

# Automatic Transformation of linear Text into Hypertext

F. Sarre

Institut für Informatik  
Technische Universität München  
Postfach 20 24 20, D-8000 München 2, FRG

U. Güntzer

Wilhelm-Schickard-Institut  
Universität Tübingen  
Sand 13, D-7400 Tübingen, FRG

## Abstract

Technical products are often shipped with voluminous documentations. As to complex software products, machine-readable manuals are provided in most cases in addition to the printed version of a documentation. Although electronically stored text has many advantages in contrast to the traditional medium "paper", the question arises how to manage the text of an online documentation. If the fact is taken into consideration that a documentation contains a great number of cross references that make the user aware of the interdependencies in the text, the application of a hypertext system seems suitable for making the knowledge included in the documentation accessible. Hypertext systems manage text in a non-linear fashion, i.e. in pieces of knowledge interconnected by logical links. However, an enormous manual and intellectual effort is necessary so far to convert already existing texts for a hypertext system. These considerations led to the development of a new hypertext system called "HyperMan". It has — in addition to the usual parts of such a system — a special component that is able to recognize dependencies and relations in a given linear text (or in a collection of linear texts of the same field), thus making the given material useful for a hypertext system. In this paper, we describe how the special component works. Details of the node generation process are explained and a link generating method based on information retrieval techniques will be proposed.

## 1 Introduction

When an author writes a technical text, he tries to put down his knowledge about a complex field. Although modern text processing systems facilitate this work, he has to deal with the problem of how he should arrange his ideas and thoughts in a certain fixed order. With the exception of figures, formulas and tables the result of his work is a one-dimensional, sequential presentation of his expert knowledge in the format of a long character string. Even if an author incorporates cross reference links, a reader of his document has to reconstruct many interdependencies between related terms and related pieces of text.

For this reason, a mainly linear representation of knowledge or a simple sequential ordering of facts might not be adequate for a reader. Why should not a system for managing the text of a technical documentation take advantage of the text's (hidden) structure?

In order to transform a linear text into hypertext, hypertext nodes and hypertext links have to be generated. Of course, the original linear representation of the text can be reconstructed at any time later on. In the HyperMan system, the result of the generation process, a hypertext structure, is stored in a relational database (see fig. 1). Afterwards, the user browses through the hypertext structure by means of a graphical interface [Hahn89]. In this paper, we do not want to consider HyperMan's browser. Instead, we want to focus on the generation of nodes and links. Although HyperMan is able to process any text, the text of a (technical) documentation is preferred, because it already has a certain structure that can be utilized to generate additional links.

## 2 Generation of Hypertext Nodes

As to the generation of hypertext nodes, a linear text has to be broken up in logically coherent information units. Afterwards, each information unit can constitute a hypertext node. Alter-

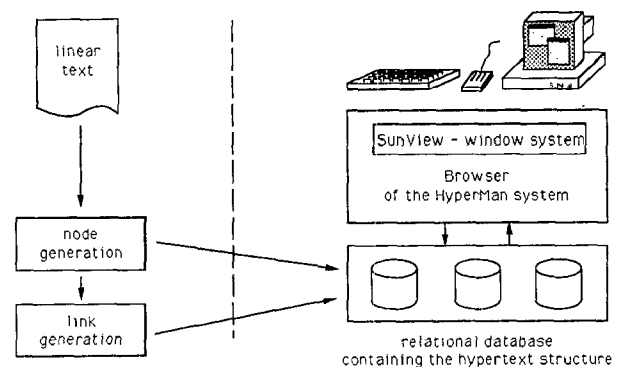


Figure 1: Architecture of the HyperMan system

natively, *several* of these information units can be assembled to form a node. According to our experience, a *static* node generation is sufficient for most applications, i.e. the node generation takes place only once and nodes are not later changed.<sup>1</sup>

An essential question for the decomposition of a linear text is, how the given text is organized from a logical point of view. Documentations, which do not serve as reference books exclusively, but which are also created for sequential reading, often possess a hierarchical structure mainly formed by chapters and sections. Sections themselves can, of course, consist of smaller units. In order to be able to process a linear text, the node generation module needs to know details about the text's structure. As a simple example, we will consider the decomposition of the text shown in figure 2. The text is taken from [Tran90].

