Journal of Universal Computer Science, vol. 4, no. 10 (1998), 100-200
submitted: 1/9/98, accepted: 25/9/98, appeared: 28/10/98 © Springer Pub. Co.

PSL: An Alternate Approach to Style Sheet Languages for the World Wide Web[1]

Philip M. Marden, Jr. and Ethan V. Munson
Department of Electrical Engineering and Computer Science
University of Wisconsin–Milwaukee
Milwaukee, WI 53201, USA
{phil,munson}@cs.uwm.edu
http://www.cs.uwm.edu/~multimedia

Abstract: Style sheets, which are used to specify the appearance of documents, are rapidly growing in their importance for the World Wide Web. Cascading Style Sheets are now in widespread use and work on a future Web standard, the Extensible Style Language (XSL), is proceeding at a rapid pace. In this paper, we show how a different style sheet language, PSL, represents an attractive midpoint between CSS and XSL in complexity and power. PSL is based on general language design principles that give it simple syntax, easily-described semantics, and considerable expressive power. Our testbed MPMosaic uses Proteus, a portable style sheet system, to support PSL.

1 Introduction

Style sheets are used to specify how a document should be presented to users. For Web documents, style sheets can be used to specify fonts, colors, borders, and overall layout, as well as other style properties. For example, a style sheet might indicate that the document's title should appear centered on the top of the first page in a 36 point Helvetica font.

Style sheets can be used for a variety of purposes:

- they can enforce consistent style across a large collection of similar documents;
- they provide the ability to tailor a document's presentation to the needs of the end user or the capabilities of the presentation device without altering the document itself; and
- for limited document representations like HTML, they can describe style effects that go beyond the style information implicit in a document's tag structure.

In the Web community, the usefulness of style sheets for HTML [Wor98c] documents has become apparent because style sheets give document designers a level of control over the appearance of their documents that HTML alone cannot provide. Several style sheet languages are being developed including Cascading Style Sheets (CSS) [Wor96, Wor98a] and the Extensible Style Language (XSL) [Wor97]. CSS has simple syntax, simple semantics, and limited power and is primarily designed to support HTML. XSL is a much larger and more powerful

[1] This is an extended version of a paper presented at the WebNet '98 conference in Orlando, Florida. The paper has received a "Top Full Paper Award".

language with significant syntactic and semantic complexity. It is part of a larger effort to move the Web beyond HTML's limitations that is also producing the Extensible Markup Language (XML) [Wor98b] for document representation.

This paper describes our study of style sheet languages for HTML documents. We believe that in an ideal world, a style sheet language would have

– simple syntax so that the language is easy to read and write;
– consistent and easily-described semantics so that style sheets are easy to understand and straightforward to explain to others; and
– the ability to specify a wide variety of useful presentations for authors and end-users.

We seek to find the best way to balance these sometimes-conflicting design goals for style sheet languages. We have also been exploring ways that style sheets can be used to enhance the user interface of Web browsers. Style sheets have considerable promise for creating improved browser interfaces, because they can be changed without touching the browser's source code or the HTML of individual documents.

Our research has focused on the application of an alternate style sheet language, the Presentation Specification Language (PSL) [Mun95], to HTML documents. We have developed an experimental Web browser, MPMosaic [MM98], that uses PSL style sheets to control the appearance of HTML documents. Our work with PSL shows that it is possible to create a style sheet language that is at an attractive midpoint between CSS and XSL in both complexity and expressive power by basing the language on very general design principles in conjunction with simple syntax and powerful semantics.

The remainder of the paper is organized as follows. Section 2 describes some other style sheet languages. Section 3 provides an introduction to PSL. Section 4 discusses the advantages of PSL's use of general language facilities in comparison to Cascading Style Sheets. Section 5 describes our experiences in using PSL in the MPMosaic system, and the final section presents our conclusions.

2 CSS, XSL, and DSSSL

Style sheets have been used for some time in the structured document community, where a central premise has been that the appearance of documents should be specified separately from their structure and content.

SGML and DSSSL are the respective ISO standards for structure and presentation. SGML [Gol86] is a meta-language for defining markup languages, such as HTML, that are used to describe the structure and content of individual documents. The elements (or tags) of each SGML-based markup language are specified by a Document Type Definition (DTD) written in this meta-language.

