ELECTRONIC PUBLISHING, VOL. 6(4), 423–434 (DECEMBER 1993)

Mediating interface between hypertext and structured documents

KOICHI HAYASHI AND AKIFUMI SEKIJIMA

Systems and Communications Laboratory, Fuji Xerox Co., Ltd.
YBP East Tower, 134 Godocho, Hodogaya-Ku, Yokohama-Shi
Kanagawa 240, Japan

email: hayashi@rst.fujixerox.co.jp

SUMMARY

In this paper we describe a unified document model for an authoring system that takes advantage of both hypertext and structured document models: hypertext, which represents a document as a network of information fragments freely referencing one another, helps users create ideas; and structured document models, which represent a document as a rigid tree structure of document components, help users organize documents and make layouts. Our document model comprises the underlying structure and the surface: the underlying structure is a network structure; and the surface is an interface providing a view of the underlying structure. The key features of our document model are: (1) the surface defines tree structures as marked parts of the underlying structure, and maintains consistency between the network and tree structures; (2) the surface monitors users' walks in the underlying network and marks the trails to define tree structures; and (3) the defined tree structures work as structured documents. Nelumbo, a prototype system, integrates different types of editors that handle features of hypertext and structured documents. Users can choose any of the tools at will, and editing with the tools affects the underlying structure consistently.

KEY WORDS Hypertext Structured documents Authoring system Writing process model Document model History tree Multi-level formatting

1 INTRODUCTION

Recent advances in computer technology have drastically changed the document preparation process. Authoring systems now cover a wide spectrum, from idea creation at the initial stage, to formatting and printing at the final stage. At the initial stage, authors can use hypertext applications to store ideas. Hypertext systems which hold information as flexible network structures enable authors to store thoughts that have yet to take shape [1,2]. At the final stage, authors can use formatters, WYSIWYG editors, or, as the best choice, structured document systems. Structured document models, which manage tree logical structures, enable authors to organize documents from a logical viewpoint and to get sophisticated layout of the documents [3].

Writing Environment [4] initially described the fundamental framework for supporting the above writing process. This framework deals with both a network structure and a tree structure, and assumes a standard writing course: an author first develops a hypertext

CCC 0894–3982/93/040423–12
Received 15 August 1993
Revised 1 December 1993

network by storing and connecting information, then organizes a tree structure with nodes extracted from the network, and finally formats the tree structure with layout styles. This approach is widely accepted and still being enhanced by current commercial systems, such as INSPIRATION [5]. However, the irreversible writing course does not always suffice for authors to complete documents. On the contrary, formatted documents often highlight the necessity for revising documents, and document revisions often require reconstruction of outlines or further creation of ideas. The essential drawback of this framework is that it scarcely allows deviation from the desired course: it makes users handle different structures that represent a document at different stages, but does not maintain consistency between the structures.

Here, we describe a unified document model for an authoring system that focuses on the stage of drafting and revising. The key ideas are three-fold: First, our document model comprises *the underlying structure* and *the surface*. The underlying structure is a free form hypertext, which is a unique representation of a document in which every operation affects changes. The surface is an interface providing a view of an underlying structure. The surface holds tree structures as marked parts of the underlying structure and maintains consistency between the network and tree structures. Second, the surface provides *the implicit and explicit marking mechanisms* to select tree structures from a network structure: the system monitors users' walks in the underlying network and marks the trails to define tree structures on the surface; users can further manually arrange the tree structures on the surface. Third, a tree structure on the surface is not a simple tree, but a document structure that has a heterogeneous feature of `tnt` [6]. With *the multi-level formatting* techniques [7,8], the surface provides utility of sophisticated formatting and WYSIWYG editing.

We are presently developing a prototype system *Nelumbo*[1], which integrates different types of editors to manage the unified document model. It provides the card-like editor called *field editor* for editing the underlying network structure, WYSIWYG editor for editing a document structure on the surface, and *surface browser* for managing structures on the surface. Users can choose any of the tools at will, and editing with the tools affects the underlying structure consistently.

