extract):

```
price::before {content:
              element("price",
              "&euro; ");}
```

Intermediary Step (extract):

```
  </publisher>
  <price>&euro;</price><price>65.95</price>
</book>
```

Rendering:

```
 € 65.95
```

Figure 4.6: Restricted Insertion of XML elements.

4.2.4 Well-Formedness

The CSS^{NG} functions `element` and `attribute` ensure well-formedness of inserted markup syntactically. If markup could be written directly as strings, insertions like the following would become possible. Although constructing XML elements via the new functions means more code, only well-formed markup can be inserted.

```
p::before { content:  "<samp><em></samp></em>"; }
```

Figure 4.7: Counter-example: Insertion of not well-formed markup.

Since XML elements can be inserted before or after each XML element in an XML document, XML elements can be inserted before or after the document root, too. Applied to the source document such a transformation would not be well-formed, because the source document would become disrooted, which conflicts with the specification of XML [BPSMM00].

In current CSS, the insertion of XML text nodes can be applied to root elements (and in HTML to the `body` element). Such an insertion does not affect the well-formedness of the source document because the insertion affects only the ‘intermediary step’ of the rendering process for rendering issues, while the source document is not changed.

To handle this phenomenon CSS^{NG} introduces a virtual root element that encapsulates the data of the intermediary step. This virtual root cannot be selected by CSS or CSS^{NG} selectors. The virtual root serves for conceptual well-formedness of the XML document in the intermediary step only. Consequently, the `:root` pseudo-class of CSS 3 does not match the virtual root but the root XML element of the source document. Due to the definition of the ‘intermediary step’ (see Section 3.1) all other CSS selectors such as `nth-child()` are applied on the intermediary step after adding and/or deleting XML elements.

4.3 Markup Visualization

Current CSS allows to visualize *known* markup structures (such as XML elements and XML attributes) but a generic visualization of markup is not offered (see Section 3.3). The extensions by which CSS^{NG} overcomes these restrictions is *markup querying*, which is explained in detail in the following subsections using the example in Figures 4.8 and 4.9.

Even before having read the detailed explanations, the reader may get a better understanding by comparing the example with Fig. 3.8. In fact the style sheet of Fig. 4.9 is a re-implementation of the style sheet of Fig. 3.8 at a generic level. It reduces the programming effort from one rule per XML element name to one single rule for an arbitrary XML document.

Another informative comparison is also possible before studying all the details: the effect of the style sheet of Fig. 4.9 can be achieved by an XSLT transformation as given in Fig. 4.10. The striking difference in conciseness is due to the declarativity of CSS^{NG}, which reduces the amount and the error-proneness of code considerably.

Source:	**Rendering:**
`<trains>`	• trains
`<train id="ICE788">`	– train (id ICE788)
`<departure>`	∗ departure
`<station>Munich</station>`	· station Munich
`<time>11:36</time>`	· time 11:36
`</departure>`	
`<arrival>`	∗ arrival
`<station>Hamburg</station>`	· station Hamburg
`<time>17:45</time>`	· time 17:54
`</arrival>`	
...	...

Figure 4.8: Rendering of markup.

```
1  *::before    { content : element-name()
2                           id { content: " (" attribute-name() " " attribute-value() ")" }
3               }
```

Figure 4.9: CSS^{NG} style sheet rendering the markup of the XML file of Fig. 4.8.

The rule in Fig. 4.9 defines the rendering of the XML document on the left side of Fig. 4.8. The result of the rendering process is shown on the right side of Fig. 4.8. Each XML element is tagged by its name caused by the CSS^{NG} function `element-name()` (see Section 4.3.1). In addition, the XML attribute `id` is visualized by the *attribute rule* in line 2. The attribute rule consists of an *attribute selector* to select the attribute `id` and a rule declaration to specify the rendering of selected attributes (see Section 4.3.2) . If no `id` attribute is attached to the current XML element, an empty string is returned. Otherwise, if the current XML element has an `id` attribute, the body of the attribute rule is evaluated to generate a rendering of the `id` attribute as follows: the name and the value of the `id` attribute are queried using the CSS^{NG} functions `attribute-name()` and `attribute-value()` . Furthermore, the results of these two CSS^{NG} functions are implicitly concatenated with strings containing parentheses and space yielding a human-readable visualization.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- generate HTML document -->
  <xsl:template match="/">
    <html>
      <body>
        <!-- render all XML elements -->
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <xsl:template match="*">
    <ul>
      <!-- render XML elements in breadth -->
      <xsl:for-each select=".">
        <li>
          <xsl:value-of select="name()" />
          <xsl:text> </xsl:text>

          <!-- render the id attribute of current XML element, if existing -->
          <xsl:if test="@id">
            <xsl:text> (id </xsl:text>
            <xsl:value-of select="@id" />
            <xsl:text>) </xsl:text>
          </xsl:if>

          <!-- render XML elements in depth -->
          <xsl:apply-templates />
        </li>
      </xsl:for-each>
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

Figure 4.10: XSLT program providing the same rendering as the style sheet in Fig. 4.9.

4.3.1 Rendering of XML elements

XML elements can be visualized by showing their names (see Fig. 4.8). However, current CSS does not allow to access the name of any XML tag for rendering issues directly. The only solution to visualize XML elements by their names is writing rules for every XML element with different name (see Fig. 3.7) indirectly. To allow access to the names of XML elements, we introduce new CSS^{NG} functions. Since the CSS function `attr` (see Section 2.7.1) already allows to access the values of XML attributes, extending CSS functions toward visualizing XML elements seems to be natural.

We extend CSS functions by a new function `element-name()` for getting the name of the currently selected XML element in a CSS rule as a String. Like all other CSS functions [BLLJ98] `element-name()` can only appear in the context of a CSS `content` declaration (see Fig. 4.9). The name of the current XML element can be derived from the selection of the corresponding CSS selector because every CSS rule is applied individually to one XML element (see Section 2.4).

4.3.2 Rendering of XML attributes using CSS^{NG} Attribute Rules

In contrast to XML elements, XML attributes cannot be selected by CSS selectors. Selections can only be constrained to XML elements having a special XML attribute configuration using square brackets (see Section 2.4). Consequently, extending CSS functions to a function `attribute-name()` (see `element-name()`, Section 4.3.1) is not sufficient because an XML element can have more than one XML attribute, and, therefore, no ‘current XML attribute’ can be selected by default like a ‘current XML element’. Therefore, we extend CSS toward an extended *attribute rule* concept capable of selecting and rendering XML attributes.

The idea of CSS^{NG} attribute rules is a transfer of the concept of ordinary CSS 3 rules selecting XML elements to XML attributes. The anatomy of attribute rules stays the same as the anatomy of ordinary CSS rules (see Fig. 1.1). The following sections 4.3.2.1 - 4.3.2.3 introduce to the head of CSS^{NG} attribute rules. Finally, Section 4.3.2.4 covers the declaration part of such rules.

4.3.2.1 CSS^{NG} Attribute Selectors

Since each XML attribute belongs to exactly one XML element, the ‘XML element context’ of an attribute can be selected by CSS simple selectors selecting XML attribute names instead of XML element names. Of course, attribute constraints defined with square brackets of CSS simple selectors are not applicable in this context. XML attributes are flatly structured and can be seen as a set of keys and values pairs [BPSMM00].

The position of the attribute selector is located in the `declaration` part of a CSS rule. Obviously, the only reasonable place in the `declaration` part is as value of a CSS `content` declaration because the extended selector concept is meant to realize insertions to the rendering of a source document (see Fig. 4.9) using the pseudo-elements `::before` and `::after`.

Since there is no order defined on XML attributes, visualizing XML attributes cannot be deterministic. Therefore, if using the wildcard pattern `*` for selecting all XML attributes, the XML attributes are rendered as given by the serialization of the source document.

4.3.2.2 CSS^{NG} Adopting CSS Grouping

Other selector concepts of CSS (see Section 2) cannot be adopted reasonably except for grouping (see Section 2.6) and except for parts of the CSS structural pseudo-classes (see Section 2.7.3).

In current CSS, grouping is syntactic sugar to integrate CSS rules having the same CSS `declaration`. This concept can be adopted to attribute selectors. In this case the order of the rendering of XML attributes can be interpreted, like in the following example (Figures 4.11 and 4.12):

```
a::before { content: element-name()
                     href, title { content: "(" attribute-name() " "
                                             attribute-value() ")"
                                 }
          }
```

Figure 4.11: Attribute Selector using Grouping.

```
a::before { content: element-name()
                     href  { content: "(" attribute-name() " " attribute-value() ")" }
                     title { content: "(" attribute-name() " " attribute-value() ")" }
          }
```

Figure 4.12: Rule of Fig. 4.11 without Grouping.

The attribute selectors in the example of the Figures 4.11 and 4.12 use grouping indicated by the "`,`". Since `href` appears before `title` in the attribute selector, `href` is rendered before `title`.

4.3.2.3 Adopting CSS 3 Structural Pseudo-classes

XML attributes have no order but adopting CSS structural pseudo-classes is reasonable for rendering issues. Structural pseudo-classes taking into account selection in breadth can be transferred to the serialized structure of XML attributes. The following example illustrates how to avoid the insertion of the last separator sign such as a comma, if listing XML attributes:

```
1  *::before { content : element-name()
2                *:not(:last) { content: "(" attribute-name() " " attribute-value() "), " }
3                *:last       { content: "(" attribute-name() " " attribute-value() ")" }
4            }
```

Figure 4.13: Attribute Selectors with Structural Pseudo-classes.

In Fig. 4.13 a comma is inserted after each XML attribute according to line 2 of the style sheet. The rendering of the last XML attribute, however, is defined by the rule in line 3, where no comma is inserted after the rendering of the XML attribute.

The following list itemizes all structural pseudo-classes of CSS 2.1 and CSS 3 that can be mapped to the structure of XML attributes:

CSS (2.1 and) 3.0:	Transfer to Attributes:	New Semantics:
`:nth-child()`	`:nth-attribute()`	the n-th attribute in document order
`:nth-last-child()`	`:nth-last-attribute()`	the n-th last attribute in document order
`:first-child`	`:first-attribute`	the first attribute in document order
`:last-child`	`:last-attribute`	the last attribute in document order
`:empty`	`:empty`	matches, if no attribute is defined
`:not()`	`:not()`	selects every attribute of the current element except of the attributes specified as parameter (see Fig. 4.13)

Figure 4.14: Transfer of Existing Pseudo-classes to Attribute Selectors.

4.3.2.4 Declaration of CSS^{NG} Attribute Rules

In addition to the constructs that are allowed in the `VALUE` of a `content` declaration , new CSS functions for accessing the name and the value of XML attributes are needed. In analogy to the function `element-name()` (see Section 4.3.1) CSS^{NG} introduces two new functions, see Fig. 4.15:

CSS^{NG} Function:	Semantics:
`attribute-name()`	Yields the name of the currently selected XML attribute.
`attribute-value()`	Yields the value of the currently selected XML attribute.

Figure 4.15: Transfer of Existing Pseudo-classes to Attribute Rules.

Note that CSS 2.1 already offers the CSS function `attr(X)` to access the value of a certain XML attribute (see Section 2.7.1). Nevertheless, this thesis decides in favor of proposing a new function instead of extending the existing one because of symmetry reasons to the function name `element-name()`.

4.3.3 Well-Formedness of Insertions using Attribute Rules

Introducing attribute rules reminds of extending CSS with the insertion of markup (see Section 4.2): Extending CSS with insertion of markup was not possible without restrictions to avoid, infinite and, hence, ill-formed insertion. Extending CSS with XML attribute selectors, however, does not cause recursive insertion.

Recursive insertion (like in Fig. 3.4) is not possible using attribute rules. Indeed XML attributes and XML elements can be inserted using attribute rules. The inserted attributes, however, cannot be selected using the same attribute selector because the space for insertion is either before or after the corresponding XML element, and is, hence, not in the scope of the same attribute selector. The following example (see Fig. 4.16) illustrates why recursive insertion is not possible using attribute selectors.

Source (extract):

```
  </publisher>
            <!-- virtual XML text node -->
  <price newElement="euro">65.95</price>
            <!-- virtual XML text node -->
</book>
```

CSS Style Sheet (extract):

```
price::before { content:
 newElement {
  content:element(attribute-value(),
                 "&euro;");
          }
}
```

Intermediary Step (extract):

```
  </publisher>
  <euro>&euro;</euro>
  <price newElement="euro">65.95</price>
</book>
```

Rendering:

```
€ 65.95
```

Figure 4.16: Insertion of XML Elements via Attribute Rule.

The style sheet in Fig. 4.16 applied to the source document on the left side causes the intermediary step on the lower left side of the figure as follows: The XML element `price` is matched by the CSS simple selector `price`. Afterwards, the attribute selector (`newElement`) matches the only attribute `newElement` of the XML element `price`. According to the body of the attribute rule a new element is inserted. The name of the new element is specified by the value of the currently selected XML attribute `newElement`. Hence, the XML element `euro` is inserted as can be seen in the intermediary step. Since no new attribute can be inserted to the currently selected XML element `price`, no recursive attribute can be specified.

CSS^{NG} attribute insertion cannot violate the well-formedness of XML documents but CSS^{NG} attribute insertion can violate XML validity constraints [BPSMM00] because an element type must not have more than one `id` attribute specified and values of type `idref` must match the Name production, and values of type `idrefs` must match Names; each Name must match the value of an `id` attribute on some element in the XML document; i.e. `idref` values must match the value of some ID attribute. However, this is not an issue for the intermediary step of the CSS^{NG} rendering process because only well-formedness is needed for the rendering of the intermediary step.

4.3.4 Open Issues

The specification of XML [BPSMM00] offers more constructs than XML attributes and XML elements. Comments, Processing Instructions, CDATA Sections, the prolog and the document type declaration are not covered in this thesis. Extensions towards visualizing the remaining constructs that are possible in XML documents can be done analogously introducing new functions to CSS^{NG}.

4.4 Depth-dependant Styling

Styling depending on breadth is planned in CSS 3 [Bos05]. Tables, for instance, can be styled using alternating background colors for each line. CSS^{NG} offers in addition styling depending on the depth of an element in an XML document : The pseudo-class `:nth-descendant(An+B)` restricts selections to XML elements having $An + B$ ancestors before them.

Fig. 4.17 demonstrates the visualization of a highly nested XML document with colors repeating on every sixth level. On the left side this rendering is realized using CSS^{NG} and alternatively using CSS 3. Due to its depth-dependant styling features, the upper CSS^{NG} style sheet needs only six rules. The CSS 3 style sheet below needs one rule for every level. Hence, styling in CSS 3 is possible up to a certain depth only as shown on the right side of Fig. 4.17 using the CSS 3 style sheet on the lower left side of Fig. 4.17.

CSS^{NG} style sheet:

```
*:nth-descendant(6n+1) { background-color: A; }
*:nth-descendant(6n+2) { background-color: B; }
*:nth-descendant(6n+3) { background-color: C; }
*:nth-descendant(6n+4) { background-color: D; }
*:nth-descendant(6n+5) { background-color: E; }
*:nth-descendant(6n+6) { background-color: F; }
```

CSS 3 style sheet:

```
*                       { background-color: A; }
* *                     { background-color: B; }
* * *                   { background-color: C; }
* * * *                 { background-color: D; }
* * * * *               { background-color: E; }
* * * * * *             { background-color: F; }

* * * * * * *           { background-color: A; }
* * * * * * * *         { background-color: B; }
* * * * * * * * *       { background-color: C; }
* * * * * * * * * *     { background-color: D; }
* * * * * * * * * * *   { background-color: E; }
* * * * * * * * * * * * { background-color: F; }
```

Presentation using CSS 3:

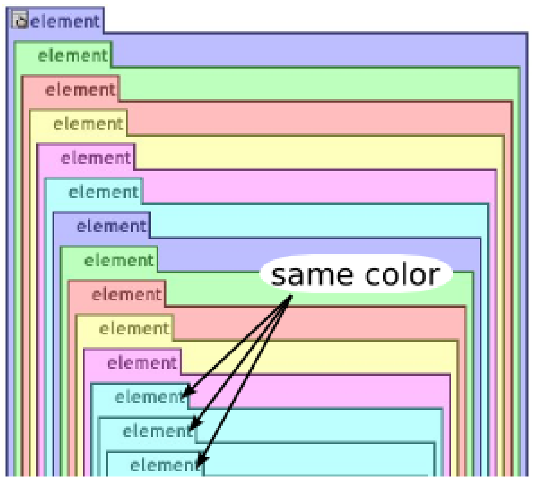


Figure 4.17: Depth-dependent Styling.

4.5 Dynamic Styling Generalized

Dynamic styling in CSS 3 is limited to the dynamic pseudo-class `:hover`. This construct allows dynamic styling in the local context of the mouse cursor only as demonstrated in Section 2. This is not sufficient to implement a behavior like folding a tab as demonstrated in Figure 4.22: when the mouse cursor moves away, the cursor does no longer hover over the selected XML element, and its tab would be automatically unfolded again.

CSS^{NG} introduces dynamic pseudo-classes for *all* HTML intrinsic events [ABC^{+}99] such as `onclick`. These intrinsic events allow a better differentiation of events, for instance `:hover` in CSS 3 can be expressed using `:onmouseover` and `:onmouseout` in CSS^{NG}.

Instead of using HTML intrinsic event attributes like for scripting languages, CSS^{NG} allows a standalone specification of dynamic styling in separate text files that can be applied to multiple documents. The following example shows a rather simple dynamic CSS^{NG} rule (Fig. 4.18):

```
a:onclick(10) { background-color: green; }
```

Figure 4.18: Dynamic CSS^{NG} rule changing the color after 10 clicks on an `a` element.

The rule above implements a 'private' adaptive hyperlink. After 10 clicks on the hyperlink the background color changes to green indicating that the link is popular depending on the so-called *history*[2] of the used Web browser.

This extension makes it possible to apply dynamic styling on different sections of an XML document at the same time. For instance if two hyperlinks were clicked ten times in a Web page, both will be presented with the different background color.

Similar extensions using HTML intrinsic events have already been proposed by the W3C (see Chapter 5). The following paragraphs introduce to novel capabilities of CSS^{NG}.

Dynamic pseudo-classes like `:hover` allow dynamic rendering in current CSS (see Section 2.7.4). For offering a more flexible rendering, CSS^{NG} extends CSS by the ability to handle additional input devices and not only the mouse, and CSS^{NG} extends CSS by supporting the selection of more actions of the input devices and not only hovering of the mouse cursor over an XML element.

4.5.1 Recurrence of Events

In the following, event selectors the parameter `RECURRENCE` specifies, how often an event has happened before. All CSS^{NG} dynamic pseudo classes support *recurrence patterns* as parameters. In analogy to the structural pseudo-class `:nth-child()` of CSS 3 (see Section 2.7.3), the parameter `RECURRENCE` has the same type of arguments of the form `(An+B)`. The semantics of that argument is as follows: The event selector matches, if there is a natural number n such that the event has occurred exactly $A \cdot n + B$ times in the past. For instance the CSS^{NG} selector `*:onclick(3n+1)` detects the first, the fourth, the seventh, etc. click on an arbitrary XML element.

On one hand such recurrence patterns allow to reuse CSS^{NG} rules for folding and unfolding as demonstrated below. On the other hand recurrence patterns allow to "delay" the application of rules up until a number of events, for instance clicks, as demonstrated in the previous section (see adaptive hyperlink above).

4.5.1.1 Permanent Changes on Rendering

In contrast to the dynamic pseudo-class `:hover`, the current state of the rendering cannot be derived from the state of the input device. For instance, if a mouse cursor was hovering over the rendering of an XML element, it can be derived that the mouse cursor has hovered at least once over the rendering of the current XML element. However, it cannot be derived from the state of the mouse, how often this event occurred exactly. Therefore an accumulator for the rendering of each XML element is needed in the 'intermediary state' of the rendering process (see Section 3.1). The accumulator counts how often a rendering event already occurred. Since various events can be specified. An accumulator for each event such as `:onkeydown(KEY,RECURRENCE)` or `:onmouseover(RECURRENCE)` is needed, as demonstrated exemplarily in the Fig. 4.19.

The XML elements in the lines 1, 2, and 7 in the example contain XML attributes saving, how often events occurred during a rendering session. The XML attributes consist of a namespace `event` to avoid conflicts with other XML attributes. The name of each XML attribute of the namespace event corresponds to the exact name of the event such as `event:onclick` for the event `onclick`. Obviously, the attributes stating the number of occurred events are

[2]Normally, Web browsers save the addresses of visited Web pages in a list called history.

```
<trains event:onkeyup="2" event:onclick="5"> <!-- event accumulators -->
   <train id="ICE788" event:onclick="4">     <!-- event accumulators -->
      <departure>
        <station>Munich</station>
        <time>11:36</time>
      </departure>
      <arrival event:onclick="3">            <!-- event accumulators -->
        <station>Hamburg</station>
        <time>17:45</time>
      </arrival>
...
```

Figure 4.19: Event Accumulators realized using XML attributes having various initial values.

positioned in the 'intermediary step' of the XML document wherever an event was performed on the rendering of the corresponding XML element. Consequently, the initial value of each such XML attribute (if not stated) is `0`. Note that the attribute `event:onkeyup` is located in the root XML element because keyboard events cannot be assigned to a specific XML element in contrast to a mouse click being assigned to the XML element where the mouse cursor is hovering on.

4.5.1.2 Acyclic Events

Omitting the first part of the argument (`An+`) allows to specify *acyclic events*. For instance, an adaptive link changing its color depending on its usage can be rendered as follows:

```
hyperlink            { color : black; }
hyperlink:onclick(1) { color : green; }
hyperlink:onclick(2) { color : red;   }
```

Figure 4.20: Acyclic Rendering: Adaptive Rendering of a Hyperlink.

The rules in the style sheet above define the rendering of the XML element `hyperlink` depending on the number of mouse clicks on the XML element. According to line 1, the standard rendering displays the hyperlink in black letters. After one click, the font color changes to green. Finally, after the second click the color changes to red indicating a frequented hyperlink. More than two clicks on the rendering of an XML element lets change the appearance of its rendering according to the static rules or the standard rendering as given by the current Web browser. (This behavior of CSS^{NG} is discussed in Section 4.5.1.4.) If the rendering should stay the same after more than 2 clicks, a cyclic selector as `hyperlink:onclick(1n+3)`[3].

4.5.1.3 Cyclic Events

Applications like folding and unfolding of XML elements require a cyclic evaluation of events . For instance, it should be possible to fold and unfold a data item several times (see Chapter 1). A cyclic matching of events can be defined as follows:

[3]Note that `1n+3` matches all natural numbers ≥ 2.

```
*:onclick(2n+1) > * { display : block; }
*:onclick(2n+2) > * { display : none;  }
```

Figure 4.21: Cyclic Rendering: Folding and Unfolding of XML elements.

According to the rule in line 1 of the style sheet above, an XML element is unfolded (displayed) if an odd number of clicks have been performed on the rendering of the parent XML element. Analogously, the rule in line 2 folds (displays not) an XML element, if an even number of mouse clicks have been performed on the parent XML element.

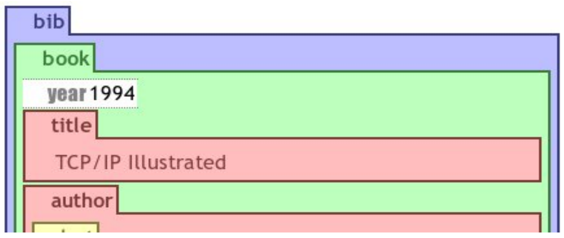
4.5.1.4 Event Interpretation

The proposed extensions on the dynamic pseudo-classes of CSS require a definition of the evaluation of CSS for events because in current CSS events are not permanent (see Section 2.7.4). In current CSS, XML elements are rendered taking account of the last and, hence, most specific CSS rule (see Section 2.8). The same principle can be applied to the interpretation of extended dynamic pseudo-classes as follows: The most specific rule is applied to render an XML element.

4.5.2 Dynamic Styling Combined

A noticeable feature of the (novel) dynamic pseudo-classes of CSS^{NG} is their compatibility with CSS 3 *combinators*, which allow to specify tree patterns.

A CSS 3 selector is an alternating sequence of so-called *simple selectors* (already informally introduced in Section 2) and combinators. For instance, the combinator + means that the simple selector on its left side must be a preceding sibling of the simple selector on the righthand side. The CSS declaration (in curly braces) is only applied to the XML element matched by the right most simple selector.



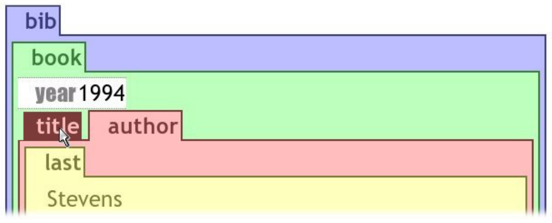


Figure 4.22: Unfolded and folded visualization of the XML element `title`.

The following example implements *alternating folding and unfolding* for the visualization of arbitrary (simple selector *) XML elements (see Fig. 4.22). A click on a tab of a visualized XML element like `title` folds its visualization. Another click on a tab unfolds it (see `title` in Fig. 4.23):

```
tab:onclick(2n+1) + * {display:none}          Fold on odd number of clicks.
tab:onclick(2n+2) + * {display:block}       Unfold on even number of clicks.
```

Figure 4.23: CSS^{NG} style sheet implementing folding and unfolding in Fig. 4.22.

In the CSS^{NG} style sheet in Fig. 4.23 it is assumed, that an XML element `tab` is inserted before each XML element containing the name of the corresponding XML element (for details see Sections 4.2 and 4.3). The lefthand *selector* of the first CSS^{NG} rule above is composed of the two *simple selectors* `tab:onclick(2n+1)` and `*` combined with the CSS 3 combinator, `+`. The visualization of an XML element matched by the simple selector `*` disappears, if a mouse click was performed on its preceding sibling XML element, which represents its tab and stays visible.

4.5.3 Extended Input Devices

4.5.3.1 Mouse

Beside the pseudo-class `:hover` CSS^{NG} introduces new dynamic pseudo-classes taking into account when a viewer clicks on the rendering. For clicking with a mouse button on the rendering of an XML element, the dynamic pseudo-class `:onclick(RECURRENCE)` is introduced. The parameter `RECURRENCE` is introduced in Section 4.5.1 and can be omitted.

The proposed pseudo-class `:onclick(RECURRENCE)` admits new rendering capabilities. For instance, the menu items in a navigation tree (such as in Fig. 3.11) can be opened by a mouse click instead of being opened, if the mouse cursor hovers over it.

Obviously, the pseudo-class `:onclick` can be applied to hyperlinks. Hence, clicking with the mouse on a hyperlink that is styled using `:onclick` triggers a styling process as well as loading the linked Internet resource. If the current Web page is substituted by a new one, the dynamic rendering is still accessible via the history of the Web browser. Basically, the two actions do not conflict because CSS does not allow to render XML elements as hyperlinks.

4.5.3.2 Keyboard

Since in Web browsers such as Mozilla Firefox[4] or Microsoft Internet Explorer[5] browsing can be controlled by keys of the keyboard, too, this thesis proposes to admit changes of the rendering caused by keys. The new dynamic pseudo-class `:onkeydown(KEY,RECURRENCE)` detects if a key is pressed, and the new dynamic pseudo-class `:onkeyup(KEY,RECURRENCE)` detects if a key is released.

