Support of Hypertextual Annotation on the Web

Huan Gao

Technical Report CS-2004-20

April 10, 2004

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

Support of Hypertextual Annotation on the Web

by

Huan Gao

Submitted in partial fulfillment of the requirements for the degree of

Master of Applied Computer Science

at

Dalhousie University

Halifax, Nova Scotia

April 2004-04-010

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a project report entitled "Support Hypertextual for Annotation" by Huan Gao in partial fulfillment of the requirements for the degree of Master of Applied Computer Science.

Dated:

Supervisor: ______________________________

Dr. James Blustein

Readers: ______________________________

DALHOUSIE UNIVERSTIY

DATE:

AUTHOR: Huan Gao

TITLE: Support Hypertextual for Annotation

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: Master of Applied Computer Science

CONVOCATION: May Year: 2004

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The aim of our project is to improve the reader’s active reading in scholarly works. We raised our ideas of designing novel linking annotation tools that enable one to better use digital documents for tasks in which paper documents are still otherwise superior to digital documents. We explored the related works XLibris and ALIEN in the literature and presented our project ideas in detail. We chose the Multivalent Document Model to implement our idea, because that model is extensively open which will help enhance all aspects of a digital document system. We designed new linking annotation patterns such as label, Multiple-destination hyperlink and internal hyperlink, and implemented them on the Multivalent Web Browser. Linking annotations are built in the web browser on the client side, and XML technology is used in the project to store annotation information. A simple graphic interface is given to enable users to manipulate the annotation tools easily. Finally we discuss the benefits of the linking tools, and introduce our future work to develop more complex annotations types with better user interfaces.

Keywords

Reading paper, on-line documents, digital document readers, hypertext, and digital libraries.

ACKNOWLEDGMENTS

I could never have completed this project without the help and support of several people. First, I would like to thank Dr. James Blustein (my supervisor) and Dr. Peter Hitchcock for their invaluable guidance and help during the progress of this work.

In addition, I would like to thank my dear parents whose love and support keeps me going through the good time then the bad.

Chapter 1 Introduction

1.1 Motivation

Web browsers as popular reading tools enable many people to access variety kinds of digital documents on the Internet. There is a general tendency for people who used to read mainly from paper to read on-line documents as part activities of daily life. Many features have been developed on the Web browsers that dominate the market to make it easier for readers to access the resources of the World Wide Web. In support of good presentation of hypertext documents and browsing functionality, current web browsers satsify the basic reading goals of entertainment or finding information. People are able to browse the World Wide Web where they can find a rich variety of nuts-and-bolts text messages and files over it, in a hard-to-use fashion. Web browsers do not just present text, but format documents with headlines, graphics, sound, video, and click-on links to other documents and sites.

However, these efforts do not support the process of "*active reading*" [Charney 1994, p.241]. Active reading is involved in intellectual activities and mostly used in scholarly work; it is a process of studying. Charney thinks that readers read to learn, to understand, evaluate the ideas of others, to come to realizations about the subject matter, and to integrate what they have learned with what they already know [Charney 1994, p.241]. In other words, active reading is associated with knowledge fetching, analyzing, reorganizing, outlining/writing, deep thinking, and problem solving. Marshall et al. observe that complex methods are applied during the process of the active reading: "selfinterruption, re-reading, time constrained skimming, reference checking, annotation and reference pursuing" [Marshall, et al. 1999, p.83].

For active reading activities, the benefits of paper far outweigh those of current on-line tools: O'Hara's research on comparing reading from paper to reading on-line indicates that paper offers major advantages in supporting annotation while reading, quick navigation, and flexibility of spatial layout [O'Hara et al. 1997]. According to Marshall et al: "Reading from paper allows readers to easily deepen their understanding of the content, extract a sense of its structure, plan for writing, cross-refer to other documents, and interleave reading and writing" [Marshall, et al. 1999, p.335]. That is a major reason why paper continues to be the preferred medium for much of our reading activity even thought alternative digital documents are available on-line. On the other hand, digital documents are superior in many ways to their paper counterparts, in particular, digital documents can be dynamically updated, easier to obtain, reproduce, distribute, and search.

Nowadays the WWW has become a popular publishing medium for scholars in many fields, and E-Journals and Digital Libraries have provided the main platforms on which to accomplish this. Researchers publish their research results in articles on the Web, and they can extend free navigation across the related research literature. Readers can immediately access definitions of key terms, cross-references, or graphic illustrations. Most digital libraries (for example: CiteSeer.IST Scientific Literature Digital Library and The ACM digital library) provide citation links that allow scholars to move quickly from a citation in one article to the cited work or to find the publications of the specific author.

Imagine a student who is studying a journal article on computerized documents. He is not merely reading the text but rather he is actively engaging with it. He may write notes in the margin referring to other passages, include definitions of technical terms, attach comments or build links.

In our work, we aim to design better hypertextual annotation tools for the web browser to help the readers improve understanding and perform their research on the Web.

1.1.1 Hypertextual annotation

In the dictionary [Lexico Publishing Group 2004], annotation is defined as the act or process of furnishing critical commentary or explanatory notes. Marshall et al. said, "Annotation is a tangible reflection of a reader's engagement with the text" [Marshall, et al. 1999].

Also in the Oxford dictionary [Simpson, J., Weiner, E. 1993], hypertext is defined as original and chiefly computing text which does not form a single sequence and which may be read in various orders; spec. text and graphics (usually in machine-readable form) which are interconnected in a such a way that a reader of the material (as displayed at a computer terminal, etc) can discontinue reading one document at certain points in order to consult other related matter. The World Wide Web is the most widespread hypertext system, consisting of many separate but interlinked Web pages. Links in a Web page can refer to any available Internet resource including text, images, sound, video and programs.

With hypertext, annotation is more than just making commentary and explanatory notes. It covers a broad territory. Research shows that it has been constructed in many ways: "link making, path building, commentary, marking in or around existing text, decentering of authority, a record of reading and interpretation, or community memory" [Marshall 1998, p.40]. In this article, we explore the integration of the traditional techniques of textual annotation with the newer communication medium of hypertext.

The key point of hypertextual annotation is that it allows the reader to become the user to choose and create the route to understanding that best suits their individual needs. Marshall et al. claim that annotation is able to increase the value of hypertext. Annotation can be looked as a basic aspect of hypertext as it reflects the interaction of the readers with the hypertext: making comments, building new connections, creating new pathways, and interpreting materials. All types of above annotations contribute to an accretion of both content and structure of existing hypertext [Marshall 1998].

We are interested in presenting annotation work as a hypertext because hypertext is able to bring comprehensive content easily and imaginatively to the readers, enable the reader to integrate multiple sources of information via the Internet directly into the annotation and offer reader multiple ways of interacting with annotation.

1.1.2 Common hypertexual elements in annotation

Hypertext-like structures exist in encyclopedias, newspapers and reference books (with their tables of contents, cross-references and indices). Such documents can be thought of as printed hypertexts where readers must manually look up the links rather than just following a link and go directly and automatically to the destination by using a web browser. As the amount of hypertext grows, it is increasingly important to find new ways to bring value to them and take advantage of it.

When we are concerned with hypertextual annotation, it is necessary to look at the annotation on paper. Marshall et al. thought, “In fact annotations on paper are hypertextual. Annotations exit in non-linear relationships to the printed linear text: they interrupt linear reading, connect disparate passages, and in general function as hypertext is intended to” [Marshall 1998, p.43]. In other words, they are a direct reflection of a reader’s engagement with the text. As our objective to this project is to improve the flexibility of user’s interaction with digital documents, we try to promote this engagement with the text on on-line web-based tools.

Before we present the idea to strengthen hypertexual annotation in our project, we discuss five common hypertexual elements that are already realized in annotations on paper.

Associations, links, and relations. There exist at least four kinds of associations that readers make in the printed books. According to [Marshall 1998], the associations can be distinguished by level. They are association at collection or composite level, association between notes and annotation links, association from an anchored portion of the text to a note, commentary, or word-to-word association.

Anchors. An anchor is a way of setting off a span of text, usually as a start or endpoint for a link [Marshall 1998].

Emphasis. Emphasis is a special mark (often a symbol) to indicate that hypertexual element nearby is very important. Emphasis also can be represented by changing the shape of highlighting marks or the use of highlighting pen in different colors [Marshall 1998].

Constructing new nodes from document segments. When the author's structure does not suit the reader's purposes, the reader allows re-segmenting the document for better understanding.

Types and categories. The annotations are given types indicated by color or some other visual property [Marshall 1998].

1.1.3 Hypertext interaction

The conclusion of an investigation of linking practices on the WWW presents that the majority of hypertext linking on the WWW is estimated to be intended for navigational purposes only [Marshall 1998]. The author creates links based on their understanding and context of the document in term of their writing goals. Associative links are designed by the author to disrupt the narrative flow by inviting readers to go elsewhere, but instead of enhancing the reader's understanding of a subject, readers often lost in the hyperspace. In addition, as linking on the WWW is restricted to serve navigational purpose only [Marshall 1998]; readers often fail to grasp the overall structure of a hyperdocument or to understand the semantics of its links. Generally, the text is a collection of ideas that the writer tries to present. The writer carefully selects content and organizes it into a coherent sequence to transfer knowledge to the readers. Readers often depend on discerning the structure to make sense of the text and to extract the writer's thoughts. We would expect well-defined structures to be more comprehensible to readers [Charney 1994].

Hypertext often suffers from these two problems known as the navigation and comprehension problem. Furthermore, there is a gap between the writing narrative and the reading process es: Although author builds static structures to allow users restricted level of interaction to decide, the hypertext structure and contents are pre-defined [Pimentel, et al. 2000]. Following someone else's path through a hypertext may be as revealing of the reader's interpretation of the work as it is of the author's, but in the real process of reading, the active reader will access and explore the hypertext in a multilinear way. The reading strategies vary in terms of reader's prior knowledge of the domain, the reader' task for reading, learning styles of the reader, and the nature of the information itself [Charney 1994, p.258]. Although hypertext designers try to anticipate readers well, it is impossible to anticipate all the paths that readers may desire to follow within and between texts as a huge number of possible combinations of such factors exist.

The research shows that in a typical process of active reading, readers first skim documents to find information relevant to some specific questions or gain the image of whole content, then use deep reading to understand and to solve their problem [O'Hara, et al. 1997]. During the course of first reading, readers may make a set of markings for later reference, and some pieces of marking text might have internal relationship contribute to one theme or might help the readers extract structure form the text on re-

reading the article [O'Hara, et al. 1997]. In some cases, the reader may only skim for main points of each paragraph or read only sections relevant to their goal.

We assume that scholar readers have ability to organize information effectively for their level of knowledge and their purpose in reading. We concern about allowing the readers to make their own reading order and to relate the text to their own experience. We are interested in enabling the reader to leap and make connections between disparate texts freely at will. The trail of associations can also be stored. We try to free the readers to choose what portions of a text to read and in what order. Moreover, we enable the readers to construct their own citation links and to allow link to have more than one destination, which allow the new link model reflect one-to-many relationships.

1.2 Project Goal

Hypertext is a technology that can be used in many different ways. The significant feature of hypertext is that hypertexts have no edge of the page. David Kolb presents: "the hypertext would provide linkage, clarity-complexity, freedom of invention and structure. An ability to perform complex linking across multiple levels of description and abstraction would provide the possibility for creating new intellectual objects and discursive moves" [Bucur, et al. 1999]. The goal of this project is to develop new hypertext applications that can be used in web browsers to support readers of electronic versions of discursive scholarly works. We aim to design new linking patterns and annotation types to help the readers set their path-following policy and navigate more deftly through electronic text. We also implement a user interface to evaluate the effectiveness of new features of improving the reader's interaction with the hypertext.

1.3 Project Content

As with other complex software, hypertext systems are often designed using certain architectural model. These models are of particular interest when planning interchangeable system and when comparing features across system. The WWW, as it was originally conceived, implemented the Dexter Model (The Dexter Model was developed at the Dexter Inn in Sunapee, NH in 1988 by a group of hypermedia designers). In the model, three conceptual layers for hypertext systems are included: the runtime layer, the storage layer, and the within-component layer. The focus of the model is on the storage layer, which models the node/link network of the hypermedia system. Dexter model supports computed as well as static links. Links can be single or bi-directional as well as multithreaded [Gottlob, et al. 1994].

In the project, we are not concerning with links created by the author; we explore new techniques that allow the readers to construct their own hypertext links as annotations in their the web browser.

First, we develop annotation tool "label" to help the readers' discursive movement and support better cross-reference in the same hypertext. Label functions like bookmark. Comparing that bookmark is used to save the URL (web address) of a page in order that the readers can easily return to it later, label saves the locations the readers are interested in the same document to enable quick access later.

Second, we give the readers ability to create internal hyperlink with flexibility to allow them to change the narrative structure predefined by author to meet their own requirements to understand.

