Personalized Information Structures†

Scott R. Tilley Michael J. Whitney Hausi A. Müller Margaret-Anne D. Storey

Department of Computer Science, University of Victoria
P.O. Box 3055, Victoria, BC, Canada V8W 3P6
Tel: (604) 721-7294, Fax: (604) 721-7292
E-mail: {stilley, mwhitney, hausi, mstorey}@csr.uvic.ca

Abstract

When moving linear documentation into a hypertext system, it is important to distinguish between *referential* and *structural* links; both are needed to model the literary paradigm. In particular, structural links facilitate navigation, tailoring, and information retrieval by imposing structure on large documents. Without them, users face the well-known "lost in hyperspace" syndrome due to disorientation caused by a tangle of referential links in the hypertext web. To be truly effective, hypertext systems should support a level of customization at least equivalent to paper-based documentation systems. The hyperdocument's structure must be malleable and user-customizable. It should be the reader who decides what is the best document architecture—not the writer. This paper describes a flexible *reverse-engineering* approach to creating, representing, and structuring online documentation. The approach permits the construction and maintenance of *personalized information structures*: multiple virtual documents over the same hypertext database.

Keywords: hypertext, information structure, layered graphs, online documentation

1 Introduction

The large body of existing textual documentation presents a serious challenge to the successful introduction of online and multimedia systems into the workplace. Hypertext systems that only support authoring are limited in their usefulness; they should (at a minimum) support the browsing of linear documents. What is needed is a way to smoothly integrate text-based documents with hypertext. It is not sufficient to simply place the original document into the hyperbase; the document's inherent structure should be extracted

†This work was supported in part by the British Columbia Advanced Systems Institute, IBM Canada Ltd., the IRIS Federal Centres of Excellence, the Natural Sciences and Engineering Research Council of Canada, the Science Council of British Columbia, and the University of Victoria.

and represented in its hypertextual counterpart.

Unfortunately, most techniques for converting linear documentation into hypertext have limited structuring capabilities. Hypertext does offer an improvement over traditional linear documentation by permitting navigation through the information web via links. However, for very large online documents, such *referential* links by themselves are insufficient; one needs to impose a *structuring mechanism* on top of the hypertext web to support more intuitive navigation and information retrieval.

Before online documentation, people personalized their printed text by writing in the margins, underlining phrases, and by putting “dog ears” on pages that were of interest to them. With the advent of online documentation, navigation became easier because of facilities supporting non-linear search, pattern-matching, and electronic “bookmarks.” While such bookmarks shorten navigation time for subsequent searches, they are essentially one-dimensional: they lack the ability to *structure* the document as the user would like. Users should be able to select related pieces of information from a large online document, and organize this information into a *virtual document*. Many users would prefer a hierarchical information structure, but containing only the information pertinent to their particular needs. Others may prefer a non-hierarchical structure suiting their own tastes and navigational abilities. Because of such personal preferences for document structure, it is unlikely that any single choice made by the writer will suit all readers. Ultimately, it is the reader who decides if the structure of a document is adequate—not the writer.

This paper describes an approach to personalizing document structures. The approach is based on exploiting the parallels between the dual problems of software maintenance and understanding, and document maintenance and understanding [1]. Conversion of text to structured hypertext is very similar to uncovering and representing program structure. Our previous work in software maintenance and program understanding has made use of *reverse engineering* to recover the structural aspects of the architecture of large software systems [2]. Reverse engineering is the process of extracting high-level abstractions and design information out of existing systems. It involves the identification of software artifacts in a particular representation of a subject system via mental pattern recognition by the software engineer, and the aggregation of these artifacts to form more abstract structured representations. The basic goals of reverse engineering as applied to software systems can be successfully applied to the conversion of linear documents to *personalized information structures*.

The next section discusses document structure and the conversion of linear text to structured hypertext. It also introduces personalized information structures as the logical successor to structured hypertext. Section 3 outlines how these structures are created by our system and describes how the conversion process is similar in nature to our previous work on software maintenance. Section 4 describes how personalized information structures are used and gives several examples of their use in our system. Section 5 summarizes our approach and contributions.

2 Document structure

Because documents are such rich sources of structure information, presenting them from various viewpoints is essential for an interactive document retrieval system [3]. For example, a typesetter may be interested in a document's physical appearance, while an editor may be more concerned with its contents [4]. While printed text has an inherently linear structure, it is not without other structuring mechanisms, such as aggregation due to section level nesting, and referential relations between spans.[1] Hence, textual documents have (at least) a double structure: one defined by inter-span relations, and one induced by the nesting of sections. In fact, they have other structural dimensions as well; when converting text to hypertext, it is important to capture and distinguish amongst them.

This section describes the translation of text documents into personalized information structures: linear text $\Rightarrow$ hypertext $\Rightarrow$ structured hypertext $\Rightarrow$ personalized information structures. The process involves the automatic translation of the original text document into structured hypertext. During the translation, the structure inherent in the text document is captured and mapped into hypertext equivalents. We first discuss document structure representation and introduce a formalism used to model textual relations. The importance of structuring hypertext documents is then discussed. The structural features which must be extracted from the original documents to model the literary paradigm in the resultant structured hypertext are presented. Personalized information structures are then introduced as an improvement over structured hypertext.

