ELECTRONIC PUBLISHING, VOL. 8(2 & 3), 125–138 (JUNE & SEPTEMBER 1995)

A new presentation language for structured documents

ETHAN V. MUNSON

Department of Electrical Engineering and Computer Science
University of Wisconsin-Milwaukee
Milwaukee, WI 53201
U.S.A.

e-mail: `munson@cs.uwm.edu`

SUMMARY

PSL is a new presentation specification language for structured documents. It is the first such language that is fully configurable and it is also extensible. PSL is able to support a very general form of out-of-order layout without having to provide a general system of tree transformations. PSL also makes an explicit distinction between the specified layout of the elements of a document and the actual layout that results from the formatting process. PSL's syntax and semantics are simple and general.

This paper describes the syntax and semantics of PSL using a simple text document as a running example and compares PSL to a number of other presentation specification languages.

1 INTRODUCTION

A central premise behind the development of the structured document model has been that the appearance of documents should be specified separately from their structure and content. This separation allows the user of a structured document system to display the same document in very different styles without modifying the document itself. The user simply chooses a different style specification and the document's appearance changes to match the appearance rules found there. This flexibility in presentation is an important advantage of structured document systems over mass-market word-processing and desktop-publishing software. Clearly, the power and flexibility of the language used to specify presentation is critical to the success of a structured document system.

This paper presents a new presentation specification language, PSL, which has four innovative qualities.

Configurability The language definition makes no mention of the features of any particular application or medium. It defines a set of medium-independent constructs which can be adapted by each structured document application to that application's particular needs.

Extensibility PSL provides an extension service which allows the application to introduce new concepts into the specification language.

Out-of-Order Layout In PSL, layout is specified by constraints between elements of the document. It is possible define constraints that place document elements on the screen in an order different from their order in a tree traversal. This approach to layout is quite different from the standard flow model and eliminates the need to support a general tree transformation system, such as can be found in DSSSL [1].

CCC 0894–3982/95/020125–14 Received 16 April 1996
Revised 28 June 1996

Actual/Specified Layout: PSL makes an important and useful distinction between the *specified* size and position of an object and the *actual* size and position that result from the formatting process. Making this distinction explicit is critical to the success of PSL's constraint-based layout specifications.

In addition, PSL is designed for ease of use. Both the syntax and semantics of PSL are quite simple and general. There are essentially no special cases and the language appears to be quite accessible to users though, to date, there has been no formal usability testing.

PSL is the specification language of Proteus [2,3], a portable presentation specification system that supports multiple, synchronized presentations. Proteus was originally designed to meet the needs of the Ensemble software development and multimedia document environment [4]. More recently, it has been made into an independent library. A modified version of the NCSA Mosaic browser for the World Wide Web which uses Proteus for presentation control has recently been created [5] and work is under way to use Proteus in the Lynx tty-based WWW browser and a Windows-based editor for canonical SGML documents.

The remainder of this paper is organized as follows. Section 2 provides some background on presentation specification languages for structured documents and on the applications for which PSL was designed. Section 3 presents the PSL language in some detail along with examples of its use. Section 4 compares PSL to other presentation specification languages and suggests future directions for the language. It is followed by conclusions in Section 5.

2 BACKGROUND

While there is long history of research on structured document editors [6,7,8,9], work on presentation specification languages became common only recently.

For PSL, the most closely related research is the pioneering work of Quint and Vatton on Grif [7,9] and its presentation schema language, P [10]. Grif was the first document system to provide significant support for multiple simultaneous presentations and compound documents (documents that include other documents in arbitrary combinations). P is a powerful, declarative presentation language providing excellent support for text formatting and more limited support for mathematics and graphics formatting.

P has an easy-to-read syntax reminiscent of Pascal, however, the grammar and semantics have many special cases which make aspects of the language difficult to grasp. In particular, many presentation rules in P are written in a manner that suggests that formatting attributes may be constrained to the value of arbitrary expressions, when in fact only a very restricted class of expressions is allowed.