The parameter `KEY` defines a single key that activates the selection of the corresponding dynamic pseudo-class (such as `:onkeydown(KEY,RECURRENCE)`. The type of the parameter `KEY` is character. Hence a single letter such as ‘a’ can be the argument. Combinations of keys could be proposed, too, but are not addressed here. The parameter `RECURRENCE` is introduced in Section 4.5.1. It can be omitted.

Unlike with the mouse device, the keyboard can cause conflicts because two buttons can be pressed at the same time. As discussed above combinations of keys are not possible for selection in one rule but two different rules can select the state of two different keys (being pressed or not). Depending on the interpretation of CSS style sheets (see Section 2.8) conflict resolution depends on the order of rule activation. In this case, the rendering of the rule of the second key shades the rendering of the rule of the first key.

These two pseudo-classes allow useful specifications of rendering. A key action such as hitting a key can show all renderings of XML elements on one level of an XML document.

[4]http://www.mozilla.org/products/firefox/
[5]http://www.microsoft.com/windows/ie/

```
1  *:root                              { display : block; }
2  *:root *                            { display : none;  }
3  *:root:nth-descendant(1):onkeydown('1') { display : block; }
4  *:root:nth-descendant(2):onkeydown('2') { display : block; }
5  *:root:nth-descendant(3):onkeydown('3') { display : block; }
6  /* ... */
```

Figure 4.24: Dynamic Rendering using the Keyboard as Input Device.

The rules in line 1 and 2 of the example in Fig. 4.24 define the static rendering (without influence of the keyboard) as follows: The root XML element is rendered but all descendant elements are deleted. According to the rules in lines 3 to 5, only the XML elements on the level one, two or three are rendered depending on the currently pressed key '1', '2', or '3'.

The three dots in line 6 of the style sheet remind of 'Styling depending on depth' (see Fig. 3.10) and 'Visualization of Markup' (see Fig. 3.8) because the depth of the source document can be higher by comparison to the rules defined in the style sheet. However, the number of keys on the keyboard is limited, too. Therefore this thesis decides in favor of not proposing a generic function to bind a key to a corresponding level in the document tree.

Note that the scope of keyboard events is the whole rendering of the source document. Hence, if a key event happens, the corresponding rules apply to all matching XML elements of the source document. In contrast, the mouse events introduced in the latter section refer to the XML elements in the scope of the mouse cursor.

The proposed pseudo-class `:onkeyup(KEY,RECURRENCE)` allows to define a rendering similar to the CSS pseudo-element `::visited` (see Section 2.7.1). Data items that have been folded and unfolded can be marked as visited, e.g., using another color.

4.6 Structure-independent Selection

CSS 3 allows to constrain selections of XML elements using combinators as introduced in Section 2.5. The following combinators or in other words structural relations between XML elements are offered: descendant, child, and following sibling.

However, the support of combinators in CSS 3 is not sufficient for use cases, e.g., given in the introductory chapter. Fig. 1.8 demonstrates how side-notes can be used to superimpose information related to the current text. Obviously, the combinators of CSS 3 can specify the rendering of side-notes being in a descendent, in a child or in a following sibling relation. However, side-notes that are referred from several text portions cannot be implemented.

CSS^{NG} introduces the new combinator `?` , pronounced `then`. This novel combinator allows selections independent of structural relations. Like the selectors of CSS 3 it is an infix selector of the form `a ? b`. If the lefthand part (`a`) was matched *then* the righthand part will be evaluated. The following example demonstrates this new combinator:

The example on the lefthand side of Fig. 4.25 demonstrates static selection using the `?` selector. The CSS^{NG} rule implements support for developing XHTML documents. During the development process text portions can be marked using a CSS class called `todo`. For instance, if a paragraph is not finished yet, the opening tag could be `<p class="todo">`. If such a class is matched by the left part of the CSS^{NG} rule, all indicator XHTML elements

Static rendering example:

```
.todo ? .ready {
   background-color: red; }
```

Dynamic rendering example:

```
.sidenote-ref:hover ? .sidenote {
   display: block; }
```

Figure 4.25: The CSS^{NG} combinator `?`.

like `<span class="ready">ready</span>` are marked red. This indicates that an unfinished part is still existing in the document.

The example on the righthand side of Fig. 4.25 demonstrates dynamic selection using the `?` selector. If the mouse-cursor hovers over a reference to a side-note (marked by the class `sidenote-ref`), its side-note (marked by the class `sidenote`) is superimposed immediately.

A strength of the `?` combinator is that the selection is directed. An approach considered in [Wie05] called `panorama` and `monorama` allowed only bidirectional selections as in the following example: `.side-note(panorama):hover { ... }`. This rule selected all XML elements belonging to the class `sidenote`. This turned out to be a problem. While superimposing a side-note by hovering over an XML element of the class `sidenote` could be easily specified, hiding side-notes affected also the references to side-notes. As consequence, side-notes could be superimposed only once because the reference was not visible anymore. However, the `?` combinator allows also such selections by repeating the simple-selector as follows: `sidenote:hover ? sidenote { ... }`. This technique works because of the cascading feature of CSS. The righthand simple selector of the example matches a `sidenote`, even if the mouse-cursor is hovering over it.

A drawback of the `?` combinator is that the lefthand side and the righthand side cannot be connected, e.g. using variable bindings, for offering more generic selections. This is a deliberate restriction with a view to keep CSS^{NG} simple. In our side-note example this means that a CSS^{NG} rule for each side-note is needed.

CHAPTER

FIVE

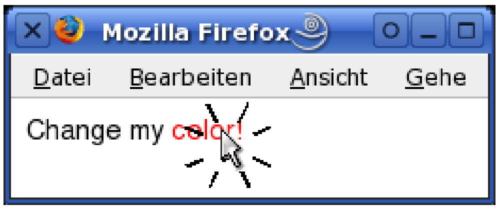
Related Extensions to CSS

This chapter introduces approaches related to CSS^{NG}, which achieve dynamic rendering features in Web-browsers. The described extensions related to CSS^{NG} afford dynamic styling for XML data. Plug-in technologies like Macromedia Flash[1], Shockwave[2], or Apple Quicktime[3], which are widespread used for providing dynamic Web sites, lie beyond the scope of this thesis. These technologies provide own approaches with their own rendering engines different from a Web browser.

5.1 Dynamic HTML (DHTML)

Dynamic HTML is a buzzword of the industry subsuming the combination of HTML data, scripting languages, and DOM [HHW+00]. In practice, scripting languages supporting the DOM interface to XML documents like ECMA Script [ECM99] are used to obtain dynamic rendering features as demonstrated in the following example:

```
<html><body>
   Change my
   <span onclick="this.style.color='red'">
      color!</span>
</body></html>
```



Change my color!

Figure 5.1: Changing the text color in HTML on mouse click to red using ECMA-script.

In the example in Fig. 5.1 the HTML so-called intrinsic event attribute [ABC+99] `onclick` is used to install an event listener to the XML element `span`. The value of the attribute `onclick` is ECMA Script code using the DOM interface. In this case the text color is set to red on mouse click.

Obviously, this technique is very powerful because

[1]http://macromedia.com/software/flash/flashpro/?promoid=BINT
[2]http://www.shockwave.com/
[3]http://www.apple.com/quicktime/

- HTML intrinsic event elements (`onmouseover`, `onkeypress`, etc.) offer rich support for keyboard and mouse events,
- ECMA-Script is a Turing-complete programming language, and
- the DOM interface allows to change XML documents arbitrarily.

The scripting approach has several drawbacks.

As a first problem, a standard interface for styling HTML exists but there is no standard styling interface for XML data to the best of the author's knowledge. In HTML so-called *Intrinsic HTML attributes* can appear in the context of an HTML element. In XML, there is a fallback construct for such cases called processing instructions (PI). Such PIs allow to link external technologies to XML documents. However, evaluating PIs is sparsely supported in Web browsers.

As a second problem, dynamic styling encoded in an HTML intrinsic event attribute is quasi interwoven with the HTML content making maintenance difficult. In practice such DHTML Web pages are therefore often generated using proprietary approaches. Use cases like *coloring each `span` element, if it is clicked* would be hard to implement without generating DHTML Web pages because query language like XPath [CD99] are not part of the DHTML technologies. Hence querying always means programming scripts using DOM instead of writing declarative queries[4].

As a third problem DOM turns out to be not suitable for accessing XML data in order to achieve dynamic document rendering. The granularity of DOM is too fine. DOM supports more than ten so-called nodes ranging from an XML element up to a text node, that need to be considered in scripts. For instance, inserting whitespace between two adjacent XML elements means inserting a new DOM text node. Hence, the position of the second XML element as a DOM child node is increased. Such effects make programming correct queries rather difficult.

As a forth problem another difficulty of DHTML is, that software producers tend to support their own interface to XML documents. The following table[5] shows a comparison of functions of the W3C DOM interface and their Microsoft counterparts. Obviously, such differences complicate providing interoperable DHTML pages on arbitrary Web browsers in practice.

Method or property	IE 5 Win	IE 6 Win	IE 5.2 Mac	Mozilla 1.75	Safari 1.3
`insertRule()` (W3C)	No	No	error	Yes	No
`addRule()` (MS)	Yes	Yes	No	No	No
`deleteRule()` (W3C)	No	No	No	Yes	No
`removeRule()` (MS)	Yes	Yes	No	No	No

Figure 5.2: Support of selected functions of the DOM interface and their Microsoft counterparts in Web browsers.

The DHTML approach can express everything CSS^{NG} can and much more but according to the problems discussed above CSS^{NG} provides

[4]This restriction is by-passed in the prototypical CSS^{NG} engine in Chapter 6 by an XPath processor implemented in ECMA-Script.

[5]`http://www.quirksmode.org/`

- a standard interface for styling XML documents,
- separation of content from static and dynamic design, and
- declarative styling rules that can be adopted rather easily

The freedom of DHTML to calculate also complex styling problems in terms of computability is bought dearly: DHTML styling specifications are rather hard to implement and to maintain by comparison to CSS^{NG} style sheets specifying the same styling.

5.2 Action Sheets

Action Sheets [AEGR98] eliminate some of the drawbacks of the DHTML approach. Action Sheets provide a mechanism for separating or in other words factoring out event-based behavior from the structure of HTML and XML documents. This is similar to the way in which style sheets provide a separation between visual presentation properties and document structure. This concept allows a document author to introduce script-based event handling into an XML document, without modifying the document.

The following example gives an impression of Action Sheets. The example is taken from the specification of Action Sheets and is re-implemented using CSS^{NG}.

Action Sheets:

```
<!DOCTYPE actionsheet SYSTEM "asheet.dtd" [] >
<actionsheet>
   <action type="text/css" codetype="text/javascript">
      .collapsible { onClick: "changeVisibility(event)" }
   </action>

   <script type="text/javascript">
      function changeVisibility(event) {
        var list = event.target.nextSibling;
        var style = list.style;
        if (style.display == "none")
           style.display = "block";
        else
           style.display = "none";
      }
   </script>
</actionsheet>
```

CSS^{NG}:

```
.collapsible:onclick(2n+1) + * { display: none; }
.collapsible:onclick(2n+2) + * { display: block; }
```

Figure 5.3: Collapsible List Use Case of Action Sheets re-implemented in CSS^{NG}.

The Action Sheet in Fig. 5.3 lets fold or unfold the next sibling of an XML element being part of the class `collapsible`. Like in the DHTML approach, scripting in combination with the DOM interface is used to achieve dynamic document rendering features. One of the main drawbacks of the DHTML approach is eliminated by using the CSS selector mechanism in line 4 of the Action Sheet in Fig. 5.3. The approach is still very expressive but at the same time it is still quite difficult to read the code.

The Action Sheet example is re-implemented using CSS^{NG} below. Only two lines of code are needed to provide the same dynamic styling. This comparison is not fair in some sense because specialized languages can offer concise constructs because of their restrictions. However, CSS^{NG} provides concise constructs for many requirements in the practice of dynamic styling, and moreover offers a scripting interface via markup insertion (see Section 4.2).

One strength of CSS^{NG} becomes clearly visible in the example above. Dynamic selectors (using `:onmouseclick`) can be combined in CSS^{NG}. Rather complex workarounds as implemented in the Action Sheet function `changeVisibility` are not needed here.

5.3 Behavioral Extension to CSS

The so-called Behavioral Extension to CSS [AGW99] , a derivative of Action Sheets [AEGR98], is a proposal for extending CSS toward dynamic styling features. Like in the Action Sheet approach the main idea is to separate scripts from content using the selector mechanism of CSS. The novelty of the Behavioral Extension to CSS approach is to specify events in the declaration of a CSS rule.

Rather simple dynamic tree patterns of CSS^{NG}, as demonstrated in Section 4.5.2, can only be simulated in the Behavioral Extension of CSS using rather complicated scripts. The following use case of Behavioral Extension of CSS is taken from the working draft of the W3C:

Behavioral Extension of CSS:

```
.Rollover {
  border     : thin solid blue;
  onmouseover: "this.src=
    this.getAttribute('oversrc');
    this.style.borderColor= 'red';
    statusText.data=
      this.getAttribute('status');"
  onmouseout : "this.src=
    this.getAttribute('outsrc');
    this.style.borderColor= 'blue';
    statusText.data= '';" }
```

CSS^{NG}:

```
.Rollover {
  border:thin solid blue;}
.Rollover:onmouseover {
  borderColor:red; }
.Rollover:onmouseout {
  borderColor:blue; }
```

Figure 5.4: Comparing the Behavioral Extension of CSS and CSS^{NG}.

The right side of the example above shows a CSS^{NG} style sheet that re-implements the style sheet implemented with the Behavioral Extension of CSS on the left side. This example demonstrates how scripting, which is also possible in CSS^{NG} via insertion of markup, can be avoided in many use cases (see [Wie05] for more use cases).

CHAPTER
SIX

Prototype of a CSS^{NG} engine

6.1 Requirements

The prototypical CSS^{NG} engine is designed according to requirements stated in this section. The main goal of the prototype is to provide a proof-of-concept implementation of CSS^{NG}, which can serve for demonstrating CSS^{NG}. The prototype of a CSS^{NG} engine is supposed to process a CSS^{NG} style sheet and an XML or HTML document for rendering issues.

For reducing implementation effort and for allowing demonstrations on various operating systems, the prototype is supposed to draw on established and widespread open source technologies. Since CSS^{NG} does not provide new “graphical features” like new borders or transparent shapes, the CSS^{NG} prototype can draw on an existing rendering engine.

The focus of the prototype is not to provide an efficient implementation of the CSS^{NG} engine. Therefore there are no peculiar requirements concerning memory or response time of the CSS^{NG} engine.

In fact, the challenge of the prototype is to develop a system that can be easily maintained and that is robust to unexpected changes caused by adjustments of the CSS^{NG} language. Convenient tools and data structures for debugging issues are indispensable.

6.2 Implementation of the CSS^{NG} Prototype

This section discusses alternative ways of extending a Web browser and its rendering engine toward interpreting static and dynamic extensions of CSS such as proposed in this thesis. Basically, two ways of extending a Web browser are possible: Extending the code of a rendering engine or influencing the output of a Web browser by using a scripting language such as ECMA script. Each of these alternatives is suited for different purposes as follows.

6.2.1 Modifying an Existing Rendering Engine toward CSS^{NG}

Changing the rendering engine of a Web browser assumes that its source code is available and can be compiled. The source code of proprietary Web browsers like the Microsoft Internet Explorer is usually not available and, therefore, such rendering engines are not amenable to our extensions. Open source browsers like the Mozilla Web browser or the Konqueror of KDE, however, can be extended in that way.

The following tables give a review on current proprietary (see Fig. 6.1) and open source (see Fig. 6.2) rendering engines and their usage in Web browsers:

Rendering Engine:	**Web browser:**
Trident	for Internet Explorer on Windows
Tasman	for Internet Explorer on Macintosh
Presto	for Opera 7 and above, Macromedia Dreamweaver MX and above, and Adobe Creative Suite 2

Figure 6.1: Proprietary and, hence, ineligible technologies.

The following technologies are open source software and are therefore basically suitable for extensions of the source code:

Rendering Engine:	**Web browser:**
Gecko	for Firefox, Camino, Mozilla Application Suite, and other Gecko-based browsers
Amaya	experimental rendering engine of the W3C
WebCore	for Safari and OmniWeb based on KHTML
GtkHTML	rendering/editing library to be easily embedded into applications that require lightweight HTML functionality and based on KHTML
KHTML	for Konqueror
Swing	rendering engine for Java applications

Figure 6.2: Open source rendering engines.

Gecko implements CSS 2.1 and parts of CSS 3. This rendering engine is implemented in C++ and, hence, it can be used on many operating systems such as Windows, Linux, and Mac OS. The source code is almost not commented and the documentation on the CSS rendering engine is outdated since March 2002. Furthermore, problems in the source code especially concerning dynamic rendering are known. Those structural problems of the source code are meant to be solved in one of the next versions[1] . According to the documentation, the implementation of the dynamic styling

is probably the biggest flaw in the design of this part of the system[2].

Due to several optimizations concerning the rendering process, the source code is hard to understand.

Amaya is an experimental Web browser of the W3C. It is used as sandbox to develop new standards and languages by the W3C. For instance, many languages such as SVG or MathML were available in Amaya at first. The source code is written in C++. The structure of the code seems to be quite clear. Tests, however, showed that some CSS properties are not correctly implemented. In order to extend Amaya its code would need a revision concerning the standard of CSS 2.1.

WebCore and GtkHTML are based on KHTML, which is the rendering engine of KDE[3]. Therefore, both rendering engines are subsumed in this paragraph. KHTML is implemented

[1]`http://www.heise.de/newsticker/result.xhtml?url=/newsticker/meldung/66968&words=Gecko`

[2]`http://www.mozilla.org/newlayout/doc/style-system.html`

[3]`http://www.kde.org`

in C++ and supports CSS 2.1. Furthermore, major parts of CSS 3 such as most of the selectors are supported. The source code is well documented and seems to be well suited for programming extensions. Since KHTML is based on KDE, it needs a common X Window environment[4], and can be ported to any operating system having a C++ compiler.

Java Swing[5] offers an HTML rendering engine, which can be used in most operating systems. CSS is supported, too, but only in version 1. An implementation of the extensions proposed in this thesis would require for an implementation of many features of CSS 2.1 and CSS 3 such as dynamic pseudo-classes because many of the extensions are based on CSS 2.1 and CSS 3.

The author is convinced that extending the implementation of KHTML is the best choice, if deciding in favor of implementing the extension toward CSS directly in a Web browser for the following reasons: The time needed to become acquainted with the source code of KHTML seems to be the shortest by comparison to the other engines listed above. Since (to the best of the author's knowledge) no other rendering engine provides better support of CSS 3, the implementation of the extensions can be based on existing code easily and therefore save much implementation effort.

Extending one of the rendering engines mentioned above toward our static and dynamic extensions of CSS yields a proof of concept and provides with a software that is suitable for the purpose of demonstrations. Much time, however, is needed for recognizing the existing source code to find the interfaces for extensions. Experiences in the field of Software Engineering showed that recognizing foreign code takes 4/5 of the time in a software project. Another disadvantage is the lack of interoperability as follows: Since Web browsers are using different rendering engines, an implementation of our extensions cannot be used in other Web browsers.

6.2.2 Using DHTML for extending a Web browser toward CSS^{NG}

An alternative to extending the rendering engine of a Web browser directly, is to use a scripting language such as ECMA script, while the implementation of the used Web browser stays the same. In other words, DHTML is an alternative approach to extending a Web browser toward dynamic document rendering features.

Obviously, it is assumed that an implementation of the same scripting language is available in all Web browsers. Since all of the technologies discussed in the latter section offer an ECMA script implementation, an implementation of the extensions offers higher interoperability by comparison to a direct implementation.

As consequence, proprietary Web browsers like the Microsoft Internet Explorer can profit from our extensions. Furthermore this alternative is well suited for rapid prototyping because the source code does not need to be compiled. The time for implementing our extensions that way is most likely significantly shorter because no foreign source code needs to be recognized in advance.

An implementation of our extensions using ECMA script in a Web browser can be realized in several stages as follows. Obviously, the style sheet of a Web page needs to be transformed to a version of CSS supported by the Web browser in order to be interpreted correctly. Our extensions can be classified into the following groups:

- Extensions needing a source-to-source transformation from CSS^{NG} to current CSS only.

[4]`http://www.x.org/`

[5]`http://java.sun.com/products/jfc/tsc/articles/index.html`

- Extensions needing CSS source-to-source transformation and a change of the source document.
- Extensions needing CSS source-to-source transformation and the ECMA script event model.
- Extensions needing changes on the CSS style sheet, the source document, and the event model of ECMA script.

6.2.3 Choice of Technologies for the CSS^{NG} engine

For achieving the goals of the project as declared in Section 6.1, the author decided in favor of implementing the prototypical CSS^{NG} engine according to the DHTML approach (see Section 6.2.2. The main reasons for this decision are

- platform independence,
- support for rapid-prototyping, and
- longevity.

The point longevity aims at the rather fast development in software engineering. In particular the implementation of Web browsers needs to be changed very often mainly because of security reasons. The CSS^{NG} prototype implemented for the specific version of a Web browser would most likely be outdated soon.

This approach also offers good opportunities for debugging, if using the Mozilla Firefox Web browser. Firefox offers an extension facility where the Java Script Debugger Venkman[6] can be plugged in. Furthermore Firefox has a Java Script console providing a user interface for error messages and for interpreting single lines of code. A third factor making Firefox a good choice is the so-called DOM inspector, which allows to examine changes on the DOM tree of a document rendered in a browser[7]. Finally, Firefox is reckoned as respecting W3C standards.

It is self-evident, that using DHTML for the CSS^{NG} engine requires pre-processing of the input document of the engine and of the CSS^{NG} style sheet. Since the input document is always in XML format, standard W3C processors can be used for pre-processing. The standard technologies of the W3C concerning XML transformations are XSLT [Cla01] and XQuery [BCF$^+$05]. Certainly both technologies are appropriate to master the requirements in this project. However, XSLT has the advantage that its template mechanism allows for implementing transformations that are rather similar to the cascading feature of CSS, therefore XSLT was chosen.

XSLT processors are implemented in many programming languages but in particular in Java and C++. Since XSLT programs do not depend on the implementing language of the XSLT processor, the choice of the XSLT processor is of little importance. However, an XSLT processor implemented in Java allows an easy client-side adoption of XSLT processing using a so-called Java Applet[8]. Hence, the CSS^{NG} engine could run server-side and client-side.

[6]`http://www.mozilla.org/projects/venkman/`

[7]Note that the changes on the DOM tree in a Web browser do change the rendered document

[8]`http://java.sun.com/applets/`

Tests using the upcoming AJAX technology implementing XPath, XSLT, and DOM in ECMA Script failed. The author tested the AJAX frameworks Sarissa 0.9.6[9] and Google AJAXSLT[10] . Needed functionalities of XSLT were not correctly implemented. However, a stable implementation of AJAX would be a good choice for implementing a CSS^{NG} engine.

In contrast to the input document of a CSS^{NG} engine, the CSS^{NG} style sheet cannot be processed using W3C technology because it is not provided in XML format. Section 6.3.1 and 6.3.2 introduce to the translation of plain text from the W3C's point of view to XML. For simplicity reasons the choice of the concerning technologies is discussed there.

6.3 Architecture

The CSS^{NG} extension of CSS 3 is planned and implemented for proving the concept of CSS^{NG}. Therefore we draw on established components for getting a transparent and easily scalable prototype instead of implementing a high-performance extension of a single Web browser.

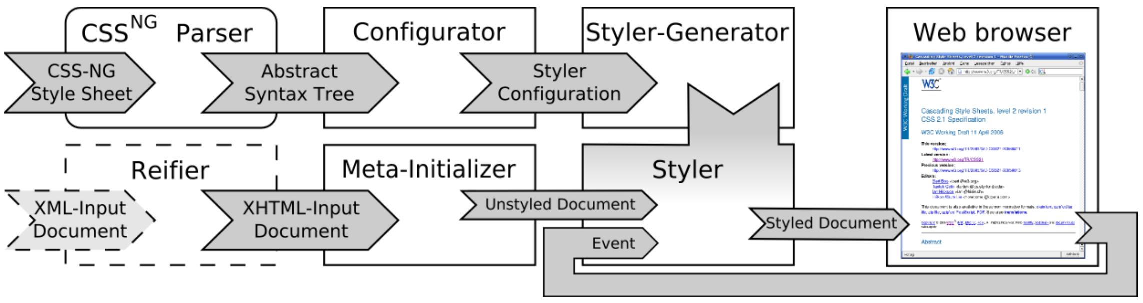


Figure 6.3: CSS^{NG} styling of an X(HT)ML document and rendering.

This system compiles XHTML as well as XML 'Input Documents' according to rules in 'CSS^{NG} style sheets' for rendering in standard Web browsers such as Mozilla Firefox [11] or MS Internet Explorer [12].

The upper row in Fig. 6.3 manages the compilation of an input 'CSS^{NG} Style Sheet' to a 'Styler' (see Section 6.3.4). On the lower row, this 'Styler' is responsible for compiling a preprocessed (see Input Preprocessing below) 'XML input document' to a 'Styled Document' that can be rendered by the Web browser. All further *dynamic* styling activities such as triggered by mouse clicks in a Web browser window update meta-data of the 'Styled Document'. Changes on these meta-data are evaluated by the 'Styler' and are finally rendered by the Web browser. The following sections address the components of the CSS^{NG} prototype in detail.

6.3.1 CSS^{NG} Lexer

The Lexer `lexer` of the CSS^{NG} prototype is based on the specification of CSS 2.1[13]. Since the specification of CSS 3 is almost finished but not yet stable and the CSS 3 grammar is very

[9]`http://sourceforge.net/projects/sarissa`
[10]`http://goog-ajaxslt.sourceforge.net/`
[11]`http://www.mozilla.com/firefox/`
[12]`http://www.microsoft.com/windows/ie/default.mspx`
[13]`http://www.w3.org/TR/CSS21/grammar.html`

complex, this thesis extends the grammar of CSS 2.1 toward CSS^{NG} constructs. However, all needed CSS 3 features for the use cases are embedded in the grammar of CSS^{NG}.

The CSS 2.1 grammar is supposed to use a Flexible Lexical Scanner (Flex)[14] syntax. When trying to compile the W3C specification using JFlex [15] it turned out that the CSS 2.1 grammar needed to be modified for the needs of JFlex as described in Fig. 6.4. The lack of documentation made these differences hard to discover. Examining the source code of JFlex was the only chance to compensate the missing documentation. See Appendix A.1 for the complete Lexer specification.

Semantic	Flex	JFlex
Case sensitivity	`%option case-insensitive`	`%ignorecase`
Separation of head and body of rules	white space	`=`
Escaping of quotes	`\\"`	`\"`

Figure 6.4: Flex vs. JFlex syntax.

JFlex was chosen instead of the original implementation Flex because it is implemented in Java. In the context of Web browsers this is an advantage because Java programs can be used easily as client software wrapped in so-called Java Applets.