First, the text is processed by a scanner. The scanner distinguishes merely between text portions that can be used to identify the (logical) structure (like chapter and section headlines) and other text. During this analyzing process, the scanner simulates

<sup>1</sup>If a user wants to integrate the information units in another way according to his needs, he can achieve this by means of "private component links" (not yet implemented in our system). However, the nodes created by the system are preserved any way.

|       |                                 |
|-------|---------------------------------|
| 1.    | Introduction                    |
|       | _____                           |
|       | _____                           |
|       | _____                           |
| 2.    | General Concepts                |
| 2.1   | Conventions for Syntax Notation |
|       | _____                           |
|       | _____                           |
| 2.2   | Data Type                       |
|       | _____                           |
|       | _____                           |
| :     |                                 |
| 2.5   | Literal                         |
|       | _____                           |
|       | _____                           |
| 2.5.1 | Integer Literal                 |
|       | _____                           |
|       | _____                           |
| 2.5.2 | Numeric Literal                 |
|       | _____                           |
|       | _____                           |
| :     |                                 |

Figure 2: Example of a coherent, linear text (hierarchically organized)

a finite automaton described by regular expressions. Expressed in the syntax of the scanner generator "lex", these regular expressions look like the following:

```

dig      [0-9]
p        "."
blk      " "
other    ["\n"]
nl       "\n"

~{dig}+{p}{blk}+{other}+{nl}
    {... return(CHAPTER_HEADLINE)};
~{dig}+{p}{dig}+{blk}+{other}+{nl}
    {... return(SECTION_HEADLINE)};
~{dig}+{p}{dig}+{p}{dig}+{blk}+{other}+{nl}
    {... return(SUBSECTION_HEADLINE)};
~{other}+{nl}
    {... return(LINE)};

```

Every time, when a regular expression is recognized in the input, a certain token is passed onto a parser (as an illustration see fig. 3). This parser processes the tokens delivered by the scanner by means of rules of a context-free grammar. These rules serve as a formal description of the document's structure. At the same time, procedures (actions) are executed that create nodes as well as links which represent the actual structure found. It is important to note that it is comparatively easy to adapt to other structures of other document types, because both the scanner and the parser are automatically generated from formal descriptions.

The structure description for the sample document (fig. 2) can be written down as (the actions to be performed are omitted for reasons of clearness):

```

%token CHAPTER_HEADLINE, SECTION_HEADLINE, SUBSECTION_HEADLINE,
      LINE

%%
document      : chapter_sequence
               | text chapter_sequence
               ;
chapter_sequence :
               | chapter_sequence chapter
               ;
chapter        : CHAPTER_HEADLINE chapter_content
               ;
chapter_content : text section_sequence
                 | section_sequence
                 ;
section_sequence :
                 | section_sequence section
                 ;
section         : SECTION_HEADLINE section_content
               ;
section_content : text subsection_sequence
                 | subsection_sequence
                 ;
subsection_sequence :
                 | subsection_sequence subsection
                 ;
subsection       : SUBSECTION_HEADLINE subsection_content
               ;
subsection_content : text
               ;
text              : LINE
                 | text LINE
                 ;

```

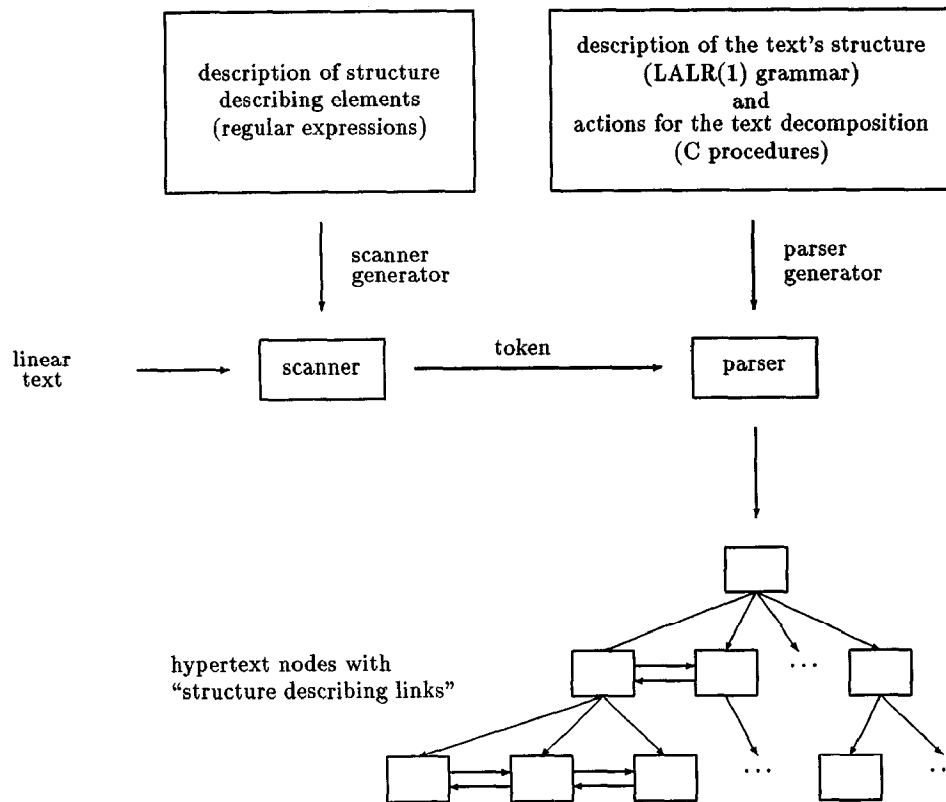


Figure 3: Overview over the node generation process based on a document structure description