Weitzman and Wittenberg showed how relational grammars can be used for multimedia document presentation [11]. Relational grammars are Lisp-based specifications of document structure which they extended to support presentation control. Document layout is specified by constraint, but only simple alignment constraints are allowed. Non-layout aspects of presentation are controlled using syntax similar to 'set' functions in an object-oriented language. Relational grammars specify structure and presentation together, which prevents them from supporting multiple presentations.

DSSSL [1] is a proposed presentation specification language for SGML that is now a Draft International Standard. While a complete implementation of DSSSL is not yet widely

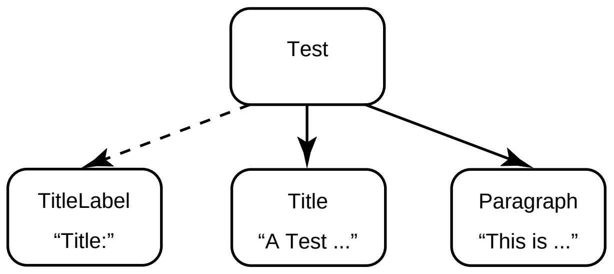


Figure 1. The trees used in the sample presentation. The nodes and arcs drawn in solid lines represent the document tree. The presentation tree contains all the nodes of the document tree plus the dashed node and arc

available, DSSSL specifications appear to work well for documents that are primarily textual. DSSSL is a complex language based on Scheme that supports many data types. While it is very powerful, it also appears inaccessible to end-users.

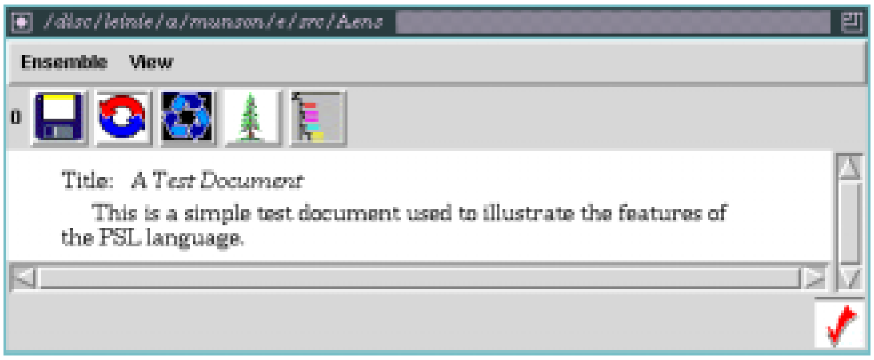
DSSSL has another important limitation. In order to support document presentations that rearrange the order of elements in a document, contain generated information like section numbers, or elide parts of the document, DSSSL has a module that can perform arbitrary transformations on the document tree. These unrestricted transformations make it difficult to map on-screen selections in a WYSIWIG editor back to the pre-transformation document tree and thus hinder the construction of direct-manipulation editors for SGML documents. As a result, DSSSL is considered unsuitable for interactive applications. Other SGML style systems [12,13] sacrifice the ability to do out-of-order layout in order to gain interactivity.

Neither P nor relational grammars are configurable or extensible in a general way. This means that it is simply not possible to add features that the designers have not already conceived of. These languages cannot be applied to new media, cannot support unanticipated formatting features, and cannot allow presentation to be based on concepts outside the existing scope of the language.

The DSSSL standard is somewhat more adaptable. Parts of the standard are optional and may be left out of particular implementations. The standard allows the definition of new flow objects and new formatting parameters (called *characteristics* and *properties*). The mechanism for defining characteristics is fully described, but the mechanisms for properties and flow objects are not explained in much detail, so it is difficult to determine how much configurability they provide. Furthermore, much of the DSSSL specification assumes the formatting is done in two dimensions, which largely restricts its use to text and static two-dimensional graphics.

3 THE PSL LANGUAGE

The PSL language is best understood by working with example documents. This section uses a running example, called the ‘sample document,’ whose document tree is shown in Figure 1. The “sample presentation” of this document (produced using Ensemble) can be seen in Figure 2.



Title: *A Test Document*

This is a simple test document used to illustrate the features of the PSL language.