2.1 Representing document structure

One way of representing document structure is with a semantic network: a labeled, directed graph. Each node in the graph represents an object, and each arc represents relations between objects. Semantic networks have been used in many areas for knowledge representation, including software engineering [6] and hypertext [7]. Since the central role of all hypertext systems is that of linking text together, semantic networks are well suited to representing them. Objects in the semantic network may be spans or they may be navigation nodes (which hold indirect pointers to the spans). In fact, one could have a network entirely devoid of text, with just the "hypertext backbone" to structure the document. The semantic and logical connections in the text may be represented as typed links in the hypertext.

The structuring information inherent in a semantic network and the relations among nodes and arcs may be described using *interconnection models* (IM's). IM's are a formalism used to describe relationships among objects as a set of tuples [8]: $\mathsf{IM} = (\{objects\}, \{relationships\})$. This set of tuples conveniently maps to a graph structure, with the objects being nodes and the relationships being attributed arcs between the nodes. Hence, interconnection models may be used to represent document structure and to model relationships in hypertext. As a formalism, they also have the advantage of allowing us to perform structure queries on the hypertext. Such queries can only be carried out if the structure is well-defined [9].

[1] We use the term "span" [5] to refer to arbitrary textual units.

2.2 Structured hypertext

The main disadvantage of using a simple (flat) semantic network to represent hypertext is that very little structure is imposed. It is useful to explore the analogy between the evolution of structured programming and the development of hypertext [10, 11, 12]. Links as described in Section 2.1 are analogous to `goto` statements in programming languages. Just as a multitude of `goto`'s renders a program incomprehensible, a multitude of links in a hypertext system renders the document equally incomprehensible. Early programs were riddled with `goto`'s, until structured programming reduced the need for them; hypertext systems need a similar structuring facility.

The well-known "lost in hyperspace" syndrome has been attributed to disorientation caused by a tangle of links in the hypertext web. The proliferation of links is often due to the weak link discipline enforced by a system using a simple node/link mechanism, allowing unrestricted linking among arbitrary objects [13]. Such linking is very powerful, but potentially disorienting [14]. The same freedom which provides hypertext's flexible structure and browsing capabilities may also be the direct cause of one of its greatest problems [15]. For users, disorientation may occur when browsing. For authors, the lack of design principles when creating associative links does not foster the creation of a consistent conceptual model [16].

Some of the solutions that have been proposed to the classical problem of user disorientation within a hypertext web include: maps, multiple windows, history lists, and tour/path mechanisms. Unfortunately, these methods do not scale up well for large hyperdocuments. A more successful approach is through the use of *composite nodes*; they reduce web complexity and simplify its structure by clustering nodes together to form more abstract, aggregate objects [17]. Composite nodes deal with sets of nodes as unique entities, separate from their components. They act as approximations to their constituent nodes [18], and may be used to capture high-level relationships.

One of the most important relationships in document structure is that of *inclusion*. It is created by the nesting of section levels in the document. Composite nodes may be used to represent document sections, giving rise to a *cluster hierarchy* in which leaf nodes contain spans and internal nodes represent document sections. Since composite nodes may be nested to an arbitrary depth, they are well-suited to represent the classical hierarchical organization of documents. Because hierarchies supply structural information not available in a directed graph, it is important to capture this relationship [19].

A hierarchy is often the optimal form for expository texts, making structured hypertext even more important in conveying information to the reader. Hierarchical organization of information is central to reading and writing; it is an ordering concept that is familiar yet powerful. In addition, documents structured hierarchically can provide numerous visual benefits to users [20, 21], something that is very important in a graphical hypertext environment.