Based on this description, hypertext nodes are formed by chapters, sections and subsections. More precisely, only units of the lowest level (subsections in the example above) are *directly* transformed into hypertext nodes by copying their full text into the nodes, while the text of subsections in sections or sections in chapters is excluded from nodes and can only be reached by *hierarchical links*. Through this procedure redundant text is avoided in the nodes. As a consequence, there will be small tables of contents with a depth of 1 in the nodes which contain a chapter or a section with substructures (see fig. 4, node (2.) and (2.5)).

In the implementation of the HyperMan system we have decided to represent “structure-describing” links by boxes around text. Links to subsections are represented by indented headlines, while the sequential links, i.e. links to the predecessor section and to the successor section of the same level (if there is one, otherwise to the section one level higher), are placed at the beginning or at the end of a node, respectively. An important feature of the sequential links is that they can help to reconstruct the original, linear sequence of the text, if necessary.

### 3 Generation of Hypertext Links

Hypertext links must be generated automatically in addition to hypertext nodes, if a linear text has to be transformed into hypertext. Hence, methods are presented in this chapter that show how the structure and the content of linear text can be utilized to create (additional) hypertext links.<sup>2</sup> We will focus particularly on the automatic generation of links that express a relationship between two pieces of text. First of all, the concept “link” informally used so far will be defined.

#### 3.1 Link Model

In the HyperMan system a link is defined as a reference from one text unit to another text unit.<sup>3</sup> The two text units may originate from two different nodes as well as from a single node. A text unit can be any coherent piece of text of a node (even the node itself).

The piece of text a link originates from will be called “source”, while the one a link points at will be designated “destination”.

<sup>2</sup>Remember that “structure-describing” links have already been created during the node generation process.

<sup>3</sup>Note that a link is defined here as a *unidirectional* link. This has proven to be useful and reasonable, because an attribute of a link can clearly express in what way two pieces of text are related.

