Defining Hypermedia: The Essential Elements

Michael A. Harrison*
Computer Science Division
University of California
Berkeley, CA 94720

July 29, 1992

Abstract

Over fifty years ago, Vannevar Bush hypothesized about what are now called hypertext and hypermedia systems. He and later visionaries discussed a number of applications which could revolutionize the way organizations and individuals access information in its various embodiments. In spite of the great attention currently being paid to this area, there are few, if any systems which fulfill the vision.

In this report, we try to isolate the essential character, the *sine qua non,* of hypermedia systems. Our methodology is to offer twelve criteria or rules about hypermedia systems. It is our thesis that any robust, or industrial strength, system must meet these criteria if we are to realize the full potential of hypermedia.

Some of the criteria serve as a challenge to the technical community. Others are of a social and political nature. Companies need to cooperate in deriving standards to enable the exchange of hypermedia information or to agree on APIs to achieve interoperability. These issues involve networking, computer hardware and software, as well as formats for certain consumer electronics products such as analog and/or digital HDTV.

Introduction

The last few years have seen an explosion of interest in multimedia, hypertext, and hypermedia systems and software. Even mass-market magazines such as Business Week and Time have prominently featured articles on multimedia while journals, conferences, books, and seminars have proliferated. The first major conference on the topic took place at the University of North Carolina in 1987. Over two hundred interested people attended and a fine intellectual and social time was had by all. While there was a sense of excitement, there have been few major developments since then. Research and development appears fragmented. In fact, there is not even a common agreement of what is needed to develop the promise of hypertext and hypermedia systems.

*Sponsored by the Defense Advanced Research Projects Agency (DoD), monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292.

The purpose of this paper is to offer a modest and non-controversial definition of Hypermedia systems. New working definitions, by their very nature, must be concise. We believe that this area is fundamentally important and that there are important principles to be found, systems to be built, and significant industrial applications to be created. Hypermedia has the potential to revolutionize the development and delivery of applications as well as the sophistication and usability of those applications. Therefore, this paper enumerates the fundamental properties that a system must have to deserve the name "Hypermedia System".

Industrial users of such systems should demand hypermedia systems that meet this set of requirements. Designers of such systems should meet these criteria and extend them. Only in this way will the technology advance and will everyone benefit.

History

Almost fifty years ago, Vannevar Bush wrote an article [2] in which he described a system called Memex which could be said to have multimedia capabilities (typed items, photographs, and hand-written annotations on microfilm). The system also had associative memory and links representing "trails" between its objects. It is fair to credit Bush with being the parent of these concepts even though the technology did not exist to implement them at the time.

Another major event was the appearance of the Augment System of Douglas Engelbart in the 1960s [5, 4]. This system employed a mouse and an optional one-handed chord keyset, and it offered genuine hypertext capability. It appears to be much harder to identify the first system that genuinely employed multiple media because the list of candidate systems is unclear. Ted Nelson's role as a visionary should also be mentioned: he first saw that these principles could be extended to a hypertext network of all of society's documents [12].

In 1987, the HyperCard program [8] was supplied to all purchasers of Macintosh computers. While HyperCard may be regarded as a primitive hypermedia system, its widespread availability stimulated interest in this area and thousands of HyperCard stacks were created. For a general discussion of such systems, see [13].

Definitions

The computer systems we discuss all operate by visiting a sequence of "pages", sometimes called cards or notecards. A "page" might be a page of text, a graphic, an animation, or other objects. A *hypertext* system is a system as described above for which a user may also

1. create or delete pages,
2. insert or delete one or more links between two previously defined pages,
3. edit pages, and
4. visit a page by following a link.

Thus it is fair to talk about traversing a graph of pages in a hypertext document.

A multimedia system is a system as above where the pages may be of different media types. We discuss in Rule 1 below the different types of media that minimally should be included here. It is important to note that the list should be open-ended since the computer field advances dramatically whenever new media types are introduced (think of I/O devices such as the laser printer which drove the development of desktop publishing).

Now, we can easily reach our goal. A *hypermedia* system is a multimedia system that is also a hypertext system. Crudely,

$$\text{HyperMedia} = \text{Hypertext} + \text{Multimedia}\ .$$

That is, a hypermedia system is a hypertext system that incorporates a significant number of distinct media types.

If this definition is an accomplishment, it is a shallow one because there exist so many vacuous examples. Even the first UNIVAC had a loudspeaker on it and no one would want to call *that* a multimedia machine, even though it could play "music".

