CMU-ITC-91-097

The Need for Style Systems in ODA

Mark SHERMAN *,
Jonathan ROSENBERG **, Ann MARKS *
and Jaap AKKERHUIS ***

* *Information Technology Center, Carnegie Mellon University, 4910 Forbes Ave., Pittsburgh, PA 15213, USA*
** *Bell Communications Research, 445 South Street, Room 2D-292, Morristown, NJ 07962-1910, USA*
*** *Mt. Xinu, Suite 312, 2560 Ninth Street, Berkeley, CA 94710, USA*

Many advanced document systems provide a formatting mechanism called 'style sheets'. Style sheets provide a great deal of flexibility in describing a document's format, and allow easy maintenance of different house styles for a collection of documents. In this paper, we describe the basics of general style sheet systems, argue that successful document interchange must include the exchange of style sheet information, and evaluate ODA's style mechanism against this requirement.

Keywords: ODA, Style sheets, Multimedia document interchange.

Mark Sherman is a Research Computer Scientist in the Information Technology Center at Carnegie Mellon University. He received his SB degrees from the Massachusetts Institute of Technology in 1977 and his PhD in Computer Science from Carnegie Mellon University in 1983. His primary interest is the application of graphical user interfaces to new domains.

Jonathan Rosenberg is a District Manager in the Multimedia Communications Research Division at Bellcore (Bell Communications Research). He received his BS degree from the University of Maryland in 1977 and his PhD in Computer Science from Carnegie Mellon University in 1983. His research interests center around multimedia systems in general, and particularly multimedia document architectures.

This excerpt is from an upcoming book, *Multi-media Document Translation: ODA and the EXPRES Project*, published by Springer-Verlag, Inc. Used with permission.

This work was performed while the authors were at Carnegie Mellon University. The work was supported in part by a joint project of Carnegie Mellon University and the IBM Corporation and in part by the National Science Foundation under contract ASC-8617695. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies of the IBM Corporation, the National Science Foundation, Carnegie Mellon University, Bell Communications Research or Mt. Xinu.

North-Holland
Computer Standards & Interfaces 11 (1990/91) 177–182

1. Introduction

In June 1986, the US National Science Foundation (NSF) solicited proposals for the Experimental Research in Electronic Submission (EXPRES) project. EXPRES was to focus on the electronic submission and processing of proposals to NSF, as well as to improve the ability of the United States' research community to interchange multimedia documents.

The authors (then all at Carnegie Mellon University) were one of a group of EXPRES participants who specified, designed and implemented document interchange systems based on ODA. The details of our project are fully documented elsewhere [8]. In this article, we wish to discuss one facet of our project: the use of style sheets in document production systems and the problems of interchanging style information. In particular, we consider how ODA can be used to exchange style sheet information. (We assume that the reader has a basic understanding of ODA. Good introductions can be found in [1–3,5,8].)

2. Manipulating Formatting Information

A document's organization reflects how the author structured the information. How the document appears on a page is called its presentation. (A presentation can be made on any imageable

Ann Marks is a System Scientist in the Information Technology Center at Carnegie Mellon University. She received her BS (1976), MEng (1977) and PhD (1980) degrees in Electrical Engineering from Cornell University. Her research interests are focused on the distribution of continuous-time media in distributed computer systems.

Jaap Akkerhuis is a Systems Developer with Mt. Xinu. He graduated from the Hogere Technische School with the Ing degree in 1975. His work involves the integration of document production systems with new environments.

medium, such as a screen, window or microfilm. In this paper, the word page is used to denote any imageable medium.) The process of creating a presentation is called formatting and is based on a specification of how to present various media as well as the document's organization. There are many ways to specify and manipulate formatting information.

It is common to separate the presentation of a document from its organization. At one extreme, SGML defines only the organization of information without specifying its presentation. An intermediate position intertwines the presentation and organization of a document. For example, early wordprocessing systems, such as runoff [4], did not differentiate substantively between the concepts of presentation and organization. At the other extreme, only the presentation of a document exists, such as in a PostScript representation.

Systems employ a variety of techniques for representing formatting information. In this section, several approaches to presenting formatting information are presented.

2.1. Embedded Commands

Early systems, such as runoff, required user-visible codes within the content to designate formatting actions. Frequently, these codes were defined with reference to a specific formatting model that was described in terms of line breaks, paragraph breaks and page breaks. (One can argue that a paragraph is a logical entity, but early systems defined such concepts only to allow paragraph specific formatting, such as indentation, to be performed. There was little else provided for document organization.)