Figure 2. The sample presentation

3.1 Overview

A PSL specification, called a *presentation schema*, is a simple, declarative specification of appearance for a class of documents. PSL expects this class to be defined by a separate grammar such as a structure schema written in S [10] or an SGML DTD [14]. However, the current Proteus runtime system does not require access to this grammar.

A presentation schema specifies how three presentation services (tree elaboration, attribute propagation, and box layout) should be applied to instances of the document class. Figure 3 shows the 'sample schema' that was used to produce the sample presentation.

A presentation schema has four sections: header, defaults, elaborations, and rules. The

```
MEDIUM text;
PRESENTATION sample FOR test;

DEFAULT {
  VertPos:
    Top = LeftSib . Actual Bottom;
}
ELABORATIONS {
  TitleLabel : Text("Title:") {}
}
RULES {
  TEST {
    Width = 432;
    HorizPos: Left = 36;
    VertPos: Top = 12;
    FontFamily =
      "new century schoolbook";
    Size = 14;
    Bold = No;
    Italic = No;
    LineSpacing = 1.2;
    Justify = LeftJustify;
    Indent = 0;
    Visible = Yes;
  }
  TITLE {
    CreateBefore(TitleLabel);
    HorizPos: Left = LeftSib .
      Actual Right + 12;
    VertPos: Top =
      LeftSib . Top;
    Italic = Yes;
  }
  PARAGRAPH {
    VertPos: Top = LeftSib .
      Actual Bottom + 5;
    Justify = BlockJustify;
    Indent = 20;
  }
}
```

Figure 3. Presentation schema used to produce the sample presentation

header section declares the schema's medium, its name, and the name of the structure schema to which it corresponds. The DEFAULTS section defines default presentation rules that are used when there are no specific rules for a node. The ELABORATIONS section declares the types of nodes that can be generated by tree elaboration. Finally, the RULES section is used to define specific presentation rules for each of the node types defined in the document class.

In general, PSL is case-insensitive. An important exception occurs for node names, which are case-sensitive. Node names are handled specially because document designers do not always have control over the grammars for their documents. This is particularly true for programming language grammars which may have to conform to standards that assume the use of case-sensitive compiler tools (such as Bison [15]). In addition, node names may either be unquoted identifiers (as shown in Figure 3) or quoted strings. This allows the use of grammars that define nodes whose names are also keywords in PSL. Such names must be quoted in order to avoid producing parse errors in the schema.

3.2 Tree elaboration

Tree elaboration is used to generate material that is not present in the document itself, such as the label 'Title:' in the sample presentation. Tree elaboration works by adding nodes to a presentation tree, which is a copy of the document tree that has been elaborated by additional nodes specified in the presentation schema. The presentation tree for the sample presentation was shown in Figure 1. It contains all three nodes of the document tree plus a new node of type `TitleLabel`.

Tree elaboration is specified in two parts: node declarations and creation commands. Nodes that can be generated are declared in the ELABORATIONS section of the schema. The elaborations section of the sample schema declares TitleLabel nodes to be of the Text primitive type and to be initialized with the string 'Title:'. The initialization argument for this node could have been any expression returning a string. It is also possible to generate internal nodes, which allows the creation of more complex structures such as tables of contents. The pair of braces at the end of the declaration could have contained presentation rules specific to TitleLabel nodes.

The actual generation of nodes is controlled by creation commands. The TitleLabel node is added to the presentation tree because the RULES section contains a creation command for the Title node type:

```
CreateBefore{TitleLabel);
```

The `CreateBefore` command causes a node of type TitleLabel to be created as a left sibling of the Title node. There are three other creation commands. The `CreateAfter` command creates right siblings, while the `CreateFirst` and `CreateLast` commands create first and last children, respectively. All creation commands take a single argument which is the name of a node declared in the ELABORATIONS section of the schema.

3.3 Attribute propagation

The central operation of Ensemble's text medium is the line-breaking of paragraphs. The line-breaking operation requires the text of a node and a number of parameters that specify font, justification, line spacing. From these inputs, it converts the words of the text into lines of characters to be drawn on a screen or page. The values of the line-breaking operation's parameters are defined by the *attribute propagation* rules of the presentation schema.