DSSSL [ISO94], SGML's companion standard for style sheets, is a powerful, Turing-complete style language based on the Scheme programming language. DSSSL is a very complex language both syntactically and semantically and has not yet seen widespread implementation or use. DSSSL is clearly very powerful, but the lack of working implementations makes comparison difficult. DSSSL could be used to describe HTML documents, but has usability problems for interactive systems that have prevented its inclusion in any widely-used editing or browsing software.

The World Wide Web Consortium (W3C) has developed a language called Cascading Style Sheets (CSS) which has become the standard style sheet language for the Web. CSS Level 1 (CSS1) [Wor96], the first version of CSS, is now supported by the major browser vendors, and a more extensive version, CSS Level 2 (CSS2) [Wor98a], has been developed. Work has also begun on another proposed style sheet standard, the Extensible Style Language (XSL) [Wor97], which is derived from DSSSL and appears to support most of DSSSL's features.

CSS is at the other end of the design spectrum from DSSSL and XSL. CSS has a very simple syntax, but limited expressive power. CSS has been explicitly designed to be written by non-programmers and often provides nice, intuitive ways to express style ideas. However, this intuitiveness comes at the cost of consistency. The semantics of CSS are not consistent across language features, and we believe the language will grow increasingly difficult to reason about as new features are added to it.

Due to length restrictions, we will not describe the features of the CSS, XSL, and DSSSL languages in any detail. We urge readers to review the specifications for CSS1 [Wor96] and CSS2 [Wor98a], if they are not familiar with them. For general information on style sheets, the Web Consortium has a number of resources at http://www.w3.org/style.

3 The PSL Language

PSL is the style sheet language for Proteus [GHM92], a portable presentation system for multimedia documents. PSL is a declarative language that was designed to be independent of any particular medium. PSL's syntax and semantics were designed to be easy to use, and the language has very few special cases. An example style sheet is shown in Figure 1.

In order to understand how PSL (or any style sheet language for HTML) works, an important fact about the structure of HTML documents must be understood. Structured markup languages like HTML annotate the document contents with tags, or markup. The tags break up the document into elements such as paragraphs, lists, headings, and emphasized text. Because elements can be nested inside other elements, they can be thought of as the nodes of a tree that represents the entire document, which we call the *document tree* (see Figure 6). All existing style sheet languages take advantage of this tree structure.

PSL provides three presentation services: property propagation, tree elaboration, and box layout. In Proteus, these services operate on a *presentation tree*, which is a copy of the document tree. The presentation process does not alter the document tree. Using presentation trees allows multiple styles of the same document to be shown simultaneously.

- *Property propagation* assigns values to the formatting properties of each element, or node. Section 3.1 describes the syntax for assigning values to properties.
- *Tree elaboration* allows style sheet authors to add content to the presentation by adding elements to the presentation tree (section 3.2).
- *Box layout* is a constraint-based layout system (section 3.3). Elements can can be placed on the screen in a different order from their order in a tree traversal. PSL distinguishes between the specified and actual size of an element.

```
MEDIUM mosaic;
PRESENTATION links FOR html;

ELABORATIONS {
    linebreak : Markup ("<BR>") {
        visible = Yes;
    }
    arrow : Markup ("<IMG src=arrow-grey.gif>") {
        visible = Yes;
    }
    url : Content (getAttribute(creator, "href")) {
        visible = Yes;
        fontSize = 12;
    }
}

DEFAULT {
        lineHeight = Self.fontSize * 1.5;
}

RULES {
    HTML {
        visible  = No;
        fgColor  = "black";
        fontSize = 14;
    }
    A {
        if ( getAttribute(self, "href") != "" ) then
            visible  = Yes;
            fgColor  = "blue";
            underlineNumber = 1;
            createRight (arrow, url, linebreak);
        endif
    }
}
```

Figure 1: A PSL style sheet. This style sheet specifies that only links and their destinations should be displayed on the screen. All other content is elided.

```
<HTML>
  <HEAD> <TITLE> A Sample HTML document </TITLE>
  <BODY>
    <H1> <A href="http://www.w3.org/Style"> Style sheets </A> </H1>
    <UL>
      <LI> Give users control of formatting
      <LI> Support the accessibility goals of the
        <A href="http://www.w3.org/WAI">
           Web Accessibility Initiative </A>
      <LI> Support multiple presentations
    </UL>
```

Figure 2: A sample HTML document.

