Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Peter King   Ethan V. Munson (Eds.)

Digital Documents: Systems and Principles

8th International Conference on Digital Documents and Electronic Publishing, DDEP 2000
5th International Workshop on the Principles of Digital Document Processing, PODDP 2000
Munich, Germany, September 13-15, 2000
Revised Papers

Springer

Volume Editors

Peter King
University of Manitoba, Department of Computer Science
Winnipeg, Manitoba R3T 2N2, Canada
E-mail: prking@cs.umanitoba.ca

Ethan V. Munson
University of Wisconsin - Milwaukee, Department of EECS
Milwaukee, WI 53201, USA
E-mail: munson@cs.uwm.edu

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.4.1, H.5.1, I.3.3, I.3.7, H.5.4, I.7, H.3

ISSN 0302-9743
ISBN 3-540-21070-9 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10988053 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of two recent conferences in the field of electronic publishing and digital documents:

– DDEP 2000, the 8th International Conference on Digital Documents and Electronic Publishing, the successor conference to the EP conference series; and
– PODDP 2000, the 5th International Workshop on the Principles of Digital Document Processing.

Both conferences were held at the Technische Universität München, Munich, Germany in September 2000.

DDEP 2000 was the eighth in a biennial series of international conferences organized to promote the exchange of novel ideas concerning the computer production, manipulation and dissemination of documents. This conference series has attempted to reflect the evolving nature and usage of documents by treating digital documents and electronic publishing as a broad topic covering many aspects. These aspects have included document models, document representation and document dissemination, dynamic and hyper-documents, document analysis and management, and wide-ranging applications. The papers presented at DDEP 2000 and in this volume reflect this broad view, and cover such diverse topics as hypermedia structure and design, multimedia authoring techniques and systems, document structure inference, typography, document management and adaptation, document collections and Petri nets. All papers were refereed by an international program committee.

PODDP 2000 was designed to provide a forum for the discussion of the modeling of systems that process digital documents using theories and techniques from such fields as computer science, mathematics and psychology. The papers presented at PODDP 2000 appearing in this volume report on such diverse topics as abstract document structures and document data structures, techniques for document transformation, the applicability of UML (Unified Modeling Language) diagrams to document specifications, and automatic link generation. Again, all papers were refereed by an international program committee.

This volume also includes two papers that were previously accepted for the journal EPODD, Electronic Publishing, Origin Dissemination and Design. One of these papers contains a comparative evaluation of two common approaches to the electronic presentation of news, while the other paper describes an agent-based toolkit for finding and remembering information in a distributed environment of rapidly changing information sources. These two papers were reviewed by the editorial board of the journal.

The editors would like to thank the members of the PODDP and DDEP program committees for their considerable assistance in providing very thorough reviews of the submissions. We gratefully acknowledge the support of the conference sponsors: Software-Offensive Bayern, Comet Computer GmbH, the Institut

für Informatik Technische Universität München, and Springer-Verlag. Prof. Dr. Anne Brüggemann-Klein worked tirelessly as the chair of DDEP 2000 and also managed local arrangements for both meetings. We also express our sincere gratitude to Frau Evelyn Gemkow of the Technische Universität München, whose service as the conference secretary was much appreciated, and to Frau Diana Gross, Dr. Stefan Hermann and other members of the Rechnerbetriebsgruppe at TUM for the design of the Web pages and other on-site technical assistance, including Internet access.

December 2000

Peter King,
Ethan V. Munson

DDEP 2000 Organization

Steering Committee

Program Committee

PODDP 2000 Organization

Steering Committee

Program Committee

Table of Contents

A Link-Oriented Comparison of Hyperdocuments and Programs

Heather Brown[1], Peter Brown[1], Les Carr[2], Wendy Hall[2], Wendy Milne[1], and Luc Moreau[2]

[1]Department of Computer Science, University of Exeter, Exeter EX4 4PT, UK
{H.Brown,P.J.Brown,W.Milne}@exeter.ac.uk

[2]Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK
{L.A.Carr,W.Hall,L.Moreau}@ecs.soton.ac.uk

Abstract. There are parallels between the construction of programs and the construction of hypertexts, and in particular between the abstractions available to the application programmer and those available to the hypertext author. In this paper we look at the distinctive element of the hypertext medium, the link, and discuss its possible programming language analogs. We go on to examine programming language abstractions that could be usefully employed by hypertext authors to control the complexity of the systems which they are engaged in building.

1 Introduction, Background, and Assumptions

Following Dijkstra's famous article 'Goto considered harmful' [8], written in 1968, the goto statement has been deprecated in most programming languages. Remaining at the lowest abstraction levels, such as assembly languages, or used for efficiency reasons, it is masked by higher-level abstractions (e.g. selections, procedure calls, method invocations and continuations). On the other hand, the hypertext link, which has been characterized as a goto [7], has remained in use in hypertext. Indeed many people see it as the essence of hypertext; most of the definitions of hypertext originally given by Nielsen [16] centre around linking.

Although programming involves many variations on the goto which are safer for programming-in-the-small (if/then, case, iterations) and -in-the-large (procedures, modules, class libraries), the simple link remains the principal tool for the hypertext engineer. The complexity inherent in the unconstrained use of this primitive construct is one of the main issues in hypertext design.

This paper tries to analyse the apparent clash of practice. It also covers a wider issue: many authors have extended the goto/link analogy by likening the authorship of a hyperdocument to the task of writing programs, or, at a higher level, have mapped out a discipline of hypermedia engineering to match software engineering [13]. These comparisons can be valuable because hyperdocument authoring is a young discipline compared with the discipline of producing programs; if, by drawing parallels with programming, we can gain new insights into hyperdocument authoring, there are big potential gains. We must ensure, however, that the parallels are valid ones, and this paper tries to help. Our main focus, reflected in the title, is at the comparatively low level of links, rather than the higher levels of structuring and engineering. Maintaining this focus, we look at ways of modelling links that are more expressive than a simple goto and look at the difference between the static and dynamic aspects of hypertext construction. Finally we list a number of control abstractions that have been used by software engineers and consider how they may help the hypertext author.

1.1 Assumptions

In order to make this paper simpler, we shall fix some of the objects we are talking about. We shall assume that the hypertext is represented in HTML and viewed on a web browser. An HTML document may host scripting components, applets and various kinds of dynamic event handlers, but if we discuss hyperdocuments that involve bits of program this will inevitably muddy our discussion. Thus we shall confine our discussion to static hypertext: no pieces of Java, no CGI scripts, no cookies, etc. Our HTML hyperdocument will, of course, consist of a number of pages, and these will in general link to pages outside the current hyperdocument.

Notwithstanding our use of HTML as a basis for example, we will refer to hypertext systems other than the World Wide Web, since many of these are more developed in the abstractions they provide. In particular HTML essentially only offers one type of link, though there is a potentially extremely rich set of link types that hypertext systems may offer [4,14].

1.2 The Author and the End-User

If we start on the programming side, the two important parties are the author(s) and the end-user(s). The author creates the program, which may be a module in a much larger program, and the end-user executes the program. The author's world is a long way away from the end-user's. Indeed the end-user is normally unaware of the nature of the source code, and whether it contains any gotos. The goto concept, and indeed all concepts of program structuring, just apply to the author's source code world.

In the hypertext world, the author prepares a document. The end-user reads the document the author has built. However the end-user's world is much closer to the author's than is the case for a program. In particular the links provided by the author are directly visible to the end-user. Thus, reflecting these two levels, we can draw two comparisons:

(a) between gotos in programs and the complete set of all links specified by hypertext authors, or

(b) between gotos in programs and the actual set of links used during a browser session by hypertext end-users.

We believe that (a) is the closer comparison, but (b) still deserves attention since the goto is principally happening to the end user. The text does not ‘go’ anywhere, instead there is an intuitive understanding that the user has ‘travelled’, hence the common reference to ‘navigating’ or ‘surfing’ the Web. The role of the author is to specify the complete set of gotos, i.e. to determine the possible navigational choices from which the user chooses the actual set.

2 Alternatives to the Goto Model

Our first point is that if one wishes to liken a hypertext link to a programming language concept, there are several alternatives to the goto model. We will discuss three of them here: we will call them the link-is-data-reference model, the link-is-procedure-call model and the link-is-a-continuation model.

The first model, the link-is-data-reference model, is simple. In this, a hyperdocument is likened to the data part of a program, not the executable part. Each hyperdocument page is likened to a particular data structure (or to an object in OO technology), and links are just references to other data structures. As an example, if the hypertext page just consists of some text T1, followed by a link L, followed by further text T2, then, using Java notion together with a very simple document object model, this is likened to the programming language data structure:

```
final String T1 = "If you are interested, please";
final Link L = new Link("click here",
        "http://site.org/data.html");
final String T2 = "for more information";
final Page P = new Page(T1, L, T2);
```

This is a simplistic view of a Web page, the Web Consortium’s DOM standard [19] is a more complete mechanism for treating a hypertext page as just such a simple data structure. The effect of this from our point of view is to reduce a link to an undistinguished component of the data structure, and requires the navigation behaviour (and the rendering activity) to be specified by external semantics in the form of stylesheet data or scripted functions.

Our second model, the link-is-procedure-call model, is closer to the goto model in that it relates to the executable part of a program. Although a Web server is stateless and is therefore unaffected by the user’s choice of link navigation (without the use of cookies or explicitly programmed work-arounds), each browser maintains both a linear history and a stack of previously-visted pages together with a Back button. Given a Back facility, a link is arguably a specification of a procedure call, not a goto at all. A model, reflecting the analogy that each hyperdocument page is a procedure that potentially calls other procedures, is that a page can be likened to a procedure with the following body:

```
public static void invoke(String url){
   boolean mustExit=false;
   while(!mustExit){
         renderThisWebPage(url);
         selectedLink=getLinkChoice();
         if(selectedLink==back)mustExit=true;
         else invoke(selectedLink.getURL());
         }
   }
```

(The use of the while loop ensures that if a procedure representing another page is called, and this subsequently returns — as it will if the user selects Back — then the original page's procedure is re-executed.)

Obviously this model needs elaboration to cover special cases, one of which is links within a page. (Splitting the procedure for the page into subprocedures may best cover within-page anchors.) Nevertheless we trust that the model captures the essence of the procedural analogy, and shows that a link can be modelled as a procedure call. The model also covers some hypertext systems that have richer types of link properties than HTML (e.g. different data types of links, authorship properties attached to links), since these extras can be added as arguments to the procedure call.

Additionally in some hypertext systems, such as Hyper-G [10] and Microcosm [3], links are two-way. Two-way links provide a rich static mechanism that complements the dynamic facility provided by Back. However, given our keep-it-basic approach, we will confine ourselves to simple, one-way, links.

However, the real problem with links-as-procedure-calls is that browsers have "Forward" buttons too, so that 'returning' from a Web page is not completely akin to exiting a procedure as the procedure may be re-entered with its state restored. Further complications arise from the browser ability to clone an existing window, thus potentially duplicating the procedure's activation record in a different context. To resolve this we present a third model, the link-is-a-continuation model.

The third model is best introduced via a sample application. The role of links in the presence of a browser forward button can be seen in the example of an educational CD-ROM [17] which we discuss here to show how an extension to the concept of a procedure call can successfully model arbitrary use of forward and back buttons as well as the history list. The CD-ROM contains software, embedded in an HTTP server, which manages complex guided tours of pages, menu choices and arbitrary computations that depend on users' runtime preferences. Consequently, with each page, there is a "computation state" which leads to the selection of the page, and which will also be used to select the next page in the tour.

Let us consider that the user jumps back to a previous page in the history list. The associated server "computation state" should be restored to what it was when the user first visited the page, so that the next page in the guided tour may be selected appropriately. Queinnec observes that the notion of continuation exactly embodies the computation state that has to be restored. Continuations are first-class objects in the language Scheme and represent the "rest of the computation". At any moment, the current continuation, *i.e.* the rest of the operations remaining to be done, may be captured and stored away in the heap. Vice-versa, a continuation, which stored in the heap, may be re-instated. Queinnec shows that for each document served by the CD-ROM server, there exists an associated continuation representing the current server computation state. When the user navigates the history list, the associated continuations are re-instated in the server. In the terms of the hypertext, the continuation provides both a specification of the page to be viewed and also a 'history so-far' of the navigation up to that page.

Can these alternatives to the goto model reverse our thinking about links? In contrast to gotos, the use of procedure calls, continuations and data references are regarded as good programming practice; does this reflect directly on linking? Perhaps instead it shows the weakness of superficial analogies. In particular, to look at the analogies more carefully, we need to examine dynamic nature of programs and hypertexts.

3 Dynamic Aspects of Programs and Hypertexts

Programs are static entities, commonly expressed as a text, which specify a dynamic execution process. Whatever the programming paradigm, execution may be modelled by a trajectory in an execution space. Structuring is indeed the focus of Dijkstra's original paper. To quote:

> " ... our intellectual powers are rather geared to master static relations and our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible."

Thus the argument is that a vital aim in the design of programming languages is to make the program easier to understand and to reason about, particularly in its dynamic behaviour.

On the hypertext side, hyperdocuments are also static, but the user's navigation by link traversal entails a dynamic change of state. In practical terms, this change of state occurs both in the browser (for instance change of history list), or in the server (for instance a change in the "computation state"). Navigation can be paralleled to a trajectory in a 'hyperspace'. The author of a static hyperdocument has also the intellectual burden of visualizing the navigation in the hyper space. There are at least four burdens on the hypertext author caused by the connectivity of the material.

1. structuring the material in the first place. Correctly structuring a program may be facilitated by a design model and by the underlying abstractions of the programming language. Successfully structuring a hyperdocument is also helped by applying a suitable design model [9, 10, 16], however the unenhanced link remains the underlying abstraction available to the author. This is indeed a challenge (see the evidence collected in Nielsen's book), but is not our focus here.
2. making the document coherent over all possible paths,

3. ensuring there are no dangling links, or, the other side of the same coin, no material that is unreachable by a link,
4. providing navigation alternatives for the end-user.

We discuss (2), (3) and (4) in turn in the three subsections below.

3.1 Coherence and Consistency

Item (2) above provides a large intellectual burden on the author. To write the content of a page, the author potentially needs to know all the possible paths by which an end-user may reach that page. This is trivial in a catalogue or encyclopaedia, since each page is authored to be self-contained to be read independently of all the other pages. The burden is significant in a tutorial document: for example if concept C2 builds on concept C1 then C1 is a prerequisite for C2, and all the possible paths to a page about C2 should pass though a page about C1 first. (Of course, the author cannot stop the end-user jumping directly to a page about C2 by means of a bookmark or a "find" facility.)

A good design strategy is to produce a "rhetoric of arrival and departure" [11] for each significant page, which must be honoured by all the pages that link to the current page. Thus in general, to make a hyperdocument coherent, the author has to visualize the connectivity of the whole hyperdocument. An obvious consequence of this is that "spaghetti linking" (lots of links with no structure to them) increases the author's burden, especially without tools to assist the process [18].

Research into programming languages has led to the definition of constructs that control the scope of names and data access (blocks, procedures or modules). On the other hand, hyperdocuments have one scope: the page. This presents a significant burden to the author who needs to be aware of all the pages that may link to the current one, either directly or via other pages, and spoils the analogy that a link is a clean procedure call, or, for that matter, a simple data reference.

A similar situation exists in the SGML world, where the primitive linking facilities provided (IDs and IDREFs) also give a single level of scope, but one that is strictly enforced as no reference can be made to IDs in other documents. HyTime added this capability, but further made it possible to reference any item of data (with or without IDs) in any kind of document (encoded in SGML or not), thus overriding and destroying the concept of scope as an authorial tool.

Turning from the author, the cognitive load on the end-user is remembering what decisions were made at previous pages visited, i.e. which link was chosen, and remembering what other information and links were on the page. (There is also the longer-term problem of remembering, perhaps several days later, where information was seen, but this is not our concern here.)

3.2 Dangling Links

For regular web users, probably the most obvious failure in hyperdocument authorship is the "dangling link", the link that tries to reference a non-existent URL. Obviously therefore there is a burden on authors to get all their links right, and this burden proves too great for many authors. The problem can occur either (a) because the author got the link wrong in the first place, or (b) because the link destination has subsequently been changed. Corresponding failures when running programs are comparatively rare, though problems of type (b) can arise in those programming languages that support dynamic casts.

The early history of programming language development made the transition from explicitly enumerated machine addresses to symbolic names, with the rules controlling the resolution of names enshrined in the definition of the programming language's semantics. In the hyperdocument world, of course, the link is defined in terms of a URL that is a machine address (quite literally). Attempts to introduce symbolic names (URNs) have so far not succeeded, and in fact the reverse is true: in situations where symbolic names are required (for example, to define XML namespaces) a hypothetical machine address is invented. As symbolic names are normally used in programming, there is inevitably a linkage process or link resolution activity in which the identity of the data being named is established.

In the programming world thorough checking is the norm, and programs are continually checked both throughout their development and throughout their usage. There have been great advances in mechanisms to allow authors to detect errors before programs are released to end-users. Some of these mechanisms, like test harnesses, simply involve simulating user behaviour; the more interesting mechanisms concern programming language features that make certain types of error impossible. One of these is strong type checking: this allows a compiler to detect certain types of error, and, if they occur, prevent a program from being run. A second is the concept of strictly defined module interfaces, which allow linkage editors to detect errors when modules are fitted together. A third is information hiding, whereby the author can control who sees what.

The culture in hypertext authoring, on the other hand, is still towards the do-what-you-like end of the spectrum. Thus authors do not generally use some equivalent of a linkage editor, which could detect dangling links within a hyperdocument, or — the converse — pages that could never be reached. Nor do authors provide the equivalent of imports and exports declarations, which specify which outside links are assumed, and which outside services the hyperdocument provides. It would be better if they did. Finally, it would help if hypertext authoring systems provided for information hiding, if only at the page level ("This page is public and anyone can link to it; this other page is an internal one that might well be changed, and thus other authors should not link to it").

On a wider scene, these extra mechanisms might be extended to cover user interfaces in general, rather than just the restricted form of interface provided by hypertext linking.

3.3 Navigation Alternatives

A facility that is radically different from programming languages is the concept, present in Microcosm, Hyper-G and HyTime, that the link structure should be separate from the document that it applies to. Indeed several separate and independent link structures could be applied to the same document, each author thereby providing an interpretation of the structure of the underlying document. Moreover the separate links might be what DeRose calls "Intensional" links [4], such as a generic link from every occurrence of a certain word — wherever it may occur — to a dictionary entry that explains that word.

4 How Connectivity Has Been Tamed in Programming Languages

Our conclusions so far are that there are, not surprisingly, big burdens on the hyperdocument author caused by connectivity. Hence we can gain by looking at the programming mechanisms that help tame connectivity. We will now do this, and we will then relate these mechanisms to hyperdocument authors and end-users.

4.1 Aggregation of Choices

Aggregation of choices in programs is provided by the if-then-else statement, and a generalisation of this, the case or switch statement. This represents the abstraction of "choose one from many". It allows nesting and thus a tree-structured choice. Aggregation of links is common in hypertext systems outside the Web either in the form of the "choose-one-from-many" pattern found in programming languages, or in the form of the trail, a sequence of links followed in turn. In the former case, the aggregation can simply be a matter of recommended style [12], perhaps using HTML's list as an existing data aggregation mechanism; alternatively in HyTime [15] and the seminal Intermedia hypermedia system [18], aggregations of links ("fat", one-to-many links) are a built-in feature. In the latter case, the trail is provided by the author to guide the end-user through a predetermined sequence of pages, designed to give the end-user an understanding of a certain subset of the information represented by the overall hyperdocument. Usually there can be any number of separate trails through a hyperdocument, and these are totally independent of each other: for example they may cross and overlap one another in arbitrary ways. The path concept can be carried a stage further by allowing choices within a path, and more generally by providing a script that takes paths though a document and performs various actions as it goes [21]. It is not clear that this abstraction is found in programming languages.

4.2 Assert Statements

Checking that a property holds at run time is an important feature both in programming languages and hypertext. Although assert statements in programming languages are normally used for dynamic states, they can also be used in conjunction with the statically-determined states in hyperdocuments. The author could use a similar facility to verify that a condition holds for the user's navigation. To return to our example of concept C2 depending on concept C1, the page for concept C2 might say: <assert> C1-covered</assert> where C1-covered is a Boolean variable set by all pages that cover C1. Thus one such page may say: <set> C1-covered </set> (Obviously <set> and <assert> are concerned with checking, and do not cause anything to appear on the screen: they are therefore likely to be coded along with other metadata in the HEAD section on the hypertext page.) Given these declarations, a checker, run every time the hyperdocument is modified, can ensure that no path through the hyperdocument can reach the <assert> statement for C0-covered without passing through a corresponding <set> statement. A mechanism that could be used for this facility exists in the form of cookies (named values that can stored in the browser), but this is at too low a level to be immediately useful to the hypertext author.

4.3 Pre- and Post-conditions

Java's finally clause guarantees that a post-action will be executed whenever control leaves a block for any reason (natural termination, a return statement, breaking out of a loop or exception throwing). More generally, Scheme's dynamic-wind provides both pre- and post-actions for a block. Such a mechanism could be used to ensure that a particular condition holds (see previous subsection) or to enforce the arrival and departure paradigm (see section 3.1) when linking into and out of the body of a hyperdocument [6].

4.4 Procedures

Procedures have been described above as models for the behaviour of a link. They also stand as useful metaphors for hypertext construction: the seminal Guide system [2] encouraged users to think of navigation in terms of embedded, nested pages which were opened (unfolded with contents visible) or closed (folded with contents hidden), corresponding very well to the activation of a procedure.

4.5 Exception Handling

The ability to handle exceptional error situations has become an important feature of programming languages. Several HTML constructs (for example the `<OBJECT>` and `<FRAMESET>` elements), can declare alternative hypertext page fragments to present to the user if the browser cannot correctly process the required elements. Similarly, Web servers and proxies can be configured to present alternative information to the user if the requested page cannot be delivered.

Most hypertext servers, when unable to resolve a URL, just present a page that contains an error message, and leave it to the user to decide what to do, i.e. the server provides its own default exception mechanism. It would be of more benefit to expose the exception handling mechanism to the hypertext author to allow better control, especially over links to remote pages over which the author has no control.

4.6 Interfaces

Interfaces (made popular through Java) have the potential to provide a useful abstraction that can be applied to a set of hyperdocument pages in the same way that a Java interface provides a useful abstraction that captures the key features of a set of data objects. By way of example, assume that author X has produced a hyperdocument that discusses geometric shapes. X recognises that different end-users, who will have different displays, will want to view these shapes in different ways. Hence, following programming practice, X decides that all the hypertext pages concerned with displaying shapes should be in a separate module. X provides one possible instantiation of this module, but wants to allow other authors to provide other ones. In order to aid this X would like to provide an interface specification, which guarantees that any module that discusses geometric shapes and gives similar kinds of information would be an acceptable replacement. Any set of Web pages (perhaps discovered by a search engine) which contains an explanation about squares, circles and triangles and taught the user how to find the perimeter and area of each shape irrespective of the tuition method or rendering technology may be an equivalent module as far as the purpose of this hyperdocument is concerned. Interfaces model an expected set of facilities and may be about the data or the links that are to be used across the boundary of the hyperdocument.

5 Concluding Comments

Programming can at many points be usefully compared to hyperdocument authorship. Both program execution and hyperdocument navigation are concerned with dynamic state. However the connectivity structure is visible to the end-user of a hyperdocument, but not to the end-user of a program. It is often the hypertext author's responsibility to add extra connectivity to aid user navigation, whereas a programmer's aim might be to minimize connectivity in order to reduce complexity or to decrease the code's 'footprint'.

The hyperdocument author needs to be aware of the incoming links to each page, whereas the programmer need not be aware of the context in which a procedure or module is to be used. Further, hyperdocuments normally have links to external hyperdocuments over which the author has no control and for which no checking or validation is performed.

Some of the abstractions in hypertext have no obvious parallels in programming: for example we have discussed trails and separating link structure from hyperdocuments. Hence likening hypertext links to programming constructs, if done superficially, is unlikely to yield valid insights. To return to the is-a-link-a-goto question, the answer is that it is an analogy that is half right, half wrong. Analogies of gotos with procedure calls or data references are just as right — and just as wrong.

It is better to look at the overall concept of connectivity, which produces similar issues in programming and in hyperdocuments. The development of programming has been dominated by introducing more disciplined ways of working. Some disciplines that can help hypermedia are:

(d) assertions, pre- and post-conditions

(b) linkage editors to check connections

(c) exception handling and

(d) interfaces to hyperdocuments.

Overall, an important requirement of hyperdocument authors is for new mechanisms in the authoring language or environment that are designed to prevent linking errors. The Xlink proposal [5] provides a policy-free container architecture for the Web which could be used to express some of the abstractions listed in this paper; further work is planned to build authoring support systems on this basis.

Acknowledgements. Peter Brown would like to thank the Leverhulme Trust for support.

References

1. Brown, P.J. “Do we need maps to navigate round hypertext systems?”, EP-odd, 2, 2, pp. 91–100, 1989.
2. Brown, P.J. “Dynamic Documentation”, Software Practise and Experience, 16, 3, (March 1986), 291–299.
3. Davis, H.C., Hall, W., Heath, I., Hill, G.J. and Wilkins, R.J. `Towards an integrated environment with open hypermedia systems', Proceedings of the ACM Conference on Hypertext: ECHT92, ACM Press, pp. 181–190, 1992.
4. DeRose, S.J. “Expanding the notion of links”, Hypertext'89 Proceedings, ACM Press, pp. 249–257, 1989.
5. DeRose, S.J. “XML Linking Language (XLink)”, W3C Working Draft 21-February-2000, http://www.w3.org/TR/xlink/ . 2000.
6. De Roure D. “The Role of Distributed Lisp” in Open Hypermedia Information Systems, In Proceedings of Parallel Symbolic Languages and Systems, 1996.
7. De Young, L. “Linking considered harmful”, Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext, INRIA, France, Cambridge University Press, pp. 238–249, 1990.
8. Dijkstra, E.W. `Goto considered harmful', Comm. ACM, 11(3), pp. 147–8, 1968.
9. Garzotto, F., Paolini, P. and Schwabe, D. “HDM – A Model-based approach to Hypermedia Application design”, ACM Transactions on Information Systems, 11(1), 1–26, 1993.

10. Kappe, F., Maurer, H., and Sherbakov, N. `Hyper-G: a universal hypermedia system', Journal of Educational Multimedia and Hypermedia, 2, 1, pp. 39–66, 1993.
11. Landow, G.P., `The rhetoric of hypertext: some rules for authors', Journal of Computing in Higher Education, 1, 1, pp. 39–64, 1989.
12. Lemay, L., Teach yourself web publishing in a week, Sams Publishing, Indianapolis, Third Edition, 1996.
13. Lowe, D. and Hall, W., Hypermedia & the web: an engineering approach, John Wiley, Chichester, 1999.
14. Moreau, L. and Hall, W. “On the expressiveness of links in hypertext systems”, Computer Journal, 41, 7, pp. 459–473, 1998.
15. Newcomb, S.R., Kipp, N.A. and Newcomb, V.T., “The HyTime hypermedia/time-based document structuring language”, Comm. ACM, 34(11), pp. 67–83, 1991.
16. Nielsen, J. Multimedia and hypermedia: the internet and beyond, Academic Press, San Diego, Ca., 1995.
17. Christian Queinnec, “The Influence of Browsers on Evaluators.” University Paris 6 — Pierre et Marie Curie. Submitted for publication, 2000.
18. Thimbleby, H. Int. J. Human-Computer Studies (1997) 47, 139–168, http://ijhcs.open.ac.uk/thimbleby/thimbleby-01.html
19. Wood, L. Level 1 Document Object Model Specification, W3C Recommendation, http://www.w3.org/TR/WD-DOM/
20. Yankelovich, N., Meyrowitz, N. and van Dam, A., “Reading and writing the electronic book”, IEEE Computer, 18, 10, pp. 15–30, 1985.
21. Zellweger, P.T. `Active paths through multi-media documents', in van Vliet (Ed.), Document manipulation and typography, Cambridge University Press, pp. 1–18, 1988.

Context-Aware Digital Documents Described in a High-Level Petri Net-Based Hypermedia System

Jin-Cheon Na and Richard Furuta

Department of Computer Science, Texas A&M University
College Station, TX 77843-3112, USA
{jincheon, furuta}@cs.tamu.edu

Abstract. As mobile computing becomes widespread, so will the need for digital document delivery by hypertextual means. A further trend will be the provision of the ability for devices to determine *where* they are, e.g., through the inclusion of GPS sensors, as will the need for devices to operate on heterogeneous networks. Consequently, hypertext systems supporting these devices will benefit if they also can become *context-aware*. In this research, we introduce caT (for Context-Aware Trellis), a context-aware hypertext model and associated tools, which supports flexible user (or agent) adaptation to changes in environmental information, such as time, location, bandwidth/cost, etc. Firstly a context-aware hypertext model is proposed by incorporating both high-level Petri net features and user-modeling into the previously-described Trellis hypertext model. Major features of the high-level Petri net that are being explored are structured tokens and flexible net description. Fuzzy knowledge (context) handling is supported by the integration of a fuzzy logic tool with the Petri net, and a flexible information presentation tool has been designed to support Web-based browsing of documents specified in the caT model. Additionally, authoring and analysis issues are explored to support structured authoring and verification of application models, respectively. A simple example, a university digital library, is introduced in the paper to explain the concept of the caT model.

1 Introduction

As mobile computing becomes widespread, so will the need for document collections that can customize themselves for reading in different environments. If mobile documents properly reflect their readers' context, they can provide more relevant information to meet their readers' dynamically changing contextual requirements. Mobile systems should provide context-aware services without requiring readers to provide explicit context data. For this purpose, over the last decade, some researchers have built context-aware applications [6, 17] that take advantage of environmental information to provide better interaction with readers. Context-aware applications typically focus on a mobile user who is carrying a portable system, such as a Personal Data Assistant (PDA), that has been augmented with environmental sensors, such as GPS receivers, active badges, electronic compasses, etc. These environmental sensors could detect location, orientation, time of day and time of year, temperature, companions or objects nearby, user identification, etc. Additionally, high-level sensors (software sensors), which correlate information from lower level sensors in order to deduce some higher level state, could detect more complex situation data.

Some context-aware document applications have been developed. Peter Brown [3] provides a framework for discrete context-aware applications and services based on a simple *Post-It note* metaphor, where discrete pieces of information are attached to individual contexts, to be triggered when the user enters those contexts. Mobile tour guides [1] have been developed to familiarize a visitor with a new area. Office awareness systems [21, 22] sense users' locations, help people find each other, and keep up awareness. Context-based retrieval applications [16] collect and save context information and support subsequent information retrieval based on context information. Conference Assistant [7], which supports conference attendees and presenters, assists users in taking notes on presentations and aids in the retrieval of conference information after the conference concludes.

Hypertext systems, including the WWW (World-Wide Web), have been widely accepted as document navigation and search tools. Hypertext systems supporting mobile systems will benefit if they also can become *context-aware*. For example, we can think of a university digital library, which wishes to provide a full version of a Web document to its on-campus patrons and a limited one to its off-campus patrons. The library system also might give more detailed documents for reference material located on the current floor than for other floors. Some documents should be made available for general use outside of working hours or when they are no longer secret.

A focal point in adaptive hypermedia systems [2, 4] has been support for user modeling. Adaptive hypermedia systems use knowledge represented in a user model to adapt the information and links being presented to the given user. Adaptive hypermedia systems are mainly used now in educational hypermedia areas where the hyperspace is reasonably large and where individuals with different goals, knowledge and background use a hypermedia application. Providing context-awareness will require user modeling.

In this research, we will introduce extensions to a previously-described hypertext model in order to support context-awareness. Trellis [8], based on colored timed Petri nets [12], is chosen since it provides good facilities for dynamic adaptations and supports formal analysis techniques. This paper is organized as follows. Related work is discussed in section 2. The research objectives and detailed research work are described in section 3. Finally, future work and conclusions are discussed in section 4.

2 Related Work

2.1 Petri Nets

Petri nets [12, 15, 24], as graphical and mathematical tools, provide a uniform environment for modeling, formal analysis, and design of systems. Formally, a basic Petri net is a bipartite directed graph defined as follows:

A Petri net structure is a tuple, <P, T, F>, in which

- $P = \{P_1, P_2, \ldots, P_n\}$ is a finite set of places with $n \geq 0$;
- $T = \{T_1, T_2, \ldots, T_m\}$ is a finite set of transitions with $m \geq 0$ and $P \cap T = \emptyset$;

- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, a mapping representing arcs between places and transitions.
  For a marking of a Petri net structure <P, T, F>
- $M: P \rightarrow I, I = \{0, 1, 2, \ldots\}$, is a function that associates a marking to each place in the net.

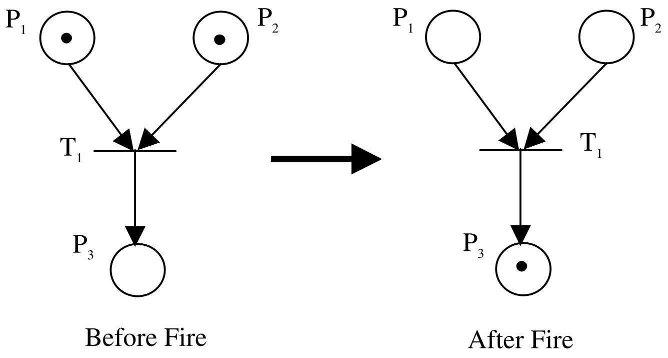


Fig. 1. A Simple Petri Net

Graphically, a Petri net is represented by indicating its places by circles, transitions by bars, arcs by arrows, and tokens by small black dots. A place containing one or more tokens is said to be marked. When each place incident on a transition is marked that transition is enabled. An enabled transition may fire by removing one token from each of its input places and putting one token into each of its output places. For example, consider the marked Petri net shown in Figure 1. The initial marking is $M_0 = [110]$ (i.e., $P_1 = 1$, $P_2 = 1$, $P_3 = 0$) and T_1 is an enabled transition under M_0. If T_1 were fired, the resulting next state would be M_1=[001], as shown in the right side of the Figure.

The basic Petri net is not always convenient for representing and analyzing complex systems because tokens in the basic Petri nets have no identity. In order to overcome this problem, Petri nets that allow tokens to have distinct identity, called high-level Petri nets, were proposed. These nets include predicate-transition nets [9], colored Petri nets [12], object-oriented Petri nets [10], and others. In high-level Petri nets, a token can be a composite object carrying data, which may be of arbitrary complexity including integers, reals, text strings, records, lists and tuples. Nonetheless, it should be noted that ordinary and high-level Petri nets have the same descriptive power, even though high-level Petri nets supply much better structuring facilities than basic nets [24]. A more thorough exposition of net theory can be found in the texts by Peterson [15] and Jensen [12].

2.2 Trellis

The Trellis project [8, 18] has investigated the structure and semantics of human computer interaction, in the context of hypertext (hypermedia) systems, program browsers, visual programming notations, and process models. The Trellis model is formally defined using colored timed Petri nets as the structure of a hyperprogram, and this gives the model an elegant structure that can be both programmed and analyzed [24].

In the form of colored timed Petri nets (CPN) used by the Trellis model, tokens have color types and a token of one color is discernible from a token of another color. However, within a color class individual tokens cannot be distinguished from one another. Each transition in a Trellis net has two time values, representing respectively a delay and a timeout. Time values in Trellis are thought of as defining ranges for the availability of an event. Formally, a colored timed Petri net in the Trellis model is defined as follows [18]:

A CPN is a tuple, $<\Sigma, P, T, F, \tau>$, in which

- Σ is a finite set of token colors, such as {black, maroon, …};
- P, T, and F are as defined earlier ;
- $\tau : T \rightarrow \{0,1,2, \dots\} \times \{0, 1, 2, \dots, \infty\}$ is a function mapping each transition to a pair of values termed release time and maximum latency respectively. For any transition $t \in T$, we write $\tau(t) = (\tau^r, \tau^m)$ and we require that $\tau^r \leq \tau^m$.

A Trellis net is completed by annotating for the components of the CPN. We can conceive of the CPN as the task description and the different annotations as the information required by the task. One important category of annotation is content. Fragments of information (text, graphics, video, audio, executable code, and other hyperprograms) are associated with the places in a CPN. Another category of annotation is events, which are mapped to the transitions of the CPN. A third category of annotation is the attribute/value (A/V) pair; a list of A/V pairs is kept with each place, each transition, each arc, and for the CPN as a whole. Annotations can be used by the Trellis implementation to build client tools, such as hypertext documents browser and authoring tools. Formally, a hypertext in the Trellis model is defined as follows [18]:

A hypertext in the Trellis structure is a tuple, $<CPN, M_0, D, W, B, P_l, P_d>$, in which

- CPN is a colored timed Petri net $<\Sigma, P, T, F, \tau>$;
- $M_0 : P \rightarrow$ token instances of Σ is an initial marking (or initial state) for CPN;
- D is a set of document contents;
- W is a set of windows;
- B is a set of buttons (or links);
- P_l is a logical projection for the document, $P_l = <D_l,W_l,B_l>$, mappings from components of a Petri net to the human-consumable portions of a hypertext;
- P_d is a display projection for the document, a collection of mappings that associate the logical buttons and windows of a hypertext with physical screen representations and locations.

3 caT

With the increased availability of mobile personal computers, in modern hypertext systems, adaptation support for users in dynamically-changing environments will be essential to meet the needs of mobile users. The main goal of this research is to continue the development of caT [14], a context-aware hypertext model and associated tools. Beginning with the χTrellis implementation by Stotts, et al., the following main research issues are being explored.

1. A context-aware hypertext model: Even though previously-described hypertext models provide a powerful modeling framework, they have a weakness in supporting context-aware adaptation, mainly due to a lack of incorporation of dynamically-changing information from the external environment. We need a new context-aware hypertext model that supports context-awareness in dynamically-changing environments.
2. Fuzzy knowledge handling: By introducing the fuzzy logic [23] concept into a hypertext model, the model can have the ability to handle uncertain knowledge (i.e., fuzzy context) in real world applications. Fuzzy logic primarily is motivated by observing that human reasoning can use concepts and knowledge that do not have well-defined, sharp boundaries (i.e., vague concepts). For example, the new model may support the following link display condition: "*shows a link when access right of the current user is rather high*". In this case, the model needs to use a fuzzy rulebase for inferring the *access right* value from the current user's information (e.g., access time and location).
3. Flexible information presentation: The Trellis model supports good separation between document specification and presentation. This allows multiple presentations of a particular document specification. To provide a WWW-based presentation, which has been widely accepted recently, we need to specify how active content elements and links of the Trellis model are to be displayed and embedded in WWW pages, respectively. The currently-existing χTrellis prototype uses a separate window (or frame) to display the net's active content and links, but in the WWW context it is necessary to have a mechanism that combines these elements into one larger document. A composition mechanism, based on the composite node concept introduced in the Dexter model [11], is explored for the presentation of multiple active content elements into one flexible (or dynamic) document.

To address these requirements, we are developing caT, which supports flexible user (or agent) adaptation to changing environments by incorporating context-awareness, user modeling, and fuzzy knowledge handling features to the current Trellis system. caT also supports Web-based browsing to enable the high usability of the model. The detailed methods of the suggested research work are described in the following subsections.

3.1 Context-Aware Hypertext Model

The foremost objective of this research work is to develop a context-aware hypertext model. Beginning with the Trellis model, the following features are added:

1. Structured tokens: Each colored token caries its own local variable/value pairs. Thus, it can represent dynamically-changing characteristics, such as access time and access location. Providing individual identity to tokens requires a mechanism to determine what values should be used if tokens are combined or replicated during transition firing. In caT, these transformations are specified through predicates associated with the outgoing arcs from the transition. For hypertext

applications, each colored token may represent each person who is in a dynamically-changing environment. With this token information, caT can provide different behavior to the user under different contexts (e.g., different access times, such as morning and evening, spring and winter, etc.). Users can share the same Petri net, but have context-aware, customized views from the net.

2. User modeling: Each colored token also can have a link to a user-modeling profile that contains the user's information (e.g., preference, background, etc.). caT can use this user modeling information in addition to local token values to customize its behavior to different users. The distinction between a token's local values and the user model profile is that the local values are expected to reflect dynamic and environmentally changing data, while values in the user modeling profile are less dynamic, reflecting user profile data. In addition, the user model profile is globally visible, while the local variables are associated with an individual token. For context-aware applications, user modeling will help supplement the limits of current sensor devices. In the real world, some sensor devices are not available to common users, and it is impossible to get some kinds of environmental data from sensor devices, such as a user's current working organization type. User modeling can address these shortcomings by enabling derivation of missing values.
3. Link adaptation: Conditional statements attached to transitions are evaluated with values from the current user model and token; conditional statements determine the threshold values for transition firing, and thus provide link adaptation. Additionally, assignment statements attached to output arcs are used for changing current token values, and function calls for getting environmental data or invoking a fuzzy inference engine are supported in conditional and assignment statements.

The formal definition of Petri net used in the caT model is as follows:

A caT Petri net structure is a tuple, $<\Sigma, P, T, F, \tau, C, G, E>$, in which

- Σ is a finite set of token types, called color sets;
- P, T, F, and τ are as defined earlier;
- $C : P \rightarrow \Sigma$ is a color function;
- $G : T \rightarrow$ *Boolean Expression* is a guard function;
- $E : (T\times P) \rightarrow$ *Arc Expression* is an arc expression function.

The caT model has additional (or enhanced) properties (or functions) in addition to the common ones in the Trellis model. In Σ, in caT, each token can have a color value and optional local token variables. Therefore, each place can have different type of tokens (i.e., tokens with different local variables), declared by "$C : P \rightarrow \Sigma$". The color function, C(p), maps each place to a color set (type), and each token on the place must have a token that belongs to the color type. The guard function, G(t), is used for mapping a Boolean expression to each transition, which specifies an additional constraint (threshold) which must be fulfilled before the transition is enabled. The arc expression function, E(t×p), is used for mapping an assignment expression to each output arc, which changes current token values when the transition is fired.

3.2 Fuzzy Logic Tool Integration

In context-aware applications, we need to handle uncertain user contexts. Consequently, caT incorporates a fuzzy logic engine for evaluating conditional statements used for link adaptation as well as for updating values in local variables. Matlab's Fuzzy Logic Toolbox [13] is used as caT's fuzzy logic engine and is invoked on transition firing. Some fuzzy Petri nets [5] use fuzzy tokens, but keeping the current Petri net model and invoking an external fuzzy logic engine when necessary seems like a simple and reasonable solution, rather than introducing fuzzy logic into the current model and making a complex fuzzy Petri-net-based model.

A function for invoking an external fuzzy logic engine is supported in caT. The first argument is a rulebase name and the following arguments are either local token values or user profile values. For example, the user's access right is inferred by invoking the external fuzzy logic engine with a target rulebase name, user's access time, and current location. A sample rulebase for this purpose is as follows:

1. *If (time is day) and (distance is close) then (accessRight is high) (1)*
2. *If (time is day) and (distance is middle) then (accessRight is middle) (1)*
3. *If (time is day) and (distance is far) then (accessRight is low) (1)*
4. *If (time is not day) then (accessRight is high) (1)*

Generally, the rulebase infers an access right value from the current user's access time and distance. When the current time is *day*, users who are in close distance get high access rights. But when the current time is not *day*, all users get high access rights, regardless of their distance. Fuzzy terms, such as *day, close, middle,* and *far*, are defined by membership functions; a membership function is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1.

3.3 Flexible Information Presentation Tool

In caT, a template file specifies how active content elements are to be displayed and how links are to be embedded. The template file is simply another Trellis content type associated with a place. It takes control when the place is marked; consequently multiple template files can be active at different times during a browsing session. The node that contains the template file can be viewed as a virtual composite node constructed from several atomic nodes. Only active atomic nodes in the composite node are used for generating the current presentation of the composite node. Thus, the content of the composite node changes dynamically based on the current net state.

As an example, consider the case of a university digital library, which provides a full version of a Web page to its on-campus patrons and a limited one to its off-campus patrons. However, faculty members off-campus retain access to the full version. Because the page includes access to a real-time help desk, only the limited version is available outside of normal operating hours. For exemplifying the above scenario in the caT model, some of the current existing library services at Texas A&M University are used. Figure 2 shows a small net fragment implementing these restrictions. The top portion of the net, invoked automatically by caT's time specifications, invokes the fuzzy inference engine to classify the user as being "on-campus" or "off-campus" based on the time of day, distance from campus, access rights, and incoming IP address. The bottom portion of the net specifies the information display in the Web browser, shown in Figure 3. The detailed behavior of the Petri net in Figure 2 is described in the following subsections.

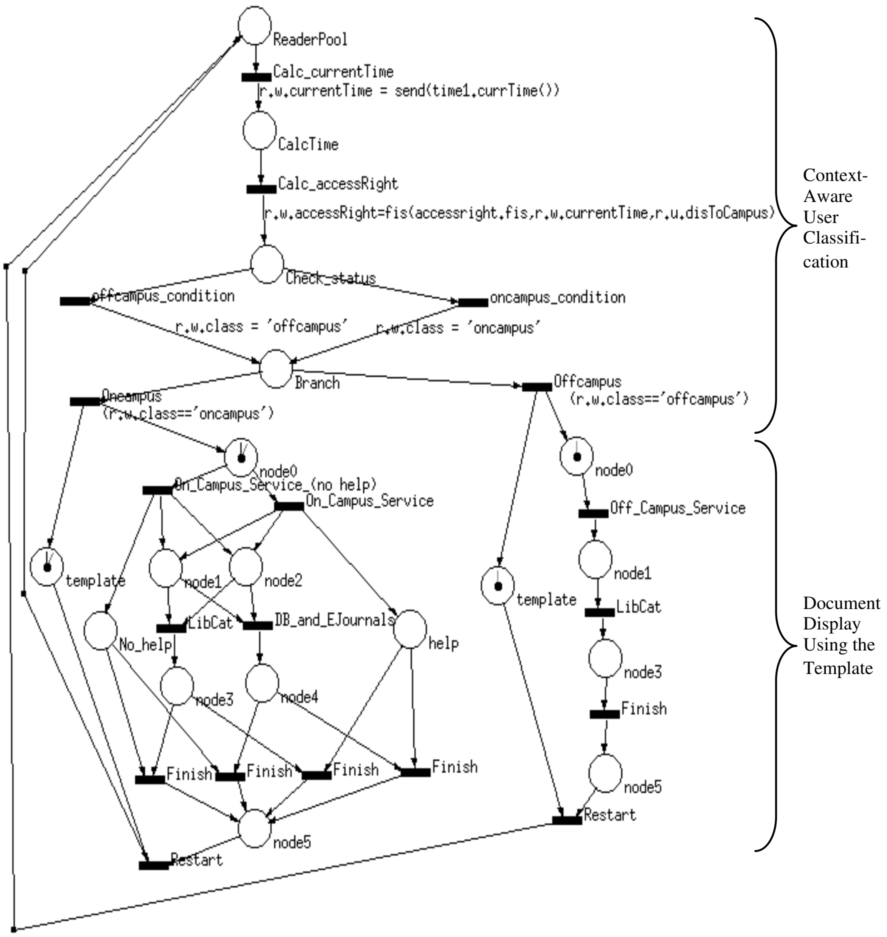


Fig. 2. Petri net of a simple digital library tour

Context-Aware User Classification. Initially, user tokens will be placed in *ReaderPool* place. Each colored token has the following local token variable/value pairs: *user name, accessRight, user class, network,* and *currentTime*. The *user name* and *network* values will be marked initially, and the other values are inferred or calculated by the system. A user profile for each user has the following values: *disToCampus* (distance from a current location to campus) and *userType* (user occupation).

The *Calc_currentTime* transition has a timing value of (0,0). (0,0) means no delay and no timeout, therefore the *Calc_currentTime* transition will be executed immediately by the system when it is enabled. When the *Calc_currentTime* transition is fired, the *currentTime* value is calculated by using an expression that is attached to the output arc of the *Calc_currentTime* transition. *"r.w.currentTime"* is used for accessing a local token variable *currentTime* of the current colored token: "*r*" is a color variable that is instantiated to the current color value, and "*w*" denotes that the following variable *currentTime* is a local token variable. "*send()*" function is used for calling system-support functions. When the *Calc_currentTime* transition is fired, a colored token in *ReaderPool* caries its own local variable/value pairs to the output place, i.e., *CalcTime*, with a new *currentTime* value.

In the next step, the *Calc_accessRight* transition is fired, and the user's *accessRight* value is inferred by invoking an external fuzzy logic engine. *"fis()"* is a function for invoking an external fuzzy logic engine. The *accessright.fis* rulebase is the one introduced in section 3.2. *"r.u.disToCampus"* is used for accessing the user profile variable *disToCampus* of the current token user. *"u"* denotes that the following variable *disToCampus* is a user profile variable. The user profile name is stored in the current token's *user name* variable.

When a token arrives at the *Check_status* place, either the *offcampus_condition* or the *oncampus_condition* transition is enabled after each condition statement attached to its transition is evaluated with the current token and user profile values. The condition in *offcampus_condition* is "*(r.w.accessRight < 0.8 && r.w.network != '128.194.147' && r.u.userType != 'faculty')";* The condition in *oncampus_condition* "*(r.w.accessRight >= 0.8 || r.w.network == '128.194.147' || r.u.userType == 'faculty')"*. When the current token's inferred *accessRight* is larger than or equal to 0.8 (i.e., rather high), or *network* is '128.198.147', or *userType* is *'faculty'*, the *oncampus_condition* transition is enabled; otherwise, the *offcampus_condition* transition is enabled. When the transition is fired, the assignment statement (*r.w.class = 'oncampus'* or *r.w.class = 'offcampus'*) attached to output arc of the transition is executed.

After the user is classified, a token is in the *Branch* place. A token with the user class value '*oncampus' (*or '*offcampus')* has the *Oncampus* (or *Offcampus*) transition enabled and fired. Now the user (on-campus or off-campus) will have tokens in both *template* and *node0* places, and the user is ready for a guided digital library tour. Up to this point, all transitions are executed by the system without interaction with the user.

Document Display Using the Template. Each classification includes a separate template file to display a full version of the page to the on-campus user and a limited one to the off-campus user. The content of “template” node for on-campus users is as follows:

```
<html>
<body>
some descriptive text ......
<Append place=”node0”>
some descriptive text ......
<Append place=”node1”>
<Append place=”node2”>
<Append place=”node3”>
<Append place=”node4”>
<Append place=”node5”>
<Append place=”help”>
<Transition text=”next”>
</body>
</html>
```

This template file is used for combining contents of active nodes (nodes that have tokens) specified in an <*Append*> construct. When only *node0* and *template* places are active, the content of *node0* is used for generating the current Web page. Then, when *node1, node2* and *help* including *template* are active (when the next transition *On_Campus_Service* of the on-campus user, who accesses the net during regular office hours, is fired), the contents of these nodes are used for generating the current Web page. Also when the on-campus user accesses the net at a non-regular office hour, the content of *help* is not used for generating the output Web page. Thus, only the limited version (no on-line help) is available outside of normal operating hours.

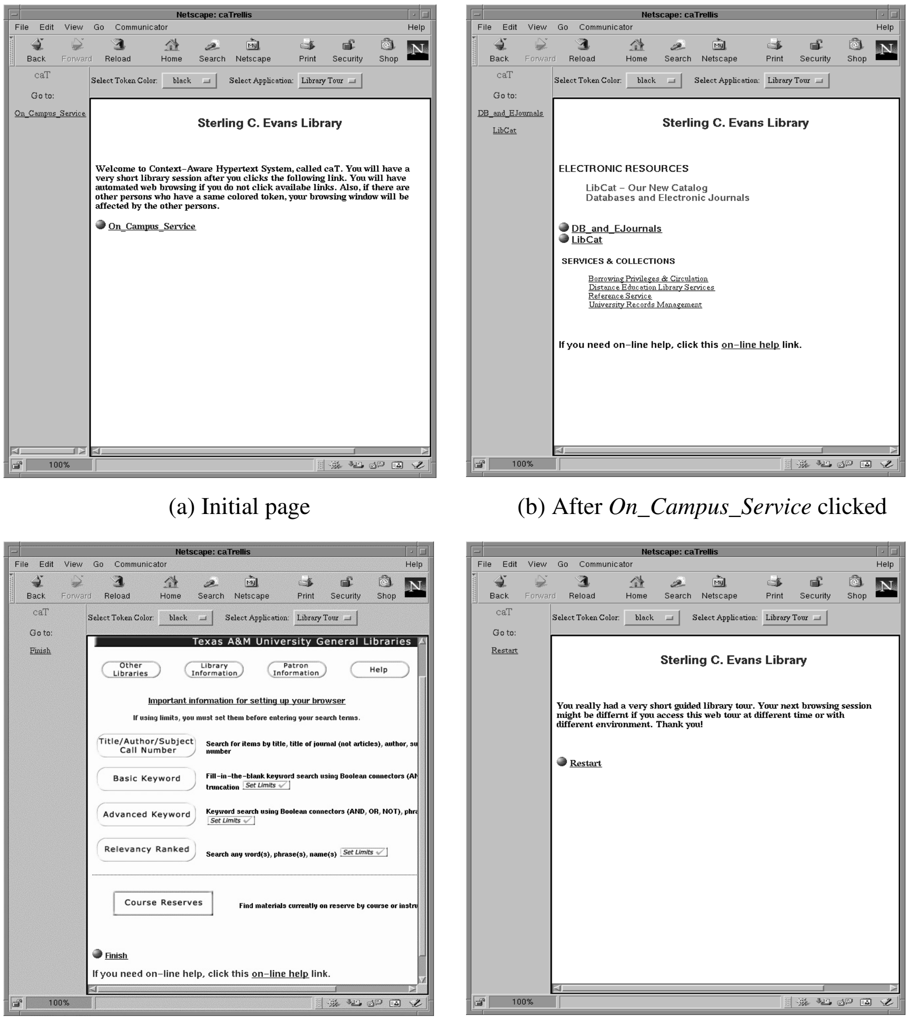
The *template* node for off-campus users may have the same template file as on-campus users. But because there are no *node2*, *node4*, and *help* in the subnet of off-campus users, only *node0*, *node1*, *node3*, and *node5* are used for generating the output Web page. Therefore, off-campus users will have a limited version compared to on-campus users. When tokens move around the net, the contents of the Web page change dynamically.

A <*Transition*> construct is used for defining an embedded link that allows the user to move to the next state. If the output transition of the *template* node is enabled, that transition link is included in the combined Web page. This enables the user to fire the transition to move to the next state. Possible internal links, such as links between *node0* and *node5*, are generated automatically in the combined Web page if these are enabled transitions.

For the detailed behavior of the bottom portion of the net, consider the case of on-campus users who access the net during regular hours. The *On_Campus_Service* transition has a condition statement “*r.w.currentTime >= 9.0 && r.w.currentTime <= 17.00*”; *Off_Campus_Service(no help)* transition has “*r.w.currentTime < 9.0* || *r.w.currentTime > 17.00*”. Therefore, when the user accesses the net during regular office hours, *On_Campus_Service* will be enabled, otherwise *On_Campus_Service(no help)* is enabled. The resulting page is shown in Figure 3 (a). Introductory text (the content of *node0*) and the *On_Campus_Service* link are displayed in a main frame, and a duplicate *On_Campus_Service* link in a left control frame. Since *On_Campus_Service* has a time value $(0,\infty)$, the system will wait for the user’s input (i.e., a mouse click). When a time value is not declared, $(0,\infty)$ is the default value.

When the *On_Campus_Service* transition is fired, the contents of *node1*, *node2*, and *help* are used for generating the current Web page (see Figure 3 (b)). *LibCat* and

DB_and_Ejournals links are displayed with additional text and WWW links, and the user can select one of these links for the next navigation. The caT links that are mapped to active transitions in the net (called "caT links" in this section for distinguishing them from WWW links) is distinguished with a preceding circle bullet, and normal WWW links are shown without preceding bullets. When the user clicks caT links, the corresponding transition is fired, the net state is changed, and the new content is regenerated using *template*. When the user clicks WWW links, the resulting effect is the same as normal WWW browsing semantics, but the user still can move to the next state using caT links placed in the left control frame. Since *LibCat* has a time value (0,20) and *DB_and_Ejournals* (0,∞), *LibCat* will be selected automatically by the system unless the user responds within 20 time units (i.e., seconds).



caT

Go to:

On_Campus_Service

Select Token Color: black

Select Application: Library Tour

Sterling C. Evans Library

Welcome to Context–Aware Hypertext System, called caT. You will have a very short library session after you clicks the following link. You will have automated web browsing if you do not click availabe links. Also, if there are other persons who have a same colored token, your browsing window will be affected by the other persons.

● On_Campus_Service

(a) Initial page

caT

Go to:

DB_and_EJournals

LibCat

Select Token Color: black

Select Application: Library Tour

Sterling C. Evans Library

ELECTRONIC RESOURCES

LibCat – Our New Catalog
Databases and Electronic Journals

● DB_and_EJournals
● LibCat

SERVICES & COLLECTIONS

Borrowing Privileges & Circulation
Distance Education Library Services
Reference Service
University Records Management

If you need on–line help, click this on–line help link.

(b) After *On_Campus_Service* clicked

caT

Go to:

Finish

Select Token Color: black

Select Application: Library Tour

Texas A&M University General Libraries

Important information for setting up your browser

If using limits, you must set them before entering your search terms.

Title/Author/Subject Call Number — Search for items by title, title of journal (not articles), author, su number

Basic Keyword — Fill–in–the–blank keyword search using Boolean connectors (AN truncation Set Limits

Advanced Keyword — Keyword search using Boolean connectors (AND, OR, NOT), phra Set Limits

Relevancy Ranked — Search any word(s), phrase(s), name(s) Set Limits

Course Reserves — Find materials currently on reserve by course or instru

● Finish

If you need on–line help, click this on–line help link.

(c) After *LibCat* clicked

caT

Go to:

Restart

Select Token Color: black

Select Application: Library Tour

Sterling C. Evans Library

You really had a very short guided library tour. Your next browsing session might be differnt if you access this web tour at different time or with different environment. Thank you!

● Restart

(d) After *Finish* clicked

Fig. 3. Context-aware display

When *LibCat* is clicked, LibCat (the catalog of Texas A&M University General Library) will be shown (see Figure 3 (c)). Now the user can search for library items using the LibCat search engine. Since the *finish* link has a time value (0,60), the user has to finish his (or her) search work within 60 time units, otherwise the user's window will be forced to move to the next state window (see Figure 3 (d)). The time value can easily be adjusted to a reasonable value by an author of this net, or, indeed, adjusted dynamically based on the reader's performance [19]. When the *restart* link is clicked, the user will be reclassified as being an "on campus" or "off campus" user based on new environmental values: time of day, distance from campus, and incoming IP address (in current example, only time of day is dynamically changing).

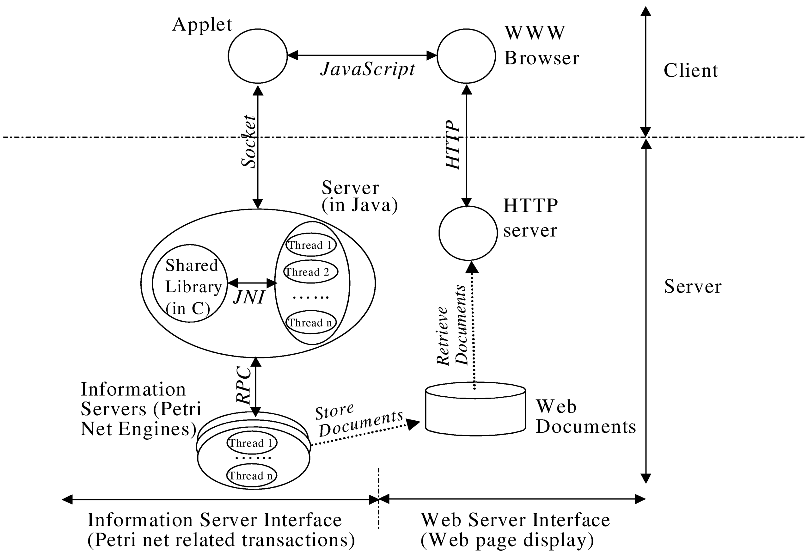


Fig. 4. General architecture of a flexible information tool

Implementation. In the caT implementation, beginning with χTrellis for the X-windows environment, a distributed client/server network achieves cooperative separation between net and interpretation. Every caT model is an instance of an information server – an engine that receives remote procedure call (RPC) requests for its services. Clients are separate processes that have visual user interfaces and communicate with one or more engines via RPC. The Trellis browser has been implemented in C, C++, and Motif, and it runs in a Unix environment. A new Web version of the browser has been designed and implemented using Java language: the Java applet first communicates with an intermediate message-handling server, and the server passes RPC requests using Java JNI (Java Native Interface) to the information servers. The general architecture of the new browsing tool is shown in Figure 4.

4 Discussion

One of the primary advantages of using Petri net models is that the same model is used for the analysis of behavioral properties and performance assessment, as well as for systematic building of systems [24]. Therefore, authors can use Petri net analysis techniques to verify and validate the behavioral characteristics of developed hypertext systems before they are handed over to users [20]. Firstly it is possible to verify that all nodes in a hypertext can be reached via some path; more important, it also is possible to verify that certain nodes cannot be reached from particular initial markings, giving the basis for access control. Additionally, the following characteristics may be verified (or simulated): terminal state existence (i.e., if a state *m* exists in which no transitions are enabled), maximum or minimum time for a specific document navigation (using timing statements in transitions: a release and a maximum latency times), certain collections of information that can be viewed simultaneously, etc. To support these features, the extended model has been designed to be consistent with Petri net theory. However we expect that dynamic characteristics of environment data including user-modeling data may make the verification of the system difficult. For example, the combinations of many dynamic data may generate very large net states. We plan to analyze the system with only a small amount of dynamic data considered at one time. We are developing analysis tools to be added to the current authoring tool, so authors can verify behavioral characteristics of developed hypertext documents.

Additionally, top-down or bottom-up structured authoring support will be essential for developing large-sized Petri net applications since it reduces graphical complexity problems that are common to graph-based modeling tools. Current Trellis supports hierarchical net browsing using a hyperprogamming feature, but it does not support built-in hierarchical nets. In other words, when a token arrives into a place where another net (i.e., a subnet) is declared to be invoked, a browsing tool can only handle the subnet invocation using a content attribute of a place, and show the output interface of the subnet in a new process. But an information server itself does not invoke the subnet since it ignores the content attribute of the place. Moreover, there are no closely coupled interactions between hierarchical nets, such as data value (or token value) passing. Shifting hierarchical net handling from content-level (i.e., a browsing tool) to net structure-level (i.e., an information server) will be more appropriate for flexible net interactions. When data passing between nets is supported, subnets can be used as templates, like functions or procedures in high-level programming languages. This will increase reusability of subnets. The structure-level hierarchical net feature will be very useful for structured authoring, and also will support easy system tracing for simulation and debugging purposes.

In summary, we have introduced caT, a context-aware hypertext model and associated tools, which supports flexible adaptation in a dynamically-changing environment. caT is an extension of the earlier Trellis model in the following ways:

- It supports both high-level Petri net and user modeling features to provide flexible context-aware adaptation,
- It integrates a fuzzy logic tool with the current model to support fuzzy (or uncertain) knowledge handling, and
- It supports a Web-based browsing tool.

The successful result of the current work shows the potential usability of context-aware hypertext systems. Potential target applications of this model are context-aware hypermedia applications running on mobile systems with attached sensors to capture their environment. The research is continuing in order to enhance the usability of the caT model.

References

1. ABOWD, G.D., ATKESON, C.G., HONG, J., LONG, S., KOOPER, R. and PINKERTON, M. "Cyberguide: A Mobile Context-Aware Tour Guide", *ACM Wireless Networks 3,* (1997), pp. 421–433
2. DE BRA, P., HOUBEN, G. and WU, H. "AHAM: A Dexter-based Reference Model for Adaptive Hypermedia", *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia,* Darmstadt, Germany, ACM Press, (1999), pp. 147–156
3. BROWN, P.J. "Triggering information by context", *Personal Technologies,* vol. 2, no. 1, (September 1998), pp. 1–9
4. BRUSILOVSKY, P. "Methods and techniques of adaptive hypermedia", *User Modeling and User Adapted Interaction,* vol. 6, no. 2-3, (1996), pp. 87–129
5. CARDOSO, J., VALETTE, R., and DUBOIS, D. "Fuzzy Petri Nets: An Overview", *13th IFAC World Congress,* San Francisco USA, (30 June – 5 July 1996), pp. 443–448
6. DEY, A.K. and ABOWD, G.D. "Toward a Better Understanding of Context and Context-Awareness", *Georgia Institute of Technology, GVU Technical Report GIT-GVU-99-22*, (June 1999)
7. DEY, A.K., SALBER, D., ABOWD, G.D., and FUTAKAWA, M. "The Conference Assistant: Combining Context-Awareness with Wearable Computing", *The Third International Symposium on Wearable Computers,* (1999), pp. 21–28
8. FURUTA, R. and STOTTS, D. "Trellis: a Formally-defined Hypertextual Basis for Integrating Task and Information", In Olson, G.M., Smith, J.B., and Malone, T.W., editors, *Coordination Theory and Collaboration Technology,* to appear in 2001
9. GENRICH, H.J. and LAUTENBACH, K. "System modeling with high-level Petri nets", *Theoret. Comp. Sci.,* vol. 13, (1991), pp. 109–136
10. GUDWIN, R. and GOMIDE, F. "Object networks – a modeling tool", *Fuzzy Systems Proceedings, 1998 IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference on Volume: 1*, (1998), pp. 77–82
11. HALASZ, F. and SCHWARTZ, M. "The Dexter Reference Model", *Proceedings of NIST Hypertext Standardization Workshop*, (1990), pp. 95-133
12. JENSEN, K. "Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1", *EATCS Monographs on Theoretical Computer Science,* Springer-Verlag, (1992)
13. The MathWorks, Inc. "Fuzzy Logic Toolbox User's Guide", (1999)
14. NA, J. and FURUTA, R. "Context-Aware Hypermedia in a Dynamically-Changing Environment, Supported by A High-Level Petri Net", short paper, *Proceedings of Hypertext '2000,* San Antonio TX, USA, (2000), pp. 222–223
15. PETERSON, J.L. "Petri Net Theory and the Modeling of Systems", *Prentice-Hall,* Englewood Cliffs, N.J., (1981)
16. RHODES, B.J. "The Wearable Remembrance Agent", *Proceedings of 1st International Symposium on Wearable Computers, ISWC '97,* Cambridge, MA, (October 1997), IEEE Press, pp. 123–128
17. SCHILIT, W.N., ADAMS, N., and WANT, R. "Context-aware computing applications", *Proceedings of the Workshop on Mobile Computing Systems and Applications,* Santa Cruz, Ca., IEEE Computer Society Press, Los Alamitos, Ca., (1994), pp. 85–90

18. STOTTS, P.D. and FURUTA, R. “Petri-net-based hypertext: Document structure with browsing semantics”, *ACM Transactions on Information Systems*, vol. 7, no. 1, (Jan. 1989), pp. 3–29
19. STOTTS, P.D. and FURUTA, R. “Dynamic Adaptation of Hypertext Structure”, *Proceedings of Hypertext ’91*, (1991), pp. 219–231
20. STOTTS, P.D., FURUTA, R., and CABARRUS, C.R. “Hyperdocuments as Automata: Verification of Trace-Based Browsing Properties by Model Checking”, *ACM Trans. On Information Systems,* vol. 16, no. 1, (1998), pp. 1–30
21. WANT, R., HOPPER, A., FALCAO, V., and GIBBONS, J. “The Active Badge Location System”, *ACM Transactions on Informations,* vol. 10, no. 1, (1992), pp. 91–102
22. WANT, R., SCHILIT, B., ADAMS, N., GOLD, R., PETERSON, K., ELLIS, J., GOLDBERG, D., and WEISER, M. “The PARCTAB Ubiquitous Computing Experiment”, *Technical Report CSL-95-1, Xerox Palo Alto Research Center,* (1995)
23. ZADEH, L.A. “Knowledge representation in fuzzy logic”, *IEEE Trans. Knowledge and Data Engineering*, vol. 1, no. 1, (1989), pp. 89–100
24. ZURAWSKI, R. and ZHOU, M. “Petri Nets and Industrial Applications: A Tutorial”, *IEEE: Transactions on Industrial electronics*, vol. 41, no. 6, (December 1994)

Robust Hyperlinks: Cheap, Everywhere, Now

Thomas A. Phelps and Robert Wilensky

Division of Computer Science
University of California, Berkeley
Berkeley, CA 94720-1776
{phelps, wilensky}@cs.berkeley.edu
http://www.cs.berkeley.edu/~phelps/Robust/

Abstract. We propose *robust hyperlinks* as a solution to the problem of broken hyperlinks. A robust hyperlink is a URL augmented with a small "signature" consisting of carefully chosen words taken from the referenced document. If the address-based portion of the URL fails, this content-based signature can be submitted as a query to web search engines to locate the document. It turns out that very small signatures are sufficient to readily locate individual documents out of the billion on the web, even if the document is modified.
Robust hyperlinks ex hibit a number of desirable qualities: They can be computed and exploited automatically, are small and cheap to compute (so that it is practical to make all hyperlinks robust), do not require new server or infrastructure support, can be rolled out reasonably well in the existing URL syntax (so they can retrofit existing links to make them robust), and are easy to understand. One can start using robust hyperlinks now, as servers and web pages are mostly compatible as is, while clients can increase their support in the future.
Robust hyperlinks is one example of using the web to bootstrap new features onto itself.

Lexical Signature: cnrp macskassy hypercafe shklar multivalent belongie blobworld bregler cityquilt cyberbelt

1 Introduction

Hypertext research has long been concerned with the problem of the persistence of hyperlinks, that is, of dealing with problems that arise when one endpoint of a link, especially the destination, is unresolvable, either because it was deleted, renamed, moved, or otherwise changed. On the Web, one manifestation of this problem is the broken hyperlink. The Georgia Tech WWW User Survey [1] reports that users rate "broken links" as the third biggest problem on the web (after "speed" and "slow ads"). A World Wide Web Consortium study of AOL server logs determined that between five and eight percent of all requested links are broken [2].

A number of solutions have been proposed to deal with this problem. Some approaches suggest reliance on some additional naming scheme, such as Uniform Re source Names (URNs) [3], handles or digital object identifiers (DOIs) [4], Persistent Uniform Resource Locator (PURLs) [5] or Common Names [6]. Other approaches involve monitoring and notification to insure referential integrity, such as [7]), [8], [9], [10], [11], and, apparently, the commercial product LinkGuard [12], among numerous others. [13] surveys previous solutions, in addition to the above (which he terms "unique names" and "hyperbases"), considering user onus (users keep their links up-to-date), forward references, and operating system extensions (that would maintain links to files as they are moved on a given hard disk).