For instance, attribute propagation rules in the sample schema set the font of the root node of the `Test` document to be 14 point New Century Schoolbook in the Roman style (which is neither bold nor italic). These values are propagated to the rest of the presentation tree by simple inheritance. The Title node has a rule that overrides the inherited value of the Italic attribute.

3.3.1 Attribute rules

Attribute rules have a simple syntax:

<*attribute name*> `=` <*expression*> `;`

Each application has its own set of attributes, determined by the demands of its formatting operations. For instance, Ensemble's text medium currently has 15 attributes controlling font (FontFamily, Size, Bold, and Italic), hyphenation (Hyphenate, MinHyph, MinLeft, MinRight), justification, indentation, line-spacing, visibility, foreground color, and background color. Each attribute has a type, which is either boolean, string, real, or an application-specific enumeration type.

The right hand side of the attribute rule can contain any expression whose type is the same as that of the attribute 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.

All the presentation rules (attribute, tree elaboration, and box layout) for a single node type are defined together in a rule list, which appears between braces after the name of the node. Typically, the rule block contains several rules, each terminated by a semi-colon. Alternate rule sets for a node type can be defined using an if-then-elsif-else syntax.

3.3.2 Defining attribute constraints

Attribute values can be constrained to depend on the attribute values of other nodes by using the attribute access expression, for which the syntax is:

<*node expression*> `.` <*attribute name*>

The value of an attribute access expression is the value of the named attribute for the node returned by the expression on the left hand side of the dot. For instance, the following expression gives the value of the parent node's `Size` attribute:

```
Parent . Size
```

It is not possible to define attributes whose type is 'node', but there are several functions that return nodes, any of which can appear in the left hand side of an attribute access expression. Some of these functions return immediate neighbors (`Parent`, `LeftSib`, `RightSib`, `FirstChild`, `LastChild`, and `NthChild`), while others return nodes that may be more distant (`Root`, `AncestorOfType`, and `Creator`). All these functions have two forms. One form takes an explicit argument of type node. The other form does not have this explicit argument and instead returns a value computed relative to the *defining node*. The defining node is the node for which the attribute rule was defined. There is also a `Self` function which returns the defining node and is used in contexts where an explicit node is required.

Collectively, these functions are designed to allow the specification of constraints between the defining node and every other node in the tree. It is easiest to define constraints with neighboring nodes, since the tree navigation functions and the `NthChild` function can be used to specify all immediate neighbors. More distant nodes in the tree can be specified through function composition, as in

```
FirstChild(LeftSib(Parent)) . Size
```

which specifies the `Size` attribute of a 'cousin' node.

3.3.3 Default rules and the order of evaluation

Proteus's presentation schema language has a system of default rules that help reduce the size and complexity of presentation schemas. There are two sets of default rules, explicit and implicit.

Explicit default rules are defined in the DEFAULTS section of the schema which contains a rule list having the same syntax and semantics as the rule lists for specific node types. These rules are primarily used when there is no node-specific rule for the attribute, but they are also used when the node-specific rule *fails*. A rule fails when 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 tree navigation. For instance, a node's first child has no left sibling, so if the `LeftSib` function is invoked with a first child as its argument, the function fails.

The *implicit default rule* is used when there is no explicit default rule for an attribute or when the explicit default rule fails. The implicit default rule uses simple inheritance, which is equivalent to a rule of the form

```
Attribute = Parent . Attribute ;
```

The implicit default rule can fail if the root node does not have a valid rule. So, to insure that attribute evaluation always returns a valid value, every attribute has a global default value which is returned in this case.

3.4 Box layout

In Proteus, the *box layout* service is used to specify the positions of the elements of the document. The box layout service is based on a model of nested boxes. In this model, each node in the presentation tree has a bounding box, which for a two-dimensional medium like text, is the smallest rectangle that encloses all of the text of the node. The nodes of the presentation tree are laid out by defining constraints between these bounding boxes.