Third, given that scholarly reading often involves following citation links to access cited material, allowing the readers rather than author to create citation links greatly improves navigation across the related research literature. Here, we design tool for constructing multi-destination links. Naturally, in the real world, there exists one-to-many relationship between a point in the text and related references that the reader could provide while reading. One possible enhancement to the simple notion of link to basic hypertext model is to allow for links having more than one destination. On activation of such a link, the system could provide the user with the possible destinations to select from.

Fourth, making notes gives the readers more places to add comments to a document. In the project, we enable all above links to work in the Post-it   Notes, an existing annotation type in MVD model, to greatly enhance and expand interaction deftly with the hypertext.

1.4 Structure of this report

The rest of this paper is organized as follows. Chapter 2 is an introduction to the background of the project. In this part, the existing linking annotation system literature is explored; the new idea is introduced and described. In the chapter 3, the Multivalent Document Model (MVD) model is briefly introduced and the detailed design process is presented. Chapter 4 is a worked example of the project. In the chapter 5, there is a discussion as conclusion about the implementation of the design, the usage of the tool, and the practical benefits of the tool. At the end of this chapter, the open issues or directions for future work are outlined. In the end, an appendix is attached including the user guide, API and UML of the main classes.

Chapter 2 Background

The most widespread hypertext system today, the WWW empowers the access to entire online information sources in the world literature by following associative links. However, the existing hypertext links are mostly for navigational purposes only. If the meaningful associative links through annotation, which mostly reflects inter-document relationship, can be built by the readers, it would not only improve the readers' understanding of the existing materials, but also contributes new ideas as associative links which can seamlessly link the new material into the global context, and help the WWW evolve to achieve all its potential hypertexual richness [Catlin, et al. 1989].

2.1 Hypertextual annotation systems review

We explore the existing two hypertexual annotation systems and the emphasis of this review is on mechanisms for reader-directed link construction, linking with annotational support for cross-reference.

2.1.1 XLibris

XLibris system provides a paper-like hypertext interface for reading, annotating, and navigating among documents. It provides a display that emulates the appearance of a sheet of paper and uses a paper document metaphor to support analytic reading activities. Its interface style known as “Linking by inking” is used for reader-directed link construction to integrate reading with browsing to enhance both activities. XLibris monitors free-form ink annotations made while reading, and uses these to organize and to search for information. Three types of reader-directed links are provided: margin links, further reading lists, and ink anchors. The readers can create margin links to related passages in the margin. Margin links allow readers to identify related information in the context of their reading instead of as part of a separate task. As reader annotates a document, the system performs queries, and displays links to related pages. At most one margin link results from each annotation. XLibris can automatically generates further reading lists for each document. Further reading list is to help go into more depth and detail to satisfy the readers’ interest. Unlike static reference, these lists reflect the interests of a specific reader and the reader’s interests are inferred from annotations [Adler, et al. 1998]. XLibris presents the further reading lists in a separate multi-page view associated with the source document. Readers can access this view at any time even through they have not made any annotation on the document, but more annotations can make the links reflect reader’s interest better. Ink anchor allows readers to link document together for their own particular requirements. Passages are linked together by annotating each passage with a similar ink anchor. Circling any mark converts the mark into an anchor. Activating an ink anchor would create a composite view of all the linked passages in which all the passages marked with the same ink anchor are displayed end to end in a linear fashion [Adler, et al. 1998].

2.1.2 Annotation Linking ENvironment (ALIEN)

The annotation Linking Environment was implemented to demonstrate the principles of “linking by annotating”. The annotation system offers for associative linking – the expression of relationships between different annotations. Four major components are included in ALIEN: the Annotator, Annotation Linker, Annotation Desk, and Annotation Server. Annotator works as a user interface “plug-in” tool for annotating web documents that is integrated into the Internet Explorer web browsing environment. The Annotation Linker component is used to express relationships between annotations of web documents. The Annotation Linker allows the reader to record the implicit associations by presenting a number of different link types. The Annotation Desk is able to build a more complex relationship between annotations. It facilitates the creation of composite nodes, labeling of links and annotations, and the capacity for the author to associate meaningfully their own statements with their annotations. Annotation Desk is currently able to recognize marks corresponding to the grouping of objects, the linking of two objects, and the linking of multiple objects to a single object. The Annotation Server is responsible for storing the annotations and associations created using the Annotator, Annotation Linker, and Annotation Desktop. It can be accessed in multiple threads. Annotation Server makes a request to invoke a script that decodes the request parameters through CGI interface. The script consults a database to store and retrieve annotation descriptions and returns results in XML format [Price, et al. 1998].

2.1.3 Discussion

The above two annotation systems (XLibris and ALIEN) provide good mechanisms that can be used to express association between information nodes. Both systems focus on how an annotation-based approach could be used to capture inter-document relationships. New linking types give the reader the capability to move from one text to another to satisfy their information needs. However, in XLibris and ALIEN system, the annotation manipulations are relatively complicated and require more effort to learn and use them. The XLibris reader's Notebook is primarily a personal reading device and annotations are expected to be stored locally and not be accessible to anyone else. By contrast, in ALIEN system, annotations are stored separately from the annotated documents on the server side. The Annotator and Annotation Server communicate each other to store, retrieve and display annotations.

In our project, we try to extend new link types and design simpler and clearer interface that give every reader an easy way to make links. We attempt to produce meaningful associative links to reduce the risk of disorientation, miscomprehension, and cognitive overload through discipline of the hypertext network [Price, et al 1998]. We are specifically not interested in automatically making links within a text, and we are not interested in sharing annotations between users to support collaborative annotation. We develop tools for personal use only. In the next section, we describe how project ideas are raised and how each part should work.

2.2 Project ideas

The new ideas include label, internal hyperlink, multiple-destination hyperlink and enabling links in the post-in notes. In the next section, the idea will be discussed in detail.

2.2.1 Label

In most of scholarly reading, reading article is for searching useful materials to solve the problem. The reader often first scans the article quickly to find what material is important, what material might require a second reading, and what material contributes to what themes. In this procedure, the reader needs to mark the article as procedural signals and these markings act as a visible trace of a reader's attention. Marking annotation here plays a role of anticipation of future attention – to designate reading assignments, responsibility for "knowing" and desire to reread [Marshall, et al. 1999]. Emphasis marks can be worked as placemarking and aids to memory. The reader also may partition and sort the gathered information into themes. Since each theme may be divided into subthemes requiring further partitioning and sorting, this process is inherently iterative. Using this structure as a guide, the reader actually produces a new material, which includes related or valuable content in the article to help the reader reach reading goal. Therefore, the first reading is skimming and it helps to extract structure from a document and to speed re-reading. We try to design a convenient tool for the reader to facilitate combination of emphasis marking and quick and more effortless navigation through the article. We designed a label tool as bookmark to help the reader get to a particular location in the same document.

In the real world, a label is a simple printed or engraved name ticket (usually paper but sometimes leather), stuck to the inside of the upper cover or one of the front flyleaves of a book, generally for purposes of identification or ownership [Roberts & Etherington, 2003]. In fact, the concept of label has already existed in the paper document. Label tags sections of the text with the equivalent of Post-it Note flags. The readers are able to select a point in the text, or a region of the text, and assign a label to it. These labels must be something that users can jump back to it later. The physical label markers might have color and text associated with them. It would be best if the labels could be assigned to classes by color. In our project, we try to emulate real labels in the electronic text.

We establish the label by first selecting text, and a small rectangle tag will appear above the selecting text. The rectangle tag with color representing a label works as an emphasis marking. Each label can be assigned a color, and be assigned a title which describe which theme the text contribute to or which problem the text is related. The color and title may be changed at any time and default title is the selecting text itself. The reader has capability to delete label and add new label. When the new label is created, it is added into the menu so that the reader can easily find the label from the menu and click the item to go directly to desired location in the document. The labels can be grouped by title or by color. Title and color give clue to the reader which labels are related together. So all related materials are collated as a cluster from diverse locations in a long document. Moreover, color may also implement “levels of importance” [O’Hara 1997]. The readers are able to assign a meaning to a specific color and define their own informal coding.

2.2.2 Internal hyperlink and multiple-destination hyperlink

Cross-referencing is the most frequently observed reading mode. A significant aspect of hypertext is that hypertext contains associative linking between documents that enable cross-referencing between multiple documents to be a common activity. As the readers read the document deeper and further, the readers often need to find more materials in the research literature as complementarities to help them understand, explain, compare and analysis existing body of material. In another case, when the readers read the article, new ideas come into their mind, they need to find new materials to proof or support their idea. If a reader has the ability to create new connections and pathways, then this may increase the value of the hypertext for future readers. Varieties of link types have already existed in hypertext, for example, external hyperlink and internal hyperlink. The author can create a hypertext document easily by using edit tool such as Microsoft FrontPage or Netscape Composer. However, when the Web page is published on the Web, it is not changeable by anyone but its owner. In our project, we design linking tools for the readers to create their own links. We allow the readers to "edit" links in the published web documents in web browser as link annotations and save them separately from the original documents and revisit the annotation links anytime when they access the annotated web document on line. Except we support the readers to create traditional link types internal hyperlink rather than authors, we further create a new link type multiple-destination hyperlink. We distinguish the annotation links from the links the author made in the original document by color. We assign green color underline for internal hyperlink and red color for multiple destination links. All links can be easily added and deleted by selecting right button menu when the mouse is over the link. Information about link annotation is saved in an XML file which record the type of link, link title, the full address of the remote resource URI or destination location in the document (for internal hyperlink), robust location of the link.

An internal hyperlink is one that takes us to a new location within the current open web document. Before we can establish an internal hyperlink, first we must create a “target”. A target defines the place within the text where we want to maneuver to. We allow the readers to define their own internal hyperlink to give them ability to build their own index structure or connect related two parts and provide a means to maneuver through the information presented in a long document.

Multiple-destination hyperlink works as the readers’ citation link. Readers can connect multiple sources to a point in the document and give the readers choice to go to different external sources. The readers may add and delete destination links and may give each destination a unique name to identify it. The system also keeps track of locations visited by the readers and provides controls for backtracking. Multiple-destination link reflects one-to-many relationship in the real word and greatly enriches traditional hypertext authoring approaches.

2.2.3 Enable linking in the Post-it notes

Making comments is another effort for annotation. When direct paper annotation is not sufficient for all tasks and purpose, it is necessary to make Post-it Notes. Post-it Notes have their advantages of visibility, removability and convenient size. It is displayed in separate window over the web browser window. Nevertheless, if all types of link above can be applied in the Post-it Note, it will greatly improve composition of a textual annotation in way that is more complicated while you are reading. It is merely convenient to use a number of forms of annotation together to a single purpose.

Chapter 3 Linking annotation tools design

We implement the new ideas by developing new linking annotation tools based on multivalent document model (MVD), which is a research project at UC Berkeley. The multivalent document model is implemented in pure Java; it supports multiple document types; it is an open source and runs everywhere; MVD provides multivalent extension mechanism, which gives developers the power of arbitrary source code modification. Multivalent Browser is an inviting platform for working out new ideas. Before we describe the design process of our new linking annotation tools, we give a brief introduction about Multivalent Browser and MVD.

3.1 Multivalent browser

Multivalent browser is a 700KB Java application. It not only display HTML format webpage, but also scanned paper (two kinds: XDOC and PDA), UNIX manual pages, TeX DVI, ASCII, Zip, PDF and local directories. Multivalent browser supports a variety of annotation features such as highlighting (in different colors), hyperlink and anchors, Post-it Notes, move text, short comment, replace with, all caps, initial cap, lowercase.

3.2 Multivalent Document Model

MVD is an architecture in which a document is viewed as a composition of intimately related but distinct layers of content and dynamically loaded program objects, called behaviors. They can be grouped in layers. Layers and behaviors are assembled by an MVD compliant browser from multiple distributed sources over the network. MVD provides an infrastructure for the meaningful composition of layers and behaviors. As a result, any media type can be bridged into the multivalent model. MVD also defines the basic document operation as a set of protocols that were supported by additional mechanisms, allows these components, often authored without specific awareness of one another, to compose as a seamless unity.

3.2.1 Behavior

Multivalent documents comprise functionality encapsulated into extensible behaviors and layers of content. Behaviors provide all the user-visible functionality in the system. The term layer refers to both a conceptual unit of document construction and a runtime data structure that treats groups of behaviors as a unit. Behaviors are implemented as Java class that participates in the communication protocols detailed in subsequent sections..

Behaviors can be generally useful, operating over many document data types and media formats, without concern for media-specific details. Behaviors are given the power and flexibility to access and potentially filter all document content and operations on documents. The behavior gives the Multivalent system its extensibility.