There are two important aspects of hypermedia systems: building applications, and making those applications available to their intended users. The process of building or creating these applications is frequently called authoring. Programming refers to creating applications in programming languages by experienced and highly trained programmers. Authoring, however, encompasses a wider range of techniques than programming and a much larger group of less experienced developers. The authors are accomplished people in their disciplines but generally not software experts. The word authoring has its roots in the publishing world since building a hypermedia application is much more similar to creating a document than it is to writing pages of programming code.

In what follows, we offer a dozen criteria by which to compare and evaluate hypermedia systems.

1 The Multimedia Rule

A hypermedia system must support both user-driven and time-driven input and output of the full range of sensory-rich multimedia types, including not only those items that are stored internally in the system but also those stored externally in other data sources.

Most popular application-development systems today support only the traditional data types common to commercial data processing: text, numbers, and dates. All of these are displayed to the application user in character form, either as dot-matrix characters on traditional CRT displays or in various fonts and sizes on high- resolution workstation display screens.

The full range of sensory-rich multimedia types includes not only these familiar business data types but also several more that can leverage the tremendous power and sophistication of human eyes and ears as peripheral devices to the brain:

- static 2D (two-dimensional) color graphics

- static 3D (three-dimensional) color graphics
- animated 2D color graphics
- animated 3D color graphics
- audio
- static images
- full-motion video

Graphics are computer-rendered representations of real-world or imaginary entities. For example, they might be drawings of every-day objects such as telephones or automobiles, drawings of surreal beings such as Terminator 2 or cartoon characters, models of phenomena or concepts such as a graph of an organization's financial performance over time, or a map of the highway and railroad systems in an area of the country. Well-designed color graphics can convey a tremendous amount of information in a very compact, memorable, and visually appealing form.

Many graphics are staticunchanging after they have been displayed on a screen. Animation is the process of bringing these static graphical objects to life. Animated objects can appear or disappear through various special effects such as slow fades, change color or shape before your eyes, or move across the screen over a variety of paths. Animation can be used to add realism to artificial objects or surrealism to images of real objects. Well-designed animation adds interest to applications and increases the memorability and retention of the multimedia information they display.

Audio refers to recorded voices, music, or other sounds that are later played back. Well-chosen audio not only increases information retention but also provides ways to communicate with application users when their eyes are not fixed on their display screen. Audio is also increasingly being used as an input medium, complementing the use of typewriter-style keyboards and pointing devices such as mice.

Images are static pictures of real-world objects or scenes that have been captured and recorded on photographic film, video tape, optical disk, or other physical media. Full-motion video refers to pictures of an event captured and recorded over a period of time at a rate that on playback can exactly duplicate the captured event. Today's video technologies record and play back at thirty frames (or pictures) per second. This rate is adequate for most applications. It is above the minimum frequency (fusion frequency) at which the human visual system interprets sequences of static images and believes it is watching real-world motion. In addition, full-motion video on playback can be slowed down, speeded up, paused, and even played backwards in time.

A hypermedia system must not only manipulate multimedia objects as described above but also store and retrieve these objects on computers. For improved system performance, these objects might be stored within the software system itself. However, bidirectional (i.e., read and write) access to objects outside the system is also a requirement. Multimedia objects, especially live-video clips, consume immense quantities of storage space. In addition, organizations will receive and manage multimedia objects in a variety of physical media such as CD-ROM. For them, the conversion and redundant storage of multimedia would be additional overhead that is both inconvenient and expensive.

Early applications of multimedia have been restricted to marketing and sales presentations. In these applications, the author edits and synchronizes separate tracks of audio, image, and video into a presentation that is played by the user in the author's predetermined sequence from beginning to end. Significant business applications that employ multimedia, by contrast, must be user driven and not time driven. The user of the application, rather than its author, must be able to control the pacing and sequencing of application elements. The user needs to respond to environmental or application-sensitive events and continually decide what actions he wants to take next at the time he is running the application.

Multimedia promises to significantly improve the processing and retention of information by application users. However, a multimedia object in isolation is not particularly interesting or beneficial. The real power of multiple multimedia objects appears when they can be used together. For example, consider a hypothetical hypermedia architectural application. The user interacts with a two-dimensional graphic of a building. With his pointing device such as a mouse, he clicks on a floor of the building. The two-dimensional building graphic is then joined by a three- dimensional representation of the floor layout. The user then clicks on a menu item that opens a window on the screen and plays a synthetic three-dimensional graphical walk-through of that floor, with the user controlling the speed and direction of the walk-through. Another mouse click invokes a video clip of a person actually walking around the floor – from the eye level and point of view of that person. Accompanying the video clip the user hears an audio recording of the person recounting his impressions of the color schemes and spaces that he encounters on this walk.