Only few new tokens needed to be inserted in the existing specification of the CSS 2.1 specification. All extensions are marked via comments in Appendix A.1. For debugging purposes all tokens of the CSS^{NG} lexer yield values in contrast to the original CSS 2.1 lexer.

6.3.2 CSS^{NG} Parser

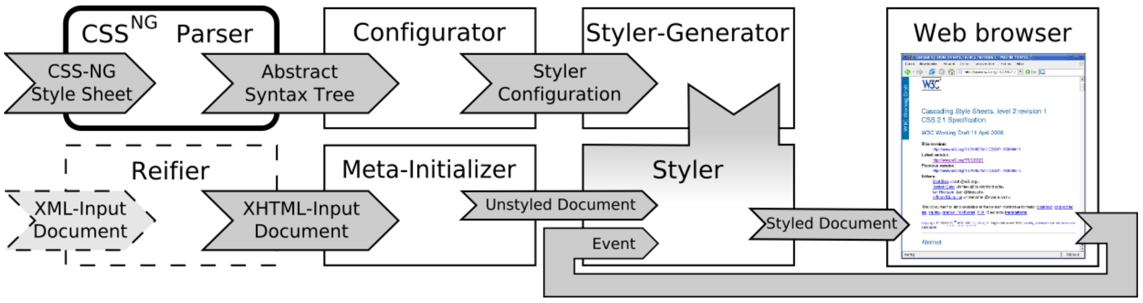


Figure 6.5: CSS^{NG} Parser with implicit lexer.

Analogous to the CSS^{NG} lexer the design of the CSS^{NG} parser tries to stay as close as possible to the W3C standards for writing specifications. Therefore, this thesis uses a Yacc Parser whose extended syntax was used to specify the CSS 2.1 grammar (Code given in Section A.2). The extensions of the Yacc syntax used by the W3C are analogous to the extensions of the Extended Backus-Naur Form (EBNF) to the standard Backus-Naur form. These extensions allow for a more concise description of the CSS 2.1 grammar.

It was not possible to take over grammars of open source rendering engines of standard Web browsers like Gecko or KHTML because the grammars diverge from the standard W3C grammar for CSS.

[14]`http://dinosaur.compilertools.net/`
[15]`http://www.jflex.de`

6.3.2.1 Translation of the CSS Grammar to Yacc syntax

Since Yacc only supports BNF constructs, the CSS 2.1 grammar needed to be translated to BNF. Hence the following EBNF constructs needed to be re-implemented by BNF constructs:

- `*`: 0 or more
- `+`: 1 or more
- `?`: 0 or 1
- `[ ]`: grouping

```
S*
                         is implemented by
s_star
  : /* empty */
  | /* recursion */ s_star S
```

Figure 6.6: Translation of EBNF in BNF.

The example on the right side of Fig. 6.6 exemplarily shows how an EBNF expression is translated to BNF. The plus operator (`+`) is analogously implemented having the symbol itself as base case (in this case `S`). The question mark is another modification of the example in Fig. 6.6 offering no recursion (`S` instead of `s_star S`). The grouping construct of the CSS 2.1 grammar is implemented by factoring out the content of a group being in brackets to new rules.

For a better orientation the new rules have generic names denoting their origin and details of the reason for the new rule. For instance the rule with the head `import_ext2opt` implements an `opt`ional non-terminal (lower case) `import` being the second new rule in the context of the origin rule body of `import`. Additionally to `opt`, the keywords `plus` for `+` and `paren` for grouping are used. Nested EBNF constructions are implemented using nested extensions to non-terminals such as `import_ext2opt_ext1star`. New rules are always placed indented after their origin rule except if the terminals and non-terminals are very often used. Rules for often used terminals and non-terminals are placed after the CSS 2.1 grammar. The extensions to the CSS 2.1 grammar can be found in the end of the specification. Obviously, these extensions are linked in the original CSS 2.1 grammar.

Note that control characters of XML like < or > are escaped during the parsing process.

6.3.2.2 Abstract Syntax Tree (AST)

The CSS^{NG} Parser generates a so-called Abstract Syntax Tree (AST) . This tree is the first representation of the CSS^{NG} style sheet in XML format. The grammar of the AST corresponds to the grammar of CSS^{NG}. Terminals and non-terminals are represented by equally named XML elements. Obviously, non-terminals are represented by XML elements having non-terminals or terminals as children. Terminals at the end of a branch in the AST have the corresponding content from the former CSS^{NG} style sheet. In other words, without the XML elements the AST looks like the former CSS^{NG} style sheet.

This technique is rather convenient for developing such a prototype especially because the CSS 2.1-based CSS^{NG} grammar has oddities like white space being a token. The resulting AST from a parsing process can be debugged easily. For a highly efficient implementation of the CSS^{NG} modern table-based techniques need to be used.

An example of the input and the output of such a parsing process can be seen in the following Fig. 6.7:

Part of a CSS^{NG} style sheet:

```
... background-color: ...
```

Abstract Syntax Tree:

```
...
<declaration>
  <property>
    <IDENT>background-color</IDENT>
  </property>
  <COLON>:</COLON>
...
```

Figure 6.7: Parsing of a part of a CSS^{NG} style sheet to an AST.

6.3.3 Configurator

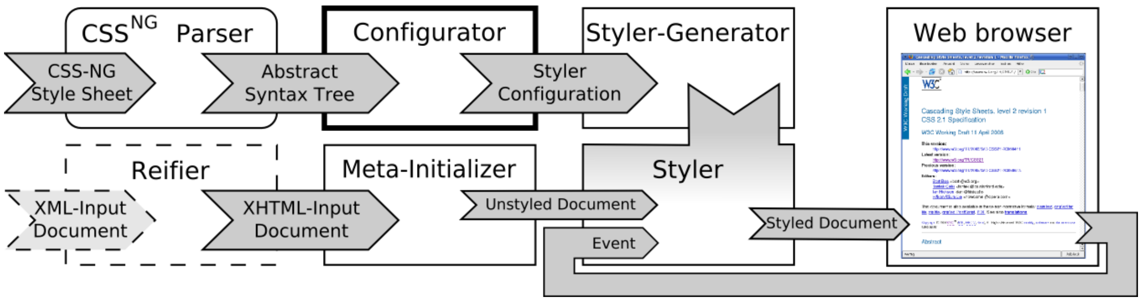


Figure 6.8: Configurator.

The Configurator is a preprocessor generating a human-readable representation of a CSS^{NG} Abstract Syntax Tree. The resulting Styler Configuration (Code given in Appendix A.3) is used to generate a transformation that applies the CSS^{NG} style-sheet to an XHTML document (or a reified XML document).

The structure of the Styler Configuration (see Fig. 6.9) is similar to the structure of a CSS^{NG} style sheet. The root XML element `stylesheet` has a list of `rules` as children. Each rule offers structured styling information, which are optimized for the configuration of the Styler.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<stylesheet>
   <rule>
      <selector> ... </selector>
      <declaration> ... </declaration>
      <insert type=" ... "> ... </insert>
   </rule>
   ...
</stylesheet>
```

Figure 6.9: Main Structure of a Style Configuration.

6.3.3.1 Translation of CSS^{NG} selectors to XPath

One of the major tasks of the Configurator is translating CSS selectors to XPath[16]. Such XPath selectors are needed for the Styler component of the prototype. This Styler checks for each XHTML element of its input document whether a CSS^{NG} rule needs to be applied or not.

CSS selectors are very similar by comparison to XPath expressions. A translation is therefore rather simple as the following table demonstrates.

CSS Selector:	XPath expression:
`a[att='x']`	`a[@att='x']`
`.folded`	`*[@class='folded']`
`#42`	`*[@id='42']`
`#42[att='x']`	`*[@att='x' and @id='42']`
`a b`	`a/descendant::b`
`a > b`	`a/child::b`
`a + b`	`a/following-sibling::b`

Figure 6.10: Translation of CSS selectors to XPath expressions.

For efficiency reasons the Styler, while checking XML elements for matching CSS^{NG} rules, does not evaluate XPath expressions from the root of the document. Each XML element is checked locally during a traversal of the Styler through the complete input document. Hence, the XPath expressions needed for the Styler needed to be “flipped”. Such a flip affects only the CSS combinators in the second part of the table in Fig. 6.10. CSS combinators are simply translated to the reverse XPath axis and the order of simple selectors and combinators is reversed as follows:

CSS Selector:	XPath expression:
`a b`	`self::b[anchestor::a]`
`a > b`	`self::b[parent::a]`
`a + b`	`self::b[preceding-sibling::a]`
`a b c`	`self::c[anchestor::b/anchestor::a]`

Figure 6.11: Translation of CSS selectors to XPath expressions.

6.3.3.2 Declaration

The `declaration` XML element is not specially structured. It contains the declaration as provided by the CSS^{NG} style sheet with one exception: The specification of markup insertion (see Section 6.3.3.3) is separated from the declaration part of a CSS^{NG} in the Styler Configuration. This design decision separates declarations that cannot be interpreted by a standard Web browser from standard CSS 3 declarations. The example in Fig. 6.12 shows the representation of a rather simple CSS^{NG} rule in the Styler Configuration format.

[16]The CSS^{NG} combinator `?` can not be translated to XPath.

```
* { color:red; background-color:green }
                                                         is represented by
<?xml version="1.0" encoding="ISO-8859-1"?>
<stylesheet>
   <rule>
      <selector>self::xhtml:*</selector>
      <declaration>color:red;background-color:green</declaration>
   </rule>
</stylesheet>
```

Figure 6.12: Rather simple CSS^{NG} rule represented in Styler Configuration Format.

6.3.3.3 Markup-Insertion

Markup Insertion is a bit complex by comparison to the specification of the declaration. First of all, the position of the insertion before or after an XHTML element is specified by a conforming value of the XML attribute `type` (see Fig. 6.13). Markup that does not depend on the input X(HT)ML document of the CSS^{NG} engine like the XHTML `div` element in Fig. 6.13 is directly materialized in the Styler Configuration.

Other markup insertions depend on the input document as caused by the CSS^{NG} function `element-name`, `attribute-name`, and `attribute-value`. Such insertions are encoded using XML elements as place-holders, which are specially translated by the Styler-Generator.

```
*::before { content: element("div",
                             attribute("class", "tab"),
                             element-name()); }
                                                         is represented by
<?xml version="1.0" encoding="ISO-8859-1"?>
<stylesheet>
   <rule>
      <selector>self::xhtml:*</selector>
      <declaration/>
      <insert type="before">
         <div xmlns="http://www.w3.org/1999/xhtml" xhtml:class="tab">
            <ELEMENT_NAME xmlns=""/>
         </div>
      </insert>
   </rule>
</stylesheet>
```

Figure 6.13: Insertion of tabs.

6.3.4 Styler-Generator

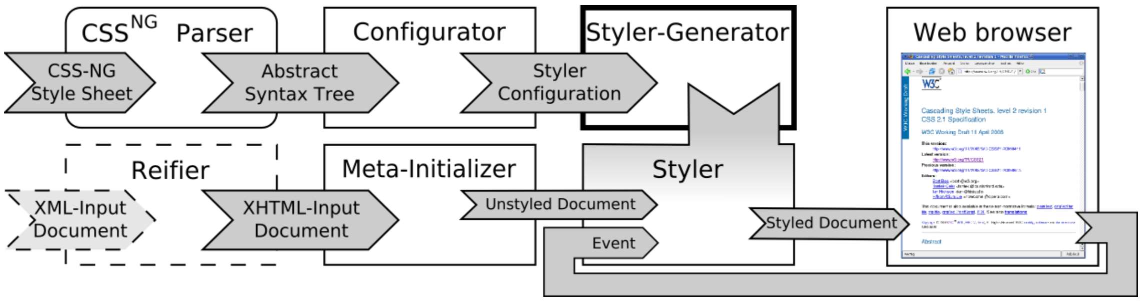


Figure 6.14: Styler-Generator.

According to Fig. 6.3 the next logical step is the Styler-Generator (Code given in Appendix A.4). Since this processor is conceptually rather complex, we give a brief introduction to the output of the Styler-Generator to give a more gentle access to this complex topic.

The Styler is a processor applying styling declarations of a CSS^{NG} style sheet to an X(HT)ML input document. Hence, there is no generic Styler for all CSS^{NG} style sheets. Depending on the CSS^{NG} the Styler is generated but can be applied without restriction to arbitrary XHTML documents.

During a styling process each XHTML element is visited in XML document order (see [BPSMM00]). Now, for each rule of the CSS^{NG} style sheet is tested whether its selector matches the element. If the selector matches the current XHTML element, the rule declaration of the CSS^{NG} rule is appended to the value of the XHTML `style` attribute of the element. Hence, former declarations are not lost but regarded in the styling process via the cascading feature of CSS^{NG}.

The content of the `style` attribute of an XHTML element describes its current styling. Inactive styling declarations depending on events are only inserted, if required. Such dynamic styling information is encoded in special meta-data.

Fig. 6.3 implies that the Styler-Generation is a special transformation. Like the other transformations in the diagram the Styler-Generator has an XML input stream that is transformed. The anomaly is the output. The output is also XML data but this XML data is a transformer itself. That makes the Styler Generator a kind of compiler compiler.

6.3.5 Reifier: Representing XML in XHTML

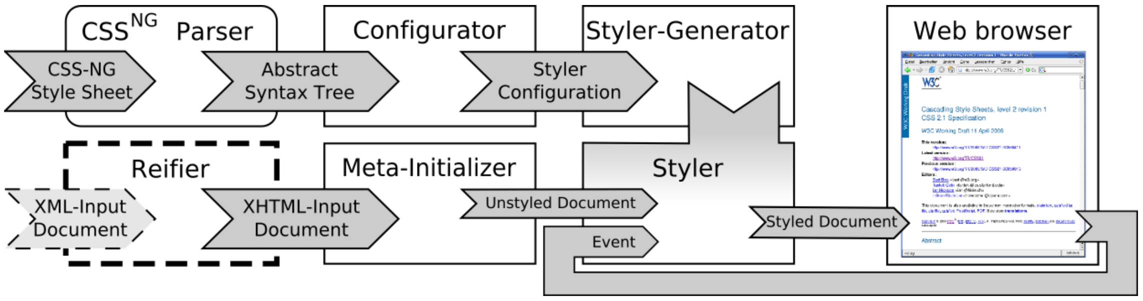


Figure 6.15: Reifier.

Since standard Web browsers generally do not offer rendering of XML data and furthermore they do not offer a standard interface for applying scripts (see Section 5.1), we reify XML documents to XHTML documents (Code given in Appendix A.6).

The challenge of reifying XML documents in this prototype is modeling all capabilities of XML in XHTML, an instance of XML. At the same time the cascading feature of CSS needs to be respected to avoid an expensive re-implementation of the cascading feature in the prototype.

According to the specification of XHTML [PAA$^+$00] no additional elements or attributes can be added to the language. Hence, only existing elements and attributes in XHTML can be used to encode general XML documents.

Several XHTML attributes are stated in the specification as the following table shows:

Classification	Name	Content Type
core	`id`	unambiguous ID
	`class`	CSS styling class
	`style`	CSS declarations
	`title`	tool-tip content
i18n	`lang`	language code
	`xml:lang`	language code
	`dir`	text orientation
intrinsic event attributes	`onclick`	interpretable script code

Figure 6.16: XHTML attributes that can occur in the context of every XHTML element.

The only two attributes admitting unrestrained content are `class` and `title`. However, the XHTML attribute `title` has an unwanted side-effect. If the mouse cursor hovers over an XHTML element with defined `title` attribute, the value of the `title` attribute is superimposed in a small window. This window is a so-called tool-tip. Hence, regarding XHTML attributes only the `class` attribute is left for encoding XML data.

XML data like tag names or attributes can be encoded in the value of the XHTML `class` attribute. Since attributes cannot be nested an encoding scheme would be needed for structuring XML data. An extreme example is shown in the following Fig. 6.17.

```
...
   <body>
      <div class="<a><b>Content of b</b><c/></a>" />
   </body>
</html>
```

Figure 6.17: Encoding XML data in an XHTML attribute.

The example in Fig. 6.17 demonstrates that a reification of XML in XHTML is generally possible. This first approach, however has two major drawbacks: An extra parser is needed in the prototype to decode the XML data and furthermore the cascading feature of CSS needs to be re-implemented.

The advantage of XHTML elements over XHTML attributes is that they pass the cascading property of CSS and that they can have the same content as XML elements. The disadvantage is that the new elements can not be introduced in XHTML. Hence, an adequate dummy XHTML element and a place for encoding the markup (like the XML element name and the XML attribute name) is needed.

Since, XHTML elements are assigned to special purposes (like the `p`-element for encoding a paragraph) the only choice for encoding XML are the XHTML elements `div` and `span` specifying the rendering as block element (like `p`) or as in-line element (like `i` for italic) only. However, CSS offers the property `display` to re-define the rendering of these XHTML elements. We take `div` XHTML element as dummy element for no special reasons.

The place for encoding the markup is still needed. As mentioned above the XHTML attribute `class` is a candidate for encoding arbitrary data. Since the value of the `class` attribute means linear modeling, once again a parser in the prototype would be needed to decode a String containing information about the XML element name and XML attributes.

Therefore, we encode the XML markup using XHTML elements and XHTML attributes. This solution allows for structuring markup data with an easy access to the markup via query languages like XPath [CD99] as consequence. Furthermore the cascading feature of CSS does not need to be re-implemented in the prototype.

Source XML:

```
<book year="1994">
  <my:title>TCP/IP Illustrated</my:title>
</book>
```

reified XML:

```
<div>
  <span class="element">
    <span class="name">book</span>
  </span>
  <span class="attribute">
    <span class="name">year</span>
    <span class="value">1994</span>
  </span>
  <div>
    <span class="element">
      <span class="namespace">my</span>
      <span class="name">title</span>
    </span>
    TCP/IP Illustrated
  </div>
</div>
```

Figure 6.18: Reification of XML data to XHTML.

The example of a reification shown in Fig. 6.18 demonstrates the reification of sample XML data. As mentioned above the XHTML `div` element is used as dummy element. The corresponding markup information is encoded in child XHTML `span` elements where the XHTML `class` attribute serves as key for selecting markup information like the attribute value. Obviously, the XHTML `span` elements must not be rendered in a Web browser. This behavior can be specified using a CSS rule like `span { display:none; }`.

6.3.6 Meta-Initializer

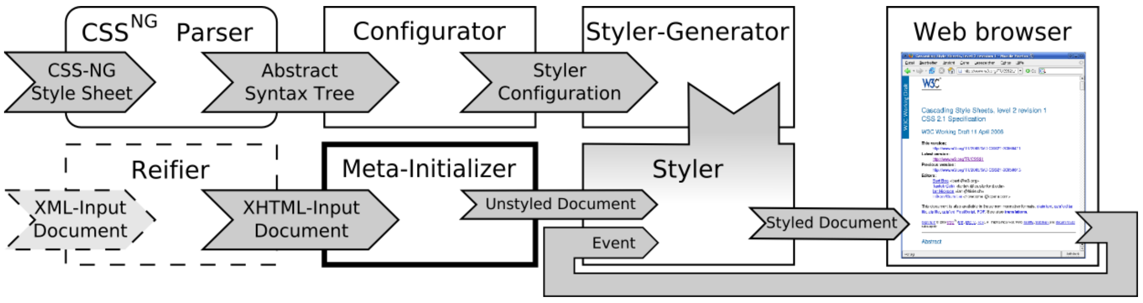


Figure 6.19: Meta-Initializer.

XHTML input documents can be initialized directly without reification: The 'Meta-Initializer' installs listeners and histories for relevant XHTML elements (Code given in Appendix A.7). An XHTML element is called dynamically relevant with respect to the **CSS^{NG} Style Sheet**, if a dynamic rule (see Dynamic Styling) defines its styling.

6.3.7 Dynamic Styler

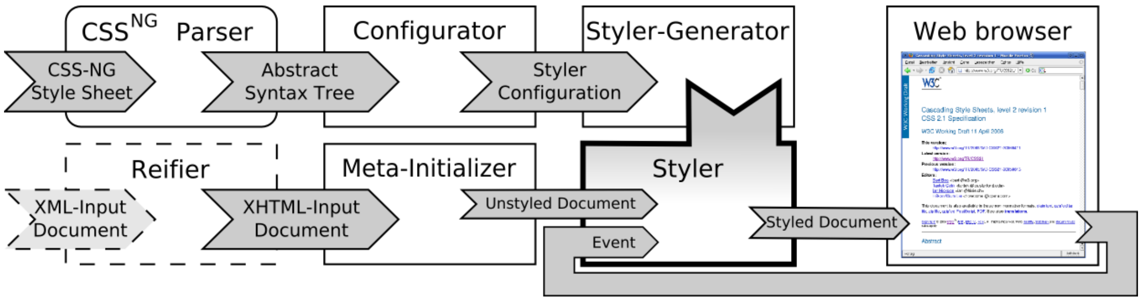


Figure 6.20: Styler.

The precondition for dynamic styling in the CSS^{NG} engine is the 'Styled Document' as produced by the static lefthand part (see Fig. 6.3) of the CSS^{NG} engine. This section focuses on the dynamic styling part while viewing a document in a Web browser (Code given in Appendix A.8). An example of such a 'Styled Document' is shown in the following Fig. 6.21:

The example in Fig. 6.21 introduces to the main principles of dynamic styling in the prototype. The 'Styled Document' is compiled according to the 'Input Document' and the 'CSS^{NG} style sheet'.

Fist of all new attributes are added to the XHTML element `p`. The attribute `onclick` is the so-called event handler. (Depending on the type of the event any other intrinsic event [ABC^{+}99] can appear here.) The event handler depends on the specification in the CSS^{NG} style sheet. In this case the event `onclick` is of interest and needs to be detected. If an

Input XHTML document:

```
...
<p>Omnia Gallia divisa est in partes tres.</p>
...
```

CSS^{NG} style sheet:

```
...
p:onclick(1) { color: green; }
...
```

Styled Document:

```
...
<p onclick="increment(this,'onclick');commit();" style="">
   <span>
      <span class="standard">
         styleMe('element','self::xhtml:p','onclick(1)','color:green')
      </span>
      <span class="nth-descendant">3</span>
      <span class="nth-child">0</span>
      <span class="onclick">0</span>
   </span>
   Omnia Gallia divisa est in partes tres.
</p>
...
```

Figure 6.21: A CSS^{NG} style sheet is applied to an XHTML input document resulting a Styled Document that can be rendered by a Web browser.

`onclick` event occurs, the function `increment` (implemented in ECMA Script) is called for incrementing the corresponding `onclick` counter in line 9 of the 'Styled Document'.

The incrementation of an event counter, for instance for onclick events, changes the state of the 'Styled Document'. Depending on the current state of the document only the set of matching CSS^{NG} rules can change. In the example of Fig. 6.21 the CSS^{NG} rule can only be applied if an XHTML `p` element was clicked once. Such changes of the state of the 'Styled Document' are evaluated using the function `commit`. Since dynamic styling (for instance using `:onclick(1)`) can affect different portions of an 'Input Document' independently, the function `commit` traverses the whole 'Styled Document' trying to apply CSS^{NG} rules represented by a list of `styleMe` functions.

6.3.7.1 Run-time evaluation of CSS^{NG} rules

Dynamic Styling requires an evaluation of CSS^{NG} rules in a Web browser at run-time (or in other words at view-time). Due to technical restrictions, such an evaluation is rather difficult to implement.

Since the Styler solves the problem of applying a CSS^{NG} style sheet to an XSLT transformation, applying the Styler XSLT program once more would be the easiest way of dynamic styling in our approach. There are two opportunities of applying XSLT transformations to XHTML pages in a Web browser. Firstly, the internal XSLT processor of the Web browser could be used, if existing. However, this option objects to the goal of the implementation being Web browser independent. Secondly, an XSLT processor implemented in ECMA Script could be used. As discussed in Section 6.2.3 the upcoming XSLT processors implemented in ECMA Script are not mature yet.

Although the 'Styler' is the consequent way for fulfilling dynamic styling requirements, the restrictions mentioned above force us to provide a workaround until adequate XSLT processors will be available. Obviously, an extreme workaround would be a complete re-implementation of the Styler XSLT transformation in ECMA Script, which is not realistic at all.

The most challenging problem of applying a CSS^{NG} rule is evaluating the CSS^{NG} selector against the current XHTML document. In pure DHTML this had to be done using the DOM interface. As consequence all CSS^{NG} selectors had to be translated to DOM. However, according to Section 6.3.3 such a translation is rather complex but in contrast to complete XSLT processors implemented in ECMA Script there are implementations of XPath[17] processors only available in ECMA Script that can solve the problem.

An important issue for the choice of the ECMA Script XPath processor is the ability to return references to the DOM nodes selected by an XPath expression[18]. This requirement filters out the XPath processors of AJAX projects because in AJAX projects implement their own DOM model instead of using the DOM model of the current Web browser. This design decision guarantees Web browser independency. To the best of the author's knowledge XPathJS[19] is the only available XPath processor implemented in ECMA Script, that fulfills our requirement. XPathJS seems to work perfectly expcet for one exception: The automatic recognition of the name spaces does not work[20]. An extension of the XPath processor

[17]XPath is used in the XSLT standard for selections

[18]These DOM references are the interfaces for changing styling parameters of match DOM nodes.

[19]`http://mcc.id.au/xpathjs/`

[20]Namespaces in the 'Styler' are needed to avoid naming conflicts between original XML or XHTML input data and meta-data, e.g., specifying the history of events occurred.

(see Appendix A.8) solves this problem.

The minor problem of evaluating CSS^{NG} rules in a Web browser is applying the declaration of CSS^{NG} rules. During the first styling process via the XSLT 'Styler', the set of all dynamic CSS^{NG} rules is preprocessed for each XHTML element that could match in future depending on user interaction with the Web browser. An example of a preprocessed CSS^{NG} rule can be seen in Fig. 6.21.

CHAPTER

SEVEN

Proof-of-Concept Applications of CSS^{NG}

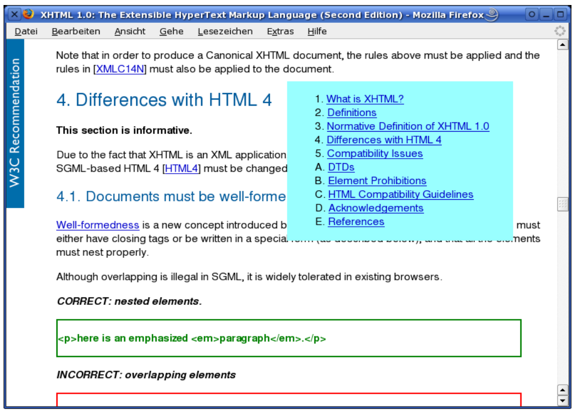
7.1 Rendering of HTML Documents

This section discusses three use cases on how CSS^{NG} can be beneficial for HTML documents. The CSS^{NG} style sheets in this section are applied to the W3C recommendation XHTML 1.0 [PAA^{+}00], which is written as an XHTML document.

7.1.1 Temporarily Superimposing the Table of Contents on Keypress

The table of contents is an important source for orienting oneself in a large document. For instance, recommendations of the W3C give such a table of contents in the beginning of the text. Obviously, this table of contents moves out of sight very soon, when reading text. However, many Web pages use a whole column for the table of contents allowing such navigation tools independently of the current scrolling position in the text. This column approach is not completely sufficient because the space for the column is lost. Furthermore the menu column offers rather often only a few menu items. Hence, the space below it is given away.