2 RELATED WORK

To facilitate writing of documents is one of the major objectives of early hypertext systems like Neptune [9] and Guide [10]; they mainly manage tree structures to represent the organization of documents. Additionally, hypertext systems like NoteCards [11] and gIBIS [12] handle network structures that represent knowledge and arguments. Recently, structured documents, which have handled tree logical structures of documents, are also being extended to cover network structures like HyperODA [13]. Although they can manage both tree and network structures, they have no methodology that helps users convert a network of ideas into a tree document structure.

WE [4] should be recognized as the first system that introduces an explicit human writing process model. It provides tools supporting steps in the writing process where an author converts a network into a hierarchy, then to a linear structure for printing. Since WE assumes the irreversible authoring process, revisions on a tree structure do not affect the original network structure. INSPIRATION [5] is a commercial authoring system that

[1] Nelumbo is a water lily native to Asia, which creates open flowers on the surface.

adopts a similar authoring process. It can automatically convert a network structure into a tree structure, and vice versa. Although the automatic conversion maintains consistency between both structures in most cases, it sometimes creates structures users do not expect.

SEPIA [14,15] introduces a distinctly improved writing process model. This model does not assume a predetermined course any more. Instead, it allows users to traverse freely through different *activity spaces*, which are designed to support different activities involved in the writing process. The activity space that covers the creation of structured documents, which is the purpose of our model, is the *rhetorical space*. The rhetorical space provides *construction kits* [16] to aid users creating final documents that are user ready. As final documents, users can create hypermedia document structures as well as conventional document structures. The construction kits involve *the content part*, which represents a network of contents, and *the organizational part*, which structures views of the content part. Therefore, users can make different views on the same contents network by redefining the organizational structure. The unique value of our model is that it provides a mechanism that dynamically constructs an organizational structure, the surface, on a content network, the underlying structure.

To help in organizing tree document structures, we have adopted a mechanism that records users' access histories as tree forms. Such a history mechanism was initially proposed by Foss [17] as one of a set of techniques for solving the disorientation problem. Foss shows *history trees* in a browser so that users can view their access histories and backtrack to the information nodes that they have visited. We have applied this mechanism to form a tree structure that is not a simple access record, but a structured document.

A structured document defined on the surface has a heterogeneous feature of `tnt`s [6]. A `tnt` is an extended version of a tree structure that can represent a nest of heterogeneous content portions, such as graphics in a table and mathematical expressions in text. While a conventional document structure is a tree whose leaves are content portions, a `tnt` further allows tree document structures to be embedded in a content portion. We defined the underlying structure by breaking down a tree document structure of a `tnt` into smaller fragments named *fields*, and extending them to form a network. Therefore, when an underlying structure forms a tree structure, it can be regarded as a `tnt`.

We follow the *multi-level formatting model* using *formatter hierarchies* [8] to format the organization of documents. Initially, a multi-level formatting model was proposed for implementing ODA-based WYSIWYG editors [7]. The model of the formatter hierarchy generalizes the formatting mechanism for structured documents including `tnt`s and shows mechanisms for parallel formatting. When formatting a document, this model generates *a formatter hierarchy*, which is a tree structure of *formatters*: a formatter executes a partial formatting process for a fragment of structures; formatters process formatting interactively, and a formatter hierarchy as a whole executes formatting of the entire document structure. Practically, this model is used for implementation of formatting functions on Akane [18], a commercially available structured document editor. Theoretically, Shin [19] shows that the layout control capability of this model is equivalent to *the generic structures* of ODA.

3 DOCUMENT MODEL

In this section we describe the two main components of our document model: *the underlying structure* and *the surface*.

