ON IMPLEMENTING A COST-EFFECTIVE HYPERTEXT SYSTEM

Khe-Sing The

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-38 September 1987

On Implementing a Cost-effective Hypertext System

KHE-SING THE
Dept. of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188

ABSTRACT. The current state of technology is still insufficient to support the ideal futuristic hypermedium. It can, however, support a hypertext for some specific applications, provided that the system is carefully designed to fit into the technological advantages and usability considerations. In this paper we first discuss the requirements for a simple and yet practically usable hypertext system, then we identify some feasible applications of the simple system in the current practices of software development. A feasible prototype is proposed as an example to illustrate the technical aspects of our hypertext specification.

KEY WORDS AND PHRASES: hypertext systems, software development tools, communication medium, non-linear documents.

1 Introduction

The recent advances in computer technology [8], in particular the availability of high resolution workstations, have saved hypertext from being implemented as a pile of index cards threaded with strings and stored in a shoe box. However, the current technology is not yet sufficient to support the ideal hypermedium [11]. Some good hypertext systems (see [2]) are available only on very expensive machines with sophisticated supporting software, in the meantime the most desirable feature of hypertext, namely the ability to access any node and to traverse any link "virtually instantly," is still an elusive dream to most hypertext users.

Here we propose a hypertext system that is simple enough to run on widely available machines and yet powerful enough to serve the needs of frequent activities in software development. While the hypertext concept itself is simple enough, we have to learn from experience to understand its optimal specification for a particular application. As the system evolves, the opinion of the end users is important to make the system usable.

Although many concepts of hypertext systems are proposed and developed to accomplish complicated tasks at the price of usability and response time, a hypertext system does not need to be very sophisticated to satisfy the average users. In Section 2, we give the requirements for a simple and yet useful hypertext system. Such a system can be utilized to perform many simple tasks in the course of accomplishing a complicated task. To demonstrate how the idea of simplicity can bring fruitful results, the feasible applications of hypertext as a universal medium in software development process are discussed in Section 3. A UNIX implementation of the proposed system is presented in Section 4 to illustrate the feasibility (and impossibility) aspects of designing a practically useful hypertext system.

2 The Requirements for a Simple Hypertext System

A hypertext network is a document in which textual and graphics information is logically organized as hypertext nodes, and the logical relation among the nodes is maintained by the links configuration. In his survey report on hypertext systems, Conklin [2] classifies hypertext systems into four categories of application: macro literary systems, problem exploration tools, browsing systems, and general hypertext technology. Our proposed system will best fit into the last category of application, it is a non-linear document editing facility designed especially for applications in software development technology.

As a system to manipulate interlinked documents, a good hypertext system should provide self-explanatory representation for each node and virtually instantaneous access to the links. Since it would be too costly to expect all the features to be present in a hypertext system implemented with the current hardware technology, we restrict ourselves to building a simple hypertext system. The features of this system are minimal to perform the tasks of assisting software developers. The only requirements for such a cost-effective system are stipulated as follows:

(i) The interface to manipulate new and existing nodes on the screen, i.e., the way we create, access, display, hide, and remove a node on the screen, is intelligible and fast. This statement includes the requirement of compatibility between the hypertext nodes and other information entities in the environment (e.g., files).

(ii) The links in a network can be conveniently and instantly traversed.

(iii) Links can be defined from any node to any node in the hypertext network (such that the system has the capability of representing non-linear text systems as defined by Shasha [9]).

(iv) The links are capable of storing (typically small amount of) information, and quick response can be expected in answering the queries about the properties of (the information augmented in) the links.

(v) A browsing utility to provide a clear picture about the position of the currently accessed node within the hypertext network configuration.

(vi) Operating system processes can be invoked on demand to perform some operation to the information contained in the nodes.

In the following sections we demonstrate how a hypertext system satisfying the set of minimum requirements as given above can help software developers in many aspects of their work, and then we propose a low-cost prototype system to support the claims.

3 A Universal Medium for Software Development Tools

Basic software development activities consists of program text editing, source code compilation, object code testing, documentation production, and design information communication. Although each of the tasks can be automated individually, the ultimate objective is the integration of the components in order to reduce the complexity of the user's interface to the system, to achieve compatibility among different tools serving different purposes, and to maintain the consistency of the tools.

Most of the industrial and academic research on automating the software design process is focused on building a special tool to solve some specific problem in software design and coding. Many of the tools are difficult to use, and the user has to learn painstakingly before the tool can offer any significant help.

In the following subsections, we identify some possible applications of general hypertext concepts as a universal medium in the approach towards finding better solutions for the above-mentioned problems.