```
.toc {
   position: fixed;
   top:      5em;
   right:    5em;
}
.toc:onkeypress('c',2n+1) {
   display: block;
}
.toc:onkeypress('c',2n+2) {
   display: none;
}
```


XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) - Mozilla Firefox
Datei
Bearbeiten
Ansicht
Gehe
Lesezeichen
Extras
Hilfe
W3C Recommendation

Note that in order to produce a Canonical XHTML document, the rules above must be applied and the rules in [XMLC14N] must also be applied to the document.

4. Differences with HTML 4

This section is informative.

Due to the fact that XHTML is an XML application
SGML-based HTML 4 [HTML4] must be changed

4.1. Documents must be well-forme

Well-formedness is a new concept introduced b … must either have closing tags or be written in a specia … ments must nest properly.

Although overlapping is illegal in SGML, it is widely tolerated in existing browsers.

CORRECT: nested elements.

```
<p>here is an emphasized <em>paragraph</em>.</p>
```

INCORRECT: overlapping elements

1. What is XHTML?
2. Definitions
3. Normative Definition of XHTML 1.0
4. Differences with HTML 4
5. Compatibility Issues

A. DTDs
B. Element Prohibitions
C. HTML Compatibility Guidelines
D. Acknowledgements
E. References

Figure 7.1: CSS^{NG} rule superimposing the table of contents on keypress.

The use case of CSS^{NG} in Fig. 7.1 implements superimposing of the table of contents on keypress. The first rule (lines 1-5) specifies basic styling of the table of contents, which is marked by the CSS class `toc` in the XHTML document. If superimposed, the table of contents appears in the upper right corner of the Web browser windows (lines 3-4).

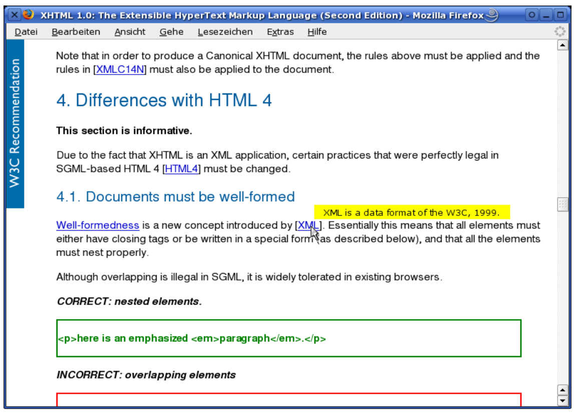
The dynamic rendering of this use case is specified in the second and the third CSS^{NG} rule of Fig. 7.1. The second rule (lines 6-8) makes the table of contents visible, if the key 'c' (table of **c**ontents) was pressed on the keyboard once, thrice, and so on. According to the third rule (lines 9-11) the table of contents is not visible after an even number of key presses on 'c'.

Using a style sheet like this, the viewer of a Web page can superimpose the table of contents on demand and the whole Web browser window can be used for text. Note that in contrast to tool-tips[1] [ABC$^+$99] this technique of superimposing allows to use not only plain text but allows also to use normal markup such as hyperlinks.

7.1.2 Superimposed Notes – Adapting Footnotes to Web Browsers

Texts that are published in traditional media such as books or in digital media like PDF documents use footnotes for giving additional explanations on text portions for instance on the bottom of the same page. Footnotes are rather convenient because additional information can be accessed easily without turning the page. Since, there is no such page concept in HTML documents viewed on Web browsers, footnotes mean scrolling to the bottom of the whole document. Looking at a document as long as the specification of XHTML [PAA$^+$00] this would be rather inconvenient.

```
* {
  position:         relative;
}
*[title]:hover::after {
  content:
   element("div", attr("title"));
}
*[title]:hover + div {
  position:         absolute;
  display:          block;
  top:              1em;
  left:             2em;
  background-color: yellow;
}
```



XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) - Mozilla Firefox

Note that in order to produce a Canonical XHTML document, the rules above must be applied and the rules in [XMLC14N] must also be applied to the document.

4. Differences with HTML 4

This section is informative.

Due to the fact that XHTML is an XML application, certain practices that were perfectly legal in SGML-based HTML 4 [HTML4] must be changed.

4.1. Documents must be well-formed

XML is a data format of the W3C, 1999.

Well-formedness is a new concept introduced by [XML]. Essentially this means that all elements must either have closing tags or be written in a special form (as described below), and that all the elements must nest properly.

Although overlapping is illegal in SGML, it is widely tolerated in existing browsers.

CORRECT: nested elements.

`<p>here is an emphasized <em>paragraph</em>.</p>`

INCORRECT: overlapping elements

Figure 7.2: CSS^{NG} rule superimposing (yellow) notes on keypress.

In this use case (see Fig. 7.2) notes are inserted, if the mouse cursor is hovering over an HTML element having a `title` attribute. The specification of the place of the insertion

[1] Tool-tips are small boxes that can be superimposed, if the mouse cursor hovers over an image or a hyperlink.

is relative (line 9) to the parent HTML element[2]. This type `position`ing allows to insert markup overlaying without displacing the rendering of following HTML elements. In other words, this is relative superimposing. Finally, the CSS^{NG} declaration in the lines 5 and 6 insert the content of the `title` attribute. Note that the CSS declarations in lines 9-13 apply to the inserted part, which in CSS 3 could only be text, but which in this CSS^{NG} example is markup.

The CSS^{NG} rules on the left side of Fig. 7.2 can be seen as the specification of an extended tool-tip [ABC$^+$99] that are specified using the `text` attribute in XHTML. In addition to HTML tool-tips CSS^{NG} allows

- to specify the place, where notes should be inserted,
- to use CSS-styled HTML markup and not only plain text,
- to insert notes on keyboard events or on mouse clicks.

The same technique can be used to superimpose notes as so-called side-notes. Here, notes are superimposed on an extra column (for instance specified using the CSS `margin` property) on the same height as their references. In other words, side-notes are footnotes displayed beside the document.

Note that according to the use cases in this section CSS^{NG} allows to specify stylings for various requirements by only a few variations on the style sheets.

7.1.3 Displaying annotations to Documents

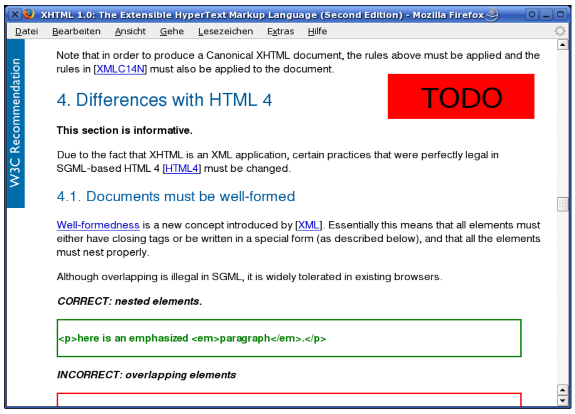
The third use case concerning rendering of HTML documents addresses authors of HTML documents. Some authors tend to mark text portions with a "todo" notice to indicate that the text portion is not finished yet. In practice authors sometimes forget to revisit and delete such labels. CSS^{NG} allows to visualize or not the status of an HTML document still having todo areas as implemented using the following style sheet.

HTML TODO message:

```
<div class="ready">
  TODO
</div>
```

CSS^{NG} style sheet:

```
.todo ? .ready {
  display:          block;
}
```


XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
W3C Recommendation
TODO

Note that in order to produce a Canonical XHTML document, the rules above must be applied and the rules in [XMLC14N] must also be applied to the document.

4. Differences with HTML 4

This section is informative.

Due to the fact that XHTML is an XML application, certain practices that were perfectly legal in SGML-based HTML 4 [HTML4] must be changed.

4.1. Documents must be well-formed

Well-formedness is a new concept introduced by [XML]. Essentially this means that all elements must either have closing tags or be written in a special form (as described below), and that all the elements must nest properly.

Although overlapping is illegal in SGML, it is widely tolerated in existing browsers.

CORRECT: nested elements.

```
<p>here is an emphasized <em>paragraph</em>.</p>
```

INCORRECT: overlapping elements

Figure 7.3: CSS^{NG} rule indicating, if `todo` areas are existing.

The CSS^{NG} combinator `?` in the selector of the CSS^{NG} style sheet in Fig. 7.3 allows to open an HTML TODO message, if *at least one* HTML element belongs to the CSS class `todo`.

[2]Obviously, the naming of `position` values in CSS 3 might be confusing.

Note that this example was already used for motivating *structure-independent selections* in Section 4.6. It is repeated in more detail for a sound demonstration of the features of CSS^{NG} in this section.

7.2 Rendering of a FOAF Definition

This section describes an approach of visualizing a so-called Friend-of-a-Friend (FOAF) [BM05] network as a proof-of-concept application for CSS^{NG}. FOAF is a data model for social networks consisting of `persons`. Information about such `persons` like `name` or `gender` can be specified using attributes of `persons`. Here the word "attribute" denotes a modeling concept of the FOAF data model, which has nothing to do with an "attribute" in XML lingo. One of the FOAF attributes is called `knows` and models which `persons` a given person knows. Putting together such information yields a social network (see Fig. 7.4), which can be expressed using the Resource Description Framework (RDF) [LS99] .

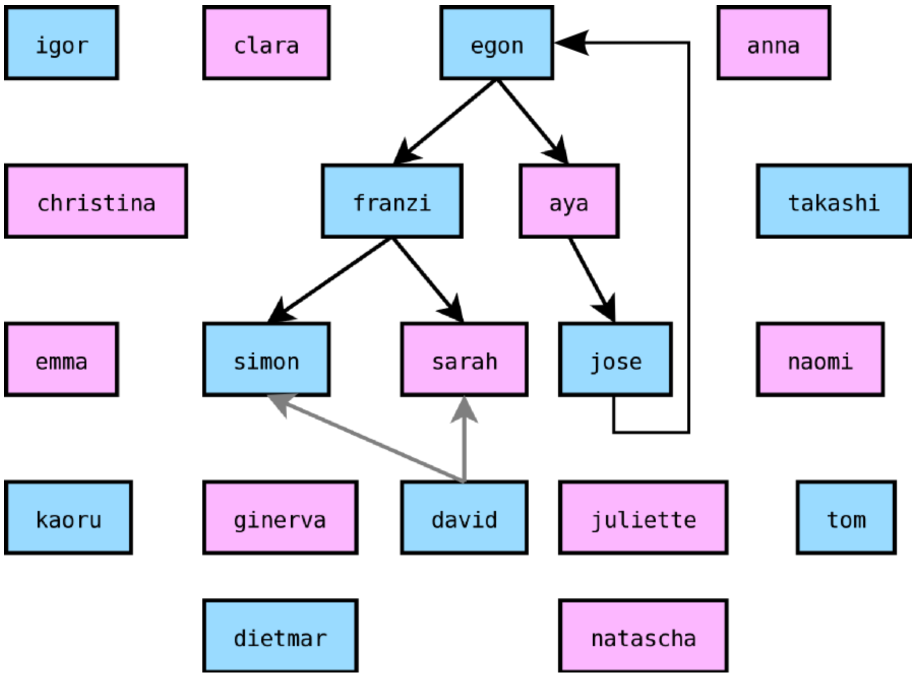


Figure 7.4: Example of a Friend-of-a-Friend (FOAF) network.

RDF is a family of specifications for a meta-data model that is often implemented as an application of XML. The RDF meta-data model is based upon the idea of making statements about resources in the form of a subject-predicate-object expression, called a triple in RDF terminology. The subject is the resource, the "thing" being described. The predicate expresses a relationship between the subject and the object. The object is the object of the relationship. Since RDF documents can describe graphs and not only trees, a smooth two-dimensional visualization of general RDF documents is a rather challenging problem.

7.2.1 A Serialization of RDF data

RDF triples:

```
(aya, knows, jose)
```

An XML serialization of RDF triples:

```
<person name="aya">
  <knows>
    <person name="jose" />
  </knows>
</person>
```

Figure 7.5: RDF Serialization of the FOAF network in Fig. 7.4.

An approach to visualizing RDF data is a pure rendering of its serialization in XML format using CSS^{NG}. RDF data can be serialized, e.g., as tree as follows: The subject and the object of an RDF triple are nested XML nodes. The relation between subject and object is expressed via another XML node nested between subject and object (see Fig. 7.5). If a subject has more than one object, a new predicate XML element holding the object is added.

Obviously, the serialization of RDF data as demonstrated in Fig. 7.5 is better suited for a direct rendering by comparison to a set of RDF triples like on the left side of Fig. 7.5. The following Fig. 7.6 specifies a part of the FOAF network of Fig. 7.4.

RDF triples:

```
(aya,   knows,jose)
(egon,  knows,aya)
(egon,  knows,franzi)
(franzi,knows,simon)
(franzi,knows,sarah)
(jose,  knows,egon)
```

An XML serialization of RDF triples:

```
<person name="egon">
  <relation type="knows">
    <person name="franzi">
      <relation type="knows">
        <person name="simon" />
      </relation>
      <relation type="knows">
        <person name="sarah" />
      </relation>
    </person>
  </relation>
  <relation type="knows">
    <person name="aya">
      <relation type="knows">
        <person name="jose">
          <relation type="knows">
            <person name="egon" />
          </relation>
        </person>
      </relation>
    </person>
  </relation>
</person>
```

Figure 7.6: Serialization of a FOAF network.

In the RDF data of Fig. 7.6 the `person egon` is duplicated because the expressivity of tree structures (like in a serialization) is not sufficient to specify graphs directly. The duplication allows to connect links to RDF resources that are already used in other parts of the tree.

7.2.2 Rendering of a FOAF Serialization using CSS^{NG}

CSS^{NG} is a styling language that can be used to specify the rendering of semi-structured data. Hence, the result of a styling process depends much on the underlying data. For instance, an RDF graph can have several serializations. Some of them may be better suited by comparison to others. Visualizations such as shown in Fig. 7.4 are beyond the scope of CSS^{NG} because NP-complete transformations are needed for getting such a visualization. The main idea of rendering a FOAF network in a Web browser is to render the serialization of a FOAF specification (see Section 7.2.1) using CSS^{NG} features as follows:

7.2.2.1 Markup Visualization

More than twenty various serializations of RDF data are available[3]. Many of these serializations are using not only X(HT)ML attribute values for encoding RDF data. Hence, the CSS 3 function `attr(X)` for visualizing the values of known X(HT)ML attributes (see Section 2.7.2) is not sufficient for visualizing FOAF networks.

CSS^{NG} offers means for markup visualization (see Section 4.3). The following CSS^{NG} rules specify the visualization of the markup.

```
* {
   text-align:  center;
   display:     block;
   width:       5.2em;
   margin-left: 6.5em;
}

person::before { // markup visualization
   content: element("span",
              attribute("class","person")
              attribute("idref",attr(name)),
              attr(name));
}

knows::before  { // markup visualization
   content: element("span",
              attribute("class","relation"),
              element-name());
}
```

```
.person {   // box for persons
   border:             thin solid black;
   border-radius:      1em;
}

.relation { // relation with arrows
   position:           relative;
   left:               -4em;
   background-image:   url(arrows.png);
   background-repeat:  no-repeat;
   padding-left:       3.8em;
   padding-right:      3em;
   padding-top:        1.2em;
   padding-bottom:     1.5em;
}

.person[idref="egon"]:hover {
   background-color:   green;
}
```

Figure 7.7: Markup Visualization.

In Fig. 7.7 the first CSS^{NG} rule specifies general rendering properties. The actual markup visualization is implemented in the second and the third rule: New markup is inserted before each RDF subject, predicate, and object lifting markup to content. The styling of the inserted markup is specified in the last two rules of Fig. 7.7. A box with round corners marks the name of a `person`. The styling of the `relation` is more complex because of arrows indicating the relationship between subject and object. The visualization of a relation box (see Box-Model [BLLJ98]) is widened with the name of the relation `knows` in the middle of the box. Around

[3]`http://www.fakeroot.net/sw/rdf-formats/`

the name is enough space for the arrows of the relation. These arrows are inserted via a background image. The result of this basic rendering is shown in Fig. 7.8.

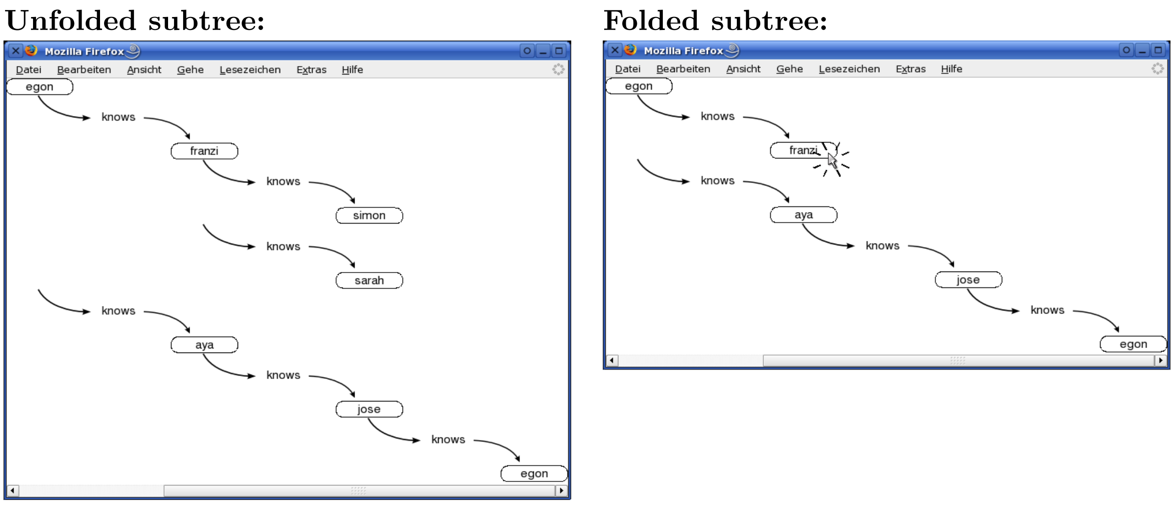


Figure 7.8: FOAF rendering.

7.2.2.2 Folding of parts of a FOAF specification

Since social networks such as specified using the FOAF format tend to become rather complex, a concept for visualizing only parts of the network is useful. The dynamic rendering features of CSS^{NG} can master such requirements using folding and unfolding. The following style sheet allows to hide branches of the FOAF network, which is visualized as tree.

```
person:onclick(2n+1) > * { display: none; }
person:onclick(2n+2) > * { display: block; }
```

Figure 7.9: CSS^{NG} style sheet folding branches of the RDF serialization.

Each click on the visualization of a `person` causes alternating folding and unfolding of the relations of this person. Fig. 7.8 on the right side shows, how the unfolded branch of `franzi` (see left side) is folded after a click. Note that the branch is not just invisible but also the branches below moved up. This behavior saves space and, hence, different parts of the network can be compared easily in a Web browser window.

7.2.2.3 Graph-Visualization

A key issue in visualizing RDF graphs and in particular FOAF networks is handling cycles like `egon knows aya knows jose knows egon`. In CSS^{NG} this problem can be handled similar to the serialization of RDF graphs im XML format: Links are used to specify circles. CSS^{NG} cannot be used to implement hyperlinks allowing to follow links but it allows to visualize all occurrences of the same name. The following style sheet implements this visualization for `egon`.

CSS^{NG} style sheet:

```
egon:hover ? egon
  { background-color: green; }
```

Dynamic Rendering:

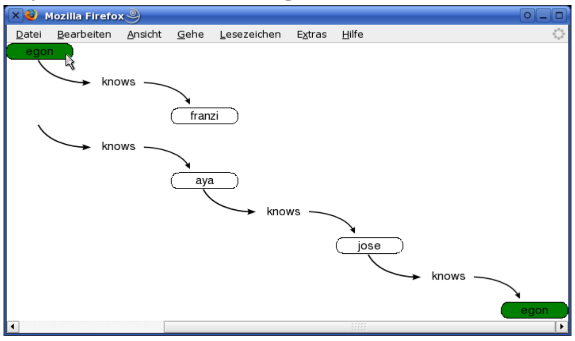


Figure 7.10: FOAF Graph-Visualization.

The question mark (`?`) in Fig. 7.10 is a CSS^{NG} combinator (see Section 4.6), which is used to combine X(HT)ML nodes independently of their position in their documents. In this case each `egon` XML element is selected, if the mouse cursor hovers over one visualization of one `egon` XML element.

7.3 Program Visualization As Textual Program Rendering: An application to Xcerpt Programs

CSS^{NG} can be applied to implement query visualization as textual query rendering as shown in Fig. 7.15. Here, the viewer of the visual interface visXcerpt [Ber03] [BBSW03] for the XML query and transformation language Xcerpt [SB04] is re-implemented by only a few CSS^{NG} rules. It is worth stressing that

- the original implementation of visXcerpt is much longer and much more complex,
- CSS^{NG} is a high level styling language applicable not only to visualize Xcerpt programs but more generally any XML document,
- specifying advanced visual features using CSS^{NG} does not require programming skills as required by ECMA script
- but instead offers much more limited programming capabilities sufficient for styling using CSS.

The following sections discuss the re-implementation of visXcerpt using CSS^{NG}. The full CSS^{NG} style sheet specifying the rendering of Xcerpt programs is given in Appendix B.

7.3.1 Visualization of Xcerpt Data Terms

Data terms are used to represent semi-structured data such as XML documents. They are similar to ground functional programming expressions and logical atoms. In addition data terms offer to specify whether the child nodes of an element are ordered or unordered [SB04].

In particular since CSS^{NG} allows markup visualization, data terms can be rendered as specified in the style sheet in Fig. 7.12. The following paragraphs discuss this style sheet considering the rendering of data terms in Fig. 7.11 as example.

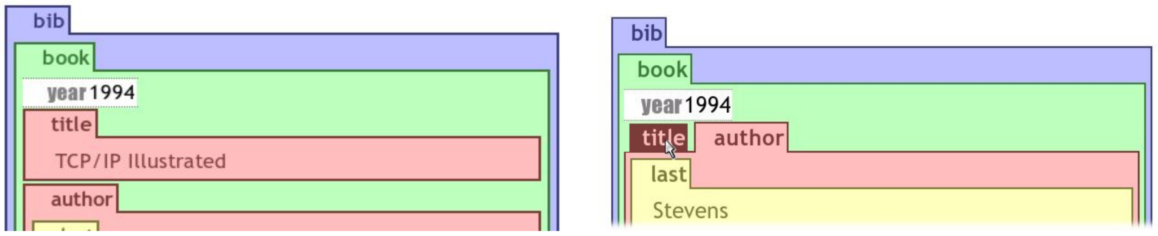


Figure 7.11: Rendering of an XML document using the CSS^{NG} style sheet in Fig. 7.12.

```
/* ### Basic Styling Specifications ### */
* {
           display:        block;
           position:       relative;          /* allow relocating elements */
           z-index:        0;                 /* neutral overlay position */
           border-width:   thin;              /* borders */
           border-style:   solid;
           margin-left:    2em;               /* indentation of elements */
           margin-right:   2em;
           padding-left:   0.5em;             /* space around text nodes */
           padding-right:  0.5em;
           padding-top:    0.5em;
           padding-bottom: 0.5em; }
.unordered { border-style:   dotted; }        /* ordered data terms */
.ordered   { border-style:   solid; }         /* unordered data terms */

/* ### Insertion of Tabs ### */
*::before {
           content: element("tab",            /* insert element "tab" */
                            attribute("order", attr("order")),
                            element-name()    /* value of the tab element */
                              element("attribute", *{ content: " " attribute-name()
                                                    " " attribute-value(); } ) ); }

/* ### Basic Tab Styling ### */
tab {
           top:            0.04em;            /* set tabs *deeper* to */
           z-index:        1;                 /* *overlay* top border of element bodies */
           border-bottom:  none;              /* no border bottom (linking tab and body) */
           padding-top:    0em;               /* no space above tab names and tab border */
           padding-bottom: 0em;               /* no space below tab names and tab border */
           margin-top:     1em;               /* interspace to elements above */
           width:          5em; }             /* tab width */
attribute {                                   /* styling of attribute visualizations */
           border:         thin dotted gray;
           color:          black;
           background-color:white; }

/* ### Folding elements on odd number of clicks ### */
tab:onclick(2n+1) {                           /* ## Folded Tab Styling ## */
           display:        inline;            /* juxtaposing of folded tabs */
           margin-left:    0em;               /* distance between folded tabs */
           left:           0.5em;             /* indent folded tabs a bit */
           z-index:        -1; }              /* disconnect tab with body */
tab:onclick(2n+1) + * {                       /* ## Folded Body Styling ## */
           display:        none; }            /* hide element */


/* ### Unfolding elements on even number of clicks ### */
tab:onclick(2n+2) {                           /* ## Unfolded Tab Styling ## */
           display:        block;             /* juxtaposing of folded tabs */
           margin-left:    2em;               /* standard indentation */
           left:           0em;               /* same indentation of tab and body */
           z-index:        1; }               /* connect tab with body */

tab:onclick(2n+2) + * {                       /* ## Unfolded Body Styling ## */
           display:        block;             /* show element */

/* ### Color Definition of nested Elements ### */
*:nth-decendent(6n+1) { background-color: #bfbfff; color: #3f3f7f; }
*:nth-decendent(6n+2) { background-color: #bfffbf; color: #3f7f3f; }
*:nth-decendent(6n+3) { background-color: #ffbfbf; color: #7f3f3f; }
*:nth-decendent(6n+4) { background-color: #ffffbf; color: #7f7f3f; }
*:nth-decendent(6n+5) { background-color: #ffbfff; color: #7f3f7f; }
*:nth-decendent(6n+6) { background-color: #bfffff; color: #3f7f7f; }
```

Figure 7.12: CSS^{NG} style sheet for rendering data terms.

The **Basic Styling Specifications** in line 1-15 of the CSS^{NG} style sheet in Fig. 7.12 specify styling that applies to all XML elements of the source XML document. Further CSS rules, which are discussed below, will overwrite these basic styling specifications using the CSS cascading feature. The main goal of the rule starting in line 2 is to specify the rendering of the bodies (and not the tabs) of XML elements in visXcerpt. Beside specifications of spacing, the nesting of XML elements is specified in the lines 3, 8, and 9. Two interesting CSS declarations are left in this rule: `position:relative` (line 4) allows to displace the rendering of XML elements relatively to the standard position using the properties `top`, `bottom`, `left`, and `right` later on. The other interesting CSS declaration is given in the next line 5: `z-index:0` sets all renderings on the same level concerning overlay. The larger a `z-index` the higher is the level of a rendering (shading renderings on lower levels). This feature is needed for connecting or disconnecting tabs and bodies of the renderings of XML elements as discussed below.

According to the CSS rules in lines 14 and 15 unordered data terms are rendered using `dotted` borders and ordered data terms are rendered using `solid` lines.

The first CSS^{NG} rule in this use case appears in the block **Insertion of Tabs**. It is a key specification of this use case. The rule beginning in line 18 inserts an XML element `tab` in front of each XML elements. The inserted `tab` element has an attribute `order` (line 20) taking over the same order as the body of the XML element. This attribute of `tab` is needed to render the element and the corresponding tab the same way like using a dotted or a solid line for both. The lines 21-23 specify the content of the inserted `tab` element. First of all it contains the name of the corresponding XML element (markup visualization) followed by a list of the XML element's attributes. This list consists of `attribute` XML elements allowing to apply individual styling for the rendering of XML attributes. Each `attribute` XML element holds the name and the value of its XML attribute. The list of attributes is generated using an attribute rule (line 22). This attribute rule iterates over the attributes of the currently selected XML element (line 18) in XML document order.

The **Basic Tab Styling** specifies the initial static styling of tabs. Once again various spacing specifications are needed. The lines 27-29 explain the connected rendering of tabs and bodies: In line 27 the tab is set down a little bit using the `top` property[4]. The next line sets the tab on a higher overlay level so that the body of the XML element lays below. The illusion of a connection between tab and element is completed in line 29, where the border on the bottom of the tab is removed. Hence, it seems that tab and element are connected because the background color of the tab shades the border of the body (see `author` in Fig. 7.11).

The basic appearance of attributes is specified in the second rule in this block (lines 34-37). According to the example in Fig. 7.11 attributes are rendered in black color, white background color, and a dotted border around their visualizations.

After standard CSS rendering CSS^{NG} is used again in the blocks **(Un)Folding elements on odd(/even) number of clicks**. Since the second block (line 50-59) is analogously specified and simply restores the initial rendering specifications it is not specially discussed.

The first rule of this block (lines 40-44) specify the styling of folded tabs. According to line 41 they are styled inline meaning that folded tabs appear in a row like the `title` tab on

[4]This property can be used only for XML elements with `relative position`.

the right side of Fig. 7.11. Line 43 causes a minor indentation to emphasize that a folded tab does not belong to the body below.

The property `z-index` in line 44 is associated with its counterpart in line 54. If the rendering of an XML element is folded, its tab needs to be disconnected from the body of a following sibling XML element. Therefore the bottom of folded tabs is minimally shaded using the lower z-level `-1` by comparison to the standard level as specified in line 5. The counterpart of this rule in line 54 sets the tab on the higher level `1` to connect tab and unfolded body again.

Having specified the rendering of folded tabs, the folding of the corresponding body of an XML element is handled in line 45 and 46. The CSS^{NG} selector matches, if an odd number of mouse clicks were performed on a tab. In this case, the following XML element is not displayed anymore, while the tab is still visible[5]. Analogously, the body is unfolded caused by the rule starting in line 56 again on an even number of mouse clicks.

The last block of the style sheet in Fig. 7.12 specifies the colors of nested XML elements. Current CSS allows to specify colors only down to a certain depth. This is not sufficient for arbitrary XML documents, which can be highly nested. The CSS^{NG} rules in lines 60 to 65 specify a recurring coloring of nested XML elements on arbitrary depth. The same color appears on every sixth nesting level.

7.3.2 Superimposing of Context Menus

The editor of visXcerpt offers a context menu that can be superimposed on demand. The context menu allows to commit changes on the XML document. However, since CSS^{NG} does not offer means for changing input X(HT)ML document, the *functionality* of the menu needs to be implemented using another technology. The following CSS^{NG} style sheet allows to superimpose such a context menu (see Fig. 7.13).

