An Experience in Building an Object-Oriented Prototype of an Advanced Hypertext System

Silvio Meira
Eduardo Albuquerque
José Fernando Tepedino

Departamento de Informática
Universidade Federal de Pernambuco
CP 7851, 50739 Recife - PE - Brasil

Abstract

We discuss the design and construction of a Hypertext System (H) in the framework of object-oriented programming. The work also shows the pros and cons of the programming paradigm —from beginner to specialist levels— as observed from the practical point of view of building (and modifying) a complex system.

The hypertext system used as example has an interest of its own, because of its visions, versions and mail capability among others, and is further detailed in the body of the paper.

Keywords:

Hypertext, Object-Oriented Programming, Rapid Prototyping.

1 Introduction

This paper deals with two subjects of recent interest in the software community, those of hypertext and object-oriented design and programming. The authors have just gone through a one-year effort in designing and prototyping a hypertext system using the object-oriented paradigm.

Initially conceived as an academic exercise to assess the use of object-oriented programming in prototyping complex systems, the project developed into the design and construction of a hypertext system capable of dealing with visions, versions, users, mail and executable code. That is no small achievement, specially when one considers the very little initial experience of the group with both subjects.

A dialect of Smalltalk-80[13], Smalltalk-V[12], was used in the implementation, which was carried out on IBM PC/AT compatibles for most of the project. Final versions of the system run on machines spanning the entire PC range. When starting the project, the group had no practical experience with object-oriented programming, and very little with the concepts and ideas behind hypertext systems and their implementation.

Based on that, we are able to give a report on how a practical experience with Smalltalk influenced our thinking about the capabilities of hypertext systems and vice-versa.

It was also very interesting to go through the process of learning Smalltalk —having to modify a very significant part of the programming environment while at that— and trying to understand some of the basic difficulties normally associated with the matter.

As we shall see, most of the problems of learning Smalltalk can be put down to Smalltalk (the language) and its implementations, having nothing to do with the paradigm.

In the sequel, we present the basic ideas behind H[1], the hypertext system which is the object of discourse (Section 2) and the concepts of object-oriented design, programming and their implementation in Smalltalk (Section 3). Then we discuss the design (Section 4) and implementation of H (Section 5). In Section 6 we consider the results obtained so far and pros and cons of using Smalltalk in projects of the same nature.

2 Basic Ideas Behind H

A *hypertext* is a non-linear document. The concepts of sequencing such as chapters, sections, pages and paragraphs, that we are used to finding in traditional written documents also apply to *hyper documents*. However, a *hypertext* is an *electronic document* with a non-linear structure (a directed graph) and can be used in ways unconceivable for a *paper document*.

References to parts of the same document, or to other documents are possible through *connections* or *links*, that relate *nodes* of information. These connections do not have to follow the linear sequence of the text. They can be strictly *syntactic*[3] so that:

- from a *button*[1], or the *point of origin* of a reference, and by clicking a *mouse* we reach —through the opening of a window with the hypertext *page* that was *pointed* by the connection— the new node.

or *semantic*, when, for example

- a *node* with the *specification* of a system can be related to another one that contains its *implementation*. Changes to one of them (detected by the system) cause modifications on the other one, and these modifications can be managed by the system.

These are only two possible kinds of relationships that a hypertext system can handle. It is possible to have an infinite number of possible relationships and kinds of information stored in such systems.

A node can be directly reached through the activation of a link, by pointing to the corresponding button. Another way to reach a node is by searching for strings within the web of nodes in a fashion very similar to *normal* text editors —this is not efficient for it requires a sequential search through all nodes—, or by *finding* keywords that are defined during the creation of the document. The system keeps a table with all occurrences of the keyword so that the *find* command is very efficient.

Since hypertexts can have, and usually do, a very complex structure, specially if they are related to other documents, it is fundamental to have a mechanism to *navigate* through the structure, so allowing the direct activation of any visible node in the *map* of the hypertext. This *map* is equivalent to "geographical maps" where the cities are nodes and the highways links between them. The *navigator* or *graphic browser* can also be used to create, on-the-fly, a superimposed track to the map, showing where the user is at the moment and the *context* that led him there.

[1] A *button* is a region in the screen sensible to a pointing device such as a mouse.

The data base that supports the hypertext system is a text (or graphics, animations, images, sounds...) graph. The windows on the screen have a one-to-one correspondence to nodes in the data base. Windows can be moved, resized, collapsed (only an icon representing the window is visible) or closed at any moment. The position, size, shape and color of the windows (or icons) should remind the users about the nature of the *contents* of that window. When a window is closed the system must inform the users if there are unsaved changes to its contents and ask whether these changes should be saved or not. When a collapsed window is activated, it must reopen *instantly* where and with the same shape and color that it had before being left.

The user can create new links to new nodes or to already existing nodes, establishing new connections. In environments where several people *cooperate* for creating products, this capacity is of fundamental importance for the development of the work. It is expected that this will be one of the main uses of future hypertext systems, substituting the old fashioned forms of electronic mail and paper messages used nowadays.