2 The Object Rule

A hypermedia system must employ object metaphors for system interaction, system storage, and application creation.

The terms "object" and "object oriented" have become so overloaded[1] that a few words of history may be helpful. Object-oriented software began at Xerox PARC (Palo Alto Research Center) in the 1970s. It may be thought of as the encapsulation of data plus the "methods" or functions that operate upon that data. Objects can communicate by sending messages to one another as in the Smalltalk-80 language [7]. For example, a laser printer and its font library might be represented as an object. Another object may "send" a message to that object with a request to print a particular file in a particular font on the printer. The object metaphor is a powerful one in modern programming and is especially useful in creating software such as direct-manipulation systems.

[1] Computer programmers use the term "overloaded" when the same name serves for several different concepts.

"Inheritance" is a mechanism that allows new classes of objects to be defined as extensions to previously defined classes. It is the key notion that provides the leverage to construct new objects from previously defined ones instead of starting from scratch. The importance to the user of being able to customize a system through simple modification of existing system objects is difficult to overstate. For example, the elegant MediaView multimedia-document publishing system [14] is relatively easy to implement by subclassing objects in the Application Kit based on Objective C and supplied by a workstation vendor. Then new behavior is added to the subclassed objects or their existing defaults are overridden.

Another use of the word "object" concerns object-oriented database-management systems (OODBMS). One needs to store heterogeneous multimedia objects such as text files, images, and video clips in a persistent store where they outlive the applications that created them. There can be many such multimedia objects in a sophisticated hypermedia application. Their granularity or size can vary widely and they can be related or "linked" in various ways for subsequent fast retrieval from large object collections. Hierarchical flat-file systems such as those of the UNIX or VMS operating systems lack important object-storage management capabilities such as version control, checkin and checkout, and fine-grain security and access control over small objects below the file level.

An object-oriented development and storage environment offers a high degree of control and flexibility in developing applications. Application authors receive precise control over basic system objects such as tools, pages, graphics, widgets, and user-defined objects. They can share and reuse these objects to develop modular applications much faster than with other programming methods.

3 The Scripting Rule

A hypermedia system must provide a rich, user-accessible scripting language for extending and modifying the behavior of the system and its application elements.

Authoring in a hypermedia system is not and must not become ordinary programming in an imperative language. A scripting language should be as non- technical as possible. For example, a scripting language such as HyperTalk [8] is much easier for a novice to use than, say, APL.

A scripting language is used to compose scripts – very short programs based upon the user's native language. Scripts can imitate user actions or respond to external events such as the clicking of a mouse button. A well-crafted hypermedia system will help its user by providing templates of code fragments for standard events. For example, the code fragment

```
on buttonPressed
     <statements>
end buttonPressed
```

can be selected from a menu of templates and then the template <statements> can be replaced by actual instructions written by the user.

How are scripts deployed? The hypermedia world is made up of objects such as documents, pages, text boxes, buttons, etc. A script can be attached to any object to specify its behavior under various conditions. The script need not be named since a good system assigns a name automatically.

Significant hypermedia applications require a functionally and semantically rich scripting language, the details of which are beyond the scope of this paper. However, a complete scripting language should provide the ability to perform at least the following functions:

- send messages to objects (for example, "turn green and blink")
- receive messages (for example, "the mouse button has been clicked on you")
- manipulate objects and modify their behavior (for example, add a script to a button so that the button changes color when it is clicked with the mouse)
- perform some data processing functions (for example, sort a list of names into alphabetical order)
- interface with arbitrary programs (for example, launch a word-processing program, pass it some text, and receive back control when the user exits the word processor).

There is a debate about how rich a scripting language should be, especially in the area of data types and data structure. However, an interesting functionality test for a hypermedia system is whether a user can construct with it an interface builder – a set of objects and behavior that can be easily subclassed and modified to construct quickly custom user interfaces for his applications.

A good scripting language shares some important attributes with the 4GLs (4th- generation languages) that are increasingly used today to build standard data- processing applications. Unlike a 3GL such as COBOL or C, a scripting language is non-sequential and much more suited for the asynchronous, non-linear control- flow environments of the object-oriented hypermedia world. A scripting language also allows environment-specific primitives to be integrated into the syntax of the language instead of appearing as externally callable routines. And a good scripting language allows an application author in a single integrated environment to move quickly back and forth between building a new routine and testing it in operation, instead of segmenting the application-building process, like 3GLs do, into compilation, linking, and execution phases.

Finally, scripting and object inheritance must work well together. For example, a message may be sent to any object – a simple button or graphic, or a larger object such as a document page that contains the simpler object. If an object receives a message and no response, i.e. message handler, is defined for that message, the system passes the message up the object inheritance hierarchy to that object's parent class. Only if no message handler in the entire object hierarchy can respond does the user receive an error response that the message was not understood.