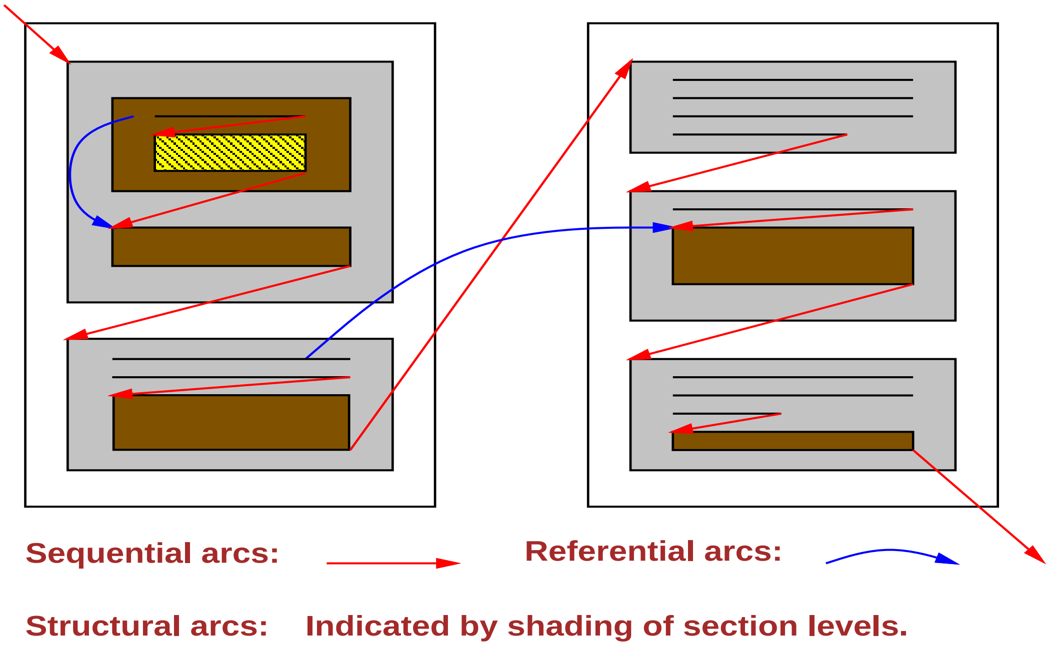

Figure 1: Document structure

2.3 Structural feature extraction

Conversion from text to hypertext is a viable alternative to manual hypertext composition [22]. In fact, when working with a large document base it becomes the only realistic choice for initial hyperbase population. One of the goals in the conversion process is to maintain the literary paradigm inherent in the source document. This implies the hypertext must have an architecture beyond simple nodes and arcs, and relations in the source text must be properly mapped into hypertext links. In other words, the product of the conversion process must be structured hypertext as in Section 2.2. To do this, several structural features must be extracted from the original document and represented in the resultant hypertext. At a minimum, the inherent links between spans which are linear (sequential) and non-linear (referential) must be extracted [23].

It should be noted that while we are mainly concerned with the conversion of documents tagged with descriptive information (for example, via LaTeX or SGML), the techniques discussed in this section will also work for unadorned text. However, by making use of existing markup describing document structure the resultant hypertext will better approximate the original text. Examples of markup for indicating relations among spans include footnotes, reference sections, quotations, citations, glossaries, indices, and tables of contents. The conversion process should capture these relations and represent them as live hypertext links. Naturally, the user should be able to create links external to the document's original structure (for example, for annotational purposes). Section 4 describes the use of such links.

There are three structural aspects of linear documentation which must be extracted to produce structured hypertext. As illustrated in Figure 1, they are:

(1) Hierarchical sectioning: Documents have a natural hierarchy created by section levels. It is important to accurately reflect this hierarchy in hypertext. Otherwise, the result of conversion is an unstructured collection of spans rather than the desired structured hypertext. This structure may be represented as a layered graph, where each layer represents a nesting level in the document. The hierarchical structure is based upon a set of inclusion relations between sections and may be represented using structural links in the hypertext. Its interconnection model is $\mathsf{Structural\ IM} = (\{section\}, \{contains\})$.

(2) Sequential ordering: The ordering between sections imposes a sequential structure which represents reading threads. In hypertext terms, this structure may be thought of as implicit links between sections which the user follows when browsing a document in a linear fashion. It is important to capture these discourse clues so as to reflect the original ordering among spans which the text's author intended [24]. Section sequencing may be modeled as a partial order and represented as a sequential link in the hypertext. Its interconnection model is $\mathsf{Sequential\ IM} = (\{span\}, \{precedes\})$.

(3) Referential relations: References from one section of the document to another, such as "See Section 2.3 for more information," represent important information to readers. Other examples of such static references are citations, footnotes, page and section references, and indices. In hypertext terms, such references should be captured and translated into referential links. Note that such relations may be made explicit in the document if it was written using descriptive markup (see Section 3 for examples of such relations in LaTeX). Relations between spans may be modeled as explicit references and represented as a referential link in the hypertext. The interconnection model for referential relations is $\mathsf{Referential\ IM} = (\{span\}, \{refers\text{-}to\})$.

2.4 Personalized information structures

Structured hypertext is an improvement over regular hypertext; document hierarchy and hypertext links provide two organizational axes for the abstraction of structure. However, this can be improved upon. A third axis may be realized through multiple, virtual arrangements of the first two. These three axes form the basis for *personalized information structures*: user-defined, multiple, virtual hyperdocuments over the same hyperbase.

As stated in Section 1, the success of moving new technology into the workplace depends crucially on the acceptance of the system by its users. If they find the system too different from what they are currently using they will be loathe to change. However, if they can tailor the new system to fit into their existing environment and work the way they want it to—not the way the designer thought they might want it to work—then the system has a much better chance of success. Hence, the goal is to provide the user with as much flexibility as possible in structuring the hypertext to suit their needs. Similar extensibility has proven effective in other hypertext systems (for example, [25]), and in our own work on program understanding [26].

One of the advantages of structured hypertext is the use of a hierarchy as a structuring mechanism. Such a hierarchy can be used to address a single concept; a (flat) semantic network has no such central theme. However, the structure of the hierarchy is fixed by the author. Moreover, it can only address a single theme at a time. Hence, the logical successor to such a single-viewed static mechanism is to allow multiple views of the same information, and to allow multiple hierarchies.

With personalized information structures the user is able to create their own view of the large underlying document, and to structure the pieces of information related to a task as a mini-document. In this way, the user creates multiple views of the document, each view pertaining to a particular task. Since the document structure created is virtual, each user is using the same underlying information base.

One of the advantages of personalizing hypertext has to do with information search and retrieval. Users often fail to find pertinent information during online searches because they describe the items they are searching for in terms different than that stored in the system [27, 28], or because they become disoriented while navigating [29]. By structuring the hypertext the way they wish, the representation and the mental model of a concept can be much closer. Searches become content-based conceptual searches with a much higher chance of success. It has been reported that long-time users of paper-based documentation can find information faster and more efficiently than in hypertext systems because of the ability of the paper to be *customized*, such as by writing in the margin, underlining parts of the text, or leaving bookmarks [30]. Personalized information structures offer a superset of the same capabilities for hypertext-based documentation.

