Crossjam: A Language for Authoring Hypermedia Scripts Qualifying Exam Proposal *

Brian M. Dennis †
University of California, Berkeley

Abstract

Hypermedia authoring systems are becoming a bottleneck in the computer industry. We have the infrastructure and incentive to deliver digital media. Authoring tools that support large scale reuse, abstraction, and full control of presentation are lacking.

For this proposal, authoring is broken down into three major areas: structural organization, interactive behavior, and graphical design. No current authoring system simultaneously supports clean formal specification of each of these elements. A well designed formal specification mechanism, e.g. a programming language, can enable high level authoring tools, document interchange, and program interoperability. I propose the design and implementation of a language, Crossjam, language as a mechanism for complete formal specification of hypermedia applications.

This work is novel in that Crossjam supports formal specification of all three aspects of hypermedia authoring by relying on a prototype-instance object model and a builtin constraint system. The programming style encouraged by this model should lead to a mostly declarative style of programming that is easily analyzable and synthesizable by high level tools. The object model also allows Crossjam to support structured document organization, presentation, and interaction.

This proposal also presents a research plan for designing and implementing the language. In addition, to demonstrate the benefits of Crossjam I will implement a prototype hypermedia authoring environment. The hope is that Crossjam will support new authoring metaphors, hypermedia presentation styles, interactive techniques, and construction tools.

*This research has been sponsored in part by the Advanced Research Projects Agency (ARPA) under Grant MDA972-92-J-1028, by the NSF under grant CDA-8722788, and an IBM Fellowship. The content of this paper does not necessarily reflect the position or the policy of the United States Government or IBM.

†xjam@cs.berkeley.edu

1 Introduction

Hypermedia authoring is becoming a bottleneck in the computer industry. The multimedia industry is making some progress in understanding how finished hypermedia documents and applications should appear and interact with the user. The actual process of building the documents and applications though is much less understood. Tools that encourage good hypermedia presentation and interaction, while supporting the construction process are practically nonexistent.

Three of the most popular ways of constructing hypermedia documents can illustrate some of the deficiencies of current hypermedia authoring. Let us take a look at authoring for the World Wide Web, using HyperCard to build hypermedia documents, and developing hypermedia through a general purpose language.

A popular means of building hypermedia documents is to use the World Wide Web's language HTML [8] to implement the document. Initially designed for the construction of distributed hypertext, HTML and the World Wide Web [8] have been coopted by many as a hypermedia delivery platform. While HTML has fine facilities for describing the structure of a hypermedia document it fails to provide adequate expression of the graphic design of a hypermedia document, an essential element of expression to digital artists. This is one reason why we see such efforts as Netscape's HTML extensions and Adobe's attempt to incorporate PDF into the World Wide Web framework. Control of presentation is important in any hypermedia system.

Another means to constructing hypermedia documents and applications is to use a commercial authoring package such as Apple's HyperCard [3], Macromind's Director, or Aimtech's IconAuthor. All of these applications provide productive development environments that even non-programmers can access. Unfortunately, all of the commercial offerings force the author into a rigid metaphor that the author has to mold his problem to. For example, there is no way to augment or alter the card/background/stack metaphor in HyperCard. This prevents authors from imposing their own logical organizations on the hypermedia application.

A third means of building hypermedia applications, is to use a general purpose programming language with extensions for multimedia content. Many multimedia designers are now taking the path of combining Visual Basic with multimedia extensions (provided by easily used VBXes) to build hypermedia applications. This allows for an increased flexibility from the programmers end and can lead to better execution performance in the application. However, this approach forsakes the favorable authoring environments of the commercial systems. In fact, this leaves hypermedia authoring completely in the hands of programmers. While programmers have much to contribute to the hypermedia authoring field, support has to be in place for such non-programmers as graphic designers, professional writers, educators, etc. To further enable hypermedia as a new medium, authoring tools have to reach out beyond professional programmers.

To get to the root of the problem I break authoring into three distinct areas. First, every hypermedia document has a logical organization that helps the author to manage the entire hypermedia document in a consistent fashion. HyperCard's card/background/stack metaphor is one way of dividing up hypermedia content into manageable units. This organization or structure has to be designed by the author. Second, the author has to develop the means by which the user can interact with the hypermedia application. Many HTML browsers allow users to follow links by simply clicking on the link's graphical representation. Advanced versions of HTML allow users browsing a document to fill in forms, who's information is then sent back to the author. Specifying how the application handles user gestures is a major part of the author's task. Finally, hypermedia authors design the presentation of the hypermedia content. This includes the graphical display of static digital elements and the temporal presentation of time based media. Authors relying on Visual Basic take responsibility for programming placement of digital images on the screen and playback of audio clips. *No hypermedia authoring system today simultaneously supports a formal specification mechanism for all three areas of the hypermedia design process.* Furthermore, I submit that all such systems have a model of hypermedia that can not be altered to incorporate such a mechanism.

Towards the aim of supporting the author's need to specify struture, behavior, and presentation, I propose the design and implementation of the language Crossjam. Crossjam is novel in that it presents a model of hypermedia applications that allows for formal specification of the structural, interactive, and presentation aspects of hypermedia documents and applications. The computational model of the language relies on a prototype-instance object system augmented with a constraint system. This basis has not been used in a hypermedia scripting language to date. The goal of this research is to demonstrate that a well designed hypermedia authoring language can support authoring environments with capabilities beyond what we have today. These environments can move authoring further towards art and away from engineering.

Why develop a new programming language instead of a stand-alone environment? One major reason is precision of expression. A programming language allows authors to *exactly* specify various aspects of the design of a hypermedia document. For example, it may be more accurate to write PostScript code to draw a Bezier curve according to a set of control points then it is to plot the points and draw the curve by hand. A programming language can also lead to dense representation of the author's intent. A few bytes of quickly input PostScript code can produce an elaborate page presentation which could take hours to render by hand. Finally, a programming language basis is a more flexible mechanism. Unless a stand-alone environment provides built-in splines (more complex curves built out of Bezier curves) drawing one becomes a tedious by-hand process. Combining PostScript's capability to draw curves with fundamental programming constructs allows users to not only easily draw splines but abstract, parameterize and reuse the code in future work. These benefits can be accrued to hypermedia authoring tasks as well.

However, many hypermedia authors do not want to be programmers per se. What can Crossjam do for the wider audience that participates in the hypermedia authoring process? A clean authoring language can enable high level synthesis tools that can make the language available to wider audiences. There are many text editors that can internalize SGML code and assist naive users in adding the correct markup. The authoring language can support notions of reuse that encourage the capture of presentation styles for hypermedia documents. An analogy would be Microsoft

PowerPoint which has templates for standard business slide presentations. Another argument for a formal language solution is that tools to operate on the language can come from any number of unanticipated sources. HTML can be generated automatically, presented by a number of browsers, and checked for consistency with simple scripting tools. This allows for tools that support, at the right level of complexity, professional web maintainers and unsophisticated web page authors.