Embedded-code systems originated in the days when document systems ran in batch mode on one machine and produced output intended for another machine or device (a display or printer, for example). With the proliferation of inexpensive machines with bitmapped displays, the concept of directly manipulating the formatting information became common. This is discussed in the next section.

2.2. Direct Manipulation

The embedded-code systems had the property that the editor of a document had to see the formatting commands in the content. With the ability to see the results of the formatting immediately on the display, a style of editing called WYSIWYG (What You See Is What You Get) became popular. Initial versions of the technique were similar to the embedded-code systems: a user designed some piece of text to be formatted along with some formatting instructions. The systems then displayed the properly formatted information.

Although the user was spared the sight of formatting codes in the content, the document system in fact performed the embedding. Thus, initial versions of WYSIWYG systems only hid the visibility of the formatting commands; the formatting information was still embedded in the content. Some contemporary systems essentially use the same technique.

2.3. Named Manipulation

In early embedded-code systems, and early WYSIWYG systems, many formatting commands were low-level. For example, both kinds of systems provided separate commands to center a line, choose a bold-face font and increase the size of a font. Users who wanted to make a heading for a report used all three commands to get a large, bold, centered title.

To relieve the user from having to specify three commands repetitively, interactive systems with macro facilities became popular. In such systems, users could name a formatting command, such as 'Make Title', and associate a list of other commands to execute when the new command was selected, such as large, bold, center.

However, these macro expansion systems lost the structure of the command application. The interpretation of the macro resulted in the low-level commands being applied to the document, not in a macro invocation being associated with the text. There is no difference in the document between applying the macro and executing the low-level commands individually. If the macro were to change – for example, if a title were to be made italic instead of bold – each application of the macro would have to be found by the user and changed manually.

2.4. Style Systems

Style systems evolved to allow users to define and maintain groups of formatting commands.

This allowed users to denote abstractly that a formatting operation should be performed on some part of a document, while allowing changes in the definition of the formatting operation. Style systems can be specified with codes in batch oriented systems (such as Scribe [10]) or in interactive WYSIWYG systems (such as Interleaf [6], Diamond [9] and Andrew [7]).

Style systems come in a variety of forms. To discuss the differences between style systems, we need first to define some terms. We can then describe the common restrictions placed on the general system.

2.4.1 Parts of Style Systems

Style systems control the definition and use of *formatting state vectors, styles* and *environments*. Each of these is defined and discussed in turn.

There is a host of formatting properties available in most document systems. These include such concepts as margins, font information, line spacing, raster densities and page orientation. This information is collected into a data structure called a formatting state vector. The value of this vector changes as the document is formatted. For example, as different parts of the document are formatted, the value for the left margin may be changed.

A style is a function that maps a formatting state vector into another formatting state vector. Most systems define a style (sometimes called style sheet, property sheet, font delta, ruler or attribute) as a set of rules that describe how to change individual components of the formatting state vector. A trivial example of a style rule is 'set the left margin to one inch'.

Many systems allow the composition of styles, that is, one style may be defined in terms of another. For example, the second-level-heading style can be defined as 'apply "first-level-heading" style and then make the font bold.' A style that is defined in terms of another is called an *inherited style*. Some systems allow more complicated calculations in their styles, such as references to current values in the formatting state vector. One such example is 'set the left margin so that it is 4 inches to the left of the right margin.' Styles that refer to the current value of the formatting state vector in setting new values are called *relative styles*. Styles that do not refer to the current value of the formatting state vector in setting new values are called *absolute styles*.

A style may be associated with several regions of a document, for example, with several different paragraphs. Each such region is called an environment, and has an associated value for a formatting state vector. The value for the formatting state vector is calculated as explained below.

The initial value of the formatting state vector is called its root value. As a document is formatted, the formatting system encounters environments with associated styles. The formatting system applies the style to the current value of the formatting state vector in order to produce the new value of the formatting state vector. The result is the formatting state vector associated with the environment. The part of a document associated with the environment is called the *scope* of the environment, or less precisely, the scope of the style. For completeness, a root environment is associated with the root value of the formatting state vector and has the scope of the entire document.

As specified either by the style or by the individual document system, changes to a formatting state vector made by a style have either local or global effects.

A local effect takes place only within the scope of the environment. The value of the formatting state vector after the scope of this environment reverts to its value before the style was applied. For example, a style called 'indent' may specify a left and right margin of two inches. Let us assume that before the 'indent' style was encountered, the left and right margins are one inch, and that 'indent' is applied to a paragraph. When the environment for the paragraph is entered, the values for the left and right margins are changed to two inches, as specified by the 'indent' style. When the paragraph is finished, the values of the left and right margin revert back to their values before the 'indent' style was applied, that is, one inch. Previous and subsequent paragraphs are beyond the scope of the application of 'indent' and will use whatever margin values are specified in the formatting state vector before 'indent' was applied.