2.1 Hypertext in Practice

Although practical implementations of hypertext systems are relatively recent, its original idea dates from 1945, when Vannevar Bush described *memex*[19], a supplement to the human memory where texts and graphics (notes, photographs, drawings, etc.) would be stored with an index scheme with the same functionality of what is today called *hypertext links*.

In spite of not having a digital computer at the time, Bush implemented his *memex* using microfilms and photocells.

Jeff Conklin classifies[19] hypertext systems in four informal categories:

- **Macro-literary systems:** a large on-line library in which inter-document links are machine supported —all reading, writing, collaboration, and criticism takes place via the hypergraph. The pioneer system *memex* belongs to this category;
- **Tools for exploring problems:** tools to support early unstructured thinking about a problem, in which many disconnected ideas come to mind, such as early authoring and outlining ("idea processor"), problem solving, programming and design;
- **Browsing systems:** by and large these are read-only systems for teaching, reference, and public information systems (where ease of use is crucial);
- **General hypertext technology:** general purpose systems to allow experimentation with a range of hypertext applications —most commonly developed for reading, writing, collaboration, etc.

3 Concepts of Object–Oriented Design

Although the literature on object-oriented programming is vast, there are few available works on *object-oriented design*.

Object-oriented design is usually developed bottom-up. While in the top-down approach the crucial design decisions must be taken in the beginning of the project —when very little information is available— using the object-oriented design approach these decisions are spread over the development cycle[14].

The major steps on the object-oriented approach, according to Booch [5] are:

- *Identify the objects and their attributes*, which involves the recognition of the major actors, agents and servers in the problem space plus their role in our model of reality.
- *Identify the operations suffered by and required of each object*, which serves to characterize the behavior of each object or class of objects.
- *Establish the visibility of each object in relation to every other object*. This is the step when the static dependencies among objects and classes of objects are established.
- *Establish the interface of each object*. In this step we produce a module specification, using some suitable notion thereof. This captures the static semantics of each object or class of objects that were established in a previous step.
- *Implement each object*. This involves choosing a suitable representation for each object or class of objects and implementing the interface defined in the previous step.

3.1 Finding Objects

Booch suggests using the *nouns* that are used in the description of the problem space to derive the objects.

Agreeing to this, Meyer[14] states that "perhaps the most useful technique for finding classes, is to look for meaningful external objects. Many classes just describe the behavior of objects from the abstract or concrete reality being modelled —missiles and radars, books and authors, figures and polygons, windows and mice, cars and drivers".

Another way of finding objects, also suggested by Meyer is simply to look at what is available in the object-oriented environment. As object-oriented design favors bottom-up design, it is natural to look into the predefined classes searching for items that can be reused.

4 The Design of System H

System H is a prototype of a Hypertext System with some advanced features, that manipulates text, code and graphics. H has been implemented in Smalltalk-V, and the project has also been useful for studying object-oriented design and programming and for the analysis of the feasibility of implementing a final version using an object-oriented language.

With the experience gathered in the development of H, we intend to have in the future a more sophisticated system derived from a formal specification written in an extension of Z[11].

This new system, **Hex**[16] is being formally specified and is to be used as an *integrator* of the several parts and tools of ADRIS[7] —A Rigorous Software Development Environment— which also has a functional language (A[9]) for generating prototypes from formal, model oriented specifications. The system is not language specific, and current work considers the rigourous development of software using the pairs VDM[10], SML[4] and VDM and Smalltalk.

This environment will have the hypertext system as a basis for its integration and support to software development. The hypertext system can be used either for integrating the documents used during the development of a software system or for automating the development process itself.

4.1 Finding H's Classes

To choose the classes of H, we tried to follow Booch's methodology. In describing a hypertext system, some nouns come up naturally:

1. *node*
2. *link*
3. *document*
4. *user*
5. *manager*, an object that controls the interaction between the main components of the system.

These were the first objects described for H. Some of them were divided into several subclasses.

The system also needed several features provided by the Smalltalk environment. Thus we would use predefined components, and extend some of them. The system is made up of over 50 classes.

4.2 An Overview of H

The upper level of H (its *manager*) understands two sorts of objects: *users (H-Users)* and *documents (H-Documents)*.

- An H-User is a person allowed to have access to the system, being able to create, edit and consult documents. All H-operations are related to an H-User. The system keeps a record of the users responsible for the alterations on documents and maintains some information, such as the address, phone number, interest areas and a picture of the users. Each user also has a *post office box*, and is able to send *hyper-mail* to any user.

- An H-Document is the upper level of the information manipulated by H. Everything that does not belong to a user must be part of a document. An H-Document is made up of *stacks of nodes* (used to keep versions, see Section 4.3.3) which are, in turn, made up of nodes that contain the *actual information* (i.e. *text*, *graphics* or *code*).

The general structure of H can be seen in Fig. 1.

We are now going to describe in some detail the main components of H.

4.3 Nodes

A hypertext is essentially a group of *linked nodes*. The linking must be computer supported. While the links provide the *essence* of hypertexts, the nodes hold the information to be linked.

