Joint NCC & IIS Bull., Comp. Science, 12 (1999), 37–41

Hypertext environment for software and documents development

S.B. Pokrovsky, G.G. Stepanov

Introduction

A hypertext system here presented supports some features essential for software and documents development environment; in particular, versioning (both logical and chronological), macro facilities, authoring facilities and interface with text representation systems (HTML, TEX).

Basically, hypertext is a graph whose nodes are data *segments* and whose links are various references among these segments. A node is typed according to the species of data it contains; e. g., prose, source code, executable code. The references are usually classified into two general kinds: the *cross-references* and the *structural links*.

The structural links serve to traverse the hypertext, more or less completely and/or systematically; e. g., to flatten a structured program into a linear text. This implies some limitations on the subgraph formed by the structural links (e. g., that it be acyclic and preferably a tree).

No restriction is put on the cross-references.

The hypertext approach has much in common with many of the more traditional concepts: DBMS, windowing systems, menu systems, structured editing, attribute grammars, object-oriented programming. Logically hypertext is a kind of DBMS with specialized (explicit) relations and efficient front-end interface.

Hypertext is a natural environment for software project development. The hypertext of a project may contain segments from various levels of specification (free prose, block diagrams, text in a formal language), documentation, source programs for individual components of the system designed. The structural links (eventually they may be typed) enable one to produce a linear text, or to get a hierarchical bird's-eye view of the project; the cross-references make possible immediate consulting on the related pieces at any specification level (in particular, within one and the same level).

To meet the needs of a software designer, the hypertext system must provide a fair amount of editing facilities and interfaces with various tools; the ability to support multiple versions, which occasionally is featured in hypertext systems, is certainly welcome in a software development environment.

1. Structure of Hypertext

1.1. Segments

Formally, a segment may be defined as the user's data combined with a name and qualified with a segment type and a version number.

The segment name is an arbitrary string; it is supplied by the user or derived by a programming tool. It is used to visualize references to (other) segments or to reference a segment from the keyboard (e. g., when the desired reference lacks on the screen). A segment name must be unique within the hypertext.

The segment type is essential to determine the set of facilities applicable to the segment and the manner in which they treat the segment (e. g., selection, visualization, execution, storing). The most useful segment types are prose, source code, template, binary.

A prose segment may contain a part of a document. Normally it is structured in (sub)sections — paragraphs — lines and is treated by a word processor; less adequately prose representation may be used for some latently more formal texts (e. g., source modules), which implies the use of a parser for structured editing or compilation.

A source code segment implies treatment similar to that of the traditional line-oriented program editors; actually it is very close to the prose type and differs primarily in formatting procedures.

A template segment is a segment with a set of parameters. Each structural link to such a segment must specify actual values of segment parameters. A template segment looks like a macro-definition, structural link to it looks like a macro-call.

The binary segment type is the most general type; consequently, the set of built-in facilities to handle binary segments is minimal.

1.2. Portions and Marks

A segment of any type is a tree of *portions*. There are three basic portion kinds:

a) User's data (e. g., a piece of text which contains no control information, i. e. no references);

b) Cross-reference or structural link;

c) Nest, i. e. a list of portions which is desirable to be treated as a whole (as a single portion).

Portions may be used as arguments for transformation operators (e. g., one can move or delete a portion). Under certain conditions it is possible to split or merge portions of the first kind; thus they are similar to the "marked area" concept of many text editors.

For any hypertext there may be defined a finite set of simple marks. A simple mark has the internal representation (its serial number) and it may be given an external representation (any character string).

Marks are used to label portions. If there are several simple marks labelling a portion, they are said to form a compound mark. Normally marks are visualized in a slightly different manner than the content of a text portion, and their presentation cannot be changed by the common text editing means.

There are various applications of the mark feature. In a syntax-oriented editor it is natural to express syntactic notions as marks; as the external representation for the latter one might choose keywords of the language. Another major application of marks is related to selective straightening (see 2.3).

Actually a mark may consist of two parts: a *prefix* which is put before the labelled portion, and a *suffix* which is appended after it. Major applications of this feature are related to the possibility to associate subroutines with any mark. Such associations may be grouped in a "configuration segment"; the user may dynamically control this association, switching from one configuration to another. E.g. there is a set of bracketing marks for font representation; it is normal to use a screen control association for the interactive mode (and hence to present fonts with various colors); but a print command would switch the configuration segment (and hence the control sequences) according to the type of the printer used.

1.3. Versions

A chronological version is defined by its descriptor which contains the version number, the version's creation date, a reference to the previous version, the tables of segment names.

A new version may emerge only if some hypertext segments get changed (e. g., in the course of editing); the changed segments reside in the main store until a *Write* (or *Quit*) instruction is issued. In the absence of changed segments the *Write* instruction is void; if there are such segments, a new version is created; it consists of all the unchanged segments from the previous version, combined with the fresh copies of the changed segments, a new segment table and a new version descriptor. No data from the previous version is lost: the changed data (including the tables) is simply added to the hypertext; on the other hand, no unchanged segment is duplicated (actually also the information in the tables is segmented and consequently shared).