3.2.2 Document Tree – central data structure

The MVD applied a document tree as central data structure. The document tree directly represents the structure of a document and internal nodes of the tree reflect the logical hierarchy of the document [Phelps & Wilensky 1998]. For good understanding this concept, we give a example of a structured article. The following figure shows that the tree root is the article that has children chapters; chapters are divided into sections, which comprise subsections, which comprise tables and paragraphs; tables comprise table cells, which comprise paragraphs. Paragraphs may comprise text, graphics, or other media types.

Figure 3.1 Document tree model [Phelps & Wilensky 1998]

The usual root of the tree is an internal node type called document, which has a URL, a style sheet, scrollable content, and holds the list of document-specific behaviors. The medium-specific elements such as words, graphics shapes, frames or figures are included at the leaves of the document tree. Medium-specific qualities of a document format are encapsulated in leaves. Media adaptor is responsible for interpreting all access to the corresponding, so that other behaviors can operate on the abstract document tree, and requests to leaves without burdening general behaviors with media-specific knowledge. The document tree has a standard set of navigation and tree management functions (adding, removing, and querying children, and so on). The division of document content into medium-independent internal nodes and medium-dependent leaves allows developers to write behaviors against an idealized abstract document tree and have the behavior operate on any concrete document format.

3.2.3 Media Adapter

Media of various type (text, video) and format (within text: HTML, PDF) are encapsulated by specialized behaviors called media adapters. During the build stage, these behaviors contribute to the construction of a document structure tree, called the Integrated Document Element Graph (IDEG). Separating the document structure from the media elements facilitates a multimedia document system. Behaviors (other than media adapters) operate on the medium-independent structural document tree and communicate with encapsulated media types through the protocols. Hence, behaviors can be written once without special accommodation for any particular medium. It applies to a given medium, and operates on all media types.

A behavior's build method specifies how that behavior modifies the IDEG. Generally, a document will contain one behavior that builds the primary structure of the IDEG from one or more layers; we call this structure informally "base" document. Other behaviors' build methods may incorporate additional layers into the document structure by modifying the IDEG [Wilensky 2001].

3.2.4 Protocol

All digital document systems share a fundamental document lifecycle: the application is loaded, the document is read in and internal data structures are built for it, the document is formatted, then it is painted on the screen, at which point the system waits for the user do something. The system is a framework where the system is in charge of the overall flow of control. To allow arbitrary extensibility of any aspect of the system, each of the fundamental runtime operations on digital documents has been opened with an extensible protocol. In the MVD model, the life cycle begins with document instantiation (restore), the assembling of components of the document, during which behaviors and layers are loaded, and the behavior methods are inserted into their appropriate places in the other protocols. Then the build protocol is started; the build methods create an internal graph data structure for the document, using the information in the layers. After `build`, `format` formats the resulting documents, and then paint renders the document on the screen. At this point, the user events protocol is started. An event loop waits for input from the keyboard, mouse or other input device, and hands it to the methods implementing the protocol. Among other things, events can trigger the save protocol, cause the document to print, or select a portion of the document.

Figure 3.2 Fundamental document lifecycle [Phelps & Wilensky 1998]

As diagrammed above, the lifecycle concerns both persistent and runtime representations of the document in several phases. Here, We give a brief description of the phases of the fundamental document lifecycle (ignore undo and clipboard protocol) [Phelps & Wilensky 1998; Wilensky 2001].

Below are described the low-level communication protocols: those performance-intensive protocols concerning construction and display of the document, and system input.

3.2.4.1 Restore protocol

The Restore protocol restores a document from a concrete document format and instantiates the building blocks of the runtime document. Often `Restore` reads a hub document specification to determine the relevant behaviors and instantiates them with attributes from the hub by which behaviors can find their corresponding layers. Usually layer data is cached in this phase, to be used in Build or another subsequent phase.

Essentially the Restore protocol prepares behaviors and their data for use. As the system reads the hub document, it reads first any global document attributes, then instantiates (create a runtime program object for) each behavior and invokes its Restore protocol method. Behaviors are listed in the hub document with highest priority listed first. A behavior's Restore method is invoked exactly once, and the behavior should at that time cache any data it needs. Besides loading of external data, any other time-consuming operations, such as initialization of data structures, should be done here. Supporting data layers may be restored from separate files or network connections or placed inline in the hub document. If placed inline, the data will be parsed into an SGML/XML parse tree, a structural tree similar to the document tree, with the nesting of the data exactly mirrored in the hierarchy of the tree.

3.2.4.2 Build protocol

The build protocol iterates over the behaviors instantiated in Restore, passing them the root of the document tree from which behaviors can traverse to any part of the tree in order to add a subtree of content of mutate existing content. It has two subphases: `Before` and `After`. In the Before half of the protocol, behaviors augment the document tree with original content, with document structure reflected as tree nodes, content and metadata as leaves and node attributes. In the Build Before subprotocol, media adaptors bridge their content from their data format into the document tree. Data formats remain encapsulated by medium, at the leaves of the document tree, leaving the internal tree uniform across media. This way, behaviors can be written against this abstract structure and yet manipulate any concrete medium. The `After` half of the protocol, behaviors can mutate the tree constructed in the Build `Before` stage and resolve saved location specifications to runtime tree nodes. They also report user interface requirements such as menu entries and tool bar real estate, which are sorted by the system into categories with requires from other behaviors. At the end of Build, the system creates the requested user interface elements [Phelps & Wilensky 1998].

3.4.4.3 Format protocol

Format annotates the logical tree representation of the document with geometric positions. Paint uses these positions when making document content visible on the screen or printed page and picking (which is not a protocol) uses them to map from screen coordinates to semantic document objects [Wilensky 2001].

Formatting occurs during a walk of the document tree, top down propagating maximum dimension constraints and property settings from style sheets, and then bottom up propagating requested dimensions to be positioned. At every parent-child node pair in the tree, the child reports its width and height dimensions, and the parent positions it at an (x, y) location. The parent in turn computes its own dimensions as the union of bounding boxes of its children and passes this information to its own parent, waiting positioning itself. A node's coordinates are only computed relative to its parent. Leaves are often specialized by media adaptors in order to report dimensions for new media.

During the tree walk, the graphics context holds the current settings of font, line justification, margins and other factors that affect the layout. Nodes are obligated to observe these settings in computing their layouts. Graphics context values are set by the prevailing active behaviors on that region of the document. The patterns of structure in the document, as encoded in the style sheet, contribute to the graphics context [Phelps & Wilensky 1998].

3.4.4.4 Paint protocol

The Paint protocol renders a formatted data structure to the screen, printer, or other device. Painting is done in a medium-specific way, often through a media adaptor or a specialized leaf type. It is very similar to Format in the management of core properties. To a large extent, inclusive of style sheets and spans, painting differs from formatting only in the action taken, drawing instead of positioning. In other way, however, Paint emphasizes different aspects of the same mechanism than Format. In the implementation, painting is done on a Java Graphics object.

3.2.4.5 Low-level Events protocol

Within the document display, the user interacts with the system through and via low-level events: keystrokes, mouse clicks, and OS window activity. The Events protocol distributes events to interested behaviors through the tree.

3.2.4. 6 Save protocol

The Save protocol iterates over the behaviors corresponding to the runtime layers of the document, invoking their save method. Layers, in turn, iterate over the behaviors nested within them, invoking save. Ordinary behaviors save behavior-specific state in XML syntax. The result is a hub document that can be read back into the system later to reconstruct salient document state. We will talk about hub document in detail in the later section.

Now we finish introducing the low-level protocol, next we will discuss high-level communication protocol: semantic events.

3.2.4.7 Semantic Events

Semantic event is a high-level communication protocol. The low-level protocols open document manifestation and interaction with the user. They do not address higher-level logical or semantic actions, and these high-level actions must be open to modification by any behavior. Semantic event consists of a message, such as openDocument, and three fields labeled argument, in, and out. Semantic events are most often acted upon by behaviors to implement a requested action, to modify the event, or to update state in response to an announcement event.

3.2.5 Persistent Manifestation: Hub Document

The particular composition of layers and behaviors that comprise a given multivalent document is captured persistently in a hub document. Written in the Extensible Markup Language (XML), the hub document lists hierarchically the behaviors for that document, and provided the relevant attributes to behaviors to find their associated layer or layers in some cases supplies layer content in the hub document itself.

Hubs are loaded by the system when the system starts up and when individual documents are loaded. The system's built-in hub is loaded first, then a hub, form the user's home directory, which can augment or delete behaviors given in the system hub. By editing the applicable hub, behaviors can be added, removed, rearranged, replaced, and specialized. Hub can be compared to style sheets in that, given a document with some structure, style sheets describe how to display the document, while hubs describe how one may interact with it and how to control the construction of the entire application.

3.2.5.1 System hub

The system hub lists behaviors applicable to all documents. It includes the basic part of the File, Edit, and Help menus; document popup menu and entries for word lookup in the dictionary, etc.

3.2.5.2 Document-specific hubs

Document-specific hub holds behaviors that apply to that document only. Generally, document-specific hubs are used to hold annotations, for which the behavior code is common, but he application instances to the particular document as specified in attributes unique.

3.3 Linking annotation tools design

We build new linking annotation tools based on Multivalent document model because it addresses the extensibility concern and provides a base for work on collaboration through annotation. The annotations are implemented by specialized behaviors. They are composable (the Multivalent framework manages them through the protocol), source format independent (they manipulate the abstract document tree and communicate to encapsulated media types through media adapters), server independent, extensible, seamlessly integrable, immediately portable over the network, and powerful (they have access to every state of the fundamental document life cycle).

The Multivalent system is written in Java. A new behavior, which is a java class, is added in the system by extending the system class Behavior and overrides the methods relevant to accomplishing its effect. However, the new behavior has to map the desired functionality into protocols. The new behavior will be called by the system framework at the right time to have its effect, and then relinquish control back to the framework in order to compose well with the other behaviors in the system. The Multivalent architecture uses event driven mechanism, but event propagation is not standard Java event propagation. Low-level mouse, keyboard, and window events are taken from the standard event queue and generated high-level semantic events are queued there, but internally events flow through the document tree or round robin through behaviors. This propagation allows behaviors to filter and short-circuit other behaviors, and in general be part of the event chain by virtue of their fundamental place in the document without additional bookkeeping.

As multivalent system is extended by adding new behavior, in my project, to implement new functionalities multiple-destination hyperlink, internal hyperlink, and label, we create new behaviors `MCHyperlinkSpan.java`, `InternalHyerlinkSpan.java`, `AddLabelSpan.java`, and `TargetAnchorSpan.java`. Some important classes such as

`Span.java`, `SpanUI.java`, `document.java`, `SemanticEvent.java`, and `System.java` are modified as well.

Before adding new behaviors into the system, we have to modify the related hubs. Hub holds a set of active behaviors that allows every document to be given a custom browser. Hub is written in XML document pointing to Java classes with supporting attributes and data and control the construction of the entire application. When the system loads a document, it first load the relevant hub from `multivalent.jar`'s hub, then all hubs from all JARs in the same directory as Multivalent.jar, and then a user's hub, if any. JARs store their hubs in a sys/hub directory, and the users hub is in his home directory at the path `system.getProperty("users.dir")/.Mutivalent/hub`. System.hub is the first hub loaded and establishes the basic behaviors always present when individual documents are loaded. System.hub lists behaviors applicable to all documents. It includes the basic parts of the menus, searching an search visualization tool bars; document popup menu, and others. We add new menu items to the system: Label and Link. The following is the modified content of `system.hub`.

```
<?xml version='1.0' ?>
<System title='System default behaviors'>
<!-- This is the main hub in the entire system. -->

<?import href='Core' ?>
<?import href='Net' ?>
<StyleSheetSetter behavior='multivalent.std.adaptor.StyleSheetSetter' />
```

```
<!-- toolbar or no UI -->
<!--Meta Behavior='DeleteBehavior' /-->
<Personal behavior=multivalent.std.adaptor.PersonalAnnos />
<!-- maybe put in Annos group -->

<Menubar behavior='multivalent.std.ui.Menubar'
  titles='File|Edit|Go|Bookmark|Lens|Style|Label|Anno|Link|Help'
/>
<Toolbar behavior='multivalent.std.ui.Toolbar' />

<?import href='Help' ?>
<?import href='File' ?>
<?import href='Edit' ?>
<?import href='Go' ?>
<?import href='Lens' ?>
<?import href='Style' ?>
<?import href='Tool' ?>
<?import href='Anno' ?>
<?import href='View' ?>

<!-- PER-USER HUB -- but as applet, users have no identity, so no per-user -->
<!-- actually, most things should go in per-user -->
<BindingsEmacs Behavior='multivalent.std.ui.BindingsEmacs' />
<BindingsWindoze Behavior='multivalent.std.ui.BindingsWindoze' />
<BindingsTk Behavior='multivalent.std.ui.BindingsTk' />
<ShowHeaders Behavior='multivalent.std.ShowHeaders' />

<?import href='RW' ?>
```

```
<render Behavior='multivalent.std.FontRender'  antialiasing='user-off' />

<!-- DebugMode needs to be before Debug layer so when DebugMode changes activation of Debug layer,
it happens after Debug layer has processed event -->

<?import href='Developer' ?>
<!--Menu Behavior='multivalent.std.ui.StandardFile' /-->
<MenuItem Behavior='SemanticUI' SCRIPT='event closeBrowserInstance' title='Close' parent='File'
category='Close' TYPE='Button' />
<MenuItem Behavior='SemanticUI' SCRIPT='event EXIT' title='Exit' parent='File' category='Quit'
TYPE='Button' />
</System>
```

We can see that the name of the layer is given by the root of the XML parse tree, Here two other hubs are imported, Core and net, which are taken from all files named Core.hub and Net.hub from Multivalent.jar, all external JARs, and from the users home directory. Behaviors are identified as such by an attribute named behavior; the name of the behavior can be given as fully qualified class names, such as `multivalent.std.ui.Menubar`, new menu items Label and Link can be added to the attribute of titles. `System.hub` will be parsed by the system framework and at that time the behavior `Menubar.java` will be loaded and attribute “titles” gives a clue about what kind of custom-build menu bar should have. Hubs can import other hubs. XML files can be included with the standard XML xinclude: include mechanism, with xinclude: include as the tag and the href attribute giving the relative path. Relative paths are relative to the current JAR; use systemresource as the protocol to reach into `Multivalent.jar`. More often all hubs by some name in whatever JAR are imported. In this case, use the processing target import, as in <? import href='Anno'? >.