For example, after searching through a document for specific information the first time, the navigation can be recorded for later use. Such searches will create sets of navigation paths and hypertext structures for each search. These paths and structures may be saved and recalled at a later date. The end result is a malleable hypertext structure with many layers of *conceptual webs* built upon it. Each web corresponds to a concept (or concepts) in the user's domain. Such webs may be shared among users and used simultaneously.

To summarize, text may be automatically converted to hypertext and represented using a semantic network. Structuring this network in a hierarchical manner reduces disorientation and increases usability for large hyperdocuments. The conversion process captures three of the most important structural features of the literary paradigm, which may be represented using the formalism of interconnection models. However, the resultant hypertext is still static and two-dimensional. Personalized information structures offer an improvement over structured hypertext by lifting the restrictions imposed by such an author-oriented environment. While other systems do exist for accessing existing documentation in a static hypertext form, they do not address authoring new information structures built upon the originals. Personalized information structures bridge the two domains of authors and end-users. The next section describes how the creation of personalized information structures from existing text is accomplished in our system.

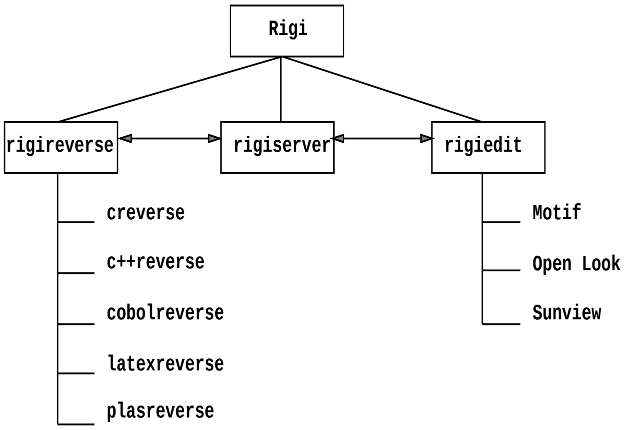

Figure 2: The Rigi system's main components

3 Supporting personalized information structures

Personalized information structures as described in Section 2 are created from linearly organized online documents using Rigi,[2] a versatile system and framework under development at the University of Victoria. The primary use of Rigi to-date has been for reverse engineering and redocumentation of software systems [31]. In this process, the discovery and analysis of very large system structure is facilitated. However, the underlying structuring mechanisms used by Rigi are well-suited as a backbone for hypertext. Indeed, the philosophy underlying reverse engineering of software is equally applicable to documentation.

In addition to parsing source code of various programming languages, Rigi has been augmented to parse documents (in particular, LaTeX files) to produce an initial representation of personalized documents based on the document's inherent structure. Relations among textual spans are used to build a semantic network, where nodes correspond to spans, and arcs represent various semantic relationships. The network is used as a basis upon which to build multiple virtual documents representing reading threads for specific tasks. Rigi's graph editor offers several semi-automatic clustering mechanisms for imposing new structure on graphs. These mechanisms are employed by users to create views. The abstraction mechanisms of Rigi accommodate structures which may or may not be hierarchical, depending on the user's preference. Thus, users may create personalized manuals that suit their own requirements and intuition.

The Rigi system consists of three main entities: the *parser*, the *database*, and the *editor*. Each of these are modifications to three similar parts of our software reverse engineering tool: *rigireverse*, *rigiserver*, and *rigiedit* respectively. Essentially, we have introduced a new application domain—that of understanding document structure—to Rigi's original purpose of program understanding. This section provides a high-level overview of the structure of these subsystems and how they communicate. The overall system structure is depicted in Figure 2.

[2]Rigi is named after a mountain in central Switzerland.

3.1 The parser

The creation of a new personalized information structure begins with the transformation of an existing linear document into a more hypertext-oriented form. Currently, we use a simple text parsing system that extracts structure, other relations, and actual text from LaTeX source. We have chosen this text markup language as the source type since LaTeX documents are in plentiful supply, and thus our system has immediate broad application. Our intention is not to duplicate the LaTeX parser in its entirety, but simply to extract typesetting features of relevance to structural and referential characteristics of documents. As such, the parser extracts the following information from the source (as discussed in Section 2.3):

Hierarchical sectioning: In LaTeX, structure is specified both in *absolute* and *relative* terms. Absolute mechanisms include the use of keywords such as `\chapter` and `\section`. An absolute textual scope defined by a particular keyword remains in existence until over-ridden by another keyword of equal or greater significance. Relative structure is specified using the "environment" constructs `\begin` and `\end`. The low-level objects are mostly paragraphs, but also include such constructs as figures, tables, and items in lists (for example, the `\enumerate` and `\itemize` relative environments). Paragraphs are usually recognized without the explicit use of an absolute keyword. Instead, one or more blank lines, and various LaTeX commands, delimit them.

Sequential ordering: A sequential link implicitly exists between adjacent portions of text in a linear document. This rather obvious observation is of great importance to our system because it implies an *order* amongst objects. For example, if Chapter 3 of a manual contains four sections, then it is expected that they occur in the order in which they are parsed, and may be named Section 3.1, 3.2, 3.3, and 3.4. The parser would output sequential links between these sections, plus one from the node representing Chapter 3 to the node representing Section 3.1.

Referential relations: In LaTeX, references include citations of bibliographical entries using `\cite`, labelling using `\label`, and explicit references (`\ref` and `\pageref`). Directed referential links are established between each reference and the label (or bibliography entry) that is referenced.