```
1  *:root::before {
2     content: element("menu", ...); }
3  tab:hover ? menu {
4     postion: fixed;
5     top:     5em;
6     left:    5em;
7     width:   10em;
8     z-index: 2; }
```

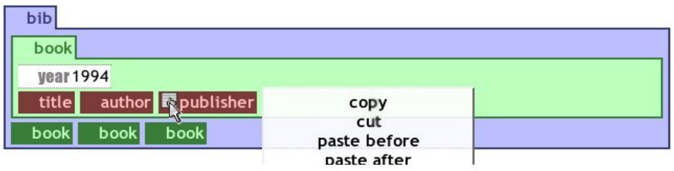


Figure 7.13: CSS^{NG} rule superimposing the visXcerpt context menu on the right side.

The first rule in the style sheet of Fig. 7.13 demonstrates how the context menu of visXcerpt can be “linked” to visXcerpt using CSS^{NG}. According to the selector in line 1, the menu is (virtually) inserted before the root XML element using the CSS 3 pseudo-class `:root`. The following line specifies the insertion of the context menu. (The three dots stand for the visXcerpt menu items like `copy` or `cut`.)

The dynamic aspect of the visXcerpt context menu is specified in the second rule of the CSS^{NG} style sheet of Fig. 7.13 as follows: if the mouse cursor is hovering over a `tab` (line 3),

[5]Otherwise unfolding the body again would be impossible.

the `menu` is superimposed according to the CSS declaration in lines 5-9. The context menu is `position`ed relatively to the Web browser window on the upper left. Finally, according to line 8 the context menu hovers over all other rendered XML elements.

7.3.3 Visualization of Xcerpt Query Programs

Xcerpt data terms, which are introduced in the latter section, are the foundation of Xcerpt *query terms* and *construction terms*. Further rendering specifications are needed for additional constructs like Xcerpt *variables* and *grouping constructs*.

The static rendering of those constructs is already specified in the diploma thesis of Sacha Berger [Ber03] and can be re-used in this use case with only a few additional rules given in Fig. 7.14:

```
1  all, and, or {
2     padding-left: 3em;
3     background-repeat:  no-repeat; }
4  all {
5     background-image:   url(all.png); }
6  and {
7     background-image:   url(and.png); }
8  or {
9     background-image:   url(or.png); }
10 head, query {
11    width:              40%; }
12 query {
13    position:           absolute;
14    right:              0em;
15    top:                1em; }
16 rule, goal {
17   background-image:    url(left-arrow.png);
18   background-position: center;
19   background-repeat:   no-repeat; }
```

Figure 7.14: Rendering of Xcerpt query programs.

The style sheet in Fig. 7.14 renders the Xcerpt constructs `and`, `or` and `all` using background images. To avoid a rendering of Xcerpt rules using blind HTML tables[6] (like in the visXcerpt prototype [Ber03]) the CSS `position` declaration is used to visualize the query part on the righthand side of the visualization of an Xcerpt rule (line 14) while the construction part is implicitly rendered on the lefthand side with a reduced width (line 11). The result of a rendering using these rules is demonstrated in Fig. 7.15.

7.3.4 Highlighting Xcerpt Variables

In the following example the identifiers `Title` and `Author` are used for Xcerpt variables. It is possible to understand the example without knowing what Xcerpt variables are used for.

[6]HTML tables without border for arranging the layout of Web pages.

In visXcerpt all occurrences of a variable can be highlighted. The following Fig. 7.15 shows, how a variable can be highlighted using a CSS^{NG} style sheet.

```
Author:hover ? Author,
Title:hover  ? Title  {
   background-color: white;
}
```

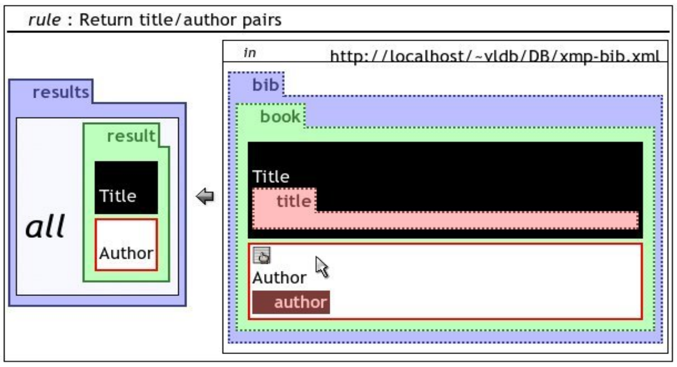

rule : Return title/author pairs
in
http://localhost/~vldb/DB/xmp-bib.xml
results
result
all
Title
Author
bib
book
Title
title
Author
author

Figure 7.15: Query Visualization as Textual Query Rendering. All occurrences of `Author` are highlighted by white background color.

The CSS^{NG} style sheet on the left side of Fig. 7.15 groups selectors for every visXcerpt variable, saying that all occurrences of a variable are highlighted, if the mouse cursor is hovering over one occurrence of a variable.

CHAPTER
EIGHT

Conclusion

8.1 Summary

The research topic investigated in this thesis is to extend the Cascading Style Sheets language (CSS) toward dynamic document rendering features. First of all, this thesis gives an introduction to CSS including its fundamental principles. Then it discusses the shortcomings and limitations of CSS concerning dynamic document rendering and markup visualization. This discussion leads to CSS^{NG}, an extension to CSS 3, trying to overcome these shortcomings and limitations. Furthermore CSS^{NG} is compared to related approaches that extend CSS with dynamic document rendering features. Finally, this thesis provides a prototypical implementation of CSS^{NG}, which is used for testing proof-of-concept applications to demonstrate the benefits of CSS^{NG}.

A first extension of CSS^{NG} is *markup insertion*. In a CSS^{NG} style sheet well-formed markup (or in other words XML data) can be specified using CSS^{NG} functions. Such markup can be inserted before and/or after arbitrary XML (and XHTML) elements for visualization purposes (while the input document remains the same). The advantage by comparison to CSS 3 is that such insertions can be styled using CSS rules offering a broad field for applications. In CSS 3 a differentiated styling of inserted text is not possible.

A second extension of CSS^{NG} is *markup visualization*. This extension is a generalization of *markup insertion* because it allows to insert markup depending on the input XML document and not on "constant" markup that is specified in the style sheet. This feature is achieved using CSS^{NG} functions that allow to query the markup of the input document. In other words *markup querying* in combination with *markup insertion* allows to visualize markup.

A third extension of CSS^{NG} is *depth-dependant styling*. This is a minor extension in the framework of CSS^{NG}. It does not only allow breadth-dependant styling like CSS 3, as used for an alternating coloring of table rows, but also depth-dependant styling as needed for the styling of highly nested structures like threads in discussion forums.

A forth and highly important extension of CSS^{NG} is the *generalization of dynamic styling*. This extension adopts the intrinsic events of HTML for achieving dynamic styling in CSS. These intrinsic events allow a better differentiation of events, for instance `:hover` in CSS 3 can be expressed using `:onmouseover` and `:onmouseout` in CSS^{NG}. Furthermore the adoption of HTML intrinsic events make the keyboard a new input device in cascading styling. So-called recurrence patterns allow to specify cyclic events (as can be used for alternating folding and unfolding) or acyclic events (as can be used for indicating a specific number of click on a hyperlink). A noticeable feature of this adaption is its compatibility to CSS combinators (in contrast to other approaches to CSS).

A fifth extension of CSS^{NG} is the *Structure-independent Selection.* This extension allows to select XML (or XHTML) elements, if a simple selector matches an element that is part of the input XML (or XHTML) document. There are no constraints on structural relations between elements for those selections. For instance, the background of a document is styled in red color, if at least one paragraph in the document is marked as "unfinished". This extension is compatible with extensions concerning *dynamic styling* because it is realized by introducing a CSS^{NG} combinator.

8.2 Contributions

The above mentioned extensions of CSS^{NG} are on the one hand designed as natural extensions to CSS 3 and on the other hand as rather slight and conservative extensions to CSS 3. This allows an easy adoption of CSS^{NG} for authors of CSS 3 style sheets. Furthermore CSS^{NG} preserves the purpose of CSS being a styling language and not a transformation language like XSLT or XQuery. From the author's point of view, concerning the extensions of CSS^{NG} the simplicity of the CSS language is preserved.

In contrast to extensions related to CSS^{NG} (like *Action Sheets* and in the *Behavioral Extension to CSS* using scripting) CSS^{NG} is rather constrained, but as shown by several use cases CSS^{NG} makes scripting needless for many applications. Furthermore CSS^{NG}

- causes no schema changes of the document to be styled,
- can be provided as standalone definition independently of the document to be styled,
- is applicable to multiple documents, and
- can be re-used easily.

One of the main advantages of CSS^{NG} is the declarativity of its rules. Implementing styles with comparatively complex programming languages like ECMA Script is no longer needed for many applications.

A comparison of the implementation of visXcerpt particularly using DHTML [Ber03] (2060 lines of code) and the implementation given in Appendix B makes clear that CSS^{NG} allows a much more concise implementation of visXcerpt with a range of 131 lines of code. Obviously, if using CSS^{NG} instead of a scripting approach, no additional language needs to be learned to achieve dynamic styling.

The prototypical implementation of CSS^{NG} fulfills the requirements laid down in Section 6.1 of a proof-of-concept implementation. Due to the various interchange formats of the prototype's processors (like the Styler Configurator) and the used standards for the implementation, debugging and changing syntax and semantic of CSS^{NG} was rather easy. An advantage of using those standards is that the prototype can be used to demonstrate CSS^{NG}

on many platforms. A drawback of the prototypical implementation is the run-time performance by comparison to the efficient rendering engines of modern Web browsers. The author is convinced that a specialization of the prototype for a single Web browser like using the Web browser's XSLT processor would yield a significantly better run-time performance of the prototype.

8.3 Further Research Directions

The author believes that CSS^{NG} allows generic visualizations of programming languages. Such visualizations realized as textual document rendering (see Fig. 7.15) could help making visual programming more widespread than today because the huge quantity of tools for textual programming languages can still be used. To the best of the author's knowledge further approaches of visual languages never allow visual and textual programming as well.

A second research issue is to investigate how CSS^{NG} might fit in *editor applications.* This thesis investigated *viewer applications* exclusively, such as the visualization of FOAF definitions or the implementation of the visXcerpt interface to Xcerpt. However, these applications of CSS^{NG} would most likely profit, if their data could be edited in a WYSIWYG[1] mode.

[1]What You See Is What You Get

APPENDIX

A

Code of the CSS^{NG} engine

A.1 CSS^{NG} Lexer

```
%%

/*
 * Due to differences of the syntax between flex and jflex,
 * I modified this Tokenizer (given in
 * http://www.w3.org/TR/CSS21/grammar.html#q2 in flex syntax)
 * as follows for gaining jflex compatibility:
 *
 * flex                      vs. jflex
 * -----------------------------------------------
 * %option case-insensitive      %ignorecase
 * rules without "="             rules with "="
 * \\"                           \"
 *
 * Running jflex:
 * java JFlex.Main <options> <inputfiles>
 * jaflex <options> <inputfiles>
 */

%byaccj
%ignorecase

%{
  /* store a reference to the parser object */
  private Parser yyparser;

  /* constructor taking an additional parser object */
  public Yylex(java.io.Reader r, Parser yyparser) {
    this(r);
    this.yyparser = yyparser;
  }
%}


h                = [0-9a-f]
nonascii         = [\200-\377]
unicode          = \\{h}{1,6}(\r\n|[ \t\r\n\f])?
escape           = {unicode}|\\[^\r\n\f0-9a-f]
nmstart          = [_a-z]|{nonascii}|{escape}
nmchar           = [_a-z0-9-]|{nonascii}|{escape}
string1          = \"([^\n\r\f\"]|\\{nl}|{escape})*\"
string2          = \'([^\n\r\f\\']|\\{nl}|{escape})*\'
invalid1         = \"([^\n\r\f\"]|\\{nl}|{escape})*
invalid2         = \'([^\n\r\f\\']|\\{nl}|{escape})*

ident            = -?{nmstart}{nmchar}*
name             = {nmchar}+
num              = [0-9]+|[0-9]*"."[0-9]+
string           = {string1}|{string2}
invalid          = {invalid1}|{invalid2}
url              = ([!#$%&*-~]|{nonascii}|{escape})*
s                = [ \t\r\n\f]
w                = {s}*
nl               = \n|\r\n|\r|\f

A                = a|\\0{0,4}(41|61)(\r\n|[ \t\r\n\f])?
C                = c|\\0{0,4}(43|63)(\r\n|[ \t\r\n\f])?
D                = d|\\0{0,4}(44|64)(\r\n|[ \t\r\n\f])?
E                = e|\\0{0,4}(45|65)(\r\n|[ \t\r\n\f])?
G                = g|\\0{0,4}(47|67)(\r\n|[ \t\r\n\f])?|\\g
H                = h|\\0{0,4}(48|68)(\r\n|[ \t\r\n\f])?|\\h
I                = i|\\0{0,4}(49|69)(\r\n|[ \t\r\n\f])?|\\i
K                = k|\\0{0,4}(4b|6b)(\r\n|[ \t\r\n\f])?|\\k
M                = m|\\0{0,4}(4d|6d)(\r\n|[ \t\r\n\f])?|\\m
N                = n|\\0{0,4}(4e|6e)(\r\n|[ \t\r\n\f])?|\\n
P                = p|\\0{0,4}(50|70)(\r\n|[ \t\r\n\f])?|\\p
R                = r|\\0{0,4}(52|72)(\r\n|[ \t\r\n\f])?|\\r
S                = s|\\0{0,4}(53|73)(\r\n|[ \t\r\n\f])?|\\s
T                = t|\\0{0,4}(54|74)(\r\n|[ \t\r\n\f])?|\\t
```

```
X               = x|\\0{0,4}(58|78)(\r\n|[ \t\r\n\f])?|\\x
Z               = z|\\0{0,4}(5a|7a)(\r\n|[ \t\r\n\f])?|\\z

%%

/* extensions */
"element("              {yyparser.yylval = new ParserVal(yytext()); return Parser.ELEMENT_FN;}
"element-name("         {yyparser.yylval = new ParserVal(yytext()); return Parser.ELEMENT_NAME;}
"attribute("            {yyparser.yylval = new ParserVal(yytext()); return Parser.ATTRIBUTE_FN;}
"content"               {yyparser.yylval = new ParserVal(yytext()); return Parser.CONTENT;}
{w}"?"                  {yyparser.yylval = new ParserVal(yytext()); return Parser.QUESTIONMARK;}
/* ---------- */


{s}+                    {yyparser.yylval = new ParserVal(yytext()); return Parser.S;}

\/\*[^*]*\*+([^/*][^*]*\*+)*\/          {/* ignore comments */}
{s}+\/\*[^*]*\*+([^/*][^*]*\*+)*\/      {/*unput(' ');*/ /*replace by space*/}

"<!--"                  {yyparser.yylval = new ParserVal(yytext()); return Parser.CDO;}
"-->"                   {yyparser.yylval = new ParserVal(yytext()); return Parser.CDC;}
"~="                    {yyparser.yylval = new ParserVal(yytext()); return Parser.INCLUDES;}
"|="                    {yyparser.yylval = new ParserVal(yytext()); return Parser.DASHMATCH;}

{w}"{"                  {yyparser.yylval = new ParserVal(yytext()); return Parser.LBRACE;}
{w}"+"                  {yyparser.yylval = new ParserVal(yytext()); return Parser.PLUS;}
{w}">"                  {yyparser.yylval = new ParserVal(yytext()); return Parser.GREATER;}
{w}","                  {yyparser.yylval = new ParserVal(yytext()); return Parser.COMMA;}

{string}                {yyparser.yylval = new ParserVal(yytext()); return Parser.STRING;}
{invalid}               {yyparser.yylval = new ParserVal(yytext()); return Parser.INVALID;}

{ident}                 {yyparser.yylval = new ParserVal(yytext()); return Parser.IDENT;}

"#"{name}               {yyparser.yylval = new ParserVal(yytext()); return Parser.HASH;}

"@import"               {yyparser.yylval = new ParserVal(yytext()); return Parser.IMPORT_SYM;}
"@page"                 {yyparser.yylval = new ParserVal(yytext()); return Parser.PAGE_SYM;}
"@media"                {yyparser.yylval = new ParserVal(yytext()); return Parser.MEDIA_SYM;}
"@charset"              {yyparser.yylval = new ParserVal(yytext()); return Parser.CHARSET_SYM;}

"!"{w}"important"       {yyparser.yylval = new ParserVal(yytext()); return Parser.IMPORTANT_SYM;}

{num}{E}{M}             {yyparser.yylval = new ParserVal(yytext()); return Parser.EMS;}
{num}{E}{X}             {yyparser.yylval = new ParserVal(yytext()); return Parser.EXS;}
{num}{P}{X}             {yyparser.yylval = new ParserVal(yytext()); return Parser.LENGTH;}
{num}{C}{M}             {yyparser.yylval = new ParserVal(yytext()); return Parser.LENGTH;}
{num}{M}{M}             {yyparser.yylval = new ParserVal(yytext()); return Parser.LENGTH;}
{num}{I}{N}             {yyparser.yylval = new ParserVal(yytext()); return Parser.LENGTH;}
{num}{P}{T}             {yyparser.yylval = new ParserVal(yytext()); return Parser.LENGTH;}
{num}{P}{C}             {yyparser.yylval = new ParserVal(yytext()); return Parser.LENGTH;}
{num}{D}{E}{G}          {yyparser.yylval = new ParserVal(yytext()); return Parser.ANGLE;}
{num}{R}{A}{D}          {yyparser.yylval = new ParserVal(yytext()); return Parser.ANGLE;}
{num}{G}{R}{A}{D}       {yyparser.yylval = new ParserVal(yytext()); return Parser.ANGLE;}
{num}{M}{S}             {yyparser.yylval = new ParserVal(yytext()); return Parser.TIME;}
{num}{S}                {yyparser.yylval = new ParserVal(yytext()); return Parser.TIME;}
{num}{H}{Z}             {yyparser.yylval = new ParserVal(yytext()); return Parser.FREQ;}
{num}{K}{H}{Z}          {yyparser.yylval = new ParserVal(yytext()); return Parser.FREQ;}
{num}{ident}            {yyparser.yylval = new ParserVal(yytext()); return Parser.DIMENSION;}

{num}%                  {yyparser.yylval = new ParserVal(yytext()); return Parser.PERCENTAGE;}
{num}                   {yyparser.yylval = new ParserVal(yytext()); return Parser.NUMBER;}

"url("{w}{string}{w}")" {yyparser.yylval = new ParserVal(yytext()); return Parser.URI;}
"url("{w}{url}{w}")"    {yyparser.yylval = new ParserVal(yytext()); return Parser.URI;}
{ident}"("              {yyparser.yylval = new ParserVal(yytext()); return Parser.FUNCTION;}

//.                     {yyparser.yylval = new ParserVal(yytext()); return *yytext;}

/* extra tokens for byacc/j compatibility */
":"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.COLON;}
";"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.SEMICOLON;}
"}"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.RBRACE;}
"/"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.SLASH;}
"-"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.MINUS;}
"."                     {yyparser.yylval = new ParserVal(yytext()); return Parser.DOT;}
"*"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.STAR;}
"["                     {yyparser.yylval = new ParserVal(yytext()); return Parser.LBRACKET;}
"]"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.RBRACKET;}
")"                     {yyparser.yylval = new ParserVal(yytext()); return Parser.RPAREN;}
"="                     {yyparser.yylval = new ParserVal(yytext()); return Parser.EQUALS;}

```

A.2 CSS^{NG} Parser

```
/* Used right recursion for a friendly stack */

%{
  import java.io.*;
%}

%token S
%token CDO
%token CDC
%token INCLUDES
%token DASHMATCH
%token LBRACE
%token PLUS
%token GREATER
%token COMMA
%token STRING
%token INVALID
%token IDENT
%token HASH
%token IMPORT_SYM
%token PAGE_SYM
%token MEDIA_SYM
%token CHARSET_SYM
%token IMPORTANT_SYM
%token EMS
%token EXS
%token LENGTH
%token ANGLE
%token TIME
%token FREQ
%token DIMENSION  /* Note: This token is used in the lexer but not in the parser */
%token PERCENTAGE
%token NUMBER
%token URI
%token FUNCTION

/* extra tokens for byacc/j compatibility */
%token COLON
%token SEMICOLON
%token RBRACE
%token SLASH
%token MINUS
%token DOT
%token STAR
```

```
%token LBRACKET
%token RBRACKET
%token RPAREN /* Note: There is no token LPAREN, since left parentheses are included in the token FUNCTION. */
%token EQUALS

/* new tokens */
%token ELEMENT_FN
%token ELEMENT_NAME
%token ATTRIBUTE_FN
%token CONTENT
%token LPAREN
%token QUESTIONMARK

%%

stylesheet
  : stylesheet_ext1opt s_cdo_cdc_star stylesheet_ext2star stylesheet_ext3star      {      String[][] ruleBody = {
                                                                                          {$1.sval, "stylesheet_ext1opt", "false"},
                                                                                          {$2.sval, "s_cdo_cdc_star", "false"},
                                                                                          {$3.sval, "stylesheet_ext2star", "false"},
                                                                                          {$4.sval, "stylesheet_ext3star", "false"} };

                                                                                          result = new StringBuffer();
                                                                                          result.append("<?xml version=\"1.0\" ");
                                                                                          result.append("encoding=\"ISO-8859-1\" ?>");
                                                                                          result.append("<stylesheet>");
                                                                                          result.append(encode(ruleBody));
                                                                                          result.append("</stylesheet>");
                                                                                          /* System.out.println(result.toString()); */
                                                                                   }
  ;
  stylesheet_ext1opt
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | CHARSET_SYM s_star STRING s_star SEMICOLON                                   {      String[][] ruleBody = {
                                                                                          {$1.sval, "CHARSET_SYM", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "STRING", "true"},
                                                                                          {$4.sval, "s_star", "false"},
                                                                                          {$5.sval, "SEMICOLON", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
  stylesheet_ext2star
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | /* recursion */ stylesheet_ext2star import s_cdo_cdc_star                    {      String[][] ruleBody = {
                                                                                          {$1.sval, "stylesheet_ext2star", "false"},
                                                                                        {$2.sval, "import", "false"},
                                                                                        {$3.sval, "s_cdo_cdc_star", "false"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
  ;
stylesheet_ext3star
  : /* empty */                                                                  { $$ = new ParserVal(); }
  | /* recursion */ stylesheet_ext3star
    stylesheet_ext3star_ext1paren s_cdo_cdc_star                                 {      String[][] ruleBody = {
                                                                                        {$1.sval, "stylesheet_ext3star", "false"},
                                                                                        {$2.sval, "stylesheet_ext3star_ext1paren", "false"},
                                                                                        {$3.sval, "s_cdo_cdc_star", "false"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
  ;
  stylesheet_ext3star_ext1paren
    : ruleset                                                                    {      String[][] ruleBody = {
                                                                                        {$1.sval, "ruleset", "false"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
    | media                                                                      {      String[][] ruleBody = {
                                                                                        {$1.sval, "media", "false"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
    | page                                                                       {      String[][] ruleBody = {
                                                                                        {$1.sval, "page", "false"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
    ;
s_cdo_cdc_star
  : /* empty */                                                                  { $$ = new ParserVal(); }
  | /* recursion */ s_cdo_cdc_star s_cdo_cdc_star_ext1paren                      {      String[][] ruleBody = {
                                                                                        {$1.sval, "s_cdo_cdc_star", "false"},
                                                                                        {$2.sval, "s_cdo_cdc_star_ext1paren", "false"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
  ;
  s_cdo_cdc_star_ext1paren
    : S                                                                          {      String[][] ruleBody = {
                                                                                        {$1.sval, "S", "true"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
    | CDO                                                                        {      String[][] ruleBody = {
                                                                                        {$1.sval, "CDO", "true"} };
                                                                                   $$ = new ParserVal(encode(ruleBody));
                                                                                 }
```

```
      | CDC                                                                  {      String[][] ruleBody = {
                                                                                    {$1.sval, "CDC", "true"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
                                                                             }