Figure 3: This figure shows the presentation produced when the style sheet in Figure 1 is applied to the document in Figure 2. The style sheet specifies that only the links and their destinations are displayed. (Captured from MPMosaic)

3.1 Properties and Rules

In the `RULES` section of the style sheet, the presentation of an element can be specified by following the element's markup name (also referred to as node type) with a list of property rules:

```
P {
    fontFamily = "times";
    fontSize = 14;
    lineHeight = Self.fontSize * 1.5;
    indent = LeftSib.indent + 10;
}
```

This PSL fragment specifies several properties for P elements: the font for text should be 14 point Times, the line height should be 21 points (1.5 times the font size of this element), and the first line of text should be indented 10 points more than this element's left sibling. Each property has a data type, which is either boolean, string, real, or predefined enumeration. Property rules have the form:

```
<property> = <expression> ;
```

The right hand side of the property rule can contain any expression whose data type is the same as the type of the property named on the left hand side. Expressions can be constructed using a variety of operations and functions common to general-purpose programming languages including standard arithmetic, comparison, and boolean operators, common mathematical functions (such as min, max, and round) and trigonometric functions.

Property values can be constrained to depend on the property values of other nodes (elements) by using the property access expression, for which the syntax is:

```
<node expression> . <property name>
```

The value of a property access expression is computed by finding the node specified by the expression on the left hand side of the dot and getting the value of the named property for that node.

There are several tree traversal functions that return nodes, any of which can appear in the left hand side of a property access expression. Some of these functions return immediate neighbors (`Parent`, `LeftSib`, `RightSib`, `FirstChild`, `LastChild`, `NthChild`), while others return nodes that may be more distant (`Root`, `AncestorOfType` for obtaining ancestors, and `Creator` which returns the generating node of an elaborated node). There is also a `Self` function which returns the defining node, the node for which the property rule was defined. Collectively, these functions are designed to allow the specification of constraints between the defining node and every other node in the tree. Distant nodes in the tree can be specified through function composition, as in `FirstChild(LeftSib(Parent)).fontSize` which specifies the `fontSize` property of a "cousin" node.

3.1.1 Default Rules

PSL style sheets can have a `DEFAULT` section which defines default rules that are used when there are no node-specific rules for a property. The `DEFAULT` section contains a rule list having the same syntax and semantics as the rule lists for specific node types.

3.1.2 Order of Evaluation

Every property in every node is assigned a value. The value of a property is determined by the first source below that returns a valid value for that property:

1. Node-specific rule.
2. `DEFAULT` section rule.
3. Inherited value.
4. Medium specified value. (CSS calls this the initial value.)

Invalid values occur when a rule fails, namely its expression cannot be computed for some reason. This could occur because of an arithmetic error (such as division by zero), but most commonly it results from a tree navigation error. For instance, a node that is a first child has no left sibling, so if the `LeftSib` function is invoked on a first child, the function fails.

All properties can be inherited during the third step in the order of evaluation. This is equivalent to a rule of the form:

```
<property> = Parent . <property> ;
```

3.2 Tree Elaboration

Tree elaboration allows style sheet authors to add content to a document's presentation by adding nodes to the presentation tree. For example, tree elaboration is used to precede list items with numbers or "bullets," and is used to create borders around elements.

Tree elaboration is specified in two parts: node declarations and creation commands. The `ELABORATIONS` section of the style sheet is used to declare the node types that can be generated. These declarations specify a primitive type: *Content* for text, *Markup* for tags, and *Graphic* for graphical objects like lines, rectangles, and circles. Declarations also require an initialization argument, which is an expression that describes the content of the generated node. Properties of generated nodes are treated in exactly the same manner as properties of other nodes.

In the style sheet in Figure 1, there are three node types declared in the `ELABORATIONS` section. The first declaration defines a node type called *linebreak*, which can be used to add `<BR>` tags into a document to force linebreaks. The next declaration creates an `arrow` node type that can be used to insert an image of an arrow into the presentation. The third declaration allows the style sheet author to display the target of a link (the `href` attribute of the `A` tag).