User-defined relations: Hierarchical structure and explicit references are recognized through the use of LaTeX keywords. We have augmented these mechanisms by providing a facility for specifying additional keywords at feature extraction time. These keywords are not LaTeX commands; they are usually technical words of special relevance to a particular domain. New referential relationships are established using locations in the text where these keywords occur, providing an initial user-defined web of hypertext links amongst textual objects.

Parsing the document populates the database. The output consists of nodes and arcs representing the document's structure, and text extracted from the document itself, including paragraphs, headings such as `\chapter{Introduction}`, and other LaTeX commands. The nodes and arcs form a semantic network, which is the dual of the resource-flow graph created by Rigi when reverse engineering software. The LaTeX commands are retained with the rest of the text, because one of the uses of the system is the re-authoring of linear documents. Moreover, typesetting can be performed from the editor itself, as described in Section 3.3.

3.2 The database

There are two main types of data that must be maintained by the system: graphical objects, and text. The database subsystem takes care of both types. Facility is provided for supporting user-supplied annotations and editing all forms of stored objects. In this way, multiple personalized structures over the same document are maintained.

The originally parsed document remains unchanged. However, a new copy is created and stored in the database, along with the structural view needed to re-create the original. Users' hypertextual views consist of structured graphs with links back to this stored document's text, plus their own personal annotations. These are stored in personal workspaces.

3.3 The editor

The editor provides the graphical user interface (GUI) through which authoring, editing, and browsing of personalized information structures is performed. Hence, this subsystem is the heart of Rigi from the user's perspective. The presentation of the editor is more hypertext-oriented than would be required for program understanding alone, although the functionality is largely the same. Individual and grouped objects, including nodes, arcs, and spans may be selected, moved, filtered, opened, edited, destroyed, created, and renamed.

Within the editor, hypertext objects may be represented in various ways, but the most important form is an iconic one. Icons represent textual scopes output by the parser. The editor provides different icons for documents, parts, chapters, sections, and so on. They are initially attributed with names derived from headings in the text, plus the appropriate number (for example, `Chapter 2 :  Background`). Names are editable, but numbers are automatically maintained properly.

Of key relevance to structural editing are the *expand* and *collapse* operations. Expansion of an icon results in that icon being replaced with the objects it contains. These revealed icons can be moved about and grouped into new clusters. Such a group may then be collapsed, resulting in a new icon that can be renamed, annotated, and moved to a new location. In this way, structure and textual scoping levels of information structures are malleable. The editor keeps track of the appropriate section numbering for new icons as well as for existing icons whose textual scopes have changed. For example, if a new collapsed icon is moved between one named Section 3.1 and one named Section 3.2, the latter is renamed Section 3.3 and the new one becomes Section 3.2.

Graphical objects and text are displayed in windows. There are four main window types used in the editor:

(1) **Standard:** These windows correspond to one node representing one particular structural scope; they contain the node's immediate subscopes. Icons appearing within a standard window need not belong to the same absolute scope. For example, a standard window for Chapter 3 might contain an icon representing an introductory paragraph, then icons for Sections 3.1, 3.2, and 3.3. Most structural editing operations are restricted to standard windows.

(2) Overview: This type of window offers improved visibility of overall document structure. Structural "contains" relationships are visible as arcs. Sequential relationships are depicted, as they are in standard windows, by enforced top-to-bottom ordering or left-to-right ordering.

(3) Projection: Projection windows are more flexible than overview windows: the number of levels visible at one time can be restricted, and the graph structure that is visible to the user is tailorable by choosing the type of relation (structural, sequential, and so on) that is depicted. For example, a projection window could show the first four nodes, in sequential order, of a given chapter. Another could show all nodes belonging to that chapter, down to the subsection level. Still another could follow referential links from a given paragraph.

(4) Text projection: These are similar to projection windows, except that text is displayed instead of icons. Projected text can be viewed in either editable form (with embedded LaTeX commands), or actually typeset as it would appear on paper. Highlighted text in the text projection window corresponds to selected nodes in the others.

In standard windows, of the four interconnection types described in Section 3.1, only referential links are normally visible as arcs linking icons. Structural links are implicit in the action of "opening" an icon, resulting in a new standard window with new icons representing textual subscopes of their parent. Sequential relationships are implicit in the left-to-right or top-to-bottom order in which icons are displayed in a window.

The editor provides the capability for automatically *synthesizing* referential links up through the tree representing a hierarchically structured view. For example, if Section 3.2 references Section 4.1.3, then a synthesized referential arc links Chapters 3 and 4. This arc is visible in any window that contains both these nodes. Synthesized arcs may be filtered if desired.

Relations based on user-supplied keywords can be created through the editor, complimenting those created at feature extraction time through the parser. These relations are different than the others in that they are not directed, they are multi-way, and they are "named". Thus, a special kind of widget is employed, in which the keyword itself is displayed, plus a scrollable list of all textual scopes containing that keyword. The scopes are identified by a numbering scheme extracted by the parser, and may be set at any level. They are highlighted (whether appearing in textual or symbolic form), wherever they occur, when selected from the widget. All user-supplied keywords are available in a sorted directory, through which the individual widgets are accessible.