1.1 Overview

The remainder of this proposal is organized as follows. First, I attempt to further motivate the need for this research by demonstrating real world problems within current hypermedia authoring systems. I also attempt to motivate this research by hypothesizing some advanced hypermedia applications that are beyond today's hypermedia authoring tools. From there, I discuss the preliminary design of Crossjam. A discussion of an authoring environment to demonstrate Crossjam's benefits follows. I then survey previous research related to this proposal. A reasearch plan is then presented outlining implementation goals and presenting a timetable of milestones. Finally, I conclude with a summary of the proposal and the research goals of the project.

2 Motivation

In this section, I will attempt to further motivate the need for an authoring language such as Crossjam. Before we move further though, I shall define some terminology that can often take on many meanings in common usage.

2.1 Defining Hypermedia

While there is no consensus on the exact constitution of hypertext [14, 27, 38] there is a broad understanding of the common elements. The basic definition of hypertext involves the non-linear navigation of information, moving control of the traversal from the author to the user. To illustrate, cinema is completely linear. While in the movie house the user cannot change the order of any of the frames. A book is somewhat non-linear. Footnotes link non-sequential portions of the text and of course the reader can always read the text in any order he pleases. Hypertext attempts to be *explicitly* non-linear. While there is some guidance as to how to follow the content, the user is constantly presented with choices for traversal. There are two crucial elements to building hypertext documents, *nodes* and *links*. Nodes are the application defined quanta of information in a hypertext system. Links are entities that connect two nodes together in some fashion. Links are traversed by the user to navigate the document and are often highlighted in the presentation of the document to alert the reader to their presence.

Similarly, *multimedia* is currently an overloaded term. For my purposes, multimedia will simply mean the use of multiple media for presenting information. In multimedia, digital video, sound, images, text, and animations contribute equally in the presentation of information. There is also another element which is important as well, the notion of interaction. Interaction implies that there is some means of communicating with the multimedia document. This takes multimedia documents from simple canned movies into a medium that is only expressible with a computing device (personal computer, settop box, personal digital assistant, kiosk, etc.).

Hypermedia is the union of hypertext and multimedia. The essence of hypermedia is to interconnect different forms of digital media in a fashion that allows the user to travese the information non-linearly. Video, sound, images, text, and animations can all appear in a node of a hypermedia document and may actually be nodes unto themselves. The interactive nature of both hypertext and multimedia documents blurs the line between documents and applications. Documents are typically thought of as being static entities that are only read and edited, yet an interactive hypermedia document could conceivably provide a visualizationi interface to a constantly changing dataset. Similarly, applications are thought of as monolithic programs for performing specific functions, yet an interactive hypermedia document is rarely specified as a program and tends to grow organically, possibly even as it is being browsed. For the purposes of this paper we will use *hypermedia document* and *hypermedia application* interchangeably.

Finally, an *authoring environment* is the set of tools that are used to create hypermedia documents. Clearly content capture and editing tools are one component of the authoring environment. Hypermedia editors that actually build the document are another part. Such editors might aid the author in the programming aspects of the task. For example, a hypermedia editor might provide a browsing interface to a content library so an author can know what material is available for inclusion in the document. The authoring environment can be thought of either as a tightly integrated environment or as a set of loosely coupled tools. From the Crossjam perspective, the language specification should be completely independent of any authoring environment, much like HTML is independent of any browser or editor.

2.2 Defining Scripting Languages

A common term in the literature is *scripting language*. The term scripting language is also overused and lacks specificity. Some usages refer to languages for high level programming of operating system services such as REXX, Perl [50], and sh. Others mean languages for specifying the interactive behavior of multimedia applications. Typical languages in this vein are HyperCard's HyperTalk [4], Macromind Director's Lingo [32], and GainMomentum's GEL [33]. A third camp looks at scripting languages as high level languages that can be easily embedded in applications to provide extensibility. This camp is often referring to such languages as Scheme [13], Tcl [39], and Visual Basic [34]. Finally, there is a fourth bailiwick which has coopted scripting languages for use in programming agents, entities for performing remote computations. Languages in this camp are again Tcl, Telescript, and Obliq [9].

To help avoid confusion with any of these camps, I will refer to Crossjam as a *script language*. Crossjam falls into none of the above camps. The language is for specifying organizational, presentation, and interactive aspects of a hypermedia application. While Crossjam's design is influenced by many of the above languages it's target audience is generally different. These languages do have features which are quite useful in a hypermedia context. Where possible Crossjam will try to recreate these features within its own model of hypermedia.

2.3 Problem Areas

While browsing the World Wide Web [7] I often encounter sets of pages that should semantically be a unit. For example, researchers often place their research papers online. To help possible readers there is usually a link to the abstract of the paper and then a link to the full document. The typical cycle when encountering such an abstract/paper combination is to: 1) follow the abstract link, 2) go back to the page containing both links, 3) follow the actual paper link. Suppose, that the HTML [8] (the World Wide Web's markup language) could actual indicate that the abstract and paper were one unit called an *abstracted paper*. More intelligent browsers might be able to understand this construct and actually provide the user with options to first read the abstract and then as the next step fetch the paper. Unfortunately, HTML has no such mechanism to express such a combination. Other systems have grouping mechanisms but these are generally for user interface elements and can in no way indicate the meaning of an author's grouping. For example, Toolbook has structured graphical objects that can be grouped and interacted with as a unit. In fact, this is one way to construct a slider graphical object out of boxes, lines, circles, text, etc. However, there is no way to indicate to the system that this object *is* a slider and distinguish it from another grouping with the same types of elements. In language design terms, what we desire here is user-defined types.

In multimedia authoring, graphical design of the document is important yet there is little to no support for formally specifying it. Netscape Communication Corporation's WWW browser [36] is an extremely successful hypermedia browsing tool. Part of the program's phenomenal success is the fact that it is free but it is also successful because it supports author control of presentation. The Netscape browser also can display HTML augmented with non-standard Netscape extensions. For example, there are a number of annotations for specifying center, left, and right alignment of text. There's even an oft maligned tag for specifying blinking text. Many authors are taking advantage of these tags despite their HTML's non-compliance with the standard. This is a clear manifestation of authors' desires to control the presentation of their document, something HTML explicitly tries to limit.

HyperCard and it's successors have good support for designing presentation through direct manipulation. Adding a colored box to a stack is a simple point, click and drag operation. Developing a complex graphical object is fairly difficult. I had occasion to use HyperTalk to draw a histogram from some data. This was more difficult than I expected because I was required by HyperTalk to implement the drawing as a sequence of user interface actions. For example, to draw a line in HyperTalk one has to select a tool, perform a click operation, and then a drag. While

HyperTalk has syntax to support such operations, the benefits of programming are lost under such circumstances.

