Visual Languages: A Matter of Style

Sacha Berger François Bry Tim Furche [1]

Institute for Informatics
University of Munich, Germany

Christoph Wieser [2]

Salzburg Research, Austria

Abstract

This articles submits the thesis that visual data modeling and programming languages are conveniently conceived as rendering, or 'styling', of conventional, textual languages. Styling has become a widespread technique with the advent of the Web and of the markup language XML. With XML, application data can be modeled after the application logic regardless of the intended rendering. Rendering of XML documents is specified using style sheet languages like CSS. Provided the styling language offers the necessary functionalities, style sheets can similarly specify a visual rendering of modeling and programming languages. The advantages of the approach are manifold: visualization is achieved in a systematic manner, i.e. the same visualization paradigms can be used for several languages; visual languages necessarily come along with textual counterparts; visual languages are much easier to develop than in ad hoc manners.
This article first introduces rather limited extensions to the style sheet language CSS that make it amenable to render data modeling and programming languages as visual languages. Then, it demonstrates the approach on a use case, the experimental logic-based Web query and transformation language Xcerpt. Finally, it is argued that the approach is especially amenable to logic-based languages.

Keywords: Visual Programming Language, Rendering, Styling

1 Introduction

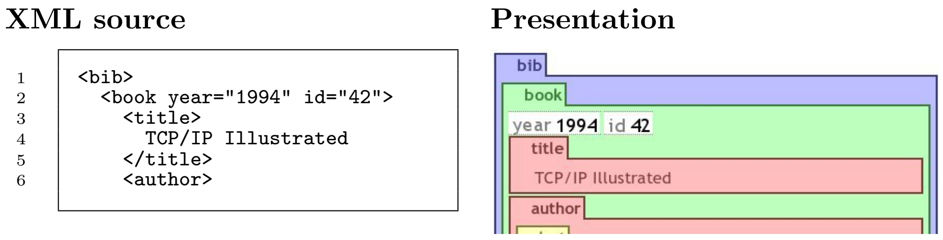
Visual data modeling and programming languages are conveniently conceived as rendering, or 'styling', of conventional, textual languages. Styling has become a widespread technique with the advent of the Web and of the markup language XML.

With XML, application data can be modeled after the application logic regardless of the intended rendering. Rendering of XML documents is specified using style sheet languages like CSS. Provided the styling language offers the necessary functionalities, style sheets can similarly specify a visual rendering of modeling and

[1] Email: {sacha.berger,francois.bry,tim.furche}@ifi.lmu.de
[2] Email: christoph.wieser@salzburgresearch.at

programming languages. The advantages of the approach are manifold: visualization is achieved in a systematic manner, i.e. the same visualization paradigms can be used for several languages; visual languages necessarily come along with textual counterparts; visual languages are much easier to develop than in ad hoc manners.

This article first introduces rather limited extensions to the style sheet language CSS that make it amenable to render data modeling and programming languages as visual languages. Then, it demonstrates the approach on a use case, the experimental logic-based Web query and transformation language Xcerpt. Finally, it is argued that the approach is especially amenable to logic-based languages.



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

Fig. 1. XML document (left side) and rendering using CSS^{NG} (right side).

CSS is chosen in this study because of its admittedly limited expressive power: if limited extensions to CSS make it possible to conceive visual languages as styling of textual languages, than the same will hold of more involved styling languages such as XSL-FO.

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.

Even though CSS^{NG} is a limited and conservative extension of CSS, 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 in comparison to query languages like XSLT [15] or scripting languages like ECMA Script [11].

2 CSS in a Nutshell

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; }
```

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 2.1 to 2.3 briefly introduce novel static CSS^{NG} rules mainly aiming at visualizing XML markup. Finally Section 2.4 introduces the rule-based interface for dynamic document styling. Full details on how CSS^{NG} extends CSS 3 can be found in [13].

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.

2.1 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. 5) 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.):

```
  *::before { content: element("a",
                         attribute("title","Tab"),
                         "elem") }
```

Fig. 2. Markup Insertion (CSS^{NG}) — `<a title="Tab">elem</a>` is inserted before each XML element by the rule.

2.2 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. 10

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

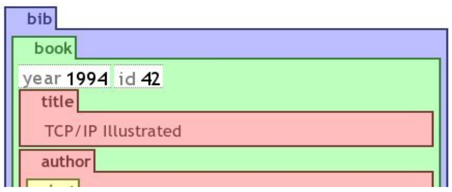


Fig. 3. XML document (left side) and rendering using CSS^{NG} (right side).

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. 4 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. 10.

XML source (see Fig. 10)

```
... <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. 4. Generation of tabs. The presentation in Fig. 10 is obtained by rendering the intermediate representation using further CSS 3 means.