The actual generation of nodes is controlled by the creation commands: `CreateLeft`, `CreateRight`, `CreateFirst`, and `CreateLast`. These commands create and attach nodes to the defining node. As the names suggest, the functions attach the nodes as left siblings, right siblings, first children, and last children, respectively, while maintaining the ordering specified in their argument list. In the style sheet in Figure 1, every `A` node will have `arrow`, `url`, and `linebreak` nodes created as right siblings as the result of the node specific rule:

```
CreateRight (arrow, url, linebreak);
```

3.3 Box Layout

In addition to the traditional "flow" layout model, PSL supports a *box layout* service for positioning the elements of a document. The box layout service is based on a model of nested boxes, or regions. Each node in the presentation tree has a bounding box, which for a two-dimensional medium such as text, is a rectangle that encloses the text of the node. The nodes of the presentation tree can be laid out by defining constraints between their bounding boxes.

The style sheet author lays out an element by specifying the size and location of its bounding box with respect to the dimensions of the document. For HTML documents, the properties `Width` and `HorizPos` specify size and position in the horizontal dimension, while `Height` and `VertPos` specify the size and position in the vertical dimension. Size properties are handled just like other properties in PSL, but position properties are treated differently. A position property references either the minimum, maximum, or center point of the bounding box along each dimension. The horizontal points are called `Left`, `Right`, and `HMiddle`, and the vertical points are `Top`, `Bottom`, and `VMiddle`. Position rules are defined with a special syntax:

```
<position name> : <point name> = <expression> ;
```

In the horizontal dimension, one possible rule would be:

```
HorizPos: Left = LeftSib.Right;
```

This rule constrains its node's left edge to be aligned with the right edge of its left sibling. Notice that point names without position names are used on the right-hand side of attribute access expressions. Examples of other layout rules are:

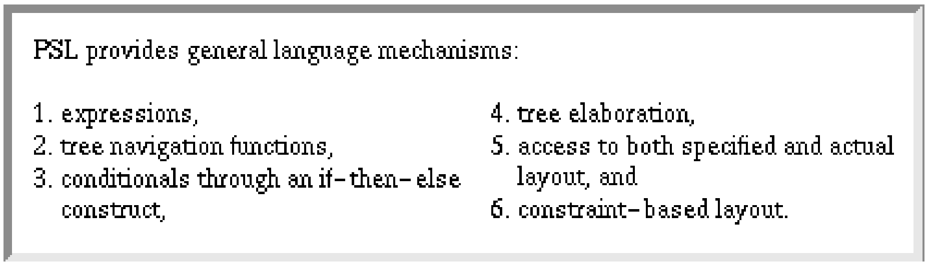
```
VertPos: VMiddle = 250;
Height = 100;
Width = LeftSib.Width * 2;
```

3.3.1 A Layout Example

PSL's box layout can describe generic layout effects. Suppose that an author would like to layout a list into two columns so that half of the items are placed in the first column and the remaining items are placed in the second column. The HTML document in Figure 5 is shown formatted in this style in Figure 4. A general rule for achieving this layout with N list items is that each item follows its predecessor (i.e. has the same left edge and has its top placed a little below its predecessor's bottom). The $(\frac{N}{2}+1)$ item is treated as an exception and is placed at the top of the second column. Since items "follow" their predecessors, only this one item needs a special layout rule — all subsequent items will appear below it in the second column. This rule can be described in PSL as follows:

```
LI {
   if (ChildNum(Self) == round(NumChildren(Parent) / 2 + 1)) then
      VertPos: Top = Parent.Top;
      HorizPos: Left = LeftSib.Left + Self.Width;
   else
      VertPos: Top = LeftSib.Actual Bottom;
      HorizPos: Left = LeftSib.Left;
   endif
   Width = 200;
}
```

The `Actual` keyword accesses the final results of the formatting process. In this example, `Actual Bottom` refers to the bottom of the formatted list item. The command `ChildNum(Self)` returns the child number of the defining node, and `NumChildren(Parent)` returns the number of children of its parent. Note that the top and left position of the first item is inherited from its parent since a first child does not have a left sibling. The width of each list item is arbitrarily set to 200 points; the width can be made dependent on the window size by using the `windowWidth` command.


PSL provides general language mechanisms:

1. expressions,
2. tree navigation functions,
3. conditionals through an if-then-else construct,
4. tree elaboration,
5. access to both specified and actual layout, and
6. constraint-based layout.