At runtime, a hub is converted into layer (Java class: `multivalent.Layer`), which consists of a list of instantiated behaviors and a list of non-behavior data subtrees. Layers are loaded from and saved to hubs. Attributes in the hub become attributes in the runtime behaviors and similarly with the data subtrees. Thus, layers can be converted between XML file and runtime data structure representation. The system grouped a set of annotation behaviors into a special hub document anno.hub. New behaviors `TargetAnchorSpan`, `MCHyperlinkSpan`, `InternalHyperlinkSpan` that expend `Multivalent.Span` can be added as menu items “Set as Target”, “Multiple-destination hyperlink” and “Internal hyperlink” in Anno.hub. Here is the modified `Anno.hub`.

```
<? xml version='1.0'?>
<Anno behavior='Layer' title='Annotation behaviors'>
<!--Note Behavior='multivalent.std.RestoreReport' /-->

<!-- ANNO -->
<!--Ink Behavior=phelps.InkUI /-->
<!--NoteMan Behavior=multivalent.std.lens.NoteMan /-->

<MenuItem Behavior=NotemarkUI title='Annotations as Notemarks' parent='View' category='ViewNB'
variable='pref.AnnoNB' seed='on' spannames='highlight,hyperlink,redact' />
<!-- 'HighlightSpan,HyperlinkSpan', MCHyperlinkSpan, InternalHyperlinkSpan -->
<MenuItem Behavior=SpanUI logical='highlight' spanname='HighlightSpan' value='(default)'
title='Highlight' parent='Anno' category='AnnoInk' nb='ANNONB' />
```

```
<MenuItem Behavior=SpanUI logical='hyperlink' spanname='HyperlinkSpan' title="<font
color='blue'><u>Hyperlink</u></font>" parent='Link' category='AnnoInk' edit nb='ANNONB' />
<MenuItem Behavior=SpanUI logical='targetanchor' spanname='multivalent.std.span.TargetAnchorSpan'
title="<font color='black'>Set as target achor</font>" parent='Link' category='AnnoInk' edit
nb='ANNONB' />
<MenuItem Behavior=SpanUI point logical='mchyperlink'
spanname='multivalent.std.span.MCHyperlinkSpan' title="<font color='red'><u>Mutiple Destination
Hyperlink</u></font>" parent='Link'  category='AnnoInk' edit nb='ANNONB' />
<MenuItem Behavior=SpanUI point logical='internalhyperlink'
spanname='multivalent.std.span.InternalHyperlinkSpan' title="<font color='green'><u>Internal
Hyperlink</u></font>"  parent='Link' category='AnnoInk' edit nb='ANNONB' />

<MenuItem Behavior='SemanticUI' title='Link Anchor' parent='Anno'   type='menubutton'
generate='LinkAnchor' />
<LinkAnchor Behavior='multivalent.std.ui.LinkAnchorMenu' />

<MenuItem Behavior=WindowUI menu='Anno' category='Note' logical='note' winclass='Note' title='New
Note' doc layer='personal' attrs='width=300; height=100' />
<!-- COPYED -->
<MenuItem Behavior=NotemarkUI title='CopyEd as Notemarks' parent='View' category='ViewNB'
variable='pref.CopyEdNB' seed='on' spannames='copyed' />
<!--
spannames='multivalent.std.span.AwkSpan,multivalent.std.span.InsertSpan,multivalent.std.span.ReplaceWi
thSpan,multivalent.std.span.MoveTextSpan,multivalent.std.span.DeleteSpan,multivalent.std.span.CapSpan,
multivalent.std.span.BIUSpan' -->
<MenuItem Behavior=SpanUI point logical='copyed' spanname='multivalent.std.span.AddLabelSpan'
title='Add Label' parent='Label' category='CopyEd' edit nb='COPYEDNB' />
```

```
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.AwkSpan' title='Short
Comment' parent='CopyEd' category='CopyEd' edit nb='COPYEDNB' />
<MenuItem Behavior=SpanUI point logical='copyed' spanname='multivalent.std.span.InsertSpan'
title='Insert Text' parent='CopyEd' category='CopyEd' edit nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.ReplaceWithSpan'
title='Replace With' parent='CopyEd' category='CopyEd' edit nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.MoveTextSpan'
title='Move Text' parent='CopyEd' category='CopyEd' edit nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.DeleteSpan' title='Delete'
parent='CopyEd' category='CopyEd' nb='COPYEDNB' />

<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.CapSpan' title='Initial
Cap' attrs='captype=ICAP' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.CapSpan' title='ALL
CAPS' attrs='captype=CAP' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.CapSpan'
title='lowercase' attrs='captype=LC' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.BIUSpan' title='Bold'
attrs='type=B' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.BIUSpan' title='Italic'
attrs='type=I' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.BIUSpan'
title='Underline' attrs='type=U' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.AwkSpan' title='awk'
attrs='comment=awk' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.AwkSpan' title='cf'
attrs='comment=cf' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
```

```
<MenuItem Behavior=SpanUI logical='copyed' spanname='multivalent.std.span.AwkSpan' title='choppy'
attrs='comment=choppy' parent='CopyEd' category='CopyEd' nb='COPYEDNB' />
<Button Behavior=multivalent.std.ui.SaveAnnoAs />
<!--
<Button Behavior=multivalent.std.ui.PublishAnno TITLE='UCB DL server'
URI='http://dlp.cs.berkeley.edu:8080/cgi-bin/save.pl' />
-->
<! -- wipe annos goes last -->
<General Behavior= multivalent.std.ui.Annos />
</Anno>
```

As we see that behaviors `TargetAnchorSpan`, `MCHyperlinkSpan`, InternalHyperlinkSpan have a chance to contribute to the menu bar. When menu bar behavior (for example, `MCHyperlinkSpan` or `InternalHyperlinkSpan` here) needs to build a menu, it sends a semantic event with message createWidget/menu-title, and seeds the outfield with an empty menu. New menu items "Multiple-destination hyperlink" and "Internal hyperlink" can be added into the drop down menu "Link" (the definition of attribute parent = 'Link' represents that the drop down menu items belong to menu items "Link" on the menu bar). Menus are built on demand. Behaviors can add to the menu, and when the event returns to the menu bar, the menu bar formats and paints the resulting menu.

3.3.1 features in common

All four behaviors `MCHyperlinkSpan, TargetAnchorSpan, InternalHyperlinkSpan` and `AddLabelSpa`n expand the class `Span.java` so that they have some common features. To describe well the design process, we first give an introduction of the common features and then talk each behavior in detail respectively.

Label, multiple-destination hyperlink and internal hyperlink are all span objects that are created by the readers on the document on line. Before creating links, a linear range of content should be selected. Here we call it span. Span is not only a range of text but also a behavior that has power to control appearance and receives events. An arbitrary amount of the content can be selected as span because it is not restricted by structural node hierarchy and efficiently covers any amount of tree. Span extends from some offset within a start leaf linearly through leaf nodes to an offset within an end leaf. Span also can be used to store metadata, via attributes. As annotation should be designed to give the readers flexibility to insert, delete or edit during runtime and across save/restore editing of base document, behaviors are defined to manage span objects. Behaviors support a user interface to allow creating and destroying span objects at user request, and save or restore them from persistent storage [Phelps & Wilensky 1998].

3.3.1.1 Appearance in situ

As we know that annotation is not part of a copy of document, instead that it can change independently of the original but should annotate "the document itself". MVD model treats annotation as a separate layer of the document districting from the underlying text.

Multivalent framework is responsible for placing the annotations at right place and calling it at the right time. Individual annotation behaviors rely on the geometric placement information of document components available in the format stage of the document life cycle. Annotations are not set in the fixed position but attached to a particular component or series of the document life cycle, and then placed in relation to them. As a result, although the format of a document is changed, the annotation is still by drawn at the right place because it is by drawn in relation to the new position.

3.3.1.2 Robust Location

Annotation is separated from the base document on the client side, but it should be robustly positioned on the document when the system loads the online document. Multivalent model use a standard system class is to take a document structure tree position and create a redundant description of that place, including its position in the structural tree and offset into the leaf node, and excerpt of the underlying text, a unique identifier of any anchor points, and other information as available. Individual layers are used to store a complex monolithic format, and in practice, layers will be created and modified by various parties at various times. System records in one layer a location in another robustly enough that the location can be predetermined in the face of change. A standard shared location module generates descriptions and resolves previously generated descriptions for locations within the document tree. A location descriptor redundantly records several types of positioning information. When a saved location description is resolved to a runtime document, if the document was unchanged since the description was generated, that same runtime location is guaranteed to be recovered. If the document has changed, the probability of correct registration is inversely proportional to the amount of change. In the face of change, a variety of backoff tactics is applied within each descriptor subtype, and a failure within one subtype drops down to the next, in decreasing order of quality. At the extreme, if a location points to a word and that word's entire chapter is deleted, that logical location no longer exists and cannot be repositioned, and so in this case, the system reports this fact with the associated, now obsolete positioning information to the user [Wilensky 2001].

3.3.1.3 Span Painting

Span insertion, deletion and movement change the value of display properties, by relying on the change of display properties, incremental algorithms reformat and redisplay the span area efficiently. All nodes from the span's start leaf to its end leaf and all their parents up to the root are marked as format-invalid because of change. The Paint protocol will detect the invalid states and reformat them so that this area is to be painted immediately once the span is visible on screen. When the document is reformatted, subtrees at any level that remain valid are reused, and a parent relative coordinate system efficiently flows changed vertical dimensions through the remainder of the document.

When the document is formatted or painted and the tree talk reaches the (point within the) leaf that begins a span, that leaf adds the span to the graphics context's list of active behaviors contributing display properties; it is removed at the (point within the) leaf that ends the span. The graphics context resolves conflicts among behaviors wishing to control the same properties by priority. By default, spans have a (self-reported) priority higher than style sheet behaviors, since spans are overrides of the appearance given by style sheet. Style sheets, spans can both modify any graphics context property [Wilensky 2001].

3.3.1.4 Span Event

Spans can receive two events synthesized by the infrastructure and delivered directly: Enter (span) and Leave (span). Multiple-destination hyperlink and internal hyperlink are all hyperlinks with the similar graphical properties (drawing an underline under covered text, distinguishing in different colors). Upon seeing an Enter span event, the hyperlink changes the cursor to indicate to the user that the cursor is over a hyperlink; likewise, it changes the cursor back upon seeing a redraws itself, and sets a grab in order to receive directly all future events, specifically the MOUSE_UP. Upon seeing the MOUSE_UP, the hyperlink release the grab and jumps to the linked page.

3.3.2 Label

The function "label" is implemented by adding behavior `AddLabelSpan.java` into the framework. `AddLabelSpan.java` extends ActionSpan.java. As behaviors are Java classes that participate in the communication protocols detailed in subsequent sections. Our design is concerned with how to map the desired functionality into protocols by understanding how System framework works. System framework is responsible for calling a behavior at the right time to have its effect, and relinquishing control back to the framework in order to compose well with the other behaviors in the system. Event driven strategy is applied in the Multivalent Architecture to manage behaviors. Ordinarily behaviors wait for something to happen, such as a call to paint itself on the screen or a semantic event. The actions were invoked by the event. After the action, it gives other behaviors a turn.