2.3 Depth-dependent Styling

Styling depending on breadth is planned in CSS 3 [6]. 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. 5 demonstrates the visualization of a highly nested XML document with colors repeating on every third 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 three 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. 5 using the CSS 3 style sheet on the lower right side of Fig. 5. Such a styling would also be useful for applications such as the visualization of threads in a discussion forum.

CSS^{NG}

```
*:nth-descendant(3n+1) { background-color: A; }
*:nth-descendant(3n+2) { background-color: B; }
*:nth-descendant(3n+3) { background-color: C; }
```

CSS 3

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

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

Presentation using CSS 3

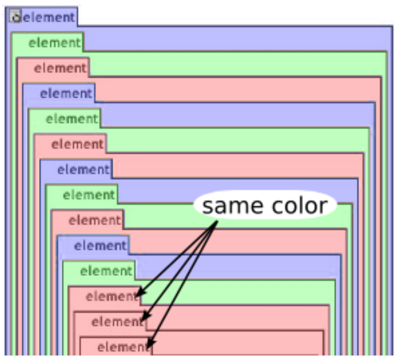


Fig. 5. Comparing Depth-dependent Styling using CSS^{NG} and CSS 3.

2.4 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 7: 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 [13] 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 applied for multiple documents. The following example in Fig. 6 shows a rather simple dynamic CSS^{NG} rule.

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

Fig. 6. Dynamic Styling of an adaptive hyperlink (CSS^{NG}).

The rule in Fig. 6 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 [8]. The following paragraphs introduce to novel capabilities of CSS^{NG}:

2.4.1 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).

2.4.2 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.

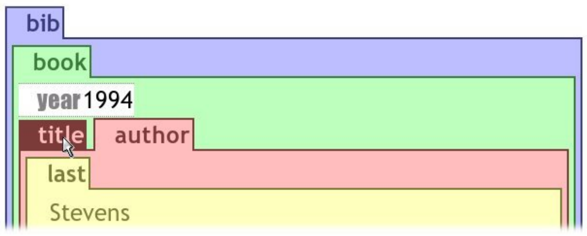


Fig. 7. Folded visualization of an XML element `title`. The corresponding unfolded example is shown in Fig. 10.

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. 8) implements *alternating folding and unfolding* for the visualization of arbitrary (simple selector `*`) XML elements (see Fig. 7). 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. 10):

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

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

selector `*` disappears, if a mouse click was performed on its preceding sibling XML element, while its tab stays visible.

2.4.3 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. **??**. 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. 9).

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

Fig. 9. 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.

A proof-of-concept prototypical implementation of CSS^{NG} was implemented as part of a diploma thesis [13] and presented [8].

3 Styling of Logic Languages

The approach described in thew previous section to conceive a visual language as a rendering, or styling, of a textual language seems for the following two reasons especially convenient for logic languages:

- Logic languages are declarative, i.e. they focus on both the structural and conceptual organization of the data.
- Logic languages come in families that share traits, like e.g. modal languages, rule-based languages, logic programming languages, frame logic languages. With the approach proposed, "visualizations" can be rather easily developed and applied to various languages of a same language family.

It is the firm belief of the authors that the approach proposed in this article has the potential to boost the development and testing of visual languages, especially of visual logic languages.

4 visXcerpt — the Visual Twin Sibling of Xcerpt

As an example of the visualization of a textual language using the presented approach and CSS^{NG}, the Web query and transformation language Xcerpt[11] and its visual counterpart visXcerpt[2] are presented. Xcerpt is a rule based deductive language. As a textual language, it comes in two syntax flavours — an abbreviated syntax and an XML syntax. Rules consist of a head, also called construct pattern and a body consisting of logically connected query patterns. Query and construction share values my means of shared variables, rules query each other heads employing forward or backward chaining. Construct patterns may contain special grouping constructs to collect multiple variable bindings in one result, queries may consist of incomplete query patterns with incompleteness in breadth and/or depth and/or order, reflecting the incertitude about size and structure of documents on the web. Patterns are hence like *'examples'* of web data searched fr in given documents.