The box layout service defines four attributes for each dimension supported by the medium. The generic names of these attributes are *extent*, *minimum*, *maximum,* and *center* but each application renames them. For example, in Ensemble the text medium has two dimensions *Horizontal* and *Vertical*. The attributes of the Horizontal dimension are `Width`,

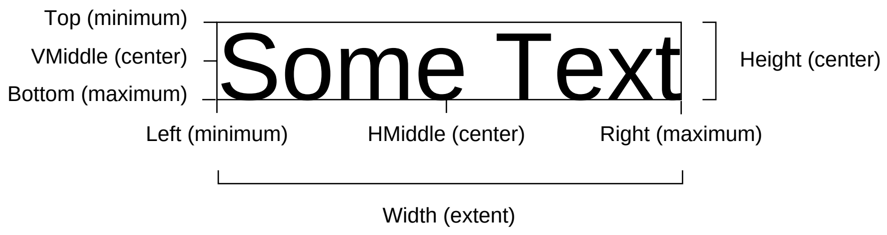


Figure 4. Box layout attributes. The rectangle around the text fragment represents the fragment's bounding box. The four box layout attributes of the horizontal dimension are shown below the bounding box. The position attributes of the vertical dimension are shown on the left side of the bounding box, while the vertical extent attribute (`Height` *is shown on the right side*

`Left`, `Right`, and `HMiddle` while those of the Vertical dimension are `Height`, `Top`, `Bottom`, and `VMiddle`. Figure 4 shows the relationship between these attributes and the bounding box of a text fragment.

While each dimension logically has four attributes, there are only two degrees of freedom among them. Thus, some special handling is required for dimensional attributes. PSL adopts the solution used in Grif's presentation language, P [10], which is to give each dimension only two real attributes, *extent* and *position*. The three non-extent attributes (minimum, maximum and center) are given a subsidiary role to the position attribute and are called *points*. In the text medium, the position attributes for the horizontal and vertical dimensions are called `HorizPos` and `VertPos`, respectively.

Extent attributes are handled just like other attributes in the presentation schema language, but rules for position attributes are defined with a special syntax:

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

In the text medium's 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, not position names are used on the right-hand side of attribute access expressions.

3.4.1 Specified and actual layout

Experience working with presentation schemas in Ensemble [16] led to the discovery that there are actually two sets of bounding boxes for the elements of a document: *specified* and *actual*. The rules in presentation schemas define the specified bounding box, which expresses a nominal size for the document element. This line from the schema for the sample presentation

```
Width = 432;
```

specifies that the width of the Test node (and by the implicit default rule, its TitleLabel, Title and Paragraph children) should be 432 points (6 inches). But the *actual* widths of the TitleLabel and Title nodes are much smaller because they don't contain enough text to fill a line.

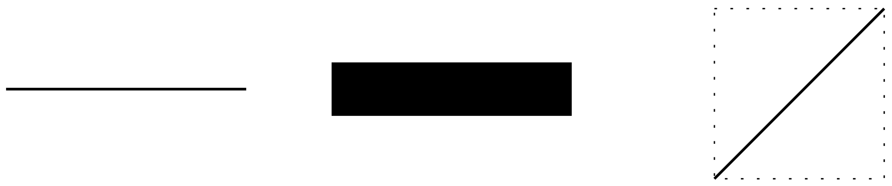


Figure 5. This figure is illustrates the effect of stroke-width and rotation on the actual bounding box of a line. These three lines are all of the same length. The left line is stroked with a .5 point pen and has a rotation of 0°*. The middle line is stroked with a 20 point pen and has a rotation of* 0°*. The right line is stroked with a .5 point pen and has a rotation of* 45°*. Its bounding box is shown as a dashed square*

It is important to give the author of a presentation schema access to both the specified and actual dimensional attributes. The schema for the sample presentation includes the following rule:

```
Title {
   HorizPos: Left = LeftSib . Actual Right + 12;
   ...
}
```