4 The Multi-User Rule

A hypermedia system must support the collaborative building of applications by multiple concurrent authors on networks of heterogeneous computers and the execution of those applications by multiple concurrent users.

We are now enjoying (some would say enduring) the fourth generation[2] of computing – networks of heterogeneous desktop and deskside computers. Network computing fosters cooperative work within groups of users by allowing group members to collaborate electronically on shared projects and to exchange data and other relevant information quickly and easily. Complex applications can now be built by teams instead of individuals and be deployed on networks of computers from multiple manufacturers with multiple operating systems and network protocols.

Hypermedia systems must support these collaborative network-computing environments. Builders of hypermedia applications must be able to work together electronically through their software. An object (for example, a control panel with radio buttons) created by one application builder must be both accessible to and shareable by the others. If multiple users attempt to access and modify the same object concurrently, the system must resolve the conflict, either by allowing only one of the users to "check out" the object or by giving each user access to his own copy of the checked-out object while keeping track of and managing the copies in their different versions.

When the developed application is placed into production, its users must also operate without interfering with or invalidating the work of others. For example, users of an order-entry application must be able to enter concurrent orders for identical inventory items without selling more items than physically exist in inventory. The system must resolve such concurrency conflicts while maintaining the integrity of the data and the application.

It is not only application building and application processing that must be distributable among groups of users. The storage and management of application data and application objects needs to be distributed as well. The system must mask the presence of the network, the size of the collaborating group, and the distribution of data and multimedia objects while creating the illusion that the network is a single computer with a single object store and a single user. And the system must sustain this illusion of network transparency as group size changes or as objects are redistributed so that applications need not also be changed.

5 The Scalability Rule

Applications developed with a hypermedia system must continue to work well and with predictable performance characteristics when deployed in production environments that contain

[2] Computing generations occur approximately in decades. The first three generations were batch processing on mainframes, timesharing on minicomputers, and personal computing. There is actually a fifth generation which was defined by a large project in Japan. This generation was described to be the information age, but the leading-edge projects at ICOT were unable to achieve their goals and so redefined their goals to be so modest that there has been little impact on industrial computing.

much more data and many more concurrent users than existed in the prototype or pilot version of the application.

The folklore of commercial application development abounds with tales of prototypes or pilots that demonstrated well but failed miserably when deployed in their target environments. Such failures are often errors of scalability, where the success of the application depends on the scale of the problem or other factors to which it is applied.

The size or complexity of application problems can rarely be predicted. Therefore, a hypermedia system must be scalable. Scalability allows an author of hypermedia applications to build prototypes for test environments with the confidence that their performance and usability in the target production environments will not depend on the number of concurrent users, on the number, size, or distribution of objects employed, or on the number or mix of media types employed. For example, if the number of application users doubles, users' response times must not double as well.

Without such scalability, a developed application would have to be extensively tested and debugged in the target environment itself, often at great financial or organizational expense. With scalable software, applications can be delivered more quickly and enjoy longer lives if their environments undergo change.

Unfortunately, scalability is difficult or impossible to prove formally in advance. One must build scalability tests into prototypes and must speak with other users of the software to determine if their scalability expectations have been met.

6 The Interoperability Rule

Hypermedia applications must be able to exchange both data and control not only among themselves but also with external applications and data stores such as SQL relational data bases.

Open systems and interoperability are requirements for survival and prosperity in today's computing world. Interoperable applications allow you to leverage the work of other application authors instead of doing all the work yourself. An interoperable application can cooperate and communicate with other applications so that you can solve new classes of more complex problems faster.

The most common example of interoperability is that of preparing a text document using a word-processing program and embedding a portion of a financial spreadsheet within that document. Of course, a static textual copy of the spreadsheet cells could be created with the spreadsheet program and then inserted into the document. However, if the spreadsheet were later changed, someone would have to notice the change and explicitly insert the updated data into the document. Interoperability between the word processor and spreadsheet programs would eliminate the need for this manual intervention. A "live" link between the document and the spreadsheet would allow the document always to see and present current data from the spreadsheet whenever the document is viewed on a screen or printed on a printer.

Application interoperability requires three important capabilities. The first is that interoperable applications can communicate and exchange data with one another. The communication may occur in various ways such as through intermediate temporary or persistent data files, through message store-and-forward systems, or via inter-process communication facilities such as "pipes" or "sockets". Inter- application data communication often must also be bidirectional, with either application being the source or destination (or both) of the communication.