The central part of visXcerpt, the visual rendering of Xcerpt, is the visualization of Web data, of XML documents. As Xcerpt itself comes in XML syntax, half the job is done by visualizing XML. Further aspects, like partiality, grouping constructs and variables are then added to term visualization to get a full featured visualization of query and construct patterns. Rules are then just represented as horizontal aligned head and body, related by an arrow.

Term Visualisation

Web data and patterns are considered to have term like structure. Terms are rendered as boxes with their name as a tab on the top, the box contains all tabbed boxes of the subterms in the order they occur. The rendering is conceived to be suitable for most web browsers, as they are a wide spread technology with high adaptability to various screen sizes and resolutions. Order is given by a left-to-right and top-to-bottom flow layout, but the layout directions should be adapted to local writing habits of the users culture. Width is given by the width of the display or browser employed. Nested boxes are further distinguished using colors, hence colors represent nesting depth. To be able to make a reasonable selection of well assorted, distinguishable and pleasant color themes, colors of upper levels are recycled for deeper nestings.

Graph Visualization

On the Web, graph structures also need to be represented, e.g. RDF[10] data representing graph sharped structures or hyperlink structure. In textual representations of graph structures, references are used along a spanning tree of the graph. The presented approach of visualizing such graph structures is to model the references as hyperlinks in a kind of browser. This way, even very large graph structures can be represented and access to any references item is achieved by user interaction with constant complexity – a click on a hyperlink. While browsers often provide some means of navigating *back* along edges represented by hyperlinks, it is arguably useful to explicitly give hyperlinks for reverse traversal of edges, as hence the user is not restricted in his backward movement along edges he just visited.

Information focusing

For large documents, it is of vital necessity to give users the ability to hide temporarily unneeded information or to focus on relevant data. This is achieved by means of folding in or out terms behind their name tagged tabs. While elements are aligned vertically, tabs are first aligned left-to-right and then vertically, saving even more space. The concept is strongly inspired by tree browser visualization as e.g. seen on the well known Windows file browser.

At this level, pure static visualization already starts to merge with user interaction. A visualization, is indeed arguably much more useful, with adequate support of user interaction, especially of editing.

Textual Xcerpt Program, and **visXcerpt rendering of it.**

```
CONSTRUCT
 results[
   all result[
     var Title,
     var Author
 ]  ]
FROM
 in(resource="file:procs04.xml")
 proceedings04[[
   papers[[
     paper[[
       var Title as title[[]] ,
       var Author as author[[]]
     ]]
   ]]
 ]]
END
```

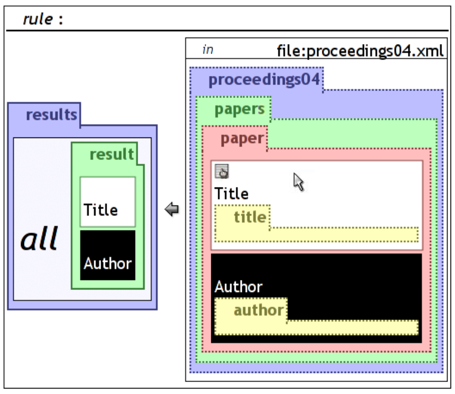


Fig. 10. A single rule Xcerpt program (in abbreviated textual syntax) along with its visXcerpt rendering — the query part exploits a partial pattern (indicated by dotted lines in the visualisation) to search for papers in a proceedings database, constructing title/author pairs all grouped in a list of results. All *Title* variables are highlighted as the mouse is hovering above one of them in visXcerpt.

A Special Purpose Editor Model

For textual languages, copy-and-paste and text typing based editors are wide spread. Central to textual editing, is a cursor concept, that usually is a separator of the one dimensional program. For the presented visual approach, a separator seemed not intuitive, hence a context metaphor is used for editing: each box is a context, it is possible to cut, copy or delete it with or without its sub boxes, it is possible to paste the content of the cut/copy buffer into, before, after or around a context and hence term. The rich copy and paste model is accompanied by a template concept, giving access to all program constructs and possibly example terms or structures that can be altered, reduced or extended.