Figure 4: A list formatted into two columns where the first column contains half of the list items. The HTML source is in Figure 5. (Captured from MPMosaic)

```
<TITLE> Another Document </TITLE>
<P> PSL provides general language mechanisms:
<OL>
  <LI> expressions,
  <LI> tree navigation functions,
  <LI> conditionals through an if-then-else construct,
  <LI> tree elaboration,
  <LI> access to both specified and actual layout, and
  <LI> constraint-based layout.
</OL>
```

Figure 5: An HTML document containing a list of items.

3.3.2 Out-of-Order Layout

PSL places no restrictions on layout constraints. This allows the style sheet author to write layout rules that draw the elements of the document on the screen in an order different from their order in a traversal of the presentation tree. For instance, paragraphs can be laid out in reverse order so that the first paragraph is displayed last and the last paragraph is displayed first by using this rule:

```
P { VertPos: Top = RightSib.Actual Bottom; }
```

For a series of paragraphs, the top of each paragraph will be below the actual bottom of the next paragraph.

3.4 Other Features

PSL contains a number of other language features. As shown earlier, PSL's grammar contains an if-then-else conditional construct that is very useful for making rules dependent on the results of arbitrary expressions. The box layout system makes a useful distinction between *specified* and *actual* layout. PSL supports the addition of function-like commands (referred to as *interface functions*). For HTML, we added the functions `getAttribute` for obtaining the value of an element's attribute, `getMarkup` for obtaining the markup string used in the document, `getText` to return the text of a node, and `windowHeight` and `windowWidth` for getting the height and width of the browser's window.

3.5 Combining PSL's Services

Complex and dramatic presentations can be produced by PSL style sheets that employ a combination of services. Figure 6 shows a presentation produced by a PSL style sheet that graphically displays the tree structure of any HTML document. For each element in the document, elaboration adds text of the element's name, an ellipse around its name, and horizontal and vertical lines. Box layout is used to position all of the displayed elements. Each document element (and its ellipse) is constrained to be centered horizontally above the bounding boxes of its children. The content of the document is elided through a visibility property.

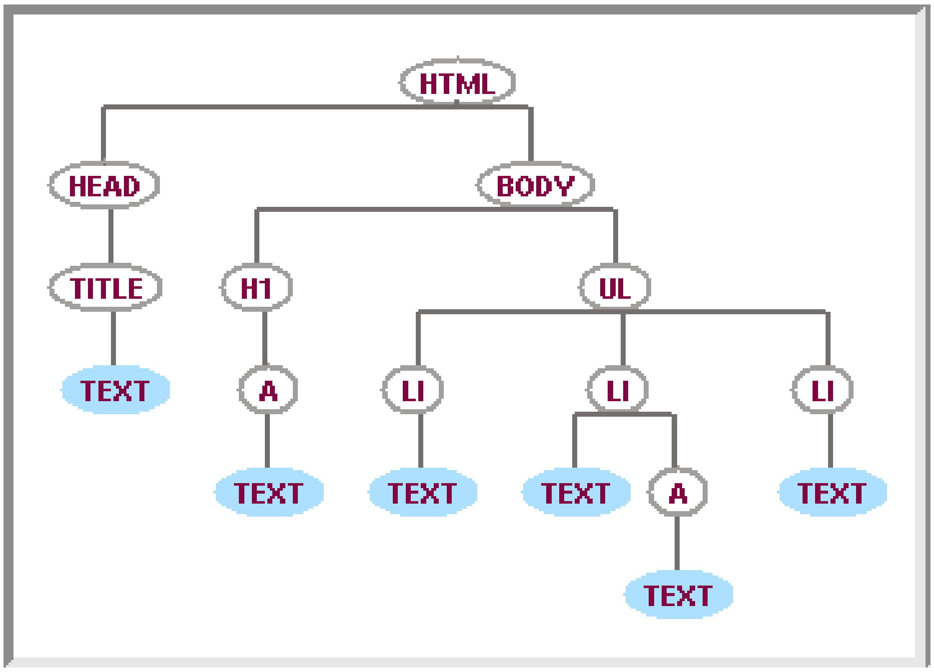


Figure 6: A presentation of a document's tree structure. The document in Figure 2 is presented according to the rules of a PSL style sheet that graphically displays the tree structure of HTML documents. (Captured from MPMosaic)

4 Comparing PSL and CSS

In this section, we compare some of the features of CSS and PSL. Our standards for comparing the two languages are based on the standards put forth in the introduction which stated that a style sheet language should have

- simple syntax;
- consistent and easily-described semantics; and
- the ability to specify a wide variety of useful presentations.