Now we are talking about how the AddLabelSpan behavior composes other behaviors together by virtue of adhering to the protocols.

Most protocols have `Before` and `After` phases. The reason of division of protocols is to help sequence behaviors. Arbitrary behaviors can be active at any point in the documents lifecycle and might potentially rely on other behaviors. However, behaviors are executed by the order of the behavior priority. Initial priority is set by the order in which behaviors were listed in the hub document. During the `Before` phase, control passes from highest priority down to lowest priority. Higher priority behaviors can establish conditions on which lower priority behaviors can depend. At any point, a behavior may respond with a request to short-circuit the operation; if so, control bypasses any remaining behaviors in the sequence as well as the usual action of the protocol, and proceeds immediately to the same behavior in the `After` sequence. During the `After` phase, which follows the usual action of the protocol in the absence of short-circuiting, control passes from lowest priority to highest. As a result, `After` actions can assume that `Before` and usual actions have been taken, and can modify or "message" the results. It is obvious that higher priority behaviors can message the results of the after actions of lower priority behaviors.

There are two kinds of protocols in the MVD model: round robin and tree based. Round robin protocols flow through the `Before` phases of all active behaviors from highest priority to lowest, then the `After` phase in reverse order. `Restore`, `Build`, `Save` and `SemanticEvent` are all round robin protocols. Tree-based protocols conduct a depth-first tree walk. Tree nodes affects control flows during the tree walk. Internal tree nodes take no actions themselves, but it passes on control to their children iteratively; and leaves implement medium-specific actions. A behavior can registers interest in a node by adding itself to the list of observers kept by that node. Behaviors are only related to a structure portion of the tree register interest to the node at the head of the subtree. The observers of a node have `Before` methods and after methods. During the tree walk, it calls `Before` methods `Before` the node and traverse its children, and calls after methods after the node is done. Behaviors can bypass the subtree by short-circuiting from `Before` to `After`, thus and `After` can short-circuit to cancel the remainder of the tree walk [Phelps, Wilensky 1998].

Behavior takes advantages of base classes by overriding a protocol method. It invokes the superclass implementation with a `super.protocolBefore` (same args) at the beginning of `Before` methods and `super.protocolAfter` (same args) at the end of `After` methods.

Behaviors communicate with one another at a very level via semantic events. Semantic events are passed through all behaviors, and nay behavior can participate. A behavior sends out a semantic event to announce its state, modify the event, update state or request actions: wants to do something, has done something or wants some action performed by another behavior.

3.3.2.1 SemanticEventBefore

At this stage, the `AddLabelSpan` behavior builds the document popup menu for the span and sets menu items: Edit Label name, Delete Span and choose label color.

`CreateUI` behavior, a convenience function for UI building, which returns created widget for further configuration is invoked to create a semantic event to add “Edit Label name” menu item into the document popup menu that is opened when the mouse is over the span and the right button is clicked. When the menu item is clicked, a message “`Span.MSG_EDIT`” can be sent and wait for the actions in Semantic Event `After` stage.

Delete Span button can be added in the same way as above. Semantic event can send a message “`Span.MSG_DELET`”. It invokes `DeleteSpan` behavior to take actions to delete Label span behavior, when repaint the document, the label is deleted and the title is removed from the label menu.

`CreateUI` behavior is invoked to create a series of semantic events to add 8 different color buttons to the document popup menu. When exact color is selected, a `MSG_CHANGE` message is sent which call the function to change the color of the Graphic2D object (label) at `SemanticEventAfter` stage.

`CreateUI` behavior is invoked to create semantic event to add label title in the top up menu “Label” (Label title is set at the stage of `SemanticEventAfter`). Thus, by selecting title, `IScrollPane` behavior sends message to call method `ScrollTo` which scroll scrollbars to (x, y) position or to pass Node. Document content are formatted on demand at the first call to paint. If any part of the node is to be painted, document must be entirely formatted.

3.3.2.2 SemanticEventAfter

At this stage, when the behavior accepts the `SystemEvents.MSG_FORM_DATA` message, it knows that a dialog window was built up inviting user to enter/edit the title for the label. Title can be recorded as string characters, but for convenience, the default content of the title is the content of span itself (The content must have been extracted from the span node in the Paint *`Before`* stage).

It sets new value for the color of the Graphic2D object (label), and then calls Repaint method. The color actually is to be changed in the Paint protocol.

3.3.2.3 Paint Before

Paint protocol like Format, low-level Events protocol is tree-based protocol. Paint `before` is called `Before` observed node has been painted in same coordinate space as node's painting. During this phase, behavior creates Graphic2D object (label), then computes and sets the position by extracting the graphic attributes from Graphic context object. Graphics context object passed from node to node and it holds graphics attributes, list of prevailing `ContextListeners`, random "signals" (name-value pairs). It is guaranteed that the label is attached to span robustly.

When print, it gets span start leaf node and end leaf node from runtime document layer, and walks through from one node to another to get the text of each leaf node until it meets the end leaf of the span. As a result, it gets the content of the span.

3.3.2.4 Appearance

Label is designed as a small rectangle tag with color appearing above the selecting text. The field `cx.spaceabove` is set for the distance between Label and the span. For adding a label, document is re-formatted to leave more space between two lines up and down the label on the base document.

3.3.2.5 Restore

Restore protocol is the first executed protocol. System initiates itself by reading the hub documents. The Restore protocol prepares behaviors and their data for use. System reads any global document attributes, and determines the relevant behaviors and instantiates them with attributes by which behaviors can find their corresponding layers. System creates a runtime program object for each behavior and invokes Restore protocol method of each behavior. Restore usually invokes its superclass, which when it chains up to the base class Behavior, sets the behavior's attributes and adds it to the passed layer.

A behavior’s Restore method is invoked exactly once, and at that time, the behavior should cache any data it needs.

Hub is a XML format document, and it only has one root. Reading hub document is a process of parsing the XML file. Behaviors can be nested in the hub document. In executing protocol, the system restores only the immediate children of the root of the hub, and these children can recursively instantiate their children. When the behavior is restored, it is given a handle to its children.

System loads the behavior form the hub document. The behavior position in document is coming from the location attributes in the hub document. Locations are given in document as robust location since the geometric position of text may vary with such factors as screen size and available fonts. Label has locations pairs, a start and end position. System also initiates the color of the label, span content and label title.

3.3.2.6 Save

The save protocol iterates over the behaviors corresponding to the runtime layers of the document. Invoking their save method. Layers, in turn, iterate over the behaviors nested within them, invoking save. Ordinary behaviors save behavior-specific state in XML syntax. Class `ESISNode` is a simple tree node object for use in building parse tree: attributes, children, write linearzed tree to string. For behavior label, location pairs, color, span content and label title are saved in the hub document.

Here is an example of the document specific hub that includes one label behavior specification.

```
<saved  Behavior='Layer'  URI='systemresource:/sys/About.html'>
<copyed Behavior='multivalent.std.span.AddLabelSpan'  CreatedAt='1057870461410'  content='Bringing
your JavaTM Application to Mac OS'  insert='Bringing your JavaTM Application to Mac OS'
color='Green'    length='43'>
 <start  Behavior='Location'  tree='0  4/Bringing 0/td 1/tr 0/tbody 29/table 0/td 0/tr 0/tbody 2/table 1/td
0/tr 0/tbody 1/table 0/td 0/tr 0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'  context='Bringing br
your' />
```

```
<end Behavior='Location' tree='0 11/X 0/td 1/tr 0/tbody 29/table 0/td 0/tr 0/tbody 2/table 1/td 0/tr
0/tbody 1/table 0/td 0/tr 0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html' context='X OS br' />
  </copyed>
</saved>
```

3.3.3 Multiple destination hyperlink

Similarly, the function Multiple destination hyperlink" is implemented by adding behavior `MCHyperlinkSpan.java` into the framework. `MCHyperlinkSpan.java` extends Span.java.

3.3.3.1 Semantic Event Before

At this stage, the `MCHyperlinkSpan` behavior builds the document popup menu for the span and sets menu items: Delete Span, Add Link URI, Delete Link URI and a list of links with title.

`CreateUI` behavior is invoked to create a semantic event to add "Add Link URI" menu item into the document popup menu that is opened when the mouse is over the span and the right button is clicked. When the menu item is clicked, a message "`Span.MSG_ADDITEM`" can be sent and wait for the actions in `SemanticEventAfter` stage.

`CreateUI` behavior is invoked to create a semantic event to add "Delete Link URI" menu item into the document popup menu that is opened when the mouse is over the span and the right button is clicked. When the menu item is clicked, a message

“`Span.MSG_DELETEITEM`” can be sent and wait for the actions in `SemanticEventAfter` stage.

Delete Span button can be added in the same way as above. Semantic event can send a message “`Span.MSG_DELET`”. It invokes DeleteSpan behavior to take actions to delete Multiple Destination Hyperlink span behavior at the repaint stage.

`CreateUI` behavior is invoked to create a list of semantic events to add link buttons to the document popup menu. Each URL link is saved in an array with a title to identify it. When exact one is selected, it invokes `Document.MSG_OPEN` event.

3.3.3.2 SemanticEventAfter

At this stage, when the behavior accepts the `SystemEvents.MSG_FORM_DATA` message, it knows that a dialog window was built up inviting user to add/delete a new hyperlink in the Multiple-destination hyperlink group. When the `MSG_DELETITEM` message is received, for deleting a link, link title should be provided. System will remove the hyperlink with that title from the multiple-destination hyperlink list. For adding a hyperlink, link title and link URI should be asked. Title can be recorded as string, but for convenience, the default content of the title is the content of span itself (Content was extracted from the span node in the Paint `Before` stage), URIs are Uniform Resource Identifies which are short strings that identify resources in the web: document, images, downloadable files, services, electronic mailboxes, and other resources. It makes resources available under a variety of naming schemes and access methods such as HTTP, FTP, and Internet mail addressable in the same way. The hyperlink title and URI will be saved in the Arraylist in pair, and URI will be encoded to set as target.

When the message `MSG_COPY_LINK` is received, the selected link can be copied to the system clipboard.

When the message `MSG_DELETE` is received, the system will destroy multiple-destination hyperlink span object (System also changes the related content of hub document at the stage of save protocol).

3.3.3.4 Appearance

Multiple-destination hyperlink span is `contextListeners` which is the behaviors that compose together to determine the Context display properties at every point in the document. A piece of text in a document has many properties such as the font family, size, style, foreground and background colors. A combination of `ContextListeners` represents the influence of style sheet settings, built-in span settings. Here, Multiple-destination hyperlink hardcodes the action of coloring the text and gives it an underline, choosing either magenta or red.

3.3.3.5 Event

Once the grab is set, subsequent events go directly to Event protocol, not to `EventBefore/EventAfter`. An intervening `MOUSE_DRAG` intiates a selection. First, it aborts the Multiple-destination hyperlink by redrawing the link and releasing the grab. Then it synthesizes and sends to the document a `MOUSE_Down` event; as this event is seen by this very link, the fact that the link's active flag is still set indicates that it was self-generated and should be ignored. Finally, the hyperlink turns off its active flag and resends the `MOUSE_DRAG` event through the document; since the selection behavior has in the meantime set the grab, it receives the event directly.

3.3.3.6 EventAfter

As a span, multiple-destination hyperlink receives low-level events in the region of the document it spans without additional registering. multiple-destination hyperlink changes the cursor shape when it is over the hyperlink and restore it when the cursor moves off. The synthesized ENTER event from the system saves the current cursor shape, sets the hyperlink shape, and sets the system status message as multiple destination hyperlink; LEAVE restores the cursor shape and erases the message. On a `MOUSE_DOWN` event, the hyperlink sets the system grab, sets its private active flag, and redraws the span in a different color to indicate a pending link. A `MOUSE_UP` without intervening `MOUSE_DRAG` redraws the link in its original color, releases the grab, and calls the system to show the status message.

3.3.3.7 Restore

Similar as Label span, all data about a Multiple-destination hyperlink behavior can be loaded at the restore stage. Locations pairs span content and a list of hyperlink destination title and corresponding link URIs can be extracted from the document-specific hub. At this stage, runtime data structure is also be created.

