Web Queries with Style: Rendering Xcerpt Programs with CSS^{NG}

François Bry and Christoph Wieser

University of Munich, Institute for Informatics
Oettingenstr. 67, 80538 München, Germany
`bry@pms.ifi.lmu.de, wieser@cip.ifi.lmu.de`
`http://www.pms.ifi.lmu.de`

Abstract. Styling and formatting of XML documents for various target media is often specified with the Cascading Style Sheet (CSS) language. An appealing feature of CSS is that it specifies formatting instructions using rather simple guarded 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. 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. This limited extension of CSS 3 turns out to make possible a rather advanced visualization of programs. This article (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} on Xcerpt query programs.

1 Introduction

CSS style sheets in the currently implemented version CSS 2.1 [6] have gained in importance, since the Web has become a mass medium. This language is used for a sophisticated rendering of semi-structured data especially expressed in XML [7]. CSS 3 [5], the newest version of CSS, is about to receive the status of a W3C recommendation, which is in fact a standard.

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 are not any longer sufficient. Sub-menus, for instance, which can be superimposed on a mouse click, are widespread on Web pages. 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 [12] interface to XML documents like ECMA Script [10] are used to obtain dynamic rendering features. In XHTML documents, for instance, scripts are rather often invoked in the context of an XHTML element by XHTML *intrinsic event* [1] 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

CSS^{NG} [16] is an extension of CSS 3. The strengths of this extension are the visualization of query languages like Xcerpt [14] as well as the visualization of RDF [13] graphs such as FOAF [8] definitions (see [16] for details on such Semantic Web applications). CSS^{NG} is a rather limited and conservative extension. Nonetheless CSS^{NG} makes it possible

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

The extension of CSS^{NG} allows for a *declarative* and, therefore, concise and quite simple specification of dynamic document rendering by comparison to query languages like XSLT [15] or scripting languages like ECMA Script [10].

Rendering, and especially layout issues, are so far no "Semantic Web issues". The authors believes that it is worth reconsidering this for the following reasons:

– Rendering and layout specification are best expressed in declarative formalisms. It is not by chance that CSS is a rule language.
– Rendering and layout specification could very well benefit from reasoning capabilities as offered by Semantic Web languages like OWL or the forthcoming Rule Interchange Format.
– Rendering and layout are a promising application field for contextual and semantic reasoning.

This paper is a first step in the above mentioned reconsideration. Admittedly, this step is a limited one. But first steps always can be seen as too limited.

2 CSS 3: A brief introduction

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. The following rule demonstrates a well-known *static* styling feature already introduced in CSS 1:

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

The left-hand *head* of the CSS rule, `a`, selects HTML anchors. The so-called *declaration* on the right-hand side assigns the styling parameter to XML elements selected by the head 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:

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

3 How CSS^{NG} extends CSS 3

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. The following subsections 3.1 to 3.3 briefly introduce novel static CSS^{NG} rules mainly aiming at visualizing XML markup. Finally Section 3.4 introduces the rule-based interface for dynamic document styling. Full details on how CSS^{NG} extends CSS 3 can be found in [16].

CSS^{NG} rules such as specified in a file can be linked in an XML document via a so-called processing instructions (PI) or in the header of an XHTML document. Note that CSS^{NG} extensions introduced for XML elements apply also to XHTML elements.

3.1 Markup Insertion

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.

CSS^{NG} extends these pseudo-elements of CSS 3. In addition to inserting plain text in CSS 3, the CSS^{NG} functions `element(NAME,ATTRIBUTES, VALUE)` and `attribute(NAME,VALUE)` provide also inserting XML elements and attributes before and after XML elements. The following example inserts tabs (see Fig. 4) inscribed with `element` before each element in an XML document (The CSS^{NG} function `element(NAME,VALUE)` has only two arguments, if there are no attributes.):

```
<a title="Tab">elem</a>
                               is inserted before each XML element by the rule
*::before { content: element("a",
                               attribute("title","Tab"),
                               "elem") }
```

Fig. 1. Markup Insertion (CSS^{NG}).