While graphical design of documents is important, how to specify graphical presentation is unexplored territory. The HTML community is attempting to address the issue of *style sheets* with DSSSL-Lite [12]. DSSSL-Lite is a reduced version of DSSSL [19], the proposed presentation and transformation language for general SGML documents. DSSSL-Lite is a language for transmitting to an HTML browser how a document is to be presented. DSSSL-Lite's runtime model is unspecified at this point. This leaves unclear what the exact results will be from browser to browser. In a similar vein, it is unclear how to achieve certain common effects using DSSSL-Lite. Two examples are combining user presentation preferences with author presentation preferences and customized presentation for singleton elements of a hypermedia document. Clearly, there is work that can be done in this area. Crossjam will attempt to address some of the issues here.

Current hypermedia applications and titles offer interaction that was state of the art in the early 1980s. One can follow links. That's almost the complete extent of interaction. HTML forms and HyperCard text fields provide a somewhat more sophisticated data entry mechanism. There are user interface hooks for such things as buttons, text entry fields, forms, etc. but little to no connection of these elements to the real content of the hypermedia document. Many pure hypertext systems allow for little more than the addition of links into the web. Can hypemedia systems support more complex interactions robustly? I believe a programming language basis such as Crossjam can support such interaction.

A major problem of many of the current so-called hypermedia tools is that they do not help the author manage the authoring process. HyperCard is often cited as a hypertext system. However, it is nearly impossible to find all of the links in a hypertext application built with HyperCard. The major failing is that HyperCard has no explicit notation for the links. Linking is essentially an artifact of navigation. Link management is an important facet of hypermedia authoring yet this system has no support for managing links. A typical example of the need for link management is to find dangling links in a hypermedia work in progress.

An even more vexing problem is the lack of a formally specified hypermedia document format in these systems. In HyperCard, since there is no concept of a formal description of the document it is impossible to write an external tool to interface to the HyperCard playback engine. This eliminates the construction of small special purpose tools built independently of HyperCard. Such tools might be useful for the batch processing and conversion of large input sets to a hypermedia format. For example, it would be interesting to have a tool that could analyze the statistics of a season's worth of major league baseball and generate a hypermedia document. Writing such a tool might be more convenient in a language such as Common Lisp, C, or Visual Basic. These general purpose languages have much better support for data structures and input/output than typical multimedia scripting languages. Presuming a formal specification, an experienced programmer could easily construct a tool to synthesize hypermedia documents. A hypermedia authoring language such as Crossjam can be the target of such synthesis.

2.4 Visionary Capabilities

Now that I have outlined some of the deficiencies in today's authoring environments I would like to present some capabilities that have yet to be seen in these systems.

One concept would be that of *trailblazing*. Assume that one user explored a web and found a particular sequence of links and nodes illuminating. If we could somehow capture the notion of that sequence as a *trail* users could interact with each other by constructing and exchanging trails for particular webs. If the trails could be reincorporated into the web, users could choose to follow particular trails based on who built the trail. There are a number of issues that need to be solved to support this concept. How are trails captured, specified, and transmitted? Can trails be easily incorporated back into the original document? How do users interact with trails?

In a simiar vein, I would like to see the concept of a *guided tour* become better supported. A guided tour would be similar to a trail except that the author can also specify and constrain the interaction with the hypermedia web. This allows the author to present the information of the trail in a fashion guaranteed to be coherent. Current authoring systems provide enough mechanism to build such guided tours, but they have no authoring framework for doing so. The author has to build all of the abstractions from scratch and has little to no support for enforcing the constraints of the tour. Hypermedia webs with guided tours would be particularly beneficial in the area of computer based training.

Authoring systems need to help take the drudgery out of hypermedia web construction. When dealing with large or continously generated input sets, the authoring system should assist the author by automatically generating portions of the web. For example, the New York Times generates daily a fax of their front page that is distributed on the World Wide Web. However, this fax is not crosslinked to any of the information in the New York Times' extensive databases of articles and photographs. I suspect that this is because there is no good way to automatically analyze the document, synthesize a specification of the links, and incorporate them into a polished hypermedia presentation. We need to provide a framework in which such machine aided generation is possible. Such support would allow rapid incorporation of more information, usefully organized, into a hypermedia web.

3 Crossjam: Language Design

3.1 Overview

Crossjam is specifically designed to support the three major authoring areas described earlier: structure, design, and interaction. The basis of the language will be a prototype-instance object system that will be consistently applied to the problems of hypermedia authoring. Combined with this object system will be a built in constraint system. Another small handful of advanced language features will be included to ease the task of authoring. This design will attempt to use a small set of language features to solve a wide set of related problems in a consistent fashion.

There are seven major goals of this design:

- To provide a formal mechanism for describing the structure, presentation, and behavior of hypermedia documents.
- To provide an easily analyzable and synthesizable formal specification for hypermedia documents.
- To make explicit in the formal specification, the link types, link instances, node types, and node instances.
- To allow the author to leave unspecified, aspects of the presentation that can be handled generically.
- To allow the author the capability to precisely specify the presentation of the document if need be.
- To allow the author to design interaction based on, but not constrained to, the presentation of the web.
- To allow the author to abstract organizational templates, styles of presentation, and libraries of interactive behavior.

Crossjam will be based on Lisp like languages such as Common Lisp and Scheme. The language will incorporate latent type checking, first class anonymous closures, lexical scoping and unlimited object extent. The Crossjam object system is currently modeled after KR [20], the object system underlying the Garnet [35] user interface system. There are a number of experimental object systems similar to KR but this system is small, easy to implement, and supports a number of the design goals of Crossjam.

Lisp like languages have some noted advantages that make their use as a basis for Crossjam inviting. First and foremost, Lisp has shown itself to be a particularly malleable language. The development of powerful and clean macro systems in Lisp implementations make it easy to embed sublanguages in Lisp. Such facilities have been used to construct object oriented, declarative, and contstraint sublanguages within Lisp. Lisps typically have simple LL(1) syntaxes supporting easy analysis and synthesis by authoring tools.

3.2 Runtime Model

Before delving into the core of the design it should be useful to have a clear picture of the runtime execution of a Crossjam script. There is one major runtime entity in existence at all times, the graph of nodes that make up the *runtime web*. Objects in the runtime web are built out of the Crossjam object system. The nature of Crossjam's object system allows the web to be defined in a mostly declarative manner. The runtime web is essentially defined by a sequence of node instantiations.