4.1 Syntactic Complexity

Both CSS and PSL have simple syntax, but CSS is still noticeably simpler than PSL.

- Where PSL style sheets have a number of different sections (e.g. `DEFAULT`, `RULES`, `ELABORATIONS`), CSS essentially has only one section that is equivalent to the `RULES` section of PSL.
- Because PSL allows the right-hand side of property rules to be general expressions, it has a standard set of syntactic rules for expressions. In contrast, the right-hand side of CSS property rules are simply constant values (generally strings, keywords, or numbers).

- PSL has a conditional rule syntax for which CSS has no equivalent.
- CSS supports a variety of contextual selectors that are used to specify rules that apply to elements only when they appear in certain contexts. The syntax for specifying contextual selectors is simple. In PSL, similar ideas can only be expressed using conditional rules.

Overall, CSS has somewhat simpler syntax than PSL, though the difference is not dramatic, especially when compared to languages like XSL and DSSSL.

4.2 Semantic Consistency

A more striking contrast between PSL and CSS can be found in the area of semantics. PSL has simple and consistent semantics that are applied uniformly throughout the language. For example, property rules have the form:

```
<property> = <expression> ;
```

The right-hand side of these rules may be *any* expression whose type matches the data type of the property. The restriction that the types match is the only restriction. Since PSL provides a wide range of expression operators, it is possible to specify almost any desired effect, but some effects may require complex expressions.

In contrast, CSS is designed to allow convenient specification of common presentation effects and does so at the price of consistency. CSS property rules are written

```
<property> : <value> ;
```

Most CSS properties define a mixture of keywords, numbers, and length measurements, and these values are predetermined to be absolute values or relative to another property value.

As an example, here are four different rules for the CSS `font-size` property:

```
P { font-size: 12pt; }
P { font-size: medium; }
BLOCKQUOTE { font-size: larger; }
EM { font-size: 200%; }
```

The first rule uses a *length* which is an absolute value. The second rule uses an *absolute-size* keyword which gets a value from a table of size preferences maintained by the browser. The third rule uses a *relative-size* keyword that specifies a size relative to the parent. The fourth rule uses a *percentage* which is also relative to the parent's size.

Nearly every CSS property has different rules for the values on its right-hand side and it is not much of an exaggeration to say that each property's right-hand side has its own specialized language. This point is illustrated by the `line-height` property. It does not accept the keywords that can be used with `font-size` and, in addition, percentages are interpreted relative to the font-size of the current element, rather than relative to the parent element's line-height. For example, this rule

```
EM { line-height: 200%; }
```

specifies that the line height for elements `EM` should be twice as large as its font size. This is a *natural* way to specify line height, but it is not consistent with the treatment of percentages in other parts of the language.

In a small language like CSS1, some inconsistency in semantics is easily tolerated. However, as a language's scope is increased, as is being done with CSS2, it becomes progressively harder to describe the language and to understand it. So, even though the semantics for each property are generally intuitive, we believe that as properties are added to CSS, its users could become overwhelmed by special cases and will be forced to depend on manuals as they write style sheets.

PSL's consistent semantic design allows it to be well described by a small set of rules that are independent of the set of properties used by a particular application. It is certainly the case that as new properties are added, they will have to be described and will make descriptions of the language more complex. However, any additional complexity is only the result of a more complex style model, not the result of greater complexity of the specification system.

4.3 Expressive Power

PSL has more expressive power than CSS because it provides general mechanisms to describe presentation styles. CSS tries to provide expressive power through a wide variety of special cases. However, the range of special cases available is limited to those cases that CSS's designers have recognized as valuable. The problem is not that CSS's designers are not perceptive and thoughtful, but that it is simply not possible for any language designer to anticipate every interesting and useful presentation style.

Suppose that a document designer would like to emphasize text by making it 20% larger than the text that precedes it. In PSL, this style is easily specified with one rule:

```
EM { fontSize = LeftSib.fontSize * 1.2; }
```

In general, CSS cannot specify this style because, for the font-size property, relative values are always relative to the parent. (Extensive use of contextual selectors might be sufficient to handle this case, but would require enumerating every possible context that `EM` elements could appear in.)