The way nodes and links can be organized and their possible attributes characterize a hypertext system, making it more or less suitable for an application.

The kinds (and attributes) of links and nodes defined for H were heavily influenced by the environment we were in (Smalltalk) and by the knowledge of hypertext and Smalltalk we had at the moment.

4.3.1 Kinds of Nodes

H supports three kinds of nodes: *text*, *graphics* and *Smalltalk source code*. As we shall see, the nodes are kept in a stack structure for version management (see Section 4.8).

- A *Text node* contains *normal* text and opens on a window associated to a text editor (the Smalltallk text editor), so it is possible to perform the usual text editing operations over it. What makes a text node different from *normal* text is the possibility of creating and following links to other nodes.

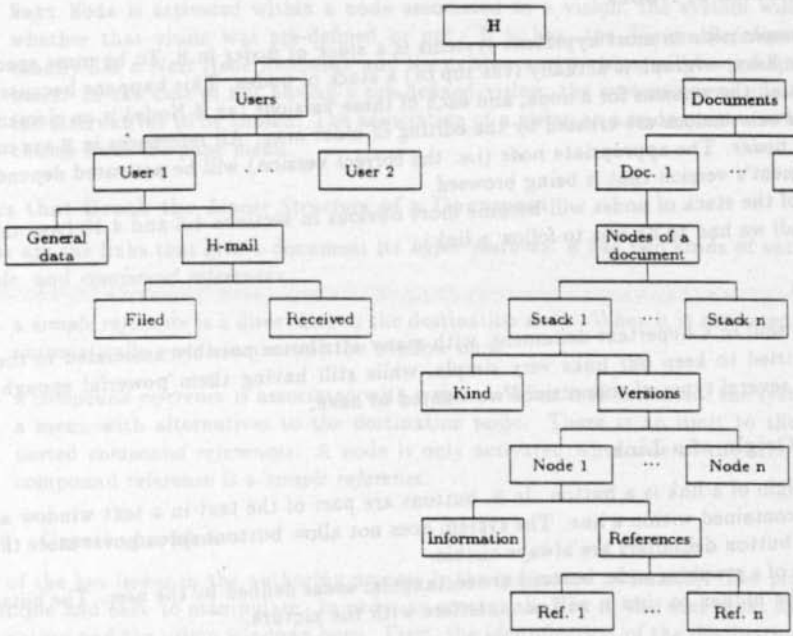

Figure 1: General Structure of H

- A *Graphics node* contains a bitmapped image and references associated to it.
- A *Smalltalk source code node* is a text node containing Smalltalk source code. The difference between them is that when a code node is activated, depending on the mode the user is browsing the document (*editing* or *executing*), the system will open a window for text editing (exactly in the same way as for a text node), or else it takes the contents of the node, compiles and executes it. Code nodes provide great power and flexibility to the system. A code node can have, for instance, the Smalltalk code to perform an animation.

4.3.2 Size of Nodes

A node should contain a *complete idea*. Usually, this means several paragraphs, but it is very hard to decide *a priori* what is going the be the adequate size for a node to contain a *complete idea*.

Some systems, like ZOG[17], have nodes of fixed size —each node occupies exactly half of the screen. That kind of system relies on the very fast activation of nodes in a way that the user does not feel the need for more nodes on the screen at the same time, as he can activate a node almost instantly. Other systems, like Guide[18], show the nodes continuously on the screen, in a way that it is invisible to the users the existence of "borders" between nodes. Still other systems have no limitation for the size of nodes and each one is shown on an individual window.

In H we decided to adopt the last alternative. The nodes can have unlimited size, although it is recommended that a node should not have more information than enough to fit in the default window size. This avoids scrolling, which is a very inefficient way to navigate[6] a hypertext.

4.3.3 The Stack of Nodes

What is a *simple node* in most hypertext systems is a *stack of nodes* in H. To be more specific, a node in the H-hyper-graph is actually (the top of) a stack of nodes. This happens because the system keeps all the versions for a node, and each of these versions (an H-Node) is an element of the stack. Newer versions are created by the editing of older ones. All references in H are made to a *stack of nodes*. The appropriate *node* (i.e. the correct version) will be activated depending on the document's version that is being browsed.

The use of the stack of nodes will become more obvious in sections 4.8 and 4.10 (was this a *hyper-paper* all we had to do was to follow a link).

4.4 Links

Links are the soul of a hypertext document, with many attributes possibly associated to them.

We have tried to keep our links very simple, while still having them powerful enough to represent the several types of connections we wanted to have.

4.4.1 The Origin of a Link

The *visible* origin of a link is a button. In H, buttons are part of the text in a text window and they must be contained within a line. The system does not allow buttons spread over more than one line. The button delimiters are always visible.

In the case of a graphics node, buttons are rectangular areas defined by the user. The button delimiter can be hidden so that it will not interfere with the picture.

4.4.2 The Destination of a Link

Some systems allow the destination of a link to be a line, or even a point in the text. In H, as we have a multi-window system, and supposing that each node encompasses a complete idea, we decided to have nodes as destinations of links.