      ;
import
  : IMPORT_SYM s_star                                                        {      String[][] ruleBody = {
                                                                                    {$1.sval, "IMPORT_SYM", "true"},
                                                                                    {$2.sval, "s_star", "false"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
                                                                             }
    import_ext1paren s_star import_ext2opt SEMICOLON s_star                  {      String[][] ruleBody = {
                                                                                    {$1.sval, "import_ext1paren", "false"},
                                                                                    {$2.sval, "s_star", "false"},
                                                                                    {$3.sval, "import_ext2opt", "false"},
                                                                                    {$4.sval, "SEMICOLON", "true"},
                                                                                    {$5.sval, "s_star", "false"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
                                                                             }
  ;
  import_ext1paren
  : STRING                                                                   {      String[][] ruleBody = {
                                                                                    {$1.sval, "STRING", "true"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
                                                                             }
  | URI                                                                      {      String[][] ruleBody = {
                                                                                    {$1.sval, "URI", "true"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
                                                                             }
  ;
  import_ext2opt
    : /* empty */                                                            { $$ = new ParserVal(); }
    | medium import_ext2opt_ext1star                                         {      String[][] ruleBody = {
                                                                                    {$1.sval, "medium", "false"},
                                                                                    {$2.sval, "import_ext2opt_ext1star", "false"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
                                                                             }
    ;
    import_ext2opt_ext1star
      : /* empty */                                                          { $$ = new ParserVal(); }
      | /* recursion */ import_ext2opt_ext1star COMMA s_star medium          {      String[][] ruleBody = {
                                                                                    {$1.sval, "import_ext2opt_ext1star", "false"},
                                                                                    {$2.sval, "COMMA", "true"},
                                                                                    {$3.sval, "s_star", "false"},
                                                                                    {$4.sval, "medium", "false"} };
                                                                               $$ = new ParserVal(encode(ruleBody));
```

```
                                                                                   }
      ;
media
  : MEDIA_SYM s_star medium media_ext1star LBRACE
    s_star ruleset_star RBRACE s_star      {      String[][] ruleBody = {
                                                                                          {$1.sval, "MEDIA_SYM", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "medium", "false"},
                                                                                          {$4.sval, "media_ext1star", "false"},
                                                                                          {$5.sval, "LBRACE", "true"},
                                                                                          {$6.sval, "s_star", "false"},
                                                                                          {$7.sval, "ruleset_star", "false"},
                                                                                          {$8.sval, "RBRACE", "true"},
                                                                                          {$9.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
  media_ext1star
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | /* recursion */ media_ext1star COMMA s_star medium                           {      String[][] ruleBody = {
                                                                                          {$1.sval, "media_ext1star", "false"},
                                                                                          {$2.sval, "COMMA", "true"},
                                                                                          {$3.sval, "s_star", "false"},
                                                                                          {$4.sval, "medium", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
medium
  : IDENT s_star                                                                   {      String[][] ruleBody = {
                                                                                          {$1.sval, "IDENT", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
page
  : PAGE_SYM s_star pseudo_page_opt s_star LBRACE
    s_star declaration page_ext1star RBRACE s_star                                 {      String[][] ruleBody = {
                                                                                          {$1.sval, "PAGE_SYM", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "pseudo_page_opt", "false"},
                                                                                          {$4.sval, "s_star", "false"},
                                                                                          {$5.sval, "LBRACE", "true"},
                                                                                          {$6.sval, "s_star", "false"},
                                                                                          {$7.sval, "declaration", "false"},
                                                                                          {$8.sval, "page_ext1star", "false"},
                                                                                          {$9.sval, "RBRACE", "true"},
                                                                                          {$10.sval, "s_star",} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
  page_ext1star
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | /* recursion */ page_ext1star SEMICOLON s_star declaration                   {      String[][] ruleBody = {
                                                                                          {$1.sval, "page_ext1star", "false"},
                                                                                          {$2.sval, "SEMICOLON", "true"},
                                                                                          {$3.sval, "s_star", "false"},
                                                                                          {$4.sval, "declaration", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
pseudo_page
  : COLON IDENT                                                                    {      String[][] ruleBody = {
                                                                                          {$1.sval, "COLON", "true"},
                                                                                          {$2.sval, "IDENT", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
operator
  : SLASH s_star                                                                   {      String[][] ruleBody = {
                                                                                          {$1.sval, "SLASH", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | COMMA s_star                                                                   {      String[][] ruleBody = {
                                                                                          {$1.sval, "COMMA", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | /* empty */                                                                    { $$ = new ParserVal(); }
  ;
combinator
  : PLUS s_star                                                                    {      String[][] ruleBody = {
                                                                                          {$1.sval, "PLUS", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | GREATER s_star                                                                 {      String[][] ruleBody = {
                                                                                          {$1.sval, "GREATER", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | s_plus                                                                         {      String[][] ruleBody = {
```

```
                                                                                          {$1.sval, "s_plus", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | QUESTIONMARK s_star /* CW: Extension */                                        {      String[][] ruleBody = {
                                                                                          {$1.sval, "QUESTIONMARK", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
unary_operator
  : MINUS                                                                          {      String[][] ruleBody = {
                                                                                          {$1.sval, "MINUS", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | PLUS                                                                           {      String[][] ruleBody = {
                                                                                          {$1.sval, "PLUS", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
property
  : IDENT s_star                                                                   {      String[][] ruleBody = {
                                                                                          {$1.sval, "IDENT", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
ruleset
  : selector ruleset_ext1star LBRACE s_star
    declaration ruleset_ext2star RBRACE s_star                                     {      String[][] ruleBody = {
                                                                                          {$1.sval, "selector", "false"},
                                                                                          {$2.sval, "ruleset_ext1star", "false"},
                                                                                          {$3.sval, "LBRACE", "true"},
                                                                                          {$4.sval, "s_star", "false"},
                                                                                          {$5.sval, "declaration", "false"},
                                                                                          {$6.sval, "ruleset_ext2star", "false"},
                                                                                          {$7.sval, "RBRACE", "true"},
                                                                                          {$8.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
  ruleset_ext1star
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | /* recursion */ ruleset_ext1star COMMA s_star selector                       {      String[][] ruleBody = {
                                                                                          {$1.sval, "ruleset_ext1star", "false"},
                                                                                          {$2.sval, "COMMA", "true"},
                                                                                          {$3.sval, "s_star", "false"},
                                                                                          {$4.sval, "selector", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
  ruleset_ext2star
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | /* recursion */ ruleset_ext2star SEMICOLON s_star declaration                {      String[][] ruleBody = {
                                                                                          {$1.sval, "ruleset_ext2star", "false"},
                                                                                          {$2.sval, "SEMICOLON", "true"},
                                                                                          {$3.sval, "s_star", "false"},
                                                                                          {$4.sval, "declaration", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
selector
  : simple_selector selector_ext1star                                              {      String[][] ruleBody = {
                                                                                          {$1.sval, "simple_selector", "false"},
                                                                                          {$2.sval, "selector_ext1star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
  selector_ext1star
    : /* empty */                                                                  { $$ = new ParserVal(); }
    | /* recursion */ selector_ext1star combinator simple_selector                 {      String[][] ruleBody = {
                                                                                          {$1.sval, "selector_ext1star", "false"},
                                                                                          {$2.sval, "combinator", "false"},
                                                                                          {$3.sval, "simple_selector", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
simple_selector
  : element_name hash_class_attrib_pseudo_star                                     {      String[][] ruleBody = {
                                                                                          {$1.sval, "element_name", "false"},
                                                                                          {$2.sval, "hash_class_attrib_pseudo_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
;
                                                                                   }
  | hash_class_attrib_pseudo_plus                                                  {      String[][] ruleBody = {
                                                                                          {$1.sval, "hash_class_attrib_pseudo_plus", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
  hash_class_attrib_pseudo_star
  : /* empty */                                                                    { $$ = new ParserVal(""); }
    | /* recursion */ hash_class_attrib_pseudo_star hash_class_attrib_pseudo       {      String[][] ruleBody = {
                                                                                          {$1.sval, "hash_class_attrib_pseudo_star", "false"},
                                                                                          {$2.sval, "hash_class_attrib_pseudo", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
  hash_class_attrib_pseudo_plus
    : hash_class_attrib_pseudo                                                     {      String[][] ruleBody = {
                                                                                          {$1.sval, "hash_class_attrib_pseudo", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    | /* recursion */ hash_class_attrib_pseudo_plus hash_class_attrib_pseudo       {      String[][] ruleBody = {
                                                                                          {$1.sval, "hash_class_attrib_pseudo_plus", "false"},
                                                                                          {$2.sval, "hash_class_attrib_pseudo", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
    hash_class_attrib_pseudo
      : HASH                                                                       {      String[][] ruleBody = {
                                                                                          {$1.sval, "HASH", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
      | class                                                                      {      String[][] ruleBody = {
                                                                                          {$1.sval, "class", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
      | attrib                                                                     {      String[][] ruleBody = {
                                                                                          {$1.sval, "attrib", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
      | pseudo                                                                     {      String[][] ruleBody = {
                                                                                          {$1.sval, "pseudo", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
      ;
class
  : DOT IDENT                                                                      {      String[][] ruleBody = {
                                                                                          {$1.sval, "DOT", "true"},
                                                                                          {$2.sval, "IDENT", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
element_name
  : IDENT                                                                          {      String[][] ruleBody = {
                                                                                          {$1.sval, "IDENT", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | STAR                                                                           {      String[][] ruleBody = {
```

```
                                                                              {$1.sval, "STAR", "true"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
  ;
attrib
  : LBRACKET s_star IDENT s_star attrib_ext1opt RBRACKET               {      String[][] ruleBody = {
                                                                              {$1.sval, "LBRACKET", "true"},
                                                                              {$2.sval, "s_star", "false"},
                                                                              {$3.sval, "IDENT", "true"},
                                                                              {$4.sval, "s_star", "false"},
                                                                              {$5.sval, "attrib_ext1opt", "false"},
                                                                              {$6.sval, "RBRACKET", "false"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
  ;
  attrib_ext1opt
    : /* empty */                                                      { $$ = new ParserVal(); }
    | attrib_ext1opt_ext1paren s_star attrib_ext1opt_ext2paren s_star  {      String[][] ruleBody = {
                                                                              {$1.sval, "attrib_ext1opt_ext1paren", "false"},
                                                                              {$2.sval, "s_star", "false"},
                                                                              {$3.sval, "attrib_ext1opt_ext2paren", "false"},
                                                                              {$4.sval, "s_star", "false"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
    ;
    attrib_ext1opt_ext1paren
      : EQUALS                                                         {      String[][] ruleBody = {
                                                                              {$1.sval, "EQUALS", "true"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
      | INCLUDES                                                       {      String[][] ruleBody = {
                                                                              {$1.sval, "INCLUDES", "true"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
      | DASHMATCH                                                      {      String[][] ruleBody = {
                                                                              {$1.sval, "DASHMATCH", "true"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
      ;
    attrib_ext1opt_ext2paren
      : IDENT                                                          {      String[][] ruleBody = {
                                                                              {$1.sval, "IDENT", "true"} };
                                                                         $$ = new ParserVal(encode(ruleBody));
                                                                       }
      | STRING                                                         {      String[][] ruleBody = {
                                                                              {$1.sval, "STRING", "true"} };
```

```
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
      ;
pseudo
  : COLON pseudo_ext1paren                                                         {      String[][] ruleBody = {
                                                                                          {$1.sval, "COLON", "true"},
                                                                                          {$2.sval, "pseudo_ext1paren", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | COLON COLON pseudo_ext1paren /* CW: added COLON */                             {      String[][] ruleBody = {
                                                                                          {$1.sval, "COLON", "true"},
                                                                                          {$2.sval, "COLON", "true"},
                                                                                          {$3.sval, "pseudo_ext1paren", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
  pseudo_ext1paren
    : IDENT                                                                        {      String[][] ruleBody = {
                                                                                          {$1.sval, "IDENT", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    | FUNCTION s_star params_opt
      /* CW: instead of: ident_opt for numbers as args */ s_star RPAREN            {      String[][] ruleBody = {
                                                                                          {$1.sval, "FUNCTION", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "params_opt", "false"},
                                                                                          {$4.sval, "s_star", "false"},
                                                                                          {$5.sval, "RPAREN", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
declaration
  : CONTENT s_star COLON s_star content_value /* CW: extension */                  {      String[][] ruleBody = {
                                                                                          {$1.sval, "CONTENT", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "COLON", "true"} ,
                                                                                          {$4.sval, "s_star", "false"},
                                                                                          {$5.sval, "content_value", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));

                                                                                   }
  | property COLON s_star expr prio_opt                                            {      String[][] ruleBody = {
                                                                                          {$1.sval, "property", "false"},
                                                                                          {$2.sval, "COLON", "true"},
                                                                                          {$3.sval, "s_star", "false"},
                                                                                          {$4.sval, "expr", "false"},
                                                                                          {$5.sval, "prio_opt", "false"} };
```

```
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
  | /* empty */                                                 { $$ = new ParserVal(); }
  ;
prio
  : IMPORTANT_SYM s_star                                        {      String[][] ruleBody = {
                                                                       {$1.sval, "IMPORTANT_SYM", "true"},
                                                                       {$2.sval, "s_star", "false"} };
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
  ;
expr
  : term expr_ext1star                                          {      String[][] ruleBody = {
                                                                       {$1.sval, "term", "false"},
                                                                       {$2.sval, "expr_ext1star", "false"} };
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
  ;
  expr_ext1star
    : /* empty */                                               { $$ = new ParserVal(); }
    | /* recursion */ expr_ext1star operator term               {      String[][] ruleBody = {
                                                                       {$1.sval, "expr_ext1star", "false"},
                                                                       {$2.sval, "operator", "false"},
                                                                       {$3.sval, "term", "false"} };
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
    ;
term
  : unary_operator_opt term_ext1paren                           {      String[][] ruleBody = {
                                                                       {$1.sval, "unary_operator_opt", "false"},
                                                                       {$2.sval, "term_ext1paren", "false"} };
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
  | STRING s_star                                               {      String[][] ruleBody = {
                                                                       {$1.sval, "STRING", "true"},
                                                                       {$2.sval, "s_star", "false"} };
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
  | IDENT s_star                                                {      String[][] ruleBody = {
                                                                       {$1.sval, "IDENT", "true"},
                                                                       {$2.sval, "s_star", "false"} };
                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                }
  | URI s_star                                                  {      String[][] ruleBody = {
                                                                       {$1.sval, "URI", "true"},
                                                                       {$2.sval, "s_star", "false"} };
```

```
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
| hexcolor                                                                      {      String[][] ruleBody = {
                                                                                       {$1.sval, "hexcolor", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
| function                                                                      {      String[][] ruleBody = {
                                                                                       {$1.sval, "function", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }

;
term_ext1paren
  : NUMBER s_star                                                               {      String[][] ruleBody = {
                                                                                       {$1.sval, "NUMBER", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
  | PERCENTAGE s_star                                                           {      String[][] ruleBody = {
                                                                                       {$1.sval, "PERCENTAGE", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
  | LENGTH s_star                                                               {      String[][] ruleBody = {
                                                                                       {$1.sval, "LENGTH", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
  | EMS s_star                                                                  {      String[][] ruleBody = {
                                                                                       {$1.sval, "EMS", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
  | EXS s_star                                                                  {      String[][] ruleBody = {
                                                                                       {$1.sval, "EXS", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
  | ANGLE s_star                                                                {      String[][] ruleBody = {
                                                                                       {$1.sval, "ANGLE", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
                                                                                }
  | TIME s_star                                                                 {      String[][] ruleBody = {
                                                                                       {$1.sval, "TIME", "true"},
                                                                                       {$2.sval, "s_star", "false"} };
                                                                                  $$ = new ParserVal(encode(ruleBody));
```

```
                                                                                   }
    | FREQ s_star                                                                  {      String[][] ruleBody = {
                                                                                          {$1.sval, "FREQ", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
    ;
function
  : FUNCTION s_star expr RPAREN s_star                                             {      String[][] ruleBody = {
                                                                                          {$1.sval, "FUNCTION", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "expr", "false"},
                                                                                          {$4.sval, "RPAREN", "true"},
                                                                                          {$5.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;

/*
 * There is a constraint on the color that it must
 * have either 3 or 6 hex-digits (i.e., [0-9a-fA-F])
 * after the "#";
 */

hexcolor
  : HASH s_star                                                                    {      String[][] ruleBody = {
                                                                                          {$1.sval, "HASH", "true"},
                                                                                          {$2.sval, "s_star", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;


/* ***************************************** */
/* extensions                                */
/* ***************************************** */

content_value
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | STRING s_star content_value                                                    {      String[][] ruleBody = {
                                                                                          {$1.sval, "STRING" ,"true"},
                                                                                          {$2.sval, "s_star" ,"false"},
                                                                                          {$3.sval, "content_value" ,"false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | ELEMENT_FN s_star element_params s_star RPAREN s_star content_value            {      String[][] ruleBody = {
```

```
                                                                                          {$1.sval, "ELEMENT_FN", "true"},
                                                                                          {$2.sval, "s_star", "false"},
                                                                                          {$3.sval, "element_params", "false"},
                                                                                          {$4.sval, "s_star", "false"},
                                                                                          {$5.sval, "RPAREN", "true"},
                                                                                          {$6.sval, "s_star" ,"false"},
                                                                                          {$7.sval, "content_value" ,"false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | ELEMENT_NAME RPAREN s_star content_value                                       {      String[][] ruleBody = {
                                                                                          {$1.sval, "ELEMENT_NAME" ,"true"},
                                                                                          {$2.sval, "RPAREN" ,"true"},
                                                                                          {$3.sval, "s_star" ,"false"},
                                                                                          {$4.sval, "content_value" ,"false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
element_params
  : STRING                                                                         {      String[][] ruleBody = {
                                                                                          {$1.sval, "STRING" ,"true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | STRING s_star attribute_star COMMA s_star content_value s_star                 {      String[][] ruleBody = {
                                                                                          {$1.sval, "STRING" ,"true"},
                                                                                          {$2.sval, "s_star" ,"false"},
                                                                                          {$3.sval, "attribute_star" ,"false"},
                                                                                          {$4.sval, "COMMA" ,"true"},
                                                                                          {$5.sval, "s_star" ,"false"},
                                                                                          {$6.sval, "content_value" ,"false"},
                                                                                          {$7.sval, "s_star" ,"false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
attribute_star
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | /* recursion */ attribute_star COMMA s_star attribute                          {      String[][] ruleBody = {
                                                                                          {$1.sval, "attribute_star", "false"},
                                                                                          {$2.sval, "COMMA" ,"true"},
                                                                                          {$3.sval, "s_star" ,"false"},
                                                                                          {$4.sval, "attribute", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
attribute
  : ATTRIBUTE_FN s_star STRING s_star COMMA s_star STRING s_star RPAREN            {      String[][] ruleBody = {
                                                                                          {$1.sval, "ATTRIBUTE_FN" ,"true"},
                                                                                          {$2.sval, "s_star" ,"false"},
                                                                                          {$3.sval, "STRING" ,"true"},
                                                                                          {$4.sval, "s_star" ,"false"},
                                                                                          {$5.sval, "COMMA" ,"true"},
                                                                                          {$6.sval, "s_star" ,"false"},
                                                                                          {$7.sval, "STRING" ,"true"},
                                                                                          {$8.sval, "s_star" ,"false"},
                                                                                          {$9.sval, "RPAREN" ,"true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
params_opt
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | IDENT                                                                          {      String[][] ruleBody = {
                                                                                          {$1.sval, "IDENT", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | NUMBER                                                                         {      String[][] ruleBody = {
                                                                                          {$1.sval, "NUMBER", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | NUMBER COMMA NUMBER                                                            {      String[][] ruleBody = {
                                                                                          {$1.sval, "NUMBER", "true"},
                                                                                          {$3.sval, "NUMBER", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
/* **************************************** */


/* **************************************** */
/* ebnf2bnf tools                           */
/* **************************************** */

/* **************************************** */
/* ebnf2bnf for token                       */

s_star
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | /* recursion */ s_star S                                                       {      String[][] ruleBody = {
                                                                                          {$1.sval, "s_star", "false"},
                                                                                          {$2.sval, "S", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
s_plus
  : S                                                                              {      String[][] ruleBody = {
```

```
                                                                                          {$1.sval, "s_plus", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  | /* recursion */ s_plus S                                                       {      String[][] ruleBody = {
                                                                                          {$1.sval, "s_plus", "false"},
                                                                                          {$2.sval, "S", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
ident_opt
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | IDENT                                                                          {      String[][] ruleBody = {
                                                                                          {$1.sval, "IDENT", "true"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;


/* **************************************** */
/* ebnf2bnf for rules                       */

prio_opt
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | prio                                                                           {      String[][] ruleBody = {
                                                                                          {$1.sval, "prio", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
pseudo_page_opt
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | pseudo_page                                                                    {      String[][] ruleBody = {
                                                                                          {$1.sval, "pseudo_page", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
ruleset_star
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | /* recursion */ ruleset_star ruleset                                           {      String[][] ruleBody = {
                                                                                          {$1.sval, "ruleset_star", "false"},
                                                                                          {$2.sval, "ruleset", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;
unary_operator_opt
  : /* empty */                                                                    { $$ = new ParserVal(); }
  | unary_operator                                                                 {      String[][] ruleBody = {
```

```
                                                                                          {$1.sval, "unary_operator", "false"} };
                                                                                     $$ = new ParserVal(encode(ruleBody));
                                                                                   }
  ;


/* **************************************** */

%%

private Ylex lexer;
private static StringBuffer result;

private static String encode(String[][] ruleBody) {
    StringBuffer result = new StringBuffer();
    for(String[] symbol : ruleBody) {
        String symbolValue = symbol[0];
        String symbolName = symbol[1];
        boolean isTerminal  = symbol[2].equals("true");
        /* Does this symbol exist and does it have a value? */
        if (symbolValue != null && !symbolValue.equals("")) {

            /* encode to XML */
            if(isTerminal) {
                symbolValue = symbolValue.replaceAll("&", "&amp;");
                symbolValue = symbolValue.replaceAll(">", "&gt;");
                symbolValue = symbolValue.replaceAll("<", "&lt;");
            }

            result.append("<" + symbolName + ">");
            result.append(symbolValue);
            result.append("</" + symbolName + ">");
        }
    }
    return result.toString();
}

private int yylex () {
    int yyl_return = -1;
    try {
        yylval = new ParserVal(0);
        yyl_return = lexer.yylex();
    }
    catch (IOException e) {
        System.err.println("IO error :"+e);
    }
    return yyl_return;
}

public void yyerror (String error) {
    System.err.println ("Error: " + error);
}


public Parser(Reader r) {
    lexer = new Yylex(r, this);
}


static boolean interactive;

public static String parse(Reader reader) {
    Parser yyparser = new Parser(reader);
    yyparser.yyparse();
    return result.toString();
}

public static void main(String args[]) throws IOException {
    Parser yyparser;
    if ( args.length > 0 ) {
        // parse a file
        yyparser = new Parser(new FileReader(args[0]));
    }
    else {
        // interactive mode
        System.out.println("[Quit with CTRL-D]");
        System.out.print("Expression: ");
        interactive = true;
        yyparser = new Parser(new InputStreamReader(System.in));
    }

    yyparser.yyparse();

    System.out.println(result.toString());

    if (interactive) {
        System.out.println();
        System.out.println("Have a nice day");
    }
}
```

A.3 Configurator

```
<!-- ******************************************************************************** -->
<!-- * Abstract: Transformation of the CSS Abstract Syntax Tree (CSS AST) to XCSS.   * -->
<!-- * CSS AST : XML Tree with CSS grammar as schema.                                 * -->
<!-- * Modes   : Templates are qualified using modes to control their output formats:  * -->
<!-- *           css:           transformation to CSS                                 * -->
<!-- *           cssStatic:     transformation to CSS without pseudo selectors        * -->
<!-- *           XPathForward: transformation of CSS selectors to XPath               * -->
<!-- *           XPathContext: view from a XML element: Do I match this CSS selector?  * -->
<!-- *             (equivalent to XPath looking backward and deleting the first axis.  * -->
<!-- *           XPath:         Where XPathForward and XPathContext would yield the same. * -->
<!-- * Example : AST("a b { }") becomes                                               * -->
<!-- *           <stylesheet>                                                         * -->
<!-- *             <rule>                                                             * -->
<!-- *               <selector type="css">a b:hover</selector>                         * -->
<!-- *               <selector type="cssStatic">a b</selector>                         * -->
<!-- *               <selector type="XPathForward">a/descendant::b</selector>          * -->
<!-- *               <selector type="XPathContext">/ancestor::a</selector>             * -->
<!-- *               <declaration/>                                                    * -->
<!-- *             </rule>                                                             * -->
<!-- *           </stylesheet>                                                        * -->
<!-- *           where AST(String css) transforms css to its CSS AST                  * -->
<!-- ******************************************************************************** -->

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xhtml="http://www.w3.org/1999/xhtml">

  <xsl:param name="STRUCTURE" />
  <xsl:variable name="NAMESPACE-PREFIX">xhtml</xsl:variable>

  <xsl:output method="xml"
              indent="yes"
              media-type="text/xml"
              encoding="ISO-8859-1"
  />

  <!-- ************************************************ -->
  <!-- XCSS - Data Format                               -->
  <!-- ************************************************ -->

  <xsl:template match="/">
    <xsl:element name="stylesheet">
      <xsl:attribute name="structure">
        <xsl:value-of select="$STRUCTURE" />
      </xsl:attribute>

      <!-- CSS-Rules -->
      <xsl:for-each select="//ruleset">
        <rule>
          <selector>
            <xsl:apply-templates mode="XPathContext" select="selector" />
          </selector>
          <xsl:apply-templates mode="dynamic" select="selector" />
          <declaration>
            <xsl:apply-templates mode="standard" select="declaration" />
          </declaration>
          <xsl:element name="insert">
            <xsl:attribute name="type">
              <xsl:copy-of select="selector/simple_selector/hash_class_attrib_pseudo_star/
                  hash_class_attrib_pseudo/pseudo/pseudo_ext1paren/IDENT" />
            </xsl:attribute>
            <xsl:apply-templates mode="insert" select="declaration" />
          </xsl:element>
        </rule>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>

  <!-- ************************************************ -->
  <!-- ************************************************ -->
```

```
<!-- Transformation of selectors                         -->
<!-- ********************************************** -->
<!-- ********************************************** -->

<!-- Transpose CSS selectors to XPathContext selectors -->
<!-- e.g.:     a b c { ... }                            -->
<!-- becomes  self::c[ancestor::b/ancestor::a] { ... } -->

<!-- Input from css-ast.xml +++++++++++ text nodes ++ -->
<!-- selector                          +             -->
<!--   simple_selector                 +   c         -->
<!--     selector_ext1star             +             -->
<!--       selector_ext1star           +             -->
<!--         (...selector_ext1star...) +             -->
<!--       combinator                  +   " "       -->
<!--       simple_selector             +   b         -->
<!--     combinator                    +   " "       -->
<!--     simple_selector               +   a         -->
<!-- +++++++++++++++++++++++++++++++++++++++++++++++ -->

<xsl:template mode="XPathContext" match="selector">
  <!-- "self::" for the first selector -->     <!-- self::c[ancestor::b/ancestor::a] -->
                                               <!-- xxxxxx                           -->
  <xsl:text>self::</xsl:text>