In this paper, we demonstrate a different approach to this problem. This is to augment URLs so that they become *robust hyperlinks*. A robust hyperlink offers a high probability of being successfully resolved even after the target page had made an unannounced move and left no forwarding address, even if the page had been edited as well.

Note that, in contrast to alternative solutions to the broken link problem, the robust hyperlink approach puts the burden of additional effort on the party creating the hyperlink, rather than the party administrating the resource. One implication of this fact is that no buy-in is required by an administrative unit, as by the group system administrator or site's webmaster. Similarly, hyperlinks could be made robust piecemeal, a link at a time, rather than require use on a more systematic basis. That is, motivated users or ISPs can take advantage of robust hyperlinks without waiting for support from web sites.

We believe these practical advantages of robust hyperlinks should facilitate their adoption. Specifically, robust hyperlinks exhibit the following desirable properties:

- Robust hyperlinks provide a very high likelihood of successful dereferencing, with very few false positives, in those cases in which an item is moved, but has otherwise been left largely unchanged. Moreover, performance gracefully degrades as document content changes from its state at the time the hyperlink was created.
- Making a conventional hyperlink robust can be fully automated, and is not computationally expensive. An author can point to a hyperlink and automatically transform it into a robust hyperlink, usually instantaneously. Similarly, one can readily automate making robust all the hyperlinks on a document or web site.
- The additional storage required for a robust hyperlink is relatively small, so that it is practical to make all URLs robust.
- Robust hyperlinks are useful without changes to web infrastructure outside of a user's control, which is to say, almost all web sites and web browsers, and thus can be exploited immediately. Likewise, robust hyperlinks are largely non-interfering with clients and services that do not support them.
- Full integration of robust hyperlinks will require support, however minimal, from clients or from proxies with which the user can interact. However, support by clients or proxies is straightforward, thus facilitating widespread adoption.
- When the robust character of the hyperlink is not needed, robustness does not impose a performance penalty.

In this paper, we first describe a mechanism to augment hyperlinks to make them robust, and present evidence of its viability. We then consider various ways to integrate this scheme into the current web infrastructure. We also discuss limitations of this approach, and consider ways it could be refined. Finally, we suggest improvements to the infrastructure to more fully support robust hyperlinks.

2 Providing Robust Hyperlinks

The basic idea for making hyperlinks robust is extremely simple. It is to include in the hyperlink, along with the URL, some part of the document content. We call this content a *lexical signature*, as it is meant to identify the given page by its content. A robust-hyperlink-aware agent will generally perform an initial attempt at traditional address-based dereferencing, that is, a URL lookup ignoring the signature. However, if traditional dereferencing fails, the client enters into a second phase of content-based dereferencing, in which it uses the signature to search for documents whose signature most closely matches that in the robust hyperlink. The user is then presented with the matching documents from which to complete the reference.

2.1 Computing Lexical Signatures

The first issue we are faced with is how to create signatures that have the desired properties. Basically, we want small signatures to effectively select the documents they were computed from, but also, be robust in the presence of document modification. More precisely, we can list the following desirable properties of signatures:

- Short signatures should effectively pick out few documents.
- Subsequent changes to a document should have minimal impact on signature effectiveness.
- The addition of new documents should have minimal impact on previous signature effectiveness.
- Signature effectiveness should be largely search-engine-independent.

Note that at times, these requirements are at odds, presenting us with a "uniqueness-robustness trade-off". That is, to effectively pick out documents, simply selecting the rarest terms would be appropriate. However, the resulting signatures, while short and effective, may not be robust with respect to minor document changes. For example, a single term that occurs just once in the entire web is by itself a very short and highly effective signature for the lone document that contains it. However, this term may be a typographic error, and hence will soon be changed, rendering it completely, irrecoverably ineffective.

However, we don't have a model of how web pages change, or what pages are likely to be added, so we have improvised heuristics to capture our beliefs about these. Specifically, we propose the following:

- First, determine the term frequency (TF), of each unique word in the document. Web search engines such as AltaVista report this number. The rare terms better distinguish among documents.
- Since rare terms in the web often occur only once in a document, favor terms that occur more frequently in the document by multiplying TF by the word's inverse document frequency (IDF). However, cap IDF at 5, as more frequent use of a term in a document does not appreciably add to editing robustness but does dilute the contribution of rarity. (Avoiding words that appear to be misspellings seems appealing. [14] attempt to determine misspellings by creating a lexicon by web crawling and removing very low frequency words, but do not describe how well this technique works.)
- Finally, select the few best such terms subject to other robustness heuristics. One such heuristic is that all the terms should not occur very close by, say, in the same sentence.

We have no model of what new documents will be added, and assume that, roughly, the current distribution of terms will persist. Therefore, choosing rare terms also minimizes the chance that another document will be added with this term.

In order to maximize the applicability of terms to various search engines, we have taken a conservative definition of "term". Terms must include at least four letters, include no numbers, and be found in "content" only (not in comments, scripts, meta tags, or within any type of tag). Terms are normalized to lowercase, as search engines may or may not support case distinction; if not, signatures that assume case distinctions can be significantly compromised. (One experiment we conducted, performed by Hao Chen, suggests that retaining case does not significantly improve results even with those search engines that make a case distinction.) Similarly, terms must be words, and not phrases, as not all search engines support phrases in queries.

Possible refinements to improve both robustness and effectiveness of signatures are considered below, in the section "Signature Creation".

2.2 Empirical Results

Preliminary empirical results suggest that lexical signatures of about five terms are sufficient to determine a web resource virtually uniquely, out of the more than one billion pages on the web [15]. That is, in most cases, a query to a search engine requesting documents which contain all of the terms in the signature will cause a unique document to be returned, namely the desired document. In those few cases in which more than one document is returned, the desired document is among the highest ranked. In those case in which a particular search engine returns no matching documents, this is generally because the document has not yet been indexed, or has been substantially edited since it was last indexed.

Indeed, fewer that five terms will probably suffice. Nevertheless, we advocate at least these many terms because the redundancy that is provided is useful with respect to document change and because the additional terms may be needed to distinguish documents as the web continues to grow.

Table: Sample Signatures and Query Results. URLs are presented along with their signatures and rank of that exact URL in the result set of each search engine to which the signature is submitted. (G=Google, A=AltaVista, Y=Yahoo, H=Hotbot, I=Infoseek.) In addition, the server software of the URL's host is listed, along with whether that server accepts the "robust URL" syntax described below. (In the online version of this paper, the contents of the "Robust ok?" cells are hyperlinks that use robust URLs, and the search engine rank cells are hyperlinks that query the respective search engine with the signature.)

URL and signature	**Server**	**Robust ok?**	**G**	**A**	**Y**	**H**	**I**
http://www.cs.berkeley.edu/~daf bregler interreflections zisserman cvpr iccv	Apache 1.3.4	yes	1	6	1	1	1
http://www-diglib.stanford.edu/diglib/pub sdlip interbib sdlt infobus testbed	Apache 1.3.4	yes	1	1	1	1	4
http://www.hotofftheweb.com/ servicemarks moskowitz mustache scrapbook surfers	ApacheSSl 2.4.1/1.3.3	yes	1	2	1	1	1
http://developer.apple.com/techpubs/macos8/Legacy/OpenDoc/opendoc.html opendoc webojbects software constants reference	Netscape-Enterprise 3.5.1G	yes	1	?	1	?	?
http://www.rightbrain.com/pages/book-download.shtml thinkinginpostscript ematter click infringe	BESTWWWD 2.4	yes	?	2	?	?	?
http://msdn.microsoft.com/workshop/author/css/css.asp ystyles dblspaced selamoglu intdev intalicizes	Microsoft-IIS 5.0	yes	1	1	1	1	1
http://www.adobe.com/products/acrobat/main.html accessibility hewson epaper gillmor workflow	Netscape-Enterprise 3.6 SP 2	yes	?	1	1	1	?
http://www.isg.sfu.ca/~duchier/misc/hypertext_review/ balasubramanian bala hypermedia Pegasus interface	Apache 1.3.6 (Unix) PHP/3.0.7 mod_ssl/2.3.11 OpenSSL/0.9.3a	yes	1	10	?	?	?
http://www.ai.univie.ac.at/~paolo/lva/vu-htmm1998/html/coklin98/conklin87.html planetext fernec synview peroperties textnet	Apache 1.3.3 (Unix) Debian/GNU	yes	?	?	?	1	?
http://www.lcc.gatech.edu/gallery/hpercafe/HT96_HTML/HyperCafe_HT96.html cityquilt hypervideo cyberbelt infrawhere videotexts	Apache 1.3.6 (Unix)	yes	1	1	1	1	1

As an example, we computed signatures for a varied, unpremeditated sample of different sorts of web pages. (For a different set of examples, see the hyperlinks in this paper.) Most of these are for papers referenced by a bibliography maintained by one of the authors, as we feel this is a realistic application of the technology. To these we added a personal home page, a research web site, and a commercial home page. For each case, we computed signatures, and then perform straightforward queries using several search engines. (That is, we just supply the engines with the signature terms, without using any advanced features of the engines.) In Table 1, we report the rank of the document in the result set (or "?" if it is absent in the first page of results, which is usually the first ten results).

These results suggest that a number of signature-based dereferencing strategies are feasible. For example, an agent could query a set of engines, and return the top few results from each one, or make one query to a metasearch engine, which translates and distributes the query to numerous other search engines. In our sample set, each hyperlink is successfully dereferenced by this strategy.

Alternatively, an agent could make "stringent" queries to one or more engines (queries insisting that all the terms be present), and, if this fails to return a result, make progressively less stringent queries. (While it is not obvious in the table, in most cases, only one or two documents are returned by the more stringent searches. In most other cases, the stringent searches returns no items, probably because the document was substantially modified since the last time the crawler reached it.) Performing the Google query, and then performing the Alta Vista query if Google fails, locates the desired reference in all but one case.

Of course, most of these search engines (but not Google) offer "advanced options" that afford the user more control, leaving open the possibility of additional strategies.

The results may actually be better than the table would suggest. For example, in several cases an identical paper with a different URL occurs earlier in the result set. Similarly, some of the other pages may produce access to the document a link or two away, so even though we do not count this as successful, it might be helpful to the user.

Indeed, in our examples above, cutting the signature length down to three terms changes the query results only slightly. (One signature would fail to readily locate its target in this case.) Note, though, that robust hyperlinks do not require a standard length signature, so different implementations are free to use different lengths as well as different methods of computing them.

These results are consistent with previous findings that it is possible to excerpt fairly short pieces of document content to serve as good retrieval keys for that document. For example, [16] generate *summary queries* by iteratively selecting terms and phrases from a document, based on the term's frequency and position within the document. They report that queries whose average length was 8.6 words returned the target document with the average rank of 1.24 in AltaVista. [14] construct *strong queries*, for the purpose detecting duplicate pages, by choosing a fixed number of document terms having the smallest frequency counts in a lexicon they construct by web crawling, eliminating very low frequency words. They suggest that 8 word queries are adequate, but do not report results about document ranks.

As [16] point out, since there are a large number of different terms on the web, if terms appear independent of each other in documents, only a small number would need to be chosen to form a query whose result was unique. Making a conservative estimate of 500,000 distinct terms, the number of distinct combinations of 5 terms is greater than 3x10^28. Assuming the web is populated by documents whose most characteristic terms are uniformly drawn at random, the probability that more than one document matches a set of 5 characteristic terms is very small indeed.

Of course, the assumption of uniform distribution is highly questionable. Perhaps the empirical results indicate that it is not that far off in practice. Interestingly, even among intuitively similar documents (for example, separate chapters of the same book), signatures seem not to overlap much. In addition, lexical signatures are by definition skewed toward infrequent terms: Most of the signatures we have seen contain domain-specific abbreviations, proper names, jargon, et cetera, which may each occur only in a few dozen documents, narrowing down the set of matching documents very rapidly.

We find these results are encouraging, if only impressionistic. We have automated the process of signature creation and subsequent searching, and have found these result to be consistent with our (still somewhat limited) experience. For example, we created signatures for all the references at the end of paper. The results are virtually identical to those in the table. However, we have resisted the temptation to report more thorough empirical testing for several reasons. In the first place, definitive proof requires testing over time, waiting while pages are moved and edited by real users, then measuring the effectiveness of signatures to find those pages. In the second place, as we discuss below, there are many ways in which one might vary and possibly improve the details of this scheme (all of which are mutually compatible). Thus, we present sample empirical results to demonstrate feasibility, and to encourage experimentation and use.

3 Integrating Robust Hyperlinks into the Web

3.1 Encoding Signatures in URLs

Given that lexical signatures are a good way to augment URLs, we are left with the issue of how to associate these with hyperlinks. Here we discuss several alternatives. For the purposes of this discussion, suppose that the hyperlink has the URL *http://www.something.com/a/b/c*, and that the designated resource has the signature *w1,...,w5*.

If signatures could be incorporated in URLs, all elements of the web infrastructure would have an opportunity to exploit them, which is highly desirable. Even without any support from existing browsers, servers, or search engines, 404 Page Not Found errors could be resolved by manually taking the signature and feeding it to a seach engine. And if one's client or proxy server is "robust-hyperlink-aware", it could strip the signature expression before attempting traditional dereferencing for maximum compatibility with old infrastructure, and automatically performing search engine lookup on 404 errors.

Of course, one alternative is to modify URLs to provide a special syntax for signatures. Doing so is incompatible with existing URLs, and makes for an awkward transition in adopting the scheme. Hence we do not consider it further, other than to note that, should robust URLs come into widespread use, such a proposal might merit further consideration.

Instead, we consider various ways signatures can be incorporated into the current URL syntax:

- Signatures as queries: A signature can be appended to a URL as if it were a query term:
  `http://www.something.com/a/b/c?lexical-signature="w1+w2+w3+w4+w5"`
  (If the URL already includes a query, append the signature expression as another name-value pair, with an "&".)
- Signature as privileged query parameter: Signature queries can be made more concise by granting them special status as the "anonymous" parameter, as in: `http://www.something.com/a/b/c?w1+w2+w3+w4+w5`
  This can be read, "Document available at URL? If not, use signature."
- Signatures as URI parameters: An alternative is to add signatures as URI parameters:
  `http://www.something.com/a/b/c;lexical-signature=w1+w2+w3+w4+w5`

Interestingly, only the first alternative seems largely non-conflicting with current web infrastructure. The privileged location conflictes with the old, deprecated ISINDEX tag, which, though obsolete since forms were introducted into HTML, can still be found on the web. URI parameters, though rarely used, cause commonly deployed services to generally return errors when given such URLs.

In contrast, most widely used HTTP servers seem to ignore query terms for non-script URLs (and most services seem to ignore what appear to be gratuitous query parameters). This is true of the major web servers in use today: Apache (with over 50% of the market), Microsoft Internet Information Server (24%), and Netscape Enterprise (7%) [17]. The table above reports, in the "Robust ok?" column, the results of submitting a robust URL in place of each original URL. All attempts succeeded.

The "signatures as queries" scheme is not foolproof, but the drawbacks seem mininal: The term "lexical-signature" might in fact be a valid search term for some (exceedingly small number of) services; but if the term becomes widely recognized as established for this purpose, well-informed web designers will know not to use it. Similarly, some web servers may be hostile to unknown parameters or ones used in a nonstandard way; again, if the scheme becomes popular, perhaps they will be updated to accept this usage. Therefore, we have decided to implement this alternative.

3.2 Recommended System Support for Robust Hyperlinks

Via the robust URL approach, robust hyperlinks can be made effective without any change to existing web infrastructure. Nevertheless, that infrastructure could be improved to more cleanly accommodate them.

Robust Browsers. To take advantage of robust hyperlinks conveniently, relieving users of the mechanics of content-based recovery, some piece of software needs to exploit them. Ideally, browsers will support them directly. Browser support would require no changes in servers or other infrastructure, so users would benefit as soon as they update their browser.

Before robust-aware clients send robust hyperlinks to a web server, the signature would be stripped off so as to eliminate the chance of adverse interactions. If a server returns an HTTP error code 404, the browser would engage in signature-based dereferencing, that is, send the signature to one or more search engines, and present the user with alternatives from with to select a document.

Furthermore, browsers could noninvasively make unsigned URLs robust by signing those pages automatically and maintaining a database that associates URLs with signatures.

Robust-aware web browsers should be configurable to consult several search engines in looking up a signature. This is useful because different search engines index different portions of the web, so using several different web-wide search engines is likely to provide better coverage. In addition, large numbers of documents live behind local firewalls, which prevent them from being indexing by general web-wide engines. However, many sites have local engines that can find local documents, so adding these to one's browser's search engine list would extend robustness to local document collections.

Robust Proxy Module. Until browser support is available, the major benefits of robust URLs can be externalized via a proxy server. A proxy can give all browsers robust URL support without modification, and also, allow sites one at a time to support robust URLs. A disadvantage is that it requires convincing the local web site administrator to install the service on a proxy server.

A robust proxy module transparently makes robust all URLs that pass through it, from any client connecting to any server, keeping the results in a table as described above. When a client requests a page with a non-robust URL, the module signs the returned incoming page optionally and sends a redirect HTTP signal to the client to transparently supply it with the robust hyperlink. This hyperlink can then be saved as a bookmark, used in an HTML document, or emailed. Otherwise, the proxy server simulates browser support by stripping the signature from robust URLs sent to servers for maximum compatibility, and taking action on 404 errors.

Web Page Authoring, Site Management Tools, and Servers. In addition to verifying current link validity, web page authoring and site management tools could compute or update page signatures.

Although such tools typically keep a database of web page locations in order to update references when a page is moved, doing so requires some bookkeeping in the form of a redirect page (or entry in a table that dynamically generates the page or HTTP redirect code) for references to the old address. Unfortunately, this bookkeeping might be needed indefinitely, as there is no way to track down all the references to the old location stuffed in individual, personal bookmark files unavailable to the site.

This bookkeeping can be done away with or allowed to expire with confidence if the server itself tries to resolve robust hyperlinks locally. Since the site is probably already indexed to enable site searches, resolving robust hyperlinks locally requires only a small amount of additional work to invoke the search engine and interpret the results as follows: If a single high confidence result is found, its location could be returned with an HTTP redirect code; otherwise, a list of likely candidates could be reported; or if no good matches are found, a 404 could be returned to trigger the client's own recovery.

Search Engines. Determining a term's document frequency is done by consulting a search engine. In practice, an implementation must glean this number by plucking out the count from the mass of dynamically generated HTML, the template for which changes regularly. Cooperative search engines could return this information in a format that can be reliable parsed.

Standard search engine queries are not tuned for finding signatures. For example, a Google query or an AltaVista query with all signature terms required succeeds only when all the terms are present in a document, which means they are not robust to a missing signature term. An AltaVista query that does not require matching terms will find the most relevant pages, but unfortunately include many which do not necessarily include all of the query terms. An obvious tactic to try is to drop one or more terms as needed until a non-empty result set is returned. However, choosing a subset of terms performing this from the client is inefficient. A search engine tuned to signature resolution would strongly favor documents that include all terms, but be robust to one or missing terms as necessary.

Ordinarily the signature terms give a small result set of matches. Even when content-based dereferencing produces more than one candidate document, when these are not duplicates, then, generally, only one of these candidate documents has a signature that in any way resembles the submitted signature. Thus, search engines could compute page signatures as the page is indexed, and be able to report singleton matches of very high confidence in almost every case when the page is indexed at all. Doing so would allow robust-aware browsers to automatically follow that link, so the user is unaware of the potential failure, and to update the link in a bookmark or moved-page map. In cases of less confidence, the search engine can rank hits by signature rather than document score to faciliate manual correction.

4 Limitations

The proposed scheme is not without its limitations:

Non-indexed documents. Many important classes of documents, for example, PostScript, DVI, Microsoft Word, and compressed documents are generally not indexed. For general document searching comprehensiveness, search engines could easily "string scrape" these formats with minimal understanding of the complete format. As they do so, the utility of signatures will improve.

For the classes of documents that are indexed, the scheme will be ineffective for a given document during the period in which the document has moved, but no search engine has yet reindexed it. As web search engines improve or worsen their reflection of the state of the web, robust hyperlink coverage will improve or worsen with it. Of course, signatures are computed independently of indexing, so that once a signed document becomes indexed, the existing signature immediately gains currency. Moreover, the method is search engine independent, however, so that one can try several search engines and succeed if any of them has indexed the document in its new locations; meta-crawlers make this convenient.

In the case of documents accessible only via a script, there may also be a helpful search engine at the site. Finding the location of such a service and automatically exploiting it remains an interesting research challenge. As of 1998, "only 1% of the requests to a variety of servers were for CGI material" [2], though one would expect to increase as web sites grow ever more technically sophisticated.

Resources with highly variable content. Some resources, as for instance newspaper home pages, vary in content very quickly over time. In these cases, a straightforward signature will not be of much use. However, if the page is changing frequently, it is likely that it is a live page so that the original URL remains valid.

Non-textual resources. Non-textual resources such as images and video are not generally indexed based on their content. Two possible extensions are as follows: (i) If an image, et cetera, is embedded within an obvious textual context, the signature of that context could be used instead. (ii) Various attempts to analyze images by content are under way (for example, [18]) in ways that provide a set of terms that are rankable and indexable; should one of these become available as the basis for a global image search engine, then analogous image signatures could conceivably be created. At this stage, it is premature to access whether image signatures will be viable.

Spoofing. It is possible for someone wishing to compromise the utility of a signature to do so by creating other pages with the same signature. To do so trivially, one need only make the signature words the only words on the page. With a bit more effort, one can make pages that look less obviously contrived, but which have the same signature.

Nevertheless, we do not expect spoofing to be a significant problem. First, a signature is used to locate a page only when address-based referencing fails, so the opportunity to spoof a page is small. Spoofed pages are most likely to fool the user during the window of vulnerability after the original page moves but before it is reindexed. Thus, a potential spoofer must create a page with the right signature before the target page is moved in order that it is indexed with the target breaks. However, signatures can be tested at creation and periodically thereafter for false positives (which would necessarily include any spoofing pages), and more terms can be added to eliminate them, to a point.

Page has been deleted. If the page has been deleted from the face of the web, one might still use a signature to check if the page has been archived in a service such as Alexa [19] (which may be restrained by copyright) or temporarily stored in a search engine's cache. Of course, a web archive that permits access by URL would provide an alternative to robust hyperlinks. But note that such an alternative would not be a complete one. Archives are also subject to the challenge of an exponentially growing web, and hence are likely to be incomplete. Moreover, until multiple archive efforts exist, one cannot exploit them as one can multiple search engines to improve coverage. Additionally, a web archive cannot archive many documents behind a firewall that would be available through a cascade of search engines that include the local site, or protected by password that a publisher might allow to be indexed by a search engine but not be copied by a third party for uncertain benefit.

5 Extensions

We believe that the results presented above suggest that a useful level of performance is readily obtainable, and hence, that the approach is viable. There are any number of ways in which one might improve these results, both in the practice of computing signatures and by the action of robust-hyperlink-aware agents in dereferencing dangling links.

Signature Creation. Upon creation of the initial candidate signature, the signature creation agent can immediately perform a search to see how well it works in locating the reference. If, as in some of our examples above, the stringent search produces no results, one may try computing an alternate signature. (One possibility here is to choose among the top 2n significant terms to produce the best performing n term signature.) [16] take this approach, iternatively additing signature terms until the level of performance no longer improves.

The automatic selection of the rarest terms generates an effective, but at times unsatisfying signature. It is unsatisfying because it does not seem to characterize the meaning of the document in the same way that keywords do. Since it is possible to produce good signatures with more frequent terms, a user could be presented with good candidates terms and choose the most satisfying ones. (Our experiments indicate that any choice of words with document frequency of under 100,000 work very well.)

One interesting case is that of resources with highly variable contents. As mentioned above, a signature is likely to be less helpful in such cases (and less needed, as well). One possibility for such cases is to incrementally compute adaptive signatures, that is, signatures that are computed over time, so that they contain terms that persist over time.

Other Signature Types. The particular lexical signatures we suggest are just one form of lexical signature, which are in turn just one form of signature. We considered other, radically different, forms of signatures, but found them to be defective in one way or another. For example, using pure TF/IDF for lexical signatures is possible, but does not work as well as the ones we discuss, because traditional TF/IDF discounts the contribution of rarity too much. Checksums, or cryptographic signatures, would allow virtually guaranteed matches of pages as opposed to the probabalistic lexical signature; however, pages are not currently indexed by search engines, but would be brittle with respect to any modifications. Manually generated keyword sets have been attempted [20], but manually generated signatures are unlikely to be practical and scalable (the cited attempt gives evidence of their impracticality). It is possible to generate long random strings and include these in documents as identifiers, but then the burden of robustness would be on the document creator, not on the link generator, and therefore would require as Step One of implementation that everyone update the web billion pages. Nevertheless, we reserve the possibility that other, non-lexical forms of signatures could be devised that would be superior to lexical signatures, or support have other attractive properties.

Robust Hyperlink Agents. Aware agents might use signatures even when traditional referencing succeeds. For example, since computing a signature is relatively cheap, one's client might always compute the signature of a document it locates, and compare it to that in the hyperlink. If the signatures depart significantly, the user might be advised, and perhaps given the choice of performing signature-based dereferencing.

A more conservative strategy might be to perform signature checking only if there is some other indication that the document located may not be the same as that originally referenced. For example, suppose a document has several hyperlinks to sub-resources of a given document. Sub-resources might be named anchors in HTML, or XPointers [21], or robust location references, such as those used in Multivalent Documents [22], [23]. In each case, it is possible for the resource referenced to be found, but for the sub-resource not to be found. For example, most web clients, when given a URL of the form http://...#name, will simply ignore the absence of an anchor named "name". Failure to resolve sub-resource references might instead be interpreted as an indication to perform signature checking.

Additional Metadata. Time Stamp, Version, and so on. It might be useful for various reasons to include other metadata along with URLs. Examples include time stamps, keywords, document version ids, and many others. Developing a standard syntax for the general encoding of metadata in URLs seems advisable.

Other Applications. Simple signatures along the lines of the ones we suggest for robust hyperlinks may have other applications as well. In particular, we speculate that they may have some utility for detecting duplicate or plagiarized documents. Investigating such applications is a topic for future research.

6 Related Work

The primary difference between robust hyperlinks and most other approaches is essentially a practical one. Namely, one does not require administrative buy-in, the creation of infrastructure, or agreement on conventions for robust hyperlinks to work. In addition, the storage, computational and communication requirements are modest. As compared to "global database" solutions, robust hyperlinks gives up guaranteed results that require careful and continuous maintenance by authorized site webmasters, in exchange for usually excellent results available at minimal effort to anyone interested. Thus, crucially for the Web, robust hyperlinks scales.

The idea that small pieces of content can be used as a content-based address for a document on the web is found in [16]. Their signatures do not take terms' web frequency into account, nor are they designed to be robust in the presence of document and web change, or work well with multiple search engines. Our lexical signatures are defined to be more similar to those of [14], although these also pay only minimal attention to robustness. In addition, we provide a means to incorporate such signatures into the web infrastructure.

This paper concerns references between documents, but robustness it is also important for references within a document. Such intra-document locations are used by external hyperlinks, annotations, transclusions, and other capabilities not yet found in mainstream web software. We address the issue of robust intra-document locations elsewhere ([24]). The two types of robustness share similar goals of gracefully degrading performance and so on, but the realizations share little in common, other than that both increase robustness with multiple, independent means of resolving a given reference.

7 Implementation

We have implemented, in Java, a web site crawler [25] that rewrites URL references in HTML pages (that is, A elements' HREF attribute values, and IMG elements' SRC attribute values) to make them Robust URLs. (The URLs in the paper were signed automatically with this software.) It employs a word frequency cache based on a list of the 50,000 most common words on the Internet, courtesy of Inktomi, which has been filtered to the 20,000 words usable as signature words (words with four more letters, without numbers, with case differences folded together) and looked up in AltaVista for frequency counts. The software is Open Source, and therefore amenable for incorporation into other web software, such as page authoring and site management tools and web browsers.

Furthermore, we have integrated support for robust URLs in our own web browser, the Multivalent browser [22], [23].

Finally, we have implemented a simple "scriptlet" that can be kept as a bookmark in Netscape, Opera, and Microsoft's Internet Explorer, so that upon encountering a 404 on a robust URL, the user can invoke the bookmark to perform signature-based resolution.

8 Conclusion

We believe our initial findings indicate that robust hyperlinks are a viable solution to a large part of the problem of broken links, scalable to the size of the World Wide Web. Namely, we can, immediately, at small cost, and fully automatically, make links that will enable us to find textual documents with highly probability when the resource has been both moved and modified.

The approach embodied in robust hyperlinks is an example bootstrapping new capabilities upon previously developed web infrastructure, namely search engines. Perhaps many other new capabilities can stand on the shoulders of what's come before.

Acknowledgements. This research was supported by Digital Libraries Initiative, under grant NSF CA98-17353.

References

(In the on-line version of this paper, the URLs in the references below are robust hyperlinks.)

1. Georgia Tech Graphics, Visualization & Usability (GVU) Center, Tenth WWW User Survey, Question 11, "Problems Using the Web", 1998. (http://www.cc.gatech.edu/ gvu/user_surveys/survey-1998-10/graphs/use/q11.htm)
2. World Wide Web Consortium, Jim Pitkow, Chair. Web Characterization Activity Answers to the W3C HTTP-NGs Protocol Design Group's Questions. 1998. (http://www.w3.org/ WCA/Reports/1998-01-PDG-answers.htm)
3. K. Sollins and L. Masinter. Functional Requirements for Uniform Resource Names, Network Working Group Request for Comments 1737, December 1994.
4. Robert Kahn and Robert Wilensky. A Framework for Distributed Digital Object Services. cnri.dlib/tn95-01, 13 May 1995.
5. OCLC PURL Service. (http://www.purl.org/)
6. Common Name Resolution Protocol, 18 October 1999. (http://www.ietf.org/ html.charters/cnrp-charter.html)

7. David Ingham, Steve Caughey, Mark Little. Fixing the "Broken-Link" Problem: The W3Objects Approach. Computing Networks & ISDN Systems, Vol. 28, No. 7–11, pp. 1255–1268: Proceedings of the Fifth International World Wide Web Conference, Paris, France, 6–10 May 1996.
8. Mind-it. (http://www.netmind.com/html/individual.html)
9. Sofus Macskassy and Leon Shklar. Maintaining information resources. Proceedings of the Third International Workshop on Next Generation Information Technologies (NGITS'97), 30 June–3 July 1997, Neve Ilan, Israel.
10. Paul Francis, Takashi Kambayashi, Shin-ya Sato and Susumu Shimizu. Ingrid: A Self-Configuring Information Navigation Infrastructure. 11–14 December 1995. (http://www.ingrid.org/francis/www4/Overview.html)
11. Michael L. Creech. Author-oriented Link Management. Proceedings of the Fifth International World Wide Web Conference, Paris, France, 6–10 May 1996.
12. LinkGuard. (http://www.linkguard.com)
13. Hugh C. Davis. Referential Integrity of Links in Open Hypermedia Systems, Proceedings of Hypertext 98, Pittsburgh, Pennsylvania, 20–24 June 1998.
14. Krishna Bharat and Andrei Broder, A Technique for Measuring the Relative Size and Overlap of Public Web Search Engines. Proceedings of the Seventh World Wide Web Conference, Brisbane, Austrailia, 14–18 April 1998. (http://www-sor.inria.fr/mirrors/www7/programme/fullpapers/1937/com1937.htm)
15. Inktomi. Web Surpasses One Billion Documents. Press Release, 18 January 2000. (http://www.inktomi.com/new/press/billion.html)
16. Joel D. Martin and Robert Holte. Searching for Content-based Addresses on the World-Wide Web. Proceedings of Digital Libraries '98. (http://ai.iit.nrc.ca/II_public/DL98paper.ps)
17. Netcraft Web Server Survey. December 1999 (http://www.netcraft.com/survey/).
18. Chad Carson, Serge Belongie, Hayit Greenspan, and Jitendra Malik. Blobworld: Image segmentation using Expectation-Maximization and its application to image querying, 4 February 1999. (http://elib.cs.berkeley.edu/~carson/papers/pami.html)
19. Alexa. (http://www.alexa.com/)
20. Jorn Barger, A New Strategy for Document Indexing? USENET posting, 15 February 1999. (http://www.deja.com/=dnc/getdoc.xp?AN=444610476)
21. XML Pointer Language (XPointer). W3C Working Draft 9 July 1999. (http://www.w3.org/ 1999/07/WD-xptr-19990709)
22. Thomas A. Phelps. Multivalent Documents: Anytime, Anywhere, Any Type, Every Way User-Improvable Digital Documents and Systems. Ph.D. Dissertation, University of California, Berkeley. UC Berkeley Division of Computer Science Technical Report No. UCB/CSD-98-1026, December 1998. Also see the general and technical home pages.
23. Robert Wilensky and Thomas A. Phelps. Multivalent Documents: A New Model for Digital Documents. UC Berkeley Division of Computer Science Technical Report, CSD-98-999, 13 March 1998.
24. Thomas A. Phelps and Robert Wilensky. "Robust Intra-document Locations", Proceedings of the Ninth World Wide Web Conference. 15–18 May 2000, Amsterdam.
25. Robust Web Site. http://www.cs.berkeley.edu/~phelps/Robust/

Inferring Structure Information from Typography

Christian Fuß, Felix Gatzemeier, Michael Kirchhof, and Oliver Meyer[⋆]

RWTH Aachen
Lehrstuhl für Informatik III
Ahornstraße 55
52074 Aachen, Germany
phone: +49 (2 41) 4 40 - 2 13 13
{cfuss, fxg, kirchhof, omeyer}@i3.Informatik.RWTH-Aachen.de

Abstract. One of the key benefits of digital documents is the possibility to add further structure. With current authoring tools, however, this imposes additional effort on the author. We propose a technique which uses everyday editing actions to infer structure. For this, we rely on the widespread use of typographical markup by authors.

We state the assumptions underlying its application, design, and possibilities. In concluding remarks, we draw connections to further work which will provide actual benefits to authors based on the inferred structure.

1 Introduction

Digital documents are frequently preferred over analog ones for their ease of transmission and processing. Automated processing depends on explicit structural information. Publishers use markup languages internally for workflow management and cross-platform publishing. Submissions from the authors, however, undergo costly conversion into a marked-up format after being accepted for publication.

With the advent of the more cheaply processable XML, it is now realistic to implement tools that may be distributed to a wide audience of authors to help them submit marked-up documents to the publisher. Several attempts have been made in this direction (see section 2), but have generally failed to reach wider acceptance, as they forced authors to change, however minimally, their accustomed working styles or tools.

This text presents a technique which may be used to reasonably infer XML structure from the text as it is written by the author the way he is used to. As structure is derived from typographical information, only simple structures can be inferred. The results are used for the development of the authoring tool aTool[1]. It is the core of a cooperation with Springer-Verlag and Technische Universität München in the context of the Global-Info project funded by the German Ministry of Education, Research and Technology

⋆ The work presented here has been supported by the German Ministry for education, research and technology as part of the Global-Info project (http://www.global-info.org).

[1] http://www11.in.tum.de/aTool/

(BMBF)[2]. At the current stage, the project is focused on scientific authors writing article-sized documents.

This paper starts out with the discussion of related work in the area of structured document generation. The body of this paper begins by stating the basic assumptions we make for the application of this technique. On this basis, the technique itself is presented. The application context of its implementation is discussed towards the end.

2 Related Work

Today's approaches to XML authoring can be classified into two categories: XML editors and batch converters for proprietary documents. Historically, syntax-directed editors are precursors of the first type of applications.

Syntax-Directed Editing

A paradigmatic syntax-directed editing system is the family of synthesizer editors and the Synthesizer Generator [13] that generates editors from descriptions of document syntax and editing commands. This structure is quite similar to validating XML editors, which are parameterized at runtime with a DTD. Pure syntax-directed editing has proven to be too restrictive for authors, so there must be at least a free editing mode, in which the document or parts of it can be manipulated at will. Such systems, however, need to rely on their own visualization and interaction mechanisms and are therefore no option for integration with existing word processors.

As the discipline of psychology has established quite rigid authoring guidelines, there have been attempts to develop authoring applications that ensure the correctness of documents according to the formal parts of these guidelines. Manuscript Manager [11] is a commercial product based on this idea. It did not reach widespread acceptance, presumably due to its nature of being another stand-alone word processor. Its current successor, the APA-Style helper [1], takes a different approach: Dialog boxes guide the author through entering the required information and references, from which a RTF document framework is generated. This can be of help to novice authors, but its restriction of very tight guidance and completely free-form editing gives no support in the core stages of document creation.

XML Editors

The purpose of XML editors is to create correct XML documents. The generally possible complexity of these documents and the lack of abstraction from XML concepts demand a rather complex user interface. Hence reviews of most XML editors (like XML Pro, XMetaL, etc.) stress that today's applications are not made for novices, but rather assume a good working knowledge of XML and a general understanding of the nomenclature.

[2] http://www.global-info.org/

Some applications make use of widely used word processing applications as the UI to edit XML documents (for example i4i's S4/Text [8]). The content author is still forced to declare the structure of the document explicitly and in a detailed way, by specifying the type of every inserted element which interrupts the author while writing.

Batch Conversion Applications

Common batch conversion applications fail on the task of inferring deep XML structure from the input documents (mainly MS Word or RTF documents). Only some sophisticated tools, like Schema's MarkupKit [7] build up more than a two-layer hierarchy.

Most of them map character formatting like bold, italic etc. to generic element types expressing the same *presentation*, disregarding structure expressed by the typography. Electronic Book Technologies (EBT)[3] have developed a suite of tools based on this idea. The Rainbow DTD describes this information and the Rainbow Maker converts documents of some formats, for example RTF to instances of this DTD. As a second step, higher-level structure can be added with DynaTag. These applications offer the user an automatic mode to extract structure. Still, most of these algorithms rely heavily on named styles and format templates rather than inferring structure from typography.

As batch mechanisms, the systems rely on appropriate configuration and entail relatively high costs for user intervention, namely offline fine-tuning. None of these methods can build logical structure as a true hierarchy by just evaluating typography without burdening the content author with explicitly specifying structure.

The CIDRE project [3] at the university of Fribourg pursues a probabilistic approach adopted from optical document recognition. The n-grams of sequential texts are generalized to include hierarchy information. Probabilities for such generalized n-grams can be inferred from existing document bases. This approach overcomes the disadvantageous extensive manual configuration and flat hierarchy of the other batch conversion mechanisms, but is not integrated into authoring systems.

3 Basic Assumptions

As only a human being can understand a text enough to introduce semantical markup in the general case, there must be some assumptions we make in comparison to that. We claim that these assumptions do not severely limit generality in our application domain of scientific publishing.

Assumption 1 (Known target document type) *There is a description of the expected structure, that is: a DTD for the XML document.*

[3] Now Electronic Business Technologies, www.ebt.com.

As we are discussing documents that are to be fed into a well-defined processing pipeline, it is safe to assume that the interchange format is well-defined on the syntactical level. The technique relies on the DTD as a source for informative element type names and, in a validating mode, to limit the number of possible types.

Assumption 2 (Textual nature) *The author produces textual documents, not general data.*

While XML can be used and is used for general data modeling (as in RDF [9]), we use it in the application domain of natural-language documents. Moreover, these documents still resemble the linear documents found in printed books and journals.

Using this assumption, we distinguish element types into those corresponding to paragraphs and those corresponding to sub-paragraph constructs.

Assumption 3 (Typography) *The author applies some typographic markup.*

This is based on the experience that authors frequently do so. They often use direct formatting instead of named styles as the applied changes are more obvious. Headlines, emphasis and other categories are usually typographically marked, but not necessarily as in the final publication. We use these marks to determine element boundaries and types. We do not require usage of named styles, but can incorporate them where present.

Authors that do not use typography may still use typewriter-style markup such as whitespace. While we do not address such markup here, we do feel confident that an extension of the typography determination routines will suffice to support these authors, too.

Assumption 4 (Correspondence) *The typographic markup corresponds with the semantical structure.*

That is, parts of the text that look different are probably used in different roles. Often the author is not aware of these roles and therefore cannot be convinced to use named styles. This assumption does not determine the degree of difference required to be classified differently. For example, an author may tweak font sizes to fit that extra paragraph on a page in his draft printouts without being required to associate some different structure to it.

Parts of the text that look the same are, special situations exempt, structurally equivalent. A sequential content model for example may lead to equally formatted paragraphs being interpreted differently.

4 Deriving Structure

In our approach, we extend the user's authoring application to maintain data structures *incrementally*, that is, while he is editing and formatting his text. These data structures can then be used to extract a structured document.

Inferring structure information from typography involves two subproblems: locating the structural elements and assigning types to these elements. Both subproblems also involve hierarchical ordering of the elements.

4.1 Locating the Elements

To solve the first problem, the flat character stream is partitioned into syntactical fragments on typographic and structural boundaries. According to assumption 4, these boundaries imply semantical boundaries. From these fragments, a hierarchy of typographic elements is constructed. The top-level elements represent the paragraphs, while nested elements correspond to typographical markup within the paragraphs (see assumption 2). Inferring the hierarchy of the paragraphs (for example chapters or sections) depends on the underlying DTD and is beyond the scope of this paper.

To determine the typographic boundaries, the style of each character is expressed as a tuple covering a selection of the typographic properties, for example a style name, font name, size, weight, and variation. An example for a format tuple of a typical heading character is

$$FT = (\text{‘Heading’}, \text{Helvetica}, \text{16pt}, \text{bold}, \text{roman})$$

Figure 1 shows the pseudocode for locating the syntactical fragments. It expects an input stream of formatted characters and produces an XML stream with mixed-content elements, whose types are discussed in the next subsection. This batch algorithm can be applied to single paragraphs for incremental operation, as the algorithm returns to its initial state at the end of each paragraph. For the sake of simplicity, paragraph breaks are determined by a special character (CR) and we expect a CR as the last character in the document.

The algorithm scans the input stream for changes in format tuples and opens or closes elements accordingly. A stack of format tuples (oStack) corresponding to the currently open elements is maintained to determine whether to open or close elements. Every paragraph break causes all open elements to be closed and the stack to be cleared.

A change of the format tuple (forTup) may indicate the end of one or more open elements or a new nested element. If the format tuple has been seen before, it is assumed that the author wants to switch back to the corresponding level. This means that the head of the stack is popped and the current element is closed until the format tuple on the stack equals the new format tuple. If the format tuple has no corresponding entry in the stack, a new element is opened and its format tuple is pushed onto the stack.

The algorithm is presented here as working on streams for reasons of simplicity. When creating a DOM-like document tree, some more anchors in the input file have to be remembered on a stack of open elements.

```
oStack.Empty
WHILE Stream.HasChars
    curChar := Stream.GetNextChar
    IF curChar == CR THEN            -- close all elements
        WHILE oStack.NotEmpty DO
            XML.CloseElement(oStack.Head)
            oStack.Pop
        ENDWHILE
    ELSE
        forTup := FormatTuple(curChar)
        IF forTup in oStack THEN     -- close contained elements
            WHILE forTup != oStack.Head DO
                XML.CloseElement(oStack.Head)
                oStack.Pop
            ENDWHILE
        ELSE                         -- open new element
            oStack.Push(forTup)
            XML.OpenElement(forTup)
        ENDIF
    ENDIF
ENDWHILE
```

Fig. 1. Pseudocode for identifying text fragments.

As inferring types from the DTD is deferred to another part of the overall algorithm, we create only preliminary dummy types reflecting only the different format tuples. The type information is stored in a table called FORMAT-TO-TYPE-MAP. Figure 2 shows an example.

Format Tuple	Element Type
('Heading', Helvetica, 16pt, bold, roman)	`<Dummy_Heading16ptBold>`
('Standard', Times, 10pt, regular, roman)	`<Dummy_Standard>`
('Standard', Times, 10pt, regular, italic)	`<Dummy_StandardItalic>`

Fig. 2. An example of a FORMAT-TO-TYPE-MAP

In building this map, the algorithm introduces *types* of elements, therefore abstracting from instances. Two elements in the text with the same format tuple are of the same type and therefore get the same temporary type name (see assumption 4).

4.2 Generalization of Format Tuples

As not all elements of the tuples are relevant for determining the type, wildcards ($*$) may be used in the table to allow arbitrary matches on single fields.

When a new dummy type is created, a minimal *format pattern* for the map entry is generated by comparing the format of the dummy element (fmt) with that of the immediately preceding (pre) and following text (suc). For a single property value of these formats, the operator Δ is defined as follows:

$$\Delta(pre, fmt, suc) ::= \begin{cases} fmt, \text{ iff } pre \neq fmt \vee suc \neq fmt \\ * \quad , \text{ iff } pre = fmt = suc \end{cases}$$

If the text is at the start or end of a paragraph the preceding or following text is not considered:

$$\Delta(pre, fmt, \bot) ::= \Delta(pre, fmt, fmt)$$
$$\Delta(\bot, fmt, suc) ::= \Delta(fmt, fmt, suc)$$

As we regard multiple format properties we extend the Δ operator to handle tuples of properties $\overline{fmt} ::= (fmt_1, \dots, fmt_n)$:

$$\overline{\Delta}(\overline{pre}, \overline{fmt}, \overline{suc}) ::= (\Delta(pre_1, fmt_1, suc_1), \dots, \Delta(pre_n, fmt_n, suc_n))$$

Figure 3 shows two examples. When calculating the pattern for "this", its format ($\overline{t_2}$) is compared to $\overline{t_1}$ and $\overline{t_3}$. As $\overline{t_1}$ and $\overline{t_3}$ describe the same format, we get $\overline{\Delta}$(('Standard', Times, 11pt, regular, roman), ('Standard', Times, 11pt, bold, roman), ('Standard', Times, 11pt, regular, roman)) = ($*$, $*$, $*$, bold, $*$). The pattern for "that" is more interesting. Style, font name and size are equal for $\overline{s_1}$ and $\overline{s_3}$, while the variation differs. Our generalization operator yields $\overline{\Delta}(\overline{s_1}, \overline{s_2}, \overline{s_3}) = ($'$*$', $*$, $*$, bold, roman$)$. This captures all differences of $\overline{s_2}$ to surrounding texts.

$$\underbrace{\text{Is}}_{\overline{t_1}}\ \underbrace{\textbf{this}}_{\overline{t_2}}\ \underbrace{\text{a dagger?}}_{\overline{t_3}} \qquad \underbrace{\text{Is}}_{\overline{s_1}}\ \underbrace{\textbf{that}}_{\overline{s_2}}\ \underbrace{\textit{a dagger?}}_{\overline{s_3}}$$

Fig. 3. Calculating the minimal pattern.

4.3 Inferring Types of Elements

The next task is to determine real element types, that is, element types that are defined in the DTD. This is carried out by the so-called *inference function* which computes DTD element types for given format tuples. In order not to make too many assumptions about the structure of documents in general and naming conventions, the element types are derived only from the DTD attached to this document (see assumption 1). There can

Format Tuple	Element Type
('Heading', $*$, 16pt, $*$, $*$)	`<Title>`
('Standard', $*$, $*$, $*$, $*$)	`<Paragraph>`
('Standard', $*$, $*$, $*$, $*$)	`<Author>`
('Standard', $*$, $*$, $*$, italic)	`<InlineEmphasize>`

Fig. 4. An example for a FORMAT-TO-TYPE-MAP with element types derived from the DTD.

be no automatic mapping from dummy types to DTD-defined types. So, somebody (the author or the publisher, for example) has to indicate which dummy types represent which element types in the DTD resulting in a map like that in figure 4.

Note that the format pattern ('Standard', $*, *, *, *$) occurs twice in the map, so `<Paragraph>` and `<Author>` are equally possible. In general, one format tuple may be mapped to multiple element types and one element type may have multiple matching format tuples.

This results in the following characterization of the inference function: The function computes a list of element types based on the text fragment, its context, and its format tuple. This *match list* is ordered by match confidence, which is in a simple form determined by the number of matching properties. A more sophisticated variant could take into account that certain properties (for example the style name) are more important than others.

The author may want to ensure that his document is valid (see [2]) with respect to a given DTD, in which case a validating mode is entered. In this mode, the inference function queries the DTD to determine which element types are allowed in the context of the text fragment. If the intersection of the match list and the set of allowed element types is not empty, the match list is restricted to this intersection. If it is empty, nothing is changed.

The element type with the highest rank in the match list is the result of the inference function and assigned to the text fragment. If several entries have an equal rank, the author has to decide which type should be assigned.

5 Application Context

We are using the technique described above in the implementation of the authoring tool aTool as an extension of MS Word since it is the word processor most authors use for non-mathematical text, according to the Springer-Verlag's experience. The implementation adopts the principles of MS Word and hides XML jargon wherever possible.

To apply their journal-specific guidelines, our project partner Springer-Verlag aims at receiving XML documents from authors. The XML documents must at least conform to a DTD provided by the publisher. In this context assumption 1 "Known target document type" and assumption 2 "Textual nature" are true. We are additionally looking at author guidelines which can not be expressed with a DTD, but can be formalized, for example length constraints. These formalized guidelines beyond the expressive power of a DTD are called *additional constraints*.

The authoring tool draws its parameter settings from the so called *aToolKit*, which consists of the DTD, the additional constraints, an initial FORMAT-TO-TYPE-MAP and other data required for presentation and formatting features. The publisher is responsible for an appropriate configuration shipped in the aToolKit.

The initial map is expected to map MS Word's named styles directly to element types of the DTD. In order to relieve the author from the burden to associate dummy types with DTD definitions the map also contains heuristic entries like ('$*$', $*$, 16pt, bold, $*$) $\mapsto$ `<title>` that map directly formatted text onto structural entities.

The aTool application keeps the additional structural information in an attributed tree which correlates to the XML structure and accompanies every MS Word document processed by aTool. The position and bounds of the elements are maintained by references in the word document model [4] which are stored using bookmarks (figure 5). The attributed tree is stored in a DOM-conforming way, using a third-party XML parser.

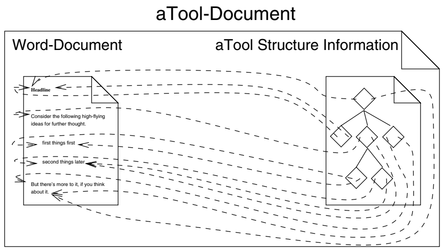

aTool-Document
Word-Document
aTool Structure Information
Headline
Consider the following high-flying ideas for further thought.
first things first
second things later
But there's more to it, if you think about it.

Fig. 5. Internal structure of an enriched document

The main goal of the implementation is to remain as unobtrusive as possible. We therefore avoid additional user interface elements and interruptions to the usual workflow wherever possible. The tool is supposed to work incrementally; the aTool structure is updated whenever the author changes the format of a text portion, creates a new paragraph or edits the text.

It is up to the author when to change a dummy type to a type given by the DTD. We suggest to use the find and replace metaphor for this purpose. For this purpose the *mapping dialog* is opened to display the format tuples and the possible matches. The author can find elements of a specific type and change their type individually or collectively. In the first case the same format pattern is associated with different types. In the latter case, the entry in the FORMAT-TO-TYPE-MAP is changed to map the pattern to the type indicated by the author. Any formatting matching this pattern after that change will create an element typed conforming to the DTD.

The Format-to-Type-Map in the aToolKit is modified to reflect user decisions in the mapping dialog. All elements of the aToolKit are stored in XML files to allow modification by an advanced user with any editing tool of his choice.

Querying the Author

User interaction is required to resolve ambiguous type matches from the Format-to-Type-Map. We delay the query through the concept of warning markers that has proven to be very useful in our PROGRES development environment [10]. A simpler form of this concept is also used in continuous spell checking or the PowerPoint 2000 light bulb tips.

Instead of an immediate query, we mark the element internally. The author does not necessarily see that mark, he can just type ahead. When he wishes to view the marks, these elements are highlighted, for example shown in a special color or underlined. The errors and warnings are also visualized in additional views provided by aTool.

The author can then select a mark in any view and correct the document to match the DTD. For ambiguous element types he is presented the list of possible types produced by the inference function. The tool supports the author with a function to navigate through the marks. This provides him with a systematic way to handle all warnings in the document.

6 Conclusion and Plans

The technique described in section 4 achieves the goals outlined in the introduction: based on a consistent application of typography and a description of the expected structure, an inference function based on a Format-to-Type-Map can be constructed that returns appropriate element types given a specific format. By providing a specific Format-to-Type-Map for user groups, the correctness of structure inference can be improved, which will also increase acceptance of aTool among authors.

As the inference function maps from format tuples, not only from style names, we do not require the usage of styles, which has proven to be a noticeable barrier for many authors.

The publisher can fit the description of the structure to his needs and therefore is in control of the resulting XML document. The elements typically required are bibliographic information, hierarchical structure, and literature references. All these element types can be determined using the technique described here.

6.1 Plans

While the inferred structure is valuable for the publisher, the gain for the author is still marginal. He has some more structural views on the document, with attached navigation and editing facilities on structural elements. The structure information may also be used to easily derive variants of the document formatted in various output forms of the further publishing chain.

If speed of processing is important, the author may value *extended validation* functionality that validates the XML structure against the DTD and checks additional constraints implied by author guidelines. Error reports could be propagated into the document by using markers as described in section 5.

The algorithm described in section 4 focuses on elements below the paragraph level. A natural extension would use paragraph format properties to derive XML elements for paragraphs. This would extend the algorithm from inline to block elements as they are distinguished by other tools for structured documents like HTML-Tidy [12]. The preceding and subsequent paragraphs can then be used to trim the match list. If the element may only appear at this position nested into other ones, these may be generated automatically. This approach may be generalized based on the results of the CIDRE project [3], attaching probabilities to optional elements.

Some semantical structure is also the prerequisite of taking word processors to the level of text processors or authoring applications. This would take up research in the line of the Writing Environment [14] up to SEPIA [15]. Simple measures would be instantiation and maintenance of patterns, visualization of semantical structure and dependencies [5], complex derivation of audience-specific documents and alternative access routes [6].

6.2 Thanks

We would like to thank Ms. Prof. Dr. Brüggemann-Klein and Ms. Gross from Technische Universität München and Ms. Hepper, Mr. Dr. Wilson, and Mr. Schreiber from Springer-Verlag for the helpful, interesting and sometimes lively discussions about these topics.

References

1. American Psychological Association. *APA-Style Helper 2.0*, 11 2000. `http://www.apa.org/apa-style/`.
2. Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, editors. *Extensible Markup Language (XML) 1.0.* W3C, February 1998.
3. Rolf Brugger, Frédéric Bapst, and Rolf Ingold. A DTD Extension for Document Structure Recognition. In Roger D. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography*, volume 1375 of *Lecture Notes in Computer Science*, page 343ff. Springer, 1998.
4. Microsoft Corporation. *Microsoft Office 2000.* Microsoft Press, 1999.
5. Felix Gatzemeier. Patterns, Schemata, and Types — Author Support Through Formalized Experience. In B. Ganter and G. Mineau, editors, *Proc. International Conference on Conceptual Structures 2000*, volume 1867 of *LNAI*. Springer, 2000.
6. Felix Gatzemeier and Oliver Meyer. Improving the Publication Chain through High-Level Authoring Support. In Manfred Nagl, Andy Schürr, and Manfred Münch, editors, *Applications of Graph Transformation with Industrial Relevance (AGTIVE)*, volume 1779 of *LNCS*, Heidelberg, 2000. Springer.

7. Schema GmbH. Word as HTML/XML/SGML-editor with the MarkupKit. `http://www.schema.de/sitehtml/site-e/wordasht.htm`.
8. i4i.S4 Text.Infrastructures for Information Inc. `http://www.i4i.com/tech_s4text.htm`.
9. Ora Lassila and Ralph R. Swick, editors. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C, Feb 1999. `http://www.w3.org/TR/REC-rdf-syntax`.
10. Manfred Nagl, editor. *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, volume 1170 of *LNCS*. Springer, Heidelberg, 1996.
11. Pergamon Press, Elmsford, NY. *Manuscript Manager (APA Style) user guide*, 1988.
12. Dave Raggett. HTML Tidy. `http://www.w3.org/People/Raggett/tidy`.
13. Thomas W. Reps and Tim Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*. Springer-Verlag, 1989.
14. J. B. Smith and M. Lansman. A Cognitive Basis for a Computer Writing Environment. In B. K. Britton and S. M. Glynn, editors, *Computer Writing Aids: Theory, Research, & Practice*, pages 17–56. Erlbaum, Hillsdale, NJ, 1989.
15. Norbert A. Streitz, Jörg M. Haake, Jörg Hannemann, Andreas Lemke, Wolfgang Schuler, Helge Schütt, and Manfred Thüring. SEPIA – A Cooperative Hypermedia Authoring System. In D. Lucarella, J. Nanard, M. Nanard, and P. Paolini, editors, *Proceedings of ECHT'92, the Fourth ACM Conference on Hypertext*, pages 11–22, Milano, Italy, 1992. ACM.

Multimedia Authoring with MAVA

Jürgen Hauser

University of Stuttgart
Institute of Parallel and Distributed High-Performance Systems (IPVR)
Breitwiesenstr. 20–22
70565 Stuttgart, Germany
hauser@informatik.uni-stuttgart.de

Abstract. Multimedia authoring is still subject to computer experts with programming knowledge. Most multimedia systems provide fixed concepts for the specification of the temporal and spatial layout of multimedia documents. Starting with interaction some approaches require script programming by authors. Special effects like animations, complex interactions or alternative concepts have to be programmed in almost every multimedia system. If a multimedia system does not provide a scripting language the realization is impossible. As different application areas for multimedia documents are considered (e.g. computer based training) authors need authoring languages with a high abstraction level that support the specification of documents in these areas. Because application areas that have to be used for authoring cannot be predicted in advance an extensible document language is required. For the presentation and authoring of multimedia documents this means that an extensible multimedia presentation system and an extensible document editor are required. The paper presents research results achieved in a project that aims at the simplification of authoring and dissemination of multimedia documents. The approach is based on an extensible document language. Separation between authoring and development of language extensions requires only scanty programming knowledge by authors. Extensions have to be developed by programmers and will be disseminated together with documents. With a growing number of available extensions a decrease of programming effort can be expected because of reuse of extensions.

1 Introduction

The following scenario has to be realized as a multimedia document. An author has to create a computer based training for chemists. The training will consist of two parts. Part one contains information on a specific subject of chemistry and the second part is a test to asses what the user has learned from the first part. In both parts a visualization of molecules is used. The visualization provides additional interaction possibilities to control the view of a molecule because of the three dimensional nature of molecules. Both parts of the document need different concepts for their specification. The first part requires concepts for an interactive presentation while the second part requires concepts for a computer based training. A multimedia system that allows the specification of this sample document permits the usage of different concepts and arbitrary media items.

After authoring the document will be made available to different users. Therefore the document is stored on a server (e.g. a Web server). Now, a user can retrieve the document from that server. To present the document the presentation system has to know how to realize the semantics of the used concepts. Suppose that the presentation system knows how to realize the concept for the interactive part but not for the computer based training. To solve this problem it is useful to separate the document description language and the realization of its semantics. Hence, the server will also provide the realization of the semantics of the computer based training concept as extension for the presentation system. The presentation system will load all required extensions besides the document. After loading and integrating the extensions the presentation system is capable of presenting the document. The same approach is used if the presentation system does not know how to display a molecule. Because the presentation system will automatically detect missing extensions the approach does not require manual configuration of the presentation system by the user. This process is called ad hoc extensibility.

The paper presents results achieved during conception and implementation of an extensible multimedia system which is capable of handling the described scenario. The developed multimedia system is called MAVA (*M*ultimedi*A* document *V*ersatile *A*rchitecture). The research project is funded by the German Research Foundation (DFG) in the research initiative “Distributed Processing and Exchange of Digital Documents (V3D2)” [11].

The paper is organized as follows. Section 2 discusses the scenario of the introduction in more detail. In section 3 an extensible document model is presented. Section 4 shows the architecture of the presentation system implementation. Section 5 explains the idea for the dissemination of MAVA documents. In Section 6 aspects of a MAVA document editor are discussed. The Internet integration is discussed in section 7. Section 8 presents related work. Finally, section 9 contains a conclusion.

2 Multimedia Authoring

Multimedia authoring is still subject to computer experts with programming knowledge. This is caused by the broad usage of script languages to adapt the behavior of a multimedia system to meet requirements of particular documents.

Multimedia systems provide fixed concepts for the specification of temporal and spatial layout of documents. This means that an author has to use these concepts. Additionally, most multimedia system provide a concept for the specification of interaction, e.g. TIEMPO [12], but sometimes a scripting language has to be used to describe the interaction semantics [3]. Hence, knowledge of computer programming principles is required from authors.

2.1 Discussion of the Scenario

For the realization of the example given in the introduction the evaluation of existing systems has shown that each system provides different concepts for the specification of the first part (interactive presentation). These systems provide different concepts for the temporal layout (e.g. a time line, interval operators) whereby each system provides one fixed concept. In practice these concepts are more or less equivalent. In theory, however, the expressiveness of these concepts is different [10]. Also these systems provide different concepts for the spatial layout of a document. Some of them use a kind of absolute positioning in a presentation area while others use relative positioning.

The scenario of the introduction section can be used to show advantages of an extensible multimedia system. The second part of the document (the test) uses a computer based training concept (i.e. branching according to correct or false answers). It can be hardly realized with only temporal and spatial concepts. Hence, the example document cannot be realized with systems providing only these concepts. A system that allows the extension of the authoring language with new concepts is required. With an extensible system it is possible to add language elements representing a computer based training concept. An author will use these language elements the same way she uses the language elements of other concepts.

An extensible multimedia system can also provide alternative concepts letting the author choose the concept which best matches the authoring task. Consider the following two cases. In the first case the author wants to place buttons on particular positions on a map. Therefore she needs the concept for absolute positioning because the position of each button cannot be computed automatically. This means that relative positioning is insufficient in this case. Accordingly, it is possible to mention an example where absolute positioning causes a lot of overhead. An author wants to specify the layout of a document where a text has to be placed below an image. If the image size changes during authoring (e.g. the image was replaced by another one) the positions of related media items can be computed automatically due to relative positioning. If absolute positioning has to be used new media positions have to be computed by hand. This example shows that an extensible multimedia system can be used to provide alternative concepts and therefore simplify authoring.

The analysis of the example document has shown that an extensible document language can simplify multimedia authoring. From the author's point of view it is sufficient to consider just the document language. But another aspect that has to be considered is how semantics of language elements are realized. This can be done by a separation of the usage of language elements and their realization. This means that there is a software component, called manager, that will realize the semantics of a particular set of language elements. The encapsulation in a manager and the separation from the document language has the advantage that it can be simply reused by other authors. In contrast to that script programs are contained in the document and a particular solution for a particular document. Hence, reuse of scripts is not as simple as one may think. A more detailed discussion of script languages can be found in section 8 about related work.

2.2 Research Goals for a New Multimedia Presentation System

The analysis of the scenarios of the introduction and further scenarios has led to the following research goals for the conception and realization of an extensible multimedia system.

- *Extensibility*
  A multimedia system should not be limited to particular concepts or media items. The application areas for multimedia documents will become more and more specific (e.g. media items for the visualization of molecules, protocols or algorithms will provide advantages in teaching). Hence, different and alternative concepts and media items have to be integrated into an existing multimedia system. Extensibility covers the specification language, the presentation system to realize the semantics and a document editor to specify documents. The specification language and the presentation system extension are separated. This allows to reuse extensions and optimizations like caching of extensions.
- *Ad hoc extensibility*
  A platform independent realization allows the presentation of documents on a wide range of platforms. The presentation system automatically loads required program code for the presentation of a particular document (ad hoc extensibility). Thus users are relieved from extensive configuration of their presentation system.
- *Simplified multimedia authoring*
  One of the major goals of the MAVA approach is to simplify the authoring process for multimedia documents. Multimedia authoring has to be feasible for authors that are not familiar with programming principles. Ideally, a graphical document language will be used for authoring. The specification language is a high-level language based on a meta language that is independent of the concept it is used to describe. The separation between document authoring and developing extensions for a multimedia system means simplified authoring because an author does not need to know about programming language extensions. Nevertheless, extensions have to be programmed by a MAVA expert.
- *Simplified multimedia document dissemination*
  For a broad acceptance of multimedia documents there are certain principles that have to be considered by multimedia presentation programs. First of all is the platform independence. This ensures that documents can be viewed from a vast variety of different platforms. Most (commercial) approaches provide platform dependent viewers (also called players) for a particular document format. Second is that only one presentation system can be used regardless of the application area. Thus document providers and users have to treat only one presentation system.

3 MAVA Document Model

Basis of the extensible multimedia system MAVA is a document model. It describes the structure of documents and it is used as model for the graphical visualization of documents in the MAVA document editor.

3.1 Introduction of the Document Model

The document model used for MAVA documents is based on an operator approach. Figure 1 shows a graphical notation of the document elements. Basically, MAVA documents are represented as graphs. There are two kinds of nodes, namely media objects and operators. Media objects represent particular media items (e.g. audio or video media). All relations or dependencies between media objects are described by operators. Therefore, edges between media objects and operators determine a relation. Operators can be divided into two different classes, namely effect operators and general operators. There is one further document element called container that allows logical grouping of document parts. Following, the four document elements are described in detail:

- *Media objects* represent media items through their attributes. Examples for attributes are the media type (e.g. audio), size of the required presentation area or a reference (e.g. an URL) for the media data that has to be presented by the media item during presentation.
- *Operators* are used to specify relations between media objects. For the specification an operator distinguishes between source and destination media objects. For example, the term "text *inFrontOf* image" specifies that the text will be displayed in front of the image and not vice versa. In this example "text" is the source media object and "image" is the destination media object. To improve readability operators are written in italic letters and source media objects are placed to the left of an operator.
- *Effect operators* are a special kind of operators. They are used to describe when a particular effect has to take place (e.g. when a media item has to be faded out). Therefore an effect operator is always used in conjunction with a temporal operator and a media object whereby the temporal operator replaces the source media object.
- *Containers* are used to build logical structures in documents. This helps authors to keep track of her document because logical interrelated parts are grouped and replaced by a container node. Containers also allow the storage of parts of the document as templates for later reuse.

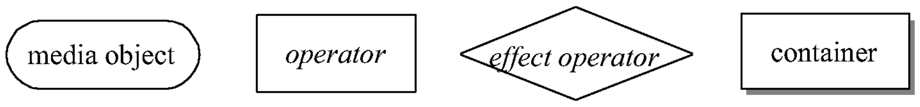


Fig.1. Elements of the MAVA document model

Compared with existing approaches it can be stated that MAVA uses the operator approach as meta document model. Existing approaches, like TIEMPO [12], Madeus [7] or SMIL [4] use also concepts respectively relations that can be represented by operators. In contrast to the MAVA approach these approaches provide a fixed set of operators. If a document cannot be specified with these operators there is no way to realize the document with these multimedia systems or languages. MAVA provides concepts for the extension with new operators and media items to support document authoring in arbitrary application areas. Hence, MAVA aims at the integration of various existing concepts instead of defining new concepts.

3.2 Extensibility of the Document Model

The presented document model serves as a meta document model. This means that only properties of the four different document elements have been described so far. For the specification of documents concrete media items and operators are required. This leads to application specific document specification languages.

A set of concrete operators is used to describe a concept. For example, a concept of temporal interval operators can be described by the seven temporal relations from Allen [1]. A lot of different concepts are possible. There are different and alternative concepts for the temporal specification (e.g. a time line), the spatial specification (e.g. absolute or relative positioning) or the specification of interaction (e.g. different menus and hyper links). And there are further application specific concepts for example a computer based training or the control of animations during presentation.

Besides operators there are also particular media items required. There are “standard” media items for multimedia documents like video or audio. Examples for time independent media items are text or images. But the number of diverse media types is not limited. To provide multimedia documents from a rich variety of application areas a lot of media items are required (e.g. different elements for interaction, visualization and simulation of protocols in computer science or chemical molecules).

The combination of concepts and media items allows to specify multimedia documents for particular application areas (e.g. a computer based training in chemistry). To provide flexibility for authors the number of concepts and media items of a multimedia system should not be limited or fixed. The MAVA approach permits extension of the presentation system and document editor with new concepts and new media items.

The definition of a meta document model and the extensibility care for simplified document authoring. The meta document model is basis of the authoring paradigm. An author creates MAVA documents by direct manipulation of a graph representing the document. Once the author knows the document model and how to use the document editor she can author documents in arbitrary application areas. Section 6 provides details about the document editor.

An assumption for simplified document authoring with MAVA is the availability of the required concepts and media items. If they are not available, the MAVA system has to be extended. For the extension of the system with a new concept operators have to be specified and their semantics have to be implemented. The semantics are realized by managers. A manager is a code unit that controls media items during presentation to assure the semantics of its associated operators. Managers will be programmed by MAVA experts who have knowledge in MAVA concepts and programming. Details about programming managers can be found in [5]. The second type of extensions, new media items, are realized similarly.

After the implementation of a concept or media item it can be used within the presentation system and the document editor. Main advantage of MAVA is the reuse of operators and media items once they have been realized. Other authors need not to realize the same concepts again. Hence, the more operators are available the less programming will be required.

Basic concepts for spatial or temporal specification are already available. Hence, compared to existing approaches extension programming is no disadvantage. On the contrary MAVA has advantages because of the extensibility different concepts (e.g. for computer based training) or simple media items for chemical visualization are already available.

3.3 Example Document

After the introduction of the document model a further sample document is shown in Figure 2. Figure 2 is a screen shot from the MAVA document editor. The document is part of a computer based training in computer science. It is used to illustrate a network protocol (stop and wait protocol) to the user. During the protocol simulation a speaker explains each step of the protocol. Hence, the document requires the following media items: a protocol visualization and audios representing the speaker's explanations.

To specify the appearance of the document several different concepts are used. First of all, the protocol visualization media item will be placed in the presentation area. The visualization media item is the only visible media item. It is connected as a destination media item to the *center* operator in the upper left of the window. The *center* operator places the media item in the center of the presentation area.

As a second step the sequence of steps and explanations has to be determined. Therefore temporal interval relations are used. A description of two different sets of interval operators can be found in [1] or [12]. The used operator is a *before* operator. Its semantics are to determine which media item is presented in advance of which other media item. For example "explanation 1" has to be presented before "explanation 2".