At any given time, the runtime web is augmented by a set of *presentation spaces*. A presentation space contains a set of objects instantiated to represent the display of a set of runtime web nodes. These presentation objects are generated as a part of a transformation process controlled by a *presentation schema* defined by the author. Each presentation space is a three dimensional finite coordinate system. Two dimensions represent the graphical rendering while the third dimension is for the time oriented media. Underlying each presentation space is a *presentation engine* responsible for coordinating the rendering of the presentation objects. The presentation engine relies heavily on the built in constraint system to handle coordination of time dependent media.

Along with each presentation space is an accompanying *interaction space*. Each interaction space is a two dimensional space overlaying a presentation space. In each interaction space are a number of regions, possibly non-rectangular and overlapping, which describe regions of interaction. Similar to a presentation schema, the instantiation of interaction regions are controlled by an *interaction schema*. The constraint system is relied on here to maintain coordination between the rendering and the interaction. The presentation engine is also responsible for correctly propogating user events.

Runtime webs, presentation spaces, and interaction spaces will all have explicit manifestations at runtime to allow interaction code to completely query the state of a Crossjam application. These manifestations will fit into the same object system used by the rest of the language. Similarly, presentation schemas and interaction schemas are simply Crossjam objects. This provides authors with one uniform data abstraction mechanism.

3.3 Structural Specification

The structural portion of the language addresses defining the abstract organization of the web structure and actually instantiating a web. Structure in this context means the ability to define classes of composite objects which have enforced constraints on the composition. Objects of a given class are distinguishable from objects of other classes. The objects of the class *may* also share behavior to support consistency. For example, Crossjam should support the ability to define a *leaf node* class which is a node containing only content references. Initial leaf node objects satisfy this constraint. Changes to such objects which invalidate the constraint are signalled as errors. Any leaf node object can be distinguished by a `leaf-node?` predicate.

A prototype based system has no concept of classes but Chambers et. al. [10] describe how to build similar *trait* objects that encapsulate shared behavior. A trait object serves as the repository for shared elements for a set of objects. For example, the code for the `leaf-node?` predicate for all leaf nodes would reside in their shared trait object. Also, code to check that mutations to leaf nodes satisfy the content only constraint would reside in the leaf node trait object.

The basic objects used to build structure will be *content*, *nodes*, and *links*. All media content is specified by referencing a conceptual *content database*. Every node consists of a set of *locations* and a set of links. Within a node all of the locations and links are uniquely identified. A link contains one *anchor* location and one or more *target* locations. A location is a reference to one of three possibilities: content, a complete node, or a subpart of a node. Built-in prototype and trait objects will exist for all of these primitive types.

While the basic object system provides enough support to actually specify trait objects, it would be beneficial to have some syntactic support for extending the trait hierarchy. Designing a mechanism for compact and clear definition of trait objects will be essential for Crossjam. KR already has a simple and flexible type system that can define constraints on slot values which are checked at runtime. Crossjam will extend this mechanism to support constraints on the complete structure of a node.

Once the node classes of a Crossjam document have been established, they must be instantiated to realize the runtime web. In a prototype-instance system, objects are instantiated by making a copy of another object where the copy operation allows for overriding the value of specific slots. In the KR model, any slots that are *not* overriden are shared with the parent. To create a leaf node object we simply invoke the `copy` operation on the prototypical leaf object. This leaf node will look like the original unless or until we override some of its slots with new content references.

The benefits of this structural model are many. First, it is simple to create new objects. The `copy` operation is consistently used to create new objects of any class. Second, any already constructed object can serve as a template. Particularly idiomatic singleton objects can be used as trait objects themselves. If we construct a particularly useful leaf node instance, for example, we can single it out and turn it into a trait object. Since object creation defaults to sharing from the original, authoring can be done simply be describing the differences between objects. Suppose leaf nodes contain pictures, among many other things, and we wish to create a new one only with a different picture. We simply copy a leaf node we already have and in that operation specify the new picture. This model also supports constraining web connection without resorting to a grammar based description. SGML based descriptions essentially describe the form of the web with a BNF-like grammar, the DTD. Actually checking the web for conformance involves full-scale parsing techniques. These techniques are only now becoming incremental which is important in an authoring environment. The Crossjam model supports webs that incrementally maintain their consistency and which do not have to be parsed in any fashion to do so.

3.4 Presentation Specification

The presentation specification subportion of Crossjam is concerned with describing the mapping from the runtime web to the actual presentation of the content. Presentation schemas are objects which know how to transform sets of runtime web nodes to presentation objects in a presentation space. These schemas serve the same purposes as DSSSL-Lite style sheets. This transformation basically instantiates objects in the presentation space along with constraints to control the further rendering of those presentation objects. The presentation schema is free to make any mapping it chooses from the runtime web to the presentation space. Applying a presentation schema to a given node set is performed by setting the distinguished `presentation-schema` slot of a presentation-space. A presentation schema may only apply to a node set that satisfies a given set of constraints and can signal an error of the given node set is unsatisfactory. For example, a simple presentation schema might only understand leaf nodes. Recall from the previous section that such nodes had content only. The schema might turn all of the text of leaf nodes into renderable text objects in the presentation space. On the other hand, it might generate icons for any video clips that appear in the node.

Once the presentation objects have been initialized, the *presentation engine* constantly examines the state of the constraint system and renders the specified content. A particularly important constraint is how long the presentation space is in the temporal dimension. There is a distinguished constraint result, called the *presentation time* that the rendering engine uses to determine how long the presentation should run. The results of this constraint can be zero, a finite amount of time, or an infinite amount of time. The rendering engine divides up the presentation time into individual frames and then sequences through the frames until completion of the presentation time. The presentation time is maintained in the presentation constraints for control by interactive handlers. A crucuial point is that the presentation time is just another constraint that is specified by the presentation schema. Harkening back to our leaf node example, at first the presentation schema would set the presentation time to 0 since it was rendering video as icons. However, once a request to play a particular video was received, schema code could change the presentation time to be the length of the video. This change would also force constraints to fire which would change how the video is actually rendered, presumably presenting the actual frames over the course of the presentation time.

We must address how exactly presentation objects are actually rendered. The presentation engine itself manages *when* each presentation object will be rendered over the presentation time. It will use typical techniques graphical interfaces to maintain redraw information and inform the appropriate objects when necessary. Each presentation object will understand a `render` message. A large library of useful rendering functions will be built in to Crossjam, but authors should be able to compose these functions to define new means of rendering content. Rendering operations will provide similar functionality to those available in the PostScript [1] page description language.

When do presentation spaces come into existence and when are presentation schemas applied? Similar to the runtime web, the initial presentation spaces are simply instantiated by copying prototypical presentation space objects and setting their `schema` and `node` slots.