  <!-- Tree selectors -->                      <!-- self::c[ancestor::b/ancestor::a] -->
                                               <!--        xxxxxxxxxxxxxxxxxxxxxxxx  -->
  <xsl:apply-templates mode="XPathContext" select="selector_ext1star" />

                                               <!-- self::c[ancestor::b/ancestor::a] -->
                                               <!--                               x  -->
  <xsl:apply-templates mode="XPath" select="simple_selector" />

  <!-- Closing bracket for tree selections --> <!-- self::c[ancestor::b/ancestor::a] -->
                                               <!--                                x -->
  <xsl:if test="selector_ext1star">
```

```
    <xsl:text>]</xsl:text>
  </xsl:if>
</xsl:template>

<xsl:template mode="XPathContext" match="selector_ext1star">
  <xsl:apply-templates mode="XPath" select="simple_selector" />

  <!-- Opening bracket for tree selections --> <!-- self::c[ancestor::b/ancestor::a] -->
                                             <!--          x                        -->
  <xsl:if test="parent::selector">
    <xsl:text>[</xsl:text>
  </xsl:if>

  <!-- Combinators                        -->   <!-- self::c[ancestor::b/ancestor::a] -->
                                             <!--           xxxxxxxxxx  xxxxxxxxxx   -->
  <xsl:apply-templates mode="XPathContext" select="combinator" />

   <!-- Left recursion                    -->   <!-- self::c[ancestor::b/ancestor::a] -->
                                             <!--                     xxxxxxxxxxxxx  -->
  <xsl:apply-templates mode="XPathContext" select="selector_ext1star" />
</xsl:template>

<!-- *************** -->
<!-- selector/dynamic -->
<xsl:template mode="dynamic" match="selector">
  <xsl:choose>
    <!-- one colon stands for dynamic pseudo-class -->
    <xsl:when test="count(simple_selector/hash_class_attrib_pseudo_star/hash_class_attrib_pseudo/
        pseudo/COLON)=1"><!-- static -->
      <xsl:apply-templates mode="dynamic" />
    </xsl:when>
    <xsl:otherwise><!-- static -->
      <xsl:apply-templates mode="dynamic" select="selector_ext1star/simple_selector" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```

  <!-- ********************************************** -->
  <!-- Transformation of combinator/*                 -->
  <!-- ********************************************** -->

  <xsl:template mode="XPathContext" match="PLUS">
    <!-- Avoid '/' after opening a constraint                       -->
    <!-- e.g.: "self::a[child::b]" instead of "self::a[/child::b]" -->
    <xsl:if test="not(parent::combinator/parent::selector_ext1star/parent::selector)">
      <xsl:text>/</xsl:text>
    </xsl:if>
    <xsl:text>preceding-sibling::</xsl:text>
  </xsl:template>

  <xsl:template mode="XPathContext" match="GREATER">
    <!-- Avoid '/' after opening a constraint, e.g. self::a[/child::b] -->
    <xsl:if test="not(parent::combinator/parent::selector_ext1star/parent::selector)">
      <xsl:text>/</xsl:text>
    </xsl:if>
    <xsl:text>parent::</xsl:text>
  </xsl:template>

  <xsl:template mode="XPathContext" match="s_plus">
    <!-- Avoid '/' after opening a constraint, e.g. self::a[/child::b] -->
    <xsl:if test="not(parent::combinator/parent::selector_ext1star/parent::selector)">
      <xsl:text>/</xsl:text>
    </xsl:if>
    <xsl:text>ancestor::</xsl:text>
  </xsl:template>


  <!-- ********************************************** -->
  <!-- Transformation of simple_selector              -->
  <!-- ********************************************** -->

  <xsl:template mode="XPath" match="simple_selector[element_name][hash_class_attrib_pseudo_star]">
```

```
    <xsl:apply-templates mode="XPath" select="element_name" />
    <xsl:apply-templates mode="XPath" select="hash_class_attrib_pseudo_star" />
  </xsl:template>

  <xsl:template mode="XPath" match="simple_selector[hash_class_attrib_pseudo_plus]">
    <xsl:if test="$STRUCTURE='hxml'"><xsl:text>xhtml:div</xsl:text></xsl:if>
    <xsl:if test="$STRUCTURE='xml'"><xsl:text>xhtml:*</xsl:text></xsl:if>
    <xsl:apply-templates mode="XPath" select="hash_class_attrib_pseudo_plus" />
  </xsl:template>

  <!-- ************************************************ -->
  <!-- Transformation of element-name                    -->
  <!-- ************************************************ -->

  <xsl:template mode="XPath" match="element_name">
    <xsl:if test="$NAMESPACE-PREFIX != ''">
      <xsl:value-of select="$NAMESPACE-PREFIX" />
      <xsl:text>:</xsl:text>
    </xsl:if>
    <xsl:if test="$STRUCTURE='hxml'">
      <xsl:text>xhtml:div[xhtml:span[@xhtml:class=&quot;element&quot;]</xsl:text>
      <xsl:text>/xhtml:span[@xhtml:class=&quot;name&quot;]=&quot;</xsl:text>
      <xsl:apply-templates mode="XPath"/>
      <xsl:text>&quot;]</xsl:text>
    </xsl:if>
    <xsl:if test="$STRUCTURE='xml'">
      <xsl:apply-templates mode="XPath" />
    </xsl:if>
  </xsl:template>

  <xsl:template mode="dynamic" match="element_name" />

  <!-- ************************************************ -->
  <!-- Transformation of hash_class_attrib_pseudo//*     -->
  <!-- ************************************************ -->

```

```
<xsl:template mode="XPath" match="*">
  <xsl:apply-templates mode="XPath" />
</xsl:template>
<xsl:template mode="XPath" match="HASH">
  <xsl:if test="$STRUCTURE='hxml'">
    <xsl:text>[xhtml:span[@xhtml:class=&quot;attribute&quot;]</xsl:text>
    <xsl:text>[xhtml:span[@xhtml:class=&quot;name&quot;]=&quot;id&quot;]</xsl:text>
    <xsl:text>[xhtml:span[@xhtml:class=&quot;value&quot;]=&quot;</xsl:text>
    <xsl:value-of select="substring-after(current(),'#')" />
    <xsl:text>&quot;]]</xsl:text>
  </xsl:if>
  <xsl:if test="$STRUCTURE='xml'">
    <xsl:text>[@xhtml:id=&quot;</xsl:text>
    <xsl:value-of select="substring-after(current(),'#')" />
    <xsl:text>&quot;]</xsl:text>
  </xsl:if>
</xsl:template>

<!-- ************************************************ -->
<!-- Transformation of class                          -->
<!-- ************************************************ -->

<xsl:template mode="XPath" match="class">
  <xsl:if test="$STRUCTURE='hxml'">
    <xsl:text>[xhtml:span[@xhtml:class=&quot;attribute&quot;]</xsl:text>
    <xsl:text>[xhtml:span[@xhtml:class=&quot;name&quot;]=&quot;class&quot;]</xsl:text>
    <xsl:text>[xhtml:span[@xhtml:class=&quot;value&quot;]=&quot;</xsl:text>
    <xsl:value-of select="IDENT" />
    <xsl:text>&quot;]]</xsl:text>
  </xsl:if>
  <xsl:if test="$STRUCTURE='xml'">
    <xsl:text>[@xhtml:class=&quot;</xsl:text>
    <xsl:value-of select="IDENT" />
    <xsl:text>&quot;]</xsl:text>
  </xsl:if>
</xsl:template>
```

```

<!-- ********************************************* -->
<!-- Transformation of attrib                      -->
<!-- ********************************************* -->

<xsl:template mode="XPath" match="attrib">
  <xsl:if test="$STRUCTURE='hxml'">
    <xsl:text>[xhtml:span</xsl:text>
    <xsl:text>[attribute::xhtml:class=&quot;attribute&quot;]</xsl:text>
    <xsl:text>[xhtml:span[attribute::xhtml:class=&quot;name&quot;]=&quot;</xsl:text>
    <xsl:value-of select="IDENT" /><!-- name -->
    <xsl:text>&quot;][xhtml:span[@xhtml:class=&quot;value&quot;]=&quot;</xsl:text>
    <xsl:apply-templates mode="XPath" select="attrib_ext1opt/attrib_ext1opt_ext2paren"/><!-- value
        -->
    <xsl:text>&quot;]]</xsl:text>
  </xsl:if>
  <xsl:if test="$STRUCTURE='xml'">
    <xsl:text>[attribute::xhtml:</xsl:text>
    <xsl:value-of select="IDENT" /><!-- name -->
    <xsl:text>=&quot;</xsl:text>
    <xsl:apply-templates mode="XPath" select="attrib_ext1opt/attrib_ext1opt_ext2paren"/><!-- value
        -->
    <xsl:text>&quot;]</xsl:text>
  </xsl:if>
</xsl:template>

<!-- ********************************************* -->
<!-- Transformation of pseudo                      -->
<!-- ********************************************* -->

<xsl:template mode="XPath" match="pseudo">
  <xsl:if test="count(COLON)=1">
    <xsl:variable name="DYNAMIC-CONSTRAINT">
    <xsl:text>[</xsl:text>
    <xsl:for-each select="pseudo_ext1paren">
      <xsl:text>xhtml:span[@xhtml:class=&quot;event&quot;]/xhtml:span[@xhtml:class=&quot;</xsl:text>
```

```
        <xsl:if test="not(IDENT) and FUNCTION">
          <xsl:value-of select="substring-before(FUNCTION, '(')" />
        </xsl:if>
        <xsl:if test="IDENT and not(FUNCTION)">
          <xsl:value-of select="IDENT" />
        </xsl:if>
      </xsl:for-each>
      <xsl:text>&quot;]</xsl:text>
      <xsl:if test="pseudo_ext1paren/params_opt/NUMBER[2]">
        <xsl:text> mod </xsl:text><!-- modulo -->
        <xsl:value-of select="pseudo_ext1paren/params_opt/NUMBER[2]" />
      </xsl:if>
      <xsl:text> = </xsl:text>
      <xsl:value-of select="pseudo_ext1paren/params_opt/NUMBER[1]" />
      <xsl:if test="pseudo_ext1paren/params_opt/NUMBER[2]">
        <xsl:text> mod </xsl:text><!-- modulo -->
        <xsl:value-of select="pseudo_ext1paren/params_opt/NUMBER[2]" />
      </xsl:if>
      <xsl:text>]</xsl:text>
      </xsl:variable>

      <xsl:element name="css-ng">
        <xsl:value-of select="$DYNAMIC-CONSTRAINT" />
      </xsl:element>
    </xsl:if>
  </xsl:template>

  <xsl:template mode="dynamic" match="pseudo">
    <xsl:apply-templates mode="dynamic" select="pseudo_ext1paren" />
  </xsl:template>

  <xsl:template mode="dynamic" match="pseudo_ext1paren">
    <!-- dynamic --><!-- insert dynamic only for the last simple-selector -->
<xsl:if test="ancestor::simple_selector/parent::selector_ext1star/parent::selector or ancestor::simple_selector[not(preceding-sibling::* or following-sibling::*)]">
    <xsl:element name="dynamic">
```

```
      <!-- Variable ACTION -->
      <xsl:variable name="ACTION">
        <xsl:if test="not(IDENT) and FUNCTION">
          <xsl:value-of select="substring-before(FUNCTION, '(')" />
        </xsl:if>
        <xsl:if test="IDENT and not(FUNCTION) and IDENT!='before' and IDENT!='after'">
          <xsl:value-of select="IDENT" />
        </xsl:if>
      </xsl:variable>
      <!-- @action CSS (e.g. hover) -->
      <xsl:attribute name="action">
        <xsl:value-of select="$ACTION" />
      </xsl:attribute>
      <!-- @event -->
      <xsl:attribute name="event">
        <xsl:choose>
          <xsl:when test="$ACTION='hover'">
            <xsl:text>onmouseover</xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$ACTION" />
          </xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
    </xsl:element>
</xsl:if>
  </xsl:template>

  <!-- ********************************************** -->
  <!-- Transformation of STRING                       -->
  <!-- ********************************************** -->

  <xsl:template mode="XPath" match="STRING">
    <xsl:value-of select="substring(., 2, string-length(.) - 2)" /><!-- Strip the first and the last character -->
  </xsl:template>
```

```

<!-- ********************************************** -->
<!-- Transformation of attrib for CSS |= and CSS ~=    -->
<!-- ********************************************** -->

<xsl:template mode="XPath"
              match="attrib[attrib_ext1opt/attrib_ext1opt_ext1paren/INCLUDES or
                     attrib_ext1opt/attrib_ext1opt_ext1paren/DASHMATCH]">

  <!-- Separator for |= and ~= -->
  <xsl:variable name="SEPARATOR">
    <xsl:choose>
      <!-- *[att~=el] in CSS selects all elements with an attribute 'att'
           having 'el' as one single value of a list of space separated values. -->
      <xsl:when test="attrib_ext1opt/attrib_ext1opt_ext1paren/INCLUDES"> </xsl:when><!-- space -->
      <!-- *[att|=el] in CSS selects all elements with an attribute 'att'
           having 'el' as one single value of a list of dash separated values. -->
      <xsl:when test="attrib_ext1opt/attrib_ext1opt_ext1paren/DASHMATCH">-</xsl:when>
    </xsl:choose>
  </xsl:variable>

  <!-- Value of Attribute -->
  <xsl:variable name="VALUE">
    <xsl:apply-templates mode="XPath" select="attrib_ext1opt/attrib_ext1opt_ext2paren"/>
  </xsl:variable>

  <!-- The following cases are possible ($SEPARATOR='-'):
       att="el- ... -XX- ... -XX" (starts-with)
       att="XX- ... -el- ... -XX" (contains)
       att="XX- ... -XX- ... -el" (ends-with)
  -->
  <xsl:text>[</xsl:text>

  <!-- starts-with -->
  <xsl:text>starts-with(</xsl:text>
  <xsl:value-of select="IDENT" />
```

```
      <xsl:text>, &quot;</xsl:text>
      <xsl:value-of select="$VALUE" />
      <xsl:value-of select="$SEPARATOR" />
      <xsl:text>&quot;)</xsl:text>

      <xsl:text> or </xsl:text>

      <!-- contains -->
      <xsl:text>contains(</xsl:text>
      <xsl:value-of select="IDENT" />
      <xsl:text>, &quot;</xsl:text>
      <xsl:value-of select="$SEPARATOR" />
      <xsl:value-of select="$VALUE" />
      <xsl:value-of select="$SEPARATOR" />
      <xsl:text>&quot;)</xsl:text>

      <xsl:text> or (</xsl:text>

      <!-- ends-with (simulated by contains and substring-after) -->
      <!--    contains -->
      <xsl:text>contains(</xsl:text>
      <xsl:value-of select="IDENT" />
      <xsl:text>, &quot;</xsl:text>
      <xsl:value-of select="$SEPARATOR" />
      <xsl:value-of select="$VALUE" />
      <xsl:text>&quot;)</xsl:text>

      <xsl:text> and </xsl:text>

      <!--    substring-after -->
      <xsl:text>substring-after(</xsl:text>
      <xsl:value-of select="IDENT" />
      <xsl:text>, &quot;</xsl:text>
      <xsl:value-of select="$SEPARATOR" />
      <xsl:value-of select="$VALUE" />
      <xsl:text>&quot;)=&quot;&quot;</xsl:text>
```

```

    <xsl:text>)]</xsl:text>
  </xsl:template>

  <!-- ********************************************** -->
  <!-- kill space                                     -->
  <!-- ********************************************** -->

  <xsl:template mode="XPathContext" match="S" />
  <xsl:template mode="XPath" match="S" />
  <xsl:template match="S" />

  <!-- ********************************************** -->
  <!-- declaration                                    -->
  <!-- ********************************************** -->

  <xsl:template mode="standard" match="declaration">
    <!-- Transform this declaration -->
    <xsl:if test="not(CONTENT)"><!-- content is handeld in mode insert -->
      <xsl:apply-templates />
      <xsl:text>;</xsl:text>
    </xsl:if>

    <!-- Append semicolon if not existing in the AST -->
    <!--<xsl:if test="not(following-sibling::ruleset_ext2star[SEMICOLON])">
      <xsl:text>;</xsl:text>
    </xsl:if>-->

    <!-- Transform next declaration -->
    <xsl:apply-templates select="following-sibling::*//declaration"/>
  </xsl:template>

  <xsl:template mode="insert" match="declaration">
    <xsl:if test="CONTENT">
      <xsl:apply-templates mode="insert" select="content_value" />
    </xsl:if>
  </xsl:template>


  <!-- ********************************************** -->
  <!-- content_value                                  -->
  <!-- ********************************************** -->

  <xsl:template mode="insert" match="content_value">
    <xsl:apply-templates mode="insert" />
  </xsl:template>

  <xsl:template mode="insert" match="STRING">
    <xsl:value-of select="substring(.,2,string-length()-2)" />
  </xsl:template>

  <xsl:template mode="insert" match="element_params">
    <xsl:variable name="ELEMENT-NAME" select="substring(STRING[1],2,string-length(STRING[1])-2)" />
    <xsl:element name="{$ELEMENT-NAME}" namespace="http://www.w3.org/1999/xhtml">
      <xsl:apply-templates mode="insert" select="//attribute" />
      <xsl:apply-templates mode="insert" select="content_value" />
    </xsl:element>
  </xsl:template>

  <xsl:template mode="insert" match="ELEMENT_NAME">
    <ELEMENT_NAME />
  </xsl:template>

  <xsl:template mode="insert" match="attribute">
    <xsl:variable name="ATTRIBUTE-NAME" select="substring(STRING[1],2,string-length(STRING[1])-2)" />
    <xsl:attribute name="xhtml:{$ATTRIBUTE-NAME}">
      <xsl:value-of select="substring(STRING[2],2,string-length(STRING[2])-2)" />
    </xsl:attribute>
  </xsl:template>

  <xsl:template mode="insert" match="*" />
</xsl:stylesheet>
```

A.4 Styler-Generator

```
<?xml version="1.0" encoding="iso-8859-1"?>

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xhtml="http://www.w3.org/1999/xhtml"
                xmlns:out="http://www.wieser.info">

  <xsl:namespace-alias stylesheet-prefix="out" result-prefix="xsl"/>

  <xsl:strip-space elements="selector" />

  <xsl:output method="xml"
              indent="yes"
              media-type="text/xml"
              encoding="ISO-8859-1"
  />

  <xsl:template match="/">
    <out:stylesheet version="1.0"

                    xmlns:xhtml="http://www.w3.org/1999/xhtml">

      <out:output method="xml"
                  indent="yes"
                  media-type="text/html"
                  encoding="ISO-8859-1"
                  doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
                  doctype-system="xhtml1-transitional.dtd" />

      <xsl:apply-templates />
    </out:stylesheet>
  </xsl:template>

  <xsl:template match="stylesheet">
```

```
    <out:template match="*">

      <!-- ***************************************************** -->
      <!-- ***************************************************** -->
      <!-- Generate a modified copy of the current element node       -->
      <!-- ***************************************************** -->
      <!-- ***************************************************** -->
      <xsl:comment>Generate a copy of the current element</xsl:comment>
      <xsl:comment>(with a modified style attribute).</xsl:comment>

      <xsl:variable name="ELEMENT">{name()}</xsl:variable>
      <out:element name="{$ELEMENT}" namespace="http://www.w3.org/1999/xhtml">

        <!-- ******************************************** -->
        <!-- event attributes (onmouseover, ...)           -->
        <!-- ******************************************** -->
        <xsl:comment>Set style attribute and dynamic attributes</xsl:comment>
        <xsl:comment>for each dynamic rule defined in xcss</xsl:comment>

        <!-- "For each distinct event in XCSS is not supported in XSLT 1.0" -->
        <!-- "Therefore these events are statical evaluated." -->
        <xsl:call-template name="computeEventAttribute">
          <xsl:with-param name="EVENT">onclick</xsl:with-param>
        </xsl:call-template>
<!--
        <xsl:call-template name="computeEventAttribute">
          <xsl:with-param name="EVENT">onmouseover</xsl:with-param>
        </xsl:call-template>
        <xsl:call-template name="computeEventAttribute">
          <xsl:with-param name="EVENT">onmouseout</xsl:with-param>
        </xsl:call-template>
-->

        <!-- ******************************************** -->
        <!-- style attribute                               -->
        <!-- ******************************************** -->
        <out:variable name="OLD-STYLE" select="@style" />
        <out:variable name="NEW-STYLE">
          <xsl:for-each select="rule[not(dynamic)]">
            <xsl:variable name="SELECTOR">
              <xsl:apply-templates mode="static" select="selector" />
            </xsl:variable>
            <out:if test="{$SELECTOR}">
              <out:text>
                <xsl:value-of select="declaration" />
              </out:text>
            </out:if>
          </xsl:for-each>
        </out:variable>
        <out:if test="ancestor::xhtml:body">
          <out:attribute name="style">
            <out:if test="$NEW-STYLE!='' or $OLD-STYLE!=''">
              <out:value-of select="$NEW-STYLE" />
              <out:value-of select="$OLD-STYLE" />
            </out:if>
          </out:attribute>
        </out:if>

        <!-- *********************************************** -->
        <!-- init dynamic styling                            -->
        <!-- *********************************************** -->
        <out:if test="self::xhtml:body">
          <out:attribute name="onload">
            <out:text>commit();</out:text>
          </out:attribute>
        </out:if>

        <!-- *********************************************** -->
        <!-- other attributes                                -->
        <!-- *********************************************** -->
        <xsl:comment>Generate a copy of the old attributes except</xsl:comment>
        <xsl:comment>the style attribute and dynamic attributes.</xsl:comment>

        <xsl:variable name="DEFINED-EVENTS">
          <xsl:for-each select="rule/dynamic/@event">
            <xsl:text> and name()!='</xsl:text>
            <xsl:value-of select="." />
            <xsl:text>'</xsl:text>
          </xsl:for-each>
        </xsl:variable>

        <!-- Copy attributes -->
        <xsl:element name="xsl:for-each">
          <xsl:attribute name="select">
            <xsl:text>@*[name()!='style'</xsl:text>
            <xsl:value-of select="$DEFINED-EVENTS" />
            <xsl:text>]</xsl:text>
          </xsl:attribute>
          <xsl:variable name="ATTRIBUTE">{name()}</xsl:variable>
          <out:attribute name="{$ATTRIBUTE}">
            <out:value-of select="." />
          </out:attribute>

        </xsl:element><!-- /out:for-each -->

        <!-- ********************************************************* -->
        <!-- ********************************************************* -->
        <!-- traverse child element nodes                              -->
        <!-- ********************************************************* -->
        <!-- ********************************************************* -->
        <xsl:comment>Traverse the child nodes.</xsl:comment>
        <out:apply-templates />

        <!-- ********************************************************* -->
        <!-- Install libraries                                         -->
        <!-- ********************************************************* -->
        <xsl:comment>Install libraries</xsl:comment>
        <out:if test="self::xhtml:head">
        <xsl:element name="script">
          <xsl:attribute name="type"><xsl:text>text/javascript</xsl:text></xsl:attribute>
          <xsl:attribute name="src"><xsl:text>../servlet/javascript/xpath.js</xsl:text></xsl:attribute>
        </xsl:element>
        <xsl:element name="script">
          <xsl:attribute name="type"><xsl:text>text/javascript</xsl:text></xsl:attribute>
          <xsl:attribute name="src"><xsl:text>../servlet/javascript/tools.js</xsl:text></xsl:attribute>
        </xsl:element>
        <xsl:comment>Dummy stylesheet for panorama</xsl:comment>
        <xsl:element name="style" />
      </out:if>

    <!-- ******************************************************* -->
    <!-- Init Meta-Data                                          -->
    <!-- ******************************************************* -->

    <out:if test="ancestor::xhtml:body and not(ancestor-or-self::xhtml:span[@class='event']) and not(self::xhtml:script) and node()">
      <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
        <out:attribute name="style"><xsl:text>display:none</xsl:text></out:attribute>
        <out:attribute name="class"><xsl:text>event</xsl:text></out:attribute>
         <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
          <out:attribute name="class">standard</out:attribute>
            <xsl:text>;</xsl:text><!-- Workaround XPath-Processor Bug -->

           <xsl:for-each select="rule">
            <xsl:element name="xsl:if">
              <xsl:attribute name="test">
                <xsl:apply-templates mode="static" select="selector" />
              </xsl:attribute>
              <out:text>
                <xsl:text>styleMe(element,'</xsl:text><!-- contextNode -->
                <xsl:apply-templates mode="dynamic" select="selector" /><!-- selector -->
                <xsl:text>','</xsl:text>
```

```
                  <xsl:value-of select="dynamic/@action" /><!-- action -->
                  <xsl:text>','</xsl:text>
                  <xsl:value-of select="dynamic/@event" /><!-- event -->
                  <xsl:text>','</xsl:text>
                  <xsl:value-of select="declaration" /><!-- declaration -->
                  <xsl:text>');</xsl:text>
                </out:text>
              </xsl:element>
            </xsl:for-each>
          </out:element>
          <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
            <out:attribute name="class">nth-descendant</out:attribute>
            <out:value-of select="count(ancestor::*)" />
          </out:element>
          <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
            <out:attribute name="class">nth-child</out:attribute>
            <out:value-of select="count(preceding-sibling::*)" />
          </out:element>
          <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
            <out:attribute name="class">onclick</out:attribute>
            <out:text>0</out:text>
          </out:element>
<!--
          <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
            <out:attribute name="class">onmouseover</out:attribute>
            <out:text>0</out:text>
          </out:element>
          <out:element name="span" namespace="http://www.w3.org/1999/xhtml">
            <out:attribute name="class">onmouseout</out:attribute>
            <out:text>0</out:text>
          </out:element>
-->
        </out:element>
      </out:if>

      </out:element>
```

```
    </out:template>

  </xsl:template>

  <xsl:template mode="static" match="css-ng" />
  <xsl:template mode="dynamic" match="css-ng">
    <xsl:value-of select="." />
  </xsl:template>

  <xsl:template mode="insert" match="insert">
    <xsl:apply-templates mode="insert" />
  </xsl:template>
  <xsl:template mode="insert" match="ELEMENT_NAME">
    <out:value-of select="name()" />
  </xsl:template>
  <xsl:template mode="insert" match="*">
     <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates mode="insert" />
    </xsl:copy>
  </xsl:template>

  <xsl:template name="computeEventAttribute">
    <xsl:param name="EVENT"/>

    <xsl:element name="xsl:if">
      <xsl:attribute name="test">
        <xsl:text>ancestor::xhtml:body</xsl:text>
        <xsl:text> and not(self::xhtml:span[@class='standard'])</xsl:text>
        <xsl:text> and not(self::xhtml:span[@class='event'])</xsl:text>
        <xsl:text> and not(self::xhtml:span[@class='</xsl:text>
        <xsl:value-of select="$EVENT" />
        <xsl:text>'])</xsl:text>
      </xsl:attribute>

      <!-- Save old event action of the source XML document (e.g. onmouseover) -->
      <xsl:element name="xsl:variable">
        <xsl:attribute name="name">
          <xsl:text>OLD-</xsl:text><xsl:value-of select="$EVENT" /><xsl:text>-VALUE</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="select">
          <xsl:text>@</xsl:text>
          <xsl:value-of select="$EVENT" />
        </xsl:attribute>
      </xsl:element><!-- /out:variable -->