The second capability is bidirectional control flow. This means that interoperable applications must be able to initiate and, where permitted, terminate one another. For example, an order-entry application may need to launch a packing-slip application to print packing slips for the shipping department and be informed when the packing slips are done.

The third capability is interoperability with foreign, external applications. For example, a hypermedia order-entry application that displays images of ordered items may need to invoke an electronic-mail application to communicate order status automatically to field salespeople and an SQL database server to retrieve and update customer credit information.

Interoperability is fostered and nurtured by standards (see Rule 12) such as OLE (Object Linking and Embedding) and ORB (Object Request Broker). Standards for bidirectional data flow and control flow between applications mean that a software vendor need implement interoperability only once according to those standards and thereby achieve interoperability with all other software whose vendors have done the same. Without such standards, interoperability among products requires all their vendors to do substantial work. If there are n vendors who want to interoperate, for example, the amount of their work increases from n to $n(n-1)/2$.

7 The Hyperlink/Hyperview Rule

A hypermedia system must allow users to establish navigable relationships among objects of different media types, to browse and navigate those relationships in an ad- hoc non-linear manner, to determine readily one's location among them at any time, and to return easily to any prior location on a navigated path.

When multimedia objects are used together, they must be linked to one another through some appropriate navigable internal mechanism. For example, a multimedia object representing an employee's personnel file might be linked to an image object containing a photograph of that employee. That same photograph might participate in a second link from an object representing the company's baseball team for which the employee plays third base. A company executive viewing an employee's personnel file will want to see the employee's picture appear as part of that file. The baseball coach, however, will want the picture to appear as part of the baseball-team object whenever he views it. In both cases, it is the same image object that appears in two different locations. A separate copy of the image object could have been created instead, but that would have required redundant storage space for each identical copy. Linking to the image object from several other objects conserves storage space and ensures that any changes to the image object (for example, a new photograph is taken because the employee grows a beard) are immediately visible from all linked objects. These kinds of read-only, single-level, automatic links are called *view links* and are an essential capability of hypermedia systems.

Now let's continue with our example. The same executive viewing the same personnel-file object notices that the employee works in the engineering department. The executive decides to visit the engineering department's project file (which may reside on another computer on the network) and creates a link to that object. Upon "arrival" in the project-file object, the executive notices several other engineers, creates and follows a link to the picture of the first of those engineers, then creates and follows a link from that engineer's picture to his employee file. These kinds of multi-level links are called *hyperlinks* and the process of traversing them in an ad-hoc non-linear manner is called browsing. Both hyperlinks and browsing are also essential capabilities of hypermedia systems.

In a large hypermedia application or electronic hypermedia document, the link structure can become quite complex. To return to our example, the executive has now forgotten exactly where in the link structure he is and how he got there. He is "lost in hyperspace", or more accurately, lost in a network of fundamental objects such as pages. He needs to be able to "step back" and view a roadmap of his current location and determine how he can return to where he began, either by retracing all of the steps he navigated or following a single link back. In some systems, the network of linked objects itself is displayed on a page where it can be edited to add and delete object nodes as needed [9]. This ability to take a bird's-eye view of the environment at any time and from any altitude is called hyperviewing and is an essential capability of hypermedia systems.

In large hypermedia systems such Intermedia [15, 11], the link structure can become complex. Conklin [3] discusses this phenomenon and provides a vivid example of the phrase "lost in hyperspace".

It should be possible to perform general computation in a hypermedia system as discussed in Rule 3. But since the hyperlinked network of objects itself is describable as a hypermedia document, it should also be possible, at least in principle, to search that entire network and compute some function on each object node (for example, the number of times an author uses the word "really" as a synonym for "very") or to generate links automatically based upon content rules.

A hypermedia system's browser and hyperviewer should support a graphical interface. That is, one's location in a complex link structure should be both diagrammable in and manipulable through a roadmap-like graph. This graph- layout problem is not easy. Designing a visually attractive, scalable, and efficient solution is a challenge for software vendors. Techniques such as elision, holophrasting, and fly-overs have been used in various viewing systems and need to be adopted and adapted here by vendors.

8 The Technology Independence Rule

A hypermedia system must provide independence from any particular vendor's technologies and guarantee that the hypermedia applications will continue to work without change as the *underlying technologies evolve.*

Applications occupy a precarious perch in the computing world. They sit atop a constantly shifting layered pyramid of hardware platforms, operating systems, network protocols, data formats, graphical user interfaces, and other technologies. Yet applications are often used to perform mission-critical functions for groups or entire organizations, so they must be fault tolerant, operate error free, and survive without modification over long periods of time.