To support generic presentation of webs, the author can leave the `schema` slot of a presentation space unspecified. This signals to the presentation space to find a presentation schema based on some *presentation selector*. I expect that experimenting with presentation selectors will be an interesting area of research. One obvious mechanism is to have a presentation schema that can present any set of web nodes and have the selector return that schema. More interestingly would be mechanisms similar to those used to resolve multi-method dispatch in object systems like CLOS [44] and Dylan [5]. The selector could have access to all available presentation schemas, rank them according to some critieria, and select the "best" one for the particular set of nodes. For example, the user might have default schemas which require hardware unavailable at presentation time. The runtime engine for the Crossjam script could fall back to a simpler schema which makes presentation possible. Finding the right mix of generality and intuitive behavior for presentation selection is an interesting question.

This presentation model meets the goals of allowing the author to control the graphical rendering of a web with as little or as much control as needed. As Garnet has demonstrated, constraints are an effective means of expressing much of the presentation task. For example, defining shadow boxes becomes much easier with constraints. We only have to define two box objects and a simple constraint defining the offset relation between the two. This constraint model has not been seen in any hypermedia system with language-based style mechanisms as of yet. I have outlined possible beginnings of how to use constraints for time based media but even Garnet did not explore this territory. It will be interesting to see if constraints can be an effective in helping to specify the presentation of such media.

3.5 Interaction Specification

The basis of interaction is a standard event driven model similar to such systems as HyperCard and Multimedia Toolbook. Conceptually, while various portions of the runtime web are being rendered, events will occur that need to be responded to. Typical of these systems, the author can be notified of keyboard and mouse events along with a platform specific set of system events that are messages from the operating system. These system events might be idle events, indicating a set amount of idle time has passed, or interprocess communication events indicating a message from another process. Interaction specification involves defining what events an interactive document is actually interested in, and registering that interest so that appropriate notification is received.

The author specifies interest in given regions of the presentation space by instantiating *interaction objects* in the interaction space. Similar to the presentation schema, a Crossjam script will have interaction schemas that define how to map presentation objects into interaction objects. Interaction schemas are applied to an interaction space by setting the `interaction-schema` slot of the space to the schema. Interaction schemas derive for interaction all of the benefits that presentation schemas did for presentation.

Interaction spaces are essentially a two dimensional coordinate system of the same size as the presentation space it's derived from. Within an interaction space is a set of *interaction regions*, which define areas of the presentation in which the application is interested catching events. An interaction region might correspond to the bounding box of a hotword in a hypertext. Crossjam event matching is taken from the NeWS [22] user interface system with appropriate modifications for the Crossjam object system. Within each interaction region is a set of *event templates* or *interests*. When an event is found to have occured in a given interaction region the event is checked against the interests within the region. An event matches an interest if each slot of the event matches each slot of the interest. An interest slot can be one of the null object, a single object, or a collection of objects. An interest slot containing a null object matches any value in the event slot. Singleton interest slots are a match only if the object is the same in the event. An object collection slot matches if the object in the event slot is in the collection. Interest in events is registered by simply copying prototype event instances, modifying the slots for the appropriate matching, and entering the interest object into the appropriate interaction region. For example, going back to our hotword example, to register interest in any mouse click, the author simply copies the prototypical mouse press object, inserts null into the slot distinguishing the button, and adds the copy to the interaction region. The process is similar for interest in only a left button except that the key for left clicks is placed in the distinguishing slot. This event model is simple, flexible, and fits cleanly into the prototype-instance model.

The interaction objects of an interaction space are ordered for event handling purposes. Thus, interaction regions can overlap but event matching will proceed along this ordering from least to greatest. Alternatively, the interaction space can be thought of as a two and a half dimensional space where each interaction region exists on a plane unto itself. Event matching then proceeds from the front of the space to the back. Crossjam will also provide a mechanism whereby interests can be specified for events that fail to fall into any interaction region or fail to match in any interaction region.

Once a handler has been found, it needs to take some action. Standard imperative language features are provided by our Lisp basis. Within the actual event object will be information such as which presentation space the object the event occured in, the event location within presentation space, the accompanying interaction space, and event specific details.

From the event, the handler should be able to trace back to parts of the web if need be for inspection and or modification.

Interaction might require intimate connection to how content is actually presented. In particular, when a web node is mapped to a graphical region, same region is used to define interaction with that node. The bounding box of our example hotword should only cover the pixels of the rendered word, even if the word changes position or size. The interaction specification has to support connection to the presentation transformations. At the same time, interaction regions are not always one-to-one to a given content object. For example, if some content text is one node, there may be multiple hotwords in that text. Having mouse presses map only to the nodes entire region leads to having an extra hand coded stage of event lookup to see if the event landed in a hotword. This is inconvenient and errorprone. Allowing for mappings that are not one-to-one is supported by separating presentation and interaction space. The constraint system underlying Crossjam supports the maintenance of presentation and interaction connections. The contstraint system allows interaction transformations to specify that interest region shapes can be constrained to be those of given presentation entities. So after a font change the bounding box of our example hotword changes. The correspondence between the two is automatically maintained by our constraint system. Meanwhile, there can still be multiple hotwords for one chunk of text.

This model of interaction supports the design goals in a flexible, extensible, consistent, manner. The mechanism uses the prototype-instance object model in a fashion consistent with the rest of Crossjam. The interaction model also recycles the concept of schemas and transformation to gain synergy with the presentation model. Interaction can be tied to presentation but is not forced to be so.

4 Crossjam: Authoring Environment

A significant portion of this proposed research is in the construction of a prototype authoring environment to demonstrate some of the benefits of the Crossjam language. Since Crossjam scripts are in essence programs the authoring environment will significantly resemble an integrated software development environment. Research ideas from the software development community will contribute in the construction of an environment that deals well with structured documents.

The anticipated environment will be a multiple view system, having displays of both the structural form of the hypermedia document and a WYSIWYG view. The former aids in organizing the web while the latter is essential for good graphic design. The environment will also support other essential elements of the development process including: content management, link management, and consistency checking.

I also anticipate developing a small set of standalone tools that should be able to operate independently of the integrated environment. This will demonstrate how an independent programming language specification can serve as a means of program interchange. One such tool might traverse a Crossjam document and catalog all of the content used. This might be useful for copyright services. Returning to my New York Times example, developing a standalone application to take the daily (hourly?) output of some service and construct a Crossjam document should be possible. For example, automatically generating hypermedia documents from freely available weather information should be possible. The resultant document should be connected to all previous documents generated and should be editable in the integrated environment. The multimedia quality and level of integration into a web will be an interesting metric to measure such a tool by.

5 Related Work

5.1 HyperMedia Systems

The Intermedia [51] system was possibly the first instance of a full hypermedia system, incorporating digital images and sound into a hypertext context. Intermedia was developed in the early to mid 1980s at Brown University's Institute for Research in Information and Scholarship. The research project was mostly aimed at identifying the right architecture for building such hypermedia systems. Authoring was not distinguished from browsing. Supporting the actual authoring process did not seem to be one of Intermedia's research goals. Intermedia points at some of the potential benefits of hypermedia documents.