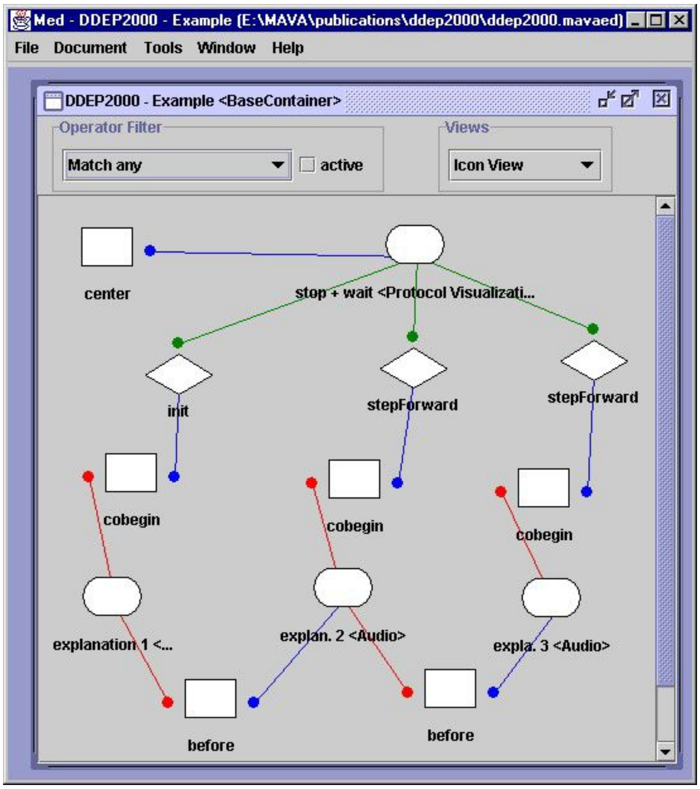


Fig. 2. Specification of a sample document

The last step is the synchronization between the explanation and the simulation is specified by the usage of effect operators. Effect operators can be used to control each step of a visualization (i.e. the concept for controlling simulations has been realized in advance). In the example two different kinds of effect operators are used. The *init* (effect) operator is used to initialize the simulation and the *stepForward* operator is used to execute one step of the simulation. The instants when the effects have to take place are determined with temporal operators. In this case the *cobegin* operator (also a temporal interval operator) is used. The semantics of the *cobegin* operator are to specify that the presentation intervals of two media items starts simultaneously. According to the definition of effect operators the *init* effect operator is connected with the *cobegin* operator. Finally, the document is specified with the desired semantics.

4 Implementation

The presented approach has been implemented using Java as programming language. This led to a platform independent realization of the presentation system. Figure 3 shows the architecture of the MAVA presentation system. It consists of the following components:

- *MAVA engine*
  The MAVA engine is the core of the presentation system. It handles all basic tasks of a presentation system like document loading, class loading and controlling the presentation.
- *Class loader*
  The class loader loads Java classes that are required for the presentation of a particular document (managers, media items and media viewers or in other words extensions).
- *Document loader*
  The document loader is responsible for loading documents from various sources like local disks or network servers. It generates the document model from the storage format of MAVA documents.
- *Manager*
  Managers are components that are responsible for the realization of semantics of particular concepts (i.e. a set of operators).
- *Operator*
  An operator is a component of the document model. It is used to describe relations between media items.
- *Internal representation*
  Internal representations are models for particular concepts that are realized with managers. They are intermediate representations of the relations described by operators. They are used to simplify the generation of the required control events. Hence, a manager will dynamically map the operators on events thus realizing their semantics [5].
- *Media objects*
  A media object is a component of the document model. It is used to realize different types of media items.

- *Media viewer*
  A media viewer is responsible for the visualization of particular media formats (e.g. a mp3 audio).

Besides the Java runtime, MAVA uses the Java Media Framework [6] to realize media viewers for continuous media items like video or audio media items.

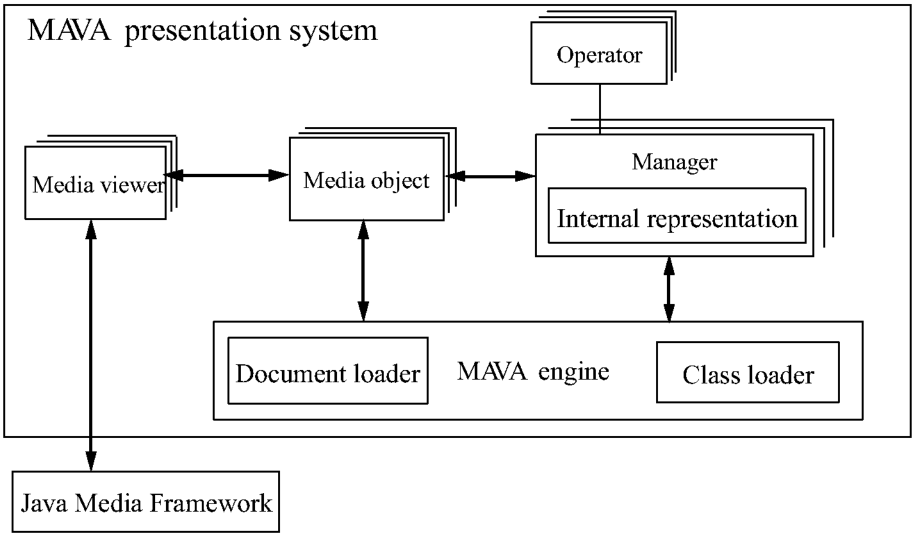


Fig. 3. Architecture of the MAVA presentation system

5 MAVA Document Dissemination

The dissemination of MAVA documents can be divided into two steps. Because of the extensibility of the document language the MAVA presentation system needs all extensions required for the presentation of a particular document (Figure 4, step 1). Therefore a document *d* comprises a sector information SI_d. This sector information describes which extensions have to be available locally or have to be downloaded (if not available) prior to the presentation (step 2). The MAVA presentation system provides a component based architecture that allows to plug in realizations for language extensions (managers, media items and media viewers).

The presentation system will automatically load and integrate all required extensions without user interaction (i.e. ad hoc extensibility). This enables the dissemination of documents that require language extensions.

The MAVA code storage and MAVA document storage in Figure 4 can be realize in different ways. MAVA documents and classes can be stored for example on a Web server or on CD-ROM without any modifications of the document. This allows an easy dissemination of MAVA documents on different storage media. Additionally, it has to be mentioned that servers providing media items have to be capable of delivering media streams for continuous media items.

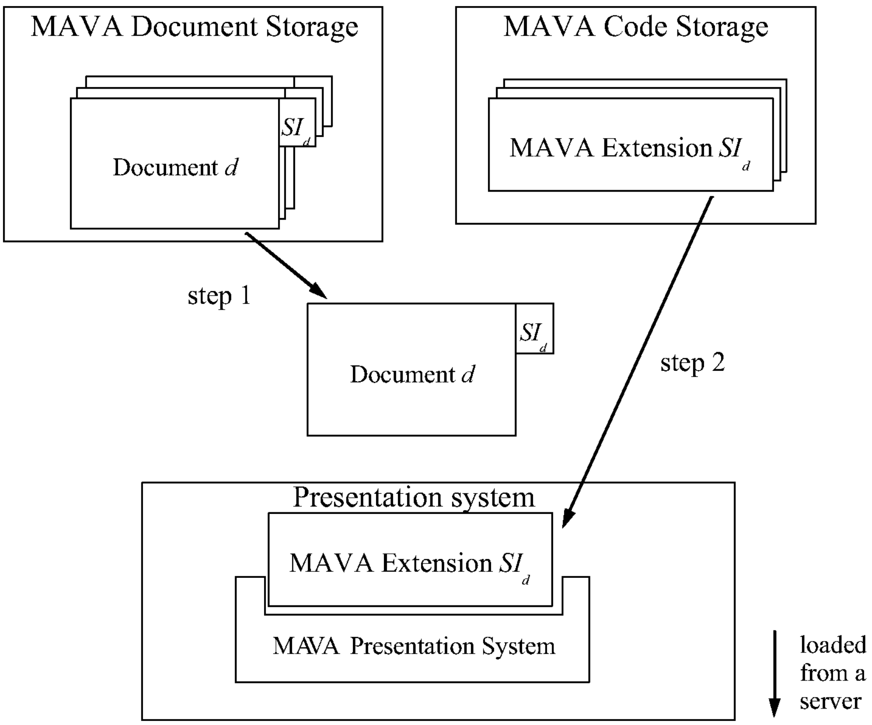


Fig. 4. Dissemination of MAVA documents

6 MAVA Document Editor

The MAVA document editor (called Med) is extensible to allow authoring of documents basing on various specification languages (i.e. application areas). This means that for each manager of the presentation system there is also an extension for the editor. This editor extension provides information required by the editor for the integration of operators of a new concept. Additionally, it is possible to integrate new media items by media descriptions.

Figure 2 shows the visualization of a document graph in the editor. The author uses direct manipulation and drag&drop to add new media items and operators to a document graph. Relations between media items are specified through edges between media items and operators. The author will connect media items and operators using the bullets to the left or right of an operator. The left bullet is used to specify the source media and the right one is used to specify the destination media. Effect operators have one bullet above the icon. That bullet is used to assign the effect to a certain media item.

The way documents are authored in MAVA is independent of used concepts and media item. This means that e.g. the temporal and spatial layout is specified in exactly the same way than any other concept (e.g. a computer based training) and requires no additional (script) programming by the author.

Because of the multitude of concepts and available operators the MAVA editor provides a filtering mechanism (see Figure 2 “Operator filter”). The user interface allows to set different filters. Filtering allows to mask out operators that do not match a filtering criteria. This permits to show only operators of a certain class (e.g. temporal operators) or of a particular concept (e.g. absolute positioning). With filtering a possible confusion of the author caused by a large number of different operators used in a document can be avoided. This ensures that the simplicity of the document authoring process is also preserved in large documents.

Another concept to simplify authoring are so called views. A view is a WYSIWYG representation of a concept. The editor window provides a selection menu for available views (see Figure 2 “Views”). For some concepts it is possible to use an alternative specification method. The concept of absolute positioning of media items for example can be easily visualized as rectangles in a presentation area. In this view the author can move and resize media items directly without using operators and specifying parameters by hand. The view will create the required operators and parameters.

7 Internet Integration

Nowadays, Internet integration of a multimedia document system is important. Almost every digital library uses a Web-based interfaces for access. Therefore a new document format and presentation system should seamless integrate with Internet technologies.

One important point is the platform independence of the storage format. A multimedia system should use a platform independent representation of the document model for the exchange of documents. For MAVA an XML-based [2] storage format was developed. The MAVA document model allows to build recursive structures with containers. These recursive structures can be represented by a tree. And because of the nature of XML a tree can be simply mapped onto an XML-based storage format. Using XML has the following three advantages: (1) it is by definition platform independent, (2) for the realization it was possible to choose between different existing parsers to realize a document loader and (3) it can be processed by other tools. MAVA and XML supplement each other well because the XML standard does not specify semantics of a markup language. This semantics realization can be provided by the MAVA approach.

In addition to the storage format also the presentation system has to be platform independent what is achieved by using Java as programming language for the implementation. This allows the presentation of MAVA documents on a vast variety of platforms. Additionally, the MAVA document editor, also implemented in Java, permits to create documents on various platforms.

The MAVA presentation system is available in two versions. The first version is a stand alone application with a graphical user interface for loading and presentation of documents. The second version is a Java applet which is used for integration into HTML pages.

Extensibility of the document language is a major advantage of the MAVA approach. It allows to author and present documents with the same system regardless of the application area. Due to platform independent realization it is possible to download extensions for both the presentation system and the document editor.

The platform independence of the MAVA approach allows a seamless integration into the Internet. Other approaches for multimedia documents (e.g. Macromedia Director) provide platform dependent plug-ins only for popular browsers. This limits the application of such approaches. Consider for example a digital library that wants to provide multimedia content to a vast community. Platform independence allows to access information from any platform and the ad hoc extensibility relieves users from configuration tasks. For users it will be transparent whether they access HTML pages or MAVA documents.

Regarding the Internet MAVA supplements the possibilities of SMIL. SMIL can be used to create documents for interactive multimedia presentations. The interaction of SMIL conforms to the hyper link concept of HTML. Compared to that MAVA allows the integration of new concepts for interactive multimedia documents or new media items.

8 Related Work

Present multimedia systems allow two different ways to extend their functionality. The first way is only possible if the source code of the system is available. Then a programmer could add new functionality. This approach is very insufficient because documents belong to particular versions of the system (which has to be available) and merging different extensions is difficult. Another way, very similar to programming is the usage of a multimedia prgramming framework. Such a framework provides programming abstractions for the development of multimedia applications and therefore simplifies the development of presentation systems for particular concepts. In [8] it was shown how such a framework (called KAOMI) can be used to realize a SMIL player or a presentation system for Madeus [7].

A second alternative is to provide a scripting language which enables the author to program particular effects into the presentation (e.g. Macromedia Director [3] or in MHEG [9]). From the point of view of simplified authoring this approach has several severe disadvantages: (1) the author has to have programming experience, (2) expressiveness and performance of the scripting language are limited, (3) reuse of scripts is difficult or impossible because they are integrated into a document and solutions for a particular problem and (4) integration of the scripting language into the document model or authoring paradigm. Especially the last point shows that if scripting languages are used, authoring tends to be more a kind of programming and thus lets the original document model loose importance. In the end, the creative part of the design task is vanishing.

Compared to existing approaches, MAVA provides a clear concept for the separation between documents and code for extensions in contrast to scripting language approaches. Scripts are included with documents what does not allow to make optimizations like caching extensions to reduce document size. Compared to approaches using a multimedia middleware, MAVA provides a higher abstraction level for the extension (e.g. a manager and operator concept). Reuse of existing concepts and a common document model require less effort for the realization of a new concept than programming a new presentation system or using a middleware.

9 Conclusion

This paper presents an approach that realizes an extensible multimedia document language, an appropriate presentation system and a document editor. Main goal of the presented approach is to support extensibility what leads to the simplification of multimedia authoring and to support dissemination of multimedia documents. The need for simplified multimedia authoring can be derived from the existence of different application areas for multimedia documents and the required programming knowledge of today's authors. Complex multimedia documents are often programmed, in particular in the commercial sector (i.e. Lingo programming with Macromedia Director). This approach shows how the requirements for authors regarding programming knowledge can be lowered. A simple authoring paradigm (i.e. direct manipulation of a document graph) is used independently of the application area of the document. Once an author has learned this authoring paradigm she only needs to understand the concept of a MAVA document language extension (i.e. the semantics of a set of operators).

For the dissemination of MAVA documents a platform independent storage format and a platform independent presentation system is required. This has been realized by the implementation of MAVA using Java and XML. Additionally MAVA provides ad hoc extensibility which means automatic update and configuration of the presentation system with extensions required for the presentation system.

References

1. Allen, J.F. “Maintaining Knowledge about Temporal Intervals”. Communication of the ACM, Vol 26, Num 11. 11/1983.
2. Bray, T., Paoli J. Sperberg-McQueen, C.M. .“Extensible Mark-up Language (XML) 1.0”. Recommendation. W3C. 1998.
3. “Director version 8.0”. URL: http://www.macromedia.com/products/director. Macromedia. 1999.
4. Hoschka, P. (Ed.) “Synchronized Multimedia Integration Language (SMIL) 1.0 Specification”. W3C Proposed Recommendation. W3C. 4/1998.
5. Hauser, J. and Rothermel, K. “Specification and Implementation of an Extensible Multimedia System“. In Proceedings of the IDMS 2000, LNCS 1905. Springer, 2000.
6. “Java Media Framework”. URL: http://java.sun.com/products/java-media/jmf/index.html.
7. Jourdan, M., Layaida N., Roisin, C., Sabry-ismail, L., Tardif, L. “Madeus, An Authoring Environment for Interactive Multimedia Documents”. In Proceedings of the ACM Multimedia 98. 1998.
8. Jourdan, M., Roisin, C., Tardif, L. “A scalable Toolkit for Designing Multimedia Authoring Environments”. In special Issue “Multimedia Authoring and Presentation: Strategies, Tools and Experiences” Multimedia Tools and Applications Journal. Kluwer Academic Publishers.1999.
9. ISO/IEC DIS 13522-5. ”Information Technology - Coding of Multimedia and Hypermedia Information - Part 5”. 1995.
10. Pérez-Luque, M.J.; Little, T.D.C. “A temporal Reference Framework for Multimedia Synchronization”. IEEE Journal on Selected Areas in Communication, 14 (1996).
11. German Research Foundation (DFG). “Distributed Processing and Exchange of Digital Documents (V3D2)“. URL: http://www.cg.cs.tu-bs.de/dfgspp.VVVDD.
12. Wirag, S. “Specification and Scheduling of Adaptive Multimedia Documents“, Faculty Report 1999/04, University of Stuttgart, 1/199

Perceptually-Tuned Grayscale Characters Based on Parametrisable Component Fonts[1]

Changyuan Hu and Roger D. Hersch

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
RD.Hersch@epfl.ch

Abstract. Our *component-based parametrisable font system* is a newly developed font description and reproduction technology. It incorporates for each basic character shape a software method responsible for the synthesis of an instance of that character. A given font is synthesized by providing appropriate font parameters to these character synthesis methods. Numerous concrete fonts can be derived by simply varying the parameters. Such variations offer high flexibility for synthesizing derived fonts (variations in condensation, weight and contrast) and enable saving a considerable amount of storage space. This paper shows that with component-based parametrisable fonts, high quality perceptually-tuned grayscale characters can be generated without requiring hinting information. Generating perceptually-tuned grayscale characters with parametrized component-based fonts consists in automatically adapting the phase of some of the character's parameters in respect to the underlying grid and in ensuring that thin character parts are strong enough not to disappear (weight-control). The presented method is especially powerful for generating high-quality characters on LCD displays (cellular phones, pen-computers, electronic books, etc..).

1 Component-Based Parametrizable Fonts

Most commercial font systems use outline fonts for the digital presentation of typefaces. Outline fonts are well suited for character rasterization and printing. They however only incorporate implicit information about important font features such as the width of stems, bars and bowls, the distance between vertical stems and bowls and the parameters determining junctions between character elements and determining the shape of terminal elements (serifs). Efforts are made to create a more flexible character representation incorporating explicit information about features of the character, thus enabling the synthesis of coherent font variations, for example by varying weight and contrast. The main idea is to describe characters by an assembly of appropriate structure elements (components). Several attempts have been made to describe characters by structure elements. The earliest attempt to describe typographic characters by parametrisable shape primitives aimed at minimizing the amount of font memory for digital typesetter controllers [1]. D. Knuth created the Metafont system for designing new font families. He described his Computer Modern fonts by

[1] A preliminary version of this paper was presented at the ICIP'99 Conference in Kobe, Japan

horizontal, vertical and diagonal strokes as well as by round parts [6]. These strokes and round parts are given by sequences of pen positions and orientations. More recently, Bauermeister et al. created the Infinifont system enabling resynthesizing an existing font from universal font generation rules and from parametric data specific to that font [7][10]. Our own efforts [12] aimed at defining character components suitable for the generation of existing and of derived typographic fonts, incorporating variations in condensation, weight and contrast.

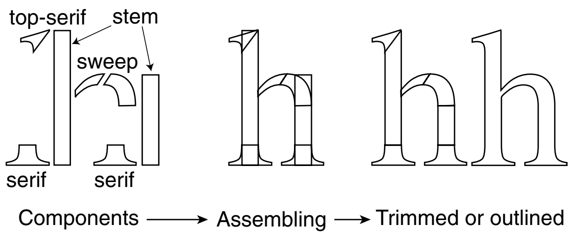


Fig. 1. Building a character with components

Fig. 1 shows a character assembled from structure elements, i.e. the two vertical stems are synthesized by the component named *stem*, the round arch is synthesized by two connecting *sweep* components, and the terminals are synthesized respectively by the *serif* and *top-serif* components. For the round character structure elements which are similar to a whole or a half of letter “o”, we synthesize them by the *loop* and respectively the *half-loop* components (letters “o” and “b” in Fig. 2).

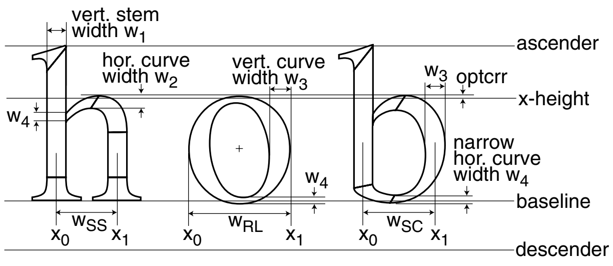


Fig. 2. Characters generated by filtering and resampling compared with manually edited grayscale characters

In our component-based font parametrization method, the shape of a character is synthesized by synthesizing the shape of each component from font parameters. Font parameters have clearly defined typographical meanings (Fig. 2). They are used to describe the characters’ typographical features, such as the character height alignment lines (reference lines), the widths of stems and arches, the metrics of serifs, the angles of junctions or slant serifs, etc. Some parameters are globally available and are applied to all or to a group of several characters to ensure the coherence of a font, respectively the coherence of a group of parameters. Some parameters have only a local meaning for one specific character. The concrete values of parameters may be the distance between two points, the intermediate position (proportion) of one point between two other points, the angle of a junction or of a slant serif, etc.

Components are assembled according to a dependency order: the main components, such as the two vertical stems in letter “h” or the left stem and the right half-loop of letter “b”, are first placed according to a parameter controlling their horizontal distance. The position of other components such as connecting sweeps and serifs are determined by their relationship to the previously placed main components. Such a construction of characters by components enables connection relationships between components to be maintained when modifying parameter values. Derived characters of varying condenstation, weight and contrast can therefore be obtained by modifying a suitable subset of parameters [12].

2 Perceptually-Tuned Grayscale Fonts

In order to improve the display of text on limited resolution computer displays, researchers have tried to trade-off the lack of spatial resolution for an increased number of intensity levels [2], [3, 4]. Especially with respect to typographic text display on limited resolution CRT and LCD display devices, it has been shown that using grayscale makes font dependent character features visible which disappear when displaying text with bilevel characters. Though they can more accurately render font differences, grayscale characters generated by filtering and resampling high-resolution bilevel master characters look rather fuzzy. Such characters do not have a high enough contrast along their horizontal and vertical edges and thin character parts tend to disappear. In addition, similar character parts (bars, stems, etc..) tend to look dissimilar.

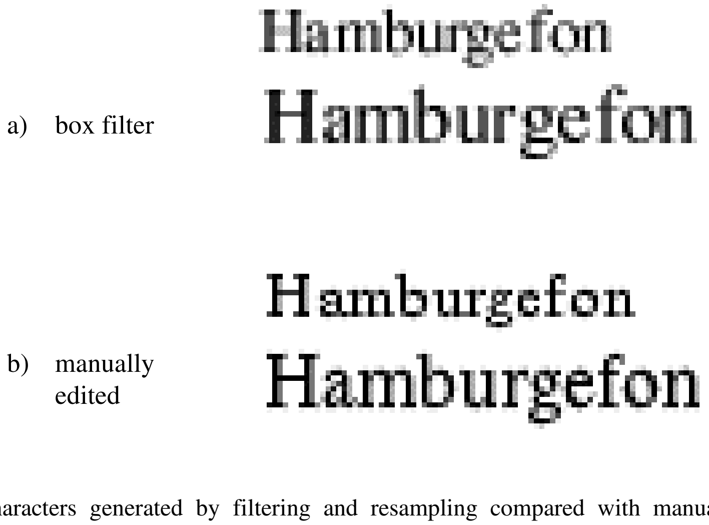


Fig. 3. Characters generated by filtering and resampling compared with manually edited grayscale characters

Different instances of similar character structure elements such as vertical bars or curved stems look dissimilar because they do not have similar intensity profiles (Fig.3a). This is due to the fact that the original bilevel master character incorporates frequencies well beyond the Nyquist limit (1/2 the resampling frequency) and that

LCD displays are capable of displaying individual pixels as constant intensity squares of unit size, thereby enabling high intensity gradients between neighbouring pixels.

The phase of character elements with respect to the target pixel grid has a strong impact on their intensity profiles, especially at small sizes and low resolution (Fig. 4a).

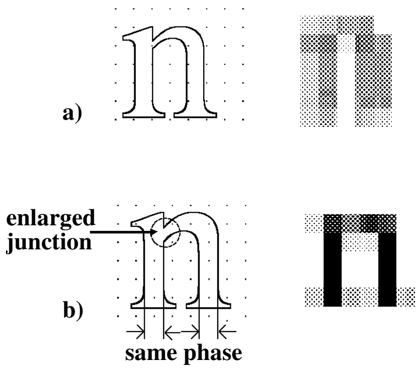


Fig. 4. The phase of the vertical bars defines their intensity profile: (a) without outline phase control and (b) with outline phase control

Creating optimal grayscale characters involves a manual pixel-by-pixel design that must follow strict typographic rules. In order to automatically generate improved quality grayscale characters similar to the characters which would be designed manually by skilled typographic designers, the so-called *perceptually-tuned grayscale font generation method* [11,12,15] successfully applies typographic rules for the automatic grayscale character generation process. Typographic rules derived from manual pixel-by-pixel design have been translated into a set of character outline grid-fitting rules, such as phase-control of the vertical main stems in order to enhance the sharpness of their left edge, thickening thin strokes when their width is too small, and phase- control of round strokes so that the extrema of round character (such as “c”, “e” and “o”) have a coherent appearance.

To fulfill these grid-fitting tasks, traditional outline font technology requires additional information called hints (or instructions in TrueType terminology) enabling the rasterizer to adapt the scaled character contour to the pixel grid by applying slight modification to the character shape contours [11].

3 Synthesis of Perceptually-Tuned Grayscale Characters with Component-Based Parametrisable Fonts

With the component-based parametrizable font technology, grid-fitting can be applied without explicit hinting information. This is possible, because the component-based description of a character explicitly specifies the typographical meaning of each character component, e.g. the *stem*, the *serif*, the *loop* and the connecting *sweep* components. The component synthesizer knows how to synthesize the shape of a component from font parameters. Our approach consists in modifying global and local parameters (Fig. 2) which control the phase of critical component contours in respect to the grid.

We consider the placement and rasterization of component-based characters on a 2D pixel grid where pixels have a size of one and where pixel centers have integer coordinates. The scaling factor *scaleFac* specifies how much the original global parameters need to be scaled to yield the final character at the target size.
We control the phase of reference lines in order to ensure that the grayscale representation of round letters (such as letter "o") remains vertically symmetric and that foot serifs remain sufficiently visible. Reference lines should therefore be placed at the boundaries between rows of pixels. Let us first place the baseline at the nearest boundary between pixels:

$$baseline^{new} = \lfloor baseline \times scaleFac \rfloor + 0.5$$

Other reference lines are placed relatively by rounding the distances between them and the baseline. Therefore, the other reference lines are also located on pixel boundaries:

$$d_1 = \lfloor (xheight - baseline) \times scaleFac + 0.5 \rfloor$$

$$xheight^{new} = baseline^{new} + d_1$$

$$d_2 = \lfloor (ascender - baseline) \times scaleFac + 0.5 \rfloor$$

$$ascender^{new} = baseline^{new} + d_2$$

$$d_3 = \lfloor (descender - baseline) \times scaleFac + 0.5 \rfloor$$

$$descender^{new} = baseline^{new} + d_3$$

where $\lfloor \;\; \rfloor$ truncates a number to its integer part

We align the left edge of all vertical stems to horizontal pixel boundary by first placing the centerline of one of the vertical stems in a character (for example the left ascender stem of letter "h") to a phase ensuring that the left side of the stem is on a pixel boundary. Parameter w_1 defines the vertical stem width (Fig. 2). The position of the second stem is calculated by adding to the position of the first stem the rounded scaled value of the *stem-to-stem distance* (parameters w_{SS} in Fig. 5).

$$w_{SS}^{new} = \lfloor w_{SS} \times scaleFac + 0.5 \rfloor$$

$$x_0^{new} = \left\lfloor \left(x_0 - \frac{w_1}{2}\right) \times scaleFac \right\rfloor + 0.5 + \frac{w_1 \times scaleFac}{2}$$

$$x_1^{new} = x_0^{new} + w_{SS}^{new}$$

For round letters, such as letter “o”, “c” and “e”, we place the horizontal position of the center of symmetry on a pixel center so as to ensure that left half and the right half of the letter are symmetric in respect to the underlying grid and are therefore rendered with a symmetric intensity profile. The *round letter width* parameter (w_{RL} in Fig. 2) needs only a simple scaling:

$$w_{RL}^{new} = w_{RL} \times scaleFac$$

In order to obtain the position x_0^{new} of the left extremity of the round letter shape, we need to center it horizontally on the grid. This left extremity is obtained by truncation of the scaled left extremity minus a fractional value representing half the difference between the full letter width and the truncated letter width.

$$x_0^{new} = \lfloor x_0 \times scaleFac \rfloor - \left(\frac{w_{RL}^{new} - \lfloor w_{RL}^{new} \rfloor}{2} \right)$$

$$x_1^{new} = x_0^{new} + w_{RL}^{new}$$

For letters with one vertical stem and a half loop, such as “b”, “d”, “p” and “q”, we control the phase of the vertical stem in the same way as for letter “h” and scale the distance between the stem and the half loop (w_{SC} in Fig. 2). No phase control is applied to the half-loop component.

Finally, the width (thickness) of a thin curved or a diagonal stroke needs to be increased if it is too thin, i.e. less than 1 pixel. This can be done by enforcing a lower limit (for example 0.75) to the corresponding width parameters (w_4 in Fig. 2) after scaling.

$$w_4^{new} = max(0.75, w_4 \times scaleFac)$$

Fig. 5a illustrates the synthesized component-based characters obtained by applying the previously described parameter scaling, phase control and thickness modification rules.
Fig. 6a shows the enlarged image of the grayscale characters generated by our component-based parametrizable character synthesis system [12], in comparison with the perceptually-tuned grayscale characters (Fig.6b) generated by the previously developed hint-based grid-fitting system [11] and in comparison with traditional grayscaling, i.e. filtering and resampling of high resolution master character (Fig. 6c).

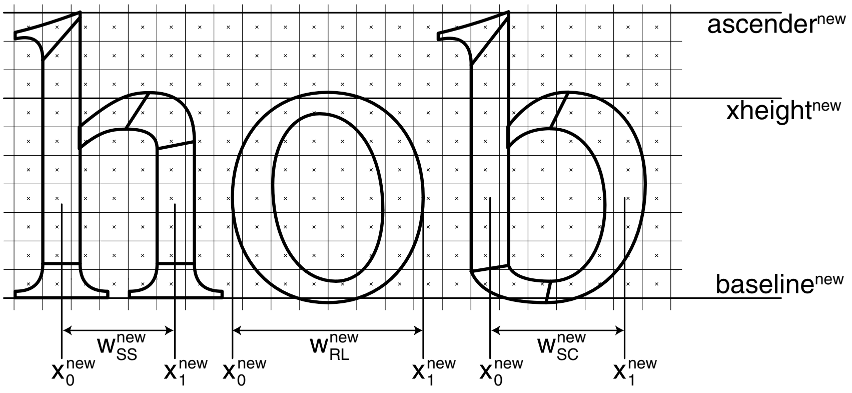


Fig. 5. The contours of synthesized component-based characters are adapted to the pixel grid by applying the on-the-fly parameter scaling and modification rules.

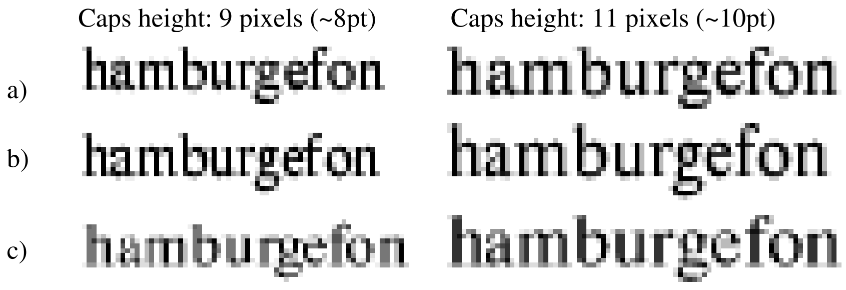


Fig. 6. (a) Component-based generation of grayscale characters, with improved contrast obtained thanks to phase control of stems and round parts and to thickness control of thin strokes; (b) Perceptually-tuned grayscale characters generated by applying hinting instructions to the grid-fitting process; (c) corresponding grayscale characters generated by filtering and resampling (traditional technique)

4 Conclusions

The component-based parametrizable font system enables the synthesis of coherent fonts. Parameters such as the relative position of reference lines, the spacing between main character parts, the bar and stem widths define to a large extent the shapes of the synthesized characters. By controling the phase of these parameters, sufficient grid-fitting is achieved to generate high-contrast uniformely looking grayscale characters. Phase control is applied to reference lines, to stem and bar centerlines and to width parameters. At small sizes, a sufficiently strong structure of individual strokes is achieved by ensuring a minimal size of stem and of curved stroke width parameters (weight control). Phase and weight control are completely independent of component-based character generation: a separate program is responsible for phase and weight control of specific fonts paramters. After phase and weight control of parameters, component based bilevel characters are generated at a resolution 4x4 times higher

than the target resolution. Each 4x4 bitmap square is in phase with one grayscale pixel and yields one grayscale pixel whose intensity is one of 17 possible intensities.
Once characters are described by components, the presented method is simple to implement and does not require, besides component contours, any additional information. It may become part of the facilities offered by a completely new generation of component-based font synthesizing systems. Its well contrasted and uniformly looking grayscale characters may considerably improve reading confort, especially on limited size LCD displays (pen computers, cellular phones).

References

1. P. Coueignoux, "Character Generation by Computer", Computer Graphics and Image Processing, Vol. 16, 1981, 240–269.
2. F. Crow, "The Use of Grayscale for Improved Raster Display of Vectors and Characters", Proc. Siggraph'78, *ACM Computer Graphics*, Vol. 12, No. 3, 1978, 1–6.
3. J. Warnock, "The Display of Characters Using Gray Level Sample Arrays", Proc. Siggraph'80, *ACM Computer Graphics*, Vol. 14, No. 3, 1980, 302–307.
4. A. Naiman, A. Fournier, "Rectangular Convolution for Fast Filtering of Characters", Proc. Siggraph'87, *ACM Computer Graphics*, Vol. 21, No. 4, 1987, 233–242.
5. P. Karow, "Automatic Hinting for Intelligent Font Scaling", *Proc. Raster Imaging and Digital Typography 89,* (Eds. J. André, R.D. Hersch), Cambridge Univ. Press, 1989, 232–241.
6. D. E. Knuth, *The METAFONT book*, Addison-Welsey, Reading Mass. 1986.
7. C. D. McQueen and R. G. Beausoleil, "Infinifont: a parametric font generation system", *RIDT 94, Electronic Publishing,* Vol.6(3), John Wiley & Sons, 1993, 117–132.
8. R. D. Hersch, C. Bétrisey, "Model-based matching and hinting of fonts", Proceedings SIGGRAPH'91*, ACM Computer Graphics*, Vol. 25, No. 4, 1991, 71–80.
9. A. Shamir, A. Rappoport, "Extraction of Typographic Elements from Outline Representations of Fonts", Proc. Eurographics 1996, *Computer Graphics Forum*, Vol. 15, No. 3, 259–268.
10. Bauermeister et al., "Method and system for creating, specifying, and generating parametric fonts", *United States Patent No. 5586241*, Dec. 17, 1996.
11. R. D. Hersch, Claude Betrisey and Justin Bur, "Perceptually Tuned Generation of Grayscale Fonts", *IEEE Computer Graphics and Applications*, Vol. 15, No. 6, 1995, 78–89.
12. C. Hu, R.D. Hersch, Parameterizable Fonts Based on Shape Components, IEEE Computer Graphics and Applications, Vol. 21, No. 3, May/June 2001, 70–85.
13. J. Hertz, C. Hu, J. Gonczarowski, R. D. Hersch, "A Window-Based Method for Automatic Typographic Parameter Extraction", *Electronic Publishing, Artistic Imaging and Digital Typography*, (Eds. R.D. Hersch, J. André, H. Brown), LNCS 1375, Springer Verlag, 1998, 44–54.
14. K. O'Regan, N. Bismuth and R. D. Hersch, "Legibility of Perceptually-Tuned Grayscale Fonts", *Proc. ICIP-96*, Ed. P. Delogne, Vol 1, 1996, 537–540.
15. R.A. Morris, R. D. Hersch, A. Coimbra, "Legibility of Condensed Perceptually-Tuned Grayscale Fonts", *Electronic Publishing, Artistic Imaging and Digital Typography*, (Eds. R.D. Hersch, J. André, H. Brown), LNCS 1375, Springer Verlag, 1998, 281–293.
16. J. Valdes, E. Martinez, US patent 5438656, Raster shape synthesis by direct multi-level filling filed June 1, 1993, Issued Aug. 1, 1995, Applicant Ductus Inc., Mountain View, CA

A Simple Management Tool for Medium-Sized Web Sites

Igor Fischer[1] and Andreas Zell[1]

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Köstlinstr. 6, 72074 Tübingen, Germany
{fischer, zell}@informatik.uni-tuebingen.de
http://www-ra.informatik.uni-tuebingen.de/

Abstract. Site management is one of the key issues in Web publishing. Whereas smaller sites can be still managed manually, large Web sites generally require a systematic approach, usually involving sophisticated tools operated by skilled personnel. Between these two extremes are medium-sized sites. Although site management tools exist for both extremes, the market coverage of the mid-size segment is somewhat sparse. In this paper we describe our own tool, aimed at small and medium sized Web sites, of some 1000 pages. It is simple to use and needs no special skills or knowledge to be operated. Yet it is powerful enough to manage a site of a small organization and to enforce a common corporate identity. We use it for management of our department site, as well as for managing contents of various on-line education projects.

1 Introduction

The continuous and exponential growth of the WWW has been marked not only by an increasing number of Web sites, but also by increasing size and complexity of the sites themselves. Today, it is not unusual for a Web site to contain tens, or even hundreds of thousands of documents linked with each other. For example, the AltaVista search engine lists over 20000 documents in the domain java.sun.com alone, and there are probably many more[1]. Moreover, large sites can even span many servers distributed around the world. Since sites are non-static entities, with pages continuously being added, removed, changed or linked in a different way, their management has increasingly become an issue.

The term “Web site management” is used to describe a wide variety of activities related to publishing contents on the WWW, ranging from authoring, over maintaining consistent structure, to traffic analysis and site optimization. For the purpose of this paper, by managing a Web site we shall mean a subset of those activities, namely keeping site structure under control, as well as ensuring common appearance of corresponding pages, regardless of their contents.

[1] Experience shows us that, as a rule, search engines do not index all pages on the WWW. For example, our department site has some 200 pages, but only 16 appear in the AltaVista listing.

Under structure we understand the way pages are linked to each other and how they interact with server-side modules, like servlets or CGI scripts. More precisely, the site structure describes which pages comprise the site and how they relate to each other. Although the structure of a web site can generally be any kind of a directed graph, for navigational simplicity tree-like structures prevail. Consequently, most relationships are of hierarchical (like super-/subordinated, or parent/child) or peer level (predecessor or successor) nature, with few exceptions, most notably the home page, which is in many sites linked to from all pages.

By common appearance we mean coherent use of fonts, colors, graphics, logotypes, navigational elements and layout in all pages.

Although smaller sites of, say, few dozen Web pages still can be managed manually, this approach relies heavily on the human factor and is therefore error-prone. For large sites, with many thousands of pages, management has to be machine-supported in some way. Different tools have been developed to cope with the challenge, ranging from simple visual tools, like Microsoft FrontPage, for managing smaller sites, to big database systems with differentiated authoring, access and version control, capable of dynamically generating personalized contents on the high end.

In the former approach, a web site manager uses a visual site editor for determining the site structure. This approach is intuitive and simple as long as the display remains comprehensible, but for large sites, the screen easily becomes confusing and jammed with pages and links.

The latter approach does not rely on a visual representation of site, although it may still include it as a convenience. In that approach, content and site management are specialized tasks, being generally performed by different people.

The rest of this document is structured as follows: We first take a look at some existing site management tools, then we state the scope and target audience of our tool, analyze the tool structure and show subsequently some example sites managed by the tool. In conclusion we summarize the results and discuss possibilities for further development.

2 Related Tools

Rapid growth of the WWW has been followed by development of various Web-related tools and applications, from rather specialized ones, like link checkers, to comprehensive site management and analysis applications. Analyzing all of them would be beyond the scope of this article, but we shall take a brief look at some of their representants.

One very popular site management tool for smaller sites is Microsoft FrontPage [1], available for Windows platforms. Developed as a complete solution for Web publishing, it concentrates mainly on page design, but also includes visual site management. Former versions relied on an integrated Web server and generated proprietary code, but in the “2000” version those issues are improved.

For the design of sites with common look-and-feel, FrontPage provides modifiable page templates. The tool is itself simple to use, especially for Microsoft Office-experienced users, but, due to predefined templates, limits the designer's freedom. A mightier tool, aimed at professional programmers, is Visual InterDev [2].

Another popular tool is NetObjects Fusion [3], designed for small business. Like FrontPage, Fusion is a visual tool with focus on desktop-publishing-like page design, but also allows drag-and-drop design of sites. It encapsulates the site in a proprietary database format, thus disallowing any manual modifications by the author. It is reportedly very good for designing pages with pixel-level precision, but has problems with importing existing sites [4]. Fusion is available for Windows and Macintosh platforms.

Symantec Visual Page [5] for Windows and Macintosh is primary a Web authoring tool, but also offers some basic site management functions. A site is stored in a project file, but it does not describe any linear or hierarchical structure. Therefore, no automatic generation of navigation elements is provided. Instead, all links have to be entered manually.

A tool more oriented on content management than on page design is the UserLand Frontier [6], specifically aimed at news-oriented sites. Common appearance of pages is ensured through use of templates, and linear navigation is simple to implement, but not site maps, table of contents etc. All relevant data (contents, templates and site structure) is stored in an object-oriented database in a proprietary format. Contrary to previously mentioned tools, a visual representation of site structure is not provided. Frontier is available for Windows and Macintosh platforms.

The SGI Site Manager [7] is capable of representing the complete site structure in three-dimensional hyperbolic space. This tool is more sophisticated, consisting of a client and a server part, and should be configured by a Webmaster or a user who has experience with server configuration. Besides for content management, this tool provides for site traffic analysis. Common appearance of pages is provided through templates, which can be created not only for HTML, but for any file format. It does not generate navigation elements automatically, and is available for IRIX 6.2 and higher platforms.

On the high end are site management tools like Interwoven TeamSite [8]. Based on a hybrid architecture (file system and database), this tool is suitable both for existing and legacy sites, as well as for well structured state-of-the-art sites, with templates, XML, dynamic contents generation and many more. It is available for Windows, Mac and Solaris platforms, but its price, starting at USD 70000, places is out of reach for smaller organizations.

3 Scope

Site management tools developed up to date mainly concentrate on large Web sites, usually involved in e-business. The coverage of small and medium sites, consisting of roughly 1000 pages and needing coherent and appealing corporate identity is notably sparser. This is especially true for non-Windows platforms. Medium-sized sites are already too large to be efficiently managed by simple visual tools. Additionally, many of those tools, most of them with emphasis on Web design, although offering a number of predefined templates, limit the authors freedom in designing his own common appearance of pages. On the other hand, a high-end professional solution is likely to be too expensive to acquire and to support for an organization with a relatively small site.

A typical example of such organization would be a small or mid-size organization or company with rented Web space. The server administration would be generally outsourced to the Internet Services Provider, but the organization is likely to remain in charge for the contents. The following model describes the constellation:

– the organization designs a visual appearance for presentation on the Web, corresponding to its corporate identity, or hires a professional designer for the job,
– it creates the contents to be published on the Web,
– it converts the contents into HTML and shapes it to suit the designed appearance, and
– passes it to the Web-space provider, who publishes it.

The whole publishing process should be preferably performed modularly, in independent steps. The Web designer needs neither contents nor Web access for designing the appearance. He or she designs only templates, which are to be filled with contents. The contents creator, on the other hand, doesn't rely on the design when creating the contents. The two components meet only when melting them together into final HTML files which are to be published. But even then, the resulting site can be viewed off-line, as long as no interactive components, like database queries, are involved. The need for a Web server comes into appearance only in the final step, when the content is published on the Web.

To meet the needs of that group, we developed a Web site management tool that is still relatively easy to use for Web authors, but powerful enough to manage such medium sized sites. In the development of our tool, we were guided by the idea of simplicity. More specifically, we wanted the tool to fulfill the following requirements:

– easy to install,
– simple to use,
– easy to maintain,
– able to maintain legacy HTML pages without further modifications,
– not to use proprietary data formats,
– independent of specialized editors or viewers,
– able to work off-line, independent of Web server

The target group we had in mind were content authors. The only assumption we made about them was possession of some HTML knowledge. Therefore, the tool has to offer maximum possible freedom in developing design and contents and spare the authors of technicalities connected with server administration.

As could be expected, developing such a tool could not be done without trade-offs. Above requirements could not be met without imposing some constraints on the site structure. They were the following:

– the site structure can be considered static,
– page contents are independent of their position in the site structure.

The first constraint simply states that the number of pages remains the same and that links between them don't change. At the first glance it might seem as a very hard constraint, since it excludes dynamically generated pages, like from a database or search engine. Fortunately, this is not the necessarily the case: if only a limited number and kind of pages is generated dynamically, one can reserve proxy pages in the site structure, which acquire their content dynamically. For example, one can reserve a "search" page in the site structure, whose content is then generated according to the search results.

What the constraint really makes impossible is management of dynamic sites, like personalized sites, where users practically create their personal view of the site, or where sites automatically adapt to users. Those features are, however, not very common yet and for small and medium sites of limited interest.

The other constraint is even easier to meet. It aims at uncoupling the development of contents from development of site structure. In practice it means that contents authors should not reference other pages relatively to the current one (like "see previous page"), but only by their file name (like "see page *introduction.html*"). For hyperlinked medium like WWW it is anyway the preferred way, because the reference can be implemented as a hyperlink to the referenced page. Those hyperlinks can still be relative inside the directory tree containing the pages. The directory structure does not have to correspond to the site structure, but site maintenance might be somewhat simplified if it does.

4 Tool Structure

The core of our tool is a Perl script which takes site description, page templates and contents pages as input and produces complete linked site as output (figure 1). Instead of using a database or some proprietary format for storing the site structure and its contents, we basically rely on HTML. This makes the tool extremely easy to use for a HTML author, because HTML can easily be edited, even with pure text editors. So, all inputs to the processing tool are HTML-based, with some extensions that control the processing and the output is strictly HTML — at least to the degree the author of contents and templates adhered to the standard.

The notorious HTML property of mixing contents and structure comes helpful in our case. We use HTML in description file to describe the site structure. In templates, we mix the structure with contents to achieve a page appearance that is dependent on the page position in the site structure. The tool imposes no restriction on the way authors write contents pages, except that it expects them to be in HTML.

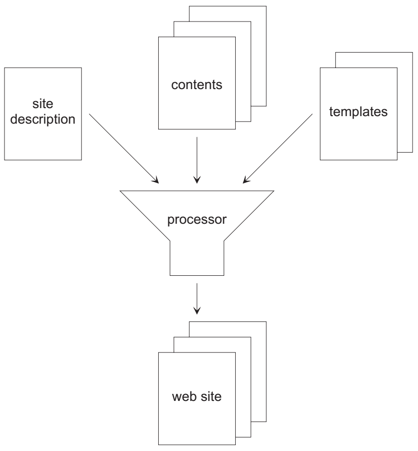


Fig. 1. Schematic tool structure.

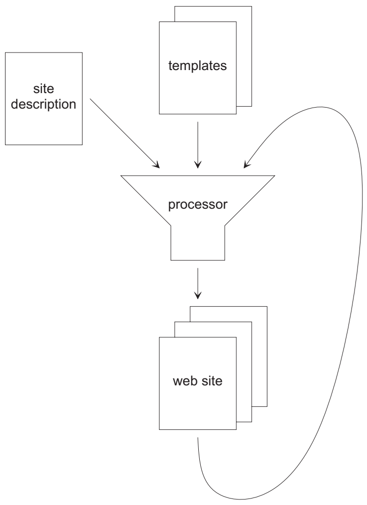


Fig. 2. Schematic tool structure for “site recycling”.

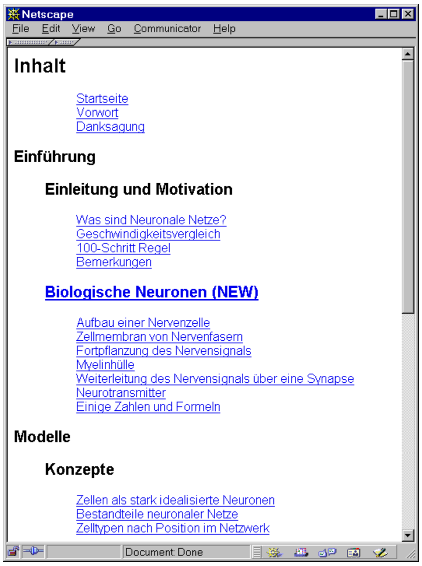
The description file describes site structure in a similar way the Netscape bookmark file organizes user’s bookmarks. The logical organization of a site is hierarchical, in a tree structure. The tree consists of nodes and branches, branches representing hierarchical relationships between nodes. Nodes are generally named, and each one *can* have a page assigned to it. This is a difference to most other site management tools, which *require* a page to be assigned to each node. By allowing nodes without assigned pages, we provide a way for jumping directly to lower-level pages, without having to stop at intermediary pages of no interest, but still maintaining a consistent site structure. This concept resembles the idea of abstract classes in object-oriented programming, which have no functionality but to serve as basis for child classes.

The tree structure in the description file is represented by nested HTML *definition lists* (`<DL>`), with each node being a *definition term* (`<DT>`). If a node has a page assigned to it, the page is represented by a HTML link to it (`<A HREF="...">`).

To facilitate the processing, we have included some proprietary extensions to standard HTML. All extensions are optional and influence in no way standard HTML parsing of the file. At the beginning of the file, a block with system and user variables can be included. This block is a HTML comment, starting with `<!--VARIABLES` and ending, like all HTML comments, with `-->`. The system variables control the processing: they specify where to find contents files, where to put the generated files, which templates to use and so on. An example is given in table 1 in the next section.

Additionally to them, a user can define his own variables it the `VARIABLES` block, like copyright note, author name and so on. These variables, if given in template files, get substituted for their values during processing.

After the `VARIABLES` block, the site structure, as nested definition lists, follows. The proprietary extensions to the `<DT>` tag, analogous to the Netscape extensions used in bookmarks file, appear after `DT` and before closing angle bracket. They are used to force use of another template, to implicitly specify a sequence of source files to be inserted at that point etc. Table 2 describes some example extensions.



Inhalt

- Startseite
- Vorwort
- Danksagung

Einführung

Einleitung und Motivation

- Was sind Neuronale Netze?
- Geschwindigkeitsvergleich
- 100-Schritt Regel
- Bemerkungen

Biologische Neuronen (NEW)

- Aufbau einer Nervenzelle
- Zellmembran von Nervenfasern
- Fortpflanzung des Nervensignals
- Myelinhülle
- Weiterleitung des Nervensignals über eine Synapse
- Neurotransmitter
- Einige Zahlen und Formeln

Modelle

Konzepte

- Zellen als stark idealisierte Neuronen
- Bestandteile neuronaler Netze
- Zelltypen nach Position im Netzwerk

Fig. 3. Site structure, viewed with a HTML browser.

Site structure can be viewed with an ordinary HTML browser, as shown in figure 3. Both hierarchical site structure, as well as linear order of pages, is easy to understand.

The template files can be seen as wrappers for contents files. They define the appearance of web pages, including navigation bars, buttons and banners. They are basically HTML pages with processing directives, specifiers and variables that during processing get substituted for currently applicable values. All these HTML extensions appear as HTML comments, i.e. surrounded by `<!--` and `-->` strings. They represent page title, hypertext references to this and other pages, and so on. For some of these elements, the current value depends only on the page being processed and for others on the site structure and the page position in it. Together, they are used for automatic generation of navigation elements, indices and site maps. Some examples are presented and described in section 5.

It is possible to use a same directory for both contents (input) pages and the resulting web site. In that case, the processing tool does not take whole HTML pages as a source, but extracts the original contents from them for processing. This feature makes it possible to update a complete site – for example, change its design or update links – without having a separate copy of it for publishing. It is very useful for maintaining existing sites, since one can make changes directly on its pages, without the need for a separate publishing step. The tool structure that corresponds to this concept is shown in figure 2.

Design of template files should preferably be done by a HTML-experienced designer. The basic layout can still be designed with a visual tool, and then manually extended and composed together to build a suitable template file. As far as contents authors are concerned, this tool imposes no obstacles, but allows them to concentrate on contents, without worrying about appearance.

5 Example Sites

The site management tool described here has been originally developed in the *VirtuGrade* project, for the management of educationally oriented Web sites. Although it is still its main application area, it can also be used for general site management. For example, we use it for the management of our department web site.

```
<!--VARIABLES
  $FILTER *.html
  $COPY_FILES logo/*.jpg logo/*.gif
  $TEMPLATE TemplateStart.html
  $SOURCE source
  $DEST dest
  $PAGE a
-->

<H2>Inhalt</H2>
<DL>
  <DL>
    <DL>
      <DT><A HREF="index.html">Startseite</A>
      <DT TEMPLATE="Template.html"><A HREF="book.00.html">Vorwort</A>
      <DT><A HREF="book.01.html">Danksagung</A>
    </DL>
  </DL>
  <DT PAGE=1><H3>Einf&uuml;hrung</H3>
  <DL>
    <DT><H3>Einleitung und Motivation</H3>
    <DL>
      <DT TEMPLATE="TemplateChapter1.html"><A HREF="chapter1/book.1.html">Was sind Neuronale Netze?</A>
      <DT><A HREF="chapter1/book.2.html">Geschwindigkeitsvergleich</A>
      <DT><A HREF="chapter1/book.3.html">100-Schritt Regel</A>
      <DT END><A HREF="chapter1/book.7.html">Bemerkungen</A>
    </DL>
```

Fig. 4. An excerpt from a site description file.

Figure 4 shows the same site description file as figure 3, but this time in source code. The `VARIABLES` block defines which files are to be processed, which simply copied, with which page number and template to start and which directories to use. The meaning of all these variables is described in table 1.

Table 1. Control keywords for `VARIBALES` block in site description file

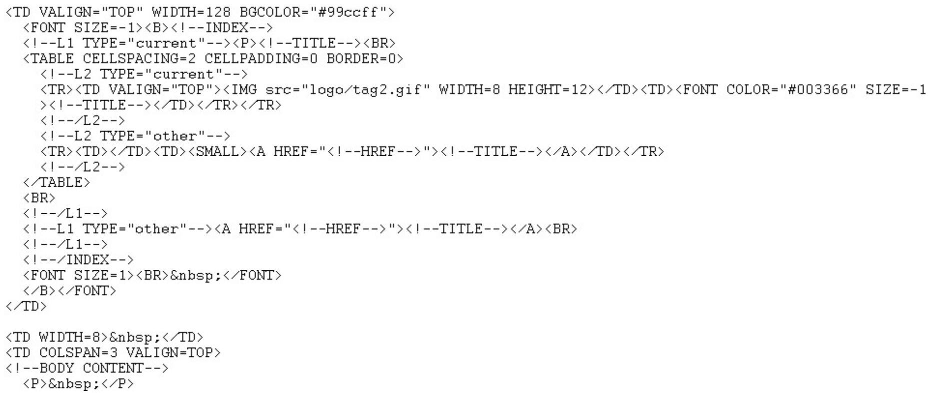
Variable	Meaning
`$SOURCE`	The directory containing source pages. The default value is `source`.
`$DEST`	The directory for processed pages. The default is `dest`. If source and destination are the same, the processing tool extracts original contents from the pages and reprocesses them.
`$FILTER`	A file filter (wildcards for listing files), defining which files other than those explicitly listed are to be processed. By default no other files are processed.
`$COPY_FILES`	Files to be automatically copied from source into destination, like images, Java classes etc. May include wildcards. Default: none.
`$TEMPLATE`	Template file to use for the pages. By default `Template.html`.
`$PAGE`	Starting value of the page counter. Default value is 1, but it can be set to any number, or to a letter, for alphabetical enumeration.

Further, the site structure is stated: the `<DT>` tags define nodes in the hierarchy given by nested `<DL>` tags. Some of these tags are used with extensions, whose meanings are described in table 2.

The files *TemplateStart.html* and *Template.html* are used as templates during processing: it starts with *TemplateStart.html* but switches later to *Template.html*. Figure 5 shows a small part of a template file which governs the appearance of the navigation bar. In this example, the whole navigation bar is packed in a table. The `<!--INDEX-->` tag in the second line starts the navigation bar description. The `<!--L1 TYPE="current"-->` defines first-level entries for index entries belonging to the same branch as current page: the name of the branch is displayed (`<!--TITLE-->` tag) and a new table started. Inside the table, the appearance of second-level entries is defined. The current entry appears with an icon left to it (`<IMG ...>` tag) and its name is shown in a different color. Other entries (`<!--L2 TYPE="other"-->`) are shown without an icon and in default color.

Table 2. Extensions to the `<DT>` tag

`<DT>` **Extension**	**Meaning**
`END`	Used in conjunction with `$FILTER` from the `VARIABLES` block. All files selected by the filter and which, alphabetically sorted, appear between currently processed file and the file given in the `END` variable, get processed and subsequently inserted at the current position in the site structure.
`TEMPLATE="..."`	Switches to another template file for processing current and subsequent source files.
`PAGE=...`	Sets the page counter to the given value.



```
<TD VALIGN="TOP" WIDTH=128 BGCOLOR="#99ccff">
  <FONT SIZE=-1><B><!--INDEX-->
  <!--L1 TYPE="current"--><P><!--TITLE--><BR>
  <TABLE CELLSPACING=2 CELLPADDING=0 BORDER=0>
    <!--L2 TYPE="current"-->
    <TR><TD VALIGN="TOP"><IMG src="logo/tag2.gif" WIDTH=8 HEIGHT=12></TD><TD><FONT COLOR="#003366" SIZE=-1
    ><!--TITLE--></TD></TR></TR>
    <!--/L2-->
    <!--L2 TYPE="other"-->
    <TR><TD></TD><TD><SMALL><A HREF="<!--HREF-->"><!--TITLE--></A></TD></TR>
    <!--/L2-->
  </TABLE>
  <BR>
  <!--/L1-->
  <!--L1 TYPE="other"--><A HREF="<!--HREF-->"><!--TITLE--></A><BR>
  <!--/L1-->
  <!--/INDEX-->
  <FONT SIZE=1><BR>&nbsp;</FONT>
  </B></FONT>
</TD>

<TD WIDTH=8>&nbsp;</TD>
<TD COLSPAN=3 VALIGN=TOP>
<!--BODY CONTENT-->
  <P>&nbsp;</P>
```

Fig. 5. An excerpt from a template file.

After the closing tag for current branch first-level entries (`<!--/L1-->`), the appearance of other first-level entries is defined. They simply get their name shown, but their subordinated branches are not shown. The `<!--/INDEX-->` tag closes the definition of the navigation bar. Finally, the body of the source file is inserted in place of the `<!--BODY CONTENT-->` tag. Table 3 summarizes the HTML extensions used in template files.

In place of `HEAD CONTENT` and `BODY CONTENT` in a template file, the HTML code from `<HEAD>` and `<BODY>` parts of source files are inserted. Further, extensions like `PREV` and `UP` are available for generating linear navigation in the site.

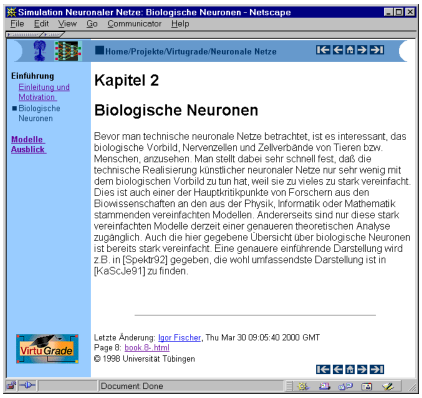
The `<INDEX>` block is used for generating flexible and powerful navigation bars. For example, if current page or branch should be highlighted in the navigation bar, then the HTML code inside the Ln block marked as "CURRENT" should differ from the code in the block marked as "OTHER". Also, if different levels are to appear in different colors in the navigation bar, the code in different `L`n tags has to control it.

A page from a toy example site, generated using the above described template and site description file, is shown in figure 6. This page has a standard banner at the top, with buttons for linear navigation on its right edge. The left side of the page contains a navigation bar, with the current page being marked with a small quadratic icon. At the bottom, the page contains a logo, some textual information about the page and the author and again buttons for linear navigation. All these elements are defined in the template file. The information content of the page, as defined in a source file, appears in the large right part of the screen.

As a real world example of sites managed by the tool, a page from our department's Web site [9] is shown in figure 7. Like the toy example above, it also contains the top banner with some information and navigation elements, and a navigation bar on the left, with highlighted current page. This is today a very frequent appearance on the WWW and can almost be considered a standard.

Table 3. HTML extensions for template files

Extension	**Meaning**
`HEAD CONTENT`	The contents of current source file's `<HEAD>` block.
`BODY CONTENT`	The contents of current source file's `<BODY>` block.
`DATE`	Current date, useful as the last modification date.
`PREV`	File name of a previous page in the site structure. If no `LEVEL` is specified, it refers to the previous page regardless of hierarchy level. Else, it referes the previous page at the given hierarchical level.
`NEXT`	File name of the next page in the site structure. Considerations similar to `PREV` hold.
`LEVEL`	Level specifier for `PREV` and `NEXT`.
`UP`	File name of the parent page in the site structure, i.e. one hierarchy level higher.
`DOWN`	File name of the next page in the hierarchically higher level, i.e. the next branch in the hierarchy.
`TITLE`	This variable can have different meanings, depending on where it appears: – inside the `<!--INDEX-->` ... `<!--/INDEX-->` block, it is the name of the index point – inside the `<TITLE>` ... `</TITLE>`, it is the title of the currently processed page
`HREF`	This variable also has different meanings, depending on where it appears: – inside the `<!--INDEX-->` ... `<!--/INDEX-->` block, it is the relative URL of the page assigned to the index point – outside the `<!--INDEX-->` ... `<!--/INDEX-->` block, it is the URL of the current page
`PAGE`	Current value of page counter (page number).
`INDEX`	Start of an index block. Index block contains complete or partial site index and is useful in navigation bars.
`/INDEX`	End of an index block.
`L`*n*	Start of *n*-th level in an index, *n* being a number between 1 and 6. Additionally, a type should be specified: – `TYPE="CURRENT"`: marks block to be used for processing index points referring to the current page or pages on other hierarchy levels belonging to the same branch. – `TYPE="OTHER"`: opposite of current: marks block to be used for processing index points referring to the pages that are neither current, nor belonging to the same branch on another level. – `TYPE="ALL"`: means CURRENT or OTHER: marks block to be used for processing all index points, regardless if they belong to the same branch or not.
`/L`*n*	End of *n*-th level in an index.


Simulation Neuronaler Netze: Biologische Neuronen - Netscape
File Edit View Go Communicator Help
Home/Projekte/Virtugrade/Neuronale Netze
Einführung
Einleitung und Motivation
Biologische Neuronen
Modelle
Ausblick
VirtuGrade
Document: Done

Kapitel 2

Biologische Neuronen

Bevor man technische neuronale Netze betrachtet, ist es interessant, das biologische Vorbild, Nervenzellen und Zellverbände von Tieren bzw. Menschen, anzusehen. Man stellt dabei sehr schnell fest, daß die technische Realisierung künstlicher neuronaler Netze nur sehr wenig mit dem biologischen Vorbild zu tun hat, weil sie zu vieles zu stark vereinfacht. Dies ist auch einer der Hauptkritikpunkte von Forschern aus den Biowissenschaften an den aus der Physik, Informatik oder Mathematik stammenden vereinfachten Modellen. Andererseits sind nur diese stark vereinfachten Modelle derzeit einer genaueren theoretischen Analyse zugänglich. Auch die hier gegebene Übersicht über biologische Neuronen ist bereits stark vereinfacht. Eine genauere einführende Darstellung wird z.B. in [Spektr92] gegeben, die wohl umfassendste Darstellung ist in [KaScJe91] zu finden.

Letzte Änderung: Igor Fischer, Thu Mar 30 09:05:40 2000 GMT
Page 8: book.8-.html
© 1998 Universität Tübingen

Fig. 6. Example page generated by the site management tool.

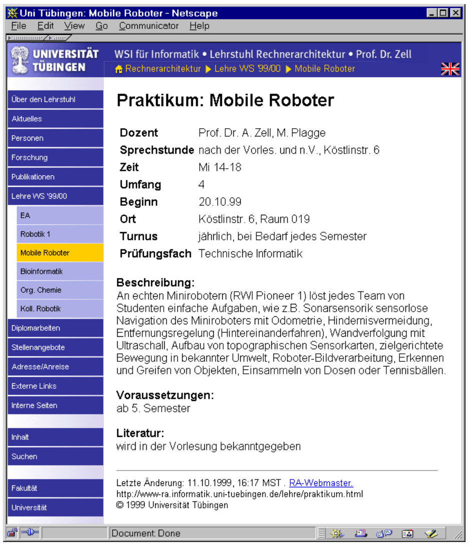
The index appears twice in this page: in a single line in the top banner, with only current branch for each level displayed, and in the left navigation bar, with current level expanded and shown with all branches. Contrary to the toy example before, which represents a small Web course book with a preferred order (a learning trajectory) of traversing it, the department site is a more free structure and does not provide for a linear (forward/backward) navigation, but only for a hierarchical one. It also includes a search function, which is accessed on a separate search page through the link "*Suchen*" in the lower part of the navigation bar. The contents of the department site changes almost daily, with different authors being responsible for different pages. Therefore, to facilitate the maintenance, only the published version of the site exists and all changes are made directly on it. Only for the case when pages are added or removed, the webmaster updates the site description file and "recycles" the site, as illustrated on figure 2 above.

6 Discussion

Effective Web site management can be done with relatively simple tools and very limited resources. In this paper, we have described a site management tool implemented in Perl, suitable for managing small and medium-sized sites.

Most commercially available site management tools rely on a database or some proprietary format for storing a site before publishing. This approach is systematical and allows management of very big sites, including dynamic generation of pages or even whole sites. The main drawback lies in its complexity: skilled personnel and maybe expensive hardware and software is needed, which can be behind the reach of smaller organizations.

The other extreme, manual site management, is impractical and can be performed only for very small sites. It is labor-intensive, error-prone and easily be-



Praktikum: Mobile Roboter

Dozent	Prof. Dr. A. Zell, M. Plagge
Sprechstunde	nach der Vorles. und n.V., Köstlinstr. 6
Zeit	Mi 14-18
Umfang	4
Beginn	20.10.99
Ort	Köstlinstr. 6, Raum 019
Turnus	jährlich, bei Bedarf jedes Semester
Prüfungsfach	Technische Informatik

Beschreibung:
An echten Minirobotern (RWI Pioneer 1) löst jedes Team von Studenten einfache Aufgaben, wie z.B. Sonarsensorik sensorlose Navigation des Miniroboters mit Odometrie, Hindernisvermeidung, Entfernungsregelung (Hintereinanderfahren), Wandverfolgung mit Ultraschall, Aufbau von topographischen Sensorkarten, zielgerichtete Bewegung in bekannter Umwelt, Roboter-Bildverarbeitung, Erkennen und Greifen von Objekten, Einsammeln von Dosen oder Tennisbällen.

Voraussetzungen:
ab 5. Semester

Literatur:
wird in der Vorlesung bekanntgegeben

Letzte Änderung: 11.10.1999, 16:17 MST . RA-Webmaster.
http://www-ra.informatik.uni-tuebingen.de/lehre/praktikum.html
© 1999 Universität Tübingen

Fig. 7. A page from the department for computer architecture's web site.

comes very expensive as site grows. Additionally, it is hard to enforce a common appearance ("corporate identity") on pages.

The tool described in this article aims to be simple enough to be used by anybody familiar with HTML, but still capable of managing sites containing hundreds of pages. It uses HTML for all its inputs (contents, templates and site description) with minor extensions, which are hidden in HTML comments and serve as control elements for page processing. As of software, it needs only a Perl interpreter and works independently of a Web server or any other back-end process, like database or search engine. That makes it a kind of "plug-and-play" program, which is not the case for more complex and sophisticated solutions.

The tool's capabilities have been shown in managing different Web sites, from different department and faculty sites at our university, to various sites in *VirtuGrade* and *Bioinform@tik* projects.

A drawback of the tool is that it can generate only static sites, i.e. sites where structure remains the same for all visitors, regardless when they request a page. Although it can be an important issue in some cases, for the users in our scope (small and mid-sized organizations), it is seldom the case.

It can also be argued that HTML is not the ideal language for describing a site structure, or even contents. Whereas there certainly is something to this argument – otherwise we would not experience continuous improvement of existing standards and establishment of new ones, like XML [10] for general data description, or WebDAV [11] for content management – HTML is for the time being still the most common format on the Internet, simple yet powerful. In authors' opinion, this balance of simplicity and capability has been one of the crucial reasons for the success of the WWW. Although the tool would perhaps benefit from a better format for site and contents description, it is likely that it would discourage many of its users. To that concern adds the fact that new standards, like XML and XLST, are still under development and sparsely supported by simple and powerful tools. And, even if the time should render the described tool obsolete, we believe that the underlying ideas and principles regarding site management will still remain valid.

Acknowledgments. The authors wish to thank Uwe Oestermeier, the originator of the concept for site management, who kindly provided his own tool as a sample. Many thanks go to the authors' colleagues Simon Wiest and Clemens Jürgens, who further extended the described tool and supported the author with new ideas. The *VirtuGrade* project is funded by the state of Baden-Württemberg and the *Bioinform@tik* project by the state of Baden-Württemberg and the Deutsche Telekom AG.

References

1. Microsoft Corporation, *Erste Schritte mit Microsoft FrontPage 98*, 1998.
2. Microsoft Corporation, "Visual interdev web solutions kit," http://msdn.microsoft.com/vinterdev/wsk/default.asp, 2000.
3. NetObjects, *NetObjects Fusion 5.0*, ftp://ftp.netobjects.com/pub/products/documentation/nf5_docs/NOF5manual.pdf, 1999.
4. B. Steppan, "World-Wide-WYSIWYG," *iX*, vol. 6, pp. 40–49, 1999.
5. Symantec Corporation., *Symantec Visual Page User's Guide*, 1998.
6. UserLand Software, "Frontier news," http://frontier.userland.com/news/, 2000.
7. Douglas B. O'Morain, *Site Manager User's Guide*, Silicon Graphics, Inc., http://techpubs.sgi.com/library/manuals/3000/007-3320-002/pdf/007-3320-002.pdf, 1998.
8. Interwoven, Inc., "Interwoven home page," http://www.interwoven.com, 2000.
9. Simon Wiest and Michael Plagge, "Praktikum: Mobile Roboter," http://www-ra.informatik.uni-tuebingen.de/lehre/praktikum.html, 1999.
10. T. Bray, J. Paoli, and C. M. Sperberg-McQuee, *Extensible Markup Language (XML) 1.0*, W^{3}C Consortium, http://www.w3.org/TR/1998/REC-xml-19980210.html, Feb. 1998.
11. David Sussman, "WebDAV: A panacea for collaborative authoring?," *IEEE MultiMedia*, vol. 6, no. 2, pp. 76–79, 1999.