Underlying technologies change frequently and quickly. For example, processors increase in speed. Memory chips become denser, faster, and less costly per bit. Color display screens become larger with higher pixel densities. New color-display standards are adopted. And new technologies such as HDTV and Laser Discs constantly appear. How can one simultaneously achieve the seemingly incompatible goals of application stability and taking advantage of new or better technologies?

You can and should demand that your hypermedia system give you technology independence today and assure you of technology independence tomorrow. The hypermedia system must run in a variety of popular technology environments in various combinations. How the vendor achieves this portability is not your concern (though we do offer vendors a methodology in Rule 12). But the benefit for you is that you won't have to play the role of a riverboat gambler and make irrevocable bets that your choice of underlying technologies will win the technology wars [1].

Numerous technology choices are available. Here are just a few examples in several categories:

- Operating systems: MS-DOS, Windows NT, UNIX (various flavors), VAX/VMS.
- Processors: Intel, Sparc, Mips, Motorola, VAX, Alpha.
- Network protocols: TCP/IP, DECnet, IPX/SPX, XNS, ISO protocols, etc.
- Multimedia hardware: TV, HDTV, CD-ROM, Laser Disc, DAT, DVI, etc.
- Graphical User Interfaces: MS Windows, Macintosh, OpenLook, Motif, DECwindows, Win32, Presentation Manager, etc.

Purely superior technologies rarely gain widespread acceptance. Market dominance for a particular technology or product is a function of so many other unpredictable factors. If you choose a hypermedia system that is inextricably bound to a particular operating system, graphical user interface, or network, your applications will also become bound to those technologies.

9 The Extensibility Rule

A hypermedia system must be easily extensible and contractible in functionality and user interface in order to solve wide classes of application problems and to accommodate users of all types.

It is impossible for those who build and sell software tools to anticipate all of the kinds of problems to which users will apply them. For example, an application- building system might be designed for creating standard data-processing applications such as order entry yet be unsuitable for computer-based training applications without the addition of new functions such as cross-matching, multiple-choice question types.

An extensible hypermedia system allows application builders to implement new system functions through a scripting language or an interactive dialog instead of through modification of the source code of the system itself, and to integrate these functions seamlessly into the system as if they had been there all along. Extensibility includes the abilities to add or delete items from standard system menus, to create custom menus, to hide or disable portions of or entire system menus, and to integrate new error messages and other communications into standard system-user advisories or dialogs.

The designers of the successful personal-computer spreadsheet programs recognized the importance of extensibility and provided facilities such as macros for the customization of menus, user interactions, and program functionality. This enabled spreadsheet programs to be applied to unanticipated classes of problems outside the sphere of standard financial modeling. Designers of hypermedia systems must provide similar extensibility in their products.

10 The Multilingual Rule

A hypermedia system must support user interaction, user communication, and data storage and retrieval in languages other than American English.

Some 60% of the users of the world's computer systems are not native English speakers. This percentage is continually growing as computer applications become more accessible to non-technically trained users. What's more, global networks, high-resolution graphical displays, and laser printers with large font libraries are continually driving this expansion.

English as a written language is remarkable for its simplicity in that there are no diacritical marks, no context sensitivity, a small character set, few ligatures, and a phonetic representation. The Roman characters of the English language are also used in other common languages such as French, German, Polish, Spanish, Norwegian, etc., but with significant differences among them in ligatures and diacritical marks.

The variety of alphabets and syllabaries involved in non-English use of computers is astounding. These include Russian and Ukrainian with their Cyrillic alphabets, Greek, Devanagari, Hebrew, Arabic, Thai, and Korean. Chinese and Japanese are also important but stand apart because of their complexity.[3]

Japanese language processing involves the use of Hiragana which is a syllabary capable of representing all the sounds of the language. Katakana, like Hiragana, is derived from Chinese, but is used for representing foreign words and for emphasis. Kanji consists of about 10,000 Chinese ideograms of which an educated adult may need to know about 2,500. The standard

[3] Japanese is one of the simplest languages phonetically while Chinese is very complex with up to eight tones per character.

computer representation for Japanese (JIS- X-0208) uses a 16-bit code. Thus software for this character representation requires two bytes per character and may necessitate changes to a computer's operating system.

A consideration of the multilingual problem in computing has led to the Unicode (for Unique Code) standard which also uses 16-bit encoding. Unicode defines alphabets for Arabic, Hebrew, Roman, Cyrillic, Greek, Japanese, Chinese, Korean, Symbols, and Han (including Hanzi, Kanji, and Hanja).