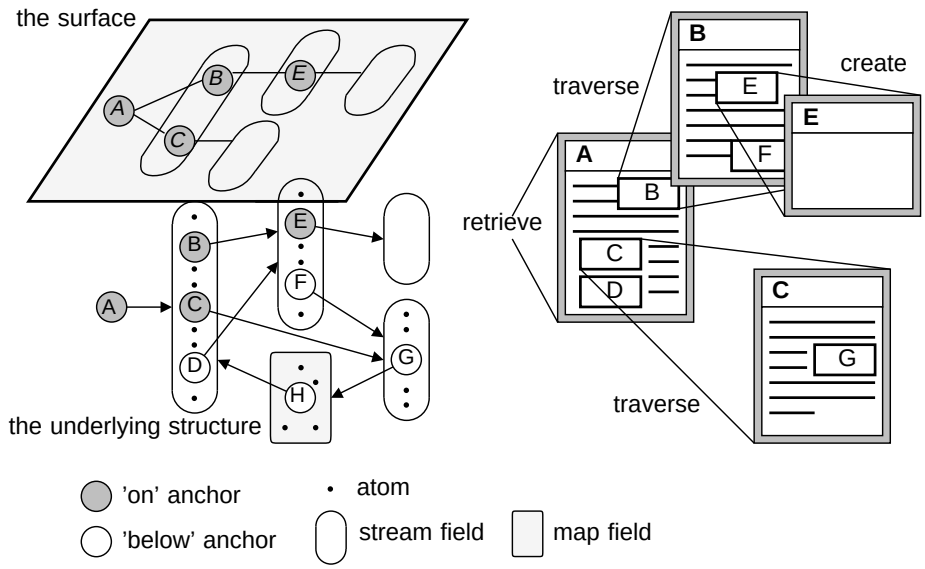

Figure 1. The underlying structure, surface, and editing sessions

3.1 The underlying structure

The underlying structure is a unique representation of a document, in which every operation affects changes. The underlying structure is a network structure that comprises *atoms*, *fields*, and *anchors* (see Figure 1).

Atom: An atom is an anchor (see below) or atomic object that can be displayed. Graphic characters, raster graphics elements, geometric graphics elements are all atoms.
Any atom other than naked anchors (see below) must be included in certain fields.

Field: A field is a set of one or more elements, each of which is an association of an atom and its position within the unique *logical space*. Atoms in a field are distinguished by their associated positions.
A field corresponds to a unit of contents structured in accordance with their intrinsic *logical structure*. An atom in a field is an element of the contents that the field represents, and the associated position defines the role in the logical structure.
Our model supports three types of logical spaces, *ordered set*, *coordinated plane*, and *lattice plane*, which represent different types of contents. Additionally, the types of logical spaces define types of fields: A field associated with a logical space of ordered set (coordinated plane, lattice plane, resp.) is referred to as the *stream* (*map*, *lattice*, resp.) type.

Stream field: A stream field is a set of elements of the form (o,i), where o is an atom which is either a graphic character or an anchor, and i is a positive integer. The position i specifies the order of the atom in the ordered set associated with the field.
Stream fields are used to represent text contents. A graphic character with the position i is the character which appears at the i-th position in the text that the field represents.

Map field: A map field is a set of elements of the form $(o,(x,y))$, where o is an arbitrary atom and x,y are real numbers. The position (x,y) specifies the coordinate pair of a point according to the two-dimensional coordinate system of the coordinated plane associated with the field. The atom is positioned at the point that the coordinate pair indicates.

Map fields are used to represent raster graphics or geometric graphics. A raster graphic element or a geometric graphic element with the position (x,y) is a pixel or a geometric primitive appearing at the point (x,y).

Lattice field: A lattice field is a set of elements of the form $(o,((i,j),(k,l)))$, where o is an anchor and i,j,k,l are non-negative integers. Both coordinate pairs (i,j) and (k,l) specify two lattice points within the two-dimensional lattice associated with the field. These two lattice points specify a pair of diagonally opposite vertices of a rectangle.

Lattice fields are used to represent tabular contents. An anchor with the position $((i,j),(k,l))$ references the contents that appear within the table cell edged with the i-th and k-th vertical table rules and j-th and l-th horizontal table rules.

Any field must be referenced by at least one anchor.

Anchor: An anchor is a particular kind of atom that references a field. By means of anchors, fields can be related to one another.

Thus, anchors and fields form an arbitrary directed (multi-)graph[2]: a field corresponds to a node; an anchor corresponds to a directed arc from the field in that the anchor is embedded to the field which it references; more than one anchors may reference a single field; a single field may include an arbitrary number of anchors; cyclic paths are allowed.