The World Wide Web has been a spectacularly successful demonstration of the capabilities of hypermedia. A large part of the World Wide Web's popularity is based upon a simple language, HTML, that provides structure for hypermedia documents. HTML is based on SGML [21] which is a specification of markup language for structured documents. Both SGML and HTML explicitly try and separate document structure from document presentation. While Crossjam attempts to avoid some of the deficiencies of HTML, the case has clearly been made that a formal markup language has great benefits for hypermedia. In particular, HTML demonstrates that explicit structure and explicit well defined links, can enable a number of hypermedia tools.

Hyper-G [2] is a parallel effort to the World Wide Web in attempting to deal with distributed hypermedia. Hyper-G's goal is to support large-scale, distributed, multi-user, hypermedia information systems. Hyper-G does have a markup language underlying the system[30] however, the author's point out that Hyper-G is a document *dissemination* system as opposed to a document preparation. Thus I must conclude that hypermedia authoring is not one of their research goals.

Hardman et. al. [25, 26, 43] at CWI in the Netherlands have developed a model of hypermedia that has similar goals to Crossjam. The Amsterdam Hypermedia Model is an attempt to merge the Dexter [23] hypertext model with the author's CMIFed model of multimedia presentation. The Amsterdam model is a high level description of the structures needed to support the combined hypertext and multimedia model. The authors rely on HyTime, described below, as the formal specification mechanism of their model. CMIFed, the authoring system based on their model, has interesting support for structural editing and temporal media presentation management.

HyTime [37] is another language design for structured hypermedia documents. HyTime like HTML is based on SGML. The HyTime design attempts to be much more comprehensive than HTML and actually delves into specification of the presentation of hypermedia documents. The language though, makes no attempt at formalizing how documents are interacted with. Also, HyTime is extremely general and complex trying to standardize all aspects of hypermedia documents in an interchangeable manner. This complexity may be contributing to the fact that there are no editing environments for HyTime to date.

5.2 Multimedia Systems

There are a number of multimedia authoring systems that are related to Crossjam. Apple's HyperCard and HyperTalk are venerable and classic tools for constructing hypermedia applications. HyperCard has propogated one of the dominant authoring metaphors (card/stack) in the industry. Descendants of HyperCard include Alliant's SuperCard, Sybase's GainMomentum, Oracle's MediaObjects, and Asymetrix's Toolbook. Similarly, Macromedia Director and Lingo have been used to implement a large number of multimedia systems. Director's timelines are another popular metaphor. Timelines successfully take into account the dynamic nature of multimedia systems. All of these systems are monolithic in nature and have little to no support for the linking portion of hypertext. They are relevant to hypermedia in that they have been used to construct hypermedia systems and corpuses.

Kaleida Labs has introduced ScriptX [28, 29] as a multimedia scripting language. ScriptX is a "second generation" object oriented language and library for developing cross-platform interactive multimedia. Essentially ScriptX is a class based object language based on generic functions with an extensive class library oriented towards multimedia. The ScriptX runtime, the Kaleida Media Player, has interesting implementation support for multimedia. ScriptX is touted as a basis on which to build hypermedia authoring tools. Similar to the monolithic multimedia authoring systems, ScriptX is not explicitly designed to support hypertext linking.

5.3 Hypertext Systems

The Symbolics Document Examiner [47] was a documentation system built by the now defunct Symbolics Corp. The Document Examiner effort is extremely relevant to Crossjam in that it was one of the attempts to understand authoring as much as browsing. The system's accompanying editor, Concordia [48, 49] demonstrated the need for high level tools, structuring, and abstraction in building large scale hypertexts.

NoteCards [24] was an early hypertext system developed at Xerox Parc. The system was closely coupled with the Lisp development environment it was built in leading to a high level of flexibility and customization. NoteCards was also had an extremely simple model based on cards (nodes), links, browsers, combining structural viewing and editing, and fileboxes for hierarchical organization. The system has been examined thoroughly in the literature providing a number of insights into building hypertext systems.

5.4 Programming Language Design and Implementation

This proposal has been influenced by a number of general purpose language designs. The largest contribution has probably been from the Common Lisp [44] and Scheme [13] communities. Almost all of the underlying concepts in Crossjam have been implemented in some dialect of Lisp at one point in time. Crossjam is the first combination of this particular set of features to support hypermedia. A number of object oriented language ideas have been contributed from SELF [46], Dylan [5], and ScriptX. SELF along with the Garnet user interface system has illustrated the potential of the prototype-instance model.

SGML [21] illustrates many of the desirable properties of a language for specifying structure. Since SGML is actually a meta-syntax for describing actual syntaxes, Crossjam might serve as an alternative to HTML and HyTime as a design that could be built on top of SGML. Crossjam is richer than HTML yet simpler and more consistent than HyTime. Crossjam also has models interaction which neither of these two standards attempts to provide.

5.5 Editing Environments

There have been a number of structure based editing environements whose design will influence the Crossjam authoring environment. Chen and Harrison [11] discuss structured editing in the context of the LaTeX [31] formatting system and illustrate many benefits of such an editing style. Concordia and CMIFed are actual implementations that have already been discussed.

There is a large body of work in relation to editing of programs since programs are highly structured documents. The Cornell Synthesizer [42], Arcadia [45], Pan [6], and Ensemble [16] have been major efforts in this area. These systems have had little or no hypermedia emphasis. They do, however, demonstrate many of the desirable features of a structured editing basis. My long term involvement with the Ensemble project has contributed many ideas to this proposal including: the value of structured document editing the concept of presentation schemas, similarly interaction schemas, and coordination of presentation and interaction.

In a similar vein, there are a large number of editors oriented around editing general documents structured with SGML and similar markup languages. Grif [40], Quill, and Rita [15] were the initial research attempts into this area. There are now a number of commercial vendors supporting products in this area.

Of particular interest is Interleaf Inc.'s, Interleaf editing system. English et. al. [17] describe a structured document system based on a dialect of Lisp with a built in object system. This object system is class based as opposed to Crossjam's prototype-instance style. While this system may seem quite similar to Crossjam, the Interleaf language is more a feature of the authoring environment than a means for specifying hypermedia applications. In short, Interleaf is a programmable editor for generating primarily paper documents. Also, both Grif [41] and Interleaf [18] point the way towards support for incorporating more interaction into hypermedia documents.

Another set of commercial offerings, related to HTML authoring, are relevant to this proposal including: SoftQuad's HotMetal, SGI's WebForce, and Pages Software's WebPages. Many of these systems attempt to incorporate such authoring features as style sheets and web structuring in an ad hoc manner demonstrating the desire for such features but not the formal support.