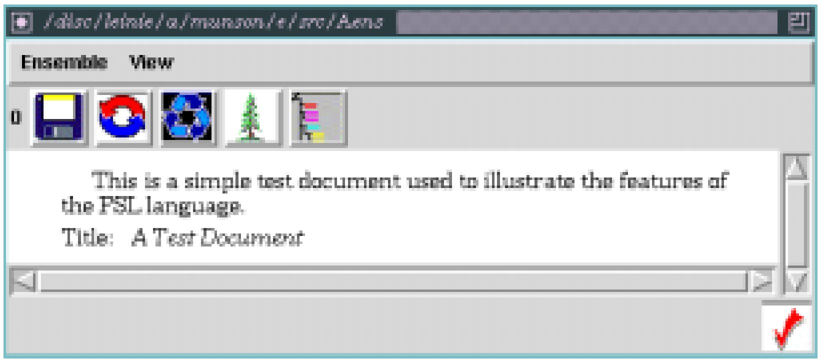
This rule sets the Title node's *specified* left edge to be 12 points to the right of the TitleLabel node's *actual* right edge, which is about 120 points from the left edge of the window in Figure 2. If the `Actual` keyword were absent, the Title node's left edge would be set at a ridiculous position 468 points from the left edge of the window (i.e. the sum of the TitleLabel node's specified `Left` and `Width` attributes).

This distinction between specified and actual layout is important in all media. Figure 5 shows some of the ways the distinction arises in graphics, where attributes like stroke-width and rotation can dramatically alter the bounding box of an object.

3.4.2 Out-of-order layout

PSL places no restrictions on layout constraints. This allows the presentation schema 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. Figure 6 shows a new presentation of the sample document in which the Title and TitleLabel nodes are displayed *after* the Paragraph node, even though they precede the Paragraph node in the presentation tree (shown in Figure 1). The box layout rules that produced this presentation were

```
ELABORATIONS {
   TitleLabel : Text("Title:") { VertPos: Top = RightSib . Top; }
}
RULES { ...
   TITLE {
      CreateBefore(TitleLabel);
      HorizPos: Left = LeftSib . Actual Right + 12;
      VertPos: Top = RightSib . Actual Bottom + 5 ;
   } ...
```



This is a simple test document used to illustrate the features of the PSL language.
Title: *A Test Document*

Figure 6. A presentation of the sample document showing out-of-order layout

Because the box layout system can support out-of-order layout, PSL does not have to include a general tree-transformation module, such as the Standard Tree Transformation Process of DSSSL [1]. All the layout effects that can be performed with tree-transformations can also be achieved with a combination of tree elaboration, out-of-order layout, and elision (using a `Visibility` attribute).

3.5 Interface functions

The features of PSL that have been described so far are quite powerful, but there can also be important information relevant to presentation that they cannot represent. To address this problem, PSL can be extended via a system of *interface functions*, so called because they cross the interface between the PSL runtime system and its client applications.

As an example, Ensemble's text medium defines an interface function called `URoman` that takes a numeric argument and returns a string containing the upper-case roman numeral representation of the number. This function can be used to number the paragraphs of a document by declaring a generated `ParaNumber` node whose content is the upper-case roman numeral representation of its creator's child number:

```
ParaNumber : Text(URoman(ChildNum(Creator))) {}
```

and then using the `CreateBefore` creation command to generate the ParaNumber as the left sibling of each Paragraph in the document.

Within presentation schemas, interface functions are simply another kind of expression. The `URoman` function can be used in any context where a string-valued expression is appropriate. Its argument can be any expression returning a number.

The `URoman` function is easy to understand, but is not a very compelling example. The interface function mechanism was primarily designed to make three kinds of information available within presentation schemas:

- Medium-specific information, such as the area of irregularly-shaped graphical objects;
- Editing state, such as the current selection and the current cursor location; and
- Results of document analysis, such as spell-checking or static semantic analysis of programs. Ensemble's text medium currently includes interface functions that allow access to the results of program analysis.

3.6 Configurability

PSL is a configurable language that is adapted to meet each client application's particular needs. The client application provides the following information in order to configure PSL:

- for tree elaboration, the names of the primitive data types that can be generated and the type signatures for their creation;
- for attribute propagation, the name and type of each attribute; and
- for box layout, the names of the attributes of each dimension.