An anchor does not need to be embedded in any field, while a field must be referenced by at least one anchor. Such an anchor is called a *naked anchor*. When the user begins constructing the underlying structure of a document, a naked anchor is produced first.

Fields and anchors correspond to *nodes* and *links* of the conventional hypertext model. The essential difference between our model and the conventional model is that a field describes a relation among anchors (and atoms) while a link describes a relation among nodes. Further, fields and anchors correspond to *content blocks* and *transition nodes* of `tnt` documents, which allow document structures within contents. When a directed graph consisting of fields and anchors forms a tree, i.e., only one root anchor exists and a unique directed path exists to an arbitrary field from the root anchor, they are naturally regarded as a `tnt` document.

3.2 The surface

The surface is an interface that provides an organizational view of the underlying structure of a document. The key role of the surface is management of trees extracted from the underlying structure of a document, which represents, in contrast, free connections of fields by means of anchors. These trees form the organization (logical structures) of the document.

Through the surface, the user can manipulate the underlying structure of a document and can obtain perceptible display of the organization in various ways.

[2] A multi-graph is defined as a graph allowing more than one arc to relate to the same two nodes

3.2.1 States and organization

Every anchor in the underlying structure of a document has a state, either *on the surface* or *below the surface* (*on* or *below* for short). The 'on' state of an anchor means that the anchor and field referenced by the anchor are components of the organization of the document; the 'below' state of an anchor means that the anchor is not a component of the organization of the document. In other words, 'on' anchors and fields referenced by the 'on' anchors form the organization of the document.

The organization of a document is constrained so that it shall be a union of one or more trees. That is, one or more 'on' anchors exist which are not embedded in any fields in the organization (called *root anchors*), and further, a unique path exists to an arbitrary field in the organization from one of the root anchors. Consequently, when a single field is referenced by more than one anchor, only one, at most, can take the 'on' state. This constraint comes from the assumption that each fragment of information should be reached by following a unique context in the organization of a document.

3.2.2 Operations

The user manipulates the underlying structure of a document through the surface. For this purpose, the surface provides several operations.

Create anchor: This operation creates a new anchor occupying a position within a field in the organization or a new naked anchor. The newly generated anchor may reference some existing field in the organization or a new field that is created at the same time as the anchor. In the former case, the state of the new anchor becomes 'below', while, in the latter case, it becomes 'on'.

Delete anchor: This operation deletes an anchor from the underlying structure. If the anchor to be deleted references a field that is not referenced by any other anchors, the operation also deletes the field referenced. If the field includes anchors, all the embedded anchors become naked anchors.

Raise anchor: This operation makes the state of an anchor 'on', and puts the anchor in the organization of a document.

Sink anchor: This operation makes the state of an anchor 'below', and puts the anchor off the organization of a document.

The above covers basic operations related to manipulation of anchors. For example, to move an anchor from one field to another, the user has to 'create' a new anchor referencing the same field that the concerned anchor references, within the latter field, and 'delete' the anchor within the former field.

In addition to these operations, a number of operations to edit the contents within an individual field should be supplied: to insert a new atom into the field at a certain position; to delete a certain atom from the field; to change the position of a certain atom in the field, and so forth.

3.2.3 Formatter and layout styles

For formatting the organization of a document, we introduce the *layout structure*, *formatter*, and *formatter hierarchy* on the basis of the multi-level formatting model [8]. A layout

structure is a tree structure composed of layout objects, such as pages, frames and blocks. A formatter is defined for each 'on' anchor, and used for creating a subtree of the layout objects that represents a perceptible display of the associated field. To format a tree logical structure, the formatters form a hierarchical structure called a formatter hierarchy.

Each formatter in the formatter hierarchy receives space information called *available area* from a superior formatter and generates a subtree of layout objects that fits in the receiving area and calculates the available area for subordinate formatters. As the result of the execution, each formatter returns a subtree composed of blocks associated with the field and the layout subtrees returned from the subordinate formatters. The types of fields determine the function of a formatter: the layout subtrees returned from the subordinate formatters are embedded at the position of the anchor in the associated logical space. As formatting proceeds, the available area is consumed. When the available area is inadequate, the formatter attached to the root anchor generates a new page to increase the available area.