Rigi's graph editor offers several semi-automatic clustering (aggregation) mechanisms for imposing additional structure on graphs. These mechanisms are employed by users to create new views and thereby new personalized information structures. Thus, the conversion of structured hypertext to personalized information structures is semi-automatic, in contrast to the automatic conversion of text to structured hypertext. The abstraction mechanisms of Rigi accommodate graph structures which may or may not be hierarchical, depending on the user's preference. The use of some of the more important of these operations is discussed in the next section.

4 Using personalized information structures

This section describes the use of personalized information structures, implemented as described in Section 3. Once the source document has been parsed and the database populated with textual artifacts, the user immediately has access to a structured hypertext version of the original text. This version is stored in their personal workspace, although it can also be stored in a shared workspace if desired (for example, when using a hyperbase which was created by someone else). Any changes which are made to the document, content-wise or structure-wise, are stored in the same workspace; the original document itself is not altered.

The user is then free to create their own personalized version of the structured hypertext document. They may do so using the tools and methods provided by the editor, as described in Section 3.3. For example, they can annotate spans using the node annotator, permanently delete spans which are of no interest to them, restructure the document to their taste by reordering sections or even moving paragraphs from one section to another, create and save different (perhaps even contrasting) views of the document by using the filtering operations on nodes and arcs, and by projecting nodes and/or text to selected depths.

As an example, consider the need to provide customers with customized copies of technical documents. In our case, we have a user's manual for Rigi which contains three distinct subsections concerning the three different GUI's that we support (as depicted in Figure 2). Most users are only concerned with a single version of the editor, say the Motif[3] version, hence the other two parts of the section are unimportant. By filtering these sections out, the user will have a manual tailored for their own use. Naturally, they can make use of the view and annotation facilities provided by the editor to make notes on the use of the system as described in the manual. They may also use the powerful collapse and expand operations, along with node and link creation, to create truly personalized versions of the user's manual.

To illustrate these concepts, the LaTeX source text to this paper is used as the base document. Two different dependency structures of Section 2 of this paper are shown in Figure 3.[4] The left hand side of the figure shows the sequential links between sections and subsections; this is the linear ordering of the spans in the paper. The right hand side shows the structural links between sections and subsections; this is the nesting of spans in the paper.

Figure 4 shows the same section of the paper as it is actually displayed for the user. The top window shows the same structural information for Section 2 as depicted in the right hand side of Figure 3, while the two windows below it show both the sequential and referential relations of subsections 2.1 and 2.3 respectively. The left window shows the contents of Section 2 projected to text level. However, for this example the projection is limited to two levels.

The parameterization of the projection operations is very powerful. For projection to text, it implies the visualization of the textual portions of the document can be restricted to higher-level sections (like an outliner), or it can show all the text if desired. Note that multiple nodes can be selected for projection at

[3](OSF/Motif) is a trademark of the Open Software Foundation.

[4]To limit the size of the figure, not all of the structural dependencies of Section 2 are shown. The dashed line between Subsection 2.2 and Section 3 indicates the presence of intervening subsections.

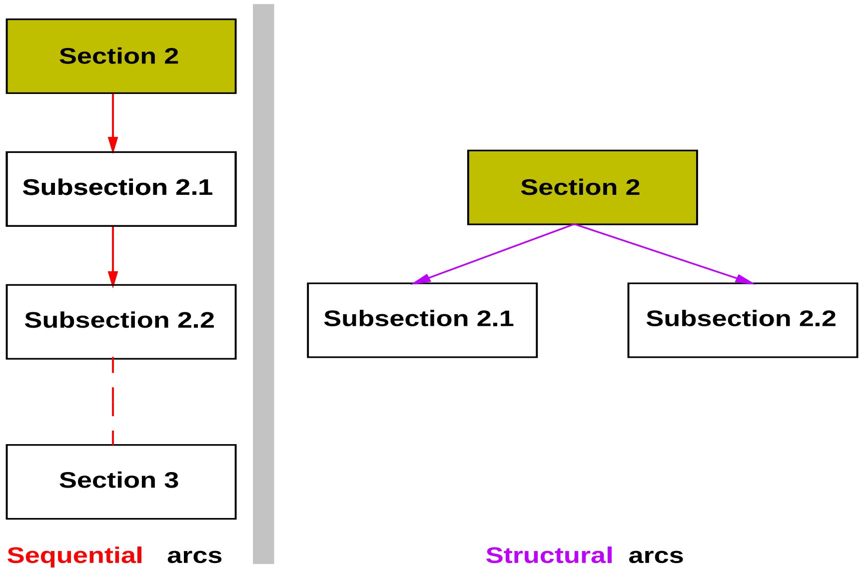

Figure 3: Forward dependency trees of Section 2

the same time. For example, the user could choose to view the text of just Chapter 1, Section 3.1, and Section 3.2. The text displayed appears in order of their sequential arcs. In addition, the node annotations may also appear in this window; if present, they are displayed in a different color. Each of the paragraphs and node annotations may be enlarged or reduced in the project-to-text window (for example, if the user is not interested in a particular paragraph at a certain level).

The user also has the option of viewing a formatted version of the raw LaTeX source if desired. The system will use the style files present in the original document (if available), "wrap" the span in other header and footer material required for the LaTeX parser and run the normal formatting command on the temporary document. The resultant PostScript file can then be previewed using tools such as `ghostview`. Sets of windows and annotations may be saved as views [31] and recalled for later use.

5 Summary

Hypertext has been described as a tool to enhance human cognitive abilities. One of the original goals of hypertext was to allow readers to impose their own structure on the information. In normal hypertext this is accomplished through the selection of appropriate nodes [32].