Structuring Access to a Dynamic Collection of Digital Documents: The Walden's Paths Virtual Directories

Unmil P. Karadkar, Luis Francisco-Revilla, Richard Furuta, and Frank M. Shipman III

Center for the Study of Digital Libraries and Dept. of Computer Science
Texas A&M University
College Station, TX 77843-3112
{unmil, l0f0954, furuta, shipman}@csdl.tamu.edu

Abstract. The Walden's Paths project facilitates incorporation of Web-based documents into the K-12 classroom environment. Currently Walden's Paths uses a static presentation mechanism, based on the authors of the paths, to display the available paths to the readers. As the number of authors and readers increases, it becomes increasingly difficult for readers to find the paths from a list that is based solely on the authors. The most suitable organization of paths varies with the task at hand and the reader's environment. The use of virtual directories has been proposed for managing dynamic collections of digital documents. These directories are not physically present in the file system; only a user query is stored. At access time, the database of files is queried to select files of interest and these are included in the directory. This mechanism allows readers to organize paths according to their needs while maintaining a hierarchy and preserving the context. This paper proposes that inclusion of the virtual directory mechanism in Walden's Paths will enable the readers to organize data to better suit their conceptual model and presents a working prototype of this feature.

1 Introduction

Traditional file systems provide access to and help organize collections of documents. The structure of most file systems looks identical to all those who access it. These file systems do not take into account the difference in users' conceptual models and needs. Users can access documents in these file systems by referring to them only by their location in the document hierarchy. Most file systems are created and managed by trained system administrators, who have a good sense of user expectations from the file system and tailor their systems to obtain optimal performance and to satisfy user requirements. These file systems have well formed document hierarchies that are either intuitive to users in a certain domain or can be remembered, due to frequent use over time. However, these file systems do not scale well from the user perspective.

This material is based upon work supported by the National Science Foundation under Grant Nos. IIS-9812040 and DUE-0085798.

While small collections of documents are manageable, users find it difficult to remember long paths to documents that are stored in intricate hierarchies [5].

This scenario seems to be changing with the emergence of the World-Wide Web (henceforth referred to as the Web or WWW), which may be viewed as a distributed hierarchical file system. While individual Web sites still follow the traditional model of file systems, the Web, as a whole, has no discernable structure that is either intuitive or documented. Also, it is unlikely that any two Web sites have identical structure. Hence, while most Web sites are managed by knowledgeable system administrators, a casual reader can easily get the impression of chaos as the lessons learnt on any Web site do not apply to another. It is not practical to expect readers to know or learn the entire document hierarchy of the Web due to the sheer volume of data and the number of Web sites involved. Typical users tend to browse many Web sites, but visit only a few of them frequently. Users perceive the Web sites they visit often to be organized, while the ones they seldom visit to be idiosyncratic. The Web also differs from the traditional file systems as it imposes a clear distinction between the authors and readers of documents. While authors of files in traditional file systems can grant modification privileges to readers, the Web does not permit modification of documents by readers. However, some Web services do allow readers to personalize and tailor the presentation to their liking and mental model.

Mostly, users organize their files based on their functional requirements. However, the organization varies between users, and for a given user, between tasks. Users tend to reorganize their files based on the current task at hand. File systems that provide a single view do little to help users reorganize files dynamically, without changing the physical hierarchy of the file system. Removing hierarchy from file systems is not a solution as hierarchy provides a context to the collection of documents. Search results returned by commercial Web search engines is an example of a non-hierarchical collection of documents. These collections are built dynamically but they lack context; an essential classifier for documents.

Users often share documents by passing document locations between them directly or indirectly. Sharing a document by passing its location via e-mail is an example of direct communication, where both readers must know the document location in order to access it. A reader may also create links to the documents of interest and allow other readers to access them via the link. The readers may not know the actual document location in this case, resulting in an indirect access to the document. Readers also share documents by creating copies in their personal space and allowing other readers to access them. With respect to the original document, this is another method of indirect sharing. The documents may be shared privately or publicly. Documents can be shared privately via email to a clique or via direct contact. Documents may be shared publicly by publishing pointers at well-known locations for public viewing.

The issues involved in presentation and sharing of documents in a file system are further compounded when the documents and the file system change often. Systems that provide access to dynamic documents must address the issues mentioned above in addition to reflecting the changes in the document collection as they happen. These systems should display only the documents that are available at the time of access.

Thus, there is a need for systems that provide context-based display and sharing of documents without requiring the users to modify the physical file hierarchy. This helps users build contexts dynamically on a collection of documents. Thus various users may model a given document collection differently for various purposes to suit their individual needs and preferences. In this paper we present the Walden's Paths virtual directories, a mechanism to address some of the issues outlined earlier, in the context of a Web-based collection of meta-documents, for use in K-12 schools.

We present a survey of the earlier research regarding virtual directories in section 2. Sections 3 and 4 provide a brief overview of the Walden's Paths path server. Section 5 describes the features provided by Walden's Paths virtual directories. Section 6 is a discussion of the features and comparison with a leading Web portal. Section 7 outlines the directions for the future and section 8 concludes the paper.

2 Virtual Directories

Managing and using large file systems is a daunting task. Users find it difficult to remember exact path names and may often confuse similar path names. Over the years, attempts have been made to provide content-based access to file systems in order to help users find the right information in a short time without having to remember elaborate file locations.

Semantic File System (SFS) [4] was one of the early efforts in this direction. SFS provides access to file systems through queries. Each query is represented as a *virtual directory*, which points to a set of files that satisfy the query. The system provides associative access to data by extracting attributes from files via the use of file type-specific transducers. Transducers are filters that accept files as input and return the files entities and their corresponding attribute values. A more recent implementation of a semantic access system provides simultaneous access using the hierarchical file-structure as well as a content-based access. This system is called HAC (Hierarchy And Content) [5]. HAC treats queries as files or directories and calls these *semantic directories*. While SFS attempts to treat queries as files, HAC extends a hierarchical file system to support queries. The Essence system [7] associates wrapper objects with ordinary files in order to support a unified data extraction mechanism over multiple file types. The Essence system also incorporates an understanding of Unix file semantics and context to generate representative summaries of large amounts of data to aid faster resource discovery in large data collections.

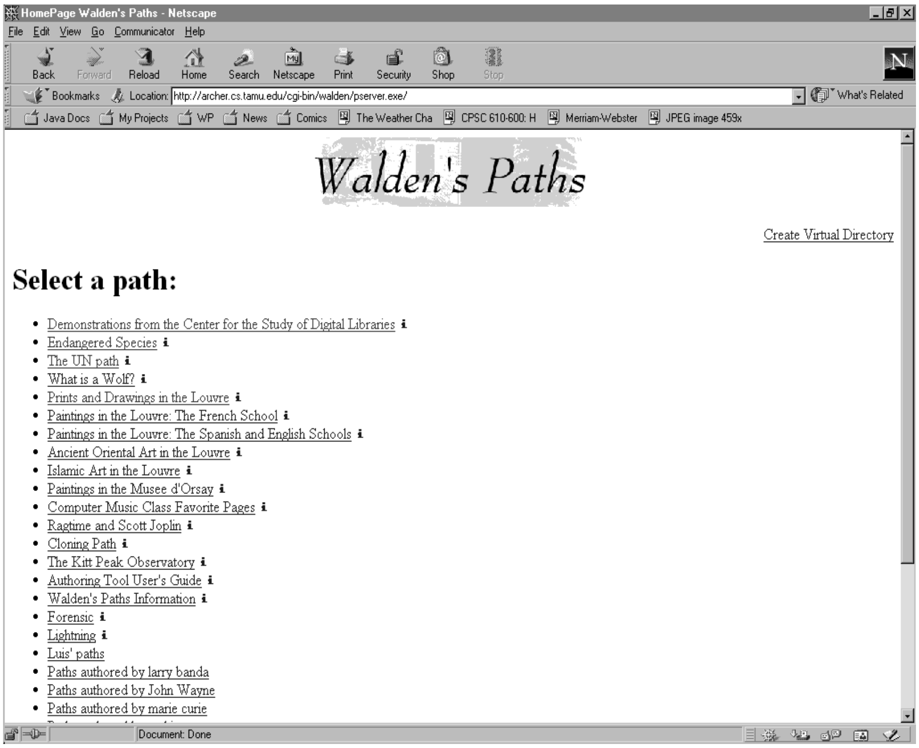
3 Walden's Paths

Vannevar Bush, in 1945, proposed hypertext paths as a means to associate two items that were conceptually related, but placed physically apart in an information web [1].

He referred to them as trails. Paths have subsequently been incorporated into many hypertext systems, most notably in NoteCards [6] as Guided Tours and TableTops [10] and in Scripted Paths [12].

Walden's Paths is a Web-based implementation of the hypertext path mechanism. Paths lay a meta-structure over the Web. The paths refer to and elucidate documents on the Web and hence can be termed as meta-documents. The Walden's Paths project aims at supporting the incorporation of Web-based documents into K-12 classrooms in order to help teachers achieve their curricular goals. Most of the material available on the WWW is not oriented towards young audience and is tailored to suit casual readers. Hence, teachers who author the paths may add HTML annotations to the paths and to individual pages on the paths to contextualize the information for easier comprehension by the students. A detailed description of various issues and experiences with Walden's Paths may be found in [2, 8 and 9].

Readers can access the paths from any standard Web browser that supports frames. Typically, readers begin a Walden's Paths session by selecting a path from the list of paths (shown in figure 1) that is displayed at startup. Figure 2 illustrates the Walden's Paths interface when viewing a path. The browser window contains three frames. The bottom frame is called the "Content Frame". It displays the Web-based document that the page refers to, as it would appear if viewed in the browser without the Walden's


HomePage Walden's Paths - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://archer.cs.tamu.edu/cgi-bin/walden/pserver.exe/ What's Related
Java Docs My Projects WP News Comics The Weather Cha CPSC 610-600: H Merriam-Webster JPEG image 459x
Walden's Paths
Document: Done

Create Virtual Directory

Select a path:

- Demonstrations from the Center for the Study of Digital Libraries i
- Endangered Species i
- The UN path i
- What is a Wolf? i
- Prints and Drawings in the Louvre i
- Paintings in the Louvre: The French School i
- Paintings in the Louvre: The Spanish and English Schools i
- Ancient Oriental Art in the Louvre i
- Islamic Art in the Louvre i
- Paintings in the Musee d'Orsay i
- Computer Music Class Favorite Pages i
- Ragtime and Scott Joplin i
- Cloning Path i
- The Kitt Peak Observatory i
- Authoring Tool User's Guide i
- Walden's Paths Information i
- Forensic i
- Lightning i
- Luis' paths
- Paths authored by larry banda
- Paths authored by John Wayne
- Paths authored by marie curie

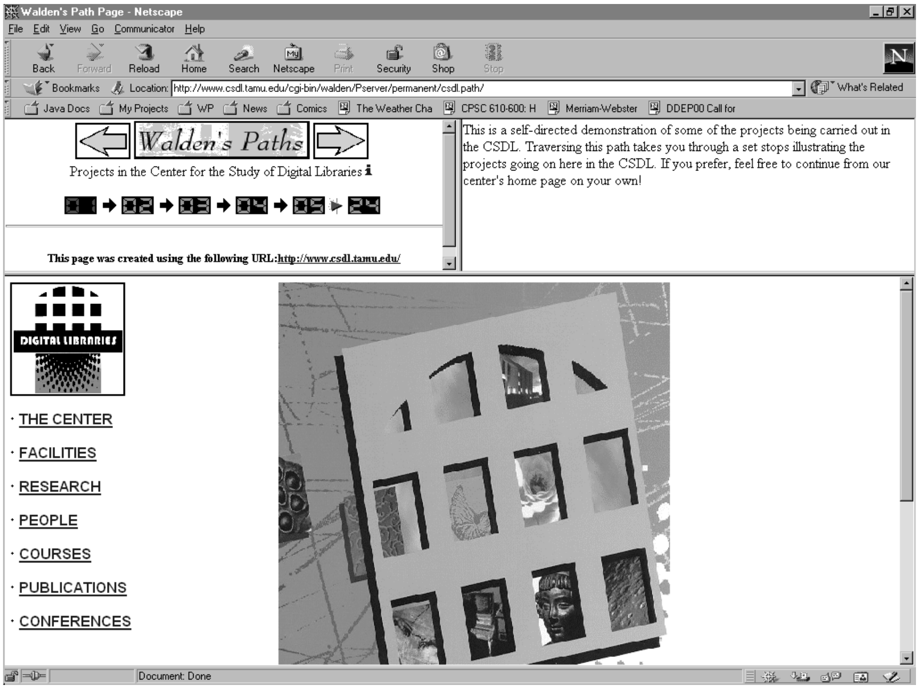
Fig. 1. The Walden's Paths Start Page

Paths interface. The top-left frame, called the "Control Frame", displays widgets for navigating alone the path. The reader may navigate along the path by clicking on the left right arrows to move along the path. The reader may also click on any of the numbered images to view that page on the path. Clicking on the image labeled as "Walden's Paths" takes the reader back to the start page. This frame displays the Web-location of the document displayed in the bottom frame. Selecting the image labeled with the letter "i" brings up an overview of the path via an information icon denoted. The top-right frame, called the "Annotation Frame", contains the additional information added by the creator of the path for the page displayed below. The annotation may contextualize, explain or analyze the information in the Content Frame. It may direct users to focus on certain issues or aspects of the information. It may point the reader towards related interesting issues or pose questions with respect to the document displayed below [9].

Along with the annotations and WWW document references, the paths also store metadata associated with the path. This information includes the name and contact information of the authors, a brief abstract and the dates of creation and expiration for the path.

4 Ephemeral Paths and Author Directories

The Walden's Paths system permits readers to customize the authored paths [3]. Readers can create a new path by selecting a subset of pages from an existing path.


Walden's Path Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location:
What's Related
Java Docs My Projects WP News Comics The Weather Cha CPSC 610-600: H Merriam-Webster DDEP00 Call for
Walden's Paths
DIGITAL LIBRARIES
· THE CENTER
· FACILITIES
· RESEARCH
· PEOPLE
· COURSES
· PUBLICATIONS
· CONFERENCES
Document: Done

http://www.csdl.tamu.edu/cgi-bin/walden/Pserver/permanent/csdl.path/

Projects in the Center for the Study of Digital Libraries i

01 → 02 → 03 → 04 → 05 ▸ 24

This page was created using the following URL:http://www.csdl.tamu.edu/

This is a self-directed demonstration of some of the projects being carried out in the CSDL. Traversing this path takes you through a set stops illustrating the projects going on here in the CSDL. If you prefer, feel free to continue from our center's home page on your own!

Fig. 2. The Walden's Paths Interface

The paths thus created are stored in a special directory and are accessible only via a handle that the system provides when creating them. As these paths are expected to have a short life span, they are termed as "ephemeral" paths.

The Walden's Paths system allows registered authors to compile and publish paths. The published paths are accessible to all readers of the system. These paths are saved in the authors' respective directories. An author may choose to provide a link to his home directory to help users easily locate his paths. The main list of paths displays links to paths as well as to author directories. In figure 1, the links that lead to author directories are indicated visually by the absence of the information icon, as well as verbally. The paths in these directories are displayed in an interface identical to figure 1 when a reader selects to view them.

While creation of ephemeral paths allows readers to customize the individual paths, the author directories impose a hierarchical structure on the paths in the system. Creation of virtual directories allows readers to create customized directories of paths based on reader-specified attribute values.

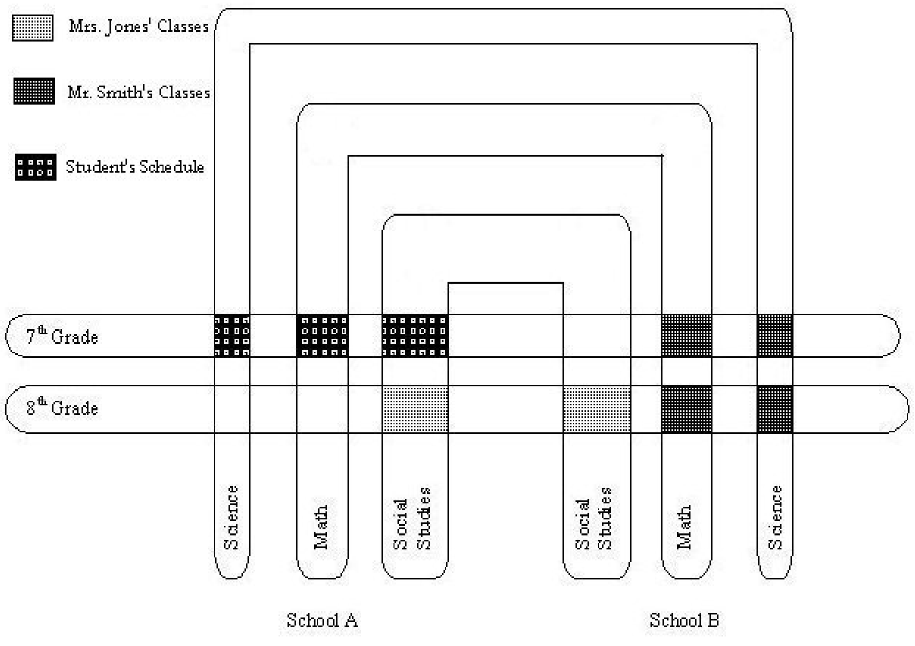
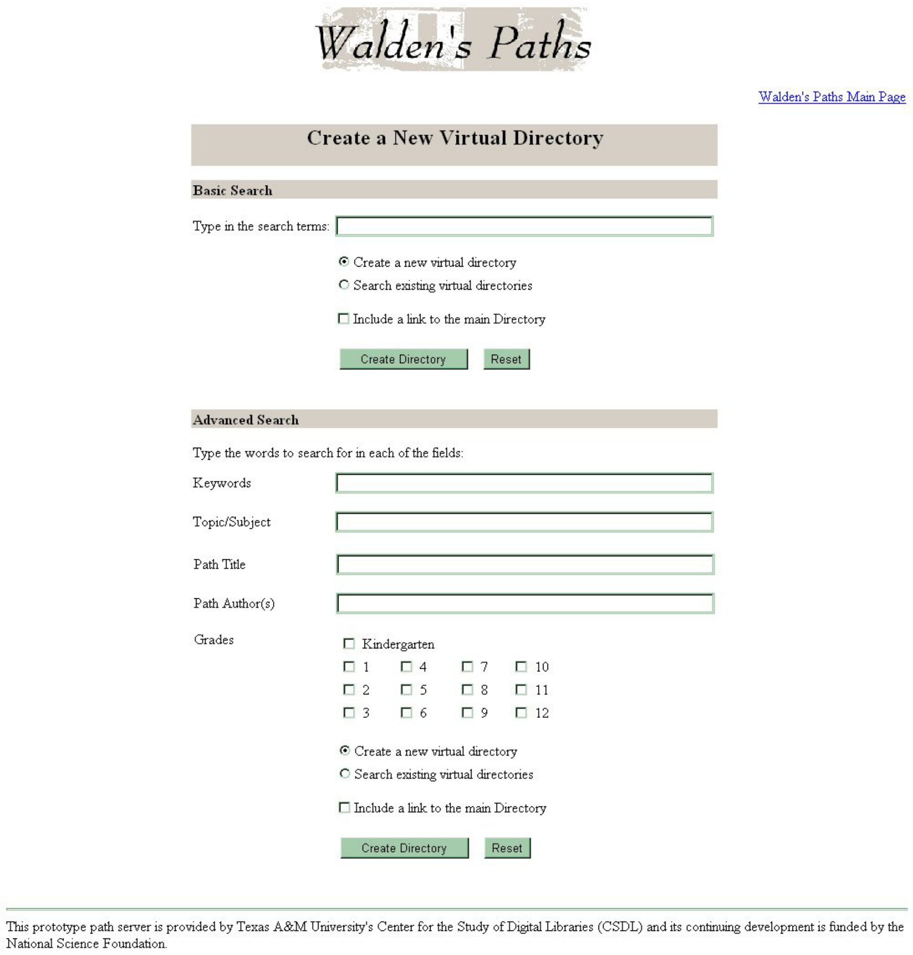


Fig. 3. Overlapping Domains

5 Virtual Directories in Walden's Paths

As the number of published paths increases, organization of paths based solely on their authors may be too constraining to be meaningful to the readers. Also, readers may like to organize the available paths along various dimensions to best suit their goals. It is possible that teachers from two or more schools may share a path server to reuse resources and/or to split the costs. In such cases, the paths authored by teachers in one school may be available to teachers in the other schools if the authors choose to do so. The domains are less clearly defined and may easily overlap across administrative boundaries. Some sample domains are shown in figure 3. Mrs. Jones travels between schools A and B and teaches Social Studies to the 7th grade in both schools. A search on the paths authored by her yields paths from both the schools. Mr. Smith teaches Math and Science to students in the 7th and 8th grades in school B. A search for paths authored by him returns paths for both the grades and subjects that he


Walden's Paths

Walden's Paths Main Page

Create a New Virtual Directory

Basic Search

Type in the search terms:

[✓ Create a new virtual directory]
Search existing virtual directories

☐ Include a link to the main Directory

Create Directory | Reset

Advanced Search

Type the words to search for in each of the fields:

Keywords

Topic/Subject

Path Title

Path Author(s)

Grades

☐ Kindergarten

☐ 1	☐ 4	☐ 7	☐ 10
☐ 2	☐ 5	☐ 8	☐ 11
☐ 3	☐ 6	☐ 9	☐ 12

[✓ Create a new virtual directory]
Search existing virtual directories

☐ Include a link to the main Directory

Create Directory | Reset

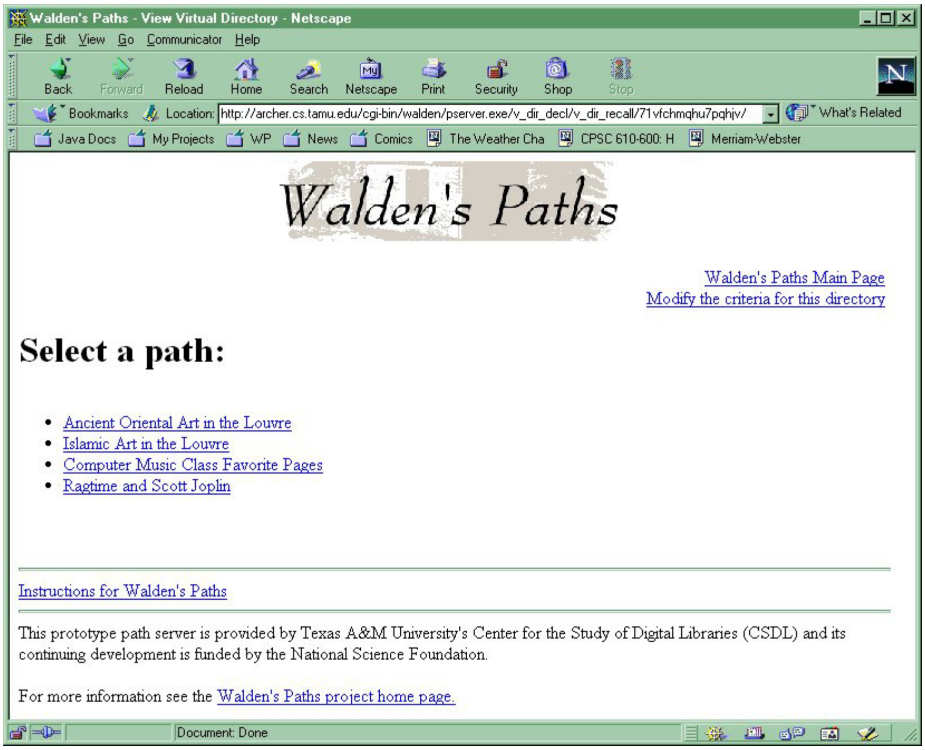
This prototype path server is provided by Texas A&M University's Center for the Study of Digital Libraries (CSDL) and its continuing development is funded by the National Science Foundation.

Fig. 4. Virtual Directory Creation Interface

teaches. A student schedule for a 7th grader in school A is also shown. In the scenario illustrated by figure 3, no fixed hierarchy of domains (and directories corresponding to these) will yield an arrangement of paths that is suitable for all the students or teachers. It is in this case of overlapping domains and unclear boundaries, that the virtual directories can be exploited to their full potential. The teachers as well as the students can create virtual directories over the physical path structure and easily access paths of their respective interests. Also, the teachers may create virtual directories and provide these to the students to protect them from a deluge of mostly irrelevant paths.

Possible search attributes for paths could be the grade levels, keywords, subjects, authors and date of creation. To support searches on these attributes, the existing path structure was augmented to include additional metadata, for example the grade levels that the path is suitable for, and lists of keywords associated with each of the pages, as well as with the path as a whole are also stored. The lists of keywords contain related words that do not appear in the annotations or anywhere else in the path.

The creation of virtual directories allows readers to view lists of paths that match certain criteria. It enables readers to search paths that have specific attribute values


Walden's Paths - View Virtual Directory - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://archer.cs.tamu.edu/cgi-bin/walden/pserver.exe/v_dir_decl/v_dir_recall/71vfchmqhu7pqhjv/
What's Related
Java Docs My Projects WP News Comics The Weather Cha CPSC 610-600: H Merriam-Webster
Walden's Paths
Document: Done

Walden's Paths Main Page
Modify the criteria for this directory

Select a path:

- Ancient Oriental Art in the Louvre
- Islamic Art in the Louvre
- Computer Music Class Favorite Pages
- Ragtime and Scott Joplin

Instructions for Walden's Paths

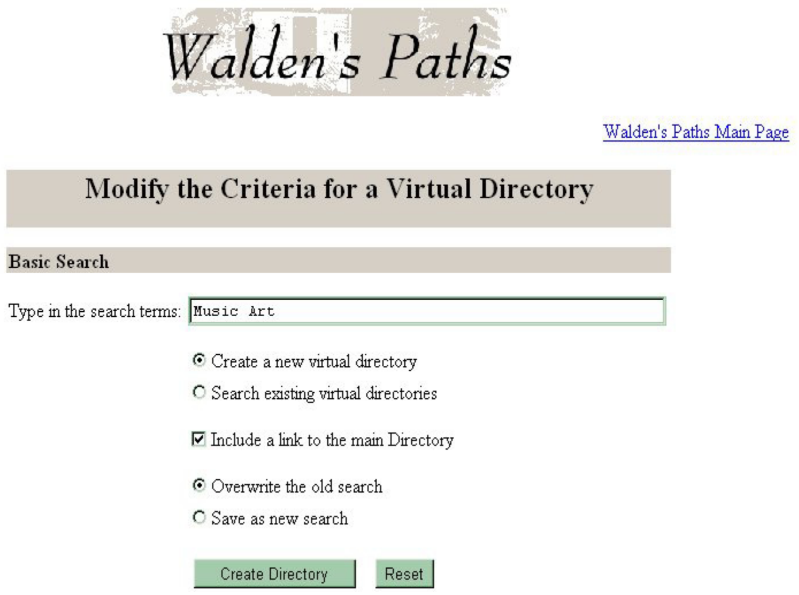
This prototype path server is provided by Texas A&M University's Center for the Study of Digital Libraries (CSDL) and its continuing development is funded by the National Science Foundation.

For more information see the Walden's Paths project home page.

Fig. 5. View of a Virtual Directory

and view them as one list. These information structures are analogous to directories that contain related files in the traditional file systems. However, these structures do not exist on the server physically as files or directories, but are dynamically generated from their criteria every time they are accessed. We call these structures as virtual directories. The virtual directories are physically stored as the parameters provided by the readers of paths as attribute-value pairs in plain text files. A virtual directory returns a URL to the reader upon its creation. This URL acts as a handle for future access to the directory. Readers can modify virtual directories by changing the search criteria associated with them. The readers may save the modified criteria as a new directory, or in the current directory, overwriting the existing criteria. By their very nature, the virtual directories display only the paths available in the system at the time of access. When the reader accesses a virtual directory, the system searches for paths that match the criteria for the virtual directory and display these paths to the reader. Visually, they have an interface identical to the one shown in figure 1.

The virtual directory creation interface is shown in figure 4. The interface supports two search modes, basic and advanced. In the basic mode, the reader only needs to type in the list of keywords to search on. These keywords are matched against all attributes in the paths and a directory of all the paths that contain the keyword is generated and displayed. In the advanced mode, the reader has a finer control on the keywords and can specify the path attributes where these keywords must appear. In this case, the returned directory lists paths that contain the keywords only in the specified attributes. The interface also allows readers to create a link back to the main



Modify the Criteria for a Virtual Directory

Basic Search

Type in the search terms: Music Art

[✓ Create a new virtual directory]
Search existing virtual directories

[✓ Include a link to the main Directory]

[✓ Overwrite the old search]
Save as new search

Fig. 6. Virtual Directory Modification Interface

page of the Walden's Paths server (displayed in figure 1).
Figure 5 displays a virtual directory of all paths that contain either of the terms "Music" or "Art". It also contains a link to the Walden's Paths server main page. The access mechanism for paths in virtual directories is identical to that for the paths that are accessible from the main page of the Walden's Paths server. When displaying a virtual directory, the location bar contains the URL that acts as an access handle for future references. A reader may bookmark this URL and return to it at will, a standard functionality provided by most Web browsers. The invocation of this URL causes the contents of this directory to be rebuilt and presents the reader with all the paths that match the specified criteria.

Readers may modify the virtual directory by following the link that is displayed in the top right corner of the browser window. Editing the search criteria used in its creation results in a different list of paths, thus modifying the virtual directory. The interface for modifying a virtual directory, displayed in figure 6, is similar to that for creation of new virtual directories. When the reader follows a link to modify the virtual directory, the current parameters of the virtual directory are displayed to assist the modification process. The modification interface allows readers to either overwrite the existing criteria for the virtual directory or create a new virtual directory with the changed criteria. This option is added only to the current search mode. Thus, for the case shown in figure 5, the option to overwrite or create a new directory is not added to the advanced search interface. Conversely, if the reader chose to modify a search that was created as an advanced search, while modifying it, this option would be added only to the advanced search and the interface for the basic search would remain as shown in figure 3.

The directory creation interface in figure 4 also allows readers to search for existing virtual directories that match the specified criteria. When the existing virtual directories are searched, no new directories are created. In this case the basic search mechanism returns a list of all existing virtual directories that were generated using any of the terms specified by the reader, while the advanced search returns a listing of all virtual directories which contain the specified search terms in the corresponding fields.

6 Discussion and Comparison

Yahoo! was the first Web portal that cataloged and categorized various Web sites. The Yahoo! repository spans over a vast range of subject areas. It is still a special case as it is one of the few portals that catalog all Web sites included in the repository manually [11]. Yahoo! provides search-based as well as navigable interfaces to the repository. The navigable interface is analogous to the hierarchical file systems, while the search interface is analogous to the virtual directory mechanism in Walden's Paths.

Yahoo! returns category matches and site matches in response to user queries. If we consider the list of returned results as a virtual directory, the category matches in these results can be considered to be subdirectories. Against this, the Walden's Paths virtual directories currently only support only single level structures. All the paths that match a reader-specified query are treated at par and returned as a list of paths. We do not make any attempt to further categorize the paths that match the search criteria.

The reuse of queries to recall an old search returns new results, based on the current contents of the repository. This is definitely an advantage while using the Walden's Paths virtual directories. Readers can recall old criteria to retrieve the current contents of the directories, thus eliminating the need to remember successful searches. Mostly, users are unable to reconstruct the exact queries that yielded the best results and hence are unable to retrieve any information that was found earlier. With the use of virtual directories, path readers can be certain that all the results that matched their query will be returned, as long as they exist (the author has not removed the path from the repository) and are relevant (the author has not modified the path).

Cataloging of data for personal use only need fit the mental model of a user and can easily be perceived by others as idiosyncratic. However, when data must be cataloged for general usage by millions of people, there needs to be a standardized categorization of concepts and a set of general decisions must be followed. However, even when rules are followed, some data fits into more than one slot in a hierarchy. Yahoo! implements this via the use of virtual links in the category hierarchy. Thus, the Yahoo! hierarchy is more controllable, and tailorable to various scenarios than most machine-generated hierarchies that depend on keywords for classification. Walden's Paths currently has no centralized scheme for categorization of paths. The onus for providing the keywords and subjects for the paths is entirely on the authors. In the absence of clear guidelines, or a designated person to catalog paths, there arises a possibility of conflict between the thoughts of different authors. This can result in the usage of different terms, leading to confusion regarding location of paths in a hierarchy.

7 Future Work

The current implementation of virtual directories is a satisfactory baseline. It must be further enhanced to provide a good utility value to the path readers and to provide better control to the teachers, path authors and path readers.

Currently, only registered authors can add paths to the Walden's Paths system. However, there is no control over the number of readers. Thus the number of virtual directories is expected to rise rapidly. The system must provide the creators of the virtual directories more control in order to protect their directories from access and from modification. Thus a reader can specify whether the directory should be returned as the result of a search over virtual directories. If the creator decides that the directory may not be listed in the results of searches over directories, it is effectively a private directory that can only be accessed via the handle returned upon its creation. The creator may allow others to view the directory but prevent anyone from modifying it. Thus, the directory created is immutable or, in more colloquial terms, a read-only directory. It must be noted that the creator cannot modify an immutable directory as the system is unaware of his identity. The Walden's Paths system must permit readers to create temporary directories. The readers may opt not to save a directory when it is created. This will enable readers to perform short-term searches and will help reduce load on the system. The directories that are saved, may also be subjected to certain timeouts, the limits of which can be set by the creator. Thus, directories can remain valid till certain dates or forever. This will further help clean up directories that are unlikely to be used in the long run.

Paths must be cataloged using a well-defined scheme. Path authors may still provide keywords and subjects for their paths. However, these will be treated as suggestions and be subject to review by designated path catalogers to ensure consistency.

The search interface must also allow authors to control their searches better. Currently the search returns all paths that match at least one of the search terms, that is, they perform a disjunctive search. Readers of the paths will require a greater control over defining their searches. The system must handle conjunctive searches, as also phrase and boolean searches. The interface must permit the readers to search based on the dates of creation of the paths. As an example, this feature will enable readers to search for all paths created since their last visit

Readers also need a facility to exclude certain paths from a search. This will help readers achieve better precision when the search is performed in the future. Realization of this feature requires that each path in the system have a unique identifier. The readers may then choose the paths that they would like to exclude from the directory when the search is performed in the future. This also highlights the need to be able to include discarded paths back into the search.

The URLs returned by the search have an identical caption. This could result in confusion if readers save multiple searches (as they probably will). A facility to clearly identify each directory must be provided. This feature may require additional work on the readers' part, but it must be performed in order to remove any ambiguities that may result from saving multiple directories in the bookmark list.

In the future, the readers may also be required to log in if they wish to have their searches stored on the server, instead of on their desktops. Some readers may be reluctant to log in as it involves an additional step in viewing the paths. Currently users are dependent on their desktops or browser installation to access their virtual directories where the handles are stored. This feature will permit readers to access their virtual directories even when they do not have access to their desktops. The login feature will also enable the system to store and recall user preferences. This feature will be considerably useful in the educational setting, where exams may be presented in the form of a path and the users are expected to take them online.

8 Conclusion

In this paper we have presented the Walden's Paths virtual directories. Readers no longer have to adapt to the single view of paths provided by the Walden's Paths path server. Virtual directories permit readers to reorganize the paths according to preferences, to suit their invidual needs and mental models without requiring modification of the underlying file structure. This feature enables users to search for and mark the paths of their interest from a large repository of paths. Readers can dynamically build contexts over the available paths and be sure that future accesses to these contexts will always return all the paths that they match. The virtual directories thus decouple the storage structure of the paths from the logical structures constructed by the readers. They also address privacy issues by allowing readers to retain control over sharing their contexts and authors over their paths. While the current implementation is a good baseline, it needs to be refined and enriched with more functionality before its true potential can be exploited.

References

1. Bush, V., "As We May Think", *The Atlantic Monthly*, August 1945, pp. 101–108.
2. Furuta, R., Shipman, F., Marshall, C., Brenner, D. and Hsieh, H., "Hypertext Paths and the World-Wide Web: Experiences with Walden's Paths", *Hypertext '97: the Eighth ACM Conference on Hypertext*, Southampton, U.K., April 1997, pp. 167–176.
3. Furuta, R., Shipman, F., Francisco-Revilla, L., Hsieh, H., Karadkar, U. and Hu, S., "Ephemeral Paths on the WWW: The Walden's Paths Lightweight Path Mechanism", WebNet (1) 1999, pp. 409–414.
4. Gifford, D., Jouvelot, P., Sheldon, P. and O'Toole, J., "Semantic File Systems", *Proceedings of the thirteenth ACM Symposium on Operating Systems Principles*, Pacific Grove, CA, October 1991, pp. 16–25.
5. Gopal, B. and Manber, U., "Integrating Content-based Access Mechanisms with Hierarchical File Systems", *Proceedings of the third symposium on Operating systems design and implementation*, New Orleans, LA, February 22-25, 1999, pp. 265–278.
6. Halasz, F., Moran, T. and Trigg, R., "Notecards in a Nutshell", *Proceedings of the ACM CHI+GI Conference 1987*, Toronto, Ontario, April 1987, pp. 45–52.
7. Hardy, D., and Schwartz, M., "Customized Information Extraction as a Basis for Resource Discovery", *ACM Transactions on Computer Systems*, 14(2), May 1996, pp. 171–199.
8. Shipman, F., Marshall, C., Furuta, R., Brenner, D., Hsieh, H. and Kumar, V., "Creating Educational Guided Paths over the World-Wide Web", *Educational Telecommunications, 1996: Proceedings of ED-TELECOM 96*, June 1996, pp. 326–331.
9. Shipman, F., Furuta, R., Brenner, D., Chung, C. and Hsieh, H., "Using Paths in the Classroom: Experiences and Adaptations", *Proceedings of Hypertext '98*, ACM, Pittsburgh, PA, June 1998, pp. 267–276.
10. Trigg, R., "Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment", *ACM Transactions on Office Information Systems*, 6(4), October 1988, pp. 398–414.
11. Yahoo! - Frequently Asked Questions, http://docs.yahoo.com/info/faq/faq.html (accessed April 2000).
12. Zellweger, P., "Scripted Documents: A Hypertext Path Mechanism", *Proceedings of Hypertext '89*, ACM, New York, November 1989, pp. 1–26.

An XML–Based Multimedia Document Processing Model for Content Adaptation

Lionel Villard, Cécile Roisin, and Nabil Layaïda

INRIA-Rhône-Alpes
{Lionel.Villard,Cecile.Roisin,Nabil.Layaida}@inrialpes.fr

Abstract. In this paper we present a general framework for document production that covers generic document model needs and adaptation needs. We define a multimedia document model called Madeus that describes multimedia scenarios. We show how this model is used for the generation of adaptable presentations. This presentation process is based on XSLT transformation techniques and constraint technologies for document formatting.

1 Introduction

Designing structured multimedia authoring systems is a great challenge for the industry which has to handle large amount of information. Several years ago, the notion of document classes was introduced for static documents (c.f SGML [6]) in order to enhance document productivity and quality. With the advent of standards like XML [22] and the increasing diversity of media types, there is also a need to have classes for multimedia documents. Typical examples of document classes are touristic guides, slideshow presentations, technical documentation (for installation and maintenance) or courseware. In addition, XML also allows the specification of the structure of document classes independent of their final presentation.

There is another feature that must be addressed when presenting multimedia documents: the adaptation of document content to the current presentation context. This operation must take into account user capabilities, user preferences, physical location, network and system resources. It also increases the complexity of editing these contextually-adaptable documents. Authoring tools must help the author to design documents that can have different renderings.

The aim of this paper is to propose a general framework for document production through the specification of a document presentation model. In our context, a document must not only be considered through its final presentation but also through a richer semantical representation defined by its class and its content. This approach is needed in order to take into account the increasing diversity of visualization and interaction devices : PDA (Personal Device Assistant), cellular phone, workstation, microphone, etc.

This paper is organized as follows: we firstly describe the general framework of our approach for multimedia document processing. We present then the multimedia document model called the *Madeus model* which aims to represent a multimedia scenario through its different dimensions: logical, temporal, spatial, etc. In the third section we describe the process that enables our system to generate the presentation of any structured document using this document model (see figure 1). The presentation is generated through a set of transformation steps. Finally we show how this model address the contextual adaptability problem and we compare it to related work. In the last section, we suggest some perspectives for addressing the problems of authoring such documents.

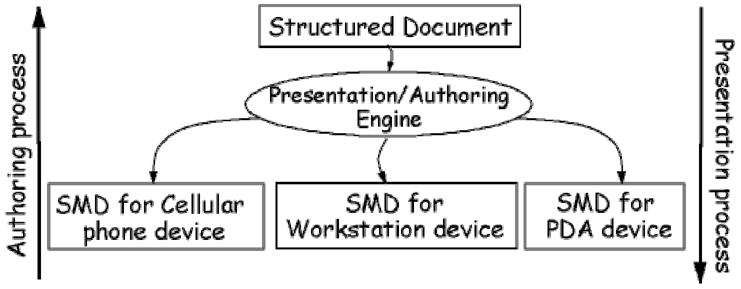

Structured Document
Presentation/Authoring Engine
SMD for Cellular phone device
SMD for Workstation device
SMD for PDA device
Authoring process
Presentation process

Fig. 1. General presentation and authoring architecture of a structured document in the context of a SMD (Structured Multimedia Document)

The ideas presented in this paper are presented through a sample document which is a slideshow (see section 2.2). Its presentation has been obtained using our presentation engine called Kaomi.

2 Document Processing Model

2.1 Motivation and Requirements

In order to provide a general framework for document processing, there is a need for an intermediate document representation between the higher level XML-based source formats and the lower level rendering format (execution-oriented) (see figure 2). The aim of this intermediate level is to capture all the information required for the formatting process. This intermediate representation can be compared to XSL-FO [20] or the Grif-Thot Abstract Image structure [5] [13].

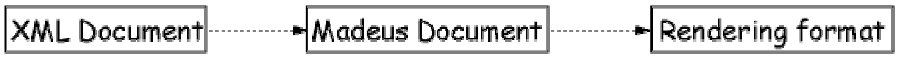


Fig. 2. Different document formats

In our system, we propose a generalization of this approach to time-based multimedia documents. Our goal is not only to handle structure and spatial layout but also synchronization and interactivity. The Madeus processing model is designed in order to render any kind of document classes encoded using a XML DTD [22] or a schema [23]. We try also to cover a wide range of multimedia scenarios using heterogeneous objects such as video, sounds, text, etc.

2.2 The Sample Document

In this paper, the sample document that we will use is presented in figure 3. It has been written according to the DTD developed by Norman Walsh which can be found at [24].

```
<slides>
  <title>XML And Web Applications</title>
  <screen>
    <videodata fileref="http://.../film.mpg"/>
  </screen>
  <foil>
    <title>Introduction</title>
    <para>...</para>
  </foil>
  <foil>
    <title>Multimedia systems</title>
    <para>...</para>
  </foil>
</slides>
```

Fig. 3. A XML source format of the slideshow

The *slides* element is the element root for a slideshow document. It contains a title element, a screen element which can be used as a background image or/and a video, and a list of foil elements. The *foil* element defines one slide. This content can be paragraphs, sounds, video, images, etc.

2.3 Architecture of the Multimedia Presentation System

The presentation process (see figure 4) used in our system takes as input any XML document, and produces a multimedia document. The source document belongs to a document class, which means that it can be validated against the class's DTD. The production of the multimedia document is achieved in two steps. First, the transformation step allows the creation of multimedia objects from the source document. It allows also us to set the synchronization relationship between these objects, the layout and the hyperlinks (see next section). In the second step, the multimedia document is formatted so that the result can be directly used by the presentation engine.

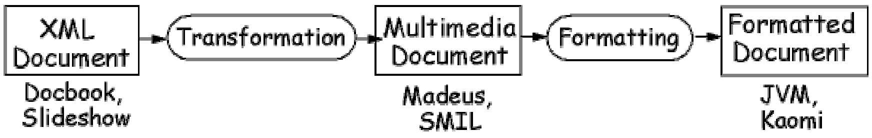
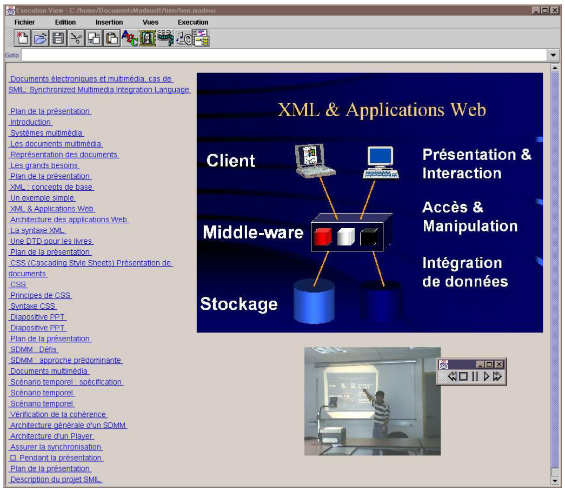


Fig. 4. Presentation process

For instance, one possible result of this entire process is the presentation of the XML source slideshow (see figure 5). On the bottom right of the screen there is the video of the show. In the top right appears the slide currently presented. The table of contents is presented on the left part of the screen. The title of slide currently presented is highlighted. The table of contents can be used to directly access a particular slide by clicking on the corresponding title.



Documents électroniques et multimédia, cas de SMIL: Synchronized Multimedia Integration Language

Plan de la présentation
Introduction
Systèmes multimédia
Les documents multimédia
Représentation des documents
Les grands besoins
Plan de la présentation
XML : concepts de base
Un exemple simple
XML & Applications Web
Architecture des applications Web
La syntaxe XML
Une DTD pour les livres
Plan de la présentation
CSS (Cascading Style Sheets) Présentation de documents
CSS
Principes de CSS
Syntaxe CSS
Diapositive PPT
Diapositive PPT
Plan de la présentation
SDMM : Défis
SDMM : approche prédominante
Documents multimédia
Scénario temporel : spécification
Scénario temporel
Scénario temporel
Vérification de la cohérence
Architecture générale d'un SDMM
Architecture d'un Player
Assurer la synchronisation
□. Pendant la présentation
Plan de la présentation
Description du projet SMIL

Fig 5. A particular presentation of slideshow documents

3 Madeus Document Model

Numerous examples of other work exist in the area of modeling multimedia documents. SMIL 2.0[19] is an XML-based synchronization language. The media objects are hierarchically organized through sequential, parallel and exclusive operators. In addition, the author can explicitly specify the beginning and the end of object with both absolute (date or user event) or relative offsets. This language is too rigid and so is not suitable for document adaptation. The document model used in ISIS [16] is a timed Petri-net TPN* for modeling timed user interaction. It allows flexible time specification thanks to a constraint-based approach. Only the temporal dimension is taken into account. Vazirgiannis's document model [17] is expressed through algebraic and spatio-temporal compositions of events (from user, media, systems, etc.). The resulting specification is compiled into a Java program which renders the multimedia scenario. The Madeus model presented below takes advantage of these different approaches in order to wider cover multimedia scenario needs: the XML-based structuration, constraint-based specification for temporal and spatial properties and external event specification.
In Madeus, the description of a multimedia document is organized around four dimensions : logical, temporal, spatial and hypermedia. In this section, we discuss the model for each of these dimensions and show how to combine them together. The syntax used for the multimedia document model presented here corresponds to the intermediate format introduced earlier. In our system this syntax is formally described as a XML DTD and therefore it takes full advantage of all the XML existing tools.

The DTD itself can be found at [18] and in the remaining part of this section we only use fragments of document instances encoded according to this DTD.
In accordance with the idea of separating document information into dimensions, the general structure of each document instance is decomposed in four main parts:

- MediaContent and MediaUse dimensions that describe the logical structure of the document
- A Temporal dimension for synchronization between document parts
- A spatial dimension for layout

Because hypermedia information is closely related to interactivity, it is encoded in either the Temporal or MediaUse parts. A Madeus XML source document look like this:

```
<Madeus>
  <MediaContent> ... </MediaContent>
  <MediaUse> ... </MediaUse>
  <Temporal> ... </Temporal>
  <Spatial> ... </Spatial>
</Madeus>
```

Each part is detailed in following sections.

3.1 Logical Model

A multimedia presentation is composed of a set of *media objects*, for instance a picture, a sound, a 3D animation, etc. In order to reuse the same content, its specification is separated from its use context. The *MediaContent* element contains raw media data, for instance, pixels of a picture, characters of a text, etc., and the intrinsic properties of the media, like the duration of a video or its size. The *MediaUse* element indicates a particular use of the content with specific style properties, for instance a line border color, a font size, etc.
The content part can also be used to refine the media description. For instance, the content of a video can be structured in sequences, scenes, shots, etc. [14]. In our sample document, this will allow synchronization between the table of contents entries and the video of the show.
The logical model allows us to hierarchically organize contents and objects. In the example of figure 6, a group element of type *C-Group* or *U-Group* just plays an aggregate role. Its semantics depend on the document type and not on its presentation. For instance, for the slideshow document, media contents can be gathered by media types and media uses can be gathered by the slide structure of the slideshow. Thanks to a simple inheritance facility that is applied on the logical structure, a group element can define default values for some attributes of its children elements (for instance the color, the character size, etc.).

```
  <MediaContent> <!-- Content specification part -->
    <C-Group>
      <C-Group ID="Text" MIMEType="text/plain">
        <DefContent ID="ST1">Introduction</DefContent>
        <DefContent ID="ST2">Multimedia</DefContent>
        <C-Group ID="Video" MIMEType="video/mpg">
          <DefContent ID="Film" src="http://./Film.mpg">
            <Scene StartFrame="0" EndFrame="20">
              <Shot StartFrame="0" EndFrame="5"/>
              <Shot StartFrame="5" EndFrame="20"/>
            </Scene>
          </DefContent></C-Group></C-Group>
  </MediaContent>
  <MediaUse>  <!-- Objects specification part -->
    <U-Group>
      <DefUse ID="U-Film" Content="Film" BorderWidth="1"
              BorderColor="black"/>
      <U-Group ID="TOC_entries" FontColor="black">
        <DefUse ID="U-title1_toc" Content="SlideTitle1"
                FontSize="12"/>
        <DefUse ID="U-title2_toc" Content="SlideTitle2"
                FontSize="12"/>
      </U-Group>
      <U-Group ID="Slide1" FontColor="blue">
        <DefUse ID="U-title1" Content="SlideTitle1"
                FontSize="32"/>
        <U-Group ID="U-Body1"> ... </U-Group></U-Group>
      <U-Group ID="Slide2"> ... </U-Group>
    </U-Group>
  </MediaUse>
```

Fig. 6. The slideshow logical model

3.2 Temporal Model

The temporal model allows the organization of media objects over time. It is based on previous work on Madeus where a specific markup (a language) has been specified to synchronize the presentation [7]. The underlying model of this language is interval-based. This means that each object has a corresponding time interval characterized by a *begin*, a *duration* and an *end* attribute. Each of these attributes has a range of values *[min, pref, max]* (from zero to *indefinite*) instead of a single value as in SMIL [19].
Every MediaUse element is associated a temporal *interval* element that carries all the temporal attributes required for its schedule. In particular the *Duration* attribute can override the intrinsic duration of the media. If this interval duration is larger than the intrinsic one, the additional attribute *Fill* allows the specification of the desired behavior: *Fill="repeat"* (the media is replayed during the interval), *Fill="freeze"* (the last image is displayed during the remaining time) or *Fill="cut"* (the media is withdrawn). The specification can state that some parts of the scenario may become active only when an external interaction is performed (for example when the user fires a hyperlink, see section 3.4 and 3.6). Such links are defined with a classical *HRef* attribute attached to the source interval. Notice that the beginning of interval for such target elements is set to the indefinite value.
For instance, the interval T-slide1_toc associated with the media slide1_toc is specified by:

```
<Interval ID="T-slide1_toc" Object="slide1_toc"
          Duration="min:50s pref:60s max:300s"
          Fill="freeze" HRef="@T-slide1.begin"/>
```

In the Madeus language, the synchronization is specified both by composite nodes and temporal relations. A composite node (*T-Group* element) is used to temporally group interval elements. A *during* constraint is set between each child and its parent.

This basic synchronization can be refined with temporal relations between the descendants of a composite node. For instance, specifying that two slides play in sequence can be done by placing the *meets* relation between them.
There are two ways to declare the intervals that are involved in a temporal relation: either by their explicit name (the *ID* attribute) or by an implicit name defined by the relative position of the intervals: *prev*, *all* and *next*. This last facility is needed because in some cases, the interval ID is not known beforehand since new elements are produced during the transformation process (see section 4). In some other situations this relative naming simplifies the description, for example when a temporal relation applies on all the children elements of a group. For instance, the specification of a sequence of slides can be achieved like this :

```
<T-Group>
  <Interval ID="T-slide1" Duration="pref:20s max:25s"/>
  <Interval ID="T-slide2" Duration="pref:20s max:25s"/>
  <T-Relations>
    <T-Relation Name="Meets" Intervals="all"/>
  </T-Relations>
</T-Group>
```

The following specification represents the part of a possible temporal scenario of the slideshow example. When the document starts, the table of contents, the first slide and the first frame of the video are displayed. From line 7 to 13, slides are presented in sequence separated in preference by 20 seconds. In order to show a particular slide, temporal links are added on each slide title entry in the table of contents (see section 3.4).

```
1.<Temporal>
2.  <T-Group ID="T-Root">
3.    <T-Group ID="T-TOC" Duration="pref:indefinite">
4.      <Interval ID="T-slide1_toc"
                  Object="U-title1_toc" Fill="freeze"
                  HRef="@T-slide1.begin"/>
5.      <Interval ID="T-slide2_toc"
                  Object="U-title2_toc" Fill="freeze"
                  HRef="@T-slide2.begin"/>
6.    </T-Group>
7.    <T-Group ID="T-Slideshow" Speed="pause">
8.      <T-Group ID="T-slide1"
              Duration="pref:20s max:25s">...</T-Group>
9.      <T-Group ID="T-slide2" Duration="pref:20s
                 max:25s"> ... </T-Group>
10.     <T-Relations>
11.       <T-Relation Name="Meets" Intervals="all"/>
12.     </T-Relations>
13.   </T-Group>
14.   <Interval ID="T-Film" Object="Film"
                Speed="pause"/>
15.   <T-Relations>
16.     <T-Relation Name="Starts" Interval1="T-Root"
                    Interval2="T-TOC"/>
17.     <T-Relation Name="Starts" Interval1="TOC"
                    Interval2="T-Slideshow"/>
18.     <T-Relation Name="Equals" Interval1="Slideshow"
                    Interval2="T-Film"/>
19.   </T-Relations>
20.  </T-Group>
21.</Temporal>
```

3.3 Spatial Model

The spatial model is basically similar to the temporal model. The main differences are the use of a spatial vocabulary (left_align, bottom_spacing, etc.) and the extension to support two dimensions unlike the temporal language which has a single dimension. In addition, a spatial attribute cannot have indefinite value. More precisely, the spatial model organizes the document space as a 2D box hierarchy. A composite node (*S-Group* element) allows the grouping of a set of 2D shapes (*Shape* element) inside 2D boxes as illustrated below.

```
<Spatial>
  <S-Group ID="SpatialRoot">
    <S-Group ID="S-TOC">
      <Shape ID="S-slide1_toc" Object="U-title1_toc"
             Left="pref:10px" Top="pref:20px"/>
      <Shape ID="S-slide2_toc"
             Object="U-title2_toc"/>
      <S-Relations>
        <S-Relation Name="Left_align"
   Shape1="S-slide1_toc" Shape2="S-slide2_toc"/>
        <S-Relation Name="bottom_spacing"
   Distance="min:5px max:15px" Shapes="all"/>
      </S-Relations>
    </S-Group>
   ...
  </S-Group>
</Spatial>
```

3.4 Hypermedia Model

The goal of this model is to describe links between elements or different parts of elements. The basis of this model is the XLink standard [21] enhanced with temporal and spatial behavior. The supported properties of hypermedia links are the following:

- **Display behavior.** When activated, the target element can either be displayed in a new frame, replace the source element or be embedded into it.
- **Target location.** This property indicates the target location of the link. Currently, this value is a URI-reference. But in a multimedia context, it can be extended to reference a temporal and a spatial location. Temporal locations can be absolute (a date) or relative to the beginning of a scenario part (a timestamp, a beginning of a T-Group element or an Interval element, etc.). Likewise spatial locations can be absolute or relative to the position of an element.
- **Activation type.** This property defines how the link must be activated. It can be specified through the following attributes: *HRefDur* (the period of time when the element can be activated), *HRefNumber* (the number of allowed activations) and *HRefActivation* (the link can be activated automatically, or interactively using differents interactive sources, see section 3.6).

For instance, the previously defined table of contents can be completed by the following temporal links:

```
...
<T-Group ID="T-TOC">
  <Interval ID="T-slide1_toc" HRef="@T-slide1.begin"
```

```
              HRefNumber="indefinite"
            HRefActivation="OnPointingCursorActivation"/>
    <Interval ID="T-slide2_toc" HRef="@T-slide2.begin"
              HRefNumber="indefinite"
            HRefActivation="OnPointingCursorActivation"/>
  </T-Group>
  ...
```

3.5 Links between Dimensions

In the previous sections, we have described the different dimensions of the multimedia documents without considering interactions between them. If examined two by two, twelve combinations may exist between these dimensions. Here are some meaningful examples of such combinations :

- **Style over time.** This combination is used to specify what is called *style animation*. A style animation is a discrete or continuous modification of a style attribute during a time interval. For example, changing the color from black to white over a two seconds period is specified by:

  ```
  <AnimateMotion ID="ColorAnim" Attribute="TextColor"
                 Values="from:black to:white"/>
  ```

- **Spatio-temporal.** This combination is used to specify *spatial animations* like motion, zoom effects, such as proposed in animation section of SVG [4]. In our model, we extend animation specification with the ability to enforce a spatial relation during a given time interval:

  ```
  <S-Relation Name="Left_align" Shapes="all"
              Interval="T-IntroInterval"/>
  ```

- **Link over time.** This combination is used to specify hypermedia properties that change over time, like activation, target location, display behavior, etc
- **Spatial-Link.** This combination is used to specify that the target location of a link depends on a given spatial location of an element.

3.6 Abstract Devices for Interactivity

Interval-based models are well adapted for storytelling documents in which no interactivity is needed. We propose to extend this model with interactivity while keeping schedule characteristics for predictive parts of the scenario.

In the section 3.2, we defined a temporal model in which the beginning of an interval is indefinite. Activating such an interval can only be done interactively. The consequence of runtime activation of an interval is the dynamic calculation of the beginning value of target intervals. Therefore, the scenario must be formatted dynamically since not all the values are known at the beginning of the presentation (c.f. section 5).

In order to render the document correctly to the user, the devices used as the source of interactivity must be specified. Given the diversity of input devices and our desire to cover different types of devices, an abstraction of these input devices is required. Each system has a device pointer, such as a mouse device for workstation, a pen

device for PDA, a tactile screen for interactive kiosks. In our model, we define several abstract devices such as the *PointerCursor* that allows the specification of that the user can cross *over* an element, can *activate* it, etc.
For instance, animating the color of a table of contents entry when the user crosses over the entry can be done as follows:

```
<MediaUse>
  <DefUse ID="slide1_toc" TextColor="Black">
    <AnimateMotion ID="ColorAnim" Attribute="TextColor"
                   Values="from:black to:white"/>
  </DefUse>
</MediaUse>
<Temporal>
  <Interval Object="slide1_toc">
    <Interval HRefNumber="indefinite"
              HRefActivation="OnPointerCursorOver"
              Object="slide1_toc.ColorAnim"/>
  </Interval>
  ...
</Temporal>
```

4 Transformation

The transformation process enables the creation of a multimedia presentation from both the source document and one or many presentation sheets that specify their temporal, spatial and navigation behavior (see the bottom part of figure 7). Moreover, this process can be used in order to generate table of contents, index, numbering, etc.
Several transformation languages exist. Balise [1] is a script language in which some functions of tree manipulation (creation and copy) are provided. Omnimark [10] is a streaming programming language. It consists of rules that define data events such as general markup events produced when parsing a XML document. XSLT [3] [9] is a semi-declarative language designed especially for XML document transformation. It consists of transformation rules (templates) associated with patterns. When a rule pattern matches in the source, the corresponding rule is instantiated to create the result tree.

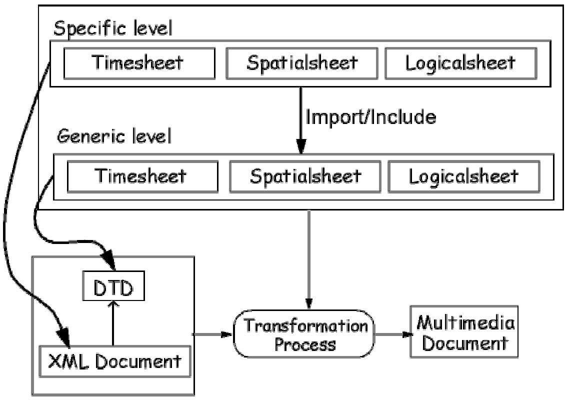


Fig. 7. Multimedia presentation with transformation sheets

As the transformation power of these three languages are quite similar, we have chosen to develop our transformation system with XSLT. This allows us to take advantage of the ongoing work on this standard. Moreover, XSLT allows us to structure and combine transformation rules as a set of modules.
As we want to benefit from generic specification (as provided by source XML DTD), we have identified two levels of transformation (see figure 7): the generic level and the specific level. The former allows the transformation of any valid XML document that conforms to the DTD (or schema) for which the transformation has been specified. The latter allows the specification of transformation behaviors only for a particular document (or instance). Moreover, it's possible to define a transformation for each dimension of a multimedia document.
In our slideshow example, we have defined several presentation sheets in order to generate the presentation illustrated in figure 5. We give below some excerpts of these sheets :

```
<!-- Root sheet -->
<xsl:stylesheet>
  <xsl:include href="genlogicalsheet.xsl"/>
  <xsl:include href="genspatialsheet.xsl"/>
  <xsl:include href="gentemporalsheet.xsl"/>
  <xsl:include href="speclogicalsheet.xsl"/>
  <xsl:include href="specspatialsheet.xsl"/>
  <xsl:include href="spectemporalsheet.xsl"/>
</xsl:stylesheet>

<!-- Fragment of logical generic sheet -->
<xsl:template match="foil/title"
              mode="content.title.toc">
<!-- {position()} is used in order to have
     unique identifier -->
  <madeus:DefContent ID="SlideTitle{position()}">
    <!-- Gets the title content -->
    <xsl:value-of select="."/>
  </madeus:DefContent>
</xsl:template>
<xsl:template match="foil/title" mode="use.title.toc">
```

```
    <madeus:DefUse ID="U-title{position()}_toc"
        Content="SlideTitle{position()}" FontSize="12"/>
  </xsl:template>

  <!-- Fragment of temporal generic sheet -->
  <xsl:template match="slides" mode="temporal">
    <madeus:Temporal>
      <madeus:T-Group ID="T-Root">
        <T-Group ID="T_TOC" Duration="pref:indefinite">
          <xsl:apply-templates select="slide"
                               mode="temporal.title.toc"/>
        </T-Group>
        ...
      </madeus:T-Group>
    </madeus:Temporal>
  </xsl:template>
  <xsl:template match="foil/title"
                mode="temporal.title.toc">
    <madeus:Interval ID="T-slide{position()}_toc"
                     Object="U-title{position()}_toc"
                     HRef="@T-slide{position}.begin"/>
  </xsl:template>
```

Notice that this process allows the definition of several presentation for the same source document. For instance, with another set of presentation sheets, the slideshow document can be presented just as a sequence of slides, without the table of contents, neither the video. It can also be presented as a table of thumbnails.

5 Formatting

The result of the transformation process is a high level specification of a multimedia scenario. In order to execute it, the formatting process calculates attribute values that will be directly used by the execution engine. The underlying techniques used for formatting heavily depend on the target application. We can identify four levels of such applications (in increasing complexity order):

- Final multimedia presentation without timing constraint (such as an interactive kiosk).
- Final multimedia presentation with timing constraints (web access). The rendering must be able to adapt to variation delays of media access.
- Adaptable rendering (see section 6).
- Presentation view in an authoring context.

We have developed our formatting process using our Kaomi presentation engine [7]. It takes as input a Madeus specification and either produces a direct rendering (using JMF library for playing dynamic media) or a portable format such as SMIL (with a potential loss of information).

In order to provide high level adaptability features with reliable formatting results, our formatting system relies extensively on constraint technology. Moreover as the Madeus model aims at maintaining relative specifications until the formatting step (through relations), it is possible to compute presentation parameters (scheduling, placement) at the earliest possible moment.

In all cases, the formatting process relies on the following steps:

- **Unit resolution.** The Madeus model allows the specification of attribute values using many units. These ones are converted into canonical units (pixels, milliseconds, etc.).
- **Consistency checking.** As the Madeus model relies on constraint specification, the system is able to verify that every multimedia document produced by the transformation process is consistent (i.e there exists at least one possible execution). This is done through algorithms on the resulting constraint network managed by Kaomi [8].
- **Search for a solution.** Basically this is the step that produces a specific presentation solution among those resulting from the Madeus specification. The presentation values are calculated using constraint technology [8]. The search of a solution can be oriented by context parameters (see next section).

6 Contextual Adaptability

Contextual adaptability is the capability for a document processing system to take into account the following parameters:

- **User profile** such as his language, his skill level, his physical deficiencies, his physical location, etc.
- **Hardware profile** such as screen size, CPU type, speakers presence, modem speed, etc.

With the emergence of new supports and devices, this function becomes more and more important, and is currently investigating by several research teams [2] [15].

Adaptation parameters can act in different places: inside a transformation sheet, in separate transformation sheets or as an input of the formatting process. A given parameter can be used at different levels of the process. For instance, a spatial-style sheet can define spatial properties for devices with similar screen sizes, while little differences in screen size can be supported by the formatting step thanks to the constraint approach. Indeed in the first case, the structure of the resulting document may be very different and so the transformation process is required. In the second case, the spatial layout can be adapted thanks to the ability to specify attributes by ranges of values. For instance, the specification of the spacing between text entries in the table of contents can be an interval from 5 to 15 pixels.

In addition to these two adaptation modes, we have identified another one which can be considered as an intermediate adaptation step (see arrow number 2 of figure 8). this adaptation step aims at providing transformations that are independent from document classes. For instance, to take into account the blindness of the user, we need a rule which transforms all text into speech sounds. Notice that this rule can be applied to the multimedia document instead of to the source one without changing the multimedia structure. More generally, this step, called decoration, can be used to add or remove style or media attributes, for instance to adapt colors to colour-blind persons. The advantage of this step relies on its simplicity compare to transformation sheets. However, it requires both flexible specification in the document model and a constraint-based formatting process to allow the final adaptation during the scheduling: for instance the media decoration of a document from text to sound is possible only if the temporal structure of document is specified through relations and range of durations.

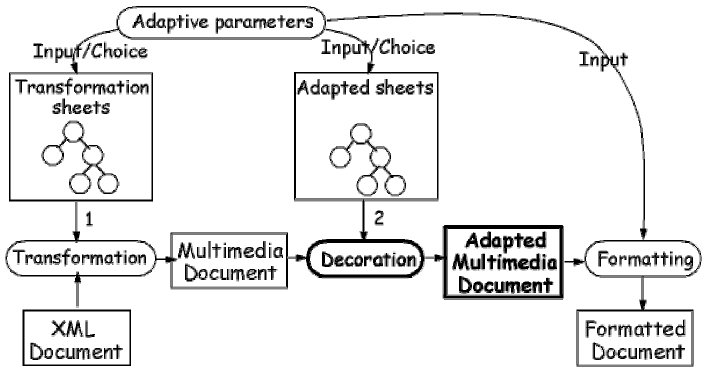


Fig. 8. Adaptive Presentation process

To enable the system to choose among several transformation and adaptation sheets, the sheets must be labeled with meta-data about that record the adaptive parameters. The transformation and decoration processes use these parameters to compare them to the effective context parameters and then produce the final adapted multimedia document. This is the subject of a W3C note called CC/PP [12]. This note propose a method for using RDF, the Resource Description Format of the W3C [11], to create a framework for describing user preferences and device capabilities.

7 Perspectives and Conclusion

This paper has proposed a general process for presenting generic and adaptable multimedia documents. The system suggests to split the presentation generation in three layer. The first layer us allows to encode the content independently from its presentation using an XML DTD. A first transformation step allows us to obtain a new representation which reflects all the dimensions needed for presentation. This step allows us to adapt the content to the presentation device and to user preferences. A second simple transformation (a decoration) is then applied in order to adapt the media to the end user. Finally, the formatting process produces a representation playable by the presentation engine. The application has been implemented in a Java prototype built on top of the Kaomi multimedia toolkit. It can take as input any XML document thanks to Xalan, a Java implementation of a XSLT processor. This prototype has been experimented for the slideshow document class and allows the production of slideshow presentations on a workstation screen and a PDA.
We are currently investigating new adaptation techniques. In particular, we are looking for a stronger integration with a content negotiation layer. We are also designing a network adaptation layer which allows us to manage the quality of service and the resource allocation between the different parallel streams. We think our architecture can be a good framework for such extensions.

Another area of investigation is related to authoring such documents. Indeed both genericity and adaptation requirements increase the complexity of editing. We can identify two levels of difficulties:

- **interface level** perception of the temporal dimension, multiple level of specification (from source document, presentation sheets and adaptation sheets) and multiple target presentations.
- **internal data level** complexity of the data management in a tree transformation process.

We promote the idea of providing several edition modes inside an integrated tool. Kaomi is also an editing toolkit which proposes high level editing functions for multimedia documents through multiple views. We are on the process to extend this toolkit in order to take into account the different levels of edition: source document, presentation sheets and adaptation sheets.