When following a link, the system opens the window on the appropriate version according to the version of the document that is being browsed. When a link is followed, the system searches for an open window already representing it. If one is not found, a window will be opened on the correct version.

4.4.3 Kinds of Links

There are two main sorts of *links* in H. Those that maintain the basic linear structure of the document, and those that break this structure down.

Links that Maintain the *Linear Structure* of a Document

H has two kinds of such links: *Next Node Reference* and *Vision Reference*.

- A *Next Node Reference* is used to keep a record of the linear structure of an H-Document. When the menu option **Next Node** is activated, the system opens the node referenced by the object *Next Node Reference*. To alter the *next node* of a node, the user must explicitly choose this option in the menu.
- A *Vision Reference* is used to maintain the visions of an H-Document. An H-Vision (see Section 4.9) affects the linear structure of an H-Document, in a way that when the option

Next Node is activated within a node associated to a vision, the system will first check whether that vision was pre-defined or not. If it has, the *Vision Reference* will work exactly like a *Next Node Reference* and the existence of a vision will be transparent to the users. In the case of not having a pre-defined vision, the system opens up a menu with the alternatives to be chosen. The association of a vision to a node must also be explicitly chosen from the pane menu.

Links that Break the *Linear Structure* of a Document

These are the links that give a document its *hyper features*. H has two kinds of such links: the *simple* and *compound references*.

- a *simple reference* is a direct link to the destination node. When it is activated, the system automatically opens (activates) the window on it.
- a *compound reference* is associated with a menu. When it is activated, the system pops-up a menu with alternatives to the destination node. There is no limit to the number of nested *compound references*. A node is only activated when one of the alternatives of a compound reference is a *simple reference*.

4.4.4 Creating a Link

One of the key issues in the authoring process is the indication of links[8]. The process should be simple and easy to manipulate. In order to create a link in H, the user must have both the destination and the origin windows open. First, the identification of the destination node must be obtained. This is accomplished by activating the destination window and choosing the option **Keep Identification** from the menu. Next, the origin window must be activated. Now we have two options: the window may contain *text* or *graphics:*

- If this window contains text (either a *text* or *code node*), the user must select the text that is going to be the button (it must be contained within a single line) and then the option **Associate Reference** must be chosen. The system will insert the button's delimiters and will ask if it is a **Simple** or **Compound** reference. If it is a simple reference the association is automatically done.

  In the case of a compound association, the system will prompt the user to input the name of the association and will ask again if it is a simple or a compound association. This loop will be kept going until a simple association is found. The other options of a compound reference are assembled (or changed) by selecting the button and choosing the **Associate Reference** option.
- In the case of a graphics node, the process is similar, but the user must choose a rectangular area within the picture. When the button is chosen, the delimiter is visible but can be hidden (the region will still be sensitive to the mouse), by choosing the icon "R" on the label of the window.

4.5 H-Documents

The upper level of the information handled by H is an H-Document. In Fig. 2 we see its general structure. The ovals represent stacks of nodes, and we see the several linking possibilities.

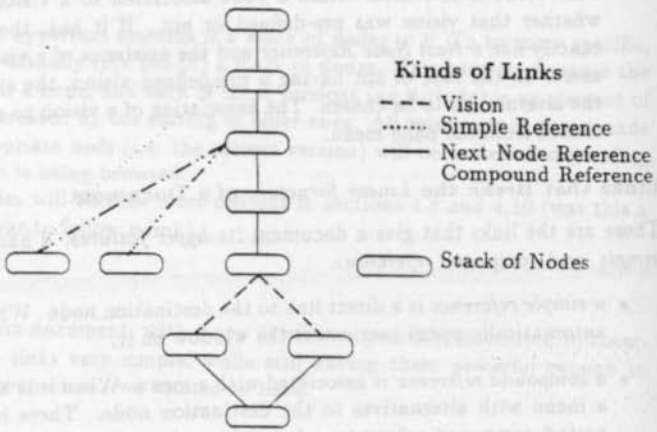

Figure 2: Example of an H-Document.

4.5.1 Structure of an H-Document

An H-Document has a basic linear structure maintained by *Next Node* and *Vision* references. The document is made up of *Stacks of Nodes* where each element represents one version of the node. The origins of references are associated to *nodes* while the destinations are associated to *Stacks of Nodes*. Each version of a node can have its own set of references, but the destination of each one is made to a stack of nodes, and the appropriate version will be activated depending on the version of the document that is being browsed.

There are two types of H-Documents:

- **Visible documents** All *normal* H-Documents are visible. This means that they are available through the H-Document window.
- **Invisible documents** These are documents such as a *hyper-letter*, that have all the features of a normal H-*Document*, but are available only to the addressee.

4.6 Users

Access to H is only allowed to authorized users. The modification of documents is associated to user ids. The system keeps a number of information about users, such as address, phone number, incoming mail, a scanned photo, etc.

4.7 HyperMail

H keeps an electronic mail where each letter is an invisible H-Document. The user has access to the *hyper post office* through his user window. When a user logs in, he is informed about the existence of incoming mail.