If a global change is specified, then all formatting state vectors in subsequent environments of the document will be changed. For example, changes to the page header component of the formatting state vector are usually global – the

specified page header remains in effect until a new page header is specified. If our previous example had specified that the change to the left margin were global, then the value of the left margin component of the formatting state vector beyond the end of the paragraph would have remained at two inches instead of reverting back to one inch.

2.4.2 Restrictions on Style Systems

Our discussion has described a general, unrestricted style system. The style systems provided on contemporary document systems contain a variety of restrictions. Some of these restrictions are considered here.

The most common restriction is that environments must be associated with only particular parts of a document's organization. For example, only paragraphs or titles of a document may have a style applied. In the model presented before, an environment (or style application) could be applied to any contiguous region of the document. Most systems are not this flexible: for example, an environment may not start in the middle of one paragraph and end in the middle of another.

Another common practice is to require an environment for every part of a document's organization. For example, the system may require that every paragraph have an associated style – the system will generate an anonymous style for a paragraph if the user does not provide one. This contrasts with the general model that assumes that current values of the formatting state vector are used if no environment boundaries are specified.

Many document systems partition the features of a style system to enforce a particular kind of functionality for a class of styles. One example is partitioning the formatting state vector components so that different classes of styles manipulate different components. Systems typically provide different names for the different partitions. In some systems, for example, styles that may affect the margin and indentation components of the formatting state vector are called *property sheets* while styles that affect font characteristics are called *deltas*. Frequently, the partitioning of the formatting state vector components is used in conjunction with another partitioning: limiting the style's association with document parts. For example, property sheet styles may be applied only to document structures that are larger than a paragraph while deltas may be applied only to document structures that are smaller than a paragraph.

Style systems have really provided simple programming languages for users to describe the formatting of their documents. Thus, style systems have provided a great deal of flexibility and functionality to document systems, but have also made the problem of interchange more difficult. In the next section, we look more closely at the issue of interchanging multimedia documents.

3. Translating Documents

When translating a document created on one system to a document that will be viewed and edited on another system, it is necessary to decide the kind of fidelity desired. The simplest kind of fidelity is *hardcopy fidelity*: the document should look the same when printed or imaged on both systems. The next level of fidelity is *content fidelity*. If the same abstract text, raster images and other content can be translated from one system to the other, then we have achieved content fidelity. Beyond content fidelity is *structural fidelity*, in which we are interested in retaining the organization of the document in addition to its contents. The highest level of fidelity that we define is *editing fidelity*, which allows the recipients to perform the same kinds of document manipulation that the sender could. This is particularly important for EXPRES, because we are concerned with allowing collaboration on the construction of a multimedia document among people using dissimilar systems.

Our most compelling example of an editing feature to be retained across translations is style information. However, as described before, style systems vary greatly among document systems. The structure of styles, the components available, the equations available to define the style and the possible ranges of environments differ substantially. However style systems are represented in a document, they are the most general way of specifying formatting and editing that we are considering. Therefore, if we can preserve the information present in a style system, we will preserve the lesser amount of information present in the simpler formatting definition schemes.

4. Experience Using ODA

This section discusses some of our experiences using ODA as an interchange medium between document processing systems. The EXPRES participants from Carnegie Mellon University, the University of Michigan and McDonnell Douglas Corporation connected their systems together and interchanged documents at several demonstrations. In this section, we discuss each of the participating systems and translators and the effectiveness of using ODA for the interchange.

4.1. EXPRES Document Systems

The systems shown in interchange demonstrations were the Andrew Toolkit, Diamond, Interleaf and troff. The Andrew Toolkit is a multimedia development system built at Carnegie Mellon University, Diamond is a multimedia document system built at Bolt, Beranek and Newman, Incorporated. (Variants of the Diamond system are also known by the names UM EXPRES and Slate.) Interleaf is commercial document production system supporting several media, and troff is the standard document formatting system provided on Unix. Andrew, Diamond and Interleaf provide style-sheet based, WYSIWYG-style editors, while troff is a batch-oriented, embedded-code system.

4.2. Experiences with Interchange

By performing interchanges, we found several classes of problems that lowered the fidelity of the translation of a document from one system to another. Different style systems were the most pervasive mismatch we found. Nearly every system had some notion of bundling formatting information into a style, but the differences between them made interoperation among style systems very difficult. Some of the differences we found involved structure between styles, types of styles and applicability of styles. We elaborate on each of these differences in turn.