To control the formatting process, users specify two types of layout styles: *the local layout style* and *the global layout style*. The local layout style corresponds to the *layout template* defined in the multi-level formatting model: it describes rules to create layout objects and calculate available areas. Every anchor has a local layout style and the associated formatter uses this style. A global layout style describes the layout of a whole document, such as page size, text area in a page, and folio positions. Only the formatters attached to root anchors use this style. Since root anchors of the organization are determined dynamically, the global layout style is attached to the surface.

We apply the above mechanism to control creation of windows of editors. Different editors operate formatters and use their results in different ways: a card editor uses a formatter to display a field as a window ignoring the layout styles; while a WYSIWYG editor operates the formatters that form a formatter hierarchy with the global layout style and local layout styles.

3.2.4 Interactive marking mechanism

To aid users in making the organization of a document, the surface has *the explicit and implicit marking mechanisms*. The explicit mechanism allows users to change states of anchors with 'raise' and 'sink' operations. The user may define a number of small trees that show different parts of a document, or define the tree that covers the whole information in the document. Therefore, the surface represents every stage of the varying process in making the organization of documents.

The implicit mechanism puts an anchor 'on' the surface when the associated formatter is invoked. When users open a window to see and edit a field, the associated anchor becomes 'on'. Figure 1 shows an example of a series of editing sessions using the implicit marking mechanism.

1. *Retrieve an anchor.* The user retrieves anchor 'A' by its name and opens a window that displays the field referenced by the retrieved anchor. In this case, retrieved anchor 'A' becomes 'on' and a root anchor of a tree.
2. *Traverse an anchor.* The user traverses an anchor by clicking anchor button 'B' within the editor of field 'A' and opens a window that displays the field referenced by the accessed anchor. In this case, accessed anchor 'B' becomes 'on' and subordinate to existing anchor 'A'.

3. *Create an anchor.* The user creates anchor ‘E’ and associated field ‘E’, inserts the new anchor into existing field ‘B’, and opens a window that displays the new field. In this case, new anchor ‘E’ becomes ‘on’ and subordinate to existing anchor ‘B’.
4. *Format a document.* The user gets the layout of a document by invoking the formatter hierarchy associated with a tree on the surface. Formatters for anchor ‘A’, ‘B’, ‘C’, and ‘E’ process formatting interactively and generate the layout of the associated tree logical structure.

We have designed the implicit marking mechanism assuming that the information fragments accessed by the user should be components of the organization of the document. When implicitly defined trees are not satisfiable, the user explicitly rearranges them.

4 NELUMBO

In this section we describe *Nelumbo*, which is a prototype authoring system based on our document model. Nelumbo is implemented in Objectworks Smalltalk release 4.1. We describe the main components of Nelumbo: *the field editor*, *WYSIWYG editor*, and *surface browser*. Figure 2 shows three field editors, a WYSIWYG editor, and a surface browser.