In Proteus, this information is represented by instances of the ProtMedium class. Currently, these ProtMedium objects are defined by hand-written C++ code, but work is underway to generate them from simple, declarative specifications.

This paper's running example was developed using Ensemble's text medium. Ensemble has other media which use PSL and Proteus independently and thus take different presentation schemas. The grammar for all PSL schemas is the same, but the details (primitive types, attributes, and dimensions) change between media. For instance, Ensemble's video medium has three dimensions (horizontal, vertical and time), while its graphics medium has a very different set of attributes including StrokeWidth and Rotation. Liu [5] recently modified NCSA Mosaic so that its presentation of HTML documents is controlled by presentation schemas written in PSL, rather than hard-coded formatting variables. Liu's version of Mosaic uses a PSL configuration similar to that for Ensemble's text medium (because most formatting *choices* for HTML revolve around text formatting).

4 COMPARISON TO RELATED WORK

PSL improves on existing presentation languages in several ways.

Configurability: No other presentation language is as configurable as PSL. The P language and relational grammars are not configurable at all. DSSSL can be configured by defining new flow objects, properties, and characteristics in the style specification, but these features are not yet fully defined and there is little experience in using them. Furthermore, it is not clear why configuration commands like these, which only need to be specified once per application, should have to appear in every document style file used with that application. Finally, PSL provides a much clearer separation between the application-independent and application-specific aspects of the language.
PSL's configurability makes it well suited for adoption by existing applications. Liu's work using Proteus/PSL with NCSA Mosaic [5] did not require any substantive changes to Mosaic's formatting model or document structure. Furthermore, the ease of configuring PSL makes its possible to add features incrementally.

Extensibility: The PSL language can be extended via its interface function mechanism. This allows a presentation schema to ask for information that would not normally be accessible with rest of the language. No other presentation language has an explicit extension service.

Out-of-Order Layout: PSL's constraint-based layout system provides the most powerful and straightforward means of specifying out-of-order layout of any existing presentation language. DSSSL supports out-of-order layout by providing a general tree-transformation component, the Standard Tree Transformation Process (STTP). But this module is so general that it makes DSSSL inappropriate for interactive, WYSIWIG applications. P and relational grammars support out-of-order via constraints, but do not allow arbitrary constraint expressions and thus lack the expressive power of PSL.

Specified vs. Actual Layout: PSL is the only presentation language to make explicit the distinction between specified layout and actual layout. This distinction is critical, because we often specify nominal widths, heights, and positions of objects but allow the formatting process to produce objects whose sizes and positions differ somewhat from what was specified.

Only the P presentation language supports a similar notion, but it does so implicitly. The Grif system creates three data structures when it presents a document: a document tree, an abstract picture, and a concrete picture [17]. Grif's document tree and abstract picture are equivalent to PSL's document tree and presentation tree. Grif's concrete picture is a tree of bounding boxes. Certain constructs in P (e.g. the `Enclosed` keyword) imply that an attribute value should be computed on the concrete picture, rather than the abstract picture. This is roughly equivalent to accessing an *actual* dimensional attribute value in PSL, but PSL has achieved the same result without introducing another data structure abstraction — all PSL rules operate on the presentation tree.

Simplicity: PSL has been designed to be accessible to end-users. It has a simple, easy-to-describe grammar with few special cases. Functions and attribute definitions accept any expression returning the correct type. The semantics of PSL are equally straightforward. In contrast, P has many special cases, DSSSL is a huge language with complex semantics, and relational grammars are based on a somewhat tricky non-deterministic parsing model. It is only fair to say that much of the simplicity of PSL derives from the fact that it does not support any specific formatting concepts (other than tree elaboration and box layout). An end-user will certainly have to understand the semantics of the application's formatting model, which may be complex.

The key limitation of PSL is lack of experience using it in a variety of settings. P, and the Grif/Thot system, have been under development for many years and have been commercialized. There is considerable experience using them for a wide variety of document types. P also has a number of useful features which do not have direct equivalents in PSL, such as counters, variables, and attributes. The DSSSL standard has also been developed over a long period and it supports a wide range of layout styles within its flow model. The language supports characters types and flow directions from many languages.