Due to its general language mechanisms, PSL is not only able to specify a much wider variety of presentations than CSS, but is dramatically better than CSS at generalizing specifications. In the two-column layout example in Section 3.3.1, PSL specifies the layout through a mathematical formulation. This specification can be applied to any list with any number of list items in any document. In contrast, CSS is unable to describe this layout in a general way because CSS does not support general mathematical expressions. A CSS author must position each list item individually to layout a list into two columns so that each column contains half of the list items. Currently, the most common practice of CSS authors is to modify the document by hard-coding identifiers into the elements of the list using the HTML `id` attribute and then to use these identifiers in the style sheet to position the elements. Figures 7 and 8 show this approach. The CSS approach requires authors to manually calculate the position values and the style rules must be edited if list items are added to the document.

```
<TITLE> Another Document </TITLE>
<P> PSL provides general language mechanisms:
<OL>
  <LI id=a> expressions,
  <LI id=b> tree navigation functions,
  <LI id=c> conditionals through an if-then-else construct,
  <LI id=d> tree elaboration,
  <LI id=e> access to both specified and actual layout, and
  <LI id=f> constraint-based layout.
</OL>
```

Figure 7: `id` attributes have been added to the list items of the Figure 5 document so that the list can be laid out by the CSS rules in Figure 8.

```
LI { width : 200px; position: absolute; }
#a { top : 0; left : 0; }
#b { top : 14px; left : 0; }
#c { top : 28px; left : 0; }
#d { top : 0; left : 200px; }
#e { top : 14px; left : 200px; }
#f { top : 42px; left : 200px; }
```

Figure 8: These CSS style rules will layout the document in Figure 7 into two columns (as shown in Figure 4). In CSS, `top` and `left` are offsets from the top and left edges of the element's containing block. The pound symbol selects elements by their `id` attribute.

5 Experience with MPMosaic

Our experiences with PSL have been very positive. Our testbed is Multiple Presentation Mosaic (MPMosaic) [MM98], a modified version of NCSA's Mosaic browser. In MPMosaic, users can select their own style sheets to control the appearance of a document regardless of the document's origin or authorship. MPMosaic also supports multiple presentations — users can open multiple windows displaying the same document with each window using a different style sheet (see Figure 9).

We have written a number of PSL style sheets that provide "views" of HTML documents. Below are a few of these views.

Table of contents view shows only the headings (the H1 through H6 elements).
Links view shows only anchors and their destinations (see Figure 3).
Embedded tags view displays the markup tags in the formatted document.
Tree-structured view graphically renders the tree structure of a document (see Figure 6).
Reduced size view shows all fonts and images at half size.

These views can be as used as visualization tools to help users understand documents. The links view, for example, can be used as a inter-document navigation tool since it shows all of the possible destinations the user can reach from the current document. Figure 3 shows the links view of the document in Figure 2. The links view is generated from the style sheet in Figure 1. Views used for visualization seem to be most useful when they are displayed side by side with the standard HTML presentation. Figure 9 shows four different views of the same document: the standard view along with the first three views listed above.

Support for PSL was added to Mosaic by using the Proteus style sheet system. Mosaic's formatter was not significantly altered when we added the Proteus library to Mosaic to make MPMosaic. Approximately 600 lines of C++ code were added to the 8000 line formatter in Mosaic.