References

1. Balise, Balise 4, http://www.us.balise.com/products/balise/index.htm. , 2000.
2. Susanne Boll, Wolfgang Klas, and Jochen Wanden, A Cross-Media Adaptation Strategy for Multimedia Presentation, Proceedings of the ACM Multimedia'99, October 30–November 5, 1999, Orlando, Florida, USA, 1999.
3. James Clark, XSL Transformation, http://www.w3.org/TR/xslt, 1999.
4. Jon Ferraiolo, Scalable Vector Graphics (SVG) 1.0 Specification, http://www.w3.org/TR/SVG, 2000.
5. R. Furuta, V. Quint, and J. André, Interactively Editing Structured Documents, Electronic Publishing – Origination, Dissemination and Design, vol. 1, num. 1, pp. 19–44, April 1988.
6. International Standard ISO 8879, Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), International Standard Organization, 1996.
7. Muriel Jourdan, Cécile Roisin, and Laurent Tardif, A Scalable Toolkit for Designing Multimedia Authoring Environments, Special number, 'Multimedia Authoring and Presentation: Strategies, Tools, and Experiences' of Multimedia Tools and Applications Journal, Kluwer Academic Publishers, 1999.
8. Muriel Jourdan, Cécile Roisin, and Laurent Tardif, Constraints Techniques for Authoring Multimedia Documents, Constraints Journal, Kluwer Academic Publishers, to appear.
9. Michael Kay, XSLT Programmer's Reference, Wrox Press, 2000.
10. Omnimark, Guide to OmniMark 5, http://www.omnimark.com/develop/om5/doc/, 2000.
11. Ora Lassila and Ralph R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/, 1999.
12. Franklin Reynolds, Johan Hjelm, and Spencer Dawkins, Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation, http://www.w3.org/TR/NOTE-CCPP/, 1999.
13. Cécile Roisin and Irène Vatton, Merging Logical and Physical Structures in Documents, Electronic Publishing – Origination, Dissemination and Design, special issue Proceedings of the Fifth International Conference on Electronic Publishing, Document Manipulation and Typography, EP94, vol. 6, num. 4 , pp. 327–337, April 1994.
14. Cécile Roisin, Tien Tran_Thuong, and Lionel Villard, Integration of structured video in a multimedia authoring system, Proc. of the Eurographics Multimedia'99 Workshop, Springer Computer Science, pp.133–142, Septembre 1999.

15. Lloyd Rutledge, Lynda Hardman, Jacco van Ossenbruggen, and Dick C.A. Bulterman, Mix'n'Match : Exchangeable Modules of Hypermedia Style, Proceedings of the ACM Hypertext '99, 1999.
16. Junehwa Song, G. Ramalingam, and Raymond E. Miller, Modeling Timed User interactions in Multimedia Documents, IEEE International Conference on Multimedia Computing and Systems, Hiroshima, pp.407–416, June 1996.
17. Michalis Vazirgiannis, Interactive Multimedia Documents, Lecture Notes in Computer Science n°1564, Springer, 1999.
18. Lionel Villard, Madeus model DTD, http://www.inrialpes.fr/opera/madeusmodel.dtd, 2000.
19. W3C, SMIL Boston Specification, http://www.w3.org/TR/smil-boston, 2000.
20. W3C, Extensible Stylesheet Language (XSL), http://www.w3.org/TR/xls, 2000.
21. W3C, XML Link Language (XLink), http://www.w3.org/TR/xlink, 2000.
22. W3C, Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/REC-xml, 2000.
23. W3C, XML Schema Part 0 : Primer, http://www.w3.org/TR/xmlschema-0, 2000.
24. Norman Walsh, Slides doctype, http://nwalsh.com/slides/index.html,2000.

Abstraction Levels in Web Document Formats

Håkon Wium Lie

Opera Software, Oslo, Norway
howcome@opera.com
http://www.opera.com/people/howcome

Abstract. The paper gives an overview of current and emerging document formats on the Web, and discusses the abstraction level of the different formats. The "ladder of abstraction" is introduced as a measuring stick for Web formats, and various levels from presentation (at the bottom of the ladder) to semantics (at the top) is described. The importance of reaching certain abstraction levels in order to support device-independent formats, universal accessibility, and scalable presentations is stressed. The document formats discussed in the paper are: HTML, PDF, GIF, PNG, MathML, and XSL-FO. Also, the effect of style and transformation languages (namely CSS and XSLT) are described.

1 Introduction

Over the last decade, the Web has established itself as an important medium for publishing documents. The simplicity of HTML [16, p. 27], which is the major document format on the Web, has been an important reason why the Web in a short time period has achieved this position. Authors without much experience in electronic publishing have quickly been learning HTML by looking at the source code of other documents and using simple text editors to author documents [16, p. 62]. HTML, in its simplest form, is also easy to implement and a number of HTML browsers appeared in the early days of the Web. These browsers supported a wide range of devices, from text terminals to high-resolution graphics screens as well as aural renderings. Today, millions of browsers around the world understand HTML and the number is quickly increasing.

Document authoring and formatting systems have a longer history than the Web. A survey paper on document formatting systems written in 1982 [6] characterizes formatting as "mapping from abstract objects to concrete objects". The web has not changed this definition, but forces one to rethink what an abstract object is, what a concrete object is, and where the formatting process should take place – on the client or server side. Also, this article claims that there exists more than those two types of objects and that there is a continuous "ladder of abstraction" between them.

A more recent article on "document species" [9] discusses document formats in the context of the Web. The article observes a preferential adoption of declarative markup over presentational markup and style sheets over inline formatting. This development would, if true, lead to documents formats at higher levels of abstractions. However, recent working drafts issued by W3C shows that the movement is not consistently going upwards on the ladder of abstraction. The section on “Style vs. transformation” discusses this further.

2 Current Web Document Formats

HTML lets authors mark the structural role of textual content. For example, headlines, paragraphs and lists can be identified as such, and HTML in its original form does not describe how the various elements are to be presented [1, p. 41]. The separation of structure and presentation stems from HTML’s ancestor SGML [7] and puts HTML on a higher level of abstraction than presentation-oriented formats, e.g., PDF [8]. PDF has no concept of paragraphs, and many users have discovered this when trying to copy content from PDF documents laid out in several columns. When selecting text, the selection will span across both columns and thereby mix text from several parts of the document into the same selection.

Bitmap image formats, e.g. [4] and PNG [15], are at an even lower level of abstraction. Normally, these are normally not considered document formats since they have no notion of text, but on the Web images are often used to convey textual content [14]. Treating text as images gives the author full control of fonts, layout and colors in the presentation. For users, however, images have several disadvantages: they take longer time to download and only allow visual access to the text. Other uses of the text – e.g., indexing of pages, cut/paste operations, and rendering through speech synthesizers – become impossible.

3 The Ladder of Abstraction

It is the view of the author that a document format’s abstraction level is important when assessing its usefulness on the Web. This section introduces the *“ladder of abstraction”* as a tool for evaluating document formats. The vertical nature of a ladder corresponds to how one describes abstraction levels as “high” or “low”.

Typical characteristics of document formats that are high on the ladder of abstraction are:

– the information needs processing in order to be presented. For example, in order to render an HTML document visually, the words must be broken into lines, fonts must be selected, and the characters must be rasterized.
– the information can be processed and presented in many different ways. Presenting a document visually is only one of several possibilities, others include aural renderings and braille embedding.
– the information is represented in a compact manner. Representing a letter with an 8-bit code is more compact than representing an image of the same character.

Conversely, document written in formats that are low on the ladder of abstraction need less processing in order to be presented, they have less flexibility of presentation, and they are less compact.

Another important observation is that it is generally possible to transform documents downward on the ladder, but much harder to move the other way [11]. For example, graphical Web browsers - in collaboration with the windowing system - rasterize HTML documents into pixels and thereby move information downwards on the ladder of abstraction. Optical Character Recognition (OCR) software attempts to climb the ladder by turning images into text, but OCR systems only work under optimal conditions, e.g. with certain font families and text sizes.

The ladder of abstraction is a simplified, one-dimensional scale. People with a background in mathematics, programming languages or structural linguistics will be familiar with the concept of abstraction. In the context of Web document formats, the author of this paper believes that the following criteria are good measuring sticks for assessing the level of abstraction:

- Is the text available? That is, does the format have a notion of characters that later can be mapped into glyphs, or does it represent text as images – in which case the text is not available.
- Is the logical order of text preserved? That is, do documents written in the format have a notion of the logical reading order of the content?
- Is the document scalable? For example, can text be laid out with different line lengths? Can the aspect ratio of the pages be changed?
- Can the roles of the various text elements be represented? For example, can the author mark part of the text as a headline? As a paragraph? As the name of a variable in a computer program? Being able to do distinguish between these roles is important e.g. when making documents available in braille since some text should be contracted (e.g. headlines), while other text should not (e.g. variable names) [12].
- Is the format device-independent? That is, can documents written in the format be rendered into many different devices (e.g. printers, screens, braille printers, and text synthesizers) or are documents intended for a single type of device? [17]
- Does the format contain application-specific semantics? HTML is a general document format that does not attempt to describe semantics from more specialized fields, e.g. mathematics, and therefore does not contain application-specific semantics.

Table 1 rates several existing and emerging Web document formats with regard to these criteria. HTML, PDF, GIF and PNG have been discussed above while the emerging XSL-FO and MathML are discussed below. XML, in which several of the emerging formats are written, is also included in the table and refers to documents published using private XML vocabularies where the sematics isn't generally known.

The rating in table 1 assumes that the document format is used in the best possible way. On the web today, this is often not the case. For example, many HTML documents use tables to enforce a certain layout. Often, due to the nature of table markup, these documents do not have text in logical order. Also, since the tables have been set to have certain widths (e.g. 600 pixels), the document is no longer device-independent.

Table 1. Existing and emerging Web document formats placed on the ladder of abstraction

	GIF PNG	PDF	private XML vocabulary	XSL-FO	HTML	MathML presentation elements	MathML content elements
application-specific semantics?	no	no	no	no	no	yes	yes
device-independent?	no	no	no	no	yes	yes	yes
roles known?	no	no	no	no	yes	yes	yes
scalable?	no	no	unknown	yes	yes	yes	yes
text in logical order?	—	no	unknown	yes	yes	yes	yes
text available?	no	yes	yes	yes	yes	yes	yes

4 Style versus Transformation

Style sheets describe how documents are to be presented for users by attaching style (e.g., fonts, spacing, and aural cues) to structured documents. Style sheets are not documents themselves, but serve a supporting role by allowing the presentation of a document to be separated from the content of the document.

Cascading Style Sheets (CSS) [2,10] allow authors to attach presentational properties to semantic elements. Before CSS, the Web experienced a strong push from authors to extend HTML into a presentational language rather than a semantically oriented language. Instead of adding new elements in order to achieve certain presentations, W3C has added new CSS properties to address requests from authors.

XSL (Extensible Stylesheet Language) [5] takes a different approach. Instead of attaching formatting properties to semantic elements, XSL suggests that authors transform the document into a set of formatting objects which can be expressed in XML. The difference between these two approaches is the topic of this section.

The XSL effort within W3C has produced two specifications. The first is a transformation language called XSLT [3], and the second is an XML vocabulary for formatting objects (called "XSL-FO" in this document) [5].

The most common use of XSLT is to transform XML data and documents into HTML on the Web server. Several implementations support XSLT and they allow content providers to use their favorite DTDs internally while serving HTML to the huge installed base of Web browsers. XSLT provides a declarative way of specifying simple transformations.

XSLT can also be used to generate XSL-FO. Formatting objects describe how chunks of information are formatted before presented to a human user. The push for XSL-FO comes from vendors with the goal of improving the quality of printed Web content. Unfortunately, when transforming documents into XSL-FO, the documents move downwards on the ladder of abstraction and only the human presentation is left. Moreover, the resulting documents are tied to a certain output media. Thus, accessibility and device-independence it threatened by the use of XSL-FO.

The transformation from semantic markup to formatting objects can also take place on the client side. Given the XML source and the XSLT transformation sheet, the client can convert the semantic markup into formatting objects. This preserves semantics, and the number of bytes sent over the Web will generally be smaller. In this scenario, however, there is no need for an XML vocabulary to express formatting objects since the formatting objects only exist within the client application. This highlights an important point: it is not formatting objects per se that are harmful (any system that does formatting uses some kind of formatting objects). The harm is done when formatting objects are stored and shipped over the Web.

4.1 Code Examples

This section will give three examples of how XSLT can be used. The first example transforms from XML to HTML, the second transforms from XML to XSL-FO and the third transforms from XML to HTML/CSS. All examples use this simple XML element as input:

```
<Heading1>The headline</Heading1>
```

Example 1: XML to HTML The first XSLT sheet transforms the XML element into HTML:

```
<xsl:template match="Heading1">
  <H1>
    <xsl:apply-templates/>
  </H1>
</xsl:template>
```

The result is:

```
<H1>The headline</H1>
```

The resulting HTML is at a high enough level of abstraction that device-independence and accessibility is preserved. What is lacking in information about how to present it.

Example 2: XML to XSL-FO In this example, the XSLT sheet transforms the XML element into a formatting object:

```
<xsl:template match="Heading1">
  <fo:block font-size="1.3em" margin-top="1.5em"
            margin-bottom="0.4em">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

The result is:

```
<fo:block font-size="1.3em" margin-top="1.5em"
          margin-bottom="0.4em">
  The headline
</fo:block>
```

The difference between example 1 and example 2 is one of semantics vs. presentation. When transformed into HTML, the semantics of the XML is preserved since the H1 element is globally recognized as being a headline of level 1. When transformed into XSL-FO, semantics is removed and replaced by presentational properties.

Example 3: XML to HTML/CSS The last example transforms XML into an HTML element with associated CSS stylistic properties:

```
<xsl:template match="Heading1">
  <H1 STYLE="font-size:1.3em; margin-top:1.5em;
             margin-bottom:0.4em">
    <xsl:apply-templates/>
  </H1>
</xsl:template>
```

The result is:

```
<H1 STYLE="font-size:1.3em; margin-top:1.5em;
           margin-bottom:0.4em">
   The headline
</H1>
```

The result preserves the semantics in the form of HTML elements, while presentational information is encoded in the CSS notation.

(When authoring with CSS, one would normally move the stylistic properties into a separate style sheet and not into an attribute as in the above example. Having separate style sheets eases maintenance and makes documents smaller. However, both forms are valid and one can programatically convert between the two.)

4.2 MathML

Mathematical Markup Language (MathML) [13] is an emerging format for describing mathematical notation. As expected from an application-specific language, it contains more semantics than general-purpose document formats like HTML

The developers of MathML recognized that different abstraction levels are needed to encode mathematics for the web. From the MathML recommendation:

> A fundamental challenge in defining a mathematics markup language for the Web is reconciling the need to encode both the presentation of a mathematical notation and the content of the mathematical idea or object which it represents.

MathML therefore contains both "presentation elements" and "content elements". This allows authors to encode semantics when available and presentation when semantics is either unavailable or not covered by the content elements.

As an example, consider this mathematical expression:

$$a - b$$

Using MathML's presentation elements, the expression can be written:

```
<mrow>
  <mi>a</mi>
  <mo>-</mo>
  <mi>b</mi>
</mrow>
```

Using MathML's content elements, the expression can be written:

```
<apply>
  <minus/>
  <ci>a</ci>
  <ci>b</ci>
</apply>
```

Although referred to as "presentation markup", it should be noted that the presentation elements are at a higher level of abstraction than e.g. HTML. In table 1, both MathML presentation elements and MathML content elements are given the same rating. This shows that the criteria used are not well suited for assessing the level of abstraction above a certain level. Further criteria should be developed as new formats better describe application-specific semantics.

5 Conclusions

Documents on the Web should strive to retain information at a high enough level of abstraction to preserve device-independence and accessibility. By performing the down-translation of text in the browser rather than in the server, the user's preferences and needs can be taken into account.

Style sheet languages like CSS can augment structured document formats by attaching presentational information to semantic elements.

Transformation languages like XSLT can move information downwards on the ladder of abstraction, but will not be able to increase the level of abstraction.

The use of XSL-FO as a document format on the Web is a step downwards on the ladder of abstraction and the move threatens device-independence and accessibility.

As a general rule, documents at higher levels of abstraction are better for the Web than documents at lower levels of abstraction. Document formats below HTML on the ladder of abstraction should not be used. The use of application-specific formats should be encouraged, while also keeping in mind that simplicity is a major reason for the success of the Web.

References

1. T. Berners-Lee. *Weaving the Web.* Harper San Francisco, 1999.
2. B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. Cascading style sheets, level 2. W3C Recommendation, World Wide Web Consortium, May 1998.
Available at http://www.w3.org/TR/CSS2.
3. J. Clark. XSL transformations (XSLT) version 1.0. W3C Recommendation, World Wide Web Consortium, December 1999.
Available at http://www.w3.org/TR/xslt.
4. Compuserve. Graphics interchange format, 1989.
5. Extensible style sheet language (XSL). W3C Working Draft, World Wide Consortium, March 2000.
Available at http://www.w3.org/TR/2000/WD-xsl-20000327.
6. Richard Furuta, Jeffrey Scofield, and Alan Shaw. Document formatting systems: Survey, concepts, and issues. *Computing Surveys*, 14(3):417–472, September 1982.
7. Charles F. Goldfarb, editor. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML).* International Organization for Standardization, Geneva, Switzerland, 1986. International Standard ISO 8879:1986.
8. Adobe Systems Incorporated. *Portable Document Format Reference Manual.* Addison-Wesley, 1993.
9. R. Khare and A. Rifkin. The origin of the (document) species. In *Proceedings of WWW7*, Brisbane, April 1998.
Available at http://decweb.ethz.ch/WWW7/1920/com1920.htm.
10. H. W. Lie and B. Bos. Cascading style sheets, level 1. W3C Recommendation, World Wide Web Consortium, December 1996.
Available at http://www.w3.org/TR/CSS1.
11. H. W. Lie and J. Saarela. Multipurpose web publishing using HTML, XSL, and CSS. *Communications of the ACM*, 42(10):95–101, October 1999.
12. P. Lorimer. *A critical evaluation of the historical development of the tactile modes of reading and an analysis and evaluation of researches carried out in endeavours to make the braille code easier to read and to write.* PhD thesis, University of Birmingham, December 1996.
13. MathML. W3C Recommendation, World Wide Web Consortium, July 1999.
Available at http://www.w3.org/1999/07/REC-MathML-19990707.
14. H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of SIGCOMM*, Cannes, 1997.
Also available at http://www.w3.org/Protocols/HTTP/Performance/Pipeline.
15. PNG (portable network graphics) specification, version 1.0. W3C Recommendation, October 1996. http://www.w3.org/TR/REC-png.
16. D. Raggett, J. Lam, and I. Alexander. *HTML 3 - Electronic Publishing on the World Wide Web.* Addison Wesley, 1996.
17. User agent accessibility guidelines 1.0. W3C Proposed Recommendation, World Wide Web Consortium, March 2000.
Available at http://www.w3.org/TR/2000/PR-UAAG10-20000310.

Automatic Geographical Hypertext "Multi-scaled Links" Generation

Nicolas Malandain and Mauro Gaio

GREYC - Computer Science Department
CNRS UMR 6072
University of Caen - France
{Nicolas.Malandain,Mauro.Gaio}@info.unicaen.fr

1 Introduction

Because text can express what can't be drawn, and graphic can draw what can't be written [Jol93], a great number of documents contain both expression modes. Importance in an understanding task of text/picture association has been highlighted in psycho-linguistic studies, especially in understanding spatial concepts and relations [Den89]. In this kind of *composite* document there are two different usual ways to link text and graphics: *explicit links*, authors use some specific linguistic expressions like : cf., see fig XX, . . . to refer from the text directly to a picture (in this case it's quite a simple problem to automatically find such links). *Implicit links*, in this case authors don't use any specific linguistic expressions they just "talk" about a same subject in the text and in the graphic. Our research has focused on developing a computational model for automatic extraction and "light" interpretation of some appropriate classes of these implicit links. To develop such a theory it's useful to select a working corpus. In this work geographic information based collection have been selected [1] to be the working corpus [RR94] and the class of implicit links is of course geographical one.

2 Implicit *Multi-scaled* Links

Several approaches propose different categories of links [TW87,VDP91,All96]. A link type is a description of the relationship between the source and the destination of a link. The types could be defined and assigned by the author or achieved automatically, anyway an appropriate browsing software can take advantage of that information.

J. Allan [All96] proposed a taxonomy of known link types based upon whether their identification can be achieved automatically or not. These links can be typed as for example *summary links* which start from a document and end to a summary of this document as opposed to *expansion links*. There are also *Comparison links* which identify similarities and differences between documents or *Equivalence links* for documents of the same topic.

[1] many thanks to Groupement d'Intérêt Public RECLUS, Maison de la Géographie and la Documentation Française

We propose to add the concept of granularity to provide precision on the size of the semantic units bound with a given link type, this is obtained by multi-scaling links (see fig. 1). We propose three major categories based upon capacity of a given typed link to include or to be included by others one —*i.e.*, a multi-scaled link is a link which contains links which themselves contain links... In the top-level category called *Global* a typed link connects two semantic units, for example : an *equivalence* link (a type of link representing strongly-related discussions of the same topic) between two documents about the same huge subject (as we gonna see in the working corpus). The included links belonging to the *Minimal* or the *Intermediate* category will join one or some smaller semantic units, for example: some paragraphs in the two documents where a same topic is discussed for an equivalence link, or two opposite ideas for a comparison link.

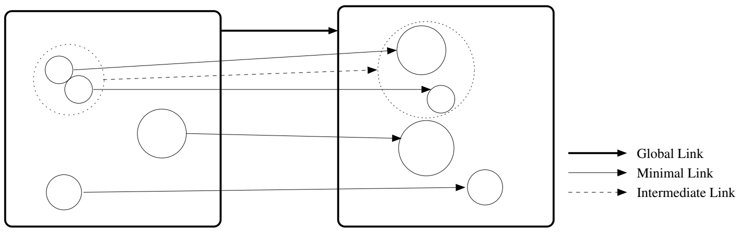


Fig. 1. General representation of multi-scaled-link : from one semantic unit (left hand image) to another one (right hand image) referring to a same subject

The position of the links in a multi-scaled link taxonomy depends on a chosen scale, so we give some general definitions with some examples:

Global Link (GL): From one semantic unit (textual or graphical, or any media) to another one (textual or graphical, or any media) referring to a same huge subject. For example, links between sections or paragraphs, links from a paragraph to the whole of an image.

Minimal Link (ML): interconnect two elements inside a GL. *Minimal* because it connects two smallest semantic units referring to a very specific entity or a particular concept. For example, links between a sentence and a sentence or a sub-set of a sentence, links between a sentence or a sub-set of a sentence with a small part of a graphic.

Intermediate Link (IL): are some sets of MLs. Such link gives the same meaning to several entities linked by MLs.

3 Multi-scaled Links in Geographical Documents

Our geographical corpus contains numerous implicit links we have selected those linking text with graphs and belonging to the equivalence category. These links build relations between graphs and parts of the text, in general paragraphs where authors comment on a geographical phenomenon. The graphs are some pie charts, histograms, or more frequently geographical maps, generally constructed with statistical data (census, rates, ...) and reveal some society's behaviors. In the text, authors comment these behaviors and points out some particular information highlighted by the graphs (like for instance maxima, minima, ...)

We have especially studied the implicit relation (link) between geographical thematic maps and their comments. An example of this kind of implicit relations can be observed in the figure 2. Our study allowed us to built up an automatic representation of some of these implicit relations by using the artifact of multi-scaled links.

Global Link (GL): from one geographical map to its comment which is usually all a paragraph. They talk about the same thing in two different ways : graphical and textual modes.

Minimal Link (ML): from a linguistic expression of a geo-referenced entity in the comment to its location in the map.

Intermediate Link (IL): from geo-referenced entities cite together (origins of MLs) to their locations (destinations of MLs) in the map. The explanation of these ILs is that if geo-referenced entities are cited together it is that they have the same behavior and in the map they have the same tones of color.

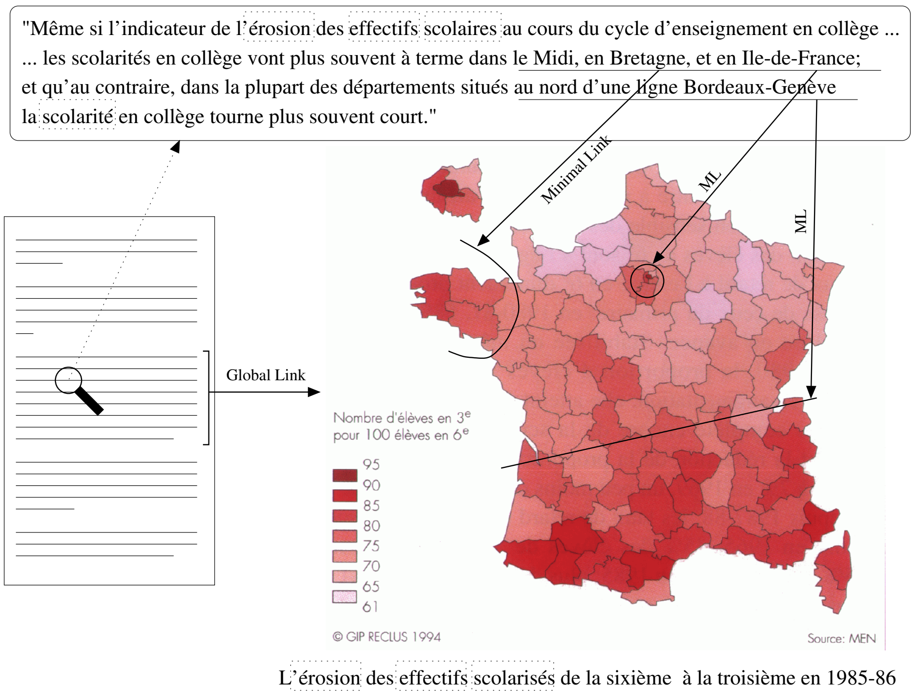
Comments consist of several parts giving different aspects of a same phenomenon: (1) the context, (2) an overall (as for example its increasing or decreasing on all the observed area), (3) some of its specific localized behaviors.

Language provides numerous lexical terms and linguistic expressions to describe space. For explaining geographical phenomenons authors use a reduced sub-set of this linguistic material, generally locative expressions [Za98]. These expressions could be:

– basic geo-referenced explicit named-entities: *en Bretagne, à Paris, dans le Massif central.*
– geo-referenced fuzzy named-entities: *dans les départements du Bassin parisien*[2]*, du Midi méditerranéen*
– orientations with respect to an explicit geo-referenced named-entity: *dans la France de l'Ouest, les départements du sud et de l'est du Massif central*[3].
– orientations with respect to an implicit geo-referenced entity : *dans le Sud-Ouest*[4]*, dans le nord*[5] (implicit geo-referenced entity : France).

[2] in departments (administrative division of French territory) of the Parisian basin
[3] south and west departments of the Massif central
[4] in the south-west
[5] in the north

- orientations with respect to a geometric explicit or implicit construction: *une ligne Bordeaux-Genève*[6], *les départements de la diagonale Lorraine-Aquitaine toulousaine, de l'Allier au Pas-de-Calais*[7].
- more complex expressions combining previous forms: *des départements situés au nord d'une ligne Bordeaux-Genève*[8], *Paris et ses banlieues ouest et sud*[9].


"Même si l'indicateur de l'érosion des effectifs scolaires au cours du cycle d'enseignement en collège ...
... les scolarités en collège vont plus souvent à terme dans le Midi, en Bretagne, et en Ile-de-France;
et qu'au contraire, dans la plupart des départements situés au nord d'une ligne Bordeaux-Genève
la scolarité en collège tourne plus souvent court."

L'érosion des effectifs scolarisés de la sixième à la troisième en 1985-86

Fig. 2. Example of our purpose in a geographical context

From the viewpoint of spatial cognition and knowledge representation, geographic maps are common means of depicting world and have proved their usefulness over centuries. In fact this particular expression mode allows user to perform operations that can not easily be performed with a text. This different property may arise from similarities between maps and humans' internal representations of space [Ba98]. This medium as a form of representation imposes certain constraints on the content encoded in it. A map is characterized by: (1) having a two-dimensional spatial extent, (2) the presence of pictorial and/or

6 a Bordeaux-Genève line
7 from Allier to Pas-de-Calais
8 departments to the north of a Bordeaux-Genève line
9 Paris and its west and south suburbs

symbolic depictions which stand for known geographic entities in a given sub-set of the world, (3) only one or few chosen aspects of a phenomenon in a given scale are strictly represented.

So for a "correct" interpretation of a map it is necessary to distinguish: (1) which part of the world the map depict (for example France in the figure 2), (2) which phenomenon aspect(s) are pictorially/symbolically represented (in the figure 2 "the schooling fall from the sixth grade to the third grade between 1985 and 1986"), (3) what spatial behavior the author want to point out. This mean that from a processing point of view the perceptive information contained in a map has to be correctly extracted to deal with an adequate body of common geographic knowledge.

On the basis of these observations we can propose the following aspects for an automatic evaluation, this list is not exhaustive, but gives a selection of more common notions of what it could be extracted in a simple thematic map:

1. *context*: the textual parts may contains information about the geographical area, the scale and the phenomenon aspects.
2. *topological*: the states or regional boundaries or other administrative boundaries and other connected lines (roads, rivers, . . .) highlight spatial relations, especially topological ones but also some metric ones.
3. *orientation*: not only the default orientation of the map (generally the north) but also orientation with respect to some specific known named-entities or axes specified within the map and orientation of one place with respect to another one.
4. *focus*: the predominance (thematic or spatial or quantitative) of a given pictorial and/or symbolic depiction represent the current focus of the map.

4 Effectiveness of the Computational Model

4.1 Automatic Generation of Global Links

Explicit links are quite easy to find, authors use some linguistic expressions like: cf., see fig. XX, . . . Implicit links have not these expressions so we have studied our corpus to find how they work. Our study leads us to make the assumption that authors build implicit links using a same set of vocabulary both in comments and in graphics' captions (see dotted boxes figure 2). To test our assumption we have implemented a classical information retrieval method improved with a psycho-cognitive aspect. In order to validate our method we have asked five geographers of the CRESO[10] to play the role of a reader and manually tag the more apposite implicit links between text and graphs in a selected sub-set of our corpus. Then we compare the results of our automatic method with the results of the geographers. The figure 3 presents the comparison.

Information Retrieval Method. In a documents database for a given user's query a searching tool selects a set of documents pertaining to the query. We are in a similar context, instead of a database composed of documents, we have

[10] Centre de Recherche Espace et SOciété (University of Caen)

a document composed of paragraphs, instead of a user's query we have the set of lexical terms extracted from a selected picture's caption, so the searching tool will point out a paragraph or a set of paragraphs pertaining to the picture.

The retrieval method uses Salton's based word frequencies [Sal88]. But apply such algorithm on all the paragraphs of a document produce too much noise (see fig. 3a).

Psycho-Cognitive Aspect. To reduce this kind of noise a psycho-cognitive aspect has been integrated. Founded on the hypothesis that human has a quite low capacity to memorize (well known as "memory factor") we have deduced that a picture can't be too far from its comment. So we have weighted retrieval method response according to the distance (in terms of number of pages) between the picture and paragraphs candidates to be picture's comment.

Thus the response is weighted with a Gaussian function, the more the distance is great, the more the weight will be near from zero. In figure 3b some results show that the link created with the Salton's retrieval method added with "memory factor" matches more better with the geographers' links.

Globally the results are good, some blanks or wrong response are still not solved because of problems like synonyms, height similarities in captions of some graphics or particular graphics which comment is included (in this case equivalence link is not present in the text).

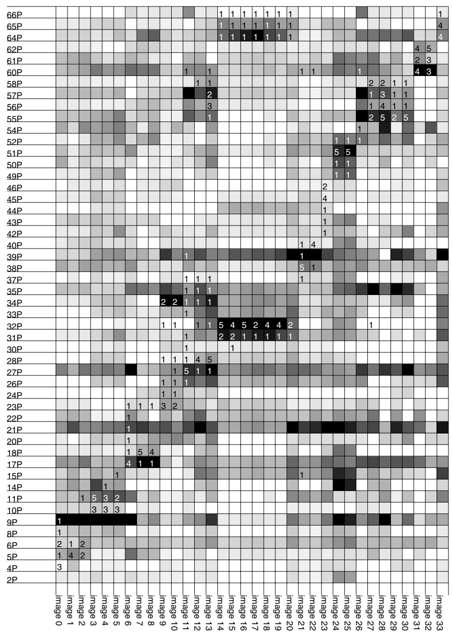
4.2 Automatic Generation for Minimal Links

As we have seen it before in geographical context, the smallest information are Geo-referenced Entities (GE) —i.e., from the text point of view: linguistic expressions in paragraphs and from the graphical one: objects gathered by a quite common pictorial and/or symbolical representation (like for instance same tones of color).

From the Text Point of View. So firstly we have studied the linguistic expressions referring to geo-referenced entities (GE). We have classified the geo-referenced entities into seven categories [NM99] and we have proposed a semantic model to represent their meaning. Then we have built a rule system to extract those entities from the text, each class corresponding to a set of rules. The rules are launched when a key word is detected [NM99]. The key words are specific lexical terms of the geographical domain like : *département* (administrative division of French territory), *région* (region), *banlieue* (suburbs).

The output of this tool is a tagged text in a XML[11] format as shown in the two examples below:

[11] eXtended Mark-up Language


(a) without psycho-cognitive aspect

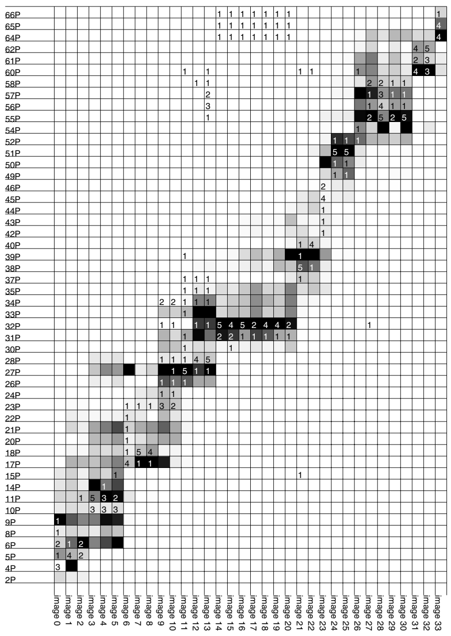

(b) with psycho-cognitive aspect

Fig. 3. Global Link Results: Values in the cells show the number of geographers who have marked the link between a paragraph (numbered on the left) and a picture (numbered at the bottom). The gray scale level indicate the results of the automatic method. Darker cells indicate the higher probability for a link to be correct.

les départements au nord de Paris[a]

```
<GE>
  <GRANULARITY name=‘‘department’’>
    <INDIRECTION>
      <DIR orientation=‘‘90’’/>
      <DGE name=‘‘Paris’’/>
    </INDIRECTION>
  </GRANULARITY>
</GE>
```

90 corresponds to 90 degree in relation with the interpretation of the north.

[a] departments in the north of Paris

au nord d'une ligne Bordeaux-Genève [a]

```
<GE>
    <INDIRECTION>
      <DIR orientation=‘‘90’’/>
      <GEOMETRIC type=‘‘line’’>
        <DGE name=‘‘Bordeaux’’/>
        <DGE name=‘‘Gen\‘eve’’/>
      </GEOMETRIE>
    </INDIRECTION>
</GE>
```

<DGE> represents a geographic entities directly named (in opposition to particular spatial concept) in the corpus and having their boundaries in our GIS (Geographical Information System).

[a] in the north of a Bordeaux-Genève line

After the extraction of a GE from the text and its representation in the semantic model, we use a GIS[12] to build a graphical representation of the GE's meaning. We have created some rules to interpret the XML extraction and to delimit some regions in the map.

For the moment we make the interpretation of EGs on an empty map similar to the map associated (Global Link) to the comment which contains the EGs. Later that empty map will be superposed to the real map to compare with the graphical analysis of the graphs (see below).

The figure 4 shows an example about the selection of departments in the north of another department (D). Firstly, we build some areas (fig. 4a) around the department D. The dark area corresponds to an area where we are sure it is the north, white areas correspond to areas where it can be the north but it is not certain. Secondly, we select the department in the areas (fig. 4b) in this example we have selected departments which intersections with areas are greater than 40% (more details will be available in [Mal00]). The figure 5 page 136 gives some examples of interpretations.

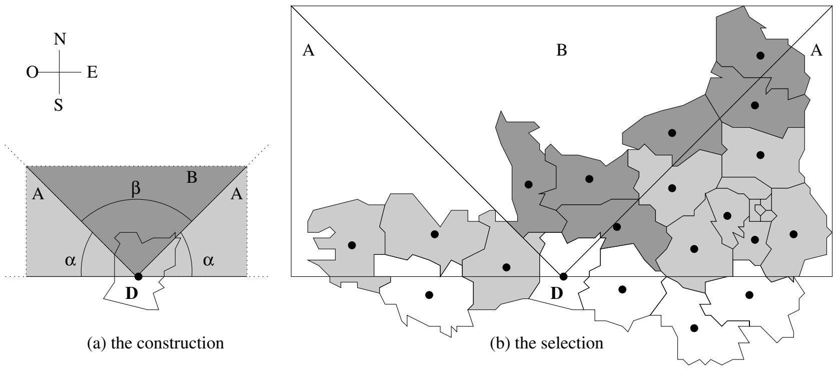
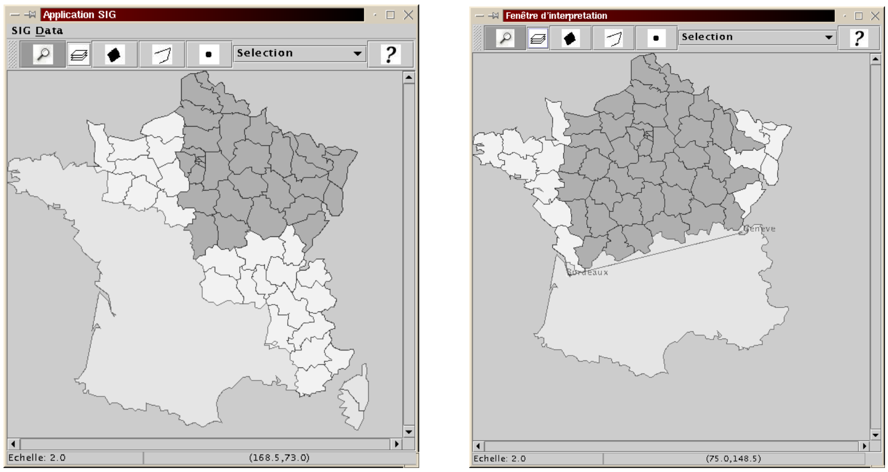


Fig. 4. Building of the north of a department

From the Graphical Point of View. Maps allow to accomplish in a much more 'natural' and easy way than text some operations like representing inter-object and intra-object spatial relationships at different levels of granularity, this benefit is due to the topological and geometric properties of the planar space [Ber67]. As a part of computer science, the processing of diagrammatic or pictorial representations are explored areas (for a collection of research papers see [Ga95]).

As a first exploratory implementation we have developed a basic image processing tool that demarcate in context relevant parts of the picture [Fau00]. In a geographic map context the tool extract marked localizations and/or administrative boundaries gathered by a common representation (quite nearest tones of

[12] Geographical Information System



(a) north-east of France

(b) north of a Bordeaux-Genève line

Fig. 5. Interpretation: Darker departments have a higher probability to belong to the interpretation, lighter departments have a lower probability.

a color or quite same textures, . . .), see in fig 6 page 137 most of north-eastern administrative entities could be gathered together because of a quite common tone of gray and also the southern entities in the map of the figure 2 page 131.

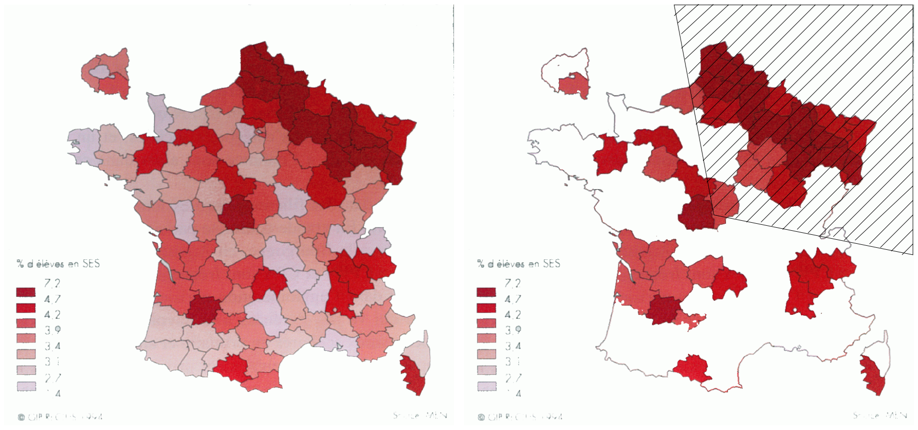
In the comment of the geographical map (fig. 6a page 137) the author cites "the north-east of the France" to point out the area in the map. By combining graphical representation of the interpreted linguistic expression (see hatched areas in fig. 6b) and the response of the geographical map processing (see map in fig. 6b), the system can point out which are relevant parts of the picture that match (cf. intersection between hatched area and the map processing in fig. 6b page 137).

5 Conclusion

In this work, we have presented a method and described the process for automatically link text and graphics (especially maps) in a geographic context with a "multi-scaling" linking artifact. Using standard Information Retrieval techniques added with a "psycho-cognitive" like aspect, and local natural language processing, we have shown that the method is computational effective.

The centerpiece of the implementation is the text processing tool, we have described aspects to model the whole processing of geographical maps.

We do not claim to have a powerful solution for finding all "implicit" links. Information retrieval techniques are perfectible the same remarks can be applied



(a) Graphical map associated to a comment in which the author cites "le nord-est de la France" (the north-east of the France)

(b) Intersection between relevant parts of the picture obtained with the geographical map processing tool and the hatched area obtained with the interpretation tool of extracted EGs (EG : "le nord-est de la France")

Fig. 6. Map and Text analysis

for local natural language processing. What our method point out is that a the double processing text and map is able to handle some fine distinctions in meaning improving the quality of the results.

Our perspectives are in the short-term to improve the Global Link generation taking into account more precisely the distance between graphs and paragraphs, as for example : is the graph above a paragraph ?, under a paragraph ?, . . . But the most interesting work in progress is the development of a common formal representation that allowed to described both analysis and interpretations of comments and maps in a same framework.

References

[All96] James Allan. Automatic hypertext link typing. In ACM, editor, *Proceedings of Hypertext'96 conference*, pages 42–52, Mars 1996.

[Ba98] Berendt B. and alter. Spatial representation with aspect maps. In Freksa C., Habel C., and Wender K.F., editors, *Spatial Cognition : An Interdisciplinary A pproach to Representing and Processing Spatial Knowledge*, Lecture Notes in Artificial Intelligence. Springer, 1998.

[Ber67] Jacques Bertin. *Sémiologie Graphique.* Mouton & Cie, 1967.

[Den89] Michel Denis. *Image et Cognition.* 1989.

[Fau00] Eric Faurot. Analyse de carte géographique : Extraction de zones pertinentes. Mémoire de dea, GREYC - University of Caen, November 2000. to appear.

[Ga95] J. Glasgow and alter. *Diagramming reasoning: Computational and cognitive perspectives.* MIT-Press, 1995.

[Jol93] Martine Joly. *Introduction à l'analyse de l'image.* NATHAN Université, 1993.

[Mal00] Nicolas Malandain. *Mise en relation Texte/Image, Application aux Documents Géographiques.* PhD thesis, University of Caen, 2000. to appear.

[NM99] Malandain Nicolas and Gaio Mauro. Extraction d'unités d'informations géographiques dans des documents composites. In *Document Électronique Dynamique*, pages 45–58. Proceedings of CIDE'99, Europia, 1999.

[RR94] Hérin Robert and Rémi Rouault. *Atlas de la France Scolaire de la maternelle au lycée.* Collection Dynamiques du territoire, 1994.

[Sal88] Gerard Salton. *Automatic Text Processing.* Addison-Wesley, 1988.

[TW87] Randall H. Trigg and Mark Weiser. Textnet: A network-based approach to text handling. In ACM, editor, *ACM Transactions on Office Information Systems*, pages 1–23, January 1987.

[VDP91] H. Van Dyke Parunak. *Ordering the information graph.* Intertext Publications, Mc. Graw-Hill Publishing Company, 1991.

[Za98] Hubert D. Zimmer and alter. The use of locative expressions in dependence of the spatial relation between traget and reference objet in two-dimensional layouts. In Fresksa C., Habel C., and Wender K.F., editors, *Spatial Cognition : An Interdisciplinary A pproach to Representing and Processing Spatial Knowledge*, Lecture Notes in Artificial Intelligence. Springer, 1998.

GODDAG: A Data Structure for Overlapping Hierarchies

C.M. Sperberg-McQueen[1] and Claus Huitfeldt[2]

[1] World Wide Web Consortium / MIT Laboratory for Computer Science
`cmsmcq@acm.org`

[2] University of Bergen / Humanistic Information Technology Centre
`Claus.Huitfeldt@hit.uib.no`

Abstract. Notations like SGML and XML represent document structures using tree structures; while this is in general a step forward from earlier systems, it creates certain difficulties for the representation of documents in which the structures of interest are not properly nested. Overlapping structures, discontinuous structures, and material which occurs in different orders in different parts, views, or versions of a document are all problems for SGML and XML. Overlapping structures have received attention from a variety of authors on SGML and XML, who have proposed various solutions including the use of non-SGML notations with translation into SGML for processing, the use of the CONCUR feature of SGML, exploitation of conditional marked sections in the DTD and document instance, the imposition of various kinds of unusual interpretations on SGML/XML elements as milestones or as fragments of some larger 'virtual' element, or the use of detailed annotation separate from the base text being annotated.

An alternative is the use of a non-SGML/XML notation which does not require that elements form a hierarchical structure. One such notation, MECS, was developed by one of the authors and has been used in practice for over a decade. Unfortunately, the element structure of a MECS document cannot conveniently be represented as a tree, so that MECS processors lack the assistance provided to SGML/XML processors by the unifying assumption of a simple standard data structure for the document. We propose a data structure for representing documents with overlapping structures (including MECS documents). As in the conventional tree representation of SGML and XML, elements are represented by nodes in a graph, and the character data content of the document by labels on the leaves of the graph. We use a directed acyclic graph in which an arc $a \rightarrow b$ indicates that node b is a child of node a. Unlike SGML/XML trees, our graph structure allows children to have multiple parents. In the general form of the data structure, an ordering is imposed on the children of each node; this gives the data structure its name: general ordered-descendant directed acyclic graph (GODDAG). A restricted form of GODDAG, in which an ordering is imposed on the leaves of the graph, cannot handle multiple orderings of the same material but can represent any legal MECS document.

The data structure here proposed should be useful in the representation of naturally occurring documents with complex structures; it may also be useful in other applications.

1 The Problem of Overlap

1.1 Document Structure and Interpretation

Those concerned with digital documents are increasingly accustomed to viewing documents as having a hierarchical structure. Such a structure lends itself to electronic representation using a tree structure, in which each node in the tree is labeled to show the kind of document component represented by that subtree. The tree structure can in turn be linearized using, for example, the syntax of SGML [6] or XML [4], as in this example (from Bashō):

```
<lg>
<l>Summer grass -</l>
<l>all that's left</l>
<l>of warriors' dreams.</l>
</lg>
```

Here, an *<lg>* (line group) element contains three *<l>* elements, each *<l>* element representing a verse line, and the *<lg>* element representing the entire haiku.

We can represent the tree graphically as shown in Fig. 1.

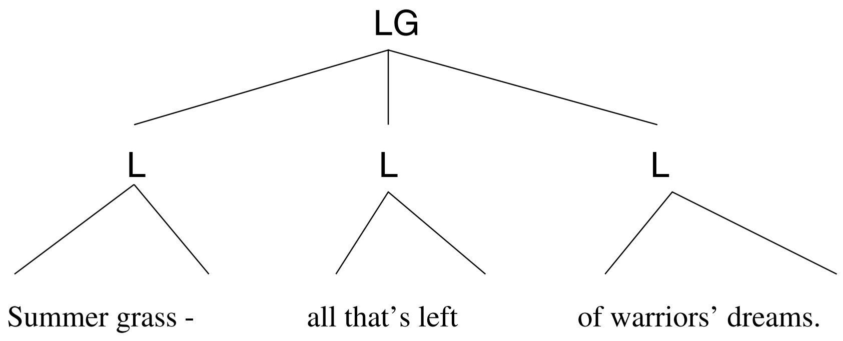


Fig. 1. A simple document hierarchy

A slightly more complex example from the posthumous notebooks of Ludwig Wittgenstein (VW 115, p. 7, Fig. 2) shows how the nodes of the tree can be associated with different types of components: here, a *<p>* (paragraph) element (*p*) contains two sentences (<*s*> nodes); within one of them, some words have been deleted (<*del*>) and others added (<*add*>).[1]

```
<p><s><del>Der Anblick</del> <add>Das Bild</add>
<del>der</del><add>einer</add> menschlichen Gestalt sowie
die menschliche Gestalt selbst sind uns wohlvertraute
Gegenst&auml;nde.</s>
<s>Von einem Wiedererkennen aber ist hier keine Rede.</s></p>
```

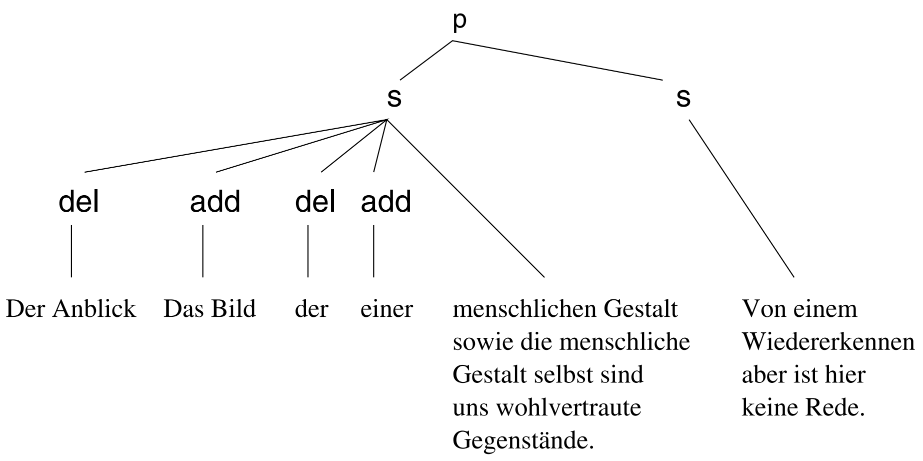

p
s
s
del
add
del
add
Der Anblick
Das Bild
der
einer
menschlichen Gestalt sowie die menschliche Gestalt selbst sind uns wohlvertraute Gegenstände.
Von einem Wiedererkennen aber ist hier keine Rede.

Fig. 2. Wittgenstein ms 115, p. 7

The start-tags mark the beginning of a passage characterized by some feature (e.g. being a paragraph, or having been deleted), the matching end-tags mark the end of the passage. In a markup language with conventional interpretation, the label on each node in the tree identifies some feature possessed by the subtree it dominates. Sometimes the feature is possessed by the subtree as a unit, as with the *p* element here; sometimes it is inherited by each part of the subtree, as with the *del* and *add* elements here.

To find all the characteristics attributed to a given passage by the markup of the document, it suffices to walk the tree from leaf to root, collecting predicates.[2]

[1] The transcription here is at the Wittgenstein Archive at the University of Bergen, but the markup has been translated into the format defined by the Text Encoding Initiative.

[2] We simplify here by ignoring markup intended to modify the meaning of other markup, such as the TEI [1] <*certain*> element, the distinction between the properties of a whole and its constituent parts, and other complications. For further discussion of the meaning and interpretation of markup, see [10].

1.2 Overlap

In some texts, however, it is difficult to see how to construct a tree structure which represents all the information of interest in a natural way. Consider, for example, the first lines of Ibsen's *Peer Gynt* I.i:

> AASE: Peer, du lyver!
> PEER GYNT : Nej, jeg gjør ej!
> AASE: Naa, saa band paa, det er sant!
> PEER GYNT: Hvorfor bande?
> AASE: Tvi, du tør ej!
> Alt ihob er Tøv og Tant!

In English:

> AASE: Peer, you're lying!
> PEER GYNT: No, I'm not!
> AASE: Well then, swear to me it's true.
> PEER GYNT: Swear? why should I?
> AASE: See, you dare not!
> Every word of it's a lie.

Dramatically, this passage consists of five speeches, which might be encoded thus:

```
<sp who="Aase"> Peer, you're lying! </sp>
<sp who="Peer"> No, I'm not! </sp>
<sp who="Aase"> Well then, swear to me
                   it's true! </sp>
<sp who="Peer"> Swear? why should I? </sp>
<sp who="Aase"> See, you dare not!
                Every word of it's a lie!
</sp>
```

Metrically, by contrast, it consists of four lines of verse:

```
<l> Peer, you're lying!  No, I'm not! </l>
<l> Well then, swear to me it's true! </l>
<l> Swear? why should I?  See, you dare not! </l>
<l> Every word of it's a lie! </l>
```

Each of these two views of the material corresponds to a different tree structure. The dramatic structure is shown in Fig. 3, the metrical structure in Fig. 4.

For simplicity of interpretation, it might be best simply to combine these two trees, as in Fig. 5. Unfortunately, the resulting graph is not a tree, and has no natural representation as a legal SGML or XML document.

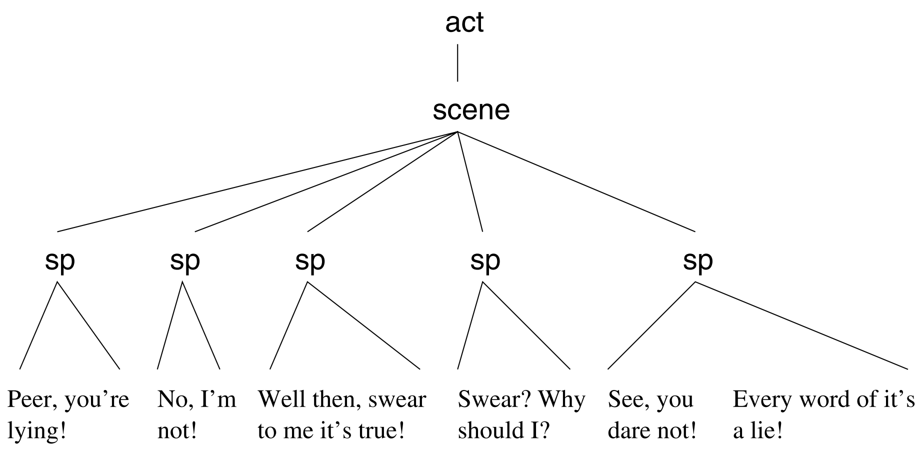


Fig. 3. Tree of dramatic view

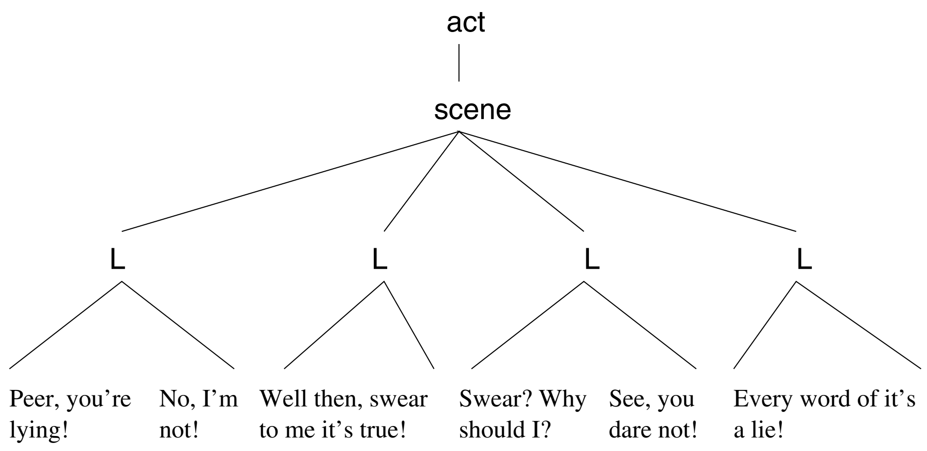


Fig. 4. Tree of metrical view

2 Dealing with Overlap

Several methods have been proposed in the literature for dealing with the problem of overlap.

2.1 Non-SGML Notations

In many ways the simplest approach is to use some notation other than SGML or XML.

In an early article on the problem of overlap, Barnard et al. [2] note that one method of handling overlap is simply to ignore the rules of SGML and XML for

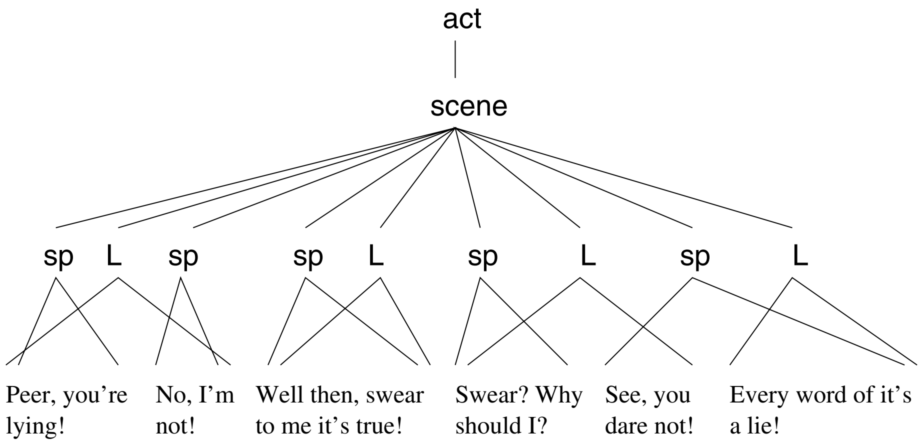


Fig. 5. Tangled hierarchy

nesting; such an approach to our example would produce an encoding something like this:

```
<l>
<sp who="Aase"> Peer, you’re lying! </sp>
<sp who="Peer"> No, I’m not! </sp>
</l>
<l>
<sp who="Aase"> Well then, swear to me
                   it’s true! </sp>
</l>
<l>
<sp who="Peer"> Swear? why should I? </sp>
<sp who="Aase"> See, you dare not!
</l>
<l>
                Every word of it’s a lie!
</l>
</sp>
```

The failure of the elements to nest properly makes it impossible to create an SGML document type definition (DTD) for the document, and thus impossible to use SGML or XML tools to process it.

Barnard et al.'s second proposal is to keep both sets of elements in a master version of the document, and to apply filters to the document before feeding it into an SGML system. Filtering out the start- and end-tags for <*l*> elements, for example, yields a dramatic structure that can be parsed with SGML or XML processors; filtering out the <*sp*> tags (and optionally the <*stage*> elements) similarly yields the verse structure. While this approach has certain virtues of simplicity, it does have some disadvantages. It is not possible to validate the master version of the document using standard tools. As Barnard et al. point out, it requires ad hoc filter programs outside the SGML/XML environment. Because the SGML/XML form is not the master form, it is difficult to apply standard SGML/XML tools for editing the document. And since the SGML/XML processors only see one structure at a time, it is impossible to examine both structures and their interaction, unless ad hoc tools are written for the task. (Barnard et al. do not mention this, and apparently do not regard it as a problem.) Perhaps for these reasons, the filtering approach described by Barnard et al. seems not to have been adopted by any software developers or text encoding projects.

2.2 CONCUR

In SGML, an optional feature named CONCUR is provided, which allows for "multiple concurrent structural views" of the document (ISO 1986 sec. C.3.1). Barnard et al. 1988 [2] mention CONCUR as a possible solution, as do Barnard et al. 1995 [3], and more recently Sperberg-McQueen and Huitfeldt [9] have actively encouraged its use.

The example from Peer Gynt might look like this, if tagged using concurrent markup:

```
<div1 type="act">
<(D,V)div2 type="scene">
<(V)l>
<(D)sp who="Aase"> Peer, you're lying! </(D)sp>
<(D)sp who="Peer"> No, I'm not! </(D)sp>
</(V)l>
<(V)l>
<(D)sp who="Aase">
                   Well then, swear to me
                   it's true! </(D)sp>
</(V)l>
<(V)l>
<(D)sp who="Peer"> Swear? why should I? </(D)sp>
<(D)sp who="Aase"> See, you dare not!
</(V)l>
<(V)l>             Every word of it's a lie!</(D)sp>
</(V)l>
...
</(D,V)div2>
</div1>
```

Note that each tag carries, in parentheses, the names of the root elements of the concurrent document types to which it applies; when the tag belongs to all concurrent views, the list may be exhaustive, as shown in the <*div2*> tags, or omitted, as shown in the <*div1*> tags.

The tagging strongly resembles the non-nesting markup described above, but because each document type (here both D and V) has a tree structure, each can be given a full document-type definition, parsed, and interpreted in a straightforward way using the normal conventions of SGML and XML. The only difference is that instead of one tree for the document, there are multiple trees, sharing the same frontier. (If entity declarations differ in the different DTDs, the trees will have slightly different frontiers.)

The CONCUR feature, however, is not widely regarded as a solution for the overlap problem. It is clear from the SGML standard (ISO 8879) and the commentary of its editor [5] that the developers of SGML expected this to arise primarily in cases where it was desirable to record both the source and the result of some processing (e.g. both the logically tagged source and the formatted output of a layout program) in the same document; Goldfarb writes [5] (p. 304): "I therefore recommend that CONCUR not be used to create multiple logical views of a document, such as verse-oriented and speech-oriented views of poetry. Hypertext links are more appropriate for such applications.. . . " Barnard et al. 1988 [2] object to CONCUR because it is not widely implemented, because it increases the volume of markup in the document, and because it has complex interactions with various markup-minimization features. The rejection of all markup-minimization features in XML has in the meantime robbed the last two objections of most of their force, but the lack of implementation support has been an incentive for practitioners to develop other methods of handling overlap. More substantial reservations about CONCUR are those expressed by Murata [8]: CONCUR is not able to model, in any convenient way, the deletion, insertion, duplication, or reordering of data in the various views.

2.3 Marked Sections and Entity References

Barnard et al. 1988 [2] propose to handle overlap with an elaborate scheme involving marked sections and specially defined entities, which will make either the tags for dramatic structure, or those for metrical structure, visible to an SGML parser, but not both. In some ways, this method resembles their proposal to maintain markup for both hierarchies in the master copy of the document, and filter the master into a single-hierarchy form for processing, with the difference that the filtering is here handled within, rather than outside, the SGML system.

Like the filtering approach, this method provides access to one hierarchical structure at a time, but not to both at once. This drawback, coupled with the lack of support for this style of markup in SGML editors, may be responsible for the fact that this proposal seems to have had little or no effect on the practice of those working with SGML systems.

This technique is not available in XML systems, because it relies on conditional sections in the document instance, and on entity replacement texts containing start- or end-tags, but not both. XML does not allow conditional sections in the instance, and it requires entity replacement text to be well-formed.

2.4 Milestones

The technique we call *milestone elements* was proposed by Barnard et al. in 1988 [2]; the term itself comes from the *Guidelines* of the Text Encoding Initiative (TEI) [1], which uses this technique for marking pagination, typographic lineation, and other asynchronous phenomena.

The basic idea is simple: of the various competing hierarchies implicit in an overlapping structure, one is chosen as the dominant hierarchy, and the start and end of what would have been elements in the other hierarchies are represented using empty elements. In sequential processing, the processor must change state as it passes such an element, in much the same way as our estimate of the distance remaining to a destination changes when we pass a milestone by the road.

Our example from *Peer Gynt* might be represented this way in XML, with line breaks in the metrical view represented as milestone elements named *lb* ('line break'):[3]

```
<sp who="Aase"><lb n="1"/>
  Peer, you're lying! </sp>
<sp who="Peer">
  No, I'm not! </sp>
<sp who="Aase"><lb n="2"/>
  Well then, swear to me it's true! </sp>
<sp who="Peer"><lb n="3"/>
  Swear? why should I? </sp>
<sp who="Aase">
  See, you dare not!
<lb n="4"/>
  Every word of it's a lie!
</sp>
```

In some cases (as described in Barnard et al. 1995 [3]), it may be desirable to provide empty elements both for the start- and for the end-tags of the elements in the non-dominant views of the text; in this particular example, it is unnecessary as each line ends only when the next begins.

The milestone technique, while convenient for data capture and simple sequential processing of the document, requires foregoing any validation of the subordinate views by the SGML or XML parser. The interpretation and processing required for milestone elements differs substantially from that for other element types. The tree structure associated with the XML fragment just given represents the dramatic structure normally, but the metrical structure must be, in effect, decoded rather than being exhibited directly by the tree structure.

[3] Strictly speaking, since the TEI [1] <*lb*> element is intended to mark typographic lines, not metrical lines, the usage shown here constitutes mild tag abuse.

2.5 Fragmentation

It is possible to gain slightly better validation of the markup, and to reduce the divergence in interpretation and processing needed for speeches and metrical lines, if we use the technique of *fragmentation* introduced (as far as we know) by the TEI *Guidelines* [1].

As with milestone elements, one hierarchy is chosen as the dominant one. Elements in the other hierarchies are represented normally as elements, as long as they nest properly within elements of the dominant hierarchy. Where an element in a subordinate view spans an element boundary in the dominant view, that element is broken into as many pieces as necessary to make each piece nest within an element of the dominant hierarchy. Attributes are used to signal that the resulting elements are, from a logical point of view, only fragments of a larger virtual element. The TEI encoding of our lines from Ibsen uses fragmentation of the verse lines:

```
<sp who="Aase">
  <l part="i">Peer, you're lying!</l>
</sp>
<sp who="Peer">
  <l part="f">No, I'm not!</l>
</sp>
<sp who="Aase">
  <l part="n">Well then, swear to me it's true!</l>
</sp>
<sp who="Peer">
  <l part="i">Swear? Why should I?</l>
</sp>
<sp who="Aase">
  <l part="f">See, you dare not!</l>
  <l part="n">Every word of it's a lie.</l>
</sp>
```

In the TEI DTD, the *part* attribute is used to specify that an <*l*> element represents a complete metrical line (`part="n"`), the initial part of the metrical line (`part="i"`), a middle part (`part="m"`), or the final part of the metrical line (`part="f"`). Applications which want to reconstruct fragmented metrical lines take any <*l*> element not marked as partial, or any sequence of <*l*> elements consisting of one initial, zero or more medial, and one final part. The SGML or XML parser does not ensure that initial, medial, and final parts occur in a proper order, which means some application-specific validation is necessary, but the parser does ensure proper nesting of <*l*> elements and proper matchup of start- and end-tags, which makes fragmentation frequently preferable to milestone elements.

It is worth noting that fragmentation provides a convenient method of handling discontinuous segments of a text (such as direct discourse of quotations interrupted by attributions), but no way of handling material which appears in other than the logical order.

2.6 Standoff Markup

A final method of handling overlap is to use what the TEI *Guidelines* call "out-of-line markup", and what McKelvie et al. [7] (sec. 3.1) call "standoff annotation".

In standoff markup, hierarchical structures may be represented by markup stored separately from the content of the structures, and connected to the content by hyperlinks. This allows the representation of multiple conflicting hierarchical structures, as well as allowing annotation of read-only material. The TEI <*join*> element allows us to represent the metrical lines of our example as virtual elements created by combining the line fragments actually encoded:

```
<sp who="Aase">
  <l id="L1a">Peer, you're lying!</l>
</sp>
<sp who="Peer">
  <l id="L1b">No, I'm not!</l>
</sp>
<sp who="Aase">
  <l id="L2">Well then, swear to me it's true!</l>
</sp>
<sp who="Peer">
  <l id="L3a">Swear? Why should I?</l>
</sp>
<sp who="Aase">
  <l id="L3b">See, you dare not!</l>
  <l id="L4" >Every word of it's a lie.</l>
</sp>
```

The <*join*> elements themselves each signal the existence of a virtual element of type <*l*>, created by concatenating the elements pointed at by the *targets* attribute:

```
<joinGrp result="l" targOrder="y" targType="L">
<join scope="branches" targets="L1a L1b"/>
<join scope="branches" targets="L3a L3b"/>
</joinGrp>
```

In practice, standoff markup plays a relatively minor role in the TEI DTD, primarily because it places a relatively heavy burden on data capture and on processors. A much more systematic exploration of the idea is given by McKelvie et al. [7], who describe a system in which various kinds of markup are systematically encoded and stored separately from the material they annotate, in order to allow multiple analyses of the same material. A tool suite is available for combining the base material with the standoff markup, in a way roughly analogous to that suggested by Barnard et al., except that instead of stripping out markup not relevant to the particular view desired by the application, the tools for standoff markup *introduce* the markup relevant to the desired view.

Standoff markup can handle both discontinuous virtual elements and virtual elements whose content appears out of order in the document. It suffers, in the view of some, from the lack of any convenient way to test, when the markup is stored in the standoff format, whether the markup actually obeys any constraints expressed in a DTD or other notation.

2.7 MECS

All the SGML/XML-based methods of handling overlap described so far have in common that they are more or less awkward. Some require applications to see one or the other, but not both, hierarchical structures; the others privilege one structure by making it the dominant hierarchy and requiring various forms of special processing, and special rules for interpreting the tree structure of the document.

One way to avoid the awkwardness is to take seriously the first method of handling overlap proposed by Barnard et al. 1988: use a non-SGML notation which does not require that elements nest. While there is no very widely used notation of this type — unless one counts the ill-formed HTML generated by users or programs unfamiliar with SGML and supported by the lenient error handling of HTML browsers — the MECS (Multi-Element Code System) notation developed by the Wittgenstein Archive at the University of Bergen provides a useful way to gain most of the benefits of descriptive markup without requiring proper nesting of elements. MECS was developed for the transcription of Wittgenstein's unpublished notebooks; it is essential to provide a clear account of the physical organization of the page, as well as of the logical structure of the text, without privileging either.

In MECS,

1. start-tags (in the form `<e/`) mark the start of a feature
2. end-tags (`/e>`) mark the end of a feature
3. no-element codes (`<e>`) mark a feature with location but no content (these correspond to SGML *empty* elements)

There are some other convenience features, which we will not describe here.

Elements are not required to nest. In MECS notation, our example might look like this (with some extra white space):

```
<sp/<speaker/Aase/speaker>
<l/ Peer, you're lying!          /sp>
<sp/<speaker>Peer/speaker>
    No, I'm not!            /l>  /sp>
<sp/<speaker/Aase/speaker>
<l/ Well then, swear to me it's true! /l>
/sp>
```

```
<sp/<speaker/Peer/speaker>
<l/ Swear? Why should I?         /sp>
<sp/<speaker/Aase/speaker>
    See, you dare not!       /l>
<l/ Every word of it's a lie! /l>  /sp>
```

MECS provides a simple notation, which ensures (as does XML) that no DTD is required in order to read the document correctly, and which (unlike SGML and XML) makes it possible to ensure a one-to-one mapping between features in the text and start-/end-tag pairs in the encoding.

MECS does, however, have some problems: it is entirely possible to encode texts with gratuitous overlap of elements which has no ascertainable meaning, it lacks any notation for cases in which two elements of the same type overlap each other, its element structure cannot be represented as a tree, and it provides no formalism for defining a document grammar analogous to a DTD.

The rest of this paper is devoted to defining a data structure intended to stand in the same relation to MECS as trees stand in relation to SGML and XML documents.

3 GODDAG

3.1 Informal Presentation

Let us begin with a trivial example of a document with overlapping structures:

```
<s/<a/ John <b/ likes /a> Mary /b>/s>
```

If for this example we draw the containment relations among the $<s>$, $<a>$, and $<b>$ elements in the usual way, ignoring for the moment the lack of nesting, the result may be a graph like that shown in Fig. 6.

If we add explicit nodes for the #PCDATA chunks in the document, the graph becomes more manageable (Fig. 7).

Overlap can be represented by graphs that are very like trees, but in which nodes may have multiple parents.

That is: overlap is multiple parentage.

Applying this idea to our *Peer Gynt* example yields the graph shown in Fig. 8.

In such a graph, we can apply many of the conventions usual in the construction and interpretation of trees representing document structures: nodes have containment relations, each node can be said to dominate some subgraph (the nodes reachable from it), one node dominates a second if and only if the second node is completely contained within the first, the features associated with nodes by the labels on the nodes are possessed by the subgraphs dominated by the nodes, etc.

The following paragraphs provide a slightly more formal characterization of such graphs, describe algorithms for constructing them, and briefly discuss some applications.

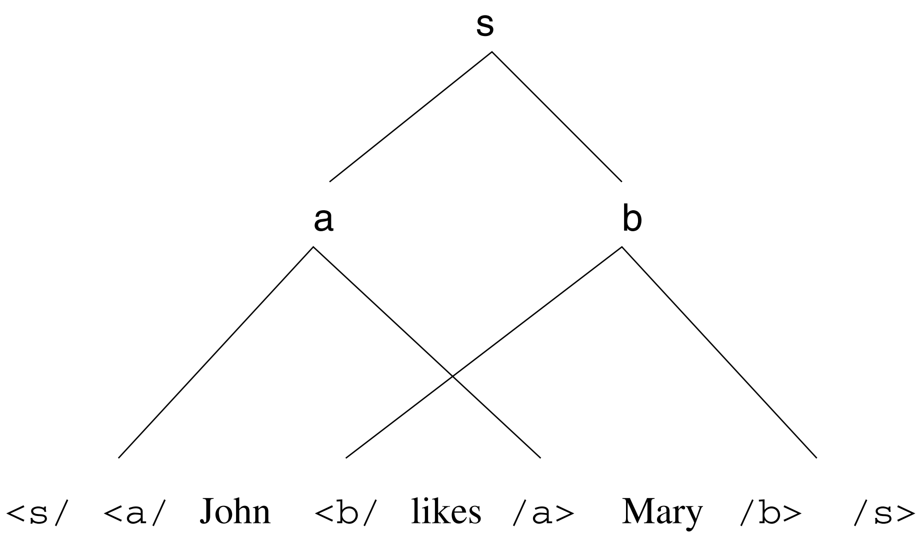


Fig. 6. A trivial overlap example: John likes Mary

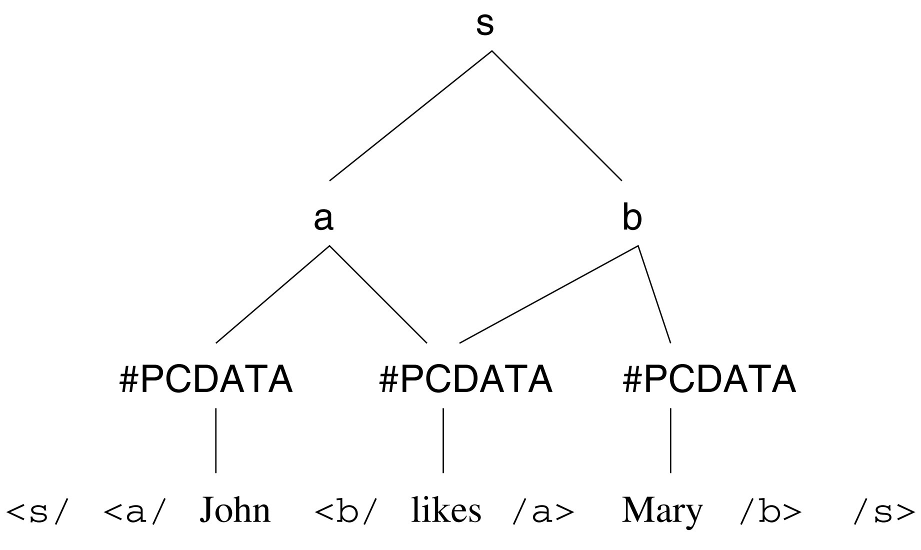


Fig. 7. Another view of John likes Mary

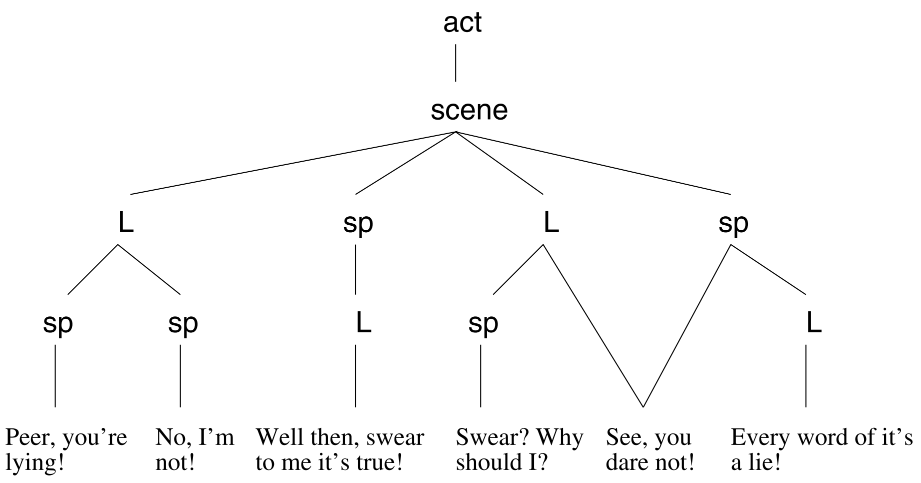


Fig. 8. *Peer Gynt* 1.1

3.2 Restricted GODDAGs

We distinguish two varieties of graph for representing documents with overlap: a restricted form which directly corresponds to MECS documents, and a generalized form which may not always have a representation in MECS.

Because nodes in such graphs have descendants, and because those descendants are ordered, we refer to these graphs as generalized ordered-descendant directed acyclic graph (GODDAG) structures.

In each case,

1. A GODDAG is a directed acyclic graph in which each node is either a leaf node or a nonterminal node.
2. Each leaf node is labeled with a string of zero or more characters.
3. Each nonterminal node is labeled with a generic identifier.
4. Directed arcs connect nodes; if an arc $a \rightarrow b$ exists, we say that node a *directly dominates* node b; we also call b a *child* of a. If a directed path of length zero or more from a to b exists, we say that node a *dominates* node b. If the path is of length one or more, then we call b a *descendant* of a.
5. Leaf nodes are those which are not the starting point of any arc.
6. Nonterminals are all others.
7. For any three nodes a, b, and c, if arcs $a \rightarrow b$ and $b \rightarrow c$ exist, then there is not an arc $a \rightarrow c$. This amounts to saying that no node dominates another node both directly and indirectly.

In restricted GODDAGs, the following constraints apply:

1. The leaf nodes form a sequence. For any two leaf nodes a and b,
   (a) $a < b$ (in which case we say that a precedes b in the document), or
   (b) $a > b$ (and b precedes a), or

   (c) $a = b$ (in which case they are the same node)

   Each leaf node except the first has a predecessor, and each leaf node except the last has a successor. The sequence of leaf nodes we call the *frontier*.
2. Each node dominates some contiguous subsequence of leaf nodes. I.e. if node n dominates leaf nodes a and b, and $a < b$, then for any leaf node c such that $a < c < b$, node n dominates c.
3. No two nodes dominate the same subsequence of the frontier. I.e. for any two nodes a and b, there will be at least one leaf node c such that a dominates c but b does not dominate c, or vice versa.

In generalized GODDAGs, these constraints are relaxed. (See below.)

From a MECS document, a restricted GODDAG structure can be constructed as follows. The procedure requires a list of open elements, which is empty at the beginning of the procedure. We conceive of the MECS document as a series of *segments*, where each segment is either a single tag, or else the sequence of zero or more data characters between two tags. Tag segments and character data segments alternate: between any two tag segments there is at least one character data segment, and vice versa.

To construct a restricted GODDAG we read and process one segment s of the document at a time.

1. If s is a start-tag, then
   (a) Make an element node n labeled with the appropriate generic identifier
   (b) For each element node e in the list of open elements, add an arc $e \rightarrow n$.
   (c) Add n to the list of open elements.
2. If s is a character data segment, then
   (a) Create a PCDATA node p labeled with the appropriate string.
   (b) For each element node e in the list of open elements, add an arc $e \rightarrow p$.
3. If s is an empty-element tag, then
   (a) Make an element node n labeled with the appropriate generic identifier
   (b) For each element node e in the list of open elements, add an arc $e \rightarrow n$.
   (c) Make a PCDATA node p labeled with the empty string.
   – Add an arc $n \rightarrow p$.
4. If s is an end-tag, then
   (a) Find the element node n created for the corresponding start-tag, and remove n from the list of open elements.
   (b) If n is not the most recently created element node in the list of open elements, then for each node m which was created more recently than n remove the arc $n \rightarrow m$. These elements are not wholly contained in n and thus should not be dominated by n.
   (c) Record the set of nodes dominated by n for later use. (This is the set of nodes x for which an arc $n \rightarrow x$ exists.)

   At this point, there may be illegal arcs from n to other nodes (n may be dominating some other nodes both directly and indirectly). We remove them thus:

   (a) For each descendant d of n, examine the set of nodes dominated by d. For each node m dominated by d, remove the arc $n \rightarrow m$ if it exists.

Note that in following this algorithm,

1. Every node created is labeled either with a string or with a generic identifier.
2. Every node labeled with a generic identifier is given at least one outgoing arc; no node labeled with a string is given any outgoing arc.
3. At completion of the end-tag processing for any node n, any arcs $n \to d$ have been removed if d is dominated by any descendant of n. Therefore, for each node n there is no node which is both directly and indirectly dominated by n.
4. If the leaf nodes are numbered in the order they are read, then we have a total ordering on the leaf nodes, and each leaf node d occurring between the start-tag and the end-tag for any nonterminal node n is dominated by n: an arc $n \to d$ is created when d is read, and the arc is removed only if n indirectly dominates d. If the arc is not removed, n dominates d directly in the resulting graph; if the arc is removed, n dominates d indirectly. Since this is true for every d between the start- and end-tags of n, we can conclude that n dominates a contiguous subsequence of the leaf nodes.
5. Since every start-tag s is immediately followed by a character data segment d (possibly containing the empty string), and d is dominated by the node for s but not by any node for elements with start-tags occurring later in the document, it follows that no two nodes dominate subsequences of the frontier which begin with the same character data segment. It follows in turn that no two nodes dominate the same subsequence of the frontier.

These characteristics of the resulting graph suffice to make it fall within the definition of a restricted GODDAG as given above.

It is easy to see that a fairly simple traversal of the GODDAG structure can be used to serialize the restricted GODDAG structure as a MECS document, as long as no two elements which overlap each other also have the same generic identifier.

Let us examine a simple example. In processing the simple document

```
<s/<a/ John <b/ likes /a> Mary /b>/s>
```

we will process, in turn, eleven segments:

1. The start-tag `<s/`.
2. A character-data segment containing the empty string.
3. The start-tag `<a/`.
4. A character-data segment containing the string “ `John` ”.
5. The start-tag `<b/`.
6. A character-data segment containing the string “ `likes` ”.
7. The end-tag `/a>`.
8. A character-data segment containing the string “ `Mary` ”.
9. The end-tag `/b>`.
10. A character-data segment containing the empty string.
11. The end-tag `/s>`.

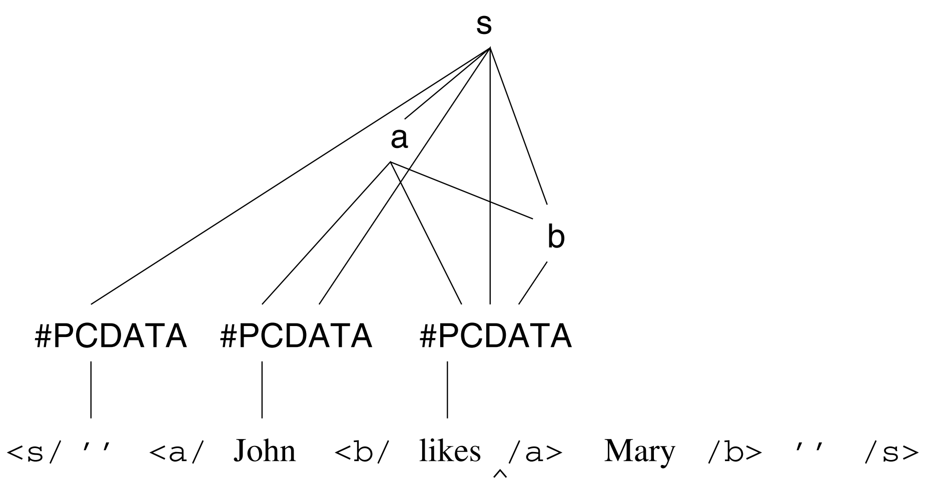


Fig. 9. GODDAG construction after the word “likes” is read

After the first six of these, the list of open elements includes a, b, and s, and the graph has the form shown in Fig. 9.

At this point we encounter our first end-tag, for $<a>$. Since the $<a>$ element was not the most recently opened, we know that the $<b>$ element, which was opened more recently, does not nest properly within $<a>$. So we remove the arc $a \rightarrow b$ (Fig. 10).

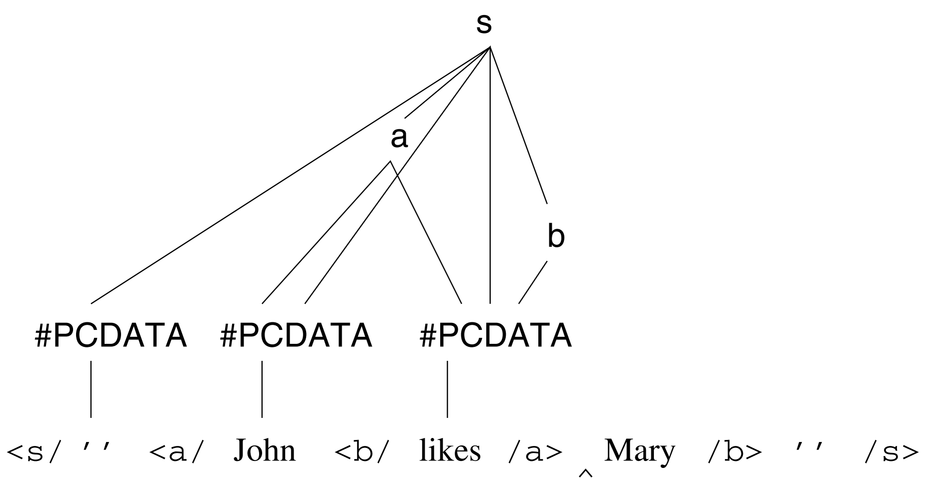


Fig. 10. GODDAG construction after “/a>” is read

There are no nodes doubly dominated by a and any descendant of a, so no further arcs are removed. After adding the character-data segment for " `Mary` ", processing the end-tag for the $<b>$ element does not cause any arcs to be removed. After processing the empty string between the $<b>$ and $<s>$ end-tags, and the final $<s>$ tag, the graph is as shown in Fig. 11.

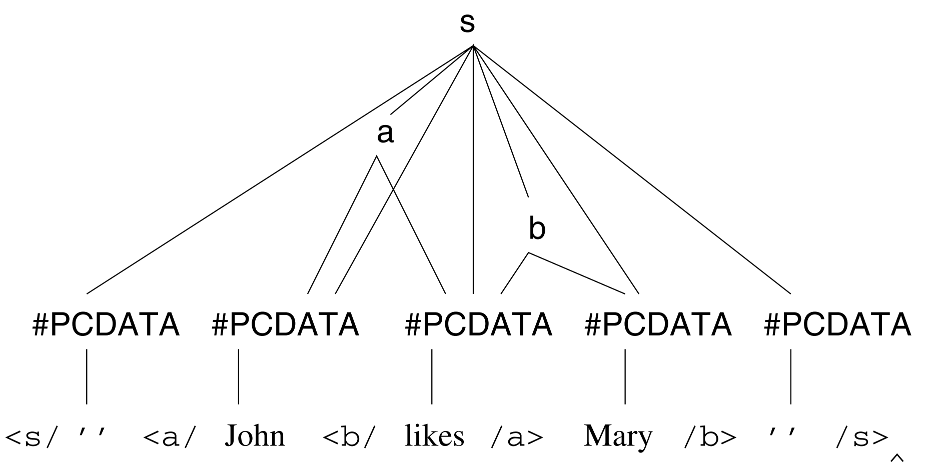


Fig. 11. GODDAG construction after "`/s>`" is read

At this point, we remove the direct arcs from s to the leaf nodes other than the first and last. The result is shown in Fig. 12.

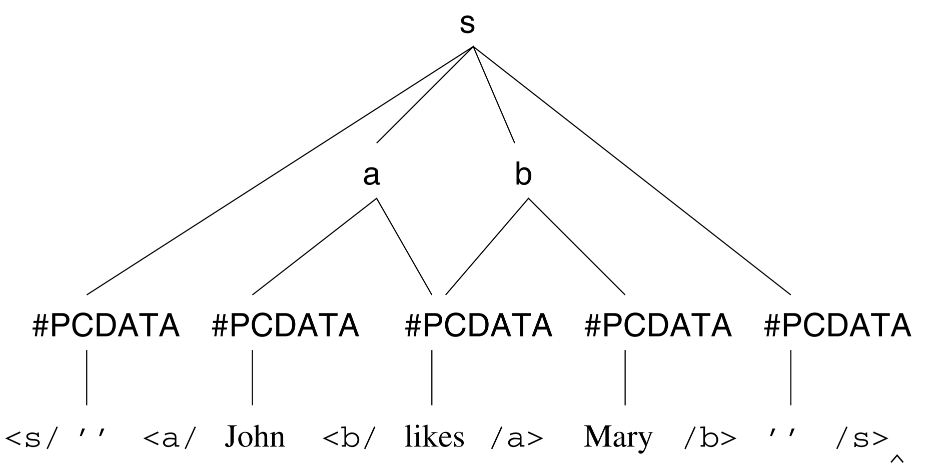


Fig. 12. GODDAG construction at completion

3.3 Generalized GODDAGs

In a generalized GODDAG, the constraints on the graph are relaxed:

1. For any nonterminal node n, the set of arcs from n to some other node are ordered. For any two arcs $n \rightarrow a$ and $n \rightarrow b$,
   (a) $n \rightarrow a < n \rightarrow b$ (in which case we say that node a precedes b among the children of n, or
   (b) $n \rightarrow a > n \rightarrow b$ (in which case we say that node b precedes a among the children of n, or
   (c) $n \rightarrow a = n \rightarrow b$ (in which case they are the same arc)
2. There is no single ordering on the leaf nodes, and no requirement that the leaf nodes dominated by any node be contiguous. (In the absence of a single ordering, there is no way to express the notion that a set of leaf nodes is or is not contiguous.)
3. There is no requirement that any two nodes dominate different sets of leaf nodes.

3.4 Applications

The principal motivation for defining GODDAGs is to provide a data structure for representing documents with overlapping structures. In the majority of cases, these can be represented using restricted GODDAGs. The similarities between tree structures and GODDAGs allow the same methods of interpreting the meaning of markup to be followed: feature values can be inherited from a parent, overridden by a descendant, and so on. There is some chance for conflict and confusion, since with multiple parents, it is possible that different parents have different and incompatible values for the same feature, etc. The formal specification of the meaning of a markup language will need to specify how to interpret such cases.

GODDAGs can be useful in the construction of various kinds of software for processing complex documents. We sketch two applications here: the elimination of 'spurious' overlap and the representation of virtual elements.

Systems which allow overlapping markup often suffer from what we term *spurious* overlap: the (usually unintentional) use of overlapping markup where the structure of the document can be captured adequately without overlap. When two elements overlap, they define three distinct regions dominated by one or the other of the overlapping elements, or both:

```
<a/ John <b/ likes /a> Mary /b>
```

If any one of the regions is empty, then the overlap is spurious: the document can be rewritten without overlap, without changing the interpretation of any character of the document. Here are three examples of spurious overlap.

```
<a/<b/ John likes /a> Mary /b>
<a/ John likes <b//a> Mary /b>
<a/ John likes <b/ Mary /a>/b>
```

A GODDAG structure lacks spurious overlap if certain conditions hold. We define the function *leafset* as returning the set of non-empty-string leaf nodes dominated by a given node. A GODDAG structure is clean if the following conditions hold:

1. If $\mathit{leafset}(b) \subset \mathit{leafset}(a)$, then $a \to b$.
2. If $\mathit{leafset}(b) = \mathit{leafset}(a)$, then either $a \to b$ or $b \to a$.

Clean GODDAGs can be created by constructing a restricted GODDAG, and then deleting every leaf node labeled with the empty string except those dependent on empty elements and ensuring that when the leafset of one node is a proper superset of the leafset of another, the first node dominates the second. Such an algorithm produces the following reformulations of the examples just given:

```
<b/<a/ John likes /a> Mary /b>
<a/ John likes /a><b/ Mary /b>
<a/ John likes <b/ Mary /b>/a>
```

The second application of GODDAGs we wish to mention is the representation of virtual elements as they are defined in the TEI *Guidelines*. Such virtual elements are represented in a generalized GODDAG as nodes like any other, and their children can be searched and processed in the same way as the children of normal elements.

3.5 Open Questions

A number of open questions remain for further work:

1. formal proof that the transformation of a MECS document into a GODDAG structure, followed by the serialization of the GODDAG structure into a MECS document, produces a new MECS document equivalent to the old
2. demonstration of the use of GODDAG structures in processing documents with overlap (e.g. in indexing and search and retrieval applications)
3. algorithms for translating SGML/XML documents which use any of the techniques described in the first part of this paper into GODDAG structures, and vice versa

References

1. Association for Computers and the Humanities (ACH), Association for Computational Linguistics (ACL), and Association for Literary and Linguistic Computing (ALLC). 1994. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*, ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: TEI, 1994.
2. Barnard, David, Ron Hayter, Maria Karababa, George Logan, and John McFadden. 1988. "SGML-based markup for literary texts: Two problems and some solutions." *Computers and the Humanities* 22: 265–276.

3. Barnard, David T., Lou Burnard, Jean-Pierre Gaspart, Lynne A. Price, C. M. Sperberg-McQueen, and Giovanni Battista Varile. 1995. “Hierarchical encoding of text: Technical problems and SGML solutions.” *Computers and the Humanities* 29: 211–231.
4. Bray, Tim, Jean Paoli, and C. M. Sperberg-McQueen, ed. *Extensible Markup Language (XML) 1.0*. [Cambridge, Mass., Sophia-Antipolis, Tokyo]: World Wide Web Consortium, 1998.
5. Goldfarb, Charles F. *The SGML Handbook*. Oxford: Clarendon Press, 1990.
6. International Organization for Standardization (ISO). 1986. *ISO 8879: Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*. [Geneva]: ISO, 1986.
7. McKelvie, D., C. Brew, and H. S. Thompson. 1998. “Using SGML as a basis for data-intensive natural language processing.” *Computers and the Humanities* 31: 367–388.
8. Murata, M. 1995. “File format for documents containing both logical structures and layout structures.” *Electronic publishing* 8: 295–317.
9. Sperberg-McQueen, C. M., and Claus Huitfeldt. 1999. “Concurrent document hierarchies in MECS and SGML.” *Literary & Linguistic Computing* 14.1: 29–42.
10. Sperberg-McQueen, C. M., Claus Huitfeldt, and Allen Renear. 2000. “Meaning and Interpretation of Markup.” *Markup Languages Theory & Practice* [forthcoming]. Paper originally presented at ALLC/ACH 2000, Glasgow, and at Extreme Markup Languages 2000, Montréal.

A Correspondence between UML Diagrams and SGML/XML DTDs

Eila Kuikka and Anne Eerola

Department of Computer Science and Applied Mathematics
University of Kuopio
P.O.Box 1627, 70211 Kuopio, Finland
{Eila.Kuikka,Anne.Eerola}@uku.fi
http://www.cs.uku.fi/~kuikka/Multi/hospital.html

Abstract. In this paper, we compare the semantics and structure of the conceptual information presented in the Unified Modeling Language (UML), which is used in analyzing object-oriented systems, and Document Type Definitions (DTD), which define the structures of SGML and XML documents. SGML (Standard Generalized Markup Language) and XML (Extensible Markup Language) are international standards for specifying the notations used for defining structured documents. We present correspondence rules for generating DTDs semiautomatically from UML diagrams. The rules have been developed as a part of the analysis and design method to create the structure definition for a document. As an example, we use a patient record.

1 Introduction

A patient record contains the identification information of a patient as well as information about the closest relatives, about medical facts, for example, allergies, blood group, etc., and about actions taken and diagnoses made during clinical visits. Traditionally, a patient record is a pile of paper forms filled by the staff of various clinics and laboratories of a hospital. Hence, its main part consists of free text written or dictated by doctors and nurses but it can also contain drawings, pictures, photos and films. A patient record is also an object for various types of queries. Moreover, it is typically a document which is updated by many users, required to be done in many different layouts, transferred from one place to another and archived for a long time. For these reasons, it is a very good candidate to be processed as a structured document and coded using SGML (Standard Generalized Markup Language) [1] or XML (Extensible Markup Language) [2] document standards.

The structure of a document defined by Document Type Definition (DTD) forms the basis for an SGML/XML-based system. There are several methods of designing DTDs. The bottom-up method was introduced by Maler and el Andaloussi [3] as a part of a global SGML project. A spiral model with several iterations was brought forward by Travis and Waldt [4]. Top-down methods have been used by Alschuler [5] and by Colby and Jackson [6]. Salminen, Kauppinen and Mehtovaara [7,8] used the object-oriented analysis method of Shlaer and Mellor [9] to describe relationships between documents in a situation where several documents have to be structured. Object-oriented approach has also been used in the ISIS European XML/EDI Pilot Project [10] that applies XML technology into electronic message interchange of business data. The definition of the generalized message description is modeled with the Unified Modeling Language (UML) diagrams [11], which are mapped onto XML DTD markup declarations.

We have developed an object-oriented method for the derivation of DTDs [12, 13,14] by starting from describing the utilization of the document in its natural environment. Our method, which combines existing analysis and design methods, emphasizes the use and reuse of the document in the structure design. The method has the following steps:

1. The flow of data in the organization is considered from the point of view of the document. Requirement analysis [15], which resembles traditional data flow analysis [16], is used in getting a general description of actions with the document.
2. The conceptual modeling is done with the UML [11]. Here it is important that the model describes the utilization of the document in a natural way.
3. Elm (enables lucid models) tree diagrams [3] of the SGML/XML DTD are created from the UML class diagrams using a set of correspondence rules introduced in this paper.

The method was developed as a part of a project the purpose of which was to create an SGML DTD for the electronic patient record of Kuopio University Hospital, especially for the outpatient ward of the Department of Pediatrics. This paper describes phase 3 and uses a patient record in the examples.

Sections 2 and 3 of the paper define briefly the general concepts of object-oriented analysis and SGML/XML, respectively. Section 4 introduces the correspondence rules in detail. Section 5 tells about the implementation of the mapping system. In section 6 we discuss the alternative solutions and future development issues. Section 7 concludes the results.

2 Object-Oriented Approach and UML Diagram

Several object-oriented analysis methodologies have been introduced in the nineties. Without doubt, the most important and widely used is the Unified Modeling Language (UML), which in fact provides a standard way to visualize, specify and document the artifacts of a software system [11]. UML is used within the process (called Unified Process), which defines who is doing what, when and how to build a software product or to enhance an existing one. The Unified Process is iterative and incremental.

UML contains several diagrams, which describe the conceptual and the functional properties of the system. In this work, we need the class diagram, which contains a large amount of the information needed in DTDs. Fig. 1 presents as an example a UML class diagram of a simplified medical record (with the most important concepts and notations).

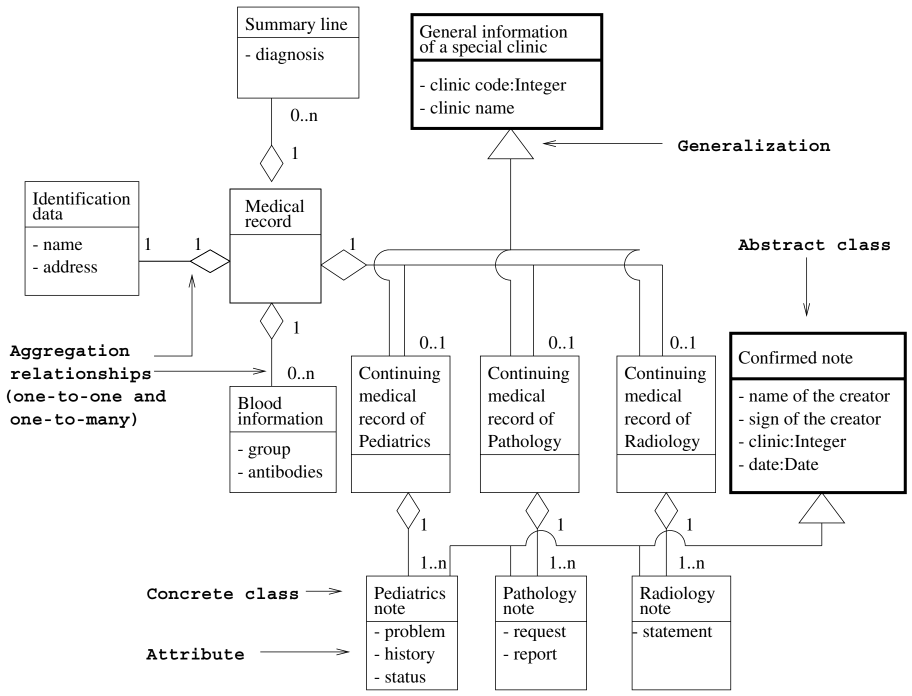


Fig. 1. A UML class diagram

The *class* (e.g. Medical record) is a description of a set of *objects* that share the same properties, i.e. *attributes, methods, relationships*, and *semantics. Concrete classes* (e.g. Pediatrics note) define objects whereas *abstract classes* (e.g. Confirmed note) are used in order to refine the properties of the classes step by step. Graphically, a class is rendered as a rectangle, that includes the name, attributes and methods. Methods are not represented in Fig. 1. After the name of the attribute, it is possible to define the type (e.g. Integer) as well as the optionality (e.g. `<0..1>`) and multiplicity (e.g. `<0..n>` or `<1..n>`). The number of presented details depends on viewpoint of the user of the diagrams.

The class hierarchy allows subclasses to inherit properties from super classes. The super class (e.g. General information of a special clinic) is a *generalization* of its subclasses (e.g. Continuing medical record of Pediatrics, Pathology, and Radiology). An attribute of an object can also be an object. This leads to the *aggregation relationship*. For example, Medical record contains exactly one Identification data, zero or more Summary lines, zero or more Blood information and one Continuing medical record, for each clinic the patient has visited. The multiplicity in the aggregation relationship is defined in the UML diagrams using numbers in both ends of the relationship notation. The possibilities are, for example, exactly one (`1`), zero or one (`0..1`), many (`0..n`), or one or more (`1..n`). The aggregation and generalization relationships are recursive.

It is also possible to describe associations between objects in the class diagram; for example, the association "patient has a Medical record". Associations lead to references between objects. Actually in Fig. 1 we should have a class Patient with the attributes name and address associated with the Medical record class. However, if the objects of a class taking part in the relationship are not needed separately, it is possible to insert the information of the referred object in the aggregate object. For this reason we have omitted the class Patient and its attributes are inserted as part called Identification data in the class Medical record.

3 SGML and XML

SGML (Standard Generalized Markup Language) is an international standard [1, 17,18] for defining the notation which is used to represent hierarchically structured documents. XML (Extensible Markup Language) [2] as its subset is a Recommendation of the World Wide Web Consortium and is meant to be used especially for structured Web documents. The structure of a document type is defined using a DTD (Document Type Definition) according to both of these notations. A document instance is marked with start and end tags for structure parts. The start tag is delimited with `<` and `>` characters and the end tag with `</` and `>` characters. The marking should be valid for a given DTD. A sample document instance could start

```
<MedRec><IdentData><Name>John Khos</Name>...</IdentData>
<ContMedRecPediatrics><ClinicName>Pediatrics ...
```

The DTD (see the example DTD on the next page) contains an *element declaration* for each structure part of the document. There are two types of elements: a *structure element* defines hierarchical structures and contains other elements and possible also text, whereas a *content element* consists only of text. The SGML element declaration has three parts: the name of the element, indicators for the existence of the tags in the document instance (`-` for a compulsory tag and `O` for an optional tag), and the definition of the content model. According to the XML notation, the second part is missing since all the tags must exist in the XML document instance. The *content model* (shown in parentheses) of an element specifies the subelements, their order and existence conditions. The order of subelements is controlled using connectors between the subelements: '`,`' for a sequence connector, '`|`' for a choice operator and '`&`' for a set (the free order of elements) connector. Subelements can be repeated or be absent. The subelement name or a group is followed by `*` for zero or more repetitions, by `+` for one or more repetitions and by `?` for an optional existence. The content is defined using `#PCDATA`, `RCDATA` or `CDATA` data type. To give metainformation usually not included in the actual content of a document, it is possible to define a list of *attributes* for each element.

The DTD also contains *entity declarations* for defining the identifiers of a character (for example, characters not in the keyboard), of a character string and of a content model. The last one is called a *parameter entity* and it exists in the DTD only for the purpose of shortening the DTD. In the parameter entity declaration, the name is preceded by a `%` character and the content model is delimited by quotation marks. The reference to a parameter entity in a content model of another element is made using the name, which is delimited by `%` and `;` characters.

The following DTD defines that the `MedRec` element consists of the identification data of the patient (`IdentData`) followed by a sequence of continuing medical records from different clinics (for the pediatric clinic `ContMedRecPediatrics`). The name and address of the patient are character sequences. The parameter entity `GenInfSpecClinic` defines the identifier for two sequential elements, the clinic code (`ClinicCode`) and the clinic name (`ClinicName`); these are used for every clinic. The continuing medical record from each clinic contains general information (`GenInfSpecClinic`) and at least one note for each visit at the clinic (`PediatricsNote`). The content of the note consists of general information (entities `ConfirmNote1` and `ConfirmNote2`) and some specific information (`Problem`, `History`, and `Status`).

```
<!ELEMENT MedRec                      - - (IdentData,
                                           ContMedRecPediatrics,...)>
<!ELEMENT IdentData                   - - (Name,Address)>
<!ELEMENT (Name,Address)              - O (#PCDATA)>
<!ENTITY % GenInfSpecClinic "ClinicCode,ClinicName">
<!ELEMENT (ClinicCode,ClinicName) - O (#PCDATA)>
<!ELEMENT ContMedRecPediatrics    - - (%GenInfSpecClinic;,
                                        PediatricsNote+)>
<!ENTITY % ConfirmNote1 "Clinic,Date">
<!ENTITY % ConfirmNote2 "CreatorName,CreatorSign">
<!ELEMENT (Clinic,Date,CreatorName,
          CreatorSign)             - O (#PCDATA)>
<!ELEMENT PediatricsNote           - - (%ConfirmNote1;,Problem,
                                        History,Status,
                                        %ConfirmNote2;)>
<!ELEMENT (Problem,History,Status) - O (#PCDATA)>
```

An elm (enables lucid models) tree diagram for a DTD, developed by Maler and el Andaloussi [3], defines graphical representations for the items of the DTD. As an example, Fig. 2 shows a part of the previous DTD as an elm tree diagram. Structure elements are illustrated as rectangles and content elements and parameter entities as ovals. Existence indicators are written next to these. The sequential order in the content model is denoted by a horizontal line. Although not shown in Fig. 2, each alternative subelement is connected to the element with a line and a set of subelements is enclosed in a round “bag”.

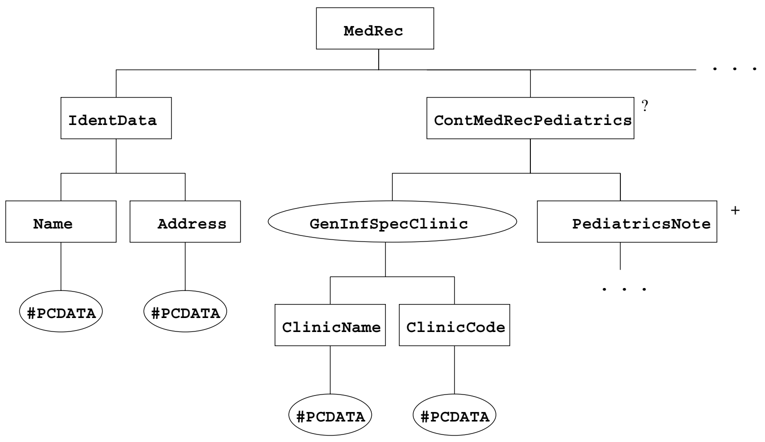


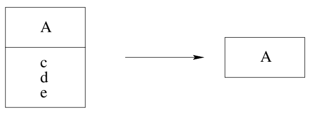
Fig. 2. The elm tree diagram of a DTD

4 Transformation of UML Diagrams to Elm Tree Diagrams

In this section, we introduce the three groups of correspondence rules suitable for semi-automatically generating DTD element and parameter entity declarations from UML diagrams. *Transformation rules* define the type of the elm tree node which an elementary part of the UML diagram (a class or an attribute) generates. *Placement rules* define how the nodes generated by the transformation rules are connected to each other as well as how aggregate and generalization relationships specify the location of the generated nodes. *Processing rules* define the order in which the class hierarchies are processed.

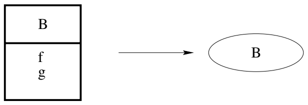
TRANSFORMATION RULES

Rule 1: A concrete class in a UML class diagram generates an SGML/XML structure element.



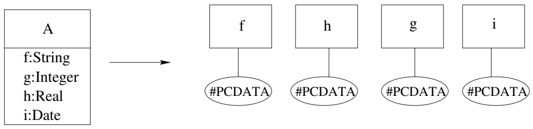
A concrete class models a real-world issue, which may have a structure and features. In SGML/XML documents, the corresponding component is a structure element with a possibility to define its subelements. In Fig. 1, concrete classes 'Medical record' and 'Identification data' are examples and should be transformed into elements.

Rule 2: An abstract class in a UML class diagram generates an SGML/XML parameter entity.



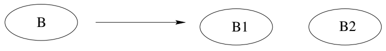
An abstract class does not model any real world issue but contains a set of general information (attributes) inherited by several subclasses. It should be possible to repeat the generated SGML/XML component as a subpart of several SGML/XML structure elements corresponding to the subclasses but it should not generate any markup to the document. For this reason, it is shown as a parameter entity. There are two examples of abstract classes in Fig. 1: 'General information of a special clinic' and 'Confirmed note'.

Rule 3: An attribute in a UML class diagram generates an SGML/XML content element.



The attributes of a concrete or an abstract class define the features of the class. The attributes may be of a certain data type (string, character, integer, etc.) and the values of attributes are defined by the user of the application; for a document, a content element is suitable for this purpose. The SGML/XML content may only be character data (`#PCDATA`, `RCDATA` or `CDATA`). It is not possible to transform the data types given for UML attributes so that they are usable for SGML/XML elements. In Fig. 1, the concrete class 'Identification data' has attributes 'Name' and 'Address' and the abstract class 'General information of a special clinic' has attributes 'Clinic code' and 'Clinic name'.

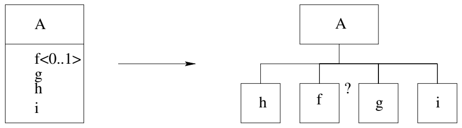
Rule 4: An SGML/XML parameter entity may be divided into many parameter entities.



If all the SGML/XML content elements generated from attributes of an abstract class using rule 3 are used in the same order in the content models of other elements, this rule is not needed. However, this rule is needed if these content elements are used in different orders or in several groups. In practise, an abstract UML class with attributes is first transformed into a parameter entity with all its attributes and then, if needed, it is divided into many parameter entities. The names of new parameter entities are indexed from the original parameter entity name. In the class diagram of Fig. 1, the attributes of 'Confirmed note' class are used in two groups and two parameter entities are created.

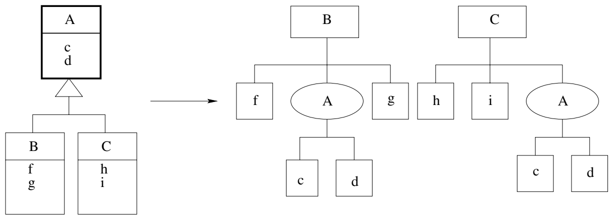
PLACEMENT RULES

Rule 5: The SGML/XML element corresponding to the attribute in a UML class diagram is placed as a subelement in the content model of the element or the *parameter entity corresponding to the class of the attribute. The place of the subelement among other subelements is fixed and the connector is selected. The repetition and optionality indicator for the SGML/XML subelement is as indicated for the attribute.*



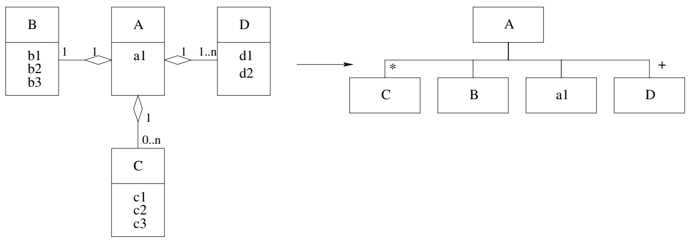
Attributes define the features of a class and the values of the attributes are given when an object is created. Therefore, the SGML/XML elements corresponding to UML attributes should be a part of the content model of the element generated from a UML class. Although UML diagrams do not consider the order of the attributes of a class, the subelements in the content model should be in the predefined order and separated by certain connectors. During the mapping, this information should be given by the user. In our example, we have always used the sequence connector although other connectors could be possible, too. The multiplicity and optionality indicators in the UML diagram correspond to those for SGML/XML elements in the following way: for `<0..1>` it is `?`, for `<0..n>` it is `*` and for `<1..n>` it is `+`.

Rule 6: The SGML/XML parameter entity corresponding to a super class of a generalization relationship in a UML class hierarchy is placed in the content models of the SGML/XML elements or in SGML/XML parameter entities corresponding to each of the subclasses. The place of the parameter entity among other elements or parameter entities is fixed and the connector is selected.



Since the attributes of a super class are inherited by its subclasses, the generated parameter entity or several entities (if rule 4 is used) are placed in the content models of the elements, each corresponding to a subclass. Since, using rule 5, the content elements corresponding to the attributes of a subclass are already included in the content model of this subclass, we have to place the generated parameter entities (one or more) among these content elements in the correct order and to select the suitable connector.

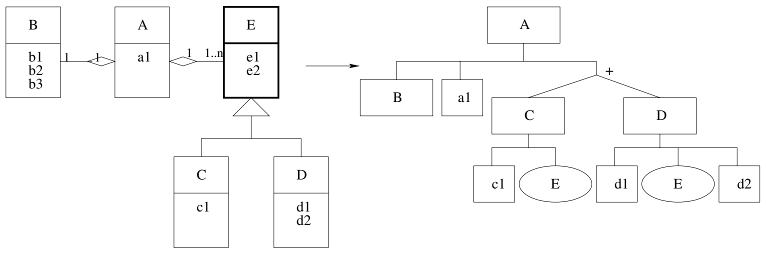
Rule 7: The part in an aggregate relationship in a UML class diagram is placed as an SGML/XML element or an SGML/XML parameter entity in the content model of the element or in the parameter entity corresponding to the aggregate. The place for the element or parameter entity among other elements or parameter entities of the content model is fixed and the connector is selected.



The aggregation relationship defines the containment between classes: an aggregate and its part classes. Thus, in the SGML/XML DTD, it defines that the subelements generated from part classes belong to the content model of the element corresponding to the aggregate. Again, UML class diagrams do not fix the order of several part classes or the connectors between them. The order is needed in the content model of an SGML/XML element and it has to be given by the user. The multiplicity indicators in the UML diagrams correspond to the existence indicators of the SGML/XML elements in the following way: for `1` there is no character , for `0..1` it is `?`, for `0..n` it is `*` and for `1..n` it is `+`.

PROCESSING RULES

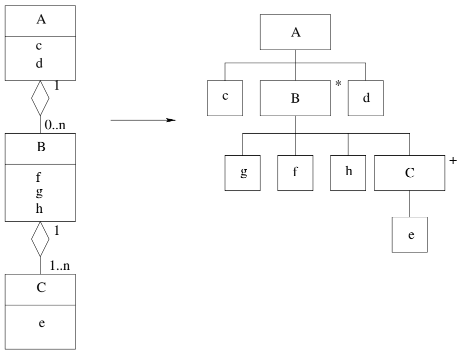
Rule 8: The generalization rule 6 is applied before aggregation rule 7.



If an abstract class in a UML class diagram is aggregated to a concrete class, actually, a set of specialized concrete classes from the abstract class is meant to be aggregated. For this reason, we first have to find out the SGML/XML elements corresponding to those concrete classes and, thus, to apply rule 6 before rule 7.

Rule 9: The aggregation hierarchies are processed top-down and the classification hierarchies are processed bottom-up.

When several aggregation relationships follow each other, our mapping processes them in the fixed order. It is natural for aggregation hierarchies to start from the top and proceed downwards.



When several generalization relationships follow each other the natural way is to start from the concrete classes which inherit the super classes and proceed from bottom to top.

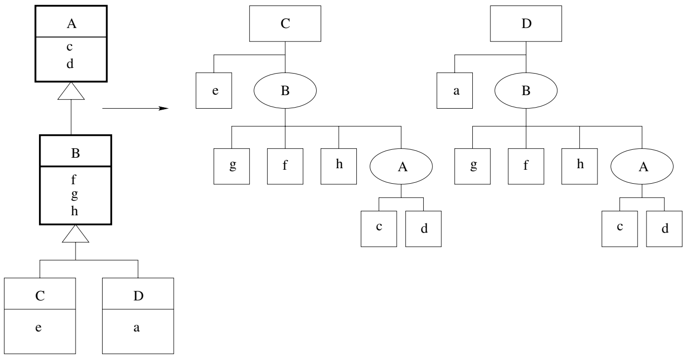


Figure 3 shows the elm tree diagrams generated from the UML diagram in Fig.1.

5 About the Implementation of the Mapping System

The UML case tools, for example Rational Rose, allow the user to export the models. This is done with the XMI interchange format [19]. This specification defines how to create XML DTDs for the stream-based data format when models are transferred from one system to another. XMI is standardized by the Object Management Group (OMG).

Our mapping system (see Fig. 4) uses the XMI-based DTD for the text export and applies our rules to the exported XML document to produce a textual representation for the DTD. The prototype has been implemented as a Java program which uses DOM (Document Object Model) API [20] to parse the XML document and create a DOM tree for navigating in the document. The UML classes and their attributes and relationships are fetched from the DOM tree and mapped onto the corresponding SGML/XML element and parameter entity declarations using the rules in Sect. 4. For each element or parameter entity a list of its subelements is generated, and thereafter, their order is asked

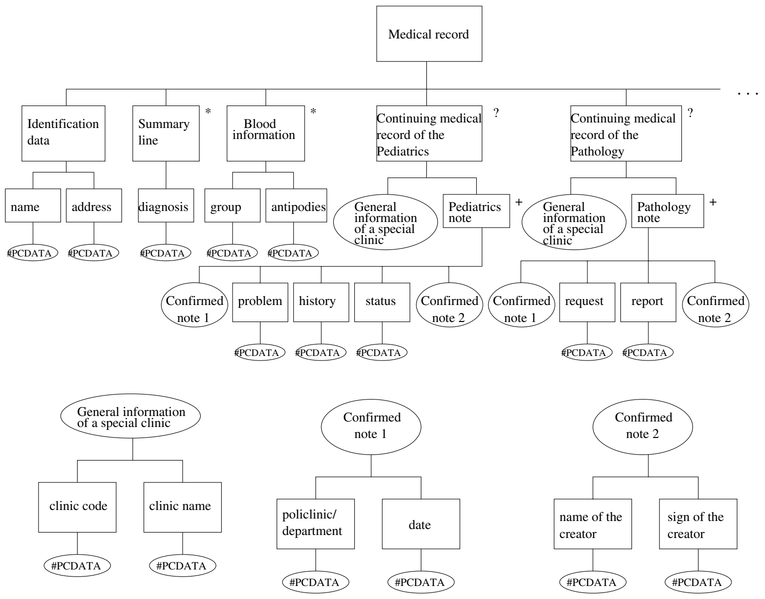


Fig. 3. The elm tree diagram of the medical record modeled in Fig. 1

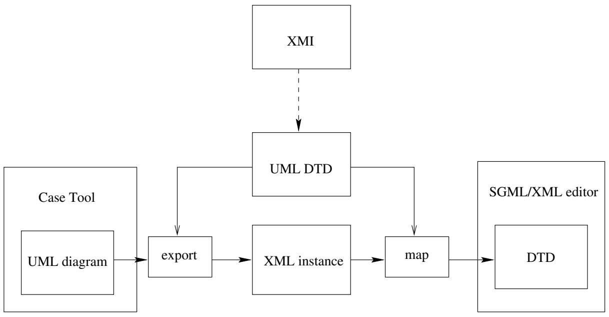
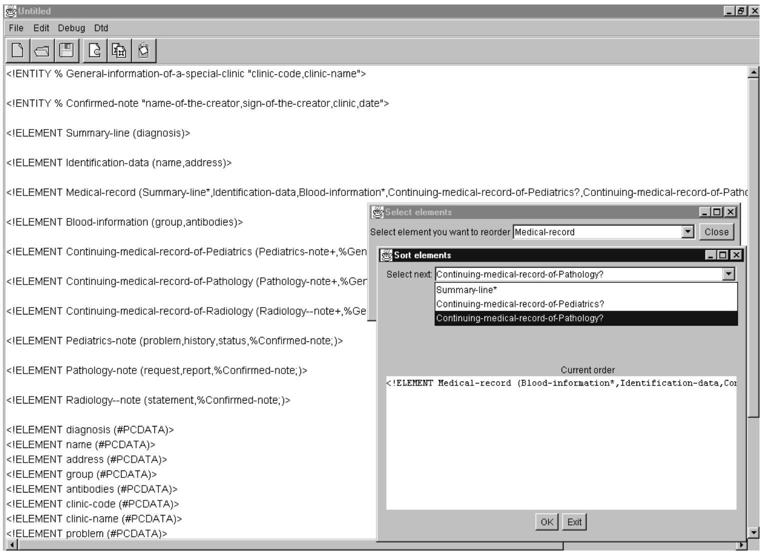


Fig. 4. Overview of the mapping system

from the user. Fig. 5 shows a situation where the user is reordering the content model for the 'Medical record' element. Until now we have tested the system using simple diagrams. The use of DOM API may restrict the size of diagrams since the exported XML document contains a large amount of data (not used by our system) for formatting diagrams and DOM parser reads into the memory the whole XML document to generate the tree. This also makes the processing a little slow. Our approach, using the XMI



```
<!ENTITY % General-information-of-a-special-clinic "clinic-code,clinic-name">

<!ENTITY % Confirmed-note "name-of-the-creator,sign-of-the-creator,clinic,date">

<!ELEMENT Summary-line (diagnosis)>

<!ELEMENT Identification-data (name,address)>

<!ELEMENT Medical-record (Summary-line*,Identification-data,Blood-information*,Continuing-medical-record-of-Pediatrics?,Continuing-medical-record-of-Patho

<!ELEMENT Blood-information (group,antibodies)>

<!ELEMENT Continuing-medical-record-of-Pediatrics (Pediatrics-note+,%Gen

<!ELEMENT Continuing-medical-record-of-Pathology (Pathology-note+,%Ger

<!ELEMENT Continuing-medical-record-of-Radiology (Radiology--note+,%Ge

<!ELEMENT Pediatrics-note (problem,history,status,%Confirmed-note;)>

<!ELEMENT Pathology-note (request,report,%Confirmed-note;)>

<!ELEMENT Radiology--note (statement,%Confirmed-note;)>

<!ELEMENT diagnosis (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT group (#PCDATA)>
<!ELEMENT antibodies (#PCDATA)>
<!ELEMENT clinic-code (#PCDATA)>
<!ELEMENT clinic-name (#PCDATA)>
<!ELEMENT problem (#PCDATA)>
```

Select element you want to reorder: Medical-record

Select next: Continuing-medical-record-of-Pathology?

- Summary-line*
- Continuing-medical-record-of-Pediatrics?
- Continuing-medical-record-of-Pathology?

Current order

```
<!ELEMENT Medical-record (Blood-information*,Identification-data,Co
```

Fig. 5. The user interface for reordering the items in a content model

standard for the textual form, would make it possible to use our prototype also with other case tools which utilize XML as an export format. However, one problem is that XMI does not define a fixed DTD but only the rules for defining DTDs. Various systems can use their own DTDs or modify the existing ones. For example, the XMI outputs of Rational Rose and IBM XMI Toolkit differ from each other: the output of the latter does not contain all the information that our rules need.

6 Discussion

The rules introduced in this paper define correspondences only to a restricted set of components of UML and SGML/XML representations.

Using rules we can generate a DTD which preserves the names of classes and attributes of the UML diagrams. However, although it is natural to use explanatory, and usually long names in UML diagrams, the same names may not be suitable for the final DTD. When selecting names for the SGML/XML elements, it is possible to use the naming rules introduced in [21].

Instead of asking the order of the generated elements and parameter entities in a content model from the user of the mapping system a more practical way would be to add the required notations to the UML diagrams already in the analysis phase.

The UML tools make it possible to define data types for the attributes of a class. When we transform attributes into content elements, we do not use UML data types; all the content is defined in SGML/XML character strings. One solution for XML documents would be to use the XML Schema specification [22, 23], which defines an extension to the XML DTD by adding data types for element contents. We should formulate the rules for generating XML schemata instead of XML DTDs.

7 Conclusion

We have introduced a set of correspondence rules for generating SGML/XML DTD element and parameter entity declarations from object-oriented UML class diagrams. The rules are a part of the analysis and design method for the derivation of an SGML/XML DTD for documents. The method was developed especially for the purpose of generating a DTD for a large structured document which is used by many people in a big organization. Our method opens up a new, pragmatic way to design structured documents and its use is not dependent on the application domain. Its result can be used to searching the possibilities to standardize the content of a document. Moreover, because DTD declarations are produced semi-automatically straight from UML diagrams, the designer can concentrate on the modeling of the structure. Our method would be a valuable module to be added to UML tools.

The method was tested on a patient record of a hospital. It produced a modular DTD by grouping together the SGML/XML markup declarations which correspond to a certain UML class. In this paper, we considered certain UML concepts (classes, attributes, and aggregate and generalization relationships) and defined their SGML/XML correspondences.

There is, however, further notations both in UML diagrams and SGML/XML DTDs which are useful or necessary to consider for the DTD design method. There are, for example, methods of classes, packages of classes and associations between classes in UML diagrams and attributes of elements, character string entities and links in SGML/XML DTDs. We will look into these issues in our further research as well as we will improve the usability of the mapping system.

Acknowledgments. The research was carried in the research project on Electronic Documents in Hospital financed by the University of Kuopio. The authors would like to thank Jorma Komulainen, Jari Porrasmaa, Ari Miettinen, Merja Ek and Tomi Tikkanen for their contribution in this work and the personality of the Department of Pediatrics of the Kuopio University Hospital for their kind help. The comments of the anonymous referees have been helpful in improving the paper.

References

1. ISO 8879, ISO, Geneva. *Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, 1986.
2. W3C Recommendation. *Extensible Markup Language (XML) 1.0*, 1998. Available at http://www.w3.org/TR/REC-xml.
3. E. Maler and J. el Andaloussi. *Developing SGML DTDs from Text to Model to Markup.* Prentice Hall, Upper Saddle River, New Jersey, 1996.
4. B. Travis and D. Waldt. *The SGML Implementation Guide.* Springer-Verlag, Berlin, 1996.
5. L. Alschuler. *ABCD...SGML: A User's Guide to Structured Information.* International Thomson Computer Press, London, Boston, 1995.
6. M. Colby and D.S. Jackson. *Special Edition: Using SGML.* QUE Corporation, Macmillan Publishing, Indianapolis, Indiana, 1996.
7. A. Salminen, K. Kauppinen, and M. Lehtovaara. Towards a methodology for document analysis. *Journal of the American Society for Information Science*, 48(7):644–655, 1997.
8. A. Salminen, M. Lehtovaara, and K. Kauppinen. Standardization of digital legistive documents. In M.S. Lynn, editor, *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, pages 72–81, Hawaii, USA, 1996. IEEE Computer Society Press.
9. S. Shlaer and S.J. Mellor. *Object Lifestyles – Modeling the World in States.* Yourdon Press, Englewood Cliffs, NJ, 1992.
10. ISIS European XML/EDI Pilot Project. ISIS XML/EDI Project Website. Available at http://www.tieke.fi/ovt/, 1999.
11. G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language Users Guide.* Addison Wesley, Reading, Massachusetts, 1999.
12. E. Kuikka, A. Eerola, J. Porrasmaa, A. Miettinen, and J. Komulainen. Design of the SGML-based electronic patient record system with the use of object-oriented analysis methods. In P. Kokol et al., editor, *Medical Informatics Europe '99*, pages 838–841, Ljubljana, Slovenia, August 1999. IOS Press.
13. E. Kuikka, A. Eerola, J. Porrasmaa, A. Miettinen, and J. Komulainen. Design of the SGML-based electronic patient record system with the use of object-oriented analysis methods. Technical Report A/1999/1, University of Kuopio, Department of Computer Science and Applied Mathematics, 1999.
14. E. Kuikka, A. Eerola, and J. Komulainen. About creation an XML DTD for an electronic patient record. In *XML Scandinavia 2000 Conference*, pages 11–31, Göteborg, Sweden, May, 2000.
15. S. Robertson and J. Robertson. *Mastering the requirements process.* Addison-Wesley, Harlow, England, 1999.
16. E. Yourdon. *Modern Structured Analysis.* Yourdon Press, Englewood Cliffs, NJ, 1989.
17. C.F. Goldfarb. *The SGML Handbook.* Oxford University Press, Oxford, UK, 1990.
18. E. van Herwijnen. *Practical SGML.* Kluwer Academic Publisher, Dordrecht, 1994.
19. Object Management Group. *XML Metadata Interchange (XMI). Proposal to the OMG Object Analysis & Design Task Force RFP 3: Stream-based Model Interchange Format (SMIF)*, 1998. Available at http://www.omg.org.
20. W3C Specification. *Document Object Model DOM 1.0 Level 1*, 1998. Available at http://www.w3.org/TR/REC-DOM-Level-1.

21. ISIS European XML/EDI Pilot Project. *Mapping from UML Generalised Message Descriptions to XML DTDs*, 1999. Available http://www.tieke.fi/isis-xmledi/D2/D22.htm.
22. W3C Working Draft. *XML Schema Part 1: Structures*, 2000. Available at http://www.w3.org/TR/xmlschema-1.
23. W3C Working Draft. *XML Schema Part 2: Datatypes*, 2000. Available at http://www.w3.org/TR/xmlschema-2.

A Conceptual Model for XML[*]

Anne Brüggemann-Klein[1] and Derick Wood[2]

[1] Institut für Informatik, Technische Universität München
Arcisstr. 21, 80290 München, Germany
`brueggem@informatik.tu-muenchen.de`
[2] Department of Computer Science
Hong Kong University of Science & Technology
Clear Water Bay, Kowloon, Hong Kong SAR
`dwood@ust.hk`

Abstract. We claim that document engineering (the systematic development of document presentations, representations and document tools) should be firmly based on explicit, formal and abstract document models. This precept is well accepted and practiced in, for example, the database community but it is, apparently, less accepted in the XML community. We want to change this position.
As a first step in the demonstration of the value of explicit, formal document models we present and discuss a conceptual model for XML DTDs and their (DTD-conforming) documents that is, indeed, abstract, formal and explicit. This new model captures what can be considered to be the essence of an XML DTD and its documents.

1 Introduction

Traditionally, the role of explicit and abstract models in the document processing and electronic publishing community has been fundamental to the development of document systems such as editors and formatters. Thus, Meyrowitz and van Dam [12], and Furuta, Quint and André [9] among others support and affim the view that: *The systematic development of document presentations, representations and tools (document engineering) should be based on document models not on document representations.* In contrast, explicit and abstract models are not discussed in the XML community and are not, apparently, considered to be crucial to the systematic development of applications. There are a few exceptions, however, such as Murata's arguments for using regular tree languages, as opposed to context-free grammars, as a model for XML grammars and documents.

The lack of apparent interest in modeling in the XML community is even more surprising when we observe that earlier languages and systems for document processing were much simpler but it was assumed that explicit, abstract

[*] The work of both authors was supported partially by a joint DAAD-HK grant. In addition, the work of the second author was supported under a grant from the Research Grants Council of Hong Kong.

models were a necessity. For example, context-free grammars and their parse trees were considered an appropriate model for structured editors for programming languages and documents. Indeed, Furuta's TNT [9] was one of the few richer models although it does not include the grammar.

It is clear that XML and SGML represent much richer documents and grammars (attributes, symbolic cross references, concurrent structures and division into physical units) but do not define an explicit, abstract and formal model. Indeed, SGML and XML are syntactic in nature; hence, are not real models and are not really models at all.

The neglect of models in the document processing (and specifically in the XML) community is in stark contrast to the role models play in other disciplines of computer science (and of engineering in general). Imagine the pros and cons of a relational database versus an object-oriented database being argued on the basis of syntactic representations rather than on high-level, abstract models (that is, on a conceptual level) or query optimization discussed on the physical representation of tables in computer memory.

We believe that modeling is a central part of document engineering (the systematic development of document presentations, representations and document tools) as it is in software engineering. We expect the following general benefits from modeling in the document context:

1. We can reason about the model and pose such questions as: What is needed in a document model for a particular type of application? How do various models compare? What are the benefits and drawbacks of one model over another?
2. It supplements document-language specification for the Web and also serves as the kernel of a concept definition from which a language representation can be derived as a secondary and perhaps even as an automatic process. This liberates us from discussions about syntax—conceptual issues can be discussed directly. It enables us to argue about concepts and ideas rather than about syntax.
3. It may and could be applied in current standardization efforts, such as XML Schema, to argue the pros and cons of the current, DTD-based model for XML documents and to communicate and reason about alternatives (such as tree automata and schemas).

As a first step to stimulate modeling, we provide a data model for XML that is simple, is rich enough to capture the essence of XML, encompasses grammars and document instances, is formal and abstracts from the syntactic representation of XML.

Our goal in creating the model is to capture the essential ideas that we believe underlie the logical part of XML in a concise, unambiguous and understandable way without being hindered by XML syntax and aspects of physical structure. Our reasons are two-fold. First, we wish to come to terms with what an XML document "is", so that we can more easily communicate the underlying concepts when teaching XML or when exploring its usefulness in document-engineering projects. Second, since XML is the obvious choice of a document representation for Web projects, which are becoming much more of an engineering challenge, we need to re-engineer XML in the following manner to complement the modeling we present here:

1. Rewriting the syntactic rules of the XML language (without changing the XML language itself) such that parsing XML can be accomplished by proven compiler-construction techniques.
2. Designing a data representation for the conceptual model and specifying access functions that can be defined in terms of the model. Implementing this ADT as a class in the object-oriented sense. Note that the class will represent both a DTD and a document instance.
3. Designing and implementing an XML parser that compiles an XML document into an instance of that class.

In a follow-up paper we intend to demonstrate how standard compiler-construction techniques can be used to obtain a parser for documents corresponding to a given XML DTD. We will describe how we can use the new approach to construct a representation of the conceptual model from a given DTD and its documents. Our new approach will free XML-ers to build their own parsers for classes of documents when needed. In addition, XML-ers can use components of our parser to parse a document instance and validate the document instance against a fixed and precompiled DTD.

The remainder of the paper is organized as follows. In Section 2, we give a brief overview of the Extensible Markup Language (XML). In Section 3, we define the **grammar-constrained structured document (GCSD) model** that captures the essence of XML DTDs and their documents. Basically, we re-engineer XML since we build the document model from document representations rather than the other way around. In Section 4, we discuss related work; and we conclude with some suggested future work in Section 5.

2 The Extensible Markup Language (XML)

The Extensible Markup Language (XML) describes a class of data objects called **XML documents**. The XML specification [3] defines the syntax of XML documents and partially describes the behavior of computer programs that process them.

XML documents are made up of what are called **entities** (essentially macros) that denote either parsed or unparsed data. Parsed data consists of characters, some of which form **character data**, and some of which form **markup**. Markup encodes a description of the document's organization into entities (**physical structure,** storage layout) and its so-called **logical structure**. XML has been widely accepted as a language to represent strcutured documents and their grammars.

The XML specification formally defines the syntax of XML documents using an extended context-free grammar and additional natural-language constraints.

The XML specification assumes that a software module called an **XML processor** is used to read XML documents and to provide access to their data. It further assumes that an XML processor is doing its work on behalf of another module, called the **application**. The XML specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The problems of the current XML approach are:

1. The document model of structured documents and their grammars that underlies the XML specification is only implicit, it is not explicit. The implicitness leaves room for different interpretations as is witnessed by the various explicit document models in current use (XSLT [6], XPATH [7] and XML Information Set [8]).
2. The XML specification focuses on syntax. An XML document is a specific syntactic representation of a structured document. Meta-syntactic sugar, such as the sequence of declarations and overwrite rules, and accidental surface properties such as character references and white space, muddy the clear picture of the "real" or "pure" structured documents underlying the syntactic surface. It remains unspecified which of this meta-syntactic sugar and these accidentals are passed on to the application by the XML processor and which are not paased on.
3. An XML document simultaneously represents the logical and the physical structure. Database design, on the other hand, separates the conceptual (or logical) and physical layers.

3 GCSD: The Conceptual Model for XML

We introduce our conceptual model of XML documents. Since XML is generally considered as a language to represent structured documents and their grammars, we call our model GCSD for Grammar-Constrained Structured Documents. Further modeling efforts may address either non-structured documents (such as GODDAG structures [15] or hypertext documents) or other constraint mechanisms (such as tree automata [13,14] or XML Schema [16,1]). In the GCSD model, we describe GCSDs as mathematical objects. We model both document instances and document grammars. We construct the model from only elementary mathematical objects and operations such as strings, sets, set operations and functions.

As have others before us (XSLT [6] and XPATH [7]), we have chosen to exclude the XML concept of entities from the GCSD model. Although we omit both the use and definition of entities in the GCSD model, we agree that they have an important part to play in practice. But, from the modeling viewpoint, we assume that all entities are already resolved.

Our aim is that the GCSD model should capture the essence of structured documents. A GCSD-modeled document is an abstract concept; we might devise an XML representation for the model as one way to make it tangible.

We present the GCSD model in the following subsections.

3.1 The GCSD Model Defined: Grammars and Instances

The GCSD model is based on a finite set *Character*[1] of characters, which we use for the content of a document, and four pairwise disjoint infinite[2] sets *DocSort*, *AttSort*, *DocID* and *AttValue* of names, which we use as a source of names for documents, attributes, document IDs and attribute values. From these five sets we build other sets using the standard set operations given in Table 1.

Table 1. Set operations and their notation.

Operation	Notation
Cross product	$\square \times \square$
Free monoid; set of sequences	$\square^*$
Set of functions	$\mathcal{F}(\square \longrightarrow \square)$
Set of partially-defined functions with finite domain	$\mathcal{F}_{\text{fin}}(\square \longrightarrow \square)$
Disjoint union	$\square \mathbin{\dot\cup} \square$
Set of subsets; power set	$\mathcal{P}(\square)$
Set of finite subsets	$\mathcal{P}_{\text{fin}}(\square)$
Finite set	$\{\square, \ldots, \square\}$
Set of regular languages over a given alphabet	$\text{Reg}(\square)$
Set of restricted-regular languages over given character and symbol sets	$\text{RReg}(\square, \square)$

We summarize the objects in the GCSD model and their definitions in Table 2.

Table 2. The objects in the GCSD model.

(1)	$\mathit{GCSDocument} \subseteq \mathit{DocInstance} \times \mathit{DocGrammar}$.
(2)	$\mathit{DocInstance} = \mathit{DocSort} \times \mathit{Attributes} \times \mathit{Content}$.
(3)	$\mathit{Attributes} = \mathcal{F}_{\text{fin}}(\mathit{AttSort} \longrightarrow \mathit{AttValue} \mathbin{\dot\cup} \mathit{Character}^* \mathbin{\dot\cup} \mathit{DocID})$.
(4)	$\mathit{Content} = (\mathit{DocInstance} \mathbin{\dot\cup} \mathit{Character})^*$.
(5)	$\mathit{DocGrammar} = \mathit{ContentRules} \times \mathit{AttRules}$.
(6)	$\mathit{ContentRules} = \mathcal{F}_{\text{fin}}(\mathit{DocSort} \longrightarrow \mathit{DocDomain})$.
(7)	$\mathit{DocDomain} = \text{Reg}(\mathit{DocSort}) \mathbin{\dot\cup} \text{RReg}(\mathit{Character}, \mathit{DocSort})$.
(8)	$\mathit{AttRules} = \mathcal{F}_{\text{fin}}(\mathit{DocSort} \times \mathit{AttSort} \longrightarrow \mathit{AttDomain})$.
(9)	$\mathit{AttDomain} = \mathcal{P}_{\text{fin}}(\mathit{AttValue}) \mathbin{\dot\cup} \{\texttt{String}, \texttt{ID}, \texttt{IDREF}\}$.

Recall that XML defines both grammars and their document instances. Each instance has a type and attributes and is recursively built from further instances

[1] Typically, this set would be the Unicode character set or some subset such as US-ASCII or ISO Latin-1.

[2] Each GCSD in the model uses finite subsets of *DocSort*, *AttSort*, *DocID* and *AttValue*.

and characters. Each grammar defines or specifies, for conforming instances and subinstances the range of their contents, their attributes and their possible values. This simple, intuitive summary of XML is reflected in the GCSD model, which we further explain in the following subsection.

3.2 The GCSD Model Explained: Objects

Mixed Content. XML has the concept of mixed content that we capture as follows. Given a finite set C of characters and a possibly infinite set S of symbols that is disjoint from C, a restricted-regular set over C and S is a set of strings $(C \cup F)^*$, where F is a finite subset of S. Hence, a restricted-regular set over C and S is a set of strings whose symbols are taken from C and from a subset F of S. We denote the restricted-regular sets over C and S by $\mathrm{RReg}(C, S)$.

Grammars and Structured Documents. We define a set *DocInstance* of structured documents and a set *DocGrammar* of context-free grammars that constrain structured documents. The set *GCSDocument* consists of all pairs (di, dg) in $\mathit{DocInstance} \times \mathit{DocGrammar}$ such that the structured document di conforms to the grammar dg. Clearly,

$$\mathit{GCSDocument} \subseteq \mathit{DocInstance} \times \mathit{DocGrammar}. \tag{1}$$

We define *GCSDocument* precisely in Section 3.4.

A structured document in *DocInstance* has a document-type name from *DocSort*, a number of attributes and some content; hence

$$\mathit{DocInstance} = \mathit{DocSort} \times \mathit{Attributes} \times \mathit{Content}. \tag{2}$$

Attributes and Content. The attribute specification for a structured document assigns values to the attribute names that are associated with the document. Hence,

$$\mathit{Attributes} = \mathcal{F}_{\mathrm{fin}}(\mathit{AttSort} \longrightarrow \mathit{AttValue}\,\dot{\cup}\,\mathit{Character}^*\,\dot{\cup}\,\mathit{DocID}). \tag{3}$$

Now, the content of a structured document is a sequence of characters and, recursively, lower-level structured documents:

$$\mathit{Content} = (\mathit{DocInstance}\,\dot{\cup}\,\mathit{Character})^*. \tag{4}$$

The recursive definition of document instances terminates, at its lowest levels, with content that is empty or consists of only characters.

Document Grammars. A document grammar has two kinds of rules: Ones that constrain the content and ones that constrain the attributes of structured documents:

$$\mathit{DocGrammar} = \mathit{ContentRules} \times \mathit{AttRules}. \tag{5}$$

This equation implies that the grammar has a rule for finitely many document names in *DocSort*.

Content Rules. The rules in *ContentRules* which constrain the content of a structured document, assign document domains to document names. Document domains, in turn, are either regular or restricted-regular sets over the alphabet *DocSort*. Hence,

$$\mathit{ContentRules} = \mathcal{F}_{\text{fin}}(\mathit{DocSort} \longrightarrow \mathit{DocDomain}), \tag{6}$$

where

$$\mathit{DocDomain} = \text{Reg}(\mathit{DocSort}) \,\dot{\cup}\, \text{RReg}(\mathit{Character}, \mathit{DocSort}). \tag{7}$$

Attribute Rules. Analogously, rules in *AttRules*, which constrain the attributes of a structured document, assign attribute domains to some combinations of document names and attribute names. An attribute domain is either a finite subset of *AttValue* or one of the symbols `STRING`, `ID` or `IDREF`. Hence,

$$\mathit{AttRules} = \mathcal{F}_{\text{fin}}(\mathit{DocSort} \times \mathit{AttSort} \longrightarrow \mathit{AttDomain}), \tag{8}$$

where

$$\mathit{AttDomain} = \mathcal{P}_{\text{fin}}(\mathit{AttValue}) \,\dot{\cup}\, \{\texttt{String}, \texttt{ID}, \texttt{IDREF}\}. \tag{9}$$

Values in the attribute domain `String` are any character strings in $\mathit{Character}^*$; values in the attribute domain `ID` and `IDREF` are any document IDs in *DocID*.

3.3 The GCSD Model Explained: Access Operators

We next define a number of operators that access parts of GCSDs, structured documents and grammars. The main purposes of an ADT are encapsulation and information hiding, and separation of concerns. We adopt a typical approach in defining the GCSD ADT; namely, we will use a mixture of mathematics and English. We have chosen to define our model directly as a mathematical object. Any set operation on the objects in the model that preserves their type is semantically meaningful in the model. Here we are just **naming** some operators that provide access to the objects in the model. We adopt the following two notational conventions:

1. For an operator X that operates on a set Y, the application of X to an element y in Y is denoted by $y.X$.
2. If a set X is defined as the cross product of sets $Y_1 \times \cdots \times Y_n$, then each Y_i, $1 \le i \le n$, denotes an operator on X; namely, the projection to the ith component. Hence, $(y_1, \ldots, y_n).Y_i = y_i$. We use this convention only if the $Y_1,\ldots,Y_n$ are all different sets so that the operators Y_i are defined unambiguously.

Following these conventions, the two operators *DocInstance* and *DocGrammar* operate on *GCSDocument*; for any grammar-constrained structured document $\mathit{gcsd} = (\mathit{di}, \mathit{dg})$, the equations

$$\mathit{gcsd}.\mathit{DocInstance} = \mathit{di}$$

and

$$gcsd.DocGrammar = dg$$

hold.

For structured documents, the convention gives us operators *DocSort*, *Attributes* and *DocContent*. We need to define three more access operators on structured documents, namely *AttSorts*, *SubDocs* and *ContentString*. For any structured document *di* in *DocInstances*: first, $di.AttSorts$ is the (finite) domain of $di.Attributes$; second, the set of *di*'s subdocuments $di.SubDocs$ forms a set of structured documents, which contains *di* itself and any subdocuments of any structured documents in the string $di.DocContent$; third, $di.ContentString$ is formed from the string $di.DocContent$ by leaving any character as it is and replacing any structured document by its *DocSort*.

For grammars, the convention gives us access operators *ContentRules* and *AttRules*.

Finally, we define two more access operators on structured documents. The operators *IDs* and *IDREFs* collect all values of ID attributes and IDREF attributes, respectively. Both operators need a grammar as additional input, since the information about attribute domains, whether an attribute is an ID attribute, an IDREF attribute or something different, is not specified in the document proper, it is only specified in the grammar.
For any structured document *di* and any grammar *dg*, the set $di.IDs(dg)$ contains $sd.Attributes(as)$, for any subdocument *sd* of *di* in $di.SubDocs$ and for any attribute sort *as* in $di.AttSorts$, such that $dg.AttRules(sd, as) = \texttt{ID}$.
Analogously, the set $di.REFIDs(dg)$ contains $sd.Attributes(as)$ for any subdocument *sd* of *di* in $di.SubDocs$ and any attribute sort *as* in $sd.AttSorts$ such that $dg.AttRules(sd, as) = \texttt{REFID}$.

Table 3 summarizes the access operators we have just defined.

Table 3. A summary of the access operators for grammars, structured documents and grammar-constrained structured documents.

Operand	Operator	Codomain
GCSDocument	*DocInstance*	*DocInstance*
	DocGrammar	*DocGrammar*
DocInstance	*DocSort*	*DocSort*
	Attributes	*Attributes*
	DocContent	*DocContent*
	AttSorts	$\mathcal{P}(AttSort)$
	SubDocs	$\mathcal{P}_{\text{fin}}(DocInstance)$
	ContentString	$(Character \mathbin{\dot\cup} DocSort)^*$
	$IDs(DocGrammar)$	$\mathcal{P}_{\text{fin}}(DocID)$
	$REFIDs(DocGrammar)$	$\mathcal{P}_{\text{fin}}(DocID)$
DocGrammar	*ContentRules*	*ContentRules*
	AttRules	*AttRules*

3.4 The GCSD Model Defined: Validity

To complete the definition of the GCSD model, we explain the conditions under which a structured document conforms to a grammar—a concept that is known as validity in XML terminology. In other words, we define exactly which pairs of structured documents and grammars form a grammar-constrained structured document in *GCSDocument*.

A pair (di, dg) of a structured document *di* and a grammar *dg* is in *GCSDocument* if and only if it satisfies the following four conditions:

1. Each subdocument *sd* of *di* in *di.SubDocs* has a content rule in *dg* and satisfies it; that is, $dg.ContentRules(sd.DocSort)$ is defined and contains $sd.ContentString$.
2. Each subdocument *sd* of *di* in *di.SubDocs* satisfies its attribute rule; that is, first, that the attribute sorts for which values are defined in $sd.Attributes$ are exactly the attribute sorts for which the attribute rule $dg.AttRules(sd, \Box)$ is defined and, second, that the value of the attribute in the subdocument is compatible to the domain in the grammar rule. Formally, two conditions must be satisfied:
   a) An attribute sort *as* in *AttSort* is in $sd.AttSorts$ if and only if $dg.AttRules(sd, as)$ is defined.
   b) For any *as* in $sd.AttSorts$, $sd.Attributes(as)$ is in $dg.AttRules(sd.as)$.
3. ID attribute values must be unique within *di*; that is, for any subdocument *sd* of *di*, if $dg.AttRules(sd, as) = \mathtt{ID}$, then $sd.Attributes(as)$ is not in $sd'.IDs$ for any subdocument sd' of *di* other than *sd*.
4. There are no dangling IDREFs; that is, $di.IDREFs \subseteq di.IDs$.

3.5 The GCSD Model Related to XML

First, in Table 4, we indicate how the GCSD model relates to XML concepts.

Next, we illustrate the relationship between the GCSD model and XML through an example. In Fig. 1, we give an example XML DTD for poems and, in Fig. 2, as an XML-document instance, a poem by Frost.

The Corresponding Poem GCSD. We now define a GCSD in *GCSDocument* that models the XML DTD and the DTD-conforming document instance.

Let *Character* contain all printable ASCII characters (which include the blank character), let the symbols poem, verse, title, author, line and emph be in *DocSort*, let the symbols editor, date, status be in *AttSort* and let the symbols prelim and checked be in *AttValue*.

We define for the 20 lines of the XML document instance 20 structured documents $line\_1, \ldots, line\_20$ of the form $(\mathsf{line}, \emptyset, \texttt{"..."})$, where $\emptyset$ denotes the function in *Attributes* with empty domain and "..." contains the line's text as a catenation of characters. For example,

$$line\_19 = (\mathsf{line}, \emptyset, \texttt{"I took the one less traveled by,"}).$$

Table 4. The relationship between the GCSD model and XML.

GCSD	XML ([< *rulenumber* >])
GCSDocument	*document* [1]
DocInstance	*element* [39]
DocSort	element type, named in the element's tags (corresponds to *Name* in [40, 42, 44, 45, 52])
Attributes	*Attribute* [41]
Content	*content* [43]
DocGrammar	*doctypedecl* [28]
ContentRules	sequence of *elementdecl* [45]
DocDomain	*contentspec* [46]
Reg(*DocSort*)	*children* [47]
RReg(*Character*, *DocSort*)	*Mixed* [51]
AttSort	name of an attribute, corresponds to *Name* in attribute definition [53]
AttDomain	*AttType* [54]
$\mathcal{P}_{\text{fin}}(\mathit{AttValue})$	*EnumeratedType* [57]
`ID`, `IDREF`	instances of *TokenizedType* [56]
`String`	*StringType* [55]
Character	*CharData* [14]