3.1 Merging and Filtering of Specialists' Views

Hypertext technology can serve as a common medium supporting "specialists representing different viewpoints, working as a team." The following two strategies are the possible provinces of specialists [4] in performance, reliability, reusability, evolvability, and other specialized viewpoints:

1. Each of the specialists model the system individually and then their views are merged. In this strategy, the outcome of each specialist's work should be completely reflected and preserved in the resulting common view.
2. A common view is formed and the specialists' views are mapped from this common view. The model representing the common view encompasses all of the features of the different models used by different specialists.

In most modeling methods, the view is represented in some form of graphs (Petri Nets, finite state machines, Pert charts, etc.) The merging and filtering transformations of a modeling graph are generally accomplished by controlling the graph's degree of elaboration in reflecting the common view, i.e., how some parts of the graph are lumped to elaborate only the parts of interest. The criteria of the transformations are: (1) preserving correctness, (2) maintaining consistency, and (3) avoiding redundancy.

A good medium to support the transformations requires the capability of intelligently storing information in the nodes (representing views) and links (representing relation between views) of the modeling graphs. Those features are provided by our hypertext system.

3.2 Organizing Code and Documentation

The hypertext system as described in Section 2 can support a scheme which allows an artifact to develop both the design (in the sense of the explanation of the system) and its implementation (as code in some automatically processable language) simultaneously and intermixed, and both the code and the explanation can be extracted separately.

Based on his research experience in software engineering, Shen [10] infers that program documentation is a major burden of software developers working on government contracts. It is desirable to enhance the existing languages and tools in order to provide documentation meeting the government standards conveniently and consistently.

The sensible way to document a design is to do the documentation concurrently with the design, and each design module in each design level should come with consistent, clear, and concise documentation. The manageability of the design artifacts will be helpful in maintaining the consistency of the design even long after the system has been delivered.

The entire design artifact can be stored consistently and comprehensibly as a hypertext network, with different link types connecting code and documentation, summary and elaboration, and other relevant design issues. The idea of using a hypertext network of code and documentation to represent a software system has been proposed by Goldstein and Bobrow [5].

The hypertext system can also be used in cooperation with the existing design and coding tools to enhance their features, for example, the code and documentation produced by a hierarchical iconic design tool (e.g., [1]) can be captured in hypertext nodes (instead of regular files) to enable non-hierarchical icon access, and the links and display facility of the hypertext system have the side benefit of improving the user interface of code management tools (such as the make utility in UNIX).

3.3 Serving as an Interface for Object Oriented Programming Systems

Object oriented programming environment offers the advantage of working closer to the conceptual problem domain. An *object* is some private data and a collection of public procedures that can access that data. A *message expression* is a modularity mechanism which tells an object to select and then perform a procedure as requested by the message. Messaging, which is analogous to ordinary function call with one extra selection step to provide dynamic binding, causes the *encapsulation* of objects [3].

Each object has a private part and a shared part. The shared part describes what is similar about the instances belonging to the same *class* (comparable to the conventional notion of types), while the private part describes what is different. *Inheritance* is a mechanism for linking object concepts into higher level concepts. It simplifies the specification of objects and preserves the consistency of object specification [3]. Encapsulation and inheritance are the distinguishing characteristics of objects.

It is often difficult to implement and prototype application programs due to the lack of appropriate programming tools. Object oriented programming has shown a good prospect as a tool for rapid prototyping and implementation. The concepts of nodes and links in hypertext naturally correspond with the concepts of encapsulation and inheritance in object oriented systems; hence, hypertext would be a proper choice to serve as a user interface for object oriented systems.

3.4 Maintaining Library of Reusable Software Components

To exercise better control over the development of large and complicated software systems, there is a trend in software development methodologies towards reusability as applied in other engineering disciplines where a large system consists of a hierarchy of building blocks, and the designer's experience is captured in an artifact such that similar components should be reusable once invented.

In their recent paper, Katz, Richter and The [6] describe the PARIS system, which is an implementation of the concept for reusing "PARtially Interpreted Schemas." A *partially interpreted schema* is a program that has some of its portions remaining abstract or undefined. These abstract entities can be either program sections or non-program entities such as functions, domains, or variables. For different interpretations of abstract entities in the schema, the results will be different programs performing different functions.

The PARIS system maintains a library of partially interpreted schemas, each of which is stored along with assertions about its applicability and results. When a user presents a problem statement (i.e. description of requirements for a program), the system will search through the library for a possible candidate schema and replace the abstract entities of the suitable candidate by concrete entities. The search through the schema library can be recursively initiated; when it becomes necessary to interpret an abstract program section (a "hole" in the schema), the system will automatically consider the requirements of the abstract program section as a lower-level problem statement and recursively invokes the interpretation mechanism.