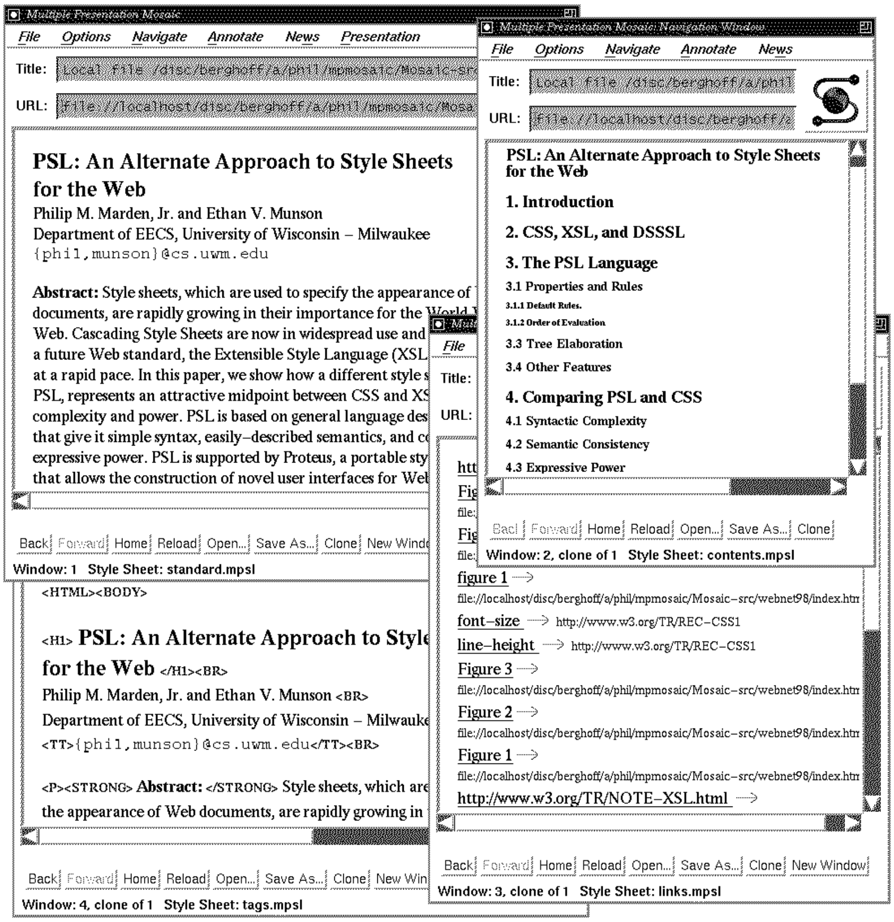


Figure 9: MPMosaic is displaying four different presentations of the same document. Listed clockwise from the upper left window, the presentations are the stardard view, the table of contents view, the links view, and the embedded tags view.

6 Conclusions

Many people in the document community have believed that in order for a style sheet language to have powerful formatting capabilities, the language must approach the complexity of a programming language and that style sheet languages with simple syntax will be special purpose languages with limited capabilities. Our experience with PSL has shown that there is a middle ground: a style sheet can have simple syntax and semantics while having considerable expressive power. PSL's power is the result a design based around general principles with an emphasis on simple and consistent language constructs.

Acknowledgements

This work was supported in part by the U.S. Department of Defense. Its successful completion and this article would not have been possible without the help and support of the members of the Multimedia Software Lab at UWM, especially Terry Cumaranatunge.

References

[GHM92] Susan L. Graham, Michael A. Harrison, and Ethan V. Munson. The Proteus presentation system. In *Proceedings of the ACM SIGSOFT Fifth Symposium on Software Development Environments*, pages 130–138, Tyson's Corner, VA, December 1992. ACM Press.

[Gol86] Charles F. Goldfarb, editor. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*. International Organization for Standardization, Geneva, Switzerland, 1986. International Standard ISO 8879.

[ISO94] ISO/IEC. *Information technology — Text and office systems — Document Style Semantics and Specification Language (DSSSL)*, August 1994. Draft International Standard ISO/IEC DIS 10179.2.

[MM98] Philip M. Marden, Jr. and Ethan V. Munson. Multiple Presentations of WWW Documents Using Style Sheets. In *Proceedings of ED-MEDIA 98, Conference on Educational Multimedia and Hypermedia*, Freiburg, Germany, June 1998. Association for the Advancement of Computing in Education.

[Mun95] Ethan V. Munson. A new presentation language for structured documents. *Electronic Publishing: Origination, Dissemination, and Design*, 8:125–138, September 1995. Originally presented at EP96, the Sixth International Conference on Electronic Publishing, Document Manipulation, and Typography, Palo Alto, CA, September 1996.

[Wor96] World Wide Web Consortium. *Cascading Style Sheets, level 1*, December 1996. http://w3c.org/TR/REC-CSS1.

[Wor97] World Wide Web Consortium. *A Proposal for XSL*, August 1997. http://www.w3.org/TR/NOTE-XSL.html.

[Wor98a] World Wide Web Consortium. *Cascading Style Sheets, level 2*, May 1998. http://www.w3.org/TR/REC-CSS2.

[Wor98b] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*, February 1998. http://www.w3.org/TR/REC-xml.

[Wor98c] World Wide Web Consortium. *HTML 4.0 Specification*, April 1998. http://www.w3.org/TR/REC-html40.