A hyper letter, as seen from the user's mail box, is a button to an invisible document. This document has all the capabilities of every other H-Document, thus creating very interesting possibilities for communication between users.

Node \ Version	1	2	3	4	5	6	7	8	9
1	•	•					•		
2	•		•			•			
3	•								
4	•				•				
5			•			•		•	
6	•			•					
7	•					•			
8	•				•	⊗			

Figure 3: The management of versions in H

4.8 Versions

By version we mean the alterations —performed through time— to an H-Document. This is one of the most important features incorporated to H, and it is essential in applications such as CAD, CAM, software development, etc.

4.8.1 The Management of Versions

Although H manages versions at the node level, they are attributes of documents. In Fig. 3 we see how the system manages versions. Initially, the document was created with nodes (*stack of nodes* actually) 1,2,3,4,6,7,8. This means that in version 1 the H-Document was composed of those nodes. Version two was created by editing node 1. If the user browses the document in **version 2**, the system is going to open version 2 of node 1, but all other nodes will have version 1 open because they were not changed. New nodes can be added (like node 5 on version 3) or old nodes can be removed (like node 8 in version 6)

4.8.2 Creating a New Version

Usually, an H-Document has *frozen versions* and one *open version*. A *frozen version* is one that the user explicitly determined should be frozen. Frozen versions are "read only". The user must inform why he froze the version and the system keeps his identification. The user who *freezes* a version is regarded as its author. The *open version* is the version currently being edited. Each *stack of nodes* can have at most one *open version* (the top of the stack). All changes can only be made to the *open version*.

The system automatically creates an *open version* (the new version) when a node of a *frozen version* is edited. The system prevents the user from trying to have more than one *open version* —by editing two *frozen versions* for instance.

4.9 Visions

Visions provide a way of looking at documents from different points of view, or with more or less details. Visions are attributes of documents and are associated to nodes. They affect the linear structure of the document. A node with an associated vision has several *next nodes*, each one corresponding to an alternative vision.

When browsing a document, the user can predefine a vision so that its existence becomes transparent to him. When the vision is not predefined, the system pops up a menu when the user chooses the option **next node** from the nodes that have that vision associated, and the user makes the selection *on the fly*.

4.10 Navigating through H-Documents

The user can navigate through an H-Document sequentially, following its basic linear structure, or by following the references available in each node, or else he can *jump* to desired nodes by using the commands `Search` and `Find`.

The results of viewing documents containing code nodes depend on the browsing mode: if in *editing mode*, when a code node is activated, the system opens a *text editor* window on it, and it all works like a normal text node. The document can also be browsed in *execution mode*, when the text of the code node is compiled, executed and no window is opened.

4.10.1 The Navigation Stack

H keeps a record of all nodes visited during a session. It is then possible to return to any visited node, reducing the disorientation within the system. At the moment, however, there is no graphic browser.

4.10.2 Searching

H allows searching for strings over documents. The search is performed sequentially through all *text* and *code* (in the appropriate version) nodes. Using this command it is possible to reach a node without having to follow the whole path that leads there. That could be used when the user knows *what* to find but does not know *where* to find it. This is not an efficient operation and should be avoided.

4.10.3 Finding and Keywords

It is possible to define keywords for H-Documents. The keywords can be associated to any kind of node (they do not have to be a string that occurs in the node). The system keeps a table with all nodes that are associated with each keyword, so that *finding* a keyword is a very efficient operation. When a keyword is associated with a code node, that node will be edited or executed depending on the mode the document is being navigated.

5 The Implementation of H

System H was not developed in the *traditional* way. By traditional we mean first making a global analysis of the features that should be incorporated, then developing a comprehensive design, writing a formal or informal definition of the whole system and finally building the implementation.

As we have said, in the beginning of the project we had very little experience with both hypertext and object-oriented design and implementation. We did not actually know where we should go and how we would make it there. Therefore, it was very difficult to decide which features we *wanted* to incorporate to the system and which ones we would actually be able to incorporate, considering the limitations of the language (we did not know exactly what they were), people and hardware. Consequently, we never had a complete definition of the whole

system, and its development can be informally divided into four phases. The phases were determined in reality by factors external to the project.

5.1 The Phases of the Development

The main characteristic of the phases were that in each one we had more experience with the implementation language and we had a more sophisticated concept of what a hypertext system should be. In each phase, several features incorporated to the system were defined while we were implementing the previous *version* of H.

The existence of these phases also proves that crucial decisions of a project using the object-oriented approach do not need to be taken at the beginning. Fundamental decisions can be taken *on the fly*, without having a great impact on the overall system.

We are now going to describe the main characteristics of each phase.

Phase 1

The first phase can be seen as a first major exercise while we were beginning to learn Smalltalk. Our intention was more to learn Smalltalk than to work on the *real* project.

The *program* written had no more than 200 lines and we did not alter system classes.