We have approached these goals of hypertext by exploiting our previous work on program understanding. There exist parallels between the representation, maintenance, and presentation of source code, and that of documentation. Our approach to reverse engineering of software systems is flexible enough to be used in a new domain: reverse engineering of documentation. The original text document is automatically converted into structured hypertext through a process which captures essential structural features of the

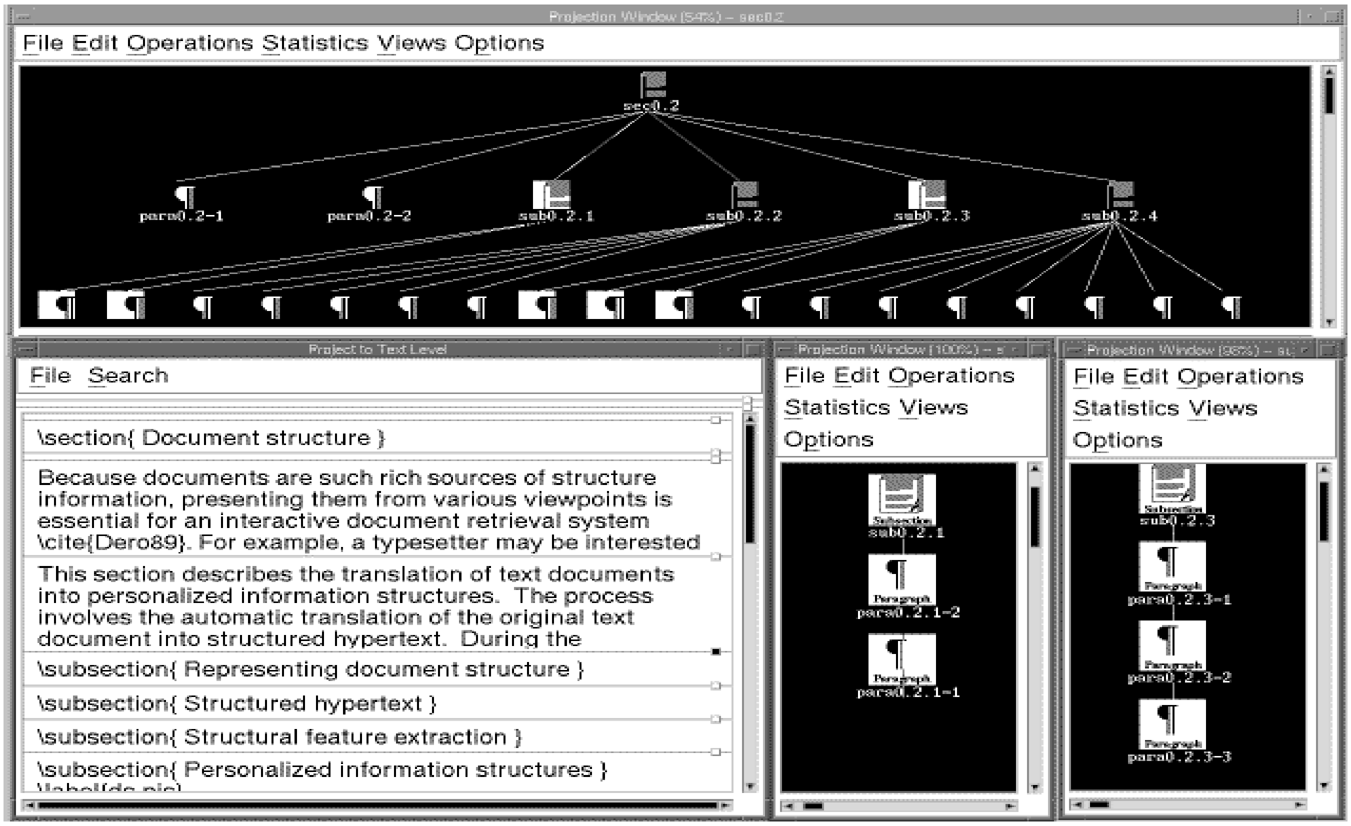

Figure 4: Personalized information structures in use

original document. These features model the literary paradigm of section nesting and span relations and are identified based on keyword sets and the document's physical structure. The graph editor is then used to semi-automatically construct personalized information structures based on the single structured hyperdocument. Interconnection models provide a formalism which allow the users to pose structural queries on their virtual documents.

We have presented a natural evolution of document structure, from linear text to hypertext to structured hypertext to personalized information structures. The latter allows the user to select their own level of detail for different parts of the document. The structure becomes dynamic, user-definable, and is not as restrictive as a single hierarchy. Personalized information structures allow the user to be in control of how a document is structured, presented, and used—not just the author.

References

[1] S. Matthews and C. Grove. Applying object-oriented concepts to documentation. In *Proceedings of SIGDOC'92: The 10th International Conference on Systems Documentation,* (Ottawa, Ontario; October 13-16, 1992), pages 265–271, October 1992. ACM Order Number 613920.

[2] H. Müller, B. Corrie, and S. Tilley. Spatial and visual representations of software structures: A model for reverse engineering. Technical Report TR-74.086, IBM Canada Ltd., April 1992.

[3] S. J. DeRose. Expanding the notion of links. In *Proceedings of Hypertext '89* (Pittsburgh, Pennsylvania; November 5-8, 1989), pages 249–257, November 1989. ACM Order Number 608891.