It is possible to access any of the previous versions, either as a whole, or by inspection of a particular segment (provided that some version of the segment already existed in the considered version of the hypertext). This facility also provides a means to cancel undesirable changes done in a particular segment: it suffices to reread the segment from the latest version; the rereading clears the changed flag for the segment in question (while retaining its value for the other changed segments).

It is possible to extract any particular version of a hypertext and to make of it a separate hypertext. This is a regular means to get rid of the superfluous versions.

1.4. Distributed Hypertexts

A hypertext may be spread over several files and, eventually, volumes. This is done through a special kind of reference: external links. An external link is a string which completely identifies an OS file (it is supposed to be a hypertext fragment) and the name of a segment contained within the file; unless a precise version number is specified, the latest version is implied.

Presently the whole responsibility for the consistency of the external links is left to the user.

2. Software Tools

In this section we consider the impact of the hypertext approach on the software tools designed to process hypertexts.

2.1. Text Editing

A screen-oriented hypertext editor is the central part of the system. A fair repertoire of the traditional text editing facilities is available within the text portions; but the special information can be treated by the special instructions only (e. g., change or rename a reference, pass along it, change a mark; changing the special information by a direct text editing facility is inhibited by the system. Text editing may also be inhibited (even within a text portion) if the user lacks the required access rights or if the segment has a special type, so that for some hypertexts or segments the text editor acts rather like a viewer.

There is a considerable set of options to control the editing modes and visualization. Among these the straightening option and filtering are essential for several other hypertext tools.

2.2. Straightening

Normally the user can inspect a segment very much like a line-structured text file; the cross-references and structural links are visualized by their external representation and occupy just as much of the screen space as is required for that representation (if the user chooses to follow one of such references, the whole working segment is changed; otherwise the reference representation is passed by just like a single character). This corresponds to the non-straightening mode, when the hypertext is considered as a graph of segments.

By turning the straightening mode on, the user makes a part of this graph into a linear text. Basically this part may be defined as a subtree (if the structural links form a tree) whose root is the current working segment. The structural links in the whole subtree are replaced with the copies of the segments they refer to, and the user is able to scan the resulting linear text without paying any attention to the segment boundaries.

The straightened text may be read as a character stream by any tool program (e. g., by a compiler). The cross-references appear in the stream in the form of their external representation and can be scanned character by character; on the other hand, the user still can use them to get to the referred segment in the normal way.

2.3. Filtering

Filter is a Boolean formula which involves marks defined for the considered hypertext, the operators negation, conjunction, disjunction and brackets. To compute the value of the filter over a portion, one has to substitute *true* for all occurrences of the marks which label the portion, and *false* for those absent from it.

By defining a filter the user can restrict the set of portions read and/or visualized by the editor: namely, a portion is processed if and only if the filter is true for it.

Filtering is applied independently from the straightening mode. In particular, if these facilities are combined, we speak of *selective straightening*, which actually enables one to create projections, or *logical versions*, of an algorithm.

2.4. Macro Facilities

These include template segments, attributes (macro variables) and macro statements for conditional and iterative text generation.

An attribute is a variable of integer, logical or string type. There is a set of standard attributes, like page or line number, the current header, the date. Any other attribute is local to some segment, but it is accessible from outside if the "base reference" to the segment is supplied. Attributes are chained in portion lists; in particular, the formal parameters of a template (see 1.1) are attributes. The declaration (i. e. creation) of an attribute may involve restrictions on its admissible values and/or the default initialization. The value of an attribute may be changed in special computational portions, which are similar to SET-statements in a macro processor.

The basic control structures are nest, repeated nest and the break portion. These facilities influence the order of the straightening process, which is analogous to macrogeneration.

Any nest is a kind of parenthesis. Among its portions there may occur a break portion, a special kind of portion which contains a condition (an attribute expression). If in the course of the straightening process the condition is satisfied, then the rest of the enclosing nest is skipped. Thus a simple nest is equivalent to an IF-statement, and the repeated nest is similar to the LOOP-statement.

2.5. Authoring Facilities

Hypertext can be used to more or less automatically generate documents. They may involve various specifications, program texts or prose articles.

A document is represented by its root segment, which may involve structural links to its parts, whether structural (chapter, section) or not. In this way one and the same segment may be shared by many documents, and its correction provides correction of all the documents involved. Using straightening, marks, filtering and macro facilities one may create variants of a document (e. g., at various specification levels) in much the same way as with a piece of a program.

A specialized set of marks provides means for font control, automatic numbering of document sections and symbolic cross-reference (at the hard copy generation the pertinent annotation references are changed into numeric section numbers), etc. Special marks serve to make a fragment indicated by a structured link into a structural subunit (chapter, section); its hierarchical level (the quantity of periods in the section number) are derived from the nesting of the structural links. The scoping rules ensures that one and the same piece of text may be correctly imported at various hierarchical levels to various documents.

Using selective straightening, macro facilities and the set of authoring facilities marks, the system may output texts into HTML-format and into TEX-format. The system also may read texts from subsets of HTML- and TEX-formats, including headers, fonts, types, lists.

3. Conclusion

The hypertext approach enables the programmers to keep the logically related matters in a single object. This is true of the various specification levels and chronological versions (which is quite common) as well as of the logical variants of the software product. Like any other hypertext system, the presented system has some features of DBMS; it is more original that its formatting and menu specification means are developed to a full scale macro capability.