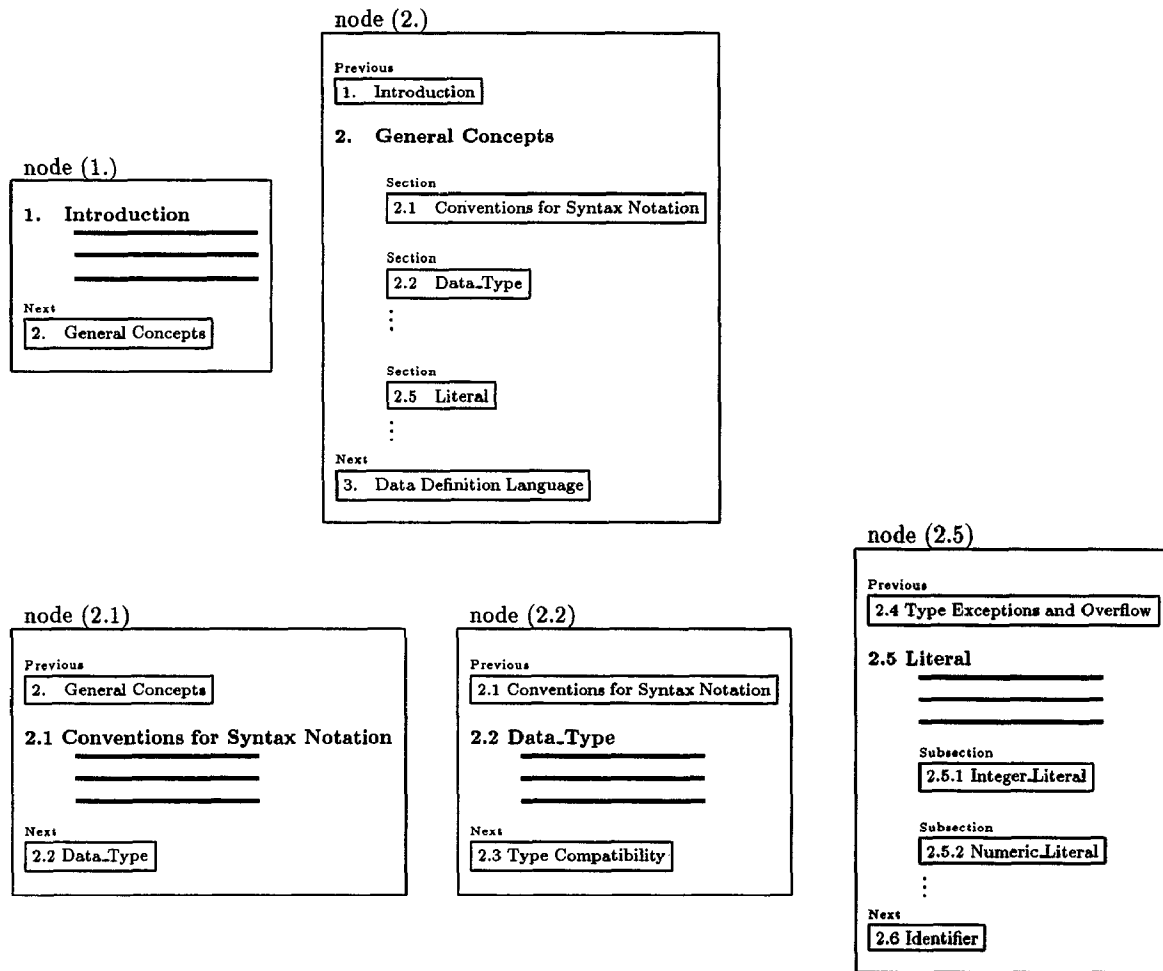


Figure 4: Generated nodes from a linear document (“structure-describing” links are boxed)

In this model, link sources will never exceed a node’s boundaries, as well as link destinations.

### 3.2 A Rough Classification of Link Generating Methods

Basically, links can be generated

1. during the node generation phase (statically),
2. after completion of the node generation process (statically),
3. during user sessions (dynamically).

It is reasonable to create only those links *during the node generation phase* that are closely related to the node generation process. These links are “structure-describing” links which reflect the original sequential and hierarchical structure of the given text. Because of the methods proposed in the previous chapter, they can be considered as a “by-product” of the node generation.

*After the node generation* has been completed, cross references explicitly occurring in the text can be made utilizable in the electronic hypertext environment. In addition to this, special link generating methods can create new links that express relationships between pieces of text.

*During user sessions* both new links can be learned and machine-generated, possibly uncertain links can be confirmed or rejected, respectively. In this way, the system can *learn* about the quality of links during user sessions. However, this topic is a part of our future work (see chapter 4, “Summary and Outlook”) and will not be discussed here in detail.

### 3.3 Static Link Generation after Completion of the Node Generation Phase

The fact that there are many different link types to be created for a hypertext document suggests that we try to classify them by formal aspects. Then, links of one category can be treated by a single method. In this context, it is helpful to distinguish between *explicit* and *implicit* link sources and link destinations.

A link *source* is called *explicit*, if an original text's text passage directly "invites" a reader to follow the corresponding reference, e.g. a footnote, a bibliographic reference or a reference starting with "see ...". Very often no natural text understanding is necessary to recognize this type of link source. In an analogous manner, an *explicit* link *destination* can be recognized by simple syntactic methods. As an example consider a proof of a theorem in a book about logic that is referred to from another theorem. The fact that all proofs are explicitly marked as proofs in the same way makes it easy to recognize them by means of simple pattern matching algorithms.

In contrast to this we want to call link *sources* and *destinations* *implicit*, if a certain amount of (expert) knowledge is necessary to investigate (implicit) dependencies in the text.

By means of these definitions, any reference in a document can be associated with one of the four categories in the following table:

| link destination | explicit | implicit |
|------------------|----------|----------|
| link source      |          |          |
| explicit         | I        | II       |
| implicit         | III      | IV       |

References of category I obviously *exist* in an original document's text, but cannot be followed very quickly there. By means of methods implemented in the HyperMan system only the addresses of the sources and the destinations will have to be determined in order to make these links utilizable in the electronic hypertext environment. A user is then able to follow a link by a simple mouse click. In this respect, the ease of following a link increases. This type of link generation could also be called "link materialization", because new links are created based on *existing* cross references in a given linear text. Although this scheme seems to be quite simple, materialized links are of a great help, especially in online-documentations.

In the example

... see below

an explicit link source and an implicit link destination (category II) is present, because this piece of text invites the reader to follow the reference. The exact destination of the reference cannot be easily determined by a computer. A possible solution might be to apply the methods presented in section 3.4 of this paper. Category II links require, however, further research.