6 Research Plan

Clearly, since I propose to design and implement a full programming language, there are a number of pitfalls to be avoided. In particularly, there is the question of whether or not one person can build, in a reasonable amount of time, the system I have described. I hope to allay those fears here. First, I describe an implementation architecture for building Crossjam. This is followed by a schedule for building the system and completing the dissertation work.

6.1 Implementation Architecture

Implementing Crossjam consists of building four essential parts: a content database, a playback engine, an authoring environment and scripts.

The first and simplest is the *content database*. The content database is a collection of the captured media that could appear in the title. Expected media types are digital video, sound, images, animations, and structured graphics. This element of the architecture is called a database but could be implemented either by organizing a set of files on a filesystem or by actually using a database management system.

The *playback engine* is a crucial part of the system but not the focus of the proposed research. This element essentially implements the hypermedia runtime system of the Crossjam language. Low level multimedia presentation services, interfaces to the graphical user interface, the Crossjam object system, and the constraint maintenance system are all implemented in this module.

An *authoring system* is the development environment that is used to construct a script and populate a content database to be executed by the playback engine. The authoring system can be either a collection of small tools that interoperate well, or a monolithic integrated environment.

The *script* is the actual programmatic specification of the hypermedia application. For this architecture, the script is an instance of a Crossjam document. The script describes the logical organization of the content, the links that connect logical elements, the applications graphical design, its temporal behavior, and the potential ways endusers can interact with the hypermedia web.

To implement Crossjam in a reasonable time frame, I expect to embed the language in a Common Lisp implementation. The macro system of Common Lisp allows me to easily build the syntactic constructs of Crossjam. KR is already implemented in Common Lisp. A number of object stores for Common Lisp have been implemented while interfacing to a non Common Lisp database should be quite easy. To further ease the implementation task, the authoring environment can be built in Common Lisp. The only significant non-dissertation implementation is augmenting some Lisp implementation with multimedia features.

There are a number of freely available Common Lisps available but I am currently in discussion with Franz Inc. regarding software support and source language agreements for their Allegro Common Lisp products. If developed on the Windows platform Crossjam Allegro Common Lisp has hooks into Microsoft's Multimedia Device Interface. This implementation should provide a reliable, robust basis to start from.

6.2 Implementation Schedule

In the extremely short term, I will be examining the Allegro Common Lisp source to evaluate its suitability to the task. In parallel with this I will be searching for an object store or multimedia database for use as a content database. I hope to have the evaluation process completed within a month of the qualifying exam. Metering the system to understand and extend its peformance will take place at this time as well.

Once the basic infrastructure implementation has been completed I can move on to finishing the design and implementation of Crossjam. This comprises both completing the specification and embedding Crossjam into the selected Common Lisp implementation. The goal is to be able to hand write scripts that can be executed. At that point, I expect to be able to demonstrate hypermedia applications, specified by Crossjam scripts and incorporating all of the digital media along with significant interaction.

From there, concentration will mainly focus on implementing the authoring environment as a catalyst for understanding the language. I forsee working on the authoring environment to feedback into the design of the script language since one of the goals of this research is to produce a system with synergy between these two elements. This environment will be implemented in stock Common Lisp. I also hope to implement a few small self contained tools in another language, probably C++, to demonstrate the ease of building synthesizing tools. This initial system should also be able to demonstrate authoring in the current common metaphors such as stack based and flowchart authoring. The system should at that time be ready for a public release.

The project will enter its final phase when it is publicly released for evaluation and feedback from actual hypermedia authors. In this phase, I will attempt to demonstrate the benefits of the Crossjam architecture by building applications which exhibit the visionary capabilities outlined earlier.

An aggressive implementation effort should lead to a public release of the language specification and implementation early in the summer of 1996. An optimistic view would have authoring environment construction beginning sometime in the spring of 1996 and continuing through the summer. Presuming feedback from the user community does not reveal major deficiencies the dissertation could be completed as early as September 1996. A slip in the schedule should lead to at worst a finishing date late in the fall semester of 1996.

Although I feel that this project is rich in research potential, I see the dissertation nearing completion when I can demonstrate the following:

- A HyperCard style authoring tool. This would demonstrate that Crossjam is capable of subsuming a popular standard metaphor. This authoring tool should also demonstrate structure based authoring.

- Demonstration of a new authoring metaphor. This new metaphor should be abstracted and reusable taking advantage of Crossjam's structural support and presentation styles.

- Demonstration of automatic script generation.

- A publicly available release of the language implementation.

- A non-trivial user community interested in advancing the design and implementation of Crossjam.

7 Conclusion

This dissertation proposal outlined the initial design of Crossjam, a language for specifying the structural, graphical, and interactive aspects of hypermedia documents. Crossjam is motivated by a number of deficiencies in hypermedia authoring tools. Some of these deficiencies include: lack of organizational flexibility, inability to fully control graphical design, and the inability to interoperate with independent tools. Crossjam is novel in allowing formal specification of all aspects of a hypermedia document. The language uses a prototype-instance object model combined with a constraint system to gain a number of desirable features including: simplicity, reuse, a declarative style, and programming by difference. Such design goals as constraining hypermedia web structure, providing many levels of presentation control, and separating presentation and interaction were achieved on the basis of this model.

A research plan was presented outlining an implementation path for Crossjam. The implementation architecture is based on a simple four part model involving a content database, a playback engine, authoring environment, and actual Crossjam scripts. The implementation will essentially be a modification of an existing Common Lisp implementation. The implementation will be augmented with multimedia support, Crossjam syntax, and Crossjam primitives. Authoring tools will also be implemented to demonstrate and evaluate the Crossjam language. The research plan aims for completion of the project by August 1996.

References

[1] Adobe Systems Inc. *PostScript Language Reference Manual*, 2nd edition, 1990.

[2] K. Andrews and F. Kappe. Soaring through hyperspace: a snapshot of Hyper-G and its Harmony client. In Wolfgang Herzner and Frank Kappe, editors, *Multimedia/Hypermedia in Open Distributed Environments*, pages 181–191. Springer-Verlag, 1994.

[3] Apple Computer Inc., Cupertino, CA. *HyperCard, Reference Manual*, 2.2 edition, 1993.

[4] Apple Computer Inc., Cupertino, CA. *HyperCard, Script Language Guide*, 2.2 edition, 1993.

[5] Apple Computer Inc. *Dylan Interim Reference Manual*, June 1994.

[6] Robert A. Ballance, Susan L. Graham, and Michael L. Van De Vanter. The Pan language-based editing system. *ACM Transactions on Software Engineering and Methodology*, 1(1):96–127, January 1992.