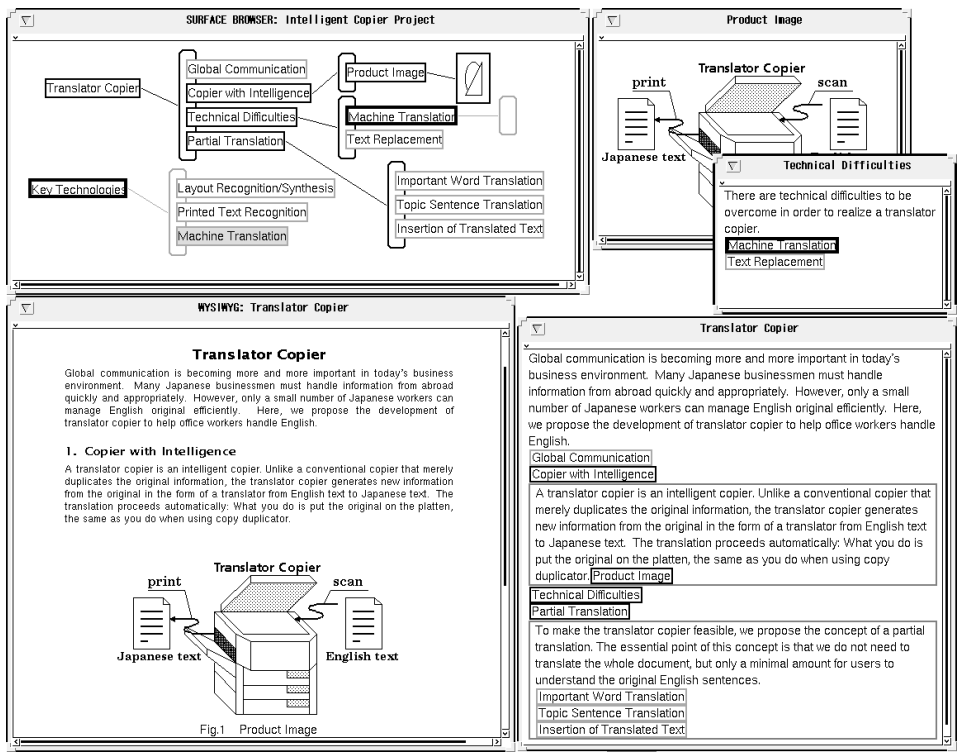

Figure 2. Overview of Nelumbo

4.1 User interface for anchor manipulation

The field editor and surface browser provide a common user interface that allow users to manipulate anchors: an anchor button indicates the name and state of the corresponding anchor; the user can point to the anchor button directly with the mouse and apply operations like 'raise' and 'sink'. As we described in the previous section, the surface introduces two states of anchors: 'on the surface' and 'below the surface'. Nelumbo further refines these states, allowing the user to effectively manipulate the anchor.

On the surface: This state has two sub-states according to the types of marking mechanisms. Since results of the explicit marking implies stronger intention of the user than results of the implicit marking.

- **Raised:** A 'raised' anchor is an anchor that the users have put 'on' the surface with the 'raise' operation. A button with a thick black frame indicates this sub-state, like the anchor button 'Key Technologies' in Figure 2.
- **Touched:** A 'touched' anchor is an anchor that the implicit mechanism has put 'on' the surface. A button with a thin black frame indicates this sub-state, like the anchor button 'Translator Copier'.

Below the surface: This state has three sub-states according to levels of accessibility, since some anchors of this state are displayed for enabling users to access them.

- **Active:** An 'active' anchor is a 'below' anchor that is displayed as a button in a window, and the user can manipulate it. A button with a gray frame indicates this sub-state, like the anchor button 'Global Communication'.
- **Inactive:** An 'inactive' anchor is a 'below' anchor that is displayed as a button in a window, but the user cannot manipulate it. A button with a gray mask indicates this sub-state, like the anchor button 'Machine Translation' subordinate to 'Key Technologies'. This sub-state comes from the constraint: when a single field is referenced by more than one anchor, only one, at most, may take the 'on' state. An 'inactive' anchor references the field that another 'on' anchor has already referenced. An 'inactive' anchor becomes 'active' when the user 'sink' the associated 'on' anchor, i.e., 'Machine Translation' subordinate to 'Technical Difficulties' in this example.
- **Invisible:** An 'invisible' anchor is a 'below' anchor that is not displayed in any window and, thus, the user cannot access it. Since anchors of the other four sub-states are displayed in windows, we generally call them 'visible' anchors.

4.2 Editors

4.2.1 Field editor

The field editor is a tool for editing the contents of a field. A field editor has a window that the associated formatter generates for displaying the field. The formatter places anchor buttons in the display at the specified position with the associated logical space. The user invokes new field editors by clicking the anchor buttons. By controlling interaction among formatters, the user defines how these editors behave as a whole: when the user locates the

invoked editor's window outside the original editor's domain, they behave as independent card editors; when the user locates the invoked editor's window inside the original editor's, they behave as an outline editor. In either case, invoking field editors triggers the implicit marking mechanism.

4.2.2 Surface browser

The surface browser is a tool for managing the organization defined on the surface. A surface browser displays the organization of a document and the 'visible' anchors from the fields in the organization.