3.3.3.8 Save

In the same way, for behavior Multiple-destination Hyperlink, location pairs, hyperlinks and corresponding destination titles and span content is saved in the hub document.
Here is an example of the document specific hub that includes one Multiple-destination hyperlink behavior specification.

```
<saved  Behavior='Layer'  URI='systemresource:/sys/About.html'>
 <mchyperlink  Behavior='multivalent.std.span.MCHyperlinkSpan'  CreatedAt='1075143931587'
link='+Java 2 Platform, Standard
Edition#http://java.sun.com/j2se#+J2EE#http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci
283984,00.html#+Java Platform 2 for Linux#http://www.blackdown.org/java-linux/java2-status/#'>
  <start  Behavior='Location'  tree='0  0/The 0/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 0/td 0/tr
0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'  context='The  Java' />
  <end  Behavior='Location'  tree='8  3/Platform 0/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 0/td 0/tr
0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'  context='Platform 2 provides' />
   </mchyperlink>
</saved>
```

The hyperlinks of the multiple-destination are saved in the attribute field “link”. link URI follows with link title separated with character “#”, and hyperlinks follows one by one separated with character “+”. As we see that this Multiple-destination hyperlink includes three hyperlinks:

Destination title	**Destination link URI**
Java 2 Platform, Standard Edition	http://java.sun.com/j2se
J2EE	http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci283984,00.html
Java Platform 2 for Linux	#http://www.blackdown.org/java-linux/java2-status

When the system framework restores the behavior, it extracts the attributes from the hub document and creates runtime data structure Arraylist. All operations are manipulated on the Arrarylist including adding new links; deleting existing links etc. finally, at the Save stage, information about link definition is combined together and written to the attribute field link in the hub document.

3.3.4 Internal hyperlink

Internal hyperlink is implemented by three behaviors: `SpanUI.java`, `TargetAnchorSpan.java` and `InternalHyperlinkSpan.java`. `InternalHyperlinkSpan.java` is a span behavior with similar functions as `HyperlinkSpan`. Instead of providing a URI and going to another document on the same website or on another website, internal hyperlink is a direct connection from the current location of the cursor to another location in the current document. `Before` creating an internal hyperlink, you have to create an anchor. Anchor (target) creates a special marker inside the page, which it can link to. The link you create in the will contain both the name of the file you are linking to and the name of that anchor. Then the browser will then jump to the location of the anchor. Generally, the connection between the hyperlink and target is represented by the html code. The browser reads the html code and parses it to build document tree, the connection between two locations can be reflected constantly by relationship of the two parts of the subtree. Here, we create annotation internal hyperlink on the published WebPages on the web browser, not going to modify the original html code, but building the connection on the runtime document tree on the client side and saving it to the hub document in order to reloading it when the browser accesses this web page again.

3.3.4.1 SpanUI

`SpanUI` is the class that creates an instance of span described by hub attributes and move to extent of current selection. Every type of span we mentioned above is created by `SpanUI`. `SpanUI` constructs the menu in the current browser instance getting data such as layer information, logical span name from the hub attributes. At the same time, it set the “`ATTR_CREATEDAT`” attributes for the span that records creation time, as given by `System.currentTimeMillis()`. Creation time can be used as a unique identity to find a behavior in the hub document because there is a big possibility of repetition if we identify a special span by using span content or span name in the hub document.

A basic class in Multivalent model is `Browser.java`. This class implements a browser window. It interfaces with the operating systems GUI, holds a document tree with both user interface and content, manages flow of control through associated behaviors according to the protocols, and holds resources shared among all documents in a window.

By using static method getBrowser(), each behavior can access the public field and method in this class. we modified the Browser.java class by add a new public field String currentanchor_ which records the last anchor set currently. In the SpanUI class, at the stage of SemanticEventAfter, when a new span is created, and the span type is TargetAnchorSpan, the currentanchor_ variable is modified as the identify of the new anchor (ATTR_CREATEDAT). When InternalHyperlinkSpan is created, put its ATTR_TARGET attribute as the value of currentanchor_. As we see, we build the relationship between the target anchor and internal hyperlink span.

3.3.4.2 Target Anchor Span

TargetAnchorSpan behavior is responsible for setting anchor span. Anchor span can be differentiated by having a small red dot symbol above the selecting text when anchor is set. The symbol red dot tells that this is an anchor. The field cx.spaceabove is set for the distance between the symbol and the span. For printing the symbol, document is re-formatted to leave more space between two lines up and down the label on the base document at the PaintBefore stage.

In the class Browser.java, we defined a public Map variable targetanchors_ in order for each behavior to access. Map is an object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. At the stage of restore of each TargetAnchorSpan, the TargetAnchorSpan loads itself into the Map targetanchors_ by getBrowser().putVar(get(ATTR_CREATEDAT), this). As we see, targetanchors_ holds all anchors created by the user in this browser instance. By giving the key, we can get specific anchor span quickly. At SemanticEventAfter stage, if `MSG_DELETE` is caught, this anchor span can be removed quickly from the Map `targetanchors_` by geBrowser().`removeVar(get(ATTR_CREATEDAT))`. All anchor spans saved in the hub document at the stage of Save.

3.3.4.3 InternalHyperlinkSpan

InternalHyperlinkSpan functions like a hyperlink span, but the difference is that the target is not a URI but an anchor span. When the link is clicked, it creates a `SemanticEvent (br,IScrollPane.MSG_SCROLL_TO,(TargetAnchorSpan)br.getVar((this.getAttr( ATTR_TARGET)))`. The internal hyperlink finds the key of the anchor it relates to by getting attributes from the hub documents, and then gets the target span from the Map `targetchors_` by providing the key.

Here is an example of the record in the hub document.

```
<saved  Behavior='Layer'  URI='systemresource:/sys/About.html'>
  <targetanchor  Behavior='multivalent.std.span.TargetAnchorSpan'  CreatedAt='1078796585152'>
    <start  Behavior='Location'  tree='0  1/Highlights 0/td 0/tr 0/tbody 29/table 0/td 0/tr 0/tbody 2/table
1/td 0/tr 0/tbody 1/table 0/td 0/tr 0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'
context='Highlights br p' />
    <end  Behavior='Location'  tree='10  1/Highlights 0/td 0/tr 0/tbody 29/table 0/td 0/tr 0/tbody 2/table
1/td 0/tr 0/tbody 1/table 0/td 0/tr 0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'
context='Highlights br p' />
  </targetanchor>
  <internalhyperlink  Behavior='multivalent.std.span.InternalHyperlinkSpan'
CreatedAt='1078796594345'  content='JavaCommunity'  length='14'  target='1078796585152'>
    <start  Behavior='Location'  tree='0  33/Java 0/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 0/td 0/tr
0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'  context='Java br Community' />
    <end  Behavior='Location'  tree='9  34/Community 0/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table
0/td 0/tr 0/tbody 0/table 5/td 0/tr 0/tbody 1/table 1/body 0/html'  context='Community Java p' />
  </internalhyperlink>
</saved>
```

3.3.5 Enable all links to work in the Post-it Notes

Post-in Note is an annotation developed with the Multivalent Document Model. The Note is not like generally annotation that has a location given in structural coordinates, but in physical coordinates.

The Note can hold not only texual content, but also annotations such as hyperlink, multiple-destination hyperlink, internal hyperlink, label we developed. The floating note is an embedded document, so it can be annotated itself. As in Post-it Notes where the text of the note is stored inline; or it can be nested behaviors, as in span annotations that nest a pair of Location. Annotations need to find the correct document root when restored in the note. Thus, Notes place their unique root names in the global namespace paired with a handle to the instantiated root node. When saved, annotations save their documents root name, if it is not the default name.

General objects can be kept in a separate global namespace. Notes, each with an individual, full-featured document tree, store mappings between their names and the root of their associated tree, so that when annotations are stored with reference to the Note name, they can when restored find the appropriate document root. Intra-document anchors associate the name of the anchor with the corresponding node to jump quickly to the anchor position without searching the entire tree for it.

Here is an example of the record in the hub document.

```
 <note  NAME='NOTE1744132702'  Behavior='Note'  X='315'  Y='363'  Width='300'  Height='100'>
<content> The java 2 platform includes three editions \nenterprise edition \nStandard edition \nmicro
edition   </content>
    <personal  Behavior='Layer'>
 <copyed  Behavior='multivalent.std.span.AddLabelSpan'  CreatedAt='1075220267151'
content='Standardedition'  insert='the Second one '  color='Red'   length='16'>
        <start  Behavior='Location'  tree='0  0/Standard 2/line 0/body 0/ed'  context='Standard  edition' />
        <end  Behavior='Location'  tree='7  1/edition 2/line 0/body 0/ed'  context='edition Standard ' />
      </copyed>
      <hyperlink  Behavior='HyperlinkSpan'  URI='http://java.sun.com'  CreatedAt='1075220340486'>
        <start  Behavior='Location'  tree='0  0/micro 3/line 0/body 0/ed'  context='micro  edition' />
        <end  Behavior='Location'  tree='7  1/edition 3/line 0/body 0/ed'  context='edition micro ' />
      </hyperlink>
    </personal>
  </note>
```

Chapter 4 A Worked Example

In this chapter, we will provide a worked example that shows the application of new linking annotation tools. The worked example is presented through a sequence of chronologically arranged screen shots.

4.1 Label

Some web pages are quite long and difficult to locate a specific place when you scroll the scroll bar. Label enables the browser to have a list of bookmarks, which are pointers to interesting places in the current webpage. To go to a bookmarked location, begin from the Navigator window:

1. Click to place the insertion point where you want to create a label, and select some text.
2. Open the Label menu and choose Add Label. You should see the Label Properties dialog box.
3. Type a unique name for the Label in the Label Name field. For convenience, the default name is set as the content of the selected text automatically.
4. Click OK. A label icon appears above selected text in the document to mark the anchor's location, and a label menu item named with Label name is added into the label menu.

Here, we give an example of adding a label in the document. We select a text and choose Add Label command from the Label menu. See the screen shot Figure 4.1.1.`

Figure 4.1.1 Add a label

Then, the dialog box prompts up and we named the label as “An expectative summary of .NET vs. J2EE”. Select “OK” and get the label icon above the text and a new menu item is added into the menu “Label” (see Figure 4.1.2).

Figure 4.1.2 Name the label

After the label is created, we can change its properties anytime when we put the mouse over the span text and click mouse right button (see Figure 4.1.3).

Figure 4.1.3 Change the properties of the label

We can change the color of the label from the color list and edit the name of the label. If we do not want this label anymore, we can easily remove it from the document by selecting Delete Span command from the mouse right button menu. After removing, all label related features are deleted, and related recording is erased from the hub document in the user's home directory at the path `system.getProperty("users.dir")/.Mutivalent/hub`.

Figure 4.1.4 Select the label from the Menu

Figure 4.1.5 jump to the label anchor location

After clicking the item “An expectative summary of .NET vs. J2EE”, the scroll bar scrolls accurately to the location of the label span (see Figure 4.1.4 and Figure 4.1.5).

Label can work well for the active reading for structuring the information. Related information can be sorted and grouped together in the label menu easily for the reader to find each topic. Here is an example of the label use in the way of information structure (see Figure 4.1.6).

Figure 4.1.6 work for structuring the information

4.2 Multiple-destination hyperlink

Working as the reader’s citation link, Multiple-destination hyperlink connects multiple external sources to the specific content in the document. The external sources can be added following the steps below:

1. Click to place the insertion point where you want to create a label, and select some text.
2. Open the Link menu and choose Multiple Destination Hyperlink. You see the Properties dialog box.
3. Type a unique name for the Label in the Link Name field and exactly URI for the destination.

4. Click OK. A red underline appears under the selected text in the document to mark as Multiple-destination hyperlink, and a new link is added into the link list of this Multiple-destination hyperlink.

The following screen shots will show you the process of creating a new Multiple-destination hyperlink (see Figure 4.2.1).

Figure 4.2.1 Create a new multiple-destination hyperlink

Select a text span “J2EE”and choose command Multiple-destination hyperlink from the Link menu. A new Multiple-destination hyperlink is created with a red underline under the text.

Figure 4.2.2 Enter link name and URI for the destination.

In the dialogue box, Type link name and its URI into the Name field and URI field, and then click “OK” (see Figure 4.2.2).

Figure 4.2.3 Multiple-destination hyperlink document popup menu

When put the mouse over the Multiple-destination span, and click right button, a document popup menu will appear to allow the user to change the properties of the link. You can add new links by selecting “Add new link” command which can prompt a add link dialogue box, and you can delete a specific link by typing a link name to find the link you want to remove. If the link name exists, it can be deleted successfully, otherwise, an error massage “The specific link name was not found, and the delete operation was denied” will be given in the state information field on the bottom of the web browser. If it happens, you must type the collect link name again. Similar as the command “Delete Span” of the label, we allow the user to delete Multiple-destination hyperlink span totally from the document (see Figure 4.2.3).

Now we manipulate the process of adding a list of links into the Multiple-destination hyperlink, and deleting a specific link from the list (see Figure 4.2.4, 4.2.5, 4.2.6).

Figure 4.2.4 A list of links in the Multiple-destination hyperlink

Figure 4.2.5 Remove one link “Java and J2EE” from the list

Figure 4.2.6 A list of links after removing link “Java and J2EE”

Figure 4.2.7 Click a link in the list of the Multiple-destination hyperlink

After selecting destination, it jumps to another web page (see Figure 4.2.7 and 4.2.8).

Figure 4.2.8 web browser loads another web page.