3.2 Markup Querying

CSS 3 provides the function `attr(X)` for querying the content of a known XML attribute `X` of an XML element. The name of an XML element and its XML attributes can not be queried. Implementing the markup visualization in Fig. 2 without generalized markup querying would mean one rule for every XML type like `bib`.

XML source

```
<bib>
  <book year="1994" id="42">
    <title>
      TCP/IP Illustrated
    </title>
    <author>
```

Presentation

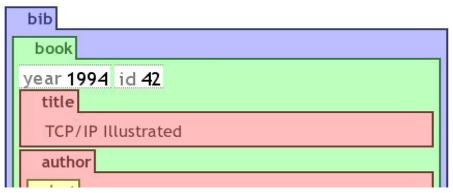


Fig. 2. XML document (left side) and rendering using CSS^{NG} (right side).

CSS^{NG} adds the function `element-name()` yielding the name of the currently selected XML element. Furthermore, one XML element can host several XML attributes. Therefore, CSS^{NG} offers *attribute rules* selecting XML attributes instead of XML elements. The CSS^{NG} functions `attribute-name()` and `attribute-value()` query XML attribute names and values in the context of a selected XML element. The example in Fig. 3 implements a tab in front of each XML element listing the XML element name and all of the XML elements' attributes including their values as shown in Fig. 2.

XML source (see Fig. 2)

```
... <book year="1994" id="42"> ... </book> ...
```

CSS^{NG} style sheet

```
*::before { content:
  element("span",       element("span", element-name())
                    * { element("span", attribute-name() " "
                                        attribute-value() )
                      } )
}
```

intermediate representation

```
... <span>
      <span>book<span>
      <span>year 1994</span>
      <span>id 42</span>
    </span>
    <book year="1994" id="42"> ... </book> ...
```

Fig. 3. Generation of tabs. The presentation in Fig. 2 is obtained by rendering the intermediate representation using further CSS 3 means.

3.3 Depth-dependent Styling

Styling depending on breadth is planned in CSS 3 [5]. Tables, for instance, can be styled using alternating background colors for each line. CSS^{NG} additionally offers styling depending on the depth of an XML element in an XML document: The pseudo-class `:nth-descendant(an+b)` restricts selections to XML elements having $an + b$ ancestors.

Fig. 4 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. Thanks to its depth-dependent 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 using the CSS 3 style sheet on the lower right side of Fig. 4. Such a styling would also be useful for applications such as the visualization of threads in a discussion forum.

CSS^{NG}

```
*: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

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

* * * * * * *          { background-color: A; }
* * * * * * * *        { background-color: B; }
...
```

Presentation using CSS 3

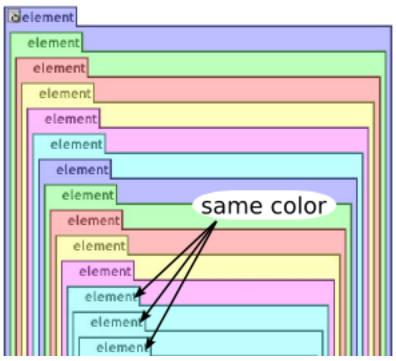


Fig. 4. Comparing Depth-dependent Styling using CSS^{NG} and CSS 3.

3.4 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 Section 6: when the mouse cursor moves away, the cursor does no longer hover over the selected XML element, and its tab would be automatically unfolded.

CSS^{NG} introduces dynamic pseudo-classes for *all* HTML intrinsic events [1] such as `onclick` or `onkeypress` (see [16] for sample applications). Instead of using HTML intrinsic event attributes like for scripting languages, CSS^{NG} allows a standalone specification of dynamic styling in separate CSS^{NG} files that can be

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

Fig. 5. Dynamic Styling (CSS^{NG}).

applied for multiple documents. The following example in Fig. 5 shows a rather simple dynamic CSS^{NG} rule.