A surface browser shows anchors as buttons, fields as patterns enclosing buttons, and super-subordinate relations as lines connecting the buttons and patterns. The type of logical space associated with the field determines the shape of the field pattern. Further, within the field pattern, the included anchor buttons are placed so that users can understand the positions of the anchors within the field. The field pattern also indicates whether the associated field editor is displaying the field. Since the 'touched' sub-state lasts after closing the editor, a 'touched' anchor does not always imply that the associated field is displayed. When a field is displayed, the associated field pattern is drawn with a black line; when not displayed, with a gray line.

A surface browser displays the current status of the organization of a document: when the status of anchors changes, the display of the surface browser changes accordingly. For example, when an 'active' anchor is put 'on' the surface, the associated button changes its appearance, and the referenced field appears within the display; when an 'on' anchor is put 'below', the associated button changes appearance, and the referenced field disappears. As the user walks in the underlying network, the implicit marking mechanism grows trees in the display of the surface browser. When users do not satisfy the implicitly generated organization of the document, they explicitly rearrange it with the 'raise' and 'sink' operations on the surface browser.

4.2.3 WYSIWYG editor

The WYSIWYG editor is a tool for displaying and editing any part of the organization of a document. When the user specifies an 'on' anchor and applies the *layout* command to it, the system makes the formatters execute this at each 'on' anchor in the tree that is subordinate to the specified anchor. These formatters form a formatter hierarchy, whose root formatter is the one associated with the specified anchor and formats the contents of their associated fields interacting with one another. The result of the formatting is displayed within a WYSIWYG editor and may be printed out. Figure 2 shows the result of applying the *layout* command to the anchor 'Translator Copier'.

What a WYSIWYG editor displays is not necessarily the final version of a document, but a snapshot at any point in the writing process. Additionally, more than one WYSIWYG editor can display different parts of the organization of a document. With WYSIWYG editors, users can edit the contents of fields. Changes on WYSIWYG editors affect the underlying structure; conversely, changes on the underlying structure affect displays of WYSIWYG editors.

5 FUTURE WORK

The document model we have described is intended to take advantage of both hypertext and structured documents. As structured documents, it benefits from such latest studies as `tnt` and multi-level formatting, but does not have generic structures like DTDs of SGML. These generic structures are widely used for guiding users to make specific structures that satisfy the requirements for various purposes, for example, for applying a corporate house-style to maintain uniform appearance. To allow users to edit a document freely, we have avoided imposing such global restrictions on users. Instead, we need a mechanism that specifies appropriate layout styles to anchors when necessary.

As hypertext, our document model provides only simple binary links for navigation and embedding. To exploit the rich potential of hypertext, we need to support more sophisticated structuring functions such as *n-array links*, which are found in systems like Aquanet [20]. As an alternative of n-array links, we are now studying the introduction of a new type of field that is defined as a combination of attributes.

We have assumed that each fragment of information in the underlying structure should be reached by following a unique context in the organization, therefore we introduced the 'inactive' sub-state to make the structure satisfy this constraint. Although this constraint may be restrictive in some cases, it ensures that the marked anchors always form tree structures. Thus, if we simply remove this constraint, conflicts occur: when defining more than one tree that share the same fields, the tree B(EF) and D(F(G)) in Figure 1, for example, anchors included in shared fields, anchor 'E', must take both 'on' and 'below' at the same time. One solution to avoid such conflicts is to introduce a mechanism for managing multi-states according to paths. This mechanism allows an anchor to hold more than one state, each of which is associated with a path to reach the anchor: anchor 'E' is treated as 'on' the surface when the trailing path is 'B-E', while it is treated as 'below' the surface when the path is 'D-E'.

6 CONCLUSION

Formatted linear documents still play an important role in today's society, not only because they can be printed on paper, but because they can convey information coherently to readers. Even though hypertext can represent more sophisticated structures of information, techniques to extract linear documents are truly indispensable. We believe that automatic conversion mechanisms do not satisfy users' requirements for linear documents. Thus, we have enabled users to control the whole extraction process; and, to aid users, provided a mechanism that maintains tree structures automatically defined from editing sessions. What we have described in this paper are techniques for building structured document views on hypertext networks.