Users of hypermedia systems must be allowed to create, store, and manipulate text objects in multiple languages. In addition, user menus and other system-user communications such as error messages and documentation must be easily customizable and switch-selectable into these languages. For example, users should be able to communicate with their hypermedia software entirely in Kanji, Hiragana, and Katakana.

Multilingual hypermedia software reduces training time, accommodates larger user populations, and increases acceptance of the software by its users. Hypermedia systems that hope to be useful globally must either support a Unicode mode or offer special provisions for simultaneous use of several languages.

11 The Performance Support Rule

A hypermedia system must accommodate both novices and power users, allowing them to author visually and to debug user-friendly applications that can become self-contained electronic performance-support systems for their users. A hypermedia system must also be its own electronic performance-support system.

Complex, highly functional software systems can become quite inscrutable and inaccessible by all except highly trained users. If hypermedia systems are to achieve their promise, they must overcome these usability barriers. And there are several important requirements they must fulfill in doing so.

A hypermedia system must carry minimal prerequisites for its use. Users with only a basic knowledge of how to use a graphical user interface such as a Macintosh must be able to begin using the system and become productive in it without further formal training. This requirement entails some other requirements which are discussed below.

A hypermedia system must accommodate expert users as well as novices. Of course this means that the system must interact with its user through a graphical interface. But the system must also allow the user to perform a wide range of application- building tasks via visual authoring – the direct manipulation of multimedia application objects and their behavior through pull-down menus, dialog boxes, and other object manipulation such as rearranging screen layout with drag-and-drop or other techniques. And while being easy to use, the system should not restrict the expert's access to all the system functions he needs. For example, creating a link between two objects (see Rule 7) should be no more complicated than pointing at the two objects and then selecting a menu item to perform the link. Yet an expert user should be able to change one of an object's links by directly editing the link information on the object's property sheet instead of by the possibly more time- consuming though simpler process of visually deleting the link and then recreating it.

A hypermedia system should also employ multiple media in communicating with its user and not restrict use of multiple media to the applications developed by the user. For example, system error messages might include audio played back through a workstation loudspeaker in addition to error text appearing on the screen.[4] Some system messages might even be communicated through audio alone. For example, a "resource busy" advisory message might be communicated with a telephone-style busy signal on the loudspeaker.

If the user requires assistance with a system feature or procedure, a simple click of a key or the mouse should invoke that assistance on-line. The system should respond with the help that is exactly appropriate for the user's context and not force the user to locate the relevant material beginning with a top-level table of contents. Yet if the user desires to browse or search through the on-line reference material, the system should support a variety of search techniques and even allow the user to annotate with electronic yellow "post it" notes what he finds. And both the on-line help and on-line reference materials must be multimedia themselves. For example, a request for help in coping with a "printer is out of paper" message might invoke a video clip showing how to refill the printer's paper tray instead of the less-helpful message "put more paper in the printer".

A hypermedia system must also allow the user to create and maintain multiple contexts within the system. For example, the user must be able to invoke the on- line help facility in a scrollable window on the screen while concurrently working in another window with that part of the system for which the help was sought.

A helpful hypermedia system not only accommodates users of different experience levels but also acts as its own self-contained electronic performance system, obviating the necessity to consult reference material in hard-copy manuals or to obtain direct assistance from other humans.

My colleagues and I debated the name of this rule. Those who work more closely with new users favored phrases like "User Friendly" or "The Helping Hand Rule." Those who associate with advanced users favor the name selected. "Performance Support" was chosen because of the importance of electronic performance support systems [6] and because this name covers both class of users.

12 The Standards Rule

A hypermedia system must support and comply with all relevant formal and market standards.

A computing-industry pundit once opined that the nice thing about standards is that there are so many of them to choose from. Yet the computing world would be truly chaotic without them. Standards reduce the number of choices to a tractable number. In so doing, they allow product vendors to innovate and add truly useful features to their products instead

[4] Such a system might be acceptable in a private office but would be inappropriate in a software factory setting.

of devoting time to proprietary implementations of common functionality. And standards-adherence by vendors gives the consumers of their products the confidence that the products will be compatible with other products in those areas.

There are two kinds of standards. *Formal* standards are debated and agreed upon by formal standards organizations or industry consortia. For example, the International Standards Organization (ISO) over the years has published standards in a variety of areas. For example, ISO has defined 8-bit encodings for Roman character sets, the architecture and functionality of network protocol stacks, and the syntax and semantics of the SQL database-access language. Formal standards are published in standards documents and widely disseminated in the hope that the relevant vendors will incorporate them in their products.

Market standards undergo a different process. They become standards by virtue of being purchased and used in significant quantities. For example, the X Window System has become a market standard because all the important hardware manufacturers offer implementations of it and all the major software vendors now write their user interfaces to conform with it. Instead of being voted upon with a show of hands by a standards committee, market standards are determined by the vote of dollars in the marketplace.