```
<?XML VERSION="1.0"?>
<!DOCTYPE poem [
<!ENTITY % text "(#PCDATA | emph)*">
<!ELEMENT poem   (title,author?,verse+)>
<!ELEMENT verse  (line+)>
<!ELEMENT title  %text;>
<!ELEMENT author %text;>
<!ELEMENT line   %text;>
<!ELEMENT emph   %text;>
<!ATTLIST poem   editor CDATA #REQUIRED
                 date   CDATA #REQUIRED>
<!ATTLIST poem   status (prelim | checked) "prelim">
]>
```

Fig. 1. An XML DTD for poems.

Further structured documents in *DocInstance* are

$$\begin{aligned} \mathit{verse} &= (\mathsf{verse}, \emptyset, \mathit{line\_1} \cdot \cdots \cdot \mathit{line\_20}), \\ \mathit{title} &= (\mathsf{title}, \emptyset, \texttt{"The Road Not Taken"}), \\ \mathit{author} &= (\mathsf{author}, \emptyset, \texttt{"Robert Frost"}) \\ \mathit{poem} &= (\mathsf{poem}, \mathit{edAtts}, \mathit{title} \cdot \mathit{author} \cdot \mathit{verse}), \end{aligned}$$

```
<poem editor = "X.YZ" date='01/01/2000'>
<title>The Road Not Taken</title>
<author>Robert Frost</author>
<verse>
  <line>Two roads diverged in a yellow wood,</line>
  <line>And sorry I could not travel both</line>
  <line>And be one traveler, long I stood</line>
  <line>And looked down one as far as I could</line>
  <line>To where it bent in the undergrowth;</line>
  <line>Then took the other, as just as fair,</line>
  <line>And having perhaps the better claim,</line>
  <line>Because it was grassy and wanted wear;</line>
  <line>Though as for that, the passing there</line>
  <line>Had worn them really about the same,</line>
  <line>And both that morning equally lay</line>
  <line>In leaves no step had trodden black.</line>
  <line>Oh, I kept the first for another day!</line>
  <line>Yet knowing how way leads to way,</line>
  <line>I doubted if I should ever come back.</line>
  <line>I shall be telling this with a sigh</line>
  <line>Somewhere ages and ages hence:</line>
  <line>Two roads diverged in a wood, and I-</line>
  <line>I took the one less traveled by,</line>
  <line>And that has made all the difference.</line>
</verse>
</poem>
```

Fig. 2. A poem-document instance.

where *edAtts* is the function

$$\mathsf{editor} \longrightarrow \texttt{"X.YZ"},\ \mathsf{date} \longrightarrow \texttt{"01/01/2000"},\ \mathsf{status} \longrightarrow \mathsf{prelim}.$$

The XML DTD for poems is modeled as the tuple

$$\mathit{poemGrammar} = (\mathit{poemContentRules}, \mathit{poemAttRules}),$$

where *poemContentRules* is the function

$$\begin{aligned} \mathsf{poem} &\longrightarrow \mathrm{L}(\mathsf{title}(\mathsf{author} \mid \epsilon)\mathsf{verse}\,\mathsf{verse}^*), \\ \mathsf{verse} &\longrightarrow \mathrm{L}(\mathsf{line}\,\mathsf{line}^*), \\ \mathsf{title}, \mathsf{author}, \mathsf{line}, \mathsf{emph} &\longrightarrow (\mathit{Character}\,\dot{\cup}\{\mathsf{emph}\})^* \end{aligned}$$

and *poemAttRules* is the function

$$\begin{aligned} (\mathsf{poem}, \mathsf{editor}) &\longrightarrow \texttt{String}, \\ (\mathsf{poem}, \mathsf{date}) &\longrightarrow \texttt{String}, \\ (\mathsf{poem}, \mathsf{status}) &\longrightarrow \{\mathsf{prelim}, \mathsf{checked}\}. \end{aligned}$$

Finally, the complete XML document with DTD and instance is modeled by the tuple $(poem, poemGrammar)$.

Comments on the Example. The example demonstrates the various ways in which the model abstracts from the syntactic sugar of XML, namely:

– Version information.
– White space in the DTD.
– Order of element declarations.
– Order and grouping of attribute declarations, conflict rules for multiply-defined attributes.
– Physical structure (entities).
– Specification of content models.
– White space between markup tags in document content.
– White space within tags.
– Order of attributes within start tags.
– Difference between the two forms `<XXX/>` and `<XXX></XXX>` of an empty element.
– Character representation (character references) and encoding.
– Delimiters for attribute values.

4 Related Work

The GCSD model borrows heavily from the abstract-data-type (ADT) or object-oriented approach to programming. Its novelty, however, is that it models both a grammar and the document instances of the given grammar.

Note that existing models such as the XPath and XSLT data models [6,7] and the proposed XML Information Set [8] do not cover grammars and secondary structures.

On the other hand, the proposed Canonical XML [2] defines a normalized representation for XML document instances, thus abstracting some of the syntactic variances allowed by XML. But Canonical XML is not a data model; it is a syntactic representation of the XPath data model. The now apparently outdated ESIS structure of James Clark's SGML parser [5] used to play a similar role.

For database applications, a number of models (or better representations) of structured documents have been suggested by researchers [4,10,11].

5 Concluding Remarks

We conclude by briefly mentioning a number of issues.

In the full version of this paper we will discuss design decisions regarding:

– Whitespace in element content.
– Defaults for attributes.
– Normalized attribute values.

– Other attribute types (ENTITY).
– Unambiguous content models.
– EMPTY and ANY content declarations.
– Several IDs per element.
– Name of document type declaration.

We will also define an XML-conforming representation for GCSDs that provides round-trip conversion from the model to XML and back.

In ongoing work we focus our attention on the implementation of the GCSD model. It involves the specification and implementation of document classes (in the sense of object orientation, not of DTDs) for various groups of requirements of functionality and performance.

We also need to develop, among other things, a model of an XML processor that builds the abstract document structure. Our XML processor follows the well-established scheme of programming-language parsers; large parts of it can be generated by tools from specifications, so they do not have to be manually programmed. To achieve this goal, we reformulate the language rules of XML so that we can express XML processing in terms of the standard phases of compiler construction, essentially re-engineering XML. In particular, our XML processor separates lexical analysis (tokenization) from syntactic analysis (parsing). Our model also deals with the novel (compared to programming languages) requirement of XML that the data are given as a collection of storage units and that the physical organization of these units is discovered only during XML processing.

Lastly, we believe it will be a very useful (and challenging) exercise to also re-engineer other Web standards such as XML Schema and XSLT.

References

1. P.V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/, October 2000. W3C Candidate Recommendation.
2. J. Boyer. Canonical XML 1.0. http://www.w3.org/TR/2000/WD-xml-c14n-20000601/, June 2000. W3C Working Draft.
3. T. Bray, J. P. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210/, February 1998.
4. L. J. Brown, M. P. Consens, I. J. Davis, C. R. Palmer, and F. W. Tompa. A structured text ADT for object-relational databases. *Theory & Practice of Object Systems*, 4(4):227–244, 1998.
5. J. Clark. Sp: An SGML system conforming to international standard ISO 8879 – standard generalized markup language. http://www.jclark.com/sp/.
6. J. Clark. XSL Transformations (XSLT) 1.0. http://www.w3.org/TR/1999/REC-xslt-19991116/, November 1999.
7. J. Clark and S. DeRose. XML Path Language (XPath) 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116/, November 1999.

8. J. Cowan and D. Megginson. XML Information Set. http://www.w3.org/TR/2000/WD-xml-infoset-20000726/, July 2000. W3C Working Draft.
9. R. Furuta, V. Quint, and J. André. Interactively editing structured documents. *Electronic Publishing—Origination, Dissemination and Design*, 1(1):19–44, April 1988.
10. M. Gyssens, J. Paredaens, and D. van Gucht. A grammar-based approach towards unifying hierarchical data models. *SIAM Journal of Computing*, 23(6):1093–1137, December 1994.
11. B. Lowe, J. Zobel, and R Sacks-Davis. A formal model for representation and querying of structured documents. *Journal of Systems Integration*, 7:31–46, 1997.
12. N. Meyrowitz and A. van Dam. Interactive editing systems (Parts I and II). *Computing Surveys*, 14(3):321–415, September 1982.
13. M. Murata. Transformation of documents and schemas by patterns and contextual conditions. In C. Nicholas and D. Wood, editors, *Proceedings of the Third International Workshop on Principles of Document Processing (PODP 96)*, pages 153–169, Heidelberg, 1997. Springer-Verlag. Lecture Notes in Computer Science 1293.
14. M. Murata. Data model for document tranformation and assembly. In E. V. Munson, C. Nicholas, and D. Wood, editors, *Proceedings of the Fourth International Workshop on Principles of Digital Document Processing (PODDP 98)*, pages 140–152, Heidelberg, 1998. Springer-Verlag. Lecture Notes in Computer Science 1481.
15. C.M. Sperberg-McQueen and C. Huitfeldt. GODDAG: A data structure for overlappting hierarchies, 2001. Proceedings of the Fifth International Workshop on Principles of Digital Document Processing (PODDP00). To appear.
16. H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/, October 2000. W3C Candidate Recommendation.

Overview of Tree Transducer Based Document Transformation System

Eila Kuikka, Paula Leinonen, and Martti Penttonen

University of Kuopio, P.O.Box 1627, FIN-70211 Kuopio, Finland
{Eila.Kuikka, Martti.Penttonen, Paula.Leinonen}@uku.fi
http://www.cs.uku.fi/~kuikka/Multi/transformation.html

Abstract. We characterize a class of document structure transformations from context-free structures to context-free structures, which can be implemented by a two-phase semi-automatic procedure. The main feature of the transformations is locality. In the first phase of our method, corresponding substructures are searched by an interactive procedure. In the second phase, the replacement of substructures is automatized by generating a tree transducer implementing the transformation. The feasibility of the method is confirmed by a preliminary implementation.

1 Introduction

Adding a logical structure to documents facilitates the managing of them in many ways. On the other hand, structure transformations of documents are needed, because even for documents within the same document type, like an article or a letter, there are different structure definitions. It may be possible to define an automatic structure transformation, if the logical structure of the documents is marked up and differences between the structures are restricted.

The Standard Generalized Markup Language (SGML) [9,7] is a syntactic metalanguage that provides the notation for defining textual markup systems. The Extensible Markup Language (XML) [3], a subset of the SGML, has been specially designed for producing and managing structured documents on Web. Those kinds of markup systems can be specified by an extended context-free grammar [17]. The extended Backus-Naur Form (BNF) of the context-free grammar is suitable and widely used for defining the structure of the documents. If the structure of the document is defined using a grammar, tree transformation methods can be used for the structure transformations. The tree transformation should be determined using the grammars for the source and target documents and the method should be automatic, or at least semi-automatic.

Some studies about general structure transformations of documents have been done. Furuta and Stotts specify the problem in [5]. Akpotsui and Quint [1] use a comparator tool to compare two grammars for creating conversion rules, which are used by a converter. Most of the few existing applications are quite restricted or the user has to undertake much manual work to define a transformation. For a transformation between two structures, there are some systems based on a Syntax-Directed Translation Schema (SDTS) [2], like SYNDOC [15, 12,16], HST [10], Integrated Chameleon Architecture (ICA) [19] or the system of Chiba and Kyojima [4]. Attribute grammars [6] and tree transformation grammars [14] are used in structure transformations for increasing the power. A tree transformation grammar is an extension of a syntax-directed translation schema. It describes an association between the source and the target grammars. In addition to defining an association between the nonterminals, the user defines an association between groups of productions. The tree transformation grammar is not restricted to a single level in a derivation tree like the SDTS is. Another approach is given in [20], where Murata specifies a model for transforming both documents and their structure definitions.

Our aim in this work is to present an experimental implementation for a semiautomatic procedure defining an automatic structure transformation [18]. In an incremental procedure, the transformation is defined by example documents in dialogue with the system. The completeness of the procedure is guaranteed by characterizing a sufficient set of examples, called characteristic trees.

After this introduction, in Sect. 2, we define associations between the tree structures of two documents. Section 3 describes how the rules of the tree transducer are constructed from the definition of the transformation. In Sect. 4 we present the implementation of the transformation system. Finally, Sect. 5 summarizes the work.

2 Associations in Structures

In this section we present a semiautomatic procedure for constructing a corresponding pair of substructures from structure trees of documents. A more detailed description is given in [18].

A *context-free grammar* consists of the set N of nonterminals, one of which is the start symbol of the grammar, the set Σ of terminals and a set of rules. Rules are written in the form $A \rightarrow a_1 a_2 \ldots a_n$, where A is a nonterminal and $a_1, a_2, \ldots, a_n$ are either terminals or nonterminals. In the syntax-directed approach to text processing, a grammar describes the structure of the document. In structured document processing, it is useful that the grammar has special kinds of nonterminals called *content nonterminals*. These nonterminals do not have rules in the grammar, but they refer to the contents of the document. Usually the content nonterminal is the only nonterminal of the right-hand side of the rule, and hence does not have siblings in the derivation tree. In structured document processing, the actual character data content is not modeled with the grammar.

The rules of the grammar are used to derive documents. Such a derivation is expressed with a *derivation tree* as follows. The start symbol S of the grammar is represented by a node labeled with S, that will be the *root* of the tree. In the beginning, when no rules have been applied, S is also a *leaf* of the derivation tree. Later, when a leaf of the tree is labeled with a nonterminal A, and there is a rule $A \rightarrow x_1 x_2 \ldots x_k$, where $x_i \in N \cup \Sigma$ $(i = 1, \ldots, k)$, the derivation tree is extended with new leaves labeled with $x_1, x_2, \ldots, x_k$ that are connected to A. The node labeled by A becomes an internal node. A string constructed by concatenating the labels of the leaves of the derivation tree from left to right is called the *frontier* of the tree.

We use a concept of characteristic trees for defining an application of the label association to infinite numbers of derivation trees of the grammar in a finite manner. We call *elementary trees* the set of derivation trees that have the start symbol S of the grammar as the root, terminals, empty string ϵ or content nonterminals as leaves, and no nonterminal occurring twice in any of the paths from the root to a leaf. We call *pumping factors* the derivation trees that have a nonterminal as the root and the same nonterminal as a leaf (or many leaves) and this is the only case when a nonterminal occurs twice on a path from the root to a leaf. *Characteristic tree* is the common name for elementary trees and pumping factors.

Because derivation trees consist of characteristic trees and the tree transformation can be made in parts, an automatic transformation for all derivation trees can be derived from the transformation definition between characteristic trees of the source grammar and corresponding structures of the target grammar. Repeating substructures are repeating both in source and target documents. For this reason, it is reasonable to assume that elements in a pumping factor are mapped to elements in a pumping factor.

As an example in this paper, we consider two different grammars for an article. They are represented in Fig. 1 as the source and target grammars. The nonterminals, which do not have any corresponding nonterminal in the other grammar, have been represented using a bold typeface.

Source grammar:

article	->	authors title **content**
article	->	authors date title **content**
authors	->	author authors
authors	->	ε
author	->	text
date	->	text
title	->	text
content	->	abstract sections
sections	->	section sections
sections	->	section
abstract	->	text
section	->	heading paragraphs
paragraphs	->	paragraphs paragraph
paragraphs	->	paragraph
heading	->	text
paragraph	->	text

Target grammar:

article	->	**titlepart** authors **intro** **body**
article	->	authors **titlepart** date **intro** **body**
titlepart	->	title **shorttitle**
authors	->	author authors
authors	->	ε
author	->	text
intro	->	abstract **keywords**
title	->	text
shorttitle	->	text
date	->	text
body	->	**preface** sections
sections	->	section sections
sections	->	section
abstract	->	text
keywords	->	text
preface	->	paragraphs
section	->	heading paragraphs
paragraphs	->	paragraphs paragraph
paragraphs	->	paragraph
heading	->	text
paragraph	->	text

Fig. 1. The source and target grammars for an article

Some of the characteristic trees of the source grammar, one elementary tree (a) and all the pumping factors (b), are described in Fig. 2.

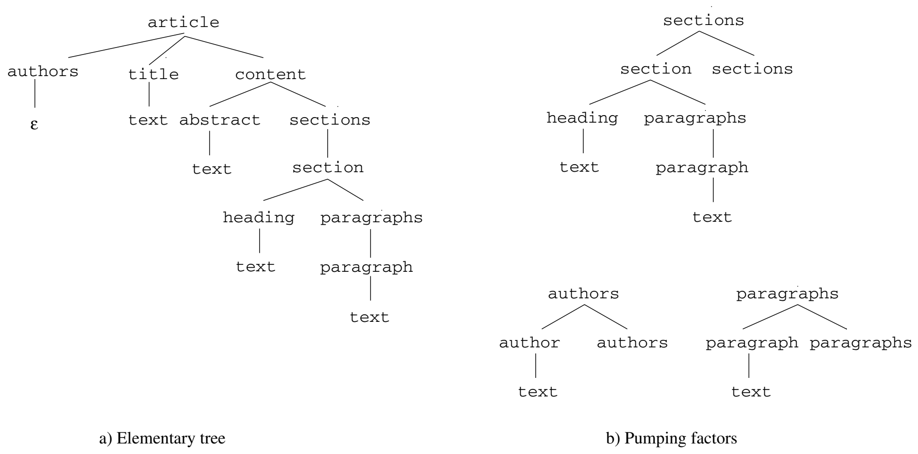


Fig. 2. Characteristic trees for the source grammar of Fig. 1

A *label association* is a relation h from the set N_s of the nonterminals of the source grammar G_s to the set N_t of the nonterminals of the target grammar G_t. Relation h is not necessarily a mapping, because the member n of the domain N_s may have several images in the range N_t. The concept of the label association is used to show the semantic similarity between the corresponding structure elements of two documents by associating the corresponding nonterminals of the source and the target grammars. The label association alone does not determine the transformation but it helps to find matching substructures.

In the example grammars of Fig. 1, the nonterminals with identical labels are associated. In the source grammar, node `content` has no associated nonterminal in the target grammar. The target grammar contains nonterminals `titlepart`, `intro`, `shorttitle`, `body`, `keywords` and `preface` which do not have associated nonterminals in the source grammar.

After the label association is defined, corresponding substructures of the documents may be possible to generate from the substructure of the source grammar and rules of the target grammar. If there is only one rule for all the nonterminals of the target grammar, at most one corresponding structure tree for every elementary tree or pumping factor of the source grammar will be generated. This can be done automatically by starting from the root and continuing until the leaves are content nonterminals or empty strings ϵ. If there are several rules for the nonterminal n_t of the target grammar, the rule can be selected automatically if only one rule $n_t \to n_{t_1}, n_{t_2}, ..., n_{t_k}$ contains associated nonterminals with the rule $n_s \to n_{s_1}, n_{s_2}, ..., n_{s_l}$ of the source grammar which is used in the derivation tree and in which n_s is associated with the nonterminal n_t. Otherwise, the user selects a suitable rule.

Consider two grammars G_s and G_t and a label association between these grammars. A *tree association* is a relation between the nodes of a derivation tree by grammar G_s and a derivation tree by grammar G_t such that the labels of related nodes are related in label association. The tree association is only meaningful when comparing the derivation trees of the same document by two grammars. If the grammars are structurally identical, the tree association is a bijective mapping. If the grammars are very different, the tree association is very partial and there is no obvious transformation of the document structure. Note that our definition allows even an empty relation to be called a tree association. Such a tree association is, of course, useless from the point of view of transformation.

We say that a tree association h is *hierarchic* if for any $x \in h(u)$, $y \in h(v)$, if v is a descendant of u, y is a descendant of x. A tree association is *dense*, if there is a constant c such that any non-root node with a nonempty label association has an ancestor with a nonempty label association within distance c. A tree association h is *local*, if there are constants d and e such that for all $x \in h(u)$, $y \in h(v)$ such that the distance of u from v is not greater than d, the distance of x from y is not greater than e. Intuitively speaking, having a hierarchic, dense and local tree association means that the transformation can be built top-down with a finite look-ahead in the tree.

Figure 3 describes a tree association between the nodes of two characteristic trees of the source grammar and the nodes of corresponding structures of the target grammar of Fig. 1.

Now we will describe a procedure to build a pair of substructures of the source and the target trees so that a label association is respected as much as possible. A sketch of the algorithm already appeared in [13]. Given grammars G_s and G_t and a subtree T_s of a structured document derived by G_s, so that the root node has a nonempty label association. If there is a subtree of a structured document T_t derived by G_t and a hierarchic, dense and local tree association between T_s and T_t, subtrees are associated by the following procedure:

Tree Association Algorithm

Associate the root n of T_s and a node $h_l(n)$ of T_t in the tree association h_t. Given grammars G_s and G_t, a subtree T_s of a derivation tree in G_s and a label association h_l such that the root node of T_s is mapped to a nonterminal of G_t.

1. Expand n in T_s until in each branch a node m with a nonempty association $h_l(m)$ or a node without associated descendants in its subtree is found. (Due to the denseness assumption, these nodes are found in bounded depth.) The nodes in the frontier of the expanded tree having a nonempty association are marked open, all other generated nodes are marked closed.
2. Expand the node $h_t(n)$ in a subtree T_t so that the open nodes generated in 1 can be associated with the nodes of the expanded subtree of $h_t(n)$ in h_t.

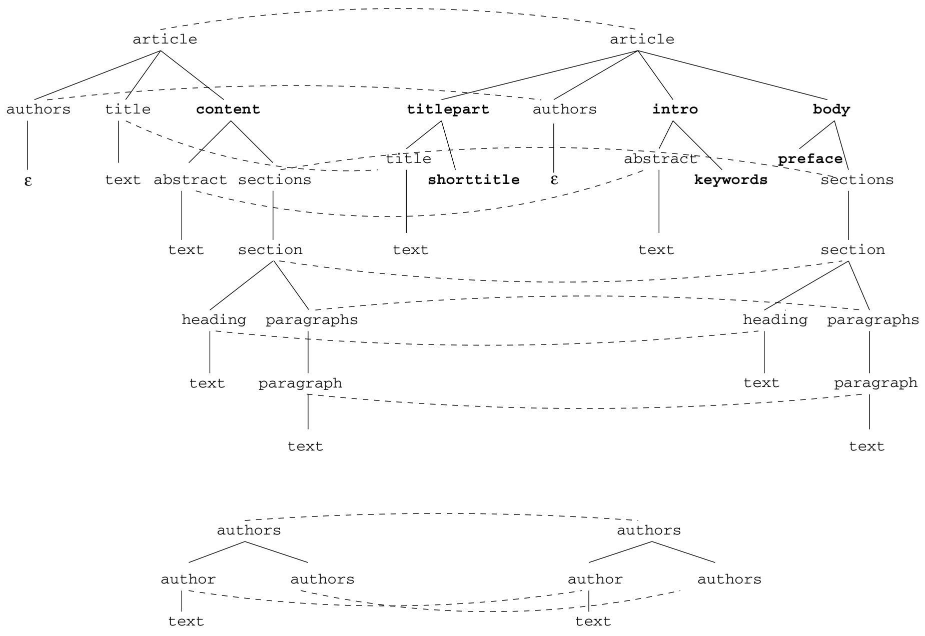


Fig. 3. Tree association

(Such nodes can be generated in finite time, because there is a hierarchic and local association.)

Step 2 is nondeterministic, as there may be several ways to expand T_t, some of which may not lead to the required tree association. However, assuming that there is a hierarchic, dense and local tree association, it can be found in a finite time by backtracking. In the real implementation the search is guided by a suitable heuristic function providing "goodness" values for subtree candidates.

3 Automatizing the Transformation

In the previous section, an interactive procedure for constructing a tree association was presented, when it is known that there is a hierarchic, dense and local transformation. The procedure can be applied to any document tree. However, the system should be as automatic as possible. We assume that the transformation is deterministic in the sense that substructures are transformed in the same way in all cases. Then, once the transformation of a structure is fixed, it can be automatically applied later on. This automatic transformation is formalized by tree transducers.

A tree transducer is a tree automaton with an output. It works like a finite state automaton, but operates on trees, not on strings. It takes a finite labeled tree as an input and produces a finite labeled tree as an output. A *descending tree transducer* [8] consists of a terminal input and output alphabet, a ranked input and output alphabet, a set of states which contains a set of start states, and a set of transition rules. Each transition rule is defined for a subtree in the source tree and a state. The subtrees of the source tree and corresponding subtrees of the target tree may consists of a node and its children, but be also a tree with more than one hierarchy level. A formal representation of an ordered labeled tree, a term, is used to define the subtree. The states are inserted into the constructed structure for defining the unprocessed parts of the source tree. They can be used to force different combinations of the transformation at different subtrees. The (descending) tree transducer starts processing the source tree from the root towards the leaves and constructs the target tree structure by structure. For the node with associated state, the proper transformation rule is selected and the structure is transformed according to the rule.

The outcome of the tree association is that a subtree $t(X_1, \ldots, X_k)$ should be replaced by a subtree $t'(Y_1, \ldots, Y_l)$, where X_i are the nodes in the frontier of t that have a label association, and Y_j is a label associated with some of $X_1, \ldots, X_k$. A problem in automatic transformation may be that in different situations $t(X_1, \ldots, X_k)$ may be replaced by different subtrees. Therefore, a state control in the finite tree transducer is useful, and we should have rules of the form

$$q : t(X_1, \ldots, X_k) \to t'(q_1 : Y_1, \ldots, q_l : Y_l).$$

The state-tree pair $q : t$ is unique and has only one right-hand side. These rules are formed by scanning the source tree from top to bottom. At the beginning we only know the initial state q_0. During the process we use state variables $x, y, \ldots$ as "place holders" before we know what states will be ultimately used.

Transducer Construction Algorithm

Given grammars G_s and G_t, a derivation tree T_s in G_s and a label association h_l such that the start symbol of G_s is mapped to the start symbol of G_t.

1. Associate the roots of T_s and T_t in the tree association h_t and mark them as *open* nodes. The state of the root is the initial state q_0.
2. If there are no more open nodes in T_s, stop. Otherwise choose an open node n and its associated node $h_t(n)$.
3. Let n be an associated node and x its state variable. (If n is the root of the whole tree, x is already bound to the initial state q_0.) Using the tree association algorithm, find a tree $t(X_1, \ldots, X_k)$ rooted at n and its matching tree $t'(Y_1, \ldots, Y_l)$, where $X_1, \ldots, X_k$ and $Y_1, \ldots, Y_l$ are the associated labels. If there is a transition $q : t(X_1, \ldots, X_k) \to t'(y_1 : Y_1, \ldots, y_l : Y_l)$ in the transition table, bind x to q and associate state variables y_j to nodes with labels Y_j. If there was no rule of the form $q : t(X_1, \ldots, X_k) \to t'(y_1 : Y_1, \ldots, y_l : Y_l)$ in the transition table, create a new state symbol q and add rule $q : t(X_1, \ldots, X_k) \to t'(y_1 : Y_1, \ldots, y_l : Y_l)$ to the transition table, where

y_j are new state variables. The node n is now closed and the nodes $n_1, \ldots, n_l$ corresponding to $Y_1, \ldots, Y_l$ are open.
4. Go to 2.

Note that all state variables will be ultimately bound. If in a rule

$$q : t(X_1, \ldots, X_m) \to t'(y_1 : Y_1, \ldots, y_n : Y_n), m = 0,$$

then q is a final state.

Our strategy to transform document trees interactively is *lazy*: always try the automatic transformation by the tree transducer first, and if it fails, use the interactive procedure described in Sect. 4 below to add rules to the transducer accordingly. The other approach is the *eager* strategy: transform first all (finitely many) characteristic trees interactively. After adding respective rules to the tree transducer, all trees can be transformed. The advantage of the lazy strategy is that it is easy, but one has to return back to the interactive procedure every time a new type of substructure is met. The eager strategy produces a complete transducer at once, but after a big effort.

4 Implementation of the Semi-automatic Transformation

In this section, we introduce the implementation of the semi-automatic tree transformation system. The prototype has been developed jointly with SYNDOC [16] which is a research prototype for a syntax-directed document processing system. In SYNDOC, the document is considered as a tree structure in all phases of processing. It defines the internal structure of the document with a context-free *skeleton grammar*. In the input phase of the text processing an editor in SYNDOC uses the input grammar to prompt the structure elements to the user. The internal representation of the document is the derivation tree of the skeleton grammar. The formatting of the document is generated with the use of an output grammar. In addition to the tools for creating and formatting documents, SYNDOC provides a simple facility for transforming them — documents can be transformed to another format if the structure does not change or changes only in the limits of the SDTS. Omissions and additions and changes of order are allowed, but deeper changes are not possible. There is also a tool for generating the transformation program for manually written rules of the tree transducer.

The semi-automatic transformation system (Fig.4) consists of two parts, a construction of the corresponding substructures of the source and target documents and generation of the tree transducer for transforming the source document. The system takes the source and target grammars as input together with a document derived from the source grammar. As an example of the construction of the transformation, we consider source and target grammars which are described in Fig. 1.

The system creates a preliminary label association based on the labels of the nonterminals in the grammars. The user can change this automatic default

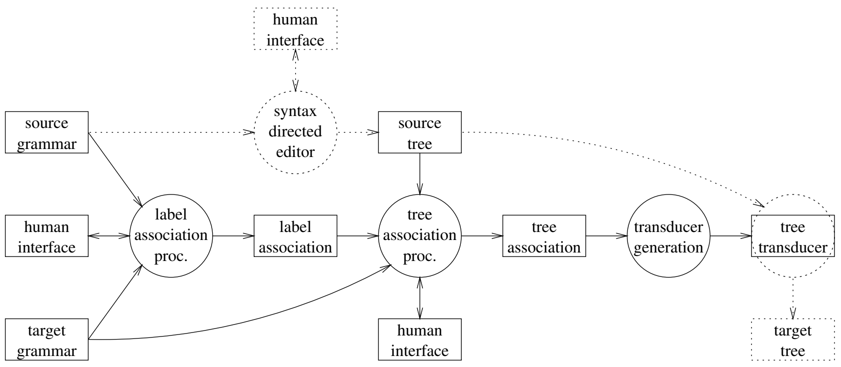


Fig. 4. Architecture of the document transformation system

association by inserting new associated nonterminals for the nonterminal of the source grammar or removing them. Between our example grammars, the label association is obvious; nonterminals with the same labels are associated.

A document, which is going to be transformed, is a derivation tree of the source grammar. A significant part of the document from the point of view of the transformation is expressed as a tree in Fig. 5. The rest of the source and target trees have identical structures according to the grammars.

The tree association algorithm of Sect. 2 is used interactively to find the corresponding structures of the documents of two grammars. The system generates a minimal substructure from the source derivation tree and from the label association. After that, corresponding target substructure is derived with the help of the user from the rules of the target grammar and from the label association. If there are several rules for the nonterminal in the target structure, the system uses a simple function for finding a rule which has the best correspondence with the rule used in the source structure. The function counts the number of the corresponding nonterminals of the rules. In addition to this, it

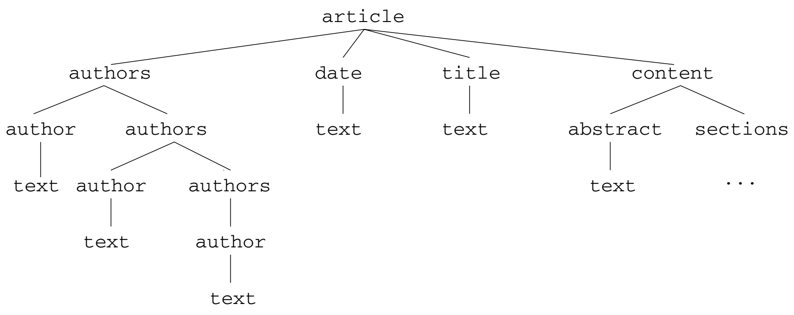
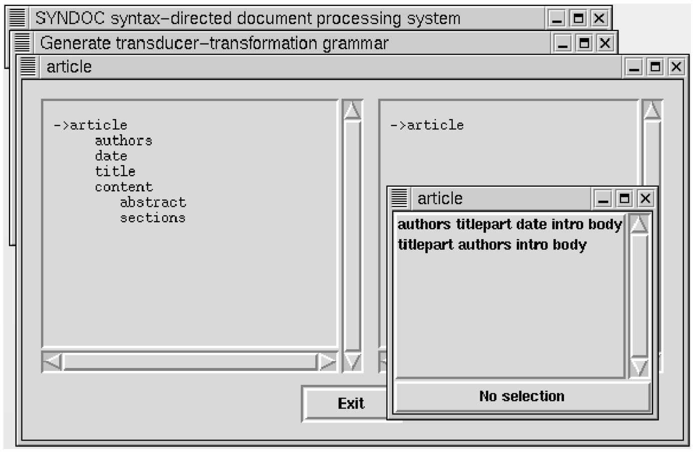


Fig. 5. Derivation tree of the source grammar


SYNDOC syntax–directed document processing system
Generate transducer–transformation grammar
article
Exit
article
No selection

```
->article
    authors
    date
    title
    content
       abstract
       sections
```

```
->article
```

authors titlepart date intro body
titlepart authors intro body

Fig. 6. Interactive construction of substructures

takes into account nonterminals which are missing or are additional and an order of associated nonterminals. The current version of the system only determines the priority between one pair of rules. In the future, we plan to generate larger tree structures for which the priorities are estimated. The system presents the possibilities to the user in the order from the best to the worst, and the user makes the final choice by selecting the rule which is to be used to expand the target structure. Figure 6 shows how the system prompts the user to select a suitable rule from the list in the situation where the nonterminal `article` has two possible rules in the target grammar.

Corresponding structures are searched for all the substructures of the source derivation tree. After that, rules of the tree transducer with necessary states are generated automatically from the pairs of the substructures according to the tree transducer construction algorithm in Sect. 3. The left hand side of the rule is constructed from the substructure of the source grammar and from the state `q0`, `q1`, ... The right hand side of the rule is constructed from the corresponding substructure of the target grammar. The associated leaf nodes of the substructures are denoted using a state and a variable `X1`, `X2`, ... The leaf nodes with an empty association are expressed using their labels. From the substructure pairs the system generates the tree transducer rules of Fig. 7.

The transformation program is generated automatically from the rules of the tree transducer. An algorithm for the transformation can be found from [13]. The system starts the transformation from the root node of the source derivation tree of Fig. 5 to which the start state `q0` of the transducer is assigned. It uses the rules of Fig. 7 for transforming the document to be according to the target grammar of Fig. 1. Rule 1 with the start state `q0` is applied first. Fig. 8 shows a derivation