PSL's box layout service is not suitable for all applications. For instance, text formatters which use the boxes-and-glue page-breaking algorithm [18] cannot use box layout alone. Such a system would either have to supplement box layout with additional attributes or would have to ignore box layout completely (by defining a configuration with zero dimensions) and use attributes alone to control layout.

5 CONCLUSIONS

PSL is a new presentation specification language that is unusually configurable and extensible. Its syntax and semantics are notable for their simplicity and generality. PSL supports an extremely general form of out-of-order layout without having support general tree transformations. PSL makes explicit the important distinction between specified layout and actual layout.

Future research on PSL will focus on making the language easier to use and increasing its power, particularly for computer programs and non-textual documents. PSL needs features that simplify presentation specifications for objects with similar rules (which are common in programs) and it would benefit from new syntax for defining complex tree elaborations (useful in graphics documents). PSL also needs support for numbering document elements and experiments must be performed to determine its suitability for applications which use layout models different from box layout.

ACKNOWLEDGEMENTS

I want to thank Vance Maverick and an anonymous reviewer for their many helpful comments on this manuscript.

This research has been supported in part by a gift from Frontier Technologies Corporation and by a grant from the UWM Graduate School.

REFERENCES

1. 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.
2. 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*, pp. 130–138, Tyson's Corner, VA, (December 1992). ACM Press.
3. Ethan Vincent Munson, *Proteus: An Adaptable Presentation System for a Software Development and Multimedia Document Environment*, PhD dissertation, University of California, Berkeley, December 1994. Also available as UC Berkeley Computer Science Technical Report UCB/CSD-94-833.
4. Susan L. Graham, 'Language and document support in software development environments', in *Proceedings of the Darpa '92 Software Technology Conference*, Los Angeles, (April 1992).
5. Hong Liu, *Multiple Presentation Mosaic*, Master's thesis, University of Wisconsin-Milwaukee, May 1996. (Expected date of completion).
6. D. D. Cowan, E. W. Mackie, G. M. Pianosi, and G. de V. Smit, 'Rita — an editor and user interface for manipulating structured documents', *Electronic Publising*, **4**(3), 125–150, (September 1991).
7. Richard Furuta, Vincent Quint, and Jacques André, 'Interactively editing structured documents', *Electronic Publishing—Origination, Dissemination and Design*, **1**(1), 20–44, (April 1988).
8. Richard Keith Furuta, *An Integrated, but not Exact-Representation, Editor/ Formatter*, Ph.D. dissertation, University of Washington, 1986.
9. Vincent Quint and Irène Vatton, 'Grif: An interactive system for structured document manipulation', in *Text processing and document manipulation*, ed., J. C. van Vliet, pp. 200–213. Cambridge University Press, (April 1986).
10. Vincent Quint. The languages of Grif. Available by anonymous ftp from ftp.imag.fr in directory `/pub/OPERA/doc`, December 1993. Translated by Ethan V. Munson.
11. Louis Weitzman and Kent Wittenberg, 'Automatic presentation of multimedia documents using relational grammars', in *Proceedings of ACM Multimedia '94*, pp. 443–451. ACM Press, (October 1994).
12. James Clark. DSSSL Lite. Available on the World Wide Web at URL `http://www.jclark.com/dsssl/`, 1995.
13. Synex Information AB's home page. World Wide Web home page at URL `http://www.synex.se`, 1995.

14. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, ed., Charles F. Goldfarb, International Organization for Standardization, Geneva, Switzerland, 1986. International Standard ISO 8879.
15. Charles Donnelly and Richard Stallman, *Bison: The YACC-compatible Parser Generator*, Free Software Foundation, Cambridge, Massachusetts, version 1.20 edition, 1992.
16. Roy Goldman, 'Variable shape specification in Proteus'. Ensemble Project internal memorandum, November 1993.
17. Cécile Roisin and Irène Vatton, 'Formatting structured documents', Research Report 2044, INRIA, (September 1993).
18. Donald E. Knuth and Michael F. Plass, 'Breaking paragraphs into lines', *Software—Practice & Experience*, **11**(11), 1119–1184, (November 1982).