[4] P. K. Garg. Abstraction mechanisms in hypertext. In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), pages 375–395, November 1987.

[5] G. H. Collier. Thoth-II: Hypertext with explicit semantics. In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), pages 269–289, November 1987.

[6] H. Müller and J. Uhl. Composing subsystem structures using (k,2)-partite graphs. In *Proceedings of the Conference on Software Maintenance 1990,* (San Diego, California; November 26-29, 1990), pages 12–19, November 1990. IEEE Computer Society Press (Order Number 2091).

[7] H. Kaindl and M. Snaprud. Hypertext and structured object representation: A unifying view. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 345–358, December 1991. ACM Order Number 614910.

[8] D. E. Perry. Software interconnection models. In *ICSE'9: Proceedings of the 9th International Conference on Software Engineering*, pages 69–69, April 1987.

[9] F. G. Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[10] A. van Dam. Hypertext '87 keynote address. In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), November 1987.

[11] R. Furuta and P. D. Stotts. Programmable browsing semantics in trellis. In *Proceedings of Hypertext '89* (Pittsburgh, Pennsylvania; November 5-8, 1989), pages 27–42, November 1989. ACM Order Number 608891.

[12] M. B. (Moderator). Panel discussion on structure, navigation, and hypertext: The status of the navigation problem. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 363–366, December 1991. ACM Order Number 614910.

[13] J. Nanard and M. Nanard. Using structured types to incorporate knowledge in hypertext. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 329–343, December 1991. ACM Order Number 614910.

[14] D. Broady, H. Haitto, P. Lidbaum, and M. Tobiasson. Darc: Document archive controller. Technical Report TRITA-NA-P9306, IPLab/NADA, Royal Institute of Technology (Sweden), March 1993.

[15] R. A. Botafogo and B. Shneiderman. Identifying aggregates in hypertext structures. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 63–74, December 1991. ACM Order Number 614910.

[16] Y. Hara, A. M. Keller, and G. Wiederhold. Implementing hypertext database relations through aggregations and exceptions. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 75–90, December 1991. ACM Order Number 614910.

[17] M. A. Casanova, L. Tucherman, M. J. D. Lima, J. L. R. Netto, N. Rodriguez, and L. F. G. Soares. The nested context model for hyperdocuments. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 193–201, December 1991. ACM Order Number 614910.

[18] H. L. Ossher. *A New Program Structuring Mechanism Based on Layered Graphs.* PhD thesis, Stanford University, 1984.

[19] J. B. Smith, S. F. Weiss, and G. J. Ferguson. A hypertext writing environment and its cognitive basis. In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), pages 195–214, November 1987.

[20] D. B. Crouch, C. J. Crouch, and G. Andreas. The use of cluster hierarchies in hypertext information retrieval. In *Proceedings of Hypertext '89* (Pittsburgh, Pennsylvania; November 5-8, 1989), pages 225–237, November 1989. ACM Order Number 608891.

[21] H. Müller, S. Tilley, M. Orgun, B. Corrie, and N. Madhavji. A reverse engineering environment based on spatial and visual software interconnection models. In *SIGSOFT '92: Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments,* (Tyson's Corner, Virginia; December 9-11, 1992), pages 88–98, December 1992. In *ACM Software Engineering Notes*, 17(5).

[22] D. R. Raymond and F. W. Tompa. Hypertext and the New Oxford English Dictionary. In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), pages 143–153, November 1987.

[23] G. A. Boy. Indexing hypertext documents in context. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 51–61, December 1991. ACM Order Number 614910.

[24] D. Charney. Comprehending non-linear text: The role of discourse cues and reading strategies. In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), pages 109–120, November 1987.

[25] P. D. Stotts and R. Furuta. Dynamic adaptation of hypertext structure. In *Proceedings of Hypertext '91* (San Antonio, Texas; December 15-18, 1991), pages 219–231, December 1991. ACM Order Number 614910.

[26] S. R. Tilley, H. A. Müller, M. J. Whitney, and K. Wong. Domain-retargetable reverse engineering. *CSM '93: The 1993 International Conference on Software Maintenance,* (Montréal, Québec; September 27-30, 1993), pages 142–151, September 1993. IEEE Computer Society Press (Order Number 4600-02).

[27] V. Bush. As we may think. *The Atlantic Monthly*, 176:101–108, 1945.

[28] J. R. Remde, L. M. Gomez, and T. K. Landauer. Superbook: An automatic tool for information exploration: Hypertext? In *Proceedings of Hypertext '87* (The University of North Carolina, Chapel Hill, North Carolina; November 13-15, 1987), pages 175–188, November 1987.

[29] H. A. Müller. $(k, 2)$-partite graphs as a structural basis for the construction of hypermedia applications. Technical Report DCS-119-IR, University of Victoria, June 1989.

[30] F. M. Fillion and C. D. Boyle. Important issues in hypertext documentation usability. In *Proceedings of SIGDOC'91: The 9th International Conference on Systems Documentation,* (Chicago, Illinois; October 10-12, 1991), pages 59–66, October 1991.

[31] S. R. Tilley, H. A. Müller, and M. A. Orgun. Documenting software systems with views. In *Proceedings of SIGDOC '92: The 10th International Conference on Systems Documentation,* (Ottawa, Ontario; October 13-16, 1992), pages 211–219, October 1992. ACM Order Number 613920.

[32] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987.