Hypermedia systems must comply with and support all relevant formal and market standards. This will not only help different standards-compliant software products to interoperate but will also helps ensure their integration into your standards- compliant operating environments. Standards also simplify a vendor's problem of making their systems portable to a variety of platforms and environments.

Unfortunately, multimedia and hypermedia are still such a young set of technologies that the standards process for them has not yet run its course. However, a few standards such as JPEG, Unicode, MIDI, HyTime, and MPEG have emerged [10]. Such standards must be supported in hypermedia systems and not replaced by equivalent proprietary methods.

Conclusion

Industrial-strength applications are built today with tools such as 4GLs and forms packages. We believe that hypermedia systems as described herein can dramatically alter the process and speed of building such applications as well as the sophistication and usability of those applications.

We have offered twelve criteria by which to judge putative hypermedia systems, and these rules are summarized below.

Twelve Rules of Hypermedia Systems

1. The Multimedia Rule
2. The Object Rule
3. The Scripting Rule
4. The Multi-User Rule
5. The Scalability Rule
6. The Interoperability Rule
7. The Hyperlink/Hyperview Rule
8. The Technology Independence Rule
9. The Extensibility Rule
10. The Multilingual Rule
11. The Performance Support Rule
12. The Standards Rule

It is interesting to compare our rules with the work of Halasz [9] on NoteCards in which he lists seven fundamental issues for hypermedia. Our two approaches are very different since NoteCards is a system to help people organize their ideas. The users of NoteCards are assumed to be authors, designers, and researchers. On the other hand, we envision hypermedia systems as tools for building large industrial- strength performance-enhancing systems. All but one of the Halasz criteria are explicitly or implicitly covered in our own rules. The possible exception is his rule on "Virtual Structures". We have deliberately not focused on what might be called AI interfaces to or AI components of hypermedia systems. We agree that these ideas merit further research and have concluded that they are not yet ready for commercial use.

It will be of interest to see if users and vendors will agree to use these criteria. And we look forward to publication of comparative analyses of commercially available systems using these criteria.

Please feel free to contact us if you believe we have omitted something important or included something extraneous. The author's e-mail address is `harrison@CS.Berkeley.EDU`.

Acknowledgements

It is a great pleasure to acknowledge the significant contributions of Eugene Shklar. Special thanks to Denise Kiser for asking the right question at the right time.

References

[1] David H. Brandin and Michael A. Harrison. *The Technology War*. John Wiley and Sons, Inc., New York, New York, 1987.

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

[3] E. Jeffrey Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987. A more detailed version is available as Technical Report STP-356-86, Microelectroinics and Computer Technology Corporartion, Austin, TX, Dec. 1986.

[4] D. Engelbart and W. English. A research center for augmenting human intellect. In *Proceedings of FJCC*, pages 395–410, Montvale, NJ, 1968. American Federation of Information Processing Societies, AFIPS Press. Vol 33, No. 1.

[5] Douglas C. Engelbart. Authorship provisions in Augment. In *Proc. of the 1984 COMPCON Conference*, pages 465–472, San Francisco, CA, Feb. 27 – Mar. 1 1984.

[6] Gloria J. Gery. *Electronic Performance Support Systems*. Weingarten Publications, Boston, MA, 1991.

[7] Adele Goldberg. *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[8] Danny Goodman. *The Complete HyperCard 2.0 Handbook*. Bantam Books, New York, third edition, 1990.

[9] Frank G. Halasz, Thomas P. Moran, and Randall H. Trigg. NoteCards in a nutshell. In *Proc. of ACM SIGCHI+GI'87 Conference*, pages 45–52, Toronto, Canada, April 1987.

[10] Brian D. Markey. Emerging hypermedia standards. In *Proceedings of the Summer '91 USENIX Meeting*, pages 59–74, 1991.

[11] Norman Meyrowitz. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. In *Proc. of 1st ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 186–201, Portland, Oregan, September 1986. Published as *ACM SIGPLAN Notices*, 21(11), November 1986.

[12] Ted Nelson. *Literary Machines*. Self published, Swarthmore, PA, 1981.

[13] Jakob Nielsen. *Hypertext and Hypermedia*. Academic Press, Harcourt Brace Jovanovich, Boston, MA, 1990.

[14] Richard L. Phillips. Mediaview: An editable multimedia publishing system developed with an object-oriented toolkit. In *Proceedings of the Summer '91 USENIX Meeting*, pages 125–136, 1991.

[15] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 21(1):81–96, January 1987.