As an example for category III we consider the explanation for the term "identifier", that is explicitly marked as an explanation:

#### Explanation 2.17 (identifier)

An identifier is a sequence of letters (A-Z, a-z) and digits (0-9), where the first character is a letter.

In this case an explicit link destination is present, whereas every other occurrence of the word "identifier" can be considered as an implicit link source. A reader who does not know the exact meaning of the technical concept "identifier", *may* look up the corresponding explanation, but is not invited to do so. In the HyperMan system category III links can be automatically generated.

Implicit dependencies in the text that result from counter arguments, generalizations, restrictions, comprehensive explanations and the like and that are not explicitly marked as such in the text, can be identified with category IV. In order to get on well without complex linguistic methods, we developed a method based on information retrieval techniques that generates "relationship links" between related text passages. The algorithm will be presented in the following.

### 3.4 Generation of "Relationship Links"

This section copes with the generation of relationship links which make the user aware of a relation between two pieces of text. These links can be particularly helpful for a user, if the displayed information does not provide a sufficient answer to his questions and if he requires additional, relevant material displayed for this reason. Although he could alternatively submit a search query, the possibility of following a relationship link is a much more efficient procedure (just a mouse click...!) and more convenient: The user does not have to learn a retrieval language and does not have to change his environment from the hypertext browser to a special information retrieval component.

To solve the task of generating relationship links, we look for the most similar pieces of text. In order to be able to make any statements about the similarity of two pieces of text, it is necessary to use an algorithm that produces a numerical similarity value. This algorithm does not have to "understand" the content of the two pieces of text, because useful results can be achieved by applying well developed information retrieval techniques.

If single words and noun phrases<sup>4</sup> of two pieces of text are considered as operational units, three preprocessing steps make a comparison much more reliable, independent of the matter, how the comparison is finally executed:

1. Filtering by means of a negative list
2. Term reduction
3. Weighting of the remaining terms.

In the following three sections we will discuss how these preprocessing steps can look in detail. It is clear that methods of text filtering, term reduction, application of thesauri and term weighting have been applied in information retrieval environments for the automatic indexing process for at least 25 years.

<sup>4</sup>Noun phrases are automatically detected in our system. We use an algorithm based on word frequencies and linguistic heuristics. Details can be found in [Sarr91].

It is new, however, to use these and modified techniques for the automatic generation of hypertext links.<sup>5</sup> The main reason why our approach leads to useful results is, that we are only trying to find related pieces of text within a restricted context, because the text passages under consideration always originate from the same documentation or document.

### 3.4.1 Negative List

Every piece of text written in natural language contains not only words useful to describe the content but also insignificant words (negative words). Some very frequent insignificant words in the English language are listed in table 1.

In order to avoid a deceiving high similarity value, when two pieces of text are compared, it is absolutely necessary to remove negative words. A nice side effect of this removal is that the total processing time is decreased.

A decision has to be made whether the general negative list like the one shown in table 1 should be *adapted* to the text under consideration. Because we will later be using a term weighting procedure, which assigns low weights to very frequent words, a general negative list presumably will do the job.

<sup>5</sup>See also [SaBu89].

|          |         |           |          |
|----------|---------|-----------|----------|
| a        | do      | must      | there    |
| able     | does    | never     | these    |
| about    | down    | next      | they     |
| above    | each    | no        | this     |
| after    | either  | none      | those    |
| again    | etc     | not       | three    |
| all      | even    | of        | through  |
| almost   | every   | off       | to       |
| alone    | except  | often     | together |
| along    | few     | on        | too      |
| already  | first   | once      | two      |
| also     | for     | one       | under    |
| although | four    | only      | unless   |
| always   | from    | onto      | until    |
| an       | had     | or        | up       |
| and      | has     | other     | very     |
| another  | have    | others    | was      |
| any      | here    | otherwise | well     |
| are      | how     | out       | were     |
| around   | however | over      | what     |
| as       | if      | rather    | when     |
| at       | in      | right     | where    |
| be       | instead | same      | whereas  |
| because  | into    | second    | which    |
| been     | is      | should    | while    |
| before   | it      | since     | whose    |
| being    | its     | so        | why      |
| below    | just    | some      | will     |
| between  | last    | sometimes | with     |
| both     | left    | still     | within   |
| but      | like    | such      | without  |
| by       | many    | than      | would    |
| can      | may     | that      | yes      |
| cannot   | might   | the       | you      |
| case     | more    | their     | your     |
| could    | most    | them      |          |
| did      | much    | then      |          |

Table 1: Frequent words in the English language

### 3.4.2 Term Reduction

After a negative list has been applied, single words and multi-word noun phrases of the same stem have to be reduced to a common word form. It is not necessary for link generation purposes to reduce terms to their *exact* linguistic word stems, because the reduced terms are an intermediate result not shown to the user. Since the basic aim of the term reduction process is to identify inflected forms belonging to one another, the common word form must not be the correct linguistic word stem, but could be any unique string or even a unique number.