The rule in Fig. 5 implements an adaptive hyperlink. After 10 clicks on the hyperlink the background color changes to green meaning that the hyperlink on the Web page is frequented by the user.

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 different background colors.

Similar extensions using HTML intrinsic events have been already proposed by the W3C (see Section 7). The following paragraphs introduce to novel capabilities of CSS^{NG}:

Recurrence Patterns. All CSS^{NG} dynamic pseudo classes support *recurrence patterns*, `an+b`, as parameters. For instance the CSS^{NG} selector `*:onclick(3n+1)` detects the first, the fourth, the seventh, etc. click on an arbitrary XML element. More generally, a CSS^{NG} selector fires, if $an + b$ events occurred before.

On one hand such recurrence patterns allow to reuse CSS^{NG} rules for folding and unfolding as demonstrated in the following paragraph. 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).

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.

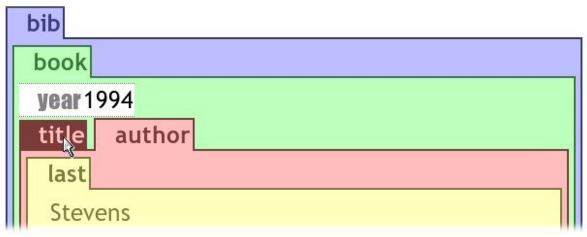


Fig. 6. Folded visualization of an XML element `title`. The corresponding unfolded example is shown in Fig. 2.

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 simple selector.

The following example (see Fig. 7) implements *alternating folding and unfolding* for the visualization of arbitrary (simple selector `*`) XML elements (see Fig. 6). 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. 2):

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

Fig. 7. Combined dynamic styling in CSS^{NG} (rendering in Fig. 6).

In the example above, 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, while its tab stays visible.

Structure-Independent Styling. A static CSS 3 styling rule is applied to all XML elements matching its selector. A dynamic CSS 3 styling rule is applied only to XML elements being in the context of an input device such as an XML element laying under the mouse cursor. CSS^{NG} abolishes this restriction and allows (novel) so-called *monorama* and *panorama* selections as demonstrated in Fig. 12. The `Author` element on the left side is highlighted, while the mouse cursor is hovering over the `Author` element on the right side (see Fig. 8).

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

Fig. 8. Highlighting of Xcerpt variables.

The CSS 3 rule in line 1 defines the standard background black for XML `Author` elements. In line 2 the CSS^{NG} combinator `?`, called *if*, is applied as follows: If an XML `Author` element is hovered in an XML document, set the background color of all XML `Author` elements to white.

4 System Architecture of the Prototype

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.

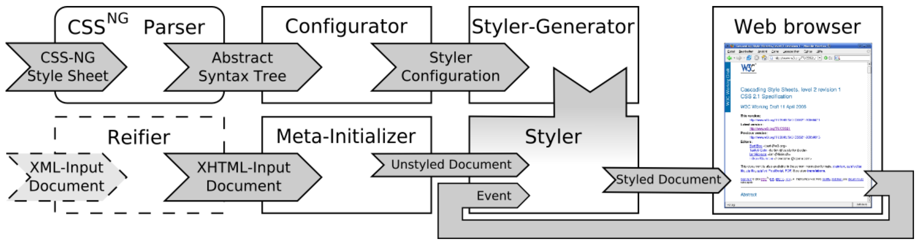


Fig. 9. CSS^{NG} styling of an X(HT)ML document and rendering.

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[1] or MS Internet Explorer[2].

The upper row in Fig. 9 manages the compilation of an input 'CSS^{NG} Style Sheet' to a 'Styler' (see Section 4.1 for details). 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 a Web browser.