[7] Tim Berners-Lee, Robert Cailliau, Jean-François Groff, and Bernd Pollermann. World-Wide Web: The information universe. *Electronic Networking: Research, Applications, and Policy*, 1(2):52–58, Spring 1992. Available via ftp://ftp.w3.org/pub/www/doc/ENRAP_9202.ps.Z.

[8] Tim Berners-Lee and Daniel Connolly. *Hypertext Markup Language: Internet Draft*. CERN, Geneva, Switzerland, 1.0 edition, July 1993. Available via ftp://ftp.w3.org/pub/www/doc/html-spec.ps.Z.

[9] Luca Cardelli. A language with distributed scope. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Programming Languages*, pages 286–297, January 1995.

[10] Craig Chambers, David Ungar, Bay-Wei Chang, and Urs Hölzle. Parents are shared parts of objects: Inheritance and encapsulation in SELF. *Lisp and Symbolic Computation*, April 1991.

[11] Pehong Chen and Michael A. Harrison. Multiple representation document development. *IEEE Computer*, 21(1):15–31, January 1988.

[12] James Clark. DSSSL-Lite. http://www.jclark.com/dsssl/lite.html.

[13] William Clinger and Jonathan Rees. Revised[4] report on the algorithmic language Scheme, November 1991.

[14] E. Jeffrey Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, September 1987.

[15] D. D. Cowan, E. W. Mackie, G. M. Pianosi, and G. de V. Smit. Rita – an editor and user interface for manipulating structured documents. *Electronic Publishing: Origination, Dissemination, and Design*, 4(3):125–150, September 1991.

[16] Brian M. Dennis, Roy Goldman, Susan L. Graham, Michael A. Harrison, William Maddox, Vance Maverick, Ethan V. Munson, and Tim A. Wagner. A document architecture for integrated software development. Available via ftp://ftp.cs.berkeley.edu/ucb/mhgs/maverick/ensemble.ps, 1994.

[17] Paul M. English, Ethan S. Jacobson, Robert A. Morris, Kimbo B. Mundy, Stephen D. Pelletier, Thomas A. Polucci, and H. David Scarbro. An extensible, object-oriented system for active documents. In Richard Furuta, editor, *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, pages 263–276. Cambridge University Press, 1990.

[18] Paul M. English and Raman Tenneti. Interleaf active documents. *Electronic Publishing: Origination, Dissemination, and Design*, 7(2):75–87, June 1994.

[19] International Organization for Standarisation. Information technology – text and office – systems document style semantics and specification language (DSSSL), 1994. Draft International Standard.

[20] Dario Giuse. *KR: Constraint-Based Knowledge Representation*. Carnegie Mellon University, Garnet Project, 1994.

[21] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, Oxford and New York, 1990.

[22] James Gosling, David S.H. Rosenthal, and Michelle Arden. *The NeWS Book*. Springer-Verlag, 1989.

[23] Frank Halasz and Mayer Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, February 1994.

[24] Frank G. Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[25] Lynda Hardman, Dick C. A. Bulterman, and Guido van Rossum. The Amsterdam hypermedia model: extending hypertext to suppport *real* multimedia. Technical Report CS-R9306, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1993.

[26] Lynda Hardman, Guido van Rossum, and Dick C. A. Bulterman. Structured multimedia authoring. Technical Report CS-R9304, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1993.

[27] Michael A. Harrison. Defining hypermedia: The essential elements. Technical Report UCB/CSD-92-671, University of California, Berkeley, Berkeley, CA 94720, July 1992.

[28] Kaleida Labs Inc. The Kaleida platform: A review of key benefits. White Paper, 1994.

[29] Kaleida Labs, Inc., Mountan View, CA. *Field Guide to the ScriptX Language*, 1.0 edition, 1994.

[30] Frank Kappe and Mansuet Gaisbauer. *Hyper-G Text Format (HTF)*, 2.10 edition, May 1994.

[31] Leslie Lamport. LaTeX*: User's Guide and Reference Manual*. Addison-Wesley, 1986.

[32] Macromedia, Inc., San Francisco. *Director for the Macintosh: Interactivity Manual*, 3.1 edition.

[33] Daniel D. McCracken. *Introduction to Gain-Momentum – Tutorial and Cookbook*. Addison-Wesley, Reading, MA, 1992.

[34] Microsoft Corporation, Redmond, WA. *Microsoft Visual Basic: Programmer's Guide*, version 3.0 edition, 1993.

[35] Brad A. Myers, Dario Giuse, Roger B. Dannenberg, Brad Vander Zanden, David Kosbie, Ed Pervin, Andrew Mickish, and Phillipe Marchal. Garnet: Comprehensive support for graphical highly-interactive user interfaces. *IEEE Computer*, 23(11), November 1990.

[36] Netscape Communications Corp. *Netscape: The Online Handbook.*

[37] Steven R. Newcomb, Neill A. Kipp, and Victoria T. Newcomb. The "HyTime" hypermedia time-based document structuring language. *Communications of the ACM*, 34(11):67–81, November 1991.

[38] Jakob Nielsen. *Multimedia and Hypertext.* AP Professional, Mountain View, CA, 1994.

[39] John Ousterhout. TCL: An embeddable command language. In *Proceedings of the 1990 USENIX Winter Conference*, pages 133–146, 1990.

[40] Vincent Quint and Irène Vatton. Grif: An interactive system for structured document manipulation. In J. C. van Vliet, editor, *Text processing and document manipulation*, pages 200–213. Cambridge University Press, April 1986.

[41] Vincent Quint and Irène Vatton. Making structured documents active. *Electronic Publishing: Origination, Dissemination, and Design*, 7(2):55–74, June 1994.

[42] Thomas W. Reps and Tim Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors.* Springer-Verlag, 1989.

[43] Guido Van Rossum, Jack Jansen, K. Sjoerd Mullender, and Dick C. A. Bulterman. Cmifed: A presentation environment for portable hypermedia documents. CS-R9305, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1993.

[44] Guy L. Steele Jr. *CommonLisp, The Language; Second Edition.* Digital Press, 1990.

[45] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michal Young. Foundations for the Arcadia environment architecture. In Peter Henderson, editor, *Proceeedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 1–13. ACM Press, November 1988.

[46] David Ungar and Randall B. Smith. SELF: The power of simplicity. *OOPSLA*, October 1987.

[47] Janet H. Walker. Document examiner: Delivery interface for hypertext documents. *Proceedings ACM Hypertext '87*, pages 13–15, 1987.

[48] Janet H. Walker. The role of modularity in document authoring systems. *Proceedings of the ACM Conference on Document Processing Systems*, pages 5–9, 1988.

[49] Janet H. Walker. Supporting document development with Concordia. *IEEE Computer*, 21(1):48–59, January 1988.

[50] Larry Wall and Randal L. Schwartz. *Programming Perl.* O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[51] 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 1988.