4.2.1 Structure among Styles

The relationships among styles offered by systems vary a great deal. ODA provides a paired, flat style system, where styles are grouped into pairs, a layout style and a presentation style, but where there is no relationship between any two styles. Andrew also has no structure on styles. However, Diamond and Interleaf allow one style to be defined in terms of another, adding structure to the styles. We found no good way to interchange this information in our demonstrations.

4.2.2 Partitioning of Style Function

We were not able to preserve the partitioning of function among styles. In representations like ODA, the fact that style functions are partitioned into different classes (layout and presentation styles) does not matter, since any particular function can fit in only one place. However, systems like Diamond allow the same kind of editing change to be in many different style classes. For example, a font change can be made in a global style, a property sheet or a font delta. Therefore, when translating between Diamond and ODA, it may not be possible to know which of the various kinds of styles should be used to represent some change to the formatting state vector.

4.2.3 Partitioning of Style Application

A third problem we had when interchanging style information was style applicability. Part of a style's specification is where it may be used in a document. Some systems, like Andrew, permit any style to be applied anywhere, assuming the applications (lexically) nest. Some systems permit styles to be applied only to a paragraph or only within paragraphs. Like the partitioning of style function, we found no good way to exchange information about partitioning of style application.

4.3 Attribute Structure

We found several problems with attribute structure. The one that affected style sheet interchange most was simply that the structure was not fine grained enough. The introduction of independently defaultable parameters to ODA was a start towards fixing the problems, but solved some problems by introducing more mechanism, and hence more implementation. Unfortunately, independently defaultable parameters do not go far enough. For example, the inability to specify individual parts of a font definition caused extraneous font definitions to be generated. As we explained elsewhere [8], the inability to separate subparameters of attributes caused us to introduce unneces-

sary style constituents into the ODA structure, thereby reducing the fidelity of the style sheet interchange.

5. Conclusions

The success of a document interchange mechanism will be determined ultimately by the level of translation fidelity that is attained. The EXPRES participants were aiming for a high level of fidelity, which retained not only the formatting information associated with a document, but also its organization and editability. We believe that the retention of this information is essential for useful interchange. We were able to exchange only the simplest form of style information, partly because the participating systems had substantially different style systems, and partly because ODA provides limited style sheet expression: there are no relative styles, there is no style sheet structure, single ODA attributes represent multiple style sheet components and there is a strong connection between a document's structure and style sheet evaluation. Some of these problems are being addressed by the standards community, but until they are resolved, the editing fidelity of interchanged multimedia documents will be limited.

References

[1] D. Ansen, Document architecture standards evolution (Office Document Architecture for structuring and encoding documents), *AT&T Tech. J.* 68 (4) (1989) 33–55.

[2] I. Campbell-Grant. 'Introducing ODA', *Comput. Standards Interfaces* 11 (1990/91) 149–157.

[3] F. Dawson and F. Nielsen, ODA and document interchange, (Office Document Architecture standard), *UNIX Rev.* (3) (1990) 50–57.

[4] DEC Standard Runoff (DSR) User's Guide, Digital Equipment Corporation, Maynard, Ma, 1979.

[5] R. Hunter, P. Kaijser and F.H. Nielsen, ODA: a document architecture for open systems. (Office Document Architecture), *Comput. Commun.* 12 (2) (1989) 69–80.

[6] R.A. Morris, Is What You See Enough to Get?: a description of the Interleaf publishing system, PROTEXT II, *Proc. Second Internat. Conf. on Text Processing Systems*, (1985) 56–81.

[7] A.J. Palay, W.J. Hansen, M. Sherman, M.G. Wadlow, T.P. Neuendorffer, Z. Stern, M Bader and Th. Peters, The Andrew Toolkit – an overview, *Proc. USENIX Winter Conf.* (1988) 9–21.

[8] J. Rosenberg, M. Sherman, A. Marks and J. Akkerhuis, *Multi-media Document Translation: ODA and the EXPRES Project* (Springer, New York, 1991).

[9] R.H. Thomas, H.C. Forsdick, T.R. Crowley, R. W. Schaaf, R.S. Tomlinson and V.M. Travers, Diamond: a multimedia message system built on a distibuted architecture, *IEEE Comput.* 18 (12) (1985) 65–78.

[10] Scribe document production software (User Manual), Unilogic, Unilogic, Ltd., Pittsburgh, PA., June 1985.