The system provides controls for backtracking. You can go back or forward one page by clicking the Back or Forward arrow on the tool bar or selecting Back or Forward command from the menu Go.

4.3 Internal hyperlink

Internal hyperlink is to create a link within the same page. The reader can use it to jump from one section to another. You must create an anchor (target location) first, and then create a link that point to the anchor. Follow these steps to create an Internal hyperlink.

1. Click to place the insertion point where you want to create an anchor, and select some text.
2. Select command "Set as Target" in the menu Link. A red dot appears above the text to show that an anchor was created successfully.
Select the text that you want to link to the anchor.
3. Open the menu Link; click the Internal hyperlink command, then a green underline will appear under the link text span.

Now we give an example to show you how it works. We still use this web page to show how internal hyperlink benefits the active reading. At the beginning of this article, the author presents his idea by raising two questions, and the following part of the article concentrates on answering these two questions. The two questions work as an index helping the reader structure the information. For the long article, if we build the connection between the index question and answers, it will be very easy for the reader to navigate and find the location of the answers (see Figure 4.3.1, 4.3.2, 4.3.3, 4.3.4).

Figure 4.3.1 select text and set it as target

Figure 4.3.2 Create an Internal hyperlink

Figure 4.3.3 Click an Internal hyperlink

Figure 4.3.4 Jump to the anchor location

4.4 Make all annotations work on the Post-it   Notes

Post-it   Note is another annotation tool developed with the MVD model. We can easily create a note by clicking Note command on the Anno menu. And move note around the web page to select a place to put it (see Figure 4.4.1).

Figure 4.4.1 Create a new note

After creating a Post-it   Note, you can type content into the note. The content can be edited in the note window. We also can easily change the properties of the note by the window size and note background color. Note can be moved and it also can be pinned to the document. A menu lists the properties of the note, and you can click the note title bar by mouse right button to select each propriety item. We can close the note window by clicking the close button on the right corner of the note window title bar. It can be reopened by selecting note title from the Anno menu. The default note title is the first word in the note. The scroll bar is set automatically when the window size is small and not enough to show all the content (see Figure 4.4.2).

Figure 4.4.2 Show a note with content

Figure 4.4.3 Note with Multiple-destination hyperlink and label

For example, we add a Multiple-destination hyperlink and a label in the same way as we described before. We see that they all works well in the note (see Figure 4.4.3, 4.4.4, 4.4.5).

Figure 4.4.4 Select a hyperlink from the Multiple-destination hyperlink in the note

Figure 4.4.5 Jump to a new web page

4.5 Save

Annotation can be recorded in the hub document, and hub document is saved to the user's home directory automatically. But the reader also can save it anytime at any place by selecting the Save Anno As command from the menu File (see Figure 4.5.1, 4.5.2).

Figure 4.5.1 select Save command from the menu

Figure 4.5.2 Save annotations in the hub document

Here is the document specific hub document:

```
<saved  Behavior='Layer'  URI='http://java.oreilly.com/news/farley_0800.html'>
  <copyed  Behavior='multivalent.std.span.AddLabelSpan'  CreatedAt='1075693049664'
content='Portability:'  insert='Compare features of .NET and J2EE -- .NET'  color='Green'
    length='13'>
      <start  Behavior='Location'  tree='0  0/Portability%3A 99/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody
1/table 1/body 0/html'  context='Portability%3A  The' />
      <end  Behavior='Location'  tree='0  1/The 99/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 1/body
0/html'  context='The Portability%3A .NET' />
  </copyed>
  <copyed  Behavior='multivalent.std.span.AddLabelSpan'  CreatedAt='1075693163197'  content='J2EE,
on the other hand,'  insert='Compare feautres of .NET and J2EE -- J2EE'  color='Green'
    length='25'>
    <start  Behavior='Location'  tree='0  0/J2EE%2C 101/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody
1/table 1/body 0/html'  context='J2EE%2C  on' />
    <end  Behavior='Location'  tree='0  5/works 101/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table
1/body 0/html'  context='works hand%2C on' />
  </copyed>
  <copyed  Behavior='multivalent.std.span.AddLabelSpan'  CreatedAt='1075693629317'  content='For
Microsoftdevelopers'  insert='Response -- for Microsoft developers'  color='Orange'
   length='24'>
    <start  Behavior='Location'  tree='0  85/For 102/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table
1/body 0/html'  context='For h3 Microsoft' />
    <end  Behavior='Location'  tree='10  87/developers%3A 102/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr
0/tbody 1/table 1/body 0/html'  context='developers%3A Microsoft ' />
  </copyed>
  <copyed  Behavior='multivalent.std.span.AddLabelSpan'  CreatedAt='1075693686830'  content='For
the Java and Open Sourcecommunities:'  insert='Response -- For the Java and Open Source communities'
color='Orange'
   length='41'>
    <start  Behavior='Location'  tree='0  0/For 107/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table
1/body 0/html'  context='For  the' />
    <end  Behavior='Location'  tree='12  6/communities%3A 107/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr
0/tbody 1/table 1/body 0/html'  context='communities%3A Source ' />
  </copyed>
  <mchyperlink  Behavior='multivalent.std.span.MCHyperlinkSpan'  CreatedAt='1075735114350'
link='+Comparing .Net framework performance and scalability to J2EE Application
Servers#http://gotdotnet.com/team/compare/middleware.aspx#'>
    <start  Behavior='Location'  tree='0  4/J2EE 23/h3 95/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody
1/table 1/body 0/html'  context='J2EE and compare%3F' />
```

<end Behavior='Location' tree='0 5/compare%3F 23/h3 95/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 1/body 0/html' context='compare%3F J2EE ' />

</mchyperlink>

<targetanchor Behavior='multivalent.std.span.TargetAnchorSpan' CreatedAt='1078720405700'>

<start Behavior='Location' tree='0 0/How 23/h3 95/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 1/body 0/html' context='How do' />

<end Behavior='Location' tree='8 5/compare%3F 23/h3 95/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 1/body 0/html' context='compare%3F J2EE ' />

</targetanchor>

<internalhyperlink Behavior='multivalent.std.span.InternalHyperlinkSpan' CreatedAt='1078720421262' content='How does the .NET architecture measure up againstJ2EE?' length='55' target='1078720405700'>

<start Behavior='Location' tree='0 0/How 1/li 0/ul 10/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 1/body 0/html' context='How does' />

<end Behavior='Location' tree='5 8/J2EE%3F 1/li 0/ul 10/p 1/td 0/tr 0/tbody 2/table 1/td 0/tr 0/tbody 1/table 1/body 0/html' context='J2EE%3F against ' />

<note NAME='NOTE529078907' Behavior='Note' X='223' Y='96' Width='227' Height='213'>

<content>

Reader about the Middleware company Application Server Case Study, and how .NET delivered better productivity and performance than two leading J2EE application servers on variety of independently-conducted scalability benchmarks. </content>

<personal Behavior='Layer'>

<mchyperlink Behavior='multivalent.std.span.MCHyperlinkSpan' CreatedAt='1075785039662' link='+Comparing .Net framework performance and scalability to J2EE Application Servers#http://gotdotnet.com/team/compare/middleware.aspx#'>

<start Behavior='Location' tree='0 3/Middleware 0/line 0/body 0/ed' context='Middleware the company' />

<end Behavior='Location' tree='5 8/Study%2C 0/line 0/body 0/ed' context='Study%2C Case and' />

</mchyperlink>

<copyed Behavior='multivalent.std.span.AddLabelSpan' CreatedAt='1075785068254' content='benchmarks.' insert='benchmarks.' color='Red'

length='11'>

<start Behavior='Location' tree='0 28/benchmarks. 0/line 0/body 0/ed' context='benchmarks. scalability ' />

<end Behavior='Location' tree='10 28/benchmarks. 0/line 0/body 0/ed' context='benchmarks. scalability ' />

</copyed>

</personal>

</note>

</saved>

Chapter 5 Conclusion and Future work

5.1 Practical Benefits of the Tool

In our project, new linking annotation patterns are raised and implemented to meet the requirements for active reading in scholarly work. Label, Multiple-destination hyperlink and Internal hyperlink are not tools for composing a web page and publishing it on the server side, but for annotating the document in the web browser on the clients. System framework makes new linking annotation tools appear in situ. It also meets this requirement by having the individual annotative behaviors rely upon the geometric placement information of behavior instances available in the format stage of the document life cycle. Annotation are attached to a particular component or series of components, and then placed in relation to them. Placement is managed by the multivalent framework calling a behavior method at the right time. Another feature of these digital annotations is that they present to the reader a different (personalized) version of the document than is stored on the server, and it is easy for the reader to create these personalized versions.

We designed a simple graphic interface to help the user to access and use all tools without considering server-side requirement and how annotation can be recorded for reuse. These three link annotation tools give the users flexibility to edit or remove text and links and to change presentation prosperities in multiple media types Because multivalent frameworks support format-independent (behaviors manipulate the abstract document tree and communicate to encapsulated media types through media adapters). In addition, as the infrastructure is written in Java, it is platform-independent. Our linking annotation tools can help readers set their path-following policy effectively, navigate through electronic document at will, and improve the reader's interaction with the hypertext.

5.2 Future Features

We have some new linking annotation tools on the web browser, and each tool can be manipulated individually. This feature is easy for the user to operate on each tool. However, as the document becomes longer, and more and more annotation tools are applied on the document, many types of annotation are diverse all over the document. If the readers have a need to deal with a group of annotations with the same type, they have to find each one browsing through the long document and operating on each one with same actions. For example, if the readers have a desire to change the color of all labels or notes, they have to find each one and change the property of the color. Furthering our goal is to design an annotation management center. It is an interface to group all annotation by type to manage them easier. We allow the reader to know how many annotations of each pattern they have, what is their name and other property, where there are located in the manager center. Manager Center can not only support changing the propriety of individual annotation, but also support the reader' fast locating to each annotation and changing common properties for arbitrarily a group of annotations.

Reference

Adler, Gujar, Harrison, O'Hara and Sellen 1998. A diary study of work-related reading: Design implications for digital reading devices. In *Proceedings of CHI'98 Human Factors in Computing Systems*, Los Angeles, California, USA, volume 1 of Reading and Writing, 241-248

Bernstein 1991.Deeply intertwingled hypertext: The navigation problem reconsidered. Technical Communication, pages 41-47

Bucur, Johanna 1999. The HyLink Framework: A Study of Link Performance. *In: Nürnberg, Peter (ed.). ACM Hypertext '99 Doctoral Consortium Final Report*

Catherine, Marshall, Price, Golovchinsky and Schilit 2001. Designing e-books for legal work**.** *ACM/IEEE Joint Conference on Digital Libraries*

Catlin, Bush, and Yankelovich 1989. InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration, *In Conference on Hypertext and Hypermedia Proceedings of the second annual ACM conference on Hypertext***,** Pittsburgh, Pennsylvania, United States, 365 – 378

Charney, D. 1994. The effect of hypertext on processes of reading and writing. In *Literacy and computers: The complications of teaching and learning with technology,* Selfe, C.L. and Hilligoss, S, Ed. The Modern Language Association of America, New York, 238-263

Cousins, Baldonado, and Paepcke 2000. A Systems View of Annotations**.** Technical Report P9910022, Xerox PARC.

DeRose 1989. Expanding the notion of links. *In Proceedings of the Hyper- text '89 Conference on Hypertext*, Pittsburgh, Pennsylvania, USA, 249-257

Gottlob, Slany 1994. The Use of Intelligent Hypermedia in Architectural Design Environments – a Conceptual Framework
URL: http://www.dbai.tuwien.ac.at/staff/herzog/thesis/dip.html

Golovchinsky 2002. *Going back in hypertext. Conference on Hypertext and Hypermedia***,** *Proceedings of the thirteenth ACM conference on Hypertext and hypermedia* College Park, Maryland, USA SESSION: Links Pages: 82 – 83, ISBN: 1-58113-477-0

Idaho State University 1995. The World Wide Web

URL: http://www.isu.edu/departments/comcom/internet/www.html

Levene and Loizou 2002.Web interaction and the navigation problem in hypertext**.** *In A. Kent, J.G. Williams, and C.M. Hall, editors, Encyclopedia of Microcomputers*. Marcel Dekker, New York, NY

Lexico Publishing Group, LLC 2004. Dictionary.com
URL: http://dictionary.reference.com/

Marshall, C.C., Price, M.N., Golovchinsky, G., and Schilit, B.N. 1999. Introducing a digital library reading appliance into a reading group. *In Proceedings of ACM Digital Libraries* 99. ACM Press, New York, 77-84

Marshall, C. 1998. Toward an ecology of hypertext annotation, *in Proceedings of ACM Hypertext '98*, ACM Press, 40-49

Marshall, C. 1997. Annotation: from paper books to the digital library. *In Proceedings of the ACM Digital Libraries '97 Conference*, Philadelphia, PA

Marshall, C. 1998. The Future of Annotation in a Digital (Paper) World. I*n Proceedings of The 35th Annual GSLIS Clinic: Successes and Failures of Digital Libraries*, University of Illinois at Urbana-Champaign

M. Levene and G. Loizou 2002. Web interaction and the navigation problem in hypertext. *In A. Kent, J.G. Williams, and C.M. Hall, editors, Encyclopedia of Microcomputers*. Marcel Dekker, New York, NY

O'Hara and Sellen, Abigail. 1997. *A Comparison of Reading Paper and On-Line Documents. In Proceedings of CHI'97*, ACM Press, New York, 335-342

Phelps, Wilensky 1998. Multivalent Documents: Anytime, Anywhere, Any Type, Every Way User Improvable Digital Documents and Systems. PhD. Dissertation

Pimentel, Abowd, and Ishiguro 2000. Linking by interacting: a paradigm for authoring hypertext. *In Proceedings of the eleventh ACM on Hypertext and Hypermedia*, 39-48

Price, Golovchinsky, and Schilit 1998. Linking By Inking: Trailblazing in a Paper-like Hypertext. In *Proceedings of Hypertext '98*, ACM Press, pp. 30-39.