4.1 Styler Generation

On the upper row (see Fig. 9), the 'CSS^{NG} style sheet' is parsed resulting in an 'Abstract Syntax Tree' (AST), which is based on a slightly extended Grammar of CSS 2.1 [6]. In a next step the 'Configurator' condenses the 'AST' to a human readable 'Styler Configuration'. Finally, this configuration is implemented by a 'Styler-Generator' yielding the 'Styler'.

4.2 Input Preprocessing

Since many of the current Web browsers do not offer standard event APIs for XML documents, we transform 'XML Input Documents' to get 'XHTML Input

[1] `http://www.mozilla.com/firefox/`
[2] `http://www.microsoft.com/windows/ie/default.mspx`

Documents'[3] as demonstrated in the following table (see Fig. 10). The XML markup on the left side is expressed using XHTML markup exclusively on the right side. (**XHTML input documents** can be initialized directly without reification.)

Source XML:

```
<workshop>
  PPSWR 2006
</workshop>
```

XML as XHTML:

```
<div>
  <span class="element">workshop</span>
  PPSWR 2006
</div>
```

Fig. 10. Expressing XML (left side) using XHTML (right side).

The 'Meta-Initializer' installs listeners as well as histories for relevant XHTML elements. 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.

5 Proof-of-Concept Implementation

The main principles of the proof-of-concept implementation are

– drawing on Web standards for
– gaining platform independency and
– reducing implementation effort.

Therefore all data formats and transformations (see Fig. 9) except **CSS^{NG} Parser** are based on W3C standards. Since the CSS 2.1 grammar [6] is specified in extended Yacc and Flex syntax, the Yacc parser and the Flex lexical scanner are used to transform **CSS^{NG} style sheets** into XML format (there are no W3C standards for this kind of transformation). All other transformations are implemented as XSL Transformations [15].

The **Styler** is the heart of the system. It processes all XHTML elements in the document tree of an **(Un)styled Document** recursively. Each XHTML element passes through one test for each CSS^{NG} rule in a CSS^{NG} style sheet. If a test succeeds, the XHTML `style` attribute of the current XHTML element is modified. The tests are implemented in XPath [9]. Since tests are executed from the perspective of each XML element, CSS^{NG} selectors need to be translated to XPath selecting XML elements in reverse direction as demonstrated in the following example (see Fig. 11):

[3] This transformation is called reification.

CSS^{NG}

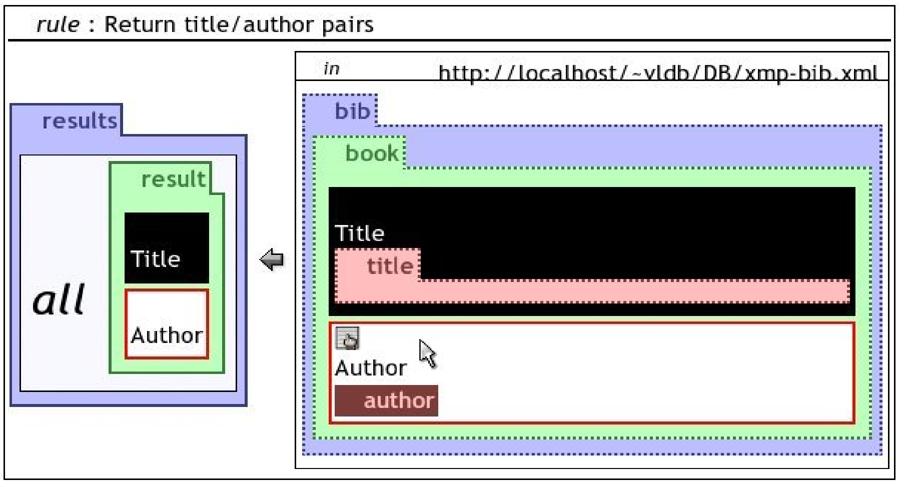
```
div :onclick(2n+1) + *
```

XPath

```
self:: * /preceding-sibling:: div [
   span[@class='onclick'] mod 2  =  1 mod 2 ]
```

Fig. 11. Translation of CSS Selectos in XPath (CSS^{NG}).