This *version 0* of H had these characteristics:

- Only text nodes.
- No version control.
- No visions.
- The identification for a link was its own text (the button itself). If the users accidentally removed or changed the button, the system would lose the reference. With this limitation, a node could not have more than one button with the same text, even if it had a different destination. In the case of having two identical buttons with different destinations, the system would recognize only the first one (both links would lead to the same destination node).
- The users had to *manually* insert button delimiters. Of course, they were characters easily generated on the keyboard, and so could occur in the text where the user did not want to define a button.
- No cursor control. The users could remove button delimiters, thus losing references.

In spite of all the limitations this phase was very important in the project because of the volume of information about Smalltalk we had to learn in a short period of time (two weeks).

Phase 2

This phase is characterized by the implementation of the first features of the final system.

The main novelties are:

- Version control.
- The button delimiters are characters that cannot be generated by the keyboard. However, the system does not control the cursor yet.

To pass from the first to the second phase we had to perform basic changes upon the data model used. To control versions we had to make some significant changes to the windowing *model* of Smalltalk. These changes were mainly to permit a *model* to have more than one open window. This was necessary to allow a stack of nodes to have more than one of its versions open at the same time.

Initially we thought not to reuse what had been written in the first phase. We then realized how easy it was to reuse code in Smalltalk. In spite of the very deep changes in the structure of what we had done in the first phase, most of what had been written was reused.

In this phase we also began to change some features of the Smalltalk text editor. The HTextEditor button delimiters are characters not available in the keyboard. We also noticed the flexibility provided by the system, allowing changes to be done even in its lower level. The Smalltalk system is totally open for changes (even for the ones not intended, which is sometimes dangerous).

Phase 3

What characterized this phase was the alterations on Smalltalk classes.

The main characteristics of the *hypertext* in this phase were:

- The implementation of *keywords*, and the associated command **Find**.
- Implementation of the command **Search**, for strings.
- Implementation of the H-*HyperMail* subsystem, where a letter has the same features of an H-Document.
- Implementation of *code nodes*.

This was the most gratifying phase of the development, because from then on we had a fairly complex system already, but still very easy to use. We started then using H in the process of documenting its own development and for communication among people involved in its development.

Phase 4

The main features added to the system in the last phase were:

- Introduction of *graphic nodes*, allowing the generation of graphics inside or outside the system (e.g. using a scanner).
- Internal representation of *links*. Up to this phase, the button itself was the key to the reference within a node. If a button was edited, the system would not recognize the reference anymore. In his phase we started identifying the references with a number generated by the system. This number is kept in 12 bits and so each node can have up to 2^{12} leaving links. The identification of the reference is kept just after the *begin button* delimiter, and is not visible on the screen.
- Full *cursor control*. The system prevents the removal of button delimiters and characters that represent the internal identification of references.

- *Migration of documents*. The documents up to now should be kept strictly within the Smalltalk image. This meant that every document created was lost when, for instance, we had to recompile the Document class. The Smalltalk system only recompiles a class if it has no instances, and so we had to throw away all the created documents.

  We created methods to save (in a primitive way) documents to disk, and so we did not have all documents at the same time in the Smalltalk image and classes used by documents could be recompiled without losing them later.

5.2 The Classes of H

We can classify the classes created for H in two sorts. Those that are extensions to System classes (most of them used to create a *Hyper environment*), and those that maintain the core characteristics of H.

5.3 Alterations on Classes of the Original Smalltalk–V

During the development of H we tried to avoid direct changes to system classes. Whenever possible, we created subclasses of system classes. As we needed to create a new environment for H, we had to alter classes like *Dispatchers*, *Panes* and *CharacterScanner*.

The Smalltalk interface is based on the triad *Model-View-Controller* (*Model-Pane-Dispatcher* in Smalltalk–V). As a hypertext system is essentially an interactive environment, we had to create subclasses to dispatchers and panes: For example, one of the affected classes was

- Class **HTextEditor**

  Hierarchy: Object — Dispatcher — ScrollDispatcher — TextEditor — HTextEditor

  This class adds *hypertext* features to the standard Smalltalk text editor. Among the functions that the **HTextEditor** handles are:

  1. handling the mouse's middle button —not used by the Smalltalk— to follow links in H.
  2. navigate through a stack of nodes.
  3. navigate linearly through an H-Document.
  4. perform the *Search* command on the associated pane.
  5. keeping the identification of the node being viewed in the associated pane. This identification is used as the identification of the destination node when linking.
  6. handling *backspace*. This is necessary to prevent the users from removing button delimiters. If users were allowed to remove these characters, the system would lose references.
  7. handling *enter*. H limits the textual button to be in one line thus it must prevent the user from typing an *enter* within a button, which would split the button into two lines.
  8. *single line button*. **HTextEditor** does not allow the creation of a button that occupies more than one line.

5.3.1 H Classes

H-*Classes* are those which are not direct extensions of system classes (i.e. they are direct subclasses of `Object`). For space reasons, we describe only two of the most important ones, namely **H** itself and **Document**.