The philosophy adopted here is to be semi-interactive. This means that the user is to be protected as much as possible from the inner workings, representations, and decision procedures of the system; only when the automatic tools are inadequate the user is to be consulted in as clear a manner as possible. To add more convenience for the user to specify his/her problem statement and to provide more interactive communication during the process of matching and verification, the schemas and the problem statements can be represented as hypertext nodes, and their logical connections are straightforward interpretation of the hypertext links.

The experience in building the PARIS prototype indicates that it is quite difficult to construct a system that is able to provide a friendly interface to the user and at the same time produce correct and efficient input to the theorem proving mechanism. A hypertext system together with a powerful logic deduction mechanism will provide the first step towards automated reusability, since developing an application system with PARIS will then guarantee (provably) correct programs.

3.5 Arbitrating Systematic Design vs. Haphazard Hacking

Although some successful software projects originally emerged from unstructured hacking activities, conservative system designers still consider hacking harmful. To be realistic, it is often very difficult to fully understand a design issue without any hacking experience, because some problems, especially the lower level ones, can only be discovered in (and solved by) experience. Since it is typical for an organization to accommodate both types of software developers, a hypertext system would be an ideal forum to let them meet each other halfway while they work concurrently.

In this context, the most promising application for hypertext is its role as a common medium in which artifacts of the "top-down" and "bottom-up" paradigms of design may be concurrently created and uniformly represented. Specification documents, design issues, source code, object code, reusable components, and documentation for a computer system can be created and stored coherently using a hypertext as the common medium to connect the pieces together. The links can effectively represent the connections between portions of the specification documents and the code sections implementing the specification, capture the relevance among system parts, and provide annotations for improvement suggestions and bug fixes. This approach will enable the whole organization to benefit from the artifacts developed by both the systematic designers and the hackers. Consistent conventions can be adopted for using hypertext as the code and configuration management tool described above.

3.6 Communication among People Involved in Information Systems Design

People working with information systems can be divided into four categories: managers, users, operators, and developers. Mills, Linger, and Hevner write in their book [7] that the deepest and most persistent problems of information systems development are people problems, not logic problems. However, to avoid unnecessary people problems, the solutions to logic problems and the usability of the system have to be designed to embody the inherent rationality of its users and operators. The rationality must be discovered and built into the system from the very beginning of the development.

To make the users be aware of their options and the developers realize the exact needs of the users, hypertext is an ideal common medium for people to communicate with each other and exchange their ideas right from the earliest stages of system development. The users can put their ideas into the hypertext network any time they feel convenient, and the designers can gradually adapt the system to the needs of the user. In this situation the users can utilize the system to best advantage and the developers are always clear about the right direction to proceed. A more concrete function of hypertext in this application is to be the organizer of electronic mail and news messages.

4 A UNIX based hypertext system

As an example to describe the technical aspects of implementing the hypertext system whose requirements are discussed in Section 2, we present the concepts of a UNIX based hypertext system called EHT (Easy HyperText). Unlike the recent trend in hypertext research, we adopt a different approach by designing a simple and yet flexible system. We chose UNIX as our working environment, because the UNIX philosophy encourages building a system (or a tool) as an integrated set of tools (or subtools) with uniform interface, and UNIX supports interprocess communication needed in an information network.

Contrary to the popular misconception that UNIX is a "user-unfriendly" operating system, it has gained more acceptance in the commercial software market, because the power and flexibility of UNIX (which can be appreciated by expert users) have attracted many software developers to build user-friendly application systems on top of it.

To have a better understanding on the feasibility and difficulty of developing a hypertext system running on UNIX machines (such as bitmapped graphics workstation or lower resolution personal computers), in the rest of this section we discuss EHT's very high level functional specification.

Name space for the nodes. Although EHT nodes are physically stored in UNIX files, we should provide a different name space for EHT nodes. To be more concrete, EHT nodes should not be identified by their UNIX file names, because many unpredictable anomalies will happen when a logical information unit is allowed to belong to two independent logical structures. The following two cases show the apparent difficulty if an EHT node is always identical to a UNIX file: (1) when we try to copy an EHT network involving files (nodes) residing in two different directories and the working directory is a third directory, it is very complicated to prevent naming inconsistency; (2) if a file (node) is unintentionally deleted from the UNIX file system, the EHT network will contain dangling links.

Storage allocation for the nodes. An EHT node can be physically stored as a whole UNIX file, as a partition of a UNIX file, or as parts across UNIX file boundaries connected with software pointers. There will be a node storage management scheme which consists mainly of a symbol table to identify the physical storage locations for EHT node names. The extra overhead introduced by this scheme will be justified by the consistent and convenient naming procedure of EHT nodes. To ensure compatibility, we also provide utility tools for copying and converting EHT nodes into UNIX files and vice versa.