6 Proof-of-Concept Application: Rendering of Xcerpt Programs



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

Fig. 12. Query Visualization as Textual Query Rendering. All occurrences (*panorama*) of `Author` are highlighted by white background color.

CSS^{NG} can be applied to implement query visualization as textual query rendering as shown in Fig. 12. Here, the viewer of the visual interface visXcerpt [3] [4] for the XML query and transformation language Xcerpt [14] is re-implemented by only a few CSS^{NG} rules (131 lines of code). It is worth stressing that

- the original implementation of visXcerpt (ECMA Script, XSLT, and Python) is much longer (2060 lines of code) 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 example above demonstrates highlighting of Xcerpt variables having the same name: All occurrences of the Xcerpt variable `Author` are highlighted because the mouse cursor is hovering over one occurrence, while Xcerpt variables named `Title` stay black. We refer to [16] for further examples of CSS^{NG} applications.

7 Related Work

The so-called Behavioral Extension of CSS [11], a derivative of Action Sheets [2], is a proposal for extending CSS toward dynamic styling features. The main idea is to separate scripts from content using the selector mechanism of CSS. The approach draws on scripting languages for implementing dynamic styling. The Behavioral Extension of CSS specifies events in the declaration of the rule. Rather simple dynamic tree patterns of CSS^{NG}, as demonstrated in Section 3.4, can only be simulated in the Behavioral Extension of CSS using rather complicated scripts. The following use case (see Fig. 13) 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; }
```

Fig. 13. Comparison of 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 [16] for more use cases).

8 Conclusion

In this article we presented how CSS^{NG} extends CSS 3 toward dynamic styling features and markup visualization. As shown in Section 6, CSS^{NG} allows program visualization of Xcerpt programs as textual program rendering. However we believe that CSS^{NG} allows generic visualizations of programming languages. Such visualizations realized as textual document rendering (see Fig. 12) 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 our knowledge further approaches of visual languages never allow visual and textual programming as well.

9 Acknowledgments

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net).

References

1. S. Adler, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles. *HTML 4.01.* W3C, 1999.
2. V. Apparao, B. Eich, R. Guha, and N. Ranjan. *Action Sheets: A Modular Way of Defining Behavior for XML and HTML.* W3C, 1998.
3. S. Berger. Conception of a Graphical Interface for Querying XML. Diploma thesis, Institute for Informatics, LMU, Munich, 2003.
4. S. Berger, F. Bry, S. Schaffert, and C. 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.
5. B. Bos. *Cascading Style Sheets Under Construction.* W3C, 2005.
6. B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. *Cascading Style Sheets.* W3C, 1998.
7. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. *Extensible Markup Language (XML) 1.0, 2nd Edition.* W3C, 2000.
8. D. Brickley and L. Miller. FOAF Vocabulary Specification, 2005.
9. J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* W3C, 1999.
10. ECMA. *Standard ECMA-262, ECMAScript Language Specification*, 1999.
11. V. A. et al. *Behavioral Extensions to CSS.* W3C, 1999.
12. A. L. Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. *Document Object Model (DOM) Level 2 Core Specification.* W3C, 2000.
13. O. Lassila and R. R. Swick. *Resource Description Framework (RDF).* W3C, 1999.
14. S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. of Extreme Markup Languages*, 2004.
15. W3C. *Extensible Stylesheet Language (XSL) 1.0*, 2001.
16. C. Wieser. CSS^{NG}: An Extension of the Cascading Styles Sheets Language (CSS) with Dynamic Document Rendering Features. Diploma thesis, Institute for Informatics, LMU, Munich, 2006. `http://www.pms.ifi.lmu.de/publikationen/`.