In the HyperMan system, we implemented a simple term reduction method, which produces useful word forms in more than 98 % of the cases. Exceptions not treated correctly by the algorithm do not matter much, first because they occur too rarely in a text to have an influence on the links generated later on, and second because the links generated can be assessed by users later on and can be removed from the hypertext base, if they are not valid.

Our method repeatedly applies the rules shown in table 2 to a given word (or a multi-word phrase). The algorithm stops, if no rule can be applied or if the word form doesn't change any more. A minimum word length is necessary to avoid overstemming.

### 3.4.3 Automatic Weighting of Terms

Because the terms of a text have different relevance with regard to their ability to describe the content of a text, it is necessary to assign weights. Methods of term weighting are usually based on term frequency observations [Salt83].

A term with a high weight should

1. be closely related to the text it originates from;
2. distinguish text fragments to which it is assigned from text fragments to which it is not assigned.

As a consequence, a "good" term should occur comparatively frequently in a small number of text fragments and should hardly occur in the rest of the texts.

In the HyperMan system we use a term weighting formula that considers the exact frequency distribution of a term in the text passage collection at hand. It is based on information theory deliberations, which will be published elsewhere:

$$W_{ki} = f_{ki} \cdot (\log_2 N - \sum_{i=1}^N \frac{f_{ki}}{F_k} \log_2 \frac{F_k}{f_{ki}})$$

with

- $W_{ki}$ : Weight of term  $T_k$  in text passage  $D_i$   
 $f_{ki}$ : Occurrence frequency of term  $T_k$  in  $D_i$   
 $N$ : Total number of text passages at hand  
 $F_k$ : Total occurrence frequency of term  $T_k$

| minimum word length | ending  | substitute | minimum word length | ending | substitute | minimum word length | ending | substitute |
|---------------------|---------|------------|---------------------|--------|------------|---------------------|--------|------------|
| 10                  | ability |            | 7                   | ibl    |            | 5                   | nt     |            |
| 7                   | abl     |            | 5                   | ic     |            | 5                   | oken   | eak        |
| 8                   | ably    |            | 5                   | ied    | y          | 5                   | or     |            |
| 6                   | al      |            | 5                   | ier    | y          | 5                   | ound   | ind        |
| 7                   | anc     |            | 5                   | ies    | y          | 6                   | ous    |            |
| 7                   | ant     |            | 5                   | iest   |            | 7                   | ply    | pl         |
| 3                   | ant     | anc        | 5                   | id     | is         | 7                   | pp     | p          |
| 7                   | at      |            | 8                   | if     |            | 7                   | rr     | r          |
| 10                  | ation   |            | 5                   | ily    | y          | 4                   | s      |            |
| 10                  | ativ    |            | 6                   | ing    |            | 5                   | sen    | ose        |
| 7                   | bly     | bl         | 6                   | ion    |            | 7                   | siz    | siz        |
| 5                   | ctic    | x          | 5                   | ipt    | ib         | 2                   | ss     | ss         |
| 5                   | cy      | t          | 6                   | is     |            | 5                   | str    | st         |
| 4                   | dd      | d          | 5                   | ison   |            | 5                   | ta     |            |
| 5                   | de      | ke         | 9                   | ition  |            | 5                   | ten    | e          |
| 4                   | e       |            | 6                   | ity    |            | 5                   | tr     |            |
| 5                   | ed      |            | 5                   | iv     |            | 2                   | tt     | t          |
| 7                   | enc     |            | 5                   | ix     |            | 6                   | ual    |            |
| 7                   | ent     |            | 6                   | iz     |            | 5                   | uent   | ut         |
| 5                   | ent     | enc        | 6                   | ly     |            | 8                   | usiv   | ud         |
| 7                   | er      |            | 5                   | ment   |            | 4                   | y      |            |
| 7                   | est     |            | 7                   | mm     | m          | 6                   | yz     | ys         |
| 5                   | ful     |            | 5                   | nat    |            |                     |        |            |
| 7                   | gg      | g          | 5                   | ness   |            |                     |        |            |
| 10                  | ibility |            | 7                   | nn     | n          |                     |        |            |

Table 2: Word stemming rules of a simple word reduction algorithm

Note that the first factor is exactly the term frequency in the text passage considered, while the second factor in brackets denotes the term specificity.