- Class **H**

  Hierarchy: Object — H

  This is the class that maintains the global control over H. It has only one instance — the global variable `HH`. Among the main functions handled by `HH` we have:

  - **Documents**, responsible for opening the H-Documents window and performing the functions available from there.
  - **Users**, responsible for opening the H-Users window and handling its operations.
  - **Path**. `HH` manages the navigation stack (see section 4.10.1).

- Class **Documento** (Document)

  Hierarchy: Object — Documento

  The document is the upper level of the data manipulated by H. Among the attributes of an H-Document there are:

  1. `Node dictionary`, an instance of `OrderedCollection` where each element is a *stack of nodes*.
  2. `Version number`, the number of the last frozen document version. This number is used to associate new *stacks of nodes* to document versions.
  3. `Authors`, the system keeps the identification of the author of a version of a document. It also keeps the reason for the creation of a new version.
  4. `Visions`, a set with all the visions defined for the document.
  5. `Selected visions`, the visions that were pre-defined by the users.
  6. `Visible`, indicates whether a document is *visible* or *invisible*. All *normal* documents are visible (i.e. can be browsed from the window H-Documents), however, a letter is invisible to all users except its receiver.
  7. `Keywords`, the *keywords* defined for the document and the nodes associated to them.

6 Results

System H, as described, is fully implemented in Smalltalk-V. The system was implemented in about nine months by two programmers, using a PC/AT with 2.6 Mbytes RAM memory. The original version was implemented in Smalltalk-V/286 and was later ported down to Smalltalk-V on PC-XT and is currently being distributed free to β test sites. The whole system is composed of 7789 lines of code, of which about a third is of high complexity.

6.1 Main Characteristics of System H

System H is a prototype of a Hypertext System with the following characteristics:

- It is a general hypertext system (according to Conklin's classification).

- It has a simple interface. H's interface uses the features provided by the Smalltalk interface, that have influenced most modern computer human interfaces.
- It is a multi-windowing system where each window corresponds to a node of the web. This makes navigation easier because it allows the user to have, at the same time, many windows opened on the information he wants. A window can be activated just by clicking the mouse within it.
- Information handling is based on the use of the keyboard and the mouse in an efficient and simple way.
- The system has mechanisms for *content search* within the data base. These mechanisms make the process of finding information easier and are accomplished by the comands **Search** and **Find**.
- It has context sensitive menus. This simplifies user interaction with the system because the possible operations on each context are explicit.
- It has a *hyper electronic mail* capability, which helps the development of cooperative work, where communication among involved people is essential.
- The management of *versions*. The system keeps a chronological control of changes over documents, and allows a user to browse a document in any desired version. It is even possible to have windows open on different versions of a node at the same time.
- The management of *visions*. A document can be structured in a way that different users can navigate through according to their own interest.
- The windows have a uniform layout. User orientation is easier in an environment where there are not many windows with different sizes at the same time.
- The nodes can hold information that can be *text*, *graphics* or *Smalltalk source code*, which provide great flexibility for creating documents.

6.2 Considerations on The Use of Smalltalk

The use of Smalltalk was of great importance to the development of H. The project can be seen as two parallel efforts: the first one being the development of a hypertext system, and the second an assessment of the use of an object-oriented language, its advantages and problems.

6.2.1 Advantages of Using Smalltalk

The main advantages we found in using Smalltalk were:

- **Code reuse.** As we have said, we never had a complete design of the system we were implementing. All the same, we were able to reuse most of the code in the several "phases" of the project, even without any plan to do so. Also, as the whole system is "open", we were able to perform modifications in its lower level without major additional difficulties than that we had in *our classes*.
- **Smalltalk environment.** The Smalltalk environment provides some tools for browsing through the system and for debugging. It is also particularly adequate for developing interactive applications due to the ease with which graphics and windows are used.

- **Interactive execution**. Smalltalk allows the immediate and interactive execution of chunks of code. This stimulates reusability because it allows the programmer to try out different possibilities on the spot[15].

We believe that the use of Smalltalk was essential for the development of the system the way it is, considering the manpower and equipment constraints we had. Furthermore, we believe Smalltalk can be strongly recommended for prototype development.

6.2.2 Disadvantages of Using Smalltalk

In spite of all the advantages of using Smalltalk, there are some drawbacks:

- **Learning curve**. It is easy and fast to write an application in Smalltalk (or in any other language) after you know it well. The problem with Smalltalk is that, because of its sheer size and the way the code is fragmented, it takes much longer for a novice to write his first Smalltalk program than it would if a *conventional language* was used.

  We have found that the best way to learn Smalltalk is to have someone to help you with the doubts as they come up. Unfortunately, we had no Smalltalk programmer available from the start, and it used to take us a long time to solve some very simple problems.

- **Finding a software component**. Because of the way code is fragmented within the environment, it is sometimes difficult to locate the desired component. The inheritance mechanism sometimes makes this process more difficult, as the user may have to follow a whole hierarchical path to find methods.

- **Performance**. Smalltalk is interpreted —this makes real-life systems far less efficient than they could be if we had a compiled implementation.