```
 1. q0(article(X1,X2,X3,content (X4,X5))) -> article(q1(X1),titlepart(q4(X3),shorttitle),q3(X2),
              intro(q6(X4),keywords), body( preface ,q5(X5))).
 2. q0(article(X1,X2, content(X3,X4))) -> article( titlepart(q4(X2),shorttitle),q1(X1),
              intro(q6(X3),keywords), body( preface ,q5(X4))).
 3. q1(authors(X1,X2))      ->   authors(q2(X1),q1(X2)).
 4. q1(authors(X1))         ->   authors(q2(X1)).
 5. q2(author(X1))          ->   author(q11(X1)).
 6. q3(date(X1))            ->   date(q11(X1)).
 7. q4(title(X1))           ->   title(q11(X1)).
 8. q6(abstract(X1))        ->   abstract(q11(X1)).
 9. q5(sections(X1))        ->   sections(q7(X1)).
10. q5(sections(X1,X2))     ->   sections(q7(X1),q5(X2)).
11. q7(section(X1,X2))      ->   section(q9(X1),q8(X2)).
12. q9(heading(X1))         ->   heading(q11(X1)).
13. q8(paragraphs(X1))      ->   paragraphs(q10(X1)).
14. q8(paragraphs(X1,X2))   ->   paragraphs(q10(X1),q8(X2)).
15. q10(paragraph(X1))      ->   paragraph(q11(X1)).
16. q11(text(X1))           ->   text(X1).
```

Fig. 7. Rules of the tree transducer

tree after Rule 1 is applied. The system continues applying of the rules until there are no states left in the derivation tree.

The transformation system is implemented with BinProlog 5.75 [22] and the graphical user interface with Tcl/Tk 8.1 [21]. An idea of representing tree structures, like documents and their substructures as a list was adopted from [11]. This data structure was used also in earlier parts of the SYNDOC system [15] in the input phase of the document.

We have tested several different types of transformations with small grammars, like those of Fig. 1. In test cases, we have made changes to the same level in the structure trees by changing the order, inserting and deleting nodes. In addition, we have inserted and deleted levels in the tree and tested more complicated transformations where the nonterminals are grouped in different ways in the source and target grammars. If the node association between the grammars is easy to define, for example the labels of corresponding nonterminals are identical and for each nonterminal, there is only one rule in the grammar, the

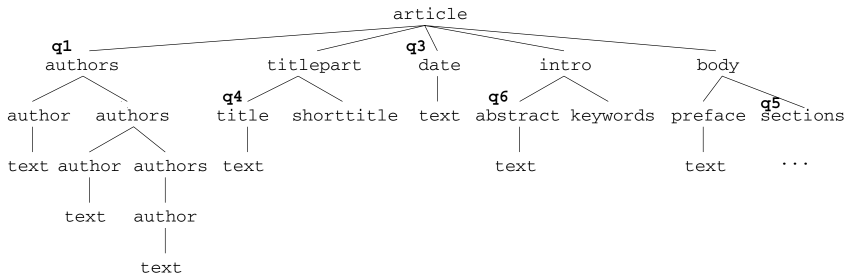


Fig. 8. Transformation by the tree transducer

procedure generates transformations automatically. On the other hand, if the labels of the nonterminals differ and structures have big differences, for example the elements are grouped in different ways, the user will need some tools for helping to define the associations.

5 Conclusions

Marking up document structures is becoming more and more common. Consequently there is also increasing need of structure transformation. We believe that a non-expert user can relatively easily recognize if a document obeys a structure definition and even manually transform a single document from a structure to another, but we think that it is too difficult for a non-expert user to automatize the transformation. We propose a two-phase method, where the user first defines corresponding substructures in an interactive session, and in the second phase, the system generates a tree transducer that realizes such transformations automatically.

Our first experimental implementation of the automatic generation of transducers is encouraging and we are working towards a more complete system. The most critical part in a transformation definition is the correct mapping between the nonterminals. We believe that an iterative process for defining corresponding nonterminals and structures under the control of the computer and a function for approximating priorities of possible structures can help the user in this process.

References

1. E. Akpotsui and V. Quint. Type transformations in structured editing systems. In C. Vanoirbeek and G. Coray, editors, *EP92, Proceedings of Electronic Publishing, International Conference on Electronic Publishing, Document Manipulation, and Typography*, pages 27–41, Lausanne, Switzerland, 1992. Cambridge University Press.
2. A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing.* Prentice-Hall, Englewood Cliffs, N.J., USA, 1972.
3. T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. Available at `http://www.w3.org/TR/1998/REC-xml-19980210`, 1998.
4. K. Chiba and M. Kyojima. Document transformation based on syntax-directed tree translation. *Electronic Publishing – Origination, Dissemination, and Design*, 8(1):15–29, 1995.
5. R. Furuta and P.D. Stotts. Specifying structured document transformations. In J.C. van Vliet, editor, *Document Manipulation and Typography*, The Cambridge Series on Electronic Publishing, pages 109–120, Nice, France, 1988. Cambridge University Press.
6. A. Feng and T. Wakayama. SIMON: A grammar-based transformation system for structured documents. *Electronic Publishing – Origination, Dissemination, and Design*, 6(4):361–372, 1993.
7. C.F. Goldfarb. *The SGML Handbook.* Oxford University Press, Oxford, UK, 1990.

8. F. Gécseg and M. Steinby. *Tree Automata.* Académiai Kiadó, Budabest, 1984.
9. ISO 8879. Information processing - Text and Office Systems - Standard Generalized Markup Language (SGML). ISO, Geneva, 1986.
10. P. Kilpeläinen, G. Lindén, H. Mannila, and E. Nikunen. A structured document database system. In R. Furuta, editor, *EP90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, The Cambridge Series on Electronic Publishing, pages 139–151, Gaithersburg, Maryland, 1990. Cambridge University Press.
11. H.J. Komorowski and J. Małuszyński. Logic programming and rapid prototyping. *Science of Computer Programming*, 9:179–205, 1987.
12. E. Kuikka and M. Penttonen. Transformation of structured documents. *Electronic Publishing – Origination, Dissemination, and Design*, 8(4):319–341, December 1995.
13. E. Kuikka and M. Penttonen. Transformation of structured documents. Technical Report CS-95-46, University of Waterloo, Department of Computer Science, 1995.
14. S.E. Keller, J.A. Perkins, T.F. Payton, and S.P. Mardinly. Tree transformation techniques and experiences. *SIGPLAN Notices*, 19(6):190–201, 1984.
15. E. Kuikka, M. Penttonen, and M.-K. Väisänen. Theory and implementation of SYNDOC document processing system. In *Proceedings of the Second International Conference on Practical Application of Prolog, PAP-94*, pages 311–327, London, UK, 1994.
16. E. Kuikka. *Processing of Structured Documents Using a Syntax-Directed Approach.* PhD thesis, Kuopio University Publications C. Natural and Environmental Sciences 53, 1996.
17. P. Kilpeläinen and D. Wood. SGML and exceptions. In *Proceedings of the Workshop on the Principles of Document Processing, PODP96*, Lecture Notes in Computer Science, Vol. 1293, pages 39–49, Palo Alto, California, 1997. Springer-Verlag.
18. P. Leinonen, E. Kuikka, and M. Penttonen. An approach to document structure transformations. Accepted to be represented in World Computer Congress 2000 (ICS2000 International Conference on Software), August 21-25, 2000, Beijing, China, 2000.
19. S. Mamrak, C.S. O'Connell, and J. Barnes. *Integrated Chameleon Architecture.* PTR Prentice Hall, Englewood Cliffs, N.J., USA, 1994.
20. M. Murata. Data model for document transformation and assembly. In *Proceedings of the Workshop on Principles of Digital Document Processing, PODDP'98*, Lecture Notes in Computer Science, Saint Malo, France, 1998. Springer-Verlag.
21. J.K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1994.
22. P. Tarau. BinProlog 5.75 User Guide. Technical Report 97-1, Département d'Informatique, Université de Moncton, 1997. Available at `http://clement.info.umoncton.ca/BinProlog`.

Browsing Agents: Automated Browsing of Distributed Information

Heather Brown, Fred Cole, Zarine Kemp, and Ning Li

Computing Laboratory,
University of Kent,
Canterbury CT2 7NF, UK

Abstract. This paper describes a toolkit for finding and remembering information in a distributed environment of rapidly-changing information sources. In these conditions users often expend significant amounts of effort searching for useful documents and objects, only to find that their information quickly becomes out-of-date and they need to start all over again.
Browsing agents are designed to minimise this wasted effort by helping users to create and execute complex queries that mimic many aspects of browsing and searching. The queries may be stored and repeated later without user intervention.

Keywords: Multimedia indexing, Searching, Browsing, Structured documents

1 Introduction

Finding useful and relevant information in a distributed environment involves considerable effort. Saving this information and checking for new or updated information can become a significant burden. Browsing agents reduce the effort involved by allowing users to create, execute, and repeat complex queries that automate many of the time-consuming features of browsing and searching. Queries can be applied to composite multimedia objects and structured documents as well as text and image objects.

Browsing agents were developed as part of a project to investigate techniques for managing large-scale personalised indexes for multimedia objects. The project initially provided a tool for creating and manipulating a hierarchical index of objects (particularly MHEG [1] objects). We called the tool a *virtual database* because it allowed users to manage and perform queries on objects that appeared to be stored within the system – but actually resided on remote machines. It allowed users to attach personal multimedia *cues* to the objects as aids for browsing and provided *search objects* for applying simple search queries to these cue values.

As the work progressed it became clear that much of the work of maintaining an up-to-date virtual database involved repeated browsing and searching sessions. Search objects helped with organisation and retrieval but were limited to internal use because they operated on the cues in the virtual database rather than on the external objects. Browsing agents were the result of later work to develop a model for automated browsing and searching that could extend beyond the confines of the virtual database.

The paper is arranged as follows: Section 1 sets the scene with a brief introduction to the main features of the virtual database; Section 2 provides an overview of automated browsing and gives an outline of the agents' structure and the types of browsing and querying supported; Sections 3 and 4 describe the agents in more detail and provide examples to show how users create complex queries; Section 5 discusses implementation details; and Section 6 evaluates the strengths and weaknesses of browsing agents and compares their facilities with other solutions.

Although virtual databases and browsing agents can handle many different protocols and object types, the examples used in this paper focus on HTML [2] documents and the World Wide Web [3] because these will be familiar to most readers.

2 The Virtual Database

A virtual database provides an index made up of a hierarchy of directories, references, and browsing agents. In most cases the references are to remote multimedia objects accessible over networks. Details of the locations of the objects, the protocols used to access them and the software needed to display them are hidden from the user who sees all objects as local (except for the time taken to display them).

The user interface to a virtual database is similar to a graphical user interface to a hierarchical filing system. The user moves around the hierarchy by clicking on objects or scrolling through the contents of a directory. Simple interactive facilities are available for adding, deleting, and moving objects and for creating and changing their attributes.

The project did not set out to develop new techniques for recognising or classifying multimedia objects. Instead, it attempted to show how existing techniques could be combined and harnessed for multimedia indexing on a large scale. Thus the virtual database provides a framework for existing browsing and searching methods, and is designed so that a different method can be incorporated by changing an entry in a configuration file. The prototype system has borrowed several concepts from the MIT semantic file store [4] and uses existing public domain software wherever possible.

This paper does not attempt to describe the virtual database in detail, but the main features of the original system – cues, transducers, references, and search objects – are introduced briefly below in order to explain the motivation for browsing agents and the environment that they work in.

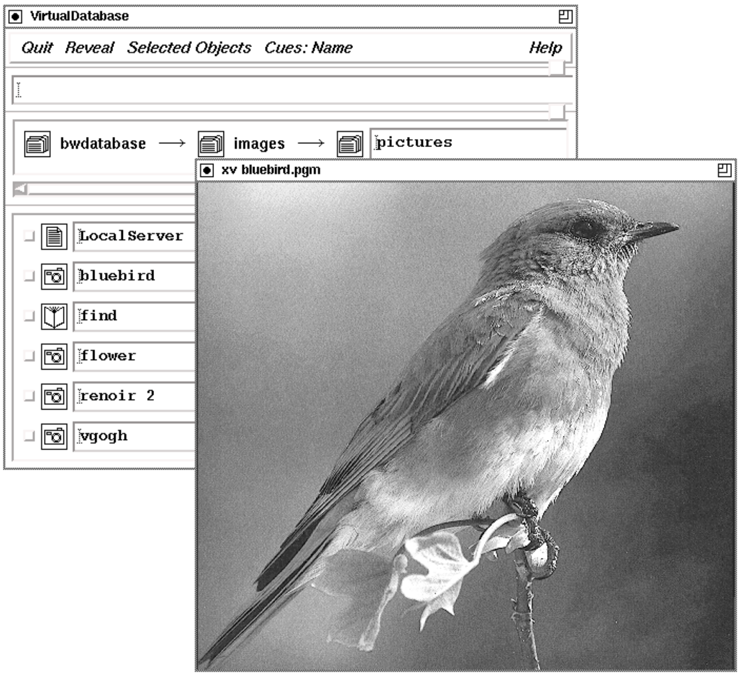

VirtualDatabase
Quit Reveal Selected Objects Cues: Name Help
bwdatabase → images → pictures
LocalServer
bluebird
find
flower
renoir 2
vgogh
xv bluebird.pgm

Fig. 1. Virtual Database Screen

Cues:

All items in the virtual database (including directories and browsing agents) have attribute-value pairs called *cues*. A few cues, such as 'owner-name', 'date', and 'keywords', are automatically supplied by the system, but users may add their own personal cues with text, image, or audio values. Cues allow users to create their own views of the objects in the index and are useful for local browsing and searching.

Transducers:

As objects are added to the virtual database, one or more transducers are invoked automatically to extract information from them to use as cue values. Different transducers are applied to objects of different types. Typically, one transducer is used to extract keywords from all text-based objects and one to create a thumbnail version of all image objects. The system can be configured to apply further transducers to all objects of a given type, to use different transducers for different directories, and to allow the user to invoke further transducers for specific objects.

Most cue values are created using public-domain software as transducers. Keywords, for example, are extracted using Essence software [5], and specialised image cues can be provided in selected directories by transducers that extract image features such as intensity histograms or perform image transformations for edge detection or other purposes.

References:

Although references normally point to remote objects, they may also point to objects stored within the system, to other directories in the local virtual database, to remote virtual databases, or to other types of remote object stores. More than one reference may point to the same target, allowing users to access the target via multiple paths using different cues to indicate different reasons for interest. This means the index may be a network rather than a simple hierarchy.

References and cues are relatively small, allowing the overall system to run on machines of modest capacity while providing access to many large objects.

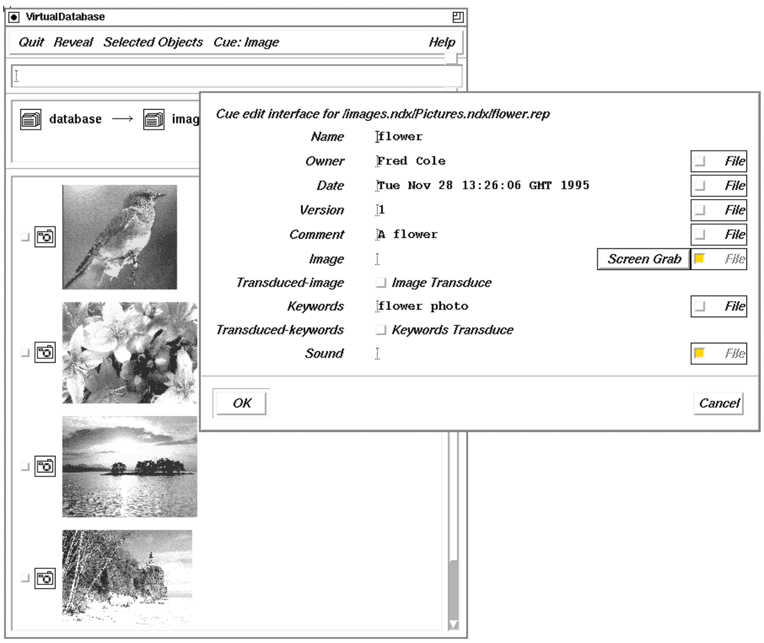
Search Objects:

The original search objects provided simple searching on the cues of objects in the virtual database. Objects satisfying the conditions of a search were presented to the user in exactly the same form as a normal directory (a virtual directory within the virtual database). Searches could be applied sequentially with different criteria for different types of objects. Thus a user could set up a search that would find all directories with 'geology' and 'Devonian' as keywords and then select only the images from those directories with thumbnail cues that satisfied an image matching query.

Search objects worked well within the local virtual database, and indeed within linked remote virtual databases, but operated only on the cues of known objects. The browsing agents that are the main topic of this paper have extended the search model to cover external and composite objects.

As an example combining all these features, a user could add a number of images of geological features to the index. As each reference is installed the system automatically creates a thumbnail version of the image as a cue. The user might also choose to attach two further cues to each image: a small area cut from the original to illustrate a particular feature, and a textual explanation of that feature. Relevant images could be found later by browsing through these cues or by using search objects. If browsing agents had been available they could have been used to scan parts of the World Wide Web to find new images to install.

Figure 1 shows a typical view of the virtual database, in which the main window is partly obscured by a window displaying the remote target of an image reference. The main window contains a scrollable display of items from the current directory. The symbol on the left indicates the type of the item (a camera for a reference to an image, a book for a search object or browsing agent, and a stack of objects for a directory). The user may choose which cues are displayed to the right of these. The other cues and the objects themselves are accessible by clicking and/or selecting from menus. Information on parent directories is shown above the main window. Figure 2 shows a similar view after the user has chosen to display image cues in the main window and has called up a further window for editing the cues of an object.



Cue edit interface for /images.ndx/Pictures.ndx/flower.rep

Name	flower
Owner	Fred Cole
Date	Tue Nov 28 13:26:06 GMT 1995
Version	1
Comment	A flower
Image	
Transduced-image	Image Transduce
Keywords	flower photo
Transduced-keywords	Keywords Transduce
Sound	

Fig. 2. User Interface Showing Image Cues and Cue Editing

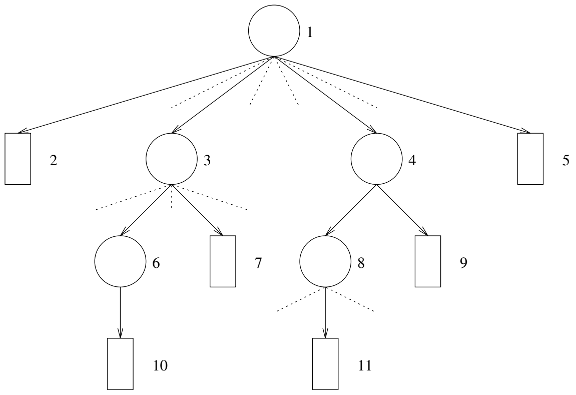


Fig. 3. Sample Network for Browsing

3 Automated Browsing

Figure 3 shows a small network of multimedia objects that a user might wish to browse to see if it contains objects of interest to be added to the virtual database. It will be used in this Section to introduce a model for browsing and in Section 3 for browsing agent examples. The numbers in the Figure represent object references, the circles represent complex objects, and the rectangles represent simple, or leaf, objects. To make the example more concrete we can think of the complex objects as HTML documents or directories in a virtual database, the arrows as references to other objects, and the leaf objects as text or image files. A tree is used here rather than a network to make the explanations easier to follow, but the more general case is dealt with in Section 5.

The following is a simplified model of the typical stages that a user might go through manually when browsing this network to see what 'interesting' objects it contains:

1. Choose some initial object(s) from which to start the search, call this the *to-be-inspected list*, and select one item from this list (object 1 in this example).
2. For any selected object
   (a) Inspect it to see if it is interesting.
   (b) If it is, put it in the results list.
3. For a selected object that may contain references
   (a) Inspect it to see if it contains references to further objects that might be interesting.
   (b) If it does, put them in the to-be-inspected list (references to objects 2, 3, 4, 5 in this example).
4. While the to-be-inspected list is not empty, take one object from it and repeat stages 2 and 3 for that object.

A browsing agent automates this process by providing a framework that follows the stages listed above. The user tailors the framework by providing suitable starting points and queries and by indicating whether the results of one query should be passed to another. When activated, an agent traverses the network of objects, selecting some paths and ignoring others depending on the results of the queries. An agent normally runs without intervention, but a user can control its execution or inspect its current state if desired. It may be saved and activated repeatedly to see if new interesting objects have appeared and indeed to check whether the previous objects still exist.

An agent does not understand or execute queries itself; it just passes them to an external process to execute and either collects the results or passes them to another query. Each query is a process that takes three inputs: a reference to an object, the query to be applied to that object, and an optional upper limit on the number of references to be returned. A query always returns a single (possibly empty) list of object references. Most queries are implemented by existing pieces of software surrounded by simple wrappers to adapt them to this model. Although almost anything can be used as a query, it is convenient to classify them as follows.

Selection queries:
inspect a single object and return its reference if it is of interest. These are used in stage 2(a).

Navigation queries:
inspect a single object containing references to further objects and return the references of interest. These are used in stage 3(a).

Structure queries:
inspect a composite object and return its reference if it is of interest *because of certain properties of its components.* These can be used in stage 2(a) instead of selection queries.

Some examples of selection and navigation queries for different types of objects are given below to illustrate the possibilities. Structure queries are more complex and their description is postponed to Section 4.

Text file queries:
– A wide variety of existing text analysis processes can be used as selection queries for text files, ranging from simple keyword matching programs to sophisticated natural language processing applications.

Image file queries:
– Few selection query mechanisms are publicly available for non-text monomedia objects, so we implemented a simple example selection query for image files. The query compares a target image with a 'search image' supplied by the user and selects the target if there is a sufficient match between the two. (The search and target images are reduced to a standard size and a limited range of colours and then compared pixel by pixel.)

Virtual database directory queries:
– A navigation query for a directory in a virtual database filters the components of the directory according to Boolean combinations of simple queries on the values of their cues (as for the original search objects).

Search engine queries:
– Navigation queries can be sent to a number of popular Web search engines. For these queries an upper limit is specified for the number of references to be returned and the wrapper for the query returns only the appropriate number.

HTML object queries:
– A navigation query for an HTML document might be a list of keywords such as `+"multimedia" -"index"`. If the document contains the word 'multimedia' but not the word 'index', then the query returns all the references (URLs) contained in the document. (A similar selection query would return just the reference of the document itself.)

Individually, such queries represent common searching and selecting actions. The power of the browsing agent comes from the way it can combine and automate the execution of such queries. Section 3 below shows how the framework does this.

4 The Top Level of the Browsing Agent

This Section describes the main features of the browsing agent by means of five simple examples. A user specifies a browsing agent through a graphical interface,

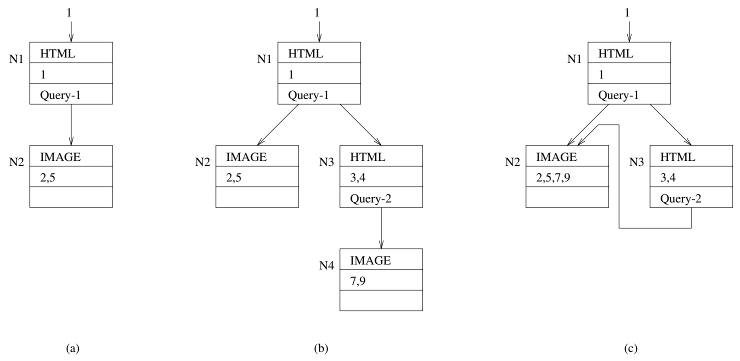


Fig. 4. Simple Browsing Agents

and the same interface is used to control its execution and to retrieve the results. The examples are described with diagrams representing simplified versions of the interface.

Figure 3 shows a small network of objects to be searched by the example agents. Readers should assume that there are many more objects in the network, but the ones shown are the only ones that satisfy the queries used.

4.1 Browsing Nodes

The user creates a browsing agent by defining some *browsing nodes* and the connections between them. Figure 4 shows three browsing agents, each made from several nodes (labelled N1, N2, etc.). The nodes have three components. The top component shows the type of object that the node deals with. This is chosen by the user from a set of types that the agent software knows about (only 'HTML' and 'IMAGE' are used in these examples). The middle component shows the set of references to be queried and the bottom component shows the (optional) query.

The user's tasks in creating a browsing node are:

- Choose an object type that is of interest, or a type that might lead to one of interest.
- Specify a query (applicable to that type) that will inspect each object the node encounters to check whether it is of interest, and/or to extract from it references to other objects that might be interesting.
- Specify the nodes to which references found by the query should be sent for collection or further processing.

The examples below show how the user can build powerful queries from the browsing nodes shown in Figure 4.

Example 1:

The first example just checks an HTML document for certain keywords and, if it finds them, returns any (direct) references to images found in the document. This is so simple it scarcely merits the name of browsing, but it is a good starting point for building up to a useful example.

Figure 4(a) shows how this can be implemented using two browsing nodes, N1 and N2. N1 represents the starting point for browsing. It is for HTML objects, and we shall assume that 'Query-1' is a navigation query that inspects the target document for the keywords `"animal OR bird"` and, if it finds either of them, returns all the object references (URLs) in the document.

The reference to the initial object is input to N1 to start the evaluation. N1 applies Query-1 to each incoming object. In this case the result of applying the query to object 1 is the set of references 2,3,4,5. The connector indicates that the results are to be sent to N2 but, as the type of N2 is 'IMAGE', only the references 2,5 are sent. N2 has no query and does not forward any objects so it simply collects the results.

Example 2:

As well as finding the images referenced directly by a single HTML document, the second example also finds images referenced indirectly via a second HTML document (provided both contain one of the keywords 'animal' or 'bird'). This example shows how the results of a query may be treated differently according to the types of the objects found.

Figure 4(b) shows how nodes may be set up to do this. N1 is as before except that it now has two output connectors. As well as forwarding the IMAGE objects {2,5} to N2, it also forwards the HTML objects {3,4} to N3. N3 then applies Query-2 (which in this example we assume is the same as Query-1) to those HTML objects, and forwards any results of type IMAGE to N4. The results of this query are thus collected in N2 and N4.

Example 3:

This example achieves exactly the same result as the previous one, but it is included to show how the results of more than one query may be directed to a single node.

N4 was not really needed in Example 2. Figure 4(c) shows how the results from both N1 and N3 may be collected directly in N2, avoiding the inconvenience of having the results split between N2 and N4.

Example 4:

The previous examples only looked through one or two levels of HTML documents to find likely images. This example extends the search through any number of HTML documents (provided they all contain one of the relevant keywords).

Figure 5(a) shows how the HTML objects found by N1 can be returned to N1 for recursive application of Query-1. This allows N1 to collect a whole network of HTML objects that satisfy the query. (Section 5 explains how possible cycles in a network of references are dealt with.) At each level, objects of type IMAGE that are found are collected in N2.

Example 5:

One of the implications of Example 4 is that the same query is used to select references to both HTML objects and IMAGE objects. This may not be what the user wants. Example 5 shows how different criteria can be used for following links to further HTML documents and for selecting image references from a particular HTML document.

Figure 5(b) shows how N1 can be used without a query to forward every object reference to more than one node. Query-1 is then used in N3 to recursively select HTML references and Query-2 is used in N2 to select image references. In this case Query-1 might be less restrictive than Query-2, perhaps with a wider range of keywords such as `"animal OR bird OR zoo OR zoology OR biology ...."` while Query-2 is the original criterion for selecting IMAGE objects from an HTML document containing 'animal' or 'bird'.

Although Examples 4 and 5 only contain a handful of nodes, they represent open-ended queries that could potentially range widely over the World Wide Web looking for images connected with animals or birds. Implementation issues concerned with controlling execution and limiting the number of references collected are dealt with in Section 5.

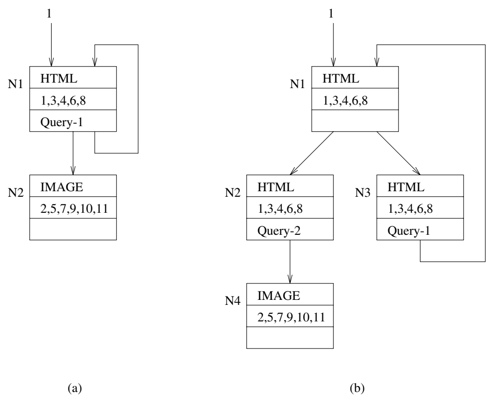


Fig. 5. More Browsing Agents

5 Structure Queries

All the browsing nodes shown so far have used selection or navigation queries. This Section describes 'structure queries'. These are similar to selection queries in that they accept or reject a single object. However, they are designed for use with composite objects, and they accept or reject the object according to properties of the the object's components. Suppose, for example, that a book is a composite object made up of separate chapter objects, each of which is made up of paragraph and image objects. To select a book if and only if it contains images, we need to express a condition that involves inspecting the book's components (and their components) yet returns the reference to the book itself, not references to its image components. Some composite objects already have selection query mechanisms that provide this facility. Structure queries supply the facility for object types that do not already have it.

We must be careful to distinguish a structure query from a browsing agent, because at first sight they seem very similar. Each is an organised collection of nodes containing simple selection or navigation queries. Neither structure queries nor browsing agents inspect individual objects; they organise simple queries to do that. Here the similarity ends. The two crucial differences are:

– browsing nodes may be connected to form a general network, but structure query nodes must be organised as a strict tree (a network would be meaningless);
– the flow of information in a browsing agent is all in one direction. Browsing nodes pass references to following nodes for further processing. Structure query nodes, in contrast, pass references to subordinate nodes in order to receive back TRUE or FALSE results indicating whether the conditions represented by the subordinates are satisfied. This two-way flow of information is explained in the examples below.

Structure queries are really intended for data that is less clearly structured than the book example given above, but for simplicity we shall continue to use that example. The subsections below introduce a few example objects and queries and show how structure queries may be built up from embedded simple queries.

5.1 Example Objects and Queries

Figure 6 represents three Book objects, 1, 2, and 3. Objects are labelled 'B' for Books, 'C' for Chapters, 'P' for Paragraphs, and 'I' for Images. Some of

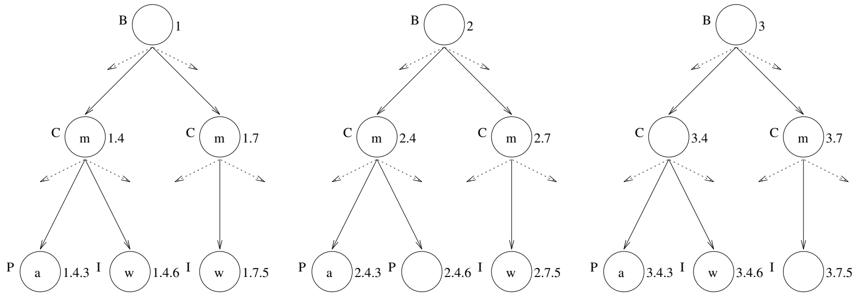


Fig. 6. Example Book Objects

the component objects have properties that are used by the queries. These are represented by the letters within the circles.

In order to build up our structure query, we need to define the following embedded selection and navigation queries.

Selection queries:

- S_a for Paragraphs. Selects a Paragraph if it contains the keyword "animal" (labelled "a" in Figure 6). So, for example, $S_a(2.4.6)$ returns $\{\}$, and $S_a(1.4.3)$ returns $\{1.4.3\}$.
- S_w for Images. Selects an Image if it is wider than 4 inches (labelled "w").

Navigation queries:

- N_c for Books. Returns references to all component Chapters. Thus N_c returns $\{1.4, 1.7\}$.
- N_m for Chapters. Tests whether the Chapter has been modified (labelled "m") and if so, returns its components. Thus $N_m(1.4)$ returns $\{1.4.3, 1.4.6\}$, but $N_m(3.4)$ returns $\{\}$.

5.2 Structure Query Examples

An additional graphical interface is supplied to help a user create structure queries as a tree of query nodes. As before, the queries are explained using diagrams based on that interface, and the examples build up from trivial to fairly complex.

Figure 7 shows five structure queries. For each example we shall assume that the objects 1, 2 and 3 from Figure 6 are to be tested. Remember that a structure query can only be used in a browsing agent in place of a selection query. Thus when the browsing node needs to test a Book for acceptability, it passes the reference of the Book (1, 2, or 3) to the structure query. The structure query evaluates to TRUE or FALSE and returns the reference or null accordingly.

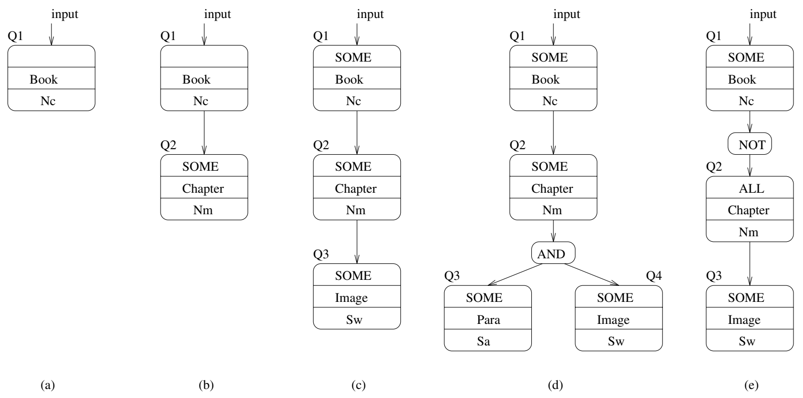


Fig. 7. Structure Queries

Example (a):
The first example simply selects Books that contain at least one Chapter. Figure 7 (a) shows how this can be done with a single query node. The query is applicable to Books and the embedded query is the navigation query N_c. When N_c is applied to object 1, it returns $\{1.4, 1.7\}$. Any non-null set returned is interpreted as success, so query node Q1 evaluates to TRUE and passes the reference to the Book back to the browsing node – object 1 has passed the test. Objects 2 and 3 also pass the test. (Notice that it is possible to use N_c as a test of acceptability, even though it is actually a navigation query rather than a selection query.)

Example (b):
The next example selects Books that contain at least one modified Chapter. Figure 7 (b) shows how this can be achieved using two nodes. Evaluation of Q1 is as before, but Q1 now has a subordinate query node Q2. So after N_c is applied to object 1, the result $\{1.4, 1.7\}$ is sent to Q2. The 'SOME' in node Q2 indicates that at least one of the input set must satisfy the embedded query N_m, otherwise the test fails. In fact both 1.4 and 1.7 satisfy the query, so Q2 returns TRUE to Q1, which in turn evaluates to TRUE. All three objects 1, 2 and 3 satisfy this condition.

Example (c):
This example selects Books with at least one modified Chapter that contains a wide Image. Figure 7 (c) shows how this is done using S_w as a third embedded query. Only objects 1 and 2 satisfy this condition.

Example (d):
Boolean functions 'AND' and 'OR' can be used to combine the results of embedded queries as they flow back up the tree. The example shown in Figure 7 (d) selects Books with at least one modified Chapter that contains BOTH at least one Paragraph containing the keyword "animal" AND at least one wide Image. Only object 1 satisfies this condition.

Example (e):
An alternative to 'SOME' is 'ALL', which requires all the objects in the input list to satisfy the embedded query for the node to return TRUE. There is also a 'NOT' function. The final example, shown in Figure 7 (e), selects Books where not all the the Chapters are modified Chapters containing wide Images. Objects 2 and 3 satisfy this condition.

These examples illustrate the main facilities available for creating structure queries from embedded selection and navigation queries. Users build structure queries from embedded queries in much the same way that they build browsing agents. However, remember the two important differences. A structure query is a tree of nodes and Boolean functions – not a network. During evaluation sets of references are initially propagated down the tree and then the result is propagated back up to give a single TRUE or FALSE answer that indicates whether the composite object passed the test.

This method makes it simple to propagate the answer back up the tree but difficult to create queries that examine components nested at an arbitrary depth.

The current implementation allows a partial solution to this problem by means of a special 'PATH' node that may be applied recursively to any depth. As a final example to illustrate the use of PATH, consider a more complex definition of a Book where Chapters may contain Sections nested to any depth (to represent sub-sections, sub-sub-sections, and so on) and only the lowest level Sections contain Paragraphs and Images. Examples (c), (d), and (e) all need to examine Images, but have no means of finding them if they are nested within an arbitrary number of Sections. This particular example could be solved in all three cases by inserting a new node immediately below the node that applies the embedded query N_m to Chapters. The new node would be a PATH node for Sections containing an embedded navigation query that simply selected all the components of the Section.

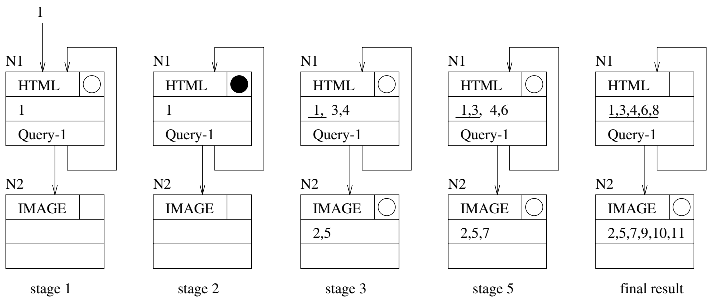


Fig. 8. Stages in the Evaluation of a Browsing Agent

6 Execution of a Browsing Agent

The examples of browsing agents given in Section 3 ignored details of implementation. This Section fills in some of those details. The main issues are how to prevent the browser from looping indefinitely when it encounters a circular path in the network, and how the user can control the number of objects found.

Agents normally run without user intervention, but there is a manual mode of operation which allows the user to control every step of evaluation during the creation and testing of an agent. The manual mode is described first because it provides a natural way to explain the details.

6.1 Manual Mode of Operation

The simple browsing agent used as an example here is the one introduced as Example 4 in Section 3.1. This is the agent that searches recursively through HTML documents containing given keywords to find all the images they reference. Figure 8 shows five stages in the evaluation of this browsing agent, assuming it is applied to the original network of Figure 3.

Before evaluation starts, the user has created the browsing agent with its two nodes, N1 and N2, and initialised N1 with object 1.

Stage 1: before evaluation starts
The button in the top right-hand corner of N1 indicates that it contains one or more object references to be processed (the initial object, 1, at this stage), whereas N2 does not yet contain any references so has no button.

Stage 2: the user presses the button in N1
The next available reference in N1 is selected for processing, the button changes colour to indicate that node N1 is busy, and Query-1 is applied to object 1. The result of this stage is to send IMAGE references 2 and 5 to N2, and HTML references 3 and 4 back to N1.

Stage 3: the initial query evaluation in N1 has finished
Reference 1 in N1 is marked as processed, but the button remains because there are new unprocessed references. A button also appears in N2 indicating references for inspection only (because the node has no query or output connector).

Stage 4: (not shown) the user presses the button in N1 again
The processing of reference 3 begins and changes similar to Stage 2 occur.

Stage 5: reference 3 has been processed
More references have been added to N1 and N2, and reference 3 is marked as processed.

Final result: all processing has finished
The final stage shown assumes the user continued to press the button in N1 until all references had been processed.

Once processing has finished (or as soon as the user chooses to stop) the results can be inspected and installed in the virtual database if desired. The browsing agent can also be stored and activated again later to capture new or updated information. Before it is restarted, the results in all the nodes are deleted, and the agent is restored to its original state. Initialising references are stored with the agent so that they do not have to be entered each time, but

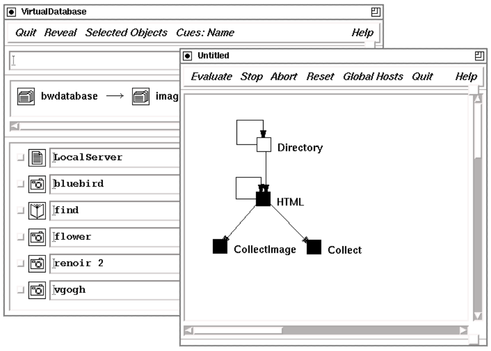
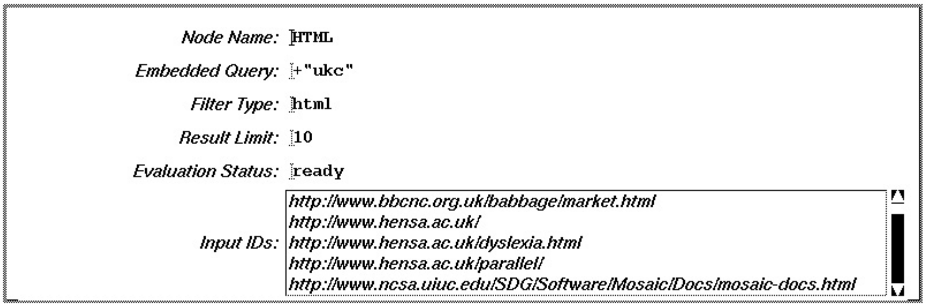


Fig. 9. A Browsing Agent Screen

the user could provide a different starting point to enable the same search to be made at (say) a different remote site. Initialising references may be provided for any node.

Note that references in nodes were marked when they had been processed, not deleted. One reason for this is that the user may be interested in the results at intermediate nodes as well as result nodes. The other reason does not show up in this example, but is crucial. Paths in a general network may be circular, leading the browsing agent back to objects already processed. To prevent processing the same object more than once, references entering a node are treated as a set rather than a list. No reference is added to the set twice.



Node Name:	HTML
Embedded Query:	+"ukc"
Filter Type:	html
Result Limit:	10
Evaluation Status:	ready
Input IDs:	http://www.bbcnc.org.uk/babbage/market.html http://www.hensa.ac.uk/ http://www.hensa.ac.uk/dyslexia.html http://www.hensa.ac.uk/parallel/ http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/mosaic-docs.html

Fig. 10. A Browsing Agent Results

Figure 9 shows a screen representation of an active agent. The nodes do not directly display the information given in the diagrams in this Section (as the query and the set of references are potentially large), but Figure 10 shows how the results are displayed when a user interrogates a node for this information during evaluation. Names, such as ‘CollectImage’ and ‘HTML’, may be specified by the user as an aid to remembering the purpose of a particular node. This example searches through successive levels of virtual database directories for references to HTML documents, then searches the HTML documents, recursively collecting the HTML documents containing a particular keyword and the images from those HTML documents. The nodes change colour to show their state – `ready`, `active` or `finished`.

6.2 Automatic Mode of Operation

A user will probably make some use of manual operation during the creation and initial testing of a browsing agent. Once the agent has been tested and proved useful, the user will normally restart it in automatic mode then attend to other work until the results are ready, or even run it overnight. This will save effort and the time wasted while waiting for each object to be retrieved for inspection, one of the main expenses incurred while browsing a widely distributed network.

During automatic operation the user controls the agent as a whole, rather than individual nodes. Commands are available to start, stop, restart and abort evaluation. To provide some control over the automatic evaluation, the user may specify in advance the maximum number of references that a node will accept. In the example shown in Figure 8, the user might wish to halt evaluation when 50 HTML documents have been found and accumulated in node N1. The limit is used to stop further evaluation at a node and may also be used to inform a query about the number of references to return.

The same graphical interface is used for both manual and automatic modes, so facilities such as the interrogation of the current set of references at a node are available in either mode. In the current implementation, a node spawns a separate process to evaluate its query on each object, but processes only one object at a time. Any number of nodes can be active at the same time. There is a timeout on evaluation of any single query.

7 The Browsing Agent and Related Work

This section provides a brief summary of the strengths and weaknesses of the browsing agent and compares it with other work directed towards the problems of finding information on the Web and keeping personalised indexes.

Although the current implementation is experimental, the virtual database provides an easy-to-manage personalised hierarchical index to local and remote multimedia objects. Browsing agents add facilities for discovering new information and ensuring that the local view is up-to-date. Implementation details are hidden from the user and existing software tools can be plugged in (with minimal wrappers) to act as transducers or queries. In all these ways we believe the system has achieved its original goal of harnessing existing searching techniques to provide personalised multimedia indexing.

As in every experimental system, however, there are areas where our work has failed to address important problems. Some of these are a consequence of the relative lack of implementation effort available, but others are unsolved problems. At present the system recognises a limited number of object types (based on a subset of Mime content-types and subtypes [6]) and uses Essence software [7] for much of its type recognition. More work is need to cope with an extended range of types in a reliable and clean manner. The structure query model also needs further work. The model introduced in Section 4 is only a partial solution to the difficult problem of querying composite objects of unknown structure. The user interface, too, has deficiences. It makes poor use of screen space, does not always present a uniform interface for query specification, and requires users to draw low-level flowcharts to express relationships between queries. Defining higher level constructs to replace the flowchart model is an interesting area for further work.

The growth of the World Wide Web has underlined the need for systems to help users index and manage large numbers of distributed objects and fuelled the development of many large-scale resource discovery systems. These include the early Archie [8] and Gopher [9] systems, Harvest's customised 'gatherers' [10] [11] for collecting and organising information, and search engines such as Lycos [12] and AltaVista [13] for full text indexing and searching. Search engines often return very large numbers of results. Muscat's [14] probabilistic search technology and AltaVista's LiveTopics are recent developments designed to minimise this problem by improving relevance rankings and helping users to refine their queries.

Work on query models has taken place in many different fields. ITASCA [15] is an example of an object database system that uses 'paths' as part of its associative query-based access mechanism. However, the paths are part of the schema rather than a means of querying the structure. GraphLog [16] is a visual structure query language built on top of a front-end to the Hypertext Abstract Machine (HAM) [17]. GraphLog is based on pattern matching in graph structures. Queries are constructed by drawing graph patterns which the system then looks for in the hypertext network. A different type of structure query is provided by HyQ [18] the query language for HyTime [19] [20]. HyQ can select all the elements in a HyTime document that satisfy given conditions relating to their content or position in the hierarchical structure. Other work on query models, including MINERVA's 'query mediators' [21], has been reported in the digital libraries field.

Our system does not attempt to compete with any of the above. Indeed, its goal is to take over where the resource discovery systems finish, helping users to organise and update the results. It uses an open-ended set of simple existing query mechanisms and is particularly useful in intranets where users can exploit their local knowledge.

Web browsers provide basic features for remembering and browsing useful references (for example, Mosaic's hotlist [22] and Netscape Navigator's hierarchical bookmarks [23]). The virtual database and browsing agent should be compared with other tools designed to extend these basic features. Warmlist [24] is a tool with some similar aims. A Warmlist can be indexed using Glimpse [25] and searched for keyword matches, but it only provides searching on known local objects. Work has also been reported on the use of non-textual cues for browsing [26], but the nearest approach to the functionality of the browsing agent that we are aware of is the client-based searches implemented in the Fish Search [27]. To execute a fish search, the user specifies an initial HTML document and a set of keywords. Links, starting from this initial document, are then followed for as long as some of the keywords are found in (or near) the retrieved documents.

Many systems for resource discovery use agents or robots (also known as worms and spiders) [28] [29] [30] [31] to trawl the Web for references and other indexing or retrieval information. The browsing agent behaves to some extent like a robot or simple agent [32]. The main difference is that a robot is usually intended to retrieve and inspect every object it can find, whereas the browsing agent is programmed to be selective about where it goes and what it collects.

8 Summary

The virtual database and browsing agent are aids for finding and organising useful information. They automate local and remote searching in a way that mimics typical browsing sessions, but the user specifies beforehand the choices the browser should make at each object it encounters.

Although there are limitations in the current design and implementation, the browsing agent allows users to deal with many different object types and to include many different 'queries' to operate on these types of objects. Queries extend transparently across local and remote sites and users can control the execution or leave the browsing agent to execute powerful and wide-ranging browsing sessions automatically.

On the basis of our work so far, we believe that the browsing agent provides a good foundation for tackling the increasingly challenging problems users face in extracting useful information from the enormous quantities now available.

Acknowledgements. We gratefully acknowledge the help of BT and the EPSRC, who have jointly funded this project and provided long-term support for research in this area at Kent.

References

1. *Information Technology – Coded Representation of Multimedia and Hypermedia Information (MHEG)*, ISO/IEC DIS 13522, 1996.
2. Dave Raggett, HTML 3.2 *Reference Specification*, W3C Recommendation, available via http://www.w3.org/pub/TR/, January 1997.
3. Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret, 'The World-Wide Web', *Communications of the ACM*, 37 (8), 76–82 (1994).
4. David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole, Jr., 'Semantic File Systems', in *Proceedings ACM SigOps 91*, 1991.
5. Darren R. Hardy and Michael F. Schwartz, 'Essence: A Resource Discovery System Based on Semantic File Indexing', in *USENIX Technical Conference Proceedings*, USENIX, San Diego, CA, Winter 1993, pp. 361–373.
6. Borenstein N. and N. Freed, *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, RFC 1521, September 1993.
7. Peter Deutsch, 'Resource Discovery in an Internet Environment – the Archie Approach', *Electronic networking*, 2 (1), 45 (1992).
8. Paul Lindner, *Internet Gopher User's Guide*, University of Minnesota, 1993.
9. C. Mic Bowman, Peter B. Danzig, Udi Manber and Michael E. Schwartz, 'The Harvest Information Discovery and Access System', *Computer Networks and ISDN Systems*, 28, 119–125 (1995).
10. *Harvest*, home page at http://harvest.transarc.com/
11. *Lycos*, home page at http://www.lycos.com/
12. *AltaVista*, home page at: http://www.altavista.digital.com/
13. *Muscat*, home page at http://athos.muscat.co.uk/ccdp/home.html

14. *ITASCA Distributed ODMS*, Technical Summary for Release 2.1, ITASCA Systems Inc., 1992.
15. Consens, M. and Mendelzon, A., 'Expressing Structural Hypertext Queries in GraphLog', in *ACM Hypertext'89 Conference*, 1989.
16. Campbell, Brad and Goodman, Joseph M., 'HAM: A General Purpose Hypertext Abstract Machine', *Communications of the ACM* (1988).
17. W. Eliot Kimber, *HyTime and SGML: Understanding the Hytime HyQ Query Language*, IBM, August 1993, version 1.1.
18. *Information Technology – Hypermedia/Time-based Structuring Language (HyTime)*, ISO/IEC 10744, 1992.
19. Stephen J. DeRose and David G. Durand, *Making Hypermedia Work: A User's Guide to HyTime*, Kluwer Academic Publishers, 1994.
20. Richard M. Tong and David H. Holzman, 'Knowledge-Based Access to Heterogeneous Information Sources', in *Digital Libraries '94: Proceedings of the First Annual Conference on the Theory and Practice of Digital Libraries*, Texas A & M University, College Station, Texas, 1994.
21. *Mosaic*, home page at: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/
22. *Netscape Navigator*, home page at http://home.netscape.com/
23. Paul Klark and Udi Manber, *Developing a Personal Internet Assistant*, HTML version of extended abstract submitted to "ED-MEDIA" conference, available at http://glimpse.cs.arizona.edu:1994/ paul/warmlist/paper.html December 1994.
24. U. Manber and S. Wu, 'Glimpse: A tool to search entire file systems.' in *Proc USENIX Winter Conference*, 23–32, 1994.
25. Daniela Rus and Kirsten Summers, 'Using Non-Textual Cues for Electronic Document Browsing', in *Digital Libraries: Current issues, Digital Libraries Workshop, 1994 (Selected Papers)*, Lecture Notes in Computer Science, ed. Nabil R. Adam, Bharat K. Bhargava, and Yelena Yesha, Springer, 1995.
26. R.D.J. Post and P.M.E. De Bra, 'Information Retrieval in the World-Wide Web: Making Client-based Searching Feasible', in *Proceedings WWW conference, 1994*, Eindhoven University of Technology, NL.
27. Oliver A. McBryan, 'GENVL and WWWW : Tools for Taming the Web', in *Proceedings, First International Conference on the World Wide Web*, ed. O. Nierstrasz, Elsevier Science BV, Geneva, 1994.
28. Martijn Koster, ALIWEB *Archie-Like Indexing in the WEB*, NEXOR Ltd, 1994.
29. Daniela Rus, Robert Gray and David Kotz, 'Transportable information agents', in *International Conference on Autonomous Agents*, February 1997, pp. 28–236.
30. *Software Agents Group*, pointers to information on software agents available via http://lcs.www.media.mit.edu/groups/agents/resources.htm
31. Stan Franklin and Art Graesser, 'Is it an Agent, or just a program?: A Taxonomy for Autonomous Agents', in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.

An Evaluation of Two Metaphors for Electronic News Presentation

C.R. Watters[1], M.A. Shepherd[1], T. Chiasson[2], and L. Manchester[3]

[1] Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada, B3H 1W5
{watters, shepherd}@cs.dal.ca
[2] The Halifax Herald Ltd.
Halifax, Nova Scotia, Canada, B3J 2T2.
theo@herald.ns.ca
[3] Department of Mathematics & Statistics, Dalhousie University
Halifax, Nova Scotia, Canada, B3H 3J5

Abstract. The metaphor for the electronic presentation of news is still evolving and no one is certain what the most effective metaphor will be. This paper presents an evaluation of two current metaphors for the presentation of electronic news; the traditional newspaper broadsheet metaphor and a document metaphor such as one might find on the World Wide Web. The task assigned the subjects was simply, "read the news." The results favored the broadsheet newspaper metaphor as it seemed better suited to complement the process of "reading the news."

1 Introduction

This paper reports the results of a user study to evaluate two metaphors for the presentation of electronic news consisting of text and photos. The study is part of The Electronic News Project [8], [9], [33], [34], a research project on the integration of news from various media delivered over high bandwidth networks in a personalized multimedia presentation. News, within the scope of this project, is information about recent events of general interest, especially as reported by newspapers, magazines, radio, or television. The deployment of electronic news represents a major shift in the infrastructure, logistics, and ethos of the traditional news (newspaper, television, and radio) delivery services. In contrast to the current model of broadcasting news by the delivery of a discrete product (a paper or a newscast), electronic news can be characterized by the narrowcast delivery of interactive electronic items of various media integrated into a single multimedia presentation.

Providing electronic delivery of news requires three components: digital libraries of news data, selection of content, and presentation of that content at the reader's site. In this paper we address only the third of these components, presentation. While there is some concern over the acceptability of the newspaper in electronic versus paper form [30], the electronic form appears to be inevitable. The presentation metaphor is changing [4] or, as Fidler [15] puts it, is in "mediamorphosis", even though the most appropriate metaphor for an electronic news reading has not yet been determined. In general, little is known about the relationship of task to how electronic publications should look, even though the presentational forms for the equivalent paper publications are well understood [17].

In developing metaphors for the presentation of electronic news, one must be careful to define the task for which the metaphor has been designed. In doing so, a distinction must be made between the task the user is performing when accessing newspaper databases or clipping services and when "getting" the news, whether digital or not. When accessing a newspaper database the user is attempting to satisfy an explicit information need and must be able to express this need in terms of a query or a profile description. When reading a newspaper or listening to the evening news, the user is attempting to satisfy a general, non-specific information need and is not expected, nor is likely to be able, to describe the content required *a priori*. Much of the satisfaction in "getting" the news lies in the process [13] while satisfaction with a database search is in the fulfillment of some explicit information need.

Newspaper databases with on-line access are essentially document retrieval systems in which individual news items are treated as discrete units, i.e., news items are treated as documents (for a review see; [6], [27], [30]). Such systems typically provide retrieval in response to a user query and/or personalized clipping services (selective dissemination of information) based on user profiles. Many newspaper databases have search engines associated with them that vary from quite rudimentary, providing simple string searches, to more sophisticated, such as WAIS [23]. The size of such databases can be enormous when one considers the amount of news data, particularly video, that is generated each day.

Many of the news-on-demand applications that integrate newspaper text, photographs, and television video newsclips are byproducts of research on distributed multimedia systems [21], [22], [29]. This is a natural fit when one considers the multimedia nature of the data, the widespread interest in the content, and the potential for very large data sets. These systems are, however, really news database systems in which the user queries the database(s) and appropriate news items of various media types are retrieved and presented to the user. The main focus of these projects is the development of the underlying distributed multimedia technologies rather than the delivery of news itself. As such these projects have focused on the digital library and content selection components of electronic news but have paid little, if any, attention to the presentation or form of such electronic news from the reader's perspective.

When "reading the news" electronically, as opposed to searching newspaper databases, the user is performing a task which is to satisfy an information need that can only be expressed generally as, "What's happening?" The need is to find information to permit the user to participate fully as a citizen in the local, national, and international community [3]. There is generally no explicit information need that can be expressed as a query or even as part of a query-based user profile as can be done for newspaper database systems. An important aspect of news is the notion that we each expect a certain breadth of coverage as well as a certain level of detail, even if we cannot articulate the specific items or amounts precisely [25]. Research has shown that *people* are unable to predict on a day-to-day basis which news items will be read by others, even when they know them well [2]. Stephenson's *ludenic* theory [35] of news reading asserts that "... the process of news reading is intrinsically pleasurable, and that intrinsic pleasure is at the root of a mature, orderly, and highly ritualistic form of news reading as well as a more casual, spontaneous, and unstructured form of news reading." As such, the presentation metaphor must complement this process, i.e., it must make it possible for readers to take pleasure in "reading the news" as opposed to simply permit the readers to retrieve content from the news.

The WWW has had a major impact on electronic news delivery because of the ease with which it makes it possible for newspapers to produce on-line versions as by products of their paper editions and the ease for readers (and softbots) to access these on-line versions. There are now hundreds of WWW accessible news sites and there is now some evidence that a common metaphor for electronic news presentation is also evolving. Most news sites now have front pages that have some columnar format with images integrated with the text and varying type fonts and sizes. Almost every site uses a shallow hierarchy of sections, headlines, and finally individual stories in the typical WWW single document window mode. As the developers begin to experiment with the presentation metaphor it is worthwhile evaluating the metaphors from the reader perspective. Individual item access is still largely based on the single document model while front pages are more newspaper like. This movement back to the newspaper broadsheet format would seem to be inconsistent with the McLuhan [24] observation that, "The objectives of new media have tended, fatally, to be set in terms of the parameters and frames of the older media."

In the remainder of this paper we discuss two metaphors for "presenting" electronic news and the methodology and results of a user evaluation of a newspaper-like metaphor and a WWW-window document metaphor, i.e., what one finds currently for electronic news stories on WWW sites. These results are then discussed within the context of relevant human-computer interaction studies and metaphor evaluation studies.

2 Metaphors for Electronic News Presentation

Because electronic news has been widely accessible for only a few years, it is fair to expect that the presentation metaphor should be still evolving. Two presentation metaphors dominate electronic news systems: document metaphor and broadsheet metaphor. The document metaphor presents the news items as documents in a shallow hierarchy, where users select a news item by reviewing all of the headlines in a section and choosing one for reading in its entirety. We call this the pick-and-read strategy. The broadsheet metaphor provides a series of pages, where each page provides multiple columns, multiple stories with varying sized headlines, and integrated photographs and graphics. Navigation through this "paper" consists of going to a new page by "turning the page," selecting a section and jumping to that section, or possibly searching for the next occurrence of words or phrases. Although most news providers base the presentation on the pick a headline and read the story model, the adoption of some features of the newspaper broadsheet metaphor, particularly the use of columns and the integration of photos with the text on the front page is now common. This incorporation of some features of the newspaper broadsheet can be understood from the point of view of glitz and impact needed for web sites to attract readers. The question that we want to address is the suitability of presentation metaphor, not so much for its potential for initial attention grabbing as for the longer-term satisfaction of the task of "reading the news". Do users prefer the collage effect in the restricted space of a computer screen over the simplicity of reading single items in their entirety? Should on-line news providers continue to provide a pick-and-read strategy past the front page or move to the broadsheet format for all items in the news?

Other electronic news presentation systems also provide a broadsheet view of the news. For example, Apple Computer, Inc., developed one of the first prototype electronic news systems for the EDUCOM 1990 Conference [18]. It downloaded integrated video, graphics, and text into Hypercard. The Newspace project [5] at the MIT Multimedia Lab provides personalized multimedia editions maintaining the newspaper metaphor. Walksoft [32] offers a weekly electronic newspaper with the layout templates at the user's PC, thus only text and photographs need be sent and layout occurs on the fly also with an adopted newspaper look and feel.

Researchers at GMP-IPSI have developed the Individualized Electronic Newspaper (IEN) [17], as part of the larger issue of active publications, i.e., publications that have programs attached to them to allow the publications to act on its environment. The IEN is an individualized publication, composed on demand for the reader and then delivered electronically. This research has experimented with different presentation metaphors implemented in the HyperNeWS systems; a newspaper metaphor and a mailtool metaphor. The initial newspaper metaphor included increased functionality such as access to background material and to databases of classified advertisements. It presented the news items in a newspaper metaphor with multiple columns and stories on a page, but used a different metaphor when presenting different types of material, such as advertisements and background information. They found, however, that having an electronic document created expectations of functionality such as cut and paste, annotation capabilities, previous article/next article, etc., that led them to develop a metaphor based on electronic mailtools that included such functionality.

The WWW was not, of course, designed specifically for newspapers, rather it was designed to handle generic documents, stored as marked up files, consisting of text and graphics. The WWW now permits audio, video, animation and two-way communication as well making it a good medium for the presentation of electronic news. It is quite easy to publish newspaper news on the WWW and the content of most of these newspapers is just a by-product of their ink-on-paper publishing. WWW browsers have traditionally used a single scrolling window, the approximate size and shape of a document page for the presentation of this data. Navigation through a newspaper presented in this metaphor consists of following a shallow hierarchy of links. The top of the hierarchy, the home page, consists of the names of news sections available. Upon selecting a section, a new page with the headlines of news items in that section is displayed. Finally, the user selects a news headline and the contents of that item are displayed. Typically, the news item is displayed in its entirety as a document, i.e., a single column the width of the window, and if the text is too large to fit in the window it is scrolled vertically. Associated photographs are represented by icons or thumbnails with the text of the news item and can be selected to present the full photograph. This maximizes the text presented in its most readable form (least fragmentation) within the fairly restricted area of a computer screen.

As more presentation features become available for browsers there has been considerable effort to mimic the newspaper broadsheet metaphor on the WWW, at least for the front page. Extensions to HTML have made it easier to juxtapose multiple columns and stories on a single page and most web-based newspapers now have a columnar feel on the home pages (i.e., the front pages). Java based functions allow the dynamic layout of items for the reader in the *Krakatoa Chronicle* [20]. This movement towards more newspaper-like formats is happening for a number of reasons, including more impact and higher glitz rating. As the presentation format effects the satisfaction of the reader while "reading" the news, the design of electronic news presentation formats should be based on various human-computer interaction factors and task analysis as well as on the visual appeal to the designers.

Almost all newspapers currently available on the WWW (literally hundreds) present the individual news items using the WWW single document metaphor, as described above. Many of these sites now offer a front page with multiple stories or multiple columns, but revert to the single document metaphor beyond the front page. Many of the experimental electronic news delivery systems, such as Fishwrap [10] available at MIT, have also adopted the single document metaphor.

The results from earlier evaluations of people reading text from computer screens indicate that many factors, such as font type, font size, pixel size, screen size, length of line, number of lines, etc., affect performance (for surveys see for example; [12], [26]). There has been little evaluation, however, of user preferences of various metaphors for the presentation of electronic news where the goal is satisfaction with "reading the news," i.e., satisfaction with the process rather than the results of the search. The results of these previous evaluations are presented in the Discussion Section, below, as a context for discussing the results of this study.

In this study, we investigated whether users had any preference for one metaphor over the other for the task of "reading the news" electronically. If we can establish that users have strong preferences, then we can begin to examine which features are important to *readers* as they address the task of gathering current news. The results of such evaluations can then be used to guide further evolution of electronic news presentation metaphors based on effective task resolution metrics rather than designer preferences.

3 Methodology

Two different metaphors for presenting electronic news were evaluated in this study; a newspaper broadsheet metaphor and a shallow hierarchy pick-and-read window metaphor, which we called the window metaphor to distinguish it from any particular commercial browser. The presentation system of the newspaper metaphor was a vanilla-flavored version of the multimedia news delivery system developed as part of this research project. For the evaluation study, no advertisements or video clips were included. The system did have multiple columns, multiple stories with headlines per page, and photographs integrated with the stories (Figure 1). If a story was too large to fit in the space allocated, the user could click anywhere on the text of the story and blowup the story in a new window to see all of the story. A click on the blownup story would return the screen to its previous state. A pull-right menu of sections and headlines was available at all times on the left of the screen and navigation was driven by selecting a page from the index or by "turning pages" and/or skipping forward or backward to the next or previous section through buttons.

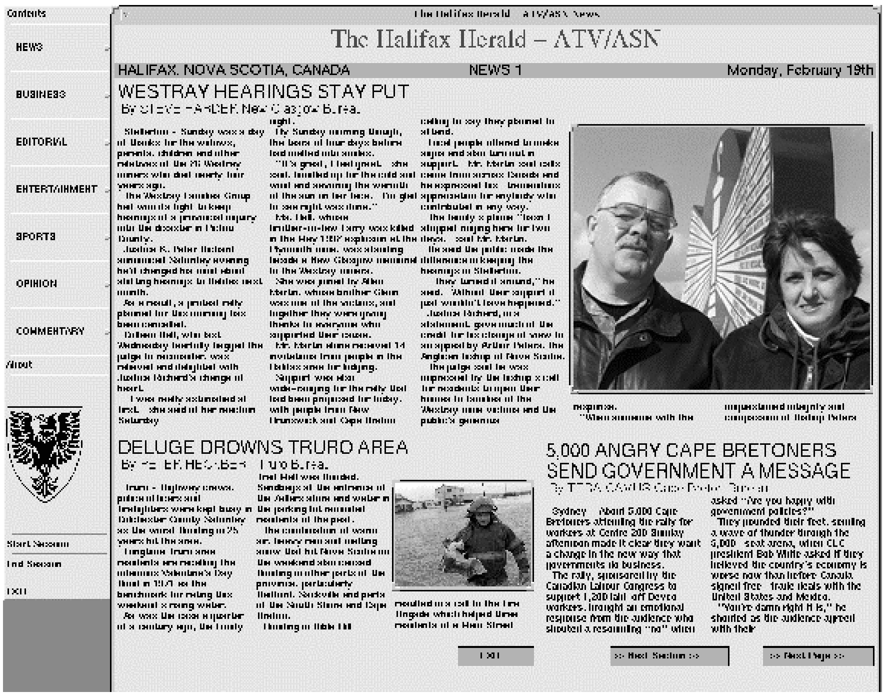
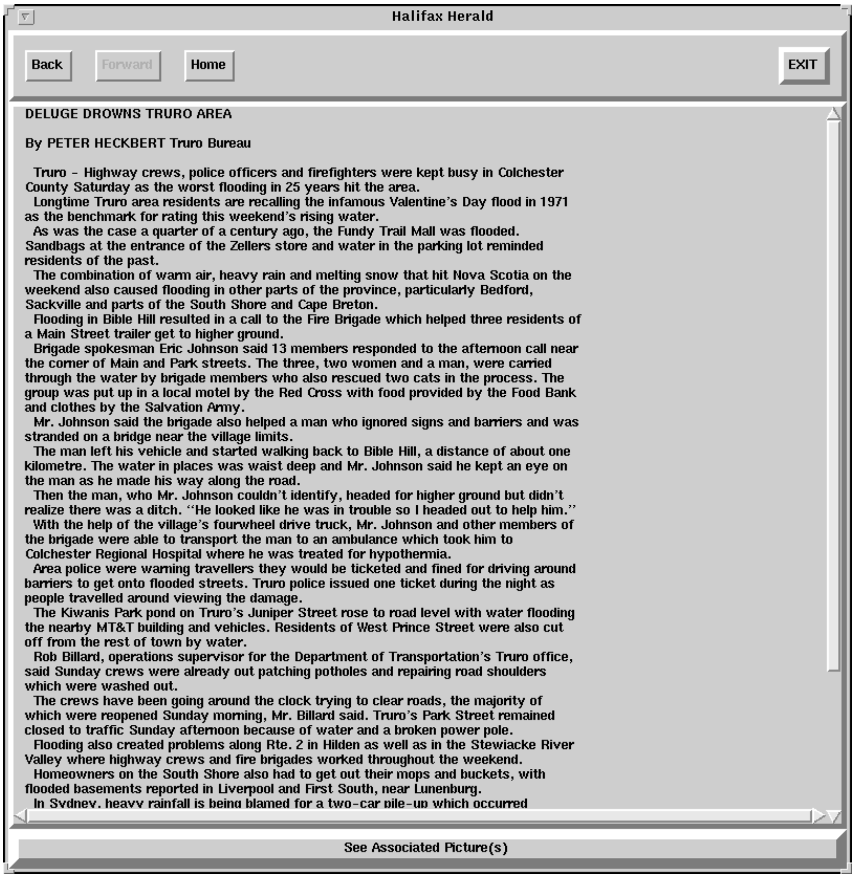


Fig. 1. Newspaper metaphor – Front page

The window document metaphor was presented by a browser with basic features similar to that found in any WWW newspaper application without a broadsheet front page. The news was presented in a hierarchical fashion with a front page listing the sections in that day's newspaper at the top of the hierarchy, second level pages listing the headlines in a selected section, and individual stories at the third level (Figure 2). Each story took up the whole window and was presented in its entirety. Whenever there was a photograph associated with a story, a button was made active at the bottom of the story window. If clicked, this button caused the photograph to be displayed. Navigation was hierarchical and the subject could always jump directly to the front page.

As much as possible we tried to minimize the differences between the systems so that the users could concentrate on the metaphor rather than implementation differences. Both systems were written in Tcl with the Tk toolkit and run on Sun Microsystems workstations under the Solaris operating system. Both systems used the same data files picked up from the ftp site of the Halifax Herald each morning. The allocation of items to sections, the order of the sections, and the choice of headlines were decided by the Herald staff and were the same in both systems. Neither system had colored photos, advertisements, video clips, or two-way interaction. Both systems used the same background color and the default Tcl font type (Adobe Helvetica) and font size (10-point). The font sizes look different in the figures of the two systems because of the resizing of the screen dumps to fit the printed page. The layout of articles and photos in the full broadsheet edition took about an hour of human intervention while the document hierarchy edition was composed fully automatically.



DELUGE DROWNS TRURO AREA

By PETER HECKBERT Truro Bureau

Truro – Highway crews, police officers and firefighters were kept busy in Colchester County Saturday as the worst flooding in 25 years hit the area.
Longtime Truro area residents are recalling the infamous Valentine's Day flood in 1971 as the benchmark for rating this weekend's rising water.
As was the case a quarter of a century ago, the Fundy Trail Mall was flooded. Sandbags at the entrance of the Zellers store and water in the parking lot reminded residents of the past.
The combination of warm air, heavy rain and melting snow that hit Nova Scotia on the weekend also caused flooding in other parts of the province, particularly Bedford, Sackville and parts of the South Shore and Cape Breton.
Flooding in Bible Hill resulted in a call to the Fire Brigade which helped three residents of a Main Street trailer get to higher ground.
Brigade spokesman Eric Johnson said 13 members responded to the afternoon call near the corner of Main and Park streets. The three, two women and a man, were carried through the water by brigade members who also rescued two cats in the process. The group was put up in a local motel by the Red Cross with food provided by the Food Bank and clothes by the Salvation Army.
Mr. Johnson said the brigade also helped a man who ignored signs and barriers and was stranded on a bridge near the village limits.
The man left his vehicle and started walking back to Bible Hill, a distance of about one kilometre. The water in places was waist deep and Mr. Johnson said he kept an eye on the man as he made his way along the road.
Then the man, who Mr. Johnson couldn't identify, headed for higher ground but didn't realize there was a ditch. "He looked like he was in trouble so I headed out to help him."
With the help of the village's fourwheel drive truck, Mr. Johnson and other members of the brigade were able to transport the man to an ambulance which took him to Colchester Regional Hospital where he was treated for hypothermia.
Area police were warning travellers they would be ticketed and fined for driving around barriers to get onto flooded streets. Truro police issued one ticket during the night as people travelled around viewing the damage.
The Kiwanis Park pond on Truro's Juniper Street rose to road level with water flooding the nearby MT&T building and vehicles. Residents of West Prince Street were also cut off from the rest of town by water.
Rob Billard, operations supervisor for the Department of Transportation's Truro office, said Sunday crews were already out patching potholes and repairing road shoulders which were washed out.
The crews have been going around the clock trying to clear roads, the majority of which were reopened Sunday morning, Mr. Billard said. Truro's Park Street remained closed to traffic Sunday afternoon because of water and a broken power pole.
Flooding also created problems along Rte. 2 in Hilden as well as in the Stewiacke River Valley where highway crews and fire brigades worked throughout the weekend.
Homeowners on the South Shore also had to get out their mops and buckets, with flooded basements reported in Liverpool and First South, near Lunenburg.
In Sydney, heavy rainfall is being blamed for a two-car pile-up which occurred

Fig. 2. Window metaphor – Full story

This study limited the electronic news to newspaper text and photographs only. There were no television video clips, audio clips, or advertisements. This limitation was imposed so that a baseline evaluation of metaphors for the electronic news could be established with the intention of adding features such as video in future studies. We also were interested in the presumed preference of thumb nail photos or integrated photos with text stories, even though they take up considerable space.

An advertisement for volunteers to participate in the study was posted to a moderated list-server for Dalhousie University employees and students. Various units at Dalhousie University, such as physical plant, personnel and payroll, and the main campus library were also contacted and volunteers requested. This resulted in a *convenience* sample (as opposed to a true random sample) of volunteers consisting of students, faculty, staff, administrators, and members of the physical plant. A random sample was impossible in this study because of the need for cooperation by the supervisors of the different employee units within the university. As it turned out we had an even distribution by gender and a useful cross-section of people, although primarily from the university community, including Vice-Presidents, faculty, students, and clerical and physical plant staff.

The study was run over a four day period, February 19 through February 22, 1996. News from *The Chronicle-Herald*, metro edition, was downloaded from The Halifax Herald's private ftp site each morning. The metro edition of *The Chronicle-Herald* is the major morning paper in the Halifax metropolitan area. Thus, each day, the subjects for that day had all of the news that appeared in *The Chronicle-Herald* for that day, although the advertising, classified sections, comics, and the editorial cartoon were not included. The same news stories and photographs were accessed by both the window metaphor system and the newspaper metaphor system.

The two systems thus provided the subjects of any given day with exactly the same news stories and photographs, all loaded from the Halifax Herald's ftp site. The same workstations and systems were used throughout all four days of the study.

The questionnaire that the subjects were asked to complete was based on the short version of the Questionnaire for User Interaction Satisfaction™ from the University of Maryland at College Park,[1] and is available from the authors. Prior to seeing either system, the subjects were asked to read and sign the informed consent form, were assigned a number to ensure confidentiality, and were asked to complete the biodata and background information sections of the questionnaire.

After completing these sections of the questionnaire, each subject was assigned randomly to either the window metaphor system or the newspaper metaphor system. We were careful not to let the subjects see the "other" system before the evaluations began. The subjects were given a very brief introduction on how to navigate the system to which they had been assigned (all navigation was point-and-click). The introductions took typically 2 or 3 minutes and we defined the task as "to read the news." The subjects could read the news for as long a period or as short a period as they wished, just as they would a paper version. When each subject was finished reading the news with their first system, they were asked to complete the questionnaire section dealing with how much they liked that particular interface, how easy the system was to use, etc. There were no questions asked to determine retention or understanding of news read. Interestingly, some people took ten to fifteen minutes while others took up to an hour to read the news.

The subjects were given a brief introduction to the other system and then asked to repeat the procedure for the other system, i.e., if they had read the news on the window metaphor system first, they were asked to now read the same news on the newspaper metaphor system and to complete the appropriate section of the questionnaire.

Note that each user read the *same news* presented in both metaphors. In this paper we are concentrating on user preference of presentation metaphor with respect to the specific task of "reading the news". After having read the news in both metaphors and completed the appropriate sections of the questionnaire, the subjects were asked to complete the section of the questionnaire which asked if the subjects had a preference for one metaphor over the other and, if so, what factors influenced their preference.

[1] Licensed to Dr. Watters.

4 Results

The tests were designed to determine if the subjects preferred one metaphor over the other metaphor for the task of reading the news from an electronic news system and, if there was a preference, to determine if this might be influenced by such factors as age, education, job, and/or previous experience with computers. Fisher's Exact Test [1] was used as many of the cell values were too small to use the chi-square test. For each statistical test in the study, the "p-value" or "observed level of significance" is reported. This is a number between 0 and 1 which measures the strength of the evidence against the null hypothesis being tested. The closer this number is to 0, the stronger is the evidence against the null hypothesis.

All 93 participants viewed the news in both metaphors. However, not all of the subjects answered all of the questions in the questionnaire, likely from simply missing a section of the questionnaire.

4.1 Preferred Metaphor

Tables 1 and 2 show the accumulated preference results. In both of these tables, 91 of the subjects stated a preference, including "no preference" and two subjects left the question unanswered. As one can see from Table 1, the vast majority of the subjects preferred the newspaper metaphor over the window metaphor.

Table 1. Metaphor preferences

Preferred Metaphor	Frequency
strongly preferred newspaper	73
slightly preferred newspaper	9
no preference	0
slightly preferred window	4
strongly preferred window	5

Table 2 shows the metaphor preference based on which metaphor the subjects viewed first. The subjects were assigned randomly to two groups: "Newspaper First" or "Window First". As the news was the same, one might suppose that exposure to one system before the other might influence their preference. In Table 2, the categories, "strongly preferred newspaper" and "slightly preferred newspaper" from Table 1 were collapsed into a single category, as were the categories, "strongly preferred window" and "slightly preferred window". Fisher's Exact Test was applied to the data in Table 2 and no significant relationship was found between the order in which a subject viewed the systems and the subject's preferred metaphor (p=0.0838).

Table 2. Effect of which metaphor viewed first on preferred metaphor

	Preferred Metaphor	
Metaphor Viewed First	Newspaper	Window
Newspaper	40	7
Window	42	2

4.2 Evaluation of Individual System Usage

In this section we will examine the results of the questionnaire sections that the users filled in immediately after using each of the two systems. These questions are related to the single system just used and do not contain any comparative or reference questions.

Tables 3 summarizes how subjects felt overall about each system after usage. The data in Table 3 has been condensed into three categories from nine categories in the questionnaire. The first factor we looked for was whether individuals tended to say the same or tended to say opposite things about the systems. So we need to know whether the subjects would generally rate one system bad if they had rated the other good or if subjects tended to give the same evaluation on an individual basis to both of the systems.

Pearson's correlation coefficient was calculated for the full nine categories in the raw data. The Pearson coefficient was 0.244, with a 95% confidence interval of (0.04, .430). This is not very strong evidence of correlation, especially as the 99% confidence interval includes 0, and supports the hypothesis that people did not have a strong tendency to say the same things (or opposite things) about the two metaphors.

Table 3 does suggest that subjects found the newspaper system to be "wonderful" (68) or "okay" (49) more often than they found the window system to be "wonderful" (24) or "okay" (21). Subjects did not, overall, dislike the window system per se as only 17 people thought it was "terrible."

Table 4 summarizes whether people found the two systems easy to use. This data is condensed into three categories from nine categories in the questionnaire. Pearson's correlation coefficient was calculated for the full nine categories in the raw data. This coefficient was 0.242, with a 95% confidence interval of (0.03, .422). This is not very strong evidence of correlation, especially as the 99% confidence interval includes 0, and supports the hypothesis that people did not have a strong tendency to say the same things (or opposite things) about the two metaphors.

Table 3. How subjects felt about the two systems

	Window System		
Newspaper System	terrible	okay	wonderful
terrible	0	0	1
okay	7	10	4
wonderful	10	39	19

Table 4. Whether people found the systems easy to use

	Window System		
Newspaper System	difficult	moderate	easy
difficult	0	0	2
moderate	0	4	11
easy	1	10	61

By far, the largest number of readers said that both systems were easy to use. The next largest number said that one was moderately easy to use and the other was easy to use. Two people said that the newspaper system was difficult to use but that the window system was easy to use. Only one person said that the newspaper system was easy to use but that the window system was difficult. One can conclude that, although people did not have a strong tendency to say the same thing about the two metaphors, both systems were found to be easy or moderately easy to use. Consequently, ease of use was not a factor in preference of metaphor.

4.3 Factors for Preference Choice

The last set of tables and tests looked at individual factors as contributing to a preference of one metaphor over the other: previous computer usage, gender, age, level of education, and job category.

All of the subjects in the study had some computer experience. Fisher's Exact Test of the data in Table 5 indicates that previous computer experience had no significant effect on the choice of a preferred metaphor (p=.4373). In fact, there was *very* little evidence supporting any connection between previous computer experience and choice of metaphor.

Table 5. Effect of computer experience on preferred metaphor

	Preferred Metaphor	
Computer Experience	Newspaper	Window
1-4 computers	57	8
5+ computers	25	1

Fisher's Exact Test of the data in Table 6 indicates that gender had no significant effect on the choice of a preferred metaphor (p=0.0901). The rightmost column indicates that there was an almost even number of male and female subjects.

Table 6. Effect of gender on preferred metaphor

	Preferred Metaphor		
Gender	Newspaper	Window	Total
male	38	7	45
female	44	2	46

Table 7. Effect of age on preferred metaphor

	Preferred Metaphor		
Age Group	Newspaper	Window	Total
20-30	29	2	31
31-40	21	0	21
41-50	26	5	31
51-60	4	2	6

Fisher's Exact Test of the data in Table 7 indicates that age did not have a significant effect on the choice of a preferred metaphor (p=0.0511). However, the data suggests that older people may have a slight tendency to prefer the windows metaphor more often than those in the lower age categories. Of course, a sample which included more people in the oldest category would provide more information about whether or not this was actually true. The rightmost column indicates the totals in each age category.

Fisher's Exact Test of the data in Table 8 indicates that a subject's level of education had no significant effect on the choice of a preferred metaphor (p=0.6712) and that there was no evidence that education level influenced the choice of metaphor.

As an interesting note, there were ten subjects in the 20-30 year age category with some post secondary education and all of these subjects preferred the newspaper metaphor. Because we recognized most of the students, it is possible to state that the majority of these ten subjects were undergraduate computing science students. We expected that these participants would have preferred the window metaphor as they were very familiar with the window metaphor used in browsing the Internet.

Table 8. Effect of education level on preferred metaphor

	Preferred Metaphor		
Education Level	Newspaper	Window	Total
high school diploma	1	1	2
some post secondary	16	2	18
diploma	5	0	5
bachelors degree	28	3	31
professional degree	3	0	3
masters degree	17	2	19
doctorate degree	12	1	13

Fisher's Exact Test of the data in Table 9 indicates that subject's job category had no significant effect on the choice of a preferred metaphor (p=0.2309). In the study we were fortunate to have subjects from a wide range of university job categories and it was interesting that there was little evidence that job category had any influence over their preference for metaphor. The only possible anomaly is the "physical plant" category. It is very difficult to get people in this category to participate in a study during working hours. Because people in this category do not use computers in their work, and they were split evenly in their preference, one wonders what the results might have been if there were more of them in the study.

Table 9. Effect of job category on preferred metaphor

	Preferred Metaphor		
Job Classification	Newspaper	Window	Total
student	29	1	30
physical plant	1	1	2
staff	23	4	27
administration	7	1	8
faculty	16	2	18
other	6	0	6

5 User Comments

In addition to the quantitative responses reported above, users were asked, in the final section of the questionnaire, for written comments on why they preferred one metaphor over the other. In this section we summarize those comments and include a few supporting quotes.

5.1 Comments of Subjects Who Preferred the Window Metaphor

Those subjects who preferred the window metaphor did so because it seemed easier and faster to find individual stories. The subjects liked being able to go directly to a section of interest and see the headlines of all the stories in that section at a single glance and select from that list. Supporting quotes include:

"If I don't have time to browse, it is good for quick reading."

"... like the full access to each section."

"... simpler to use."

5.2 Comments of Subjects Who Preferred the Newspaper Metaphor

Interestingly, those who preferred the newspaper metaphor not only indicated why they preferred that metaphor but also indicated what they did not like about the window metaphor. These comments are summarized below, followed by supporting quotes.

- They liked the columns of the newspaper metaphor and found the text lines of the window metaphor too wide.
- They liked multiple stories on a page so that they could start reading an article and switch to another article without having to back up a level of index. They did not like the back-and-forth necessary to traverse the hierarchy in the window metaphor.
- They liked having the headline and story together because the headlines alone were often misleading.
- They found the newspaper metaphor more stimulating to read and found the window metaphor too much like work.
- They really liked having the pictures displayed with the stories rather than having to take an extra step to display them.
- People did find that in the blowup in the newspaper metaphor it was hard to find the spot where they had been reading in the story.

> *"... column format very much easier to read."*
>
> *"I like to be able to scan."*
>
> *"Newspaper format allowed me to browse and pounce on eye-catching story."*
>
> *"I like to read the headlines and the first few paragraphs of a story and only sometime do I read the whole article."*
>
> *"...prefer to see pictures along side text. Pictures grab your attention to particular story."*
>
> *"Seeing the amount of space devoted to a story and where it was positioned was useful."*
>
> *"[windows] ... headlines often misleading or unclear."*
>
> *"[windows] ... straight list of topic headings irritating and hard to scan."*
>
> *"[windows] ... didn't care for back-and-forth method."*
>
> *"[windows] ... feels flat, bland."*

Suprisingly (or not), some of the users who preferred the newspaper metaphor indicated that the print of the window metaphor was too small, and at the same time some of the users who preferred the window metaphor thought that the font size of the newspaper presentation was too small. However, the same font and size was used for both systems.

6 Discussion

It is quite clear that the subjects in this study preferred the newspaper metaphor to the window metaphor for the specified task of “reading the news.” The fact that people are used to the newspaper format for receiving their news on paper, at least, may be a factor, although all of the subjects in this study had also used computers. Although there was little or no statistical evidence that such characteristics as gender or age influenced the choice of a preferred metaphor, the nine subjects who preferred the window metaphor can be described as having less computer experience, male, and over 40 years of age.

In this section, we will review the research and results on specific features of the newspaper metaphor and reading text from a screen in general. The results of research into reading text from a screen are often contradictory, and this may indicate that the role of a metaphor that supports a given task, such as “reading the news”, overrides general screen reading preferences.

6.1 Support for the Newspaper Metaphor Based on Task

From the subjects comments in the section above, it is clear that the subjects performed the assigned task of “reading the news” by browsing or skimming the material presented. This is consistent with the findings of Dozier and Rice [13] and of Paterson and Tinker [31] in which they describe the task of reading newspaper headlines as one of “skimming”. This “skimming” model fits well with how people read from a computer screen; they scan for an item of interest and then, once such an item is found, read the item in detail [19]. If people are skimming, then the display format is very important and should convey context rather than large single pages or lists of headings for users [11]. In particular, Ohkubo et al. [28] studied user information acquisition performance for two layout methods of headlines, simple listing layout and newspaper layout. This study found little difference in the time taken to locate a headline matching a given category. The study also found that users were better able to recall words appearing in the headlines in the simple listing layout than in the newspaper layout for presentation times of 1.5 to 2.0 seconds. These results suggest that, with respect to skimming or browsing, “... the newspaper layout imposed less cognitive load than the simple listing for this task.” In other words, the simple listing layout of headlines requires more concentration on the part of the reader than does the newspaper layout. In the Ohkubo study, ten of the 12 subjects preferred the newspaper layout.

The newspaper metaphor lends itself to this skimming technique. It provides multiple stories on a page and headlines and stories are together so that the user can skim the first paragraph, and the photographs aid in attracting a reader into a story [16]. The comments of our readers emphasized the enjoyment of the process of reading the news, “pounce on an eye-catching story”, “not like work”, windows was “flat, bland,” and were consistent with Stephenson’s *ludenic* theory of news reading [35]. The emphasis for those who preferred the broadsheet collage was its impact on the process of reading the news rather than the actual content of the daily news: easy to scan, headlines and few paragraphs, pictures, browsing, positioning of items, etc.

6.2 Columns

The newspaper metaphor implies the use of multiple columns of text rather than the single column extending the full width of the window as in the window metaphor. In a survey of empirical studies concerning the readability of text from computer screens, Mills and Weldon [26] found the conclusions reached with respect to the effects of multiple columns to be contradictory. Bouma [7] pointed out that for long lines of text it is difficult for the reader to accurately locate the beginnings of new lines after the long lateral eye movement. Duchnicky and Kolers [14] found that longer lines of text are read more efficiently from computer screens that shorter lines. Mills and Weldon [26] concluded that more research in this area is needed.

Comments by the subjects in this study indicate that they preferred the columns of the newspaper metaphor. This is consistent with the task of "skimming" as it is easier to skim a paragraph of short lines than of long lines as there is not the problem of locating the beginnings of new lines. That none of the readers noticed or complained about the frequent truncation of items in the broadsheet format indicates the importance of skimming and partial reading of items as a reading behavior for this task.

6.3 Scrolling versus Blowup

In the window format the reader was presented with a story in its entirety. If a news item was too large to fit in the window then it could be scrolled vertically. In a review of the literature, Dillon [12] found that there was no performance difference between scrolling and paging but that novices tended to prefer paging while more experienced users preferred scrolling through the text.

The columnar nature of the broadsheet format resulted in many partial items. The presentation software in this study provided a blowup window of the entire story, not dissimilar to paging from the abbreviated story to the full story. Because the blowup tended to reformat the story, subjects found it difficult to pickup the spot from which they had suspended reading in the original format, often in the middle of a sentence. This problem is similar to text splitting across screens which led to reduced reading performance and a lot of "back and forth" between screens to find the place and context [12]. Most of our subjects made infrequent use of the blowup feature, getting most of what they wanted from the first part of the item and were not that interested in reading full items.

7 Summary

Subjects in this study strongly preferred the newspaper metaphor over the WWW-window metaphor for the task of "reading the news". This supports the importance of presentation modes that reflect the task, particularly for recreational or non-work tasks. Satisfaction for the task of "reading the news" involves more satisfaction with the process than would a clipping service or database query. This suggests that the move towards a broadsheet presentation for electronic news is appropriate for most readers and would be appropriate for the entire paper, not just the front page.

There are, however, subjects who did not read the electronic news in this *ludenic* browse-mode. These individuals wanted to see all the headlines for a section at one time so they could select and read only those stories in which they thought they would be interested. This suggests that letting presentation metaphors vary for the user and for the task may be appropriate and that users may want to have control over the presentation metaphor. Fortunately, one of the advantages of electronic news presentation is that this flexibility is certainly feasible.

Acknowledgments. This research has been supported in part by Natural Sciences and Engineering Research Council of Canada. We would like to thank The Halifax Herald Limited permission to use their daily newspapers for this study. In addition, we would like to thank Dwayne McKinnon for his programming of the Window system.

References

1. Agresti, A: Categorical Data Analysis. Wiley, New York (1990)
2. Allen, R.: User Models: Theory, Method, and Practice. International Journal of Man-Machine Studies. 3 (1990) 511–543
3. Asp, K: Mass Media as Molders of Opinion and Suppliers of Information. In: Wilhoit, C., Whitney, C. (eds.): Mass Communication Review Yearbook, Vol. 2. Sage Publications , Beverly Hills (1981) 332–354
4. Ashton, E., Cruickshank, G.: The Newspaper of the Future: A Look Beyond the Front Porch. Proceedings of the 14th National On-line Meeting. (1993) 11–16
5. Bender, W., Lie, H., Orwant, J., Teodosio, L., Abramson, N.: Newspace: Mass Media and Personal Computing. Proceedings of the Summer 1991 USENIX Conference, (1991) 329–349
6. Berman, M.A.: Today's World News – Creating a Desktop News Delivery System. Proceedings of the 14th National On-line Meeting. (1992) 33–38
7. Bouma, I.P.: Visual Reading Processes and the Quality of Text Displays. In: Grandjean, E., Vigliani, E. (eds.): Ergonomic Aspects of Visual Display Terminals. Taylor & Francis, London (1980) 101–114
8. Burkowski, F.J., Watters, C.R., Shepherd, M.A.: Electronic News Delivery. Technical Report PR-94-01. Dept. of Computer Science, University of Waterloo, Canada (1994)
9. Burkowski, F.J., Watters, C.R., Shepherd, M.A.: Delivery of Electronic News: A Broadband Application. Proceedings of CASCON'94. (1994) cd-rom
10. Chesnais, P.R., Mucklo, M.J., Sheena, J.A.: The Fishwrap Personalized News System. Proceedings of the 1995 IEEE Second International Workshop on Community Networking Integrating Multimedia. (1995) http://fishwrap-docs.www.media.mit.edu/docs/dev/CNGlue/cnglue.html
11. Chimera, R., Shneiderman, B.: Three Interfaces for Browsing. ACM Transactions on Information Systems. 12 (1994) 383–406
12. Dillon, A.: Reading from Paper Versus Screens: A Critical Review of the Empirical Literature. Ergonomics. 35 (1992) 1297–1326
13. Dozier,D., Rice, R.: Rival Theories of Electronic News Reading. In: Rice, R. (ed.): The New Media. Sage Publications, London (1984) 103–128
14. Duchnicky, R., Kolers, P.A.: Readability of Text Scrolled on Visual Display Terminals as a Function of Window Size. Human Factors. 25 (1983) 683–692

15. Fidler, R.: Mediamorphosis, or the Transformation of Newspapers into a New Medium. Media Studies Journal. 5 (1991) 115–125
16. Garcia, M., Stark, P.: Eyes on the News. The Poynter Institute, St. Petersburg, Florida (1991)
17. Haake,A., Huser, C., Reichenberger, K.: The Individualized Electronic Newspaper: An Example of an Active Publication. Electronic Publishing. 7 (1994) 89–111
18. Hoffert, E.M., Gretsch, G.: The Digital News System at EDUCOM: Computing, Newspapers, Television and High-Speed Networks. Communications of the Association for Computing Machinery. 34 (1991) 113–116
19. Horton, W.: Visual Rhetoric for Online Documents. IEEE Transactions on Professional Communication. 33 (1990) 108–114
20. Kamba, T., Bharat, K., Albers, M.C.: The Krakatoa Chronicle - An Interactive, Personalized, Newspaper on the Web. Proceedings of the Fourth International World Wide Web Conference (1995) http://www.w3.org/pub/Conferences/WWW4/Papers/93/
21. Karmouch, A.: A Distributed Interactive Multimedia System Over an ATM Network. Project summary distributed at CASCON'95. Dept. of Electrical Engineering, University of Ottawa, Ottawa, Canada (1995)
22. Lamont, L., Georganas, N.D.: Synchronization Architecture and Protocols for a Multimedia News Service Application. Proceedings of the International Conference on Multimedia Computing and Systems. IEEE Computer Science Press, Los Alamitos, California (1994) 3–8
23. Markoff, J.: A New Generation of On-Line Services is Coming to the Internet.The New York Times, Business Section, (January 30, 1994) 10
24. McLuhan, M.: Is It Natural That One Medium Should Appropriate and Exploit Another? In: Stern, Gerald E. (ed.): McLuhan: Hot and Cool. Signet Books. New American Library, New York (1967). Reprinted In: McLuhan, E., Zingrone, F. (eds): Essential McLuhan. House of Anansi Press Limited, Concord, Ontario (1995).
25. McQuail, D.: Media Performance. Sage Publications, London (1992)
26. Mills, C.B., Weldon, L.J.: Reading Text from Computer Screens. ACM Computing Surveys. 19 (1987) 329–358
27. Noras, S.R.: All the News That's Fit to Screen – The Development of Fulltext Newspaper Databases. Australian Library Journal. 38 (1989) 17–27
28. Ohkubo, M., Kobayashi, N., Nakagawa, T.: Design of an Information Skimming Space. Proceedings of ACM Multimedia'93. (1993) 365–371
29. Ozsu, M.T., Szafron, D., El-Medani, G., Vittal, C.: An Object-Oriented Multimedia Database System for a News-On-Demand Application. Multimedia Systems. 3 (1995) 182–203
30. Pack, T.: Electronic Newspapers – The State of the Art. Proceedings of the 14th National Online Meeting. (1993) 331–335
31. Paterson, D.G., Tinker, M.A.: Readability of Newspaper Headlines Printed in Capitals and in Lower Case. Journal of Applied Psychology. 30 (1946) 161–168
32. Reinhardt, A.: Electronic Newspaper Offers Color Pictures. Byte. (September 1993) 40
33. Shepherd, M.A., Watters, C.R., Burkowski, F.J.: Digital Libraries for Electronic News. Advances in Digital Libraries. (1995) 13–20
34. Shepherd, M.A., Watters, C.R., Burkowski, F.J.: Delivery of Electronic News. The Second International Workshop on Next Generation Information Technologies and Systems. (1995) 124–131
35. Stephenson, W.: The Play Theory of Mass Communication. University of Chicago Press, Chicago (1967)

Author Index