#### 3.4.4 Similarity of Text Passages

The three preprocessing steps described so far are used to gain a characteristic set of (weighted) terms for each text passage  $D_i$ . Now, all the sets can be compared in twos by a similarity function. A typical similarity function that works well is the normalizing cosine formula:

$$Sim(D_p, D_q) = \frac{\vec{x}_p \circ \vec{x}_q}{|\vec{x}_p| \cdot |\vec{x}_q|} = \frac{\sum_{k=1}^T (W_{kp} \cdot W_{kq})}{\sqrt{\sum_{k=1}^T W_{kp}^2} \cdot \sqrt{\sum_{k=1}^T W_{kq}^2}}$$

By means of *Sim*, a text passage / text passage similarity matrix can be computed.

#### 3.4.5 Links to Related Text Passages

Proceeding from the similarity values computed, “relationship links” can be established in the next step. To achieve this, the  $N - 1$  similarity values of every  $D_j$  ( $j = 1 \dots N$ ) are sorted in descending order. Then, the question arises how to reduce the possible  $N - 1$  links for each text passage to a reasonable (small) number. As the choice of a similarity threshold strongly depends

on the user community, we suggest to use the 5 best similarity scores for each piece of text to create the links. Therefore, a similarity threshold is not needed. Five relationship links are sufficient, because the user may not want to follow more than 5 alternative links in most cases.

In order to indicate, how “good” the links presumably are, marks from 1 – 5 are calculated based on the similarity scores. The links corresponding to the best 20 % scores are marked by a “1”, the next 20 % by a “2” and so on (see fig. 5).

### 3.5 Experimental Results

As an experiment relationship links (as well as other links of links) have been generated for a database documentation. The paper version of the text sample consists of 554 pages of text. The given text was divided up into 284 nodes by the node generation component described in chapter 2 of this paper. However, for the generation of relationship links we didn’t use these nodes, because they consist in their turn sometimes of several hundreds of lines of text. Instead, we used single text paragraphs as text units (total 2204).

A text paragraph seems to have the ideal size, because an idea is usually expressed in a paragraph and very rarely, for example, in a sentence. A complete node seems to be too large, because it might cope with various topics. The generation of relationship links might make no sense in this case.

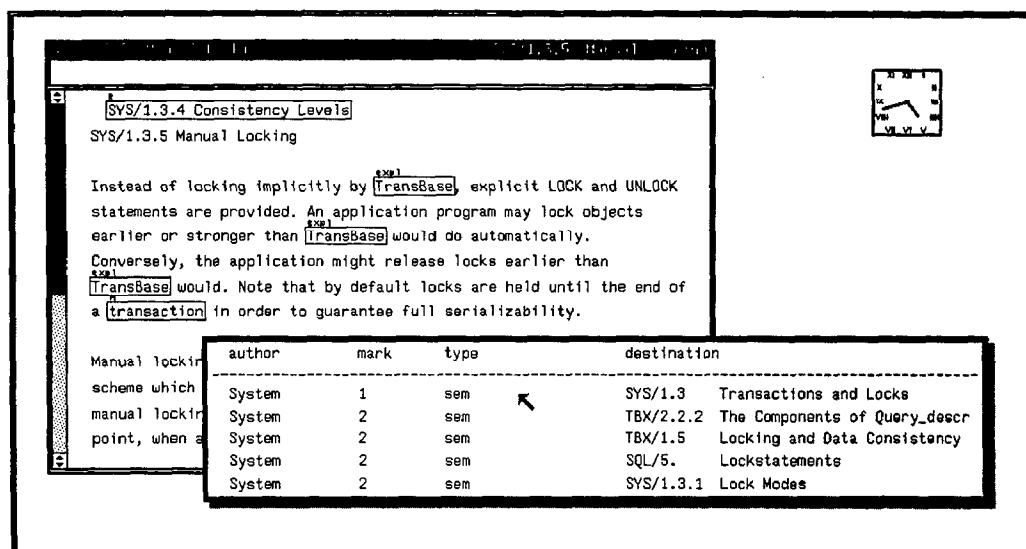


Figure 5: Automatically generated "relationship links"

At the present time, we evaluate the quality of the machine generated links. Especially, we want to learn whether there is a correlation between the quality of the links established and the similarity scores. For this purpose, we randomly chose 20 different text paragraphs. As a consequence, we have to look at 100 links total. Every single link will be carefully examined and will be assigned to one of the four categories:

☐ very relevant   ☐ relevant   ☐ not helpful   ☐ wrong

We will judge the proposed scheme useful, if more than 50 % of the generated links are very relevant or at least relevant.