7 Future Work

The main features that we intend to add to a future version of System H are:

- Development of a graphic browser. One of the main problems in using hypertext systems is *getting lost*. Some authors like Akscyn[6] say that a graphic view is only useful when you have a very extensive and non-linear structure. Others, like Garrett[21], state that a graphic representation or map is useful in helping the user to understand the information of the network. We believe that a graphic browser with *global* and *local maps* could only improve the navigation through the hyper-graph.

- Development of *search by structure*. H supports only *search by content*, and it is often desirable to search, for instance, "all nodes that have references to nodes that have the word *hypertext*".

- Saving *paths*. A path is a sequence of nodes to be visited. For example, the system could have a path that is the suggestion of a *minimum* class saved by a professor. The student can follow that path, and make his own deviations where he thinks he needs more information, or where he is more interested in.

- Introducting annotations. An annotation is a piece of text that is introduced in the document but is not part of it. It works like the side notes we usually jot down in books. An annotation can be hidden (or not) from other users and the user who placed them (or who he allows) can have a commented version of a hyper-document.

- Status for users. H does not classify users, any one can create, edit or remove any document, node or link. In a future system the users should be classified so that some users can read and edit their own documents, while others can only read and place annotations on them. Some users can only have access to certain visions of documents, etc.
- Store management. System H relies on the management provided by the Smalltalk system, which keeps all data in main memory. However, any practical hypertext system must have an efficient way of saving and retrieving information from a data base kept in a permanent medium and possibly distributed through different machines.

  The development of a general store management system for hypertext is in progress since the beginning of the year, and we shall have a prototype working soon.
- Implementation of a *garbage collector*. The flexibility provided by the system for changing the structure of documents allows the creation of unreachable cycles that are not detected by the Smalltalk garbage collector. The system should have its own garbage collector, and should be able to inform the user about the existence of such cycles before their removal.

References

[1] E. S. Albuquerque: *O Sistema de Hipertexto H*. Dissertação de Mestrado, UFPE, Dezembro de 1989. (In Portuguese).

[2] A. M. L. Vasconcelos, A. C. V. Melo e S. R. L. Meira: *Hex —Hipertexto como Suporte a Ambientes de Desenvolvimento de Software*. Dep. de Informática, UFPE, Janeiro 1989. (In Portuguese).

[3] S. Meira, E. Albuquerque, J. Martins, A. Melo e A. Vasconcelos: *Hipertexto: O Projeto do Sistema H*. III Simpósio Brasileiro de Engenharia de Software, Recife-PE 1989. (In Portuguese).

[4] R. Harper, D. MacQueen and R. Milner: *Standard ML*. Edinburgh University, 1986.

[5] G. Booch: *Object-oriented Development*. IEEE Transactions on Software Engineering. Vol. SB-1, N 12, Feb. 1986.

[6] R. M. Akscyn, D. L. McCraken and E. A. Yoder: *KMS: A Distributed Hypermedia System for Managing Knowledge in Organization*. Comm ACM, July 1988.

[7] S. R. L. Meira: *Ambiente para Desenvolvimento Rigoroso de Software. Notes* **Nr. 1**, Agosto 1988. (In Portuguese).

[8] B. Shneiderman: *Reflections on Authoring, Editing, and Managing Hypertext*. CAR-TR-410, University of Maryland, 1988.

[9] S. R. L. Meira: *Introdução à Programação Funcional* . Versão preliminar à VI Escola de Computação, 1988. (In Portuguese).

[10] C. B. Jones: *Systematic Software Development Using the VDM Approach*. Prentice-Hall Int, 1986.

[11] B. Sufrin: *Z Handbook*. Oxford University Computing Laboratory - PRG, 1986.

[12] Digitalk Inc. : *Smalltalk-V 286. Tutorial and Programming Handbook*. 1988.

[13] A. Goldberg and D. Robson: *Smalltalk-80 The language*.Addison-Wesley Publishing Company, 1989.

[14] B. Meyer: *Object-Oriented Software Construction*. Prentice Hall International, 1988.

[15] J. Nielsen and J. T. Richards: *The Experience of Learning and Using Smalltalk*. IEEE Software, May 1989.

[16] A. C. V. Melo: *Especificação Formal de Links e Nós em Sistemas de Hipertexto*. Dissertação de Mestrado, UFPE, Dezembro de 1989. (In Portuguese).

[17] A. Newell, D. L. McCracken, G. Robertson and R. M. Akscyn: *ZOG and the USS Carl Vinson*, Carnegie-Mellon University, 1982.

[18] P. J. Brown: *Turning Ideas into Products: the Guide System*. Personal Communication.

[19] J. Conklin: *A Survey of Hypertext. MCC TR*, Nr. STP-356-86, Rev 1, Austin, TX, Feb 1987.

[20] A. L. Cavalcanti, J. Kelner e A. Pardo: *LindA: Uma Linguagem de Autoria Automática para Hipertexto*. III Simpósio Brasileiro de Engenharia de Software 1989. (In Portuguese).

[21] L. J. Garrett, K. E. Smith and N. Meyrowitz: *Intermedia: Issues, Strategies, and Tatics in the Design of a Hypermedia Document System*. RI 02912, Brown University.