5 A realisation using CSS and CSS^{NG}

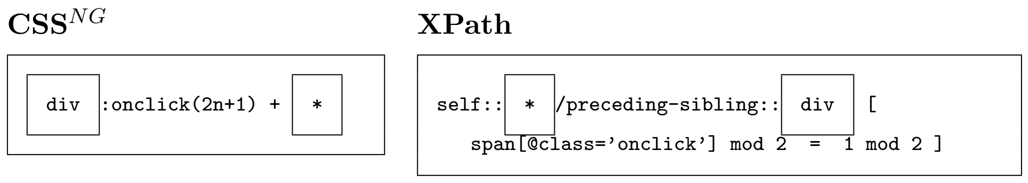
5.1 Principles

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 except **CSS^{NG} Parser** are based on W3C standards. Since the CSS 2.1 grammar [7] 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 [12].

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):



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

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

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

6 Outlook and Conclusion

The presented approach — obtaining a visual language by mere rendering or styling of a textual language — has been explored with the textual query language Xcerpt. To the largest extend, this has been achieved using standard CSS, for the most salient features however an extension of CSS has been conceived.

6.1 Conclusion

visXcerpt has been prototypically implemented and successfully applied for the presentation of Xcerpt[5][4], widely easing the comprehension of the concepts of Xcerpt. visXcerpt's editor model turned out convenient for a wide scale of Xcerpt programming tasks from the area of HTML content extraction, creation and wrapping, over XML data transformation to Semantic Web and hybrid Web and Semantic Web reasoning[3].

CSS^{NG} as an extension of CSS turned out to be easily realizable without heavy computational overhead compared to CSS3 and CSS3. It proved itself to be not only a tool for the implementation of visXcerpt, but especially for sophisticated visualization of XML data or arbitrary term structures with easily realizable domain specific behavior.

The approach of conceiving a visual language based on a textual back-end turned out convenient in both cases, for the programmer using the language as well as for the creator of the visual language — creating a visual language as a rendering of a textual one was reasonably easy, and programmers using it where pleased to be able to switch between textual and visual representation, hence combining the best editing features of visual and textual world.

To the best of the knowledge of the authors similar generic approaches of developing visual languages as mere rendering using CSS and extensions have not been proposed so far.

6.2 Outlook

Further interesting research in the area of Xcerpt/visXcerpt is to investigate about type support, not only in the textual language for checking and validation of programs, but also in the editing process. This could help novice users to by just providing editing features that lead from one valid program to another, as well as providing a type based template approach over the example based approach.

In the area of generic visualization of textual languages, it is needed to systematically investigate further features/functionalities that would be desirable for visual languages and what existing styling languages would be a convenient basis for adding these features.

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] S. Berger. Conception of a Graphical Interface for Querying XML. Diploma thesis, Institute for Informatics, LMU, Munich, 2003.

[3] S. Berger, F. Bry, O. Bolzer, T. Furche, S. Schaffert, and C. Wieser. Querying the standard and semantic web using xcerpt and visxcerpt. In *Proceedings of European Semantic Web Conference, Heraklion, Crete, Greece (29th May–1st June 2005)*, 2005.

[4] S. Berger, F. Bry, and T. Furche. Xcerpt and visxcerpt: Integrating web querying. In *Proceedings of Programming Language Technologies for XML, Charleston, South Carolina (14th January 2006)*, 2006.

[5] 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.

[6] B. Bos. *Cascading Style Sheets Under Construction.* W3C, 2005.

[7] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. *Cascading Style Sheets.* W3C, 1998.

[8] F. Bry and C. Wieser. Web queries with style: Rendering xcerpt programs with css-ng. In *Proc. of 4th Workshop on Principles and Practice of Semantic Web Reasoning*, 2006.

[9] J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* W3C, 1999.

[10] O. Lassila and R. R. Swick. *Resource Description Framework (RDF).* W3C, 1999.

[11] S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. of Extreme Markup Languages*, 2004.

[12] W3C. *Extensible Stylesheet Language (XSL) 1.0*, 2001.

[13] 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/.