### 3.5.1 Shortcomings of the Method Proposed

Although the first results obtained are encouraging, we clearly see some shortcomings, which have to be overcome in the future:

1. Thesaurus relations except inflected forms are not incorporated in the algorithm so far. However, it is not difficult to incorporate a thesaurus into the system, if it is available (e.g. for the computation of similarity values or for the enrichment of a user's search queries). We plan to generalize the word reduction algorithm.
2. The complexity of the scheme proposed is  $O(N^2)$  ( $N$  = number of pieces of text). Although this is not critical for a documentation of several hundred pages, for several thousand pages, it would be. For this reason, we are working on alternative algorithms of  $O(N)$ . A feasible method might be to determine the three best terms of every piece of text. Then, a query searches for text units containing at least two of these three terms to compute the similarity values in twos and to establish relationship links.

3. The method proposed does not take into consideration whether a term occurs in a headline, in a definition or in regular text. This can be remedied by assigning a higher weight to terms occurring in headlines, for example.

## 4 Summary and Outlook

Hypertext systems are useful tools for managing textual data. In many cases, it may be desirable to manage existing (linear) documents. However, a great manual and intellectual effort is necessary so far to create hypertext nodes and hypertext links, if the hypertext structure is not generated automatically. This has led to the development of a hypertext system designated "HyperMan" which provides tools for creating the desired hypertext structure very quickly. The node generation is based on a formal description of the linear text to be converted. With regard to the generation of links two main categories of links have to be distinguished: explicit (visible) and implicit (hidden) links. While explicit links can be made useful in the electronic hypertext by calculating the links' physical addresses, implicit links can be established by finding related pieces of text. One possible method of doing this has been proposed in this paper.

In the future users of hypertext systems should be observed in order to gain additional information about possible links. If, for example, a user submits a search query and if he asks for the results offered (e.g. for printing, saving or editing), there is obviously a relationship between the selected nodes. This relationship could even be established as a "hyperlink", i.e. a link connecting not only two, but many nodes. Moreover, the system could learn about the quality of links: If a user gets a destination node displayed for a very short time, if he does not proceed from this node and if he does not copy the node into his interest list (a feature available in the HyperMan system to

collect relevant material while browsing the hypertext structure), the link followed is obviously not relevant — at least for this user.

### Acknowledgements

The authors would like to thank Maximilian Fichtl, who implemented the algorithms described in this paper [Fich90]. This work is a part of the project “Objektbanken für Experten” and is supported by the “Deutsche Forschungsgemeinschaft” (DFG).

### References

- [Conk87] Conklin, J.: *Hypertext: An Introduction and Survey*. COMPUTER, Sept. 1987, p. 17–41.
- [Fich90] Fichtl, M.: *Knoten- und Linkgenerierungstechniken in den Hypertextsystemen “HyperMan” und “HyperTeX”*. Diploma Thesis, Technische Universität München, FRG, July 1990.
- [Hahn89] Hahn, A.: *Eine hypertext-ähnliche Oberfläche für die UNIX-Manuale als Beispiel einer kooperativen Benutzerschnittstelle*. Diploma Thesis, Technische Universität München, FRG, Nov. 1989.
- [Salt83] Salton, G.; McGill, M.: *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983, 448p.
- [SaBu89] Salton, G.; Buckley, C.: *On the Automatic Generation of Content Links in Hypertext*. Technical Report TR 89-993, Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, April 1989, 14 p.
- [Sarr89] Sarre, F.; Güntzer, U.: *Einsatz von Hypertextsystemen für Dokumentationen technischer Systeme*. Proceedings of the GI-Fachtagung “Interaktive Schnittstellen für Informationssysteme”, S. 133–148, Clausthal-Zellerfeld / FRG, Nov. 1989.
- [Sarr90] Sarre, F.; Güntzer, U.: *Einsatz des Hypertextsystems HyperMan für Online-Datenbankmanuale*. In: Proceedings of the “Hypertext/Hypermedia’90”-Workshop Darmstadt / FRG (April 23 – 24, 1990), pp. 112 – 123, Springer 1990.
- [Sarr91] Sarre, F.: *Zur Generierung von Hypertextstrukturen aus linearen Texten durch Partitionierung und Strukturanreicherung*. Unpublished Paper, Technische Universität München, FRG, 1991.
- [Tran90] *TransBase Relational Database System, Version 3.3* Programming Interface, System Guide, TB/SQL Reference Manual, TBI Manual, UFI User’s Guide. TransAction Software GmbH, D-8000 München, FRG, 1990.