ACKNOWLEDGEMENTS

We would like to thank Kil-Ho Shin for his cooperation during the entire process of writing this paper. He gave us valuable advice for formalizing the document model. We also thank Murata Makoto, Noriyuki Kamibayashi, and the other members of Systems and Communications Research Laboratory for fruitful discussions. The description of the translator copier is based on inventions from a project directed by Yoshifumi Matsunaga.

REFERENCES

1. J. Conklin, 'Hypertext: An Introduction and Survey', *IEEE Computer*, **20**(9), 17–41, (1987).
2. *Hypertext and Hypermedia*, ed., J. Nielsen, Academic Press, Orlando, FL, 1990.
3. *Structured Documents*, eds., J. André, R. Furuta, and V. Quint, The Cambridge series on electronic publishing, 2, Cambridge University Press, Cambridge, UK, 1989.
4. J.B. Smith, S.F. Weiss, and G.J. Ferguson, 'A Hypertext Writing Environment and its Cognitive Basis', in *Proceedings of Hypertext'87*, 195–214, ACM Press, New York, (1987).
5. Ceres Software, *INSPIRATION Owner's Manual*, 1991.
6. R. Furuta, 'Concepts and Models for Structured Documents', in *Structured Documents*, eds., J. André, R. Furuta, and V. Quint, 7–38, Cambridge University Press, Cambridge, UK, (1989).
7. P. Pederson, 'Streams and the Layout Process for Formatted-processable Documents, ISO/IEC JTC1/SC18/WG3/N1408', Technical report, International Standard Organization, (1989).
8. M. Murata and K. Hayashi, 'Formatter Hierarchy for Structured Documents', in *Proceedings of Electronic Publishing, 1992 (EP92)*, eds., C. Vanoirbeek and G. Coray, 77–94, Cambridge University Press, Cambridge, UK, (1992).
9. N.M. Delisle and M.D. Schwartz, 'Contexts — A Partitioning Concept for Hypertext', *ACM Transactions on Office Information Systems*, **5**(2), 168–186, (April 1987).
10. P.J. Brown, 'Turning ideas into products: The Guide system', in *Proceedings of Hypertext'87*, 33–40, ACM Press, New York, (1987).
11. F.G. Halasz, T.P. Moran, and R.H. Trigg, 'NoteCards in a nutshell', in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 45–52, (1987).
12. J. Conklin and M.L. Bergman, 'gIBIS: A Hypertext Tool for Exploratory Policy Discussion', *ACM Transactions on Office Information Systems*, **6**(4), 303–331, (October 1988).
13. A. Scheller, 'HyperODA — extensions for non-linear structures (2nd PDAM), ISO/IEC JTC1/SC18/WG3/N2515', Technical report, International Standard Organization, (1993).
14. N. Streitz, J. Hannemann, and M. Thüring, 'From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces', in *Proc. of Hypertext'89*, 343–364, ACM Press, NY, (1989).
15. N. Streitz et al., 'SEPIA: A Cooperative Hypermedia Authoring Environment', in *Proceedings of ECHT'92*, ed., D. Lucarella et al., 11–22, ACM Press, NY, (1992).
16. M. Thüring, J. Haake, and J. Hannemann, 'What's Eliza doing in the Chinese room? Incoherent hyperdocuments — and how to avoid them', in *Proceedings of Hypertext'91*, 161–177, ACM Press, New York, (1991).
17. C.L. Foss, 'Effective Browsing in Hypertext Systems', in *Proceedings of the Conference on User-Oriented Content-Based Text and Image Handling (RIAO'88)*, 82–98, (1988).
18. Fuji Xerox, *Akane: User's Manual for Document Editor*, 1992.
19. K. Shin, 'Equivalence theorem for multi-level layout', in *PODP92*, (1992).
20. C.C. Marshall, F.G. Halasz, R.A. Rogers, and W.C. Janssen, 'Aquanet: A Hypertext Tool to Hold Your Knowledge in Place', in *Proceedings of the Third ACM Conference on Hypertext (Hypertext'91)*, 261–275, ACM Press, New York, (1991).