      <!-- Set event actions -->

      <xsl:element name="xsl:attribute">
        <xsl:attribute name="name">
          <xsl:value-of select="$EVENT" />
        </xsl:attribute>
        <out:text>increment(this,'</out:text>
        <xsl:value-of select="$EVENT" />
        <out:text>');commit();</out:text>

        <out:if test="$OLD-{$EVENT}-VALUE!=''">
          <out:value-of select="$OLD-{$EVENT}-VALUE" />
        </out:if>
      </xsl:element><!-- /out:attribute -->
    </xsl:element><!-- /out:if -->
  </xsl:template>

</xsl:stylesheet>
```

A.5 Insertion Preprocessor

```
<?xml version="1.0" encoding="iso-8859-1"?>

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xhtml="http://www.w3.org/1999/xhtml"
                xmlns:out="http://www.wieser.info">

  <xsl:namespace-alias stylesheet-prefix="out" result-prefix="xsl"/>

  <xsl:output method="xml"
              indent="yes"
              media-type="text/xml"
              encoding="ISO-8859-1"
  />

  <xsl:template match="/">
    <out:stylesheet version="1.0"

                    xmlns:xhtml="http://www.w3.org/1999/xhtml">

      <out:output method="xml"
                  indent="yes"
                  media-type="text/html"
                  encoding="ISO-8859-1"
                  doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
                  doctype-system="xhtml1-transitional.dtd" />

      <xsl:apply-templates />
    </out:stylesheet>
  </xsl:template>

  <xsl:template match="stylesheet">
    <out:template match="*">

    <!-- ************************************************************ -->
    <!-- Insert before                                                -->
    <!-- ************************************************************ -->
    <xsl:for-each select="rule[insert/@type='before']">
      <xsl:variable name="SELECTOR" select="selector" />
      <out:if test="{$SELECTOR} and ancestor::xhtml:body">
        <xsl:apply-templates mode="insert" select="insert" />
      </out:if>
    </xsl:for-each>

    <!-- ************************************************************ -->
    <!-- ************************************************************ -->
    <!-- Generate a exact copy of the current element node       -->
    <!-- ************************************************************ -->
    <!-- ************************************************************ -->
    <xsl:comment>Generate a copy of the current element</xsl:comment>

    <xsl:variable name="ELEMENT">{name()}</xsl:variable>
    <out:element name="{$ELEMENT}" namespace="http://www.w3.org/1999/xhtml">

      <xsl:element name="xsl:for-each">
        <xsl:attribute name="select">
          <xsl:text>@*</xsl:text>
        </xsl:attribute>
        <xsl:variable name="ATTRIBUTE">{name()}</xsl:variable>
        <out:attribute name="{$ATTRIBUTE}">
          <out:value-of select="." />
        </out:attribute>

      </xsl:element><!-- /out:for-each -->

      <!-- ************************************************************ -->
      <!-- ************************************************************ -->
      <!-- traverse child element nodes                                 -->
      <!-- ************************************************************ -->
      <!-- ************************************************************ -->
```

```
        <xsl:comment>Traverse the child nodes.</xsl:comment>
        <out:apply-templates />

      </out:element>

      <!-- ****************************************************** -->
      <!-- Insert after                                            -->
      <!-- ****************************************************** -->
      <xsl:for-each select="rule[insert/@type='after']">
        <xsl:variable name="SELECTOR" select="selector" />
        <out:if test="{$SELECTOR} and ancestor::xhtml:body">
          <xsl:apply-templates mode="insert" select="insert" />
        </out:if>
      </xsl:for-each>
    </out:template>

  </xsl:template>

  <xsl:template mode="insert" match="insert">
    <xsl:apply-templates mode="insert" />
  </xsl:template>
  <xsl:template mode="insert" match="ELEMENT_NAME">
    <out:value-of select="name()" />
  </xsl:template>
  <xsl:template mode="insert" match="*">
     <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates mode="insert" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

A.6 Reifier

```
<?xml version="1.0" encoding="iso-8859-1"?>

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"
              indent="yes"
              media-type="xml/html"
              encoding="ISO-8859-1"
  />

  <!-- generate HTML document -->
  <xsl:template match="/">

    <html>
      <head>
        <title>XHTMLalized XML</title>
        <!--<link rel="stylesheet" type="text/css" href="style.css" />-->
        <style type="text/css">
          span { display:none; }
        </style>
      </head>
      <body>
        <!-- transform all XML elements -->
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <!-- encode each child XML element -->
  <xsl:template match="*">
    <xsl:element name="div">
      <!-- Install XHTML event handler in the div-element.
           see mapping of other attributes in the following -->
      <xsl:for-each select="@*[substring-before(name(), ':') = 'metaxhtml']">
        <xsl:attribute name="{substring-after(name(), ':')}">
          <xsl:value-of select="." />
        </xsl:attribute>
      </xsl:for-each>

      <!-- iterate over children -->
      <xsl:for-each select=".">

        <!-- XML element -->
        <span class="element">
          <!-- separate namespace-prefix, if existing -->
          <xsl:choose>
            <xsl:when test="contains(name(),':')">
              <span class="name"><xsl:value-of select="substring-after(name(),':')" /></span>
              <span class="namespace-prefix"><xsl:value-of select="substring-before(name(),':')" /></
                  span>
            </xsl:when>
            <xsl:otherwise>
              <span class="name"><xsl:value-of select="name()" /></span>
            </xsl:otherwise>
          </xsl:choose>
          <xsl:variable name="NAMESPACE-URI" select="namespace-uri()" />
          <xsl:if test="$NAMESPACE-URI != ''">
            <span class="namespace"><xsl:value-of select="$NAMESPACE-URI" /></span>
          </xsl:if>
        </span>

        <!-- XML attributes of XML element -->
        <xsl:for-each select="@*">

          <!-- Prevent insertion of XHTML event handler -->
          <xsl:if test="not(contains(name(),'metaxhtml:'))">

            <span class="attribute">
              <!-- separate namespace-prefix, if existing -->
              <xsl:choose>
                <xsl:when test="contains(name(),':')">
                  <span class="name"><xsl:value-of select="substring-after(name(),':')" /></span>
                  <span class="namespace-prefix"><xsl:value-of select="substring-before(name(),':')" /></span>
                </xsl:when>
                <xsl:otherwise>
                  <span class="name"><xsl:value-of select="name()" /></span>
                </xsl:otherwise>
              </xsl:choose>
              <span class="value"><xsl:value-of select="." /></span>
            </span>
          </xsl:if>
        </xsl:for-each>

        <!-- recursive call -->
        <xsl:apply-templates />
      </xsl:for-each>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

A.7 Meta-Initializer

```
<?xml version="1.0" encoding="iso-8859-1"?>

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xhtml="http://www.w3.org/1999/xhtml">

  <xsl:output method="xml"
              indent="yes"
              media-type="xml/html"
              encoding="ISO-8859-1" />

  <!-- encode each child XML element -->
  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:for-each select="@*">
        <xsl:attribute name="{name()}">
          <xsl:value-of select="." />
        </xsl:attribute>

      </xsl:for-each>
      <!-- Init counters for dynamic events -->
      <xsl:if test="ancestor::xhtml:body">
        <span class="events">
          <span class="onclick">0</span>
          <span class="onmouseover">0</span>
          <span class="onmouseout">0</span>
        </span>
      </xsl:if>

      <xsl:apply-templates />
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

A.8 Dynamic Styler

```

function styleMe(contextNode, selector, action, event, style) {

    // alert(selector + " " + action + " " + event + " " + style);

   /* ************************************* */
   /* dynamic styling                       */
   /* ************************************* */

   /*
    The parameter action is different from the parameter event because
    there are values like 'hover' being no DOM event but an css action.
   */

   if (evalXPath(contextNode, selector).length > 0) {
      /* ************************************* */
      /* mark element                          */
      /* ************************************* */

      var event = ''; // An action can consist of at least one event.
      if (action == "hover") { event = "onmouseover"; } else { event = action; }

      var actionCounterInt = 0;
      if (action != '') {
        var actionCounter = evalXPath(contextNode, "xhtml:span[@xhtml:class='event']/xhtml:span[@xhtml
            :class='" + event + "']")[0];
        actionCounterInt = parseInt(actionCounter.innerHTML);
    }

      /* ************************************* */
      /* panorama                              */
      /* ************************************* */

      if (panorama != '') {

         // apply style
         var panoramaStyle = panorama + "{" + style + "}";
         var lastStylesheet = document.styleSheets.length - 1;
         var stylesheet = document.styleSheets[lastStylesheet];
         var ruleNumber = stylesheet.cssRules.length;
         stylesheet.insertRule(panoramaStyle, ruleNumber);

         // install undo
         if (action == 'hover') {
            var setOldStyle = "document.styleSheets[" + lastStylesheet + "].deleteRule(" + ruleNumber + ");";
            setAttributeValue(contextNode, 'onmouseout', setOldStyle);
         }
       } else {

      /* ************************************ */
      /* monorama                             */
      /* ************************************ */

         /* install undo */
         if (action == 'hover') {
            var oldStyle = getAttributeValue(contextNode, 'style');
            var codeSetOldStyle = "setAttributeValue(this, 'style', '" + oldStyle + "');";
            setAttributeValue(contextNode, 'onmouseout', codeSetOldStyle);
         }

         /* apply style */
         var oldStyle = getAttributeValue(contextNode, 'style');

         setAttributeValue(contextNode, 'style', oldStyle+style);
      }

//    }

}
```

```

function commit() {
      mapElementNodes(getBody(), applyStyling);
}

function increment(contextNode, event) {
      var actionCounter = evalXPath(contextNode, "xhtml:span[@xhtml:class='event']/xhtml:span[@xhtml:
         class='" + event + "']")[0];
      var actionCounterInt = parseInt(actionCounter.innerHTML) + 1;
      actionCounter.innerHTML = actionCounterInt;
}

function decrement(contextNode, event) {
      var actionCounter = evalXPath(contextNode, "xhtml:span[@xhtml:class='event']/xhtml:span[@xhtml:
         class='" + event + "']")[0];
      var actionCounterInt = parseInt(actionCounter.innerHTML) -1;
      actionCounter.innerHTML = actionCounterInt;
}

function getBody(){
    return document.getElementsByTagName("BODY")[0]; // 1st body element in an HTML document
}

function getAttributeValue(contextNode, attributeName) {
   for (var i=0; i<contextNode.attributes.length; i++) {
      if (contextNode.attributes[i].nodeName == attributeName) {
         return contextNode.attributes[i].nodeValue;
      }
   }
   return ""; // no value
}

function setElement(contextNode, name) {
   var newElement = document.createElement(name);
   contextNode.appendChild(newElement);
   return newElement;
```

```
}

function setAttributeValue(contextNode, name, value) {
   var newAttribute = document.createAttribute(name);
   newAttribute.value=value;
   contextNode.setAttributeNode(newAttribute);
}

function evalXPath(contextNode, xpathExpression) {

   // MyNamespaceResolver
   MyNamespaceResolver.prototype = new NamespaceResolver();
   MyNamespaceResolver.prototype.constructor = MyNamespaceResolver;
   MyNamespaceResolver.superclass = NamespaceResolver.prototype;

   function MyNamespaceResolver() {
   }

   MyNamespaceResolver.prototype.getNamespace = function(prefix, n) {
      // Always resolve the prefix "xhtml".
      if (prefix == "xhtml") {
         return "http://www.w3.org/1999/xhtml";
      }
      return MyNamespaceResolver.superclass.getNamespace(prefix, n);
   };

   var parser = new XPathParser();
   var xpath = parser.parse(xpathExpression);
   var context = new XPathContext();

   var varRes = new VariableResolver();
   var nsRes = new MyNamespaceResolver();
   var funRes = new FunctionResolver();
   var context = new XPathContext(varRes, nsRes, funRes);

   context.expressionContextNode = contextNode;

   var result = xpath.evaluate(context);
   return result.toArray();
}

function applyStyling(element) {
   // XPath-Processor Bug: Binding of text results on the first level instead of the second level, if element on the second level is empty. Workaround: Default value "true();"
   var styling = evalXPath(element, "xhtml:span[@xhtml:class='event']/xhtml:span['standard']")[0];
   if (styling != null) {
     if (styling.innerHTML != ';') { // Workaround: Logical AND does not work.
       eval(styling.innerHTML);
     }
   }

}

function mapElementNodes(root, f) {
    f(root);
    var childElementNodes = getChildElementNodes(root);
    for (var i=0; i<childElementNodes.length; i++) {
        mapElementNodes(childElementNodes[i], f);
    }
}

function hasChildElementNodes(root) {
    return getChildElementNodes(root).length > 0;
}

function getChildElementNodes(root) {
    var result = new Array(0);
    var ELEMENT_NODE = 1;            // standardized number of DOM type 'element node'

    // Iterate over all DOM child nodes and filter out all element nodes
    for(var i=0; i<root.childNodes.length; i++) {
        if (root.childNodes[i].nodeType==ELEMENT_NODE) {
                result.push(root.childNodes[i]);
            }
        }
        return result;
}

function next(element) {
    var mod = parseInt(evalXPath(element, "xhtml:span[@xhtml:class='event']/xhtml:span[@xhtml:class='
        onclick']")[0].innerHTML)   2;var following = evalXPath(element, "following-sibling::*")[0];if (mod==1) var oldStyle =
        getAttributeValue(following, 'style');setAttributeValue(following, 'style', oldStyle + 'display:none');if (mod==0) var oldStyle =
        getAttributeValue(following, 'style');setAttributeValue(following, 'style', oldStyle + 'display:block');
```

APPENDIX
B

Code of the visXcerpt Viewer in CSS^{NG}

```
/* ### Basic Styling Specifications ### */
* {
          display:        block;
          position:       relative;          /* allow relocating elements */
          z-index:        0;                 /* neutral overlay position */
          border-width:   thin;              /* borders */
          border-style:   solid;
          margin-left:    2em;               /* indentation of elements */
          margin-right:   2em;
          padding-left:   0.5em;             /* space around text nodes */
          padding-right:  0.5em;
          padding-top:    0.5em;
          padding-bottom: 0.5em; }
.unordered { border-style:   dotted; }       /* ordered data terms */
.ordered   { border-style:   solid; }        /* unordered data terms */

/* ### Insertion of Tabs ### */
*::before {
          content: element("tab",            /* insert element "tab" */
                           attribute("order", attr("order")),
                           element-name()    /* value of the tab element */
                             element("attribute", *{ content: " " attribute-name()
                                                   " " attribute-value(); } ) ); }

/* ### Basic Tab Styling ### */
tab {
```

```
           top:            0.04em;            /* set tabs *deeper* to */
           z-index:        1;                 /* *overlay* top border of element bodies */
           border-bottom:  none;              /* no border bottom (linking tab and body) */
           padding-top:    0em;               /* no space above tab names and tab border */
           padding-bottom: 0em;               /* no space below tab names and tab border */
           margin-top:     1em;               /* interspace to elements above */
           width:          5em; }             /* tab width */
attribute {                                   /* styling of attribute visualizations */
           border:         thin dotted gray;
           color:          black;
           background-color:white; }

/* ### Folding elements on odd number of clicks ### */
tab:onclick(2n+1) {                           /* ## Folded Tab Styling ## */
           display:        inline;            /* juxtaposing of folded tabs */
           margin-left:    0em;               /* distance between folded tabs */
           left:           0.5em;             /* indent folded tabs a bit */
           z-index:        -1; }              /* disconnect tab with body */
tab:onclick(2n+1) + * {                       /* ## Folded Body Styling ## */
           display:        none; }            /* hide element */


/* ### Unfolding elements on even number of clicks ### */
tab:onclick(2n+2) {                           /* ## Unfolded Tab Styling ## */
           display:        block;             /* juxtaposing of folded tabs */
           margin-left:    2em;               /* standard indentation */
           left:           0em;               /* same indentation of tab and body */
           z-index:        1; }               /* connect tab with body */

tab:onclick(2n+2) + * {                       /* ## Unfolded Body Styling ## */
           display:        block;             /* show element */

/* ### Color Definition of nested Elements ### */
*:nth-decendent(6n+1) { background-color: #bfbfff; color: #3f3f7f; }
*:nth-decendent(6n+2) { background-color: #bfffbf; color: #3f7f3f; }
*:nth-decendent(6n+3) { background-color: #ffbfbf; color: #7f3f3f; }
*:nth-decendent(6n+4) { background-color: #ffffbf; color: #7f7f3f; }
*:nth-decendent(6n+5) { background-color: #ffbfff; color: #7f3f7f; }
*:nth-decendent(6n+6) { background-color: #bfffff; color: #3f7f7f; }

/* ### Additional Xcerpt rendering rules by Christoph Wieser ### */
all, and, or {
   padding-left: 3em;
   background-repeat:  no-repeat; }
all {
   background-image:   url(all.png); }
```

```
and {
   background-image:   url(and.png); }
or {
   background-image:   url(or.png); }
head, query {
   width: 40%; }
query {
   position:absolute;
   right:0em;
   top:1em; }
rule, goal {
  background-image:   url(left-arrow.png);
  background-position: center;
  background-repeat: no-repeat; }

/* ### Xcerpt Rendering from Sacha Berger ### */
xcerpt-rule {
           border-style:    solid;
           border-width:    1px;
           border-color:    black;
           margin:          10px 0px; }
rule-titlebar {
           background-image:url(../IMAGES/halftone_light90.png);
           border-style:    none none solid none;
           border-color:    black;
           border-width:    2px; }
goal {
           background-color:yellow; }

query-resource {
           border-style:    none none solid none;
           border-color:    black;
           border-width:    1px;
           margin:          0 0 4px 0; }
query {
           border-style:    solid;
           border-color:    black;
           margin:          4px 0;
           border-width:    1px; }
query-content {
           padding:4px; }
and {
           background-color:white;
           border-style:    solid;
           border-width:    1px;
           margin:          4px 0px;
           padding:         4px;
           border-color:    black; }
or {
           background-color:white;
           border-style:    solid;
           border-width:    1px;
           margin:          4px 0px;
           padding:         4px;
           border-color:    black; }
resource {
           color:           black;
           margin:          0 4px;
           padding:         0px 5px; }
```

INDEX

> (CSS), 13
`+` (CSS), 13
`~` (CSS), 13
`:nth-descendant(An+B)` (CSS^{NG}), 38
`:onkeydown` (CSS^{NG}), 40, 43
`:onkeyup` (CSS^{NG}), 43
`:onmouseout` (CSS^{NG}), 39
`:onmouseover` (CSS^{NG}), 39
`:root (CSS)`, 27
`?` (CSS^{NG}), 44
`attr()` (CSS), 25
`attribute()` (CSS^{NG}), 31
`attribute-name()` (CSS^{NG}), 33, 35
`attribute-value()` (CSS^{NG}), 34
`element()` (CSS^{NG}), 30
`element-name()` (CSS^{NG}), 33, 35

Abstract Syntax Tree, 57
accumulator, 40
Action Sheets, 49
acyclic event, 41
adding XML elements, 21
adding XML text nodes, 21
AJAX, 55
annotation, 71
AST, 57
attribute-name function, 37
attribute-value function, 37
attribute rule (CSS^{NG}), 35, 37

Behavioral Extension to CSS, 50
BNF, 56
Box-Model, 74

cascading styling, 16
combinator (CSS), 13
combined complexity (CSS^{NG}), 29
Configurator, 58
CSS
    :root, 27
    >, 13
    `+`, 13
    `~`, 13
    `attr()`, 25
    combinator, 13
    declaration, 11
    depth-dependant styling, 26
    dynamic pseudo-classes, 15
    dynamic rendering, 26
    functions, 14
    grouping, 13
    insertion of markup, 24
    interpretation, 16
    property, 11
    pseudo-element, 14
    pseudo-selector, 13
    rule anatomy, 1
    scope, 16
    selector, 11
    separation of concerns, 19
    simple selector, 12
    structural pseudo-class, 15
    visualization of markup, 24
CSS and XML, 10
CSS^{NG}
    `:nth-descendant(An+B)`, 38
    `:onkeydown`, 40, 43
    `:onkeyup`, 43
    `:onmouseout`, 39
    `:onmouseover`, 39
    `?`, 44
    `attribute()`, 31
    `attribute-name()`, 33, 35

- `attribute-value()`, 34
- `element()`, 30
- `element-name()`, 33, 35
- attribute rule, 35, 37
  - declaration, 37
  - grouping, 36
  - selector, 35
  - structural pseudo-selectors, 36
- combined complexity, 29
- depth-dependant, 38
- downward compatibility, 29
- dynamic styling, 39
- event interpretation, 42
- framework, 29
- markup querying, 33
- markup insertion, 29
- structure-independent selection, 44

cyclic event, 41

data terms, 77
declaration (CSS), 11
deleting of XML elements, 20
depth-dependant (CSS^{NG}), 38
depth-dependant styling (CSS), 26
DHTML, 47
DOM, 3
downward compatibility (CSS^{NG}), 29
dummy element, 63
dynamic pseudo-classes (CSS), 15
dynamic styling, 4
Dynamic HTML, 47
dynamic rendering (CSS), 26
dynamic styling (CSS^{NG}), 39

EBNF, 56
ECMA, 3
event accumulator, 40
event interpretation (CSS^{NG}), 42

Flex, 56
FOAF, 72
framework (CSS^{NG}), 29
functions (CSS), 14

graph-visualization, 76
grouping (CSS), 13

hovering, 2
HTML
- intrinsic event attribute, 39
- intrinsic event, 3
- tool-tip, 62

hyperlink, 27

insertion of markup (CSS), 24
intermediary step, 20
interpretation (CSS), 16
intrinsic event, 3
intrinsic event attribute (HTML), 39
iteration, 19
iterative markup insertion, 22

JFlex, 56

keyboard, 26, 43, 71

lexer, 55

markup insertion
- iteratively, 19

markup querying (CSS^{NG}), 33
markup insertion (CSS^{NG}), 29
markup visualization, 33
Meta-Initializer, 64
monorama, 45
mouse, 43, 71

panorama, 45
parameterization, 26
parser, 56
PI, 10
Processing Instruction, 10
property (CSS), 11
pseudo-classes, 14
pseudo-element (CSS), 14, 20
pseudo-selector (CSS), 13

RDF, 72
recurrence pattern, 15, 40
reification, 62
rendering, 19
rendering engine, 51
rule anatomy (CSS), 1
run-time evaluation, 66

scope (CSS), 16
selector (CSS), 11

separation of concerns (CSS), 19
simple selector (CSS), 12
slight transformation, 19
social network, 72
static styling, 3
structural pseudo-class (CSS), 15
structure-independent selection (CSS^{NG}), 44
Styled Document, 64
Styler-Generator, 61
superimposing, 2

tool-tip, 62
transformation, 19

variadic function, 30
Venkman, 54
virtual root element, 32
visualization, 20
visualization of markup (CSS), 24
visXcerpt, 77

well-formedness, 32

Xcerpt, 77
    construction term, 81
    data terms, 77
    grouping constructs, 81
    query term, 81
    variable, 81
XPathJS, 66

Yacc, 56

BIBLIOGRAPHY

[ABC$^+$99] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. *HTML 4.01.* W3C, 1999. 3, 39, 47, 64, 70, 71

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000. 1

[AEGR98] Vidur Apparao, Brendan Eich, Ramanathan Guha, and Nisheeth Ranjan. *Action Sheets: A Modular Way of Defining Behavior for XML and HTML.* W3C, 1998. 49, 50

[AGW99] Vidur Apparao, Daniel Glazman, and Chris Wilson. *Behavioral Extensions to CSS.* W3C, 1999. 50

[BBSW03] Sacha Berger, François Bry, Sebastian Schaffert, and Christoph Wieser. Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In *Proceedings of 29th Intl. Conference on Very Large Databases*, 2003. 77

[BCF$^+$05] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. *XQuery 1.0: An XML Query Language.* W3C, 2005. 54

[Ber03] Sacha Berger. Conception of a Graphical Interface for Querying XML. Diploma thesis, Institute for Informatics, LMU, Munich, 2003. 77, 81, 84

[BLLJ98] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs. *Cascading Style Sheets, level 2.* W3C, 1998. 1, 4, 9, 10, 14, 15, 20, 35, 74

[BM05] Dan Brickley and Libby Miller. FOAF Vocabulary Specification, 2005. 3, 72

[Bos05] Bert Bos. *Cascading Style Sheets Under Construction.* W3C, 2005. 1, 10, 15, 38

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. *Extensible Markup Language (XML) 1.0, 2nd Edition.* W3C, 2000. 1, 14, 30, 32, 35, 38, 61

[BW96] Bert Bos and Håkon Wium. *Cascading Style Sheets, level 1.* W3C, 1996. 10

[BWLJ96] Bert Bos, Håkon Wium, Chris Lilley, and Ian Jacobs. *Changes from CSS1.* W3C, 1996. 10

[CD99] James Clark and Steve DeRose. *XML Path Language (XPath) Version 1.0.* W3C, 1999. 48, 63

[Cla99] James Clark. *Associating Style Sheets with XML documents.* W3C, 1999. `http://www.w3.org/TR/xml-stylesheet/`. 10

[Cla01] James Clark. *Extensible Stylesheet Language (XSL) 1.0.* W3C, 2001. 1, 3, 19, 54

[DMO01] Steve DeRose, Eve Maler, and David Orchard. *XML Linking Language (XLink) Version 1.0.* W3C, 2001. 27

[ECM99] ECMA. *Standard ECMA-262, ECMAScript Language Specification*, 1999. 3, 5, 47

[HHW$^+$00] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. *Document Object Model (DOM) Level 2 Core Specification.* W3C, 2000. 3, 47

[Kep04] Stephan Kepser. A Simple Proof for the Turing-Completeness of XSLT and XQuery. In *Proceedings of Extreme Markup Languages 2004, Montreal, Canada (2nd–6th August 2004)*, 2004. 23

[LB99] Håkon Wium Lie and Bert Bos. *Cascading Style Sheets, designing for the Web.* Addison Wesley, 1999. 10

[Lie94] Håkon Wium Lie. *Cascading HTML style sheets – a proposal.* W3C, 1994. 9

[LS99] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF).* W3C, 1999. 3, 72

[PAA$^+$00] Steven Pemberton, Daniel Austin, Jonny Axelsson, Tantek Çelik, Doug Dominiak, Herman Elenbaas, Beth Epperson, Masayasu Ishikawa, Shin'ichi Matsui, Shane McCarron, Ann Navarro, Subramanian Peruvemba, Rob Relyea, Sebastian Schnitzenbaumer, and Peter Stark. *Extensible HyperText Markup Language (XHTML) 1.0.* W3C, 2000. 10, 62, 69, 70

[PI02] Steven Pemberton and Masayasu Ishikawa. *Link recognition for the XHTML Family.* W3C, 2002. 27

[SB04] Sebastian Schaffert and François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. of Extreme Markup Languages*, 2004. 3, 77

[Wie05] Christoph Wieser. Toward Extending Stylesheet Languages with Dynamic Document Rendering Features, 2005. `http://www.pms.ifi.lmu.de/publikationen/`. 1, 9, 29, 45, 50