Screen manager. An EHT node can be in either one of the following three states of screen visibility: invisible, open (as a window display), or shrunken (as an icon). Pop-up menus are used to change the state of a node and to access invisible nodes via its links. Menu selection can be done by the keyboard and also by the mouse.

Multiple buffer text editor. The screen manager can be considered as an extended multi-buffer text editor (such as Emacs), and not as a general purpose window manager used for a special purpose. It is a multi-buffer editor capable of displaying many buffers at the same time, without requiring to have a process running for every buffer. Each node has a node type, and a window can display a node correctly by referring to the node type, for example, textual nodes and graphics nodes are handled in different ways to have them displayed as a window on the screen.

Process control. Running too many processes simultaneously will considerably degrade the performance of a UNIX graphics workstation. A workstation running 25 processes concurrently (even if it is supported by a very fast file server) will degrade its response unbearably, therefore EHT should keep the number of running processes as low as possible.

Since most of the time EHT users will do browsing and reading, it is wise to treat a node as a display buffer without having any process associated with it.

Screen display and process creation. Currently available window managers support only active windows, i.e., windows that need one process or two associated with each of them, so we need to develop our own screen manager for EHT. Since our windows are passive, if we want to invoke a process to manipulate a node, say to compile the C source code contained in the node, we have to select a menu button to invoke the compiling process. The same thing goes when we edit a node, the editing session is done in the buffer and the results are not saved until the user issues a "save" command.

Links processing and database support. To manipulate the links between nodes of EHT, we rely on a powerful database system. Each EHT link is represented as a record in the database system. The database is also expected to speed up access and navigation in the network and to maintain consistent links. For view filtering, the database system can perform keyword search in the (link) records. The fields of a record in the database (i.e., the types of information stored in each link) are defined when a new hypertext network is being set up.

If the description above gives an impression that EHT is not really a hypermedium of the *future*, it is intentionally designed this way with a hope that the EHT concepts will lead to an optimal hypertext system (in terms of usability/cost) in the *present* state of hardware technology.

5 Concluding Remarks

The challenge of software development technology is in the integration of automated programming tools. We have shown how hypertext as a tool can provide some solutions to the problems in creating an optimal programming environment. With the availability of powerful personal computers and sophisticated microcomputer systems, appropriately designed hypertext systems can become more tangible to the average end-users.

The lesson we have learned from the evolution of operating system technology over the past two decades strongly indicates that the usability of a system does not entirely depend on how powerful the system is. A user-friendly and cost-effective hypertext system can be useful to attract more users to explore the wonder world of hypertext. The acquired experience will contribute to the understanding about hypertext developments and applications in general.

ACKNOWLEDGMENTS. The author would like to thank Jeff Conklin of MCC Software Technology Program for his guidance into the delight of the hypertext world, and also Eric Gullichsen, Dilip D'Souza, and Pat Lincoln for sharing the joy of developing a hypertext system. Chung-Kuo Chang reviewed an earlier manuscript and provided many useful suggestions.

References

[1] R. J. A. BUHR, *System Design with ADA*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[2] J. CONKLIN, "Hypertext: An Introduction and Survey," *IEEE Computer*, vol. 20, no. 9, Sep. 1987, pp. 17–41.

[3] B. J. COX, *Object Oriented Programming: An Evolutionary Approach*. Reading, MA: Addison-Wesley, 1986.

[4] S. L. GERHART, "Requirements for Environments for Analysts," MCC Tech. Rep. *STP-229-86*, July 1986.

[5] I. P. GOLDSTEIN AND D. G. BOBROW, "Descriptions for A Programming Environment," *Proc. AAAI 1st Annual National Conf. AI*, Stanford, CA, Aug. 1980, pp. 187–189.

[6] S. KATZ, C. A. RICHTER, AND K. THE, "PARIS: A System for Reusing Partially Interpreted Schemas," *Proc. 9th Annual International Conf. Software Eng.*, Monterey, CA, Mar. 1987, pp. 377–385.

[7] H. D. MILLS, R. C. LINGER, AND A. R. HEVNER, *Principles of Information Systems Analysis and Design*. Orlando, FL: Academic Press, 1986.

[8] N. NEGROPONTE, "Books without Pages," *Proc. IEEE Int. Conf. Communications IV*, July 1979, pp. 1–8.

[9] D. SHASHA, "When Does Non-linear Text Help?" Tech. Rep. *178*, Dept. Comp. Sci., New York Univ., Sep. 1985.

[10] V. Y. SHEN, MCC Software Tech. Prog., Mar. 1986, private communication.

[11] J. S. YOUNG, "Hypermedia," *MacWorld*, Mar. 1986, pp. 116–121.