Phelps & Wilensky 1998 Multivalent Documents: A New Model for Digital Documents, Technical Report, CSD-98-999 (1998)

Roberts & Etherington 2003. A Dictionary of Descriptive Terminology
URL: http://sul3.stanford.edu:10001/

Schilit, Bill, Price, Morgan, and Golovchinsky 1998. *The Digital Library Information Appliance*. *In Proceedings of ACM Digital Libraries '98*. New York: ACM Press.

Schilit, Price, and Golovchinsky 1998. Library Information Appliances. In *Proceedings of Digital Libraries '98,* ACM Press, Pittsburgh, PA

Schilit, Golovchinsky and Price 1998. Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations. *In Proceedings of CHI98*, Los Angeles, CA, ACM Press.

Simon Kampa, Timothy Miles-Board, Les Carr, and Wendy Hall 2001. Linking with meaning: Ontological hypertext for scholars. Technical Report 0-854327-37-1, University of Southampton, Southampton, UK,

Simpson, J.AND Weiner, E.(Eds.) 1993. *Oxford English Dictionary Additions Series, Volume 2,* Clarendon Press, New York, 38.

Wilensky 1997. Multivalent Annotations. *Proceedings of First European Conference on Research and Advanced Technology for Digital Libraries*

Wilensky 2001. *The Multivalent Browser: A Platform for New Ideas*. *In Proceedings of Document Engineering 2001*, Atlanta, Georgia.

Appendix

A User Guide

1. Label

2. Multiple-destination hyperlink

3. Internal hyperlink

1. Label

1.1 Creating a label

To create a label in a web page or in a Post-in note, you must create an *anchor* (target location), and then add a label in the Label menu that points to the anchor.

1. Click to place the insertion point where you want to create a label, and select some text.
2. Open the Label menu and choose Add Label. You see the Label Properties dialog box.
3. Type a unique name for the Label in the Label Name field. For convenience, the default name is set as the content of the selected text automatically.
4. Click OK. A label icon appears above selected text in the document to mark the anchor's location, and a label menu item named with Label name is added into the label menu.

1.2 Selecting a label

Open the Label menu and click the label name that you want to choose. The system responses your action by scrolling the scroll bar accurately to the location of the label spans.

1.3 Editing label name

1. Put the mouse over the label span, click right mouse button and select Edit Span command from the pop-up document menu.
2. The Label Properties dialog box will pop up. Delete the label name in the Label Name field, and type a new unique name again. Click Ok button.

1.4 Changing the color of the label

Put the mouse over the label span, click right mouse button and select color from the pop-up document menu. There are eight colors available to the label: yellow, orange, green, blue, red, black, white, and gray.

1.5 Deleting a label

Put the mouse over the label span, click right mouse button and select Delete Span command from the pop-up document menu.

2. Multiple-destination hyperlink

2.1 Creating a Multiple-destination hyperlink

1. Click to place the insertion point where you want to create a label (in a web document or in a Post-it Note), and select some text.

2. Open the Link menu and choose Multiple Destination Hyperlink. You see the Properties dialog box.

3. Type a unique name for the hyperlink in the Link Name field and exactly URI for the destination.

4. Click OK. A red underline appears under the selected text in the document to mark as Multiple-destination hyperlink, and a new link is added into the link list of this Multiple-destination hyperlink.

2.2 Selecting a hyperlink from the Multiple-destination hyperlink

Put the mouse over the Multiple-destination hyperlink span, and click right mouse button. A pop up menu will show up which lists the hyperlinks. Click the hyperlink name of the destination, and then the new web page is loaded in the web browser.

2.3 Adding a new hyperlink

1. Put the mouse over the Multiple-destination hyperlink span, click right mouse button and select Add new Link command from the pop-up document menu.

2. The Properties dialog box will pop up. Type a unique name for the hyperlink in the Link Name field and exactly URI for the destination.

3. Click Ok. A new link is added in the list of the multiple-destination hyperlink, and a message “the new link *** was added” in the state information field on the bottom of the web browser.

2.4 Deleting a hyperlink

1. Put the mouse over the Multiple-destination hyperlink span, click right mouse button and select Add new Link command from the pop-up document menu.

2. The Properties dialog box will pop up. Type the name of hyperlink which wishes to be deleted.

3. Click Ok. When the name exists, the link can be deleted successfully with a message “hyperlink *** was deleted in the state information field on the bottom of the web browser, otherwise, the deletion operation is denied with an error massage “The specific link name was not found, and the deletion operation was denied.”

2.5 Deleting a Multiple-destination hyperlink span

Put the mouse over the Multiple-destination hyperlink span, click right mouse button and select Delete Span command from the pop-up document menu.

3 Internal hyperlink

3.1 Creating an Internal hyperlink

Internal hyperlink is to create a link within the same page. The reader can use it to jump from one section to another. You must create an anchor (target location), and then create the link that point to the anchor.

1. Click to place the insertion point where you want to create an anchor (in a web document or in a Post-it Note), and select some text.
2. Select command “Set as Target” in the menu Link. A red dot appears above the text to show that an anchor was created successfully.
3. Select the text that you want to link to the anchor.
4. Open the menu Link; click the Internal hyperlink command, then a green underline will appear under the link text span.

3.2 Deleting an Internal hyperlink

Put the mouse over the target anchor span that the internal hyperlink points to, click right mouse button and select Delete Span command from the pop-up document menu.

Put the mouse over the Internal hyperlink span, click right mouse button and select Delete Span command from the pop-up document menu.

B API

1. AddLabelSpan

2. MCHyperlinkSpan

3. TargetAnchorSpan

4. InternalHyperlinkSpan

multivalent.std.span

Class AddLabelSpan

```
java.lang.Object

  |

  +--multivalent.VObject

        |

        +--multivalent.Behavior

              |

              +--multivalent.Span

                    |

                    +--multivalent.std.span.AddLabelSpan
```

All Implemented Interfaces:

ContextListener, EventListener, java.util.EventListener

public class **AddLabelSpan**
extends Span

Copy editor markup: insert text at point.

Version:

$Revision: 1.3 $ $Date: 2004/02/02 13:16:27 $

Field Summary

public static final String	**ATTR_INSERT**
public static final String	**ATTR_CHANGE**
public static final String	**ATTR_COLORS**
public static final String	**ATTR_COLOR**
public static final String	**MSG_EDIT**
Static String[]	**Choices**
Static string	**Oldchoices**

Static Color	**defaultColor**
Color	**Color**
Public String	**Nametxt**
Public String	**Spancontent**
Public Map	**pstart**
Public Map	**pend**
Public static final String	**GI_START**
Public static final String	**GI_END**
Public static final String	**ATTR_ADDED**

Fields inherited from class multivalent.Span
GI_END, GI_START, MSG_DELETE, MSG_EDIT, MSG_MORPH, MSG_UNATTACHED, pend, pstart
Fields inherited from class multivalent.Behavior
ATTR_BEHAVIOR, classname , name

Fields inherited from class multivalent.VObject
attr

Fields inherited from interface multivalent.ContextListener
LITTLE, LOT, PRIORITY_LENS, PRIORITY_MAX, PRIORITY_MIN, PRIORITY_SELECTION, PRIORITY_SPAN, PRIORITY_STRUCT, SOME

Constructor Summary

AddLableSpan()

Method Summary

`Color`	`getColor()` return the color of the label.
`Void`	`SetColor(Color color)` set the color of the label.
`viod`	`setInsertText(String inserttxt)` set the title for the label.
`boolean`	`appearance(Context cx, boolean all)` Resets the graphics context every time the object is added to or dropped from the set active over the portion of the document being drawn.
`boolean`	`paintBefore(Context cx, Node node)` A tree walk protocol, called before observed node has been painted.
`void`	`restore(ESISNode n, java.util.Map attr, Layer layer)` Given ESIS subtree, pluck class-specific information from attributes, call super.restore() for locations.
`ESISNode`	`save()` Stuff instance state into attributes; if save buffer not null, write out corresponding XML.
`boolean`	`semanticEventAfter(SemanticEvent se, java.lang.String msg)` Build up dialogue window and respond to changing color command
`boolean`	`semanticEventBefore(SemanticEvent se, java.lang.String msg)` Build up function buttons in span's popup menu.

Methods inherited from class multivalent.Span

`buildAfter, close, contains, contains, destroy, dumpPending, getEnd, getPriority, getStart, isSet, markDirty, move, move, move, moveq, moveq, moveq, moveqSwap, open, repaint, repaint, stretch, toString, validate`

Methods inherited from class multivalent.Behavior

`buildBefore, clipboardAfter, clipboardBefore, createUI, eventBefore, formatAfter, formatBefore, getBrowser, getDocument, getInstance, getInstance, getLayer, getName, getPreference, getRoot, isEditable, paintAfter, putPreference, redo, restoreChildren, setName, stats, undo`

Methods inherited from class multivalent.VObject

`ASSERT, attrEntrySetIterator, attrKeysIterator, clearAttributes, getAttr, getAttr, getAttributes, getGlobal, getValue, hasAttributes,`

`putAttr, removeAttr, setAttributes`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

ATTR_INSERT

```
public static final java.lang.String ATTR_INSERT
```

See Also:

Constant Field Values

Constructor Detail

AddLabelSpan

```
public AddLabelSpan()
```

Method Detail

appearance

```
public boolean appearance(Context cx,
                          boolean all)
```

Description copied from interface: `ContextListener`

Resets the graphics context every time the object is added to or dropped from the set active over the portion of the document being drawn. These behaviors can come from the style sheet, be ad hoc spans, be lenses, or come from elsewhere. Should be fast.

Specified by:

`appearance` in interface `ContextListener`

Overrides:

`appearance` in class `Span`

Parameters:

`all` - all attributes or exclude those that are not inherited

paintBefore

```
public boolean paintBefore(Context cx,
                           Node node)
```

Description copied from class: `Behavior`
A tree walk protocol, called before observed node has been painted. Called in same coordinate space as node's painting. Can be used draw special background, but usual background setting is done by spans or style sheets.
Overrides:
`paintBefore` in class `Behavior`
Returns:
`true` to short-circuit to paintAfter at that node, bypassing painting of the subtree.

---

semanticEventBefore

```
public boolean semanticEventBefore(SemanticEvent se,
                                   java.lang.String msg)
```

Navigate to referring links in same document, in span's popup menu.
Overrides:
`semanticEventBefore` in class `Span`

---

semanticEventAfter

```
public boolean semanticEventAfter(SemanticEvent se,
                                  java.lang.String msg)
```

Description copied from class: `Span`
Recognize "deleteSpan " .
Overrides:
`semanticEventAfter` in class `Span`

---

save

```
public ESISNode save()
```

Description copied from class: `Span`
Stuff instance state into attributes; if save buffer not null, write out corresponding XML. Subclass should override if have interesting content (can stuff content into attr then super.save()). If span is not attached to tree at save time, its old attachment points are retained. This way, spans that can't be attached presently can be tried again without degradation.
Overrides:
`save` in class `Span`

---

restore

```
public void restore(ESISNode n,
                    java.util.Map attr,
                    Layer layer)
```

Description copied from class: `Span`
Given ESIS subtree, pluck class-specific information from attributes, call super.restore() for locations. Attributes named `start` and `end` are reserved to hold Robust Location data.
Overrides:
`restore` in class `Span`

---

multivalent.std.span

Class MCHyperlinkSpan

```
java.lang.Object
  |
  +--multivalent.VObject
        |
        +--multivalent.Behavior
              |
              +--multivalent.Span
                    |
                    +--multivalent.std.span.MCHyperlinkSpan
```

All Implemented Interfaces:
ContextListener, EventListener, java.util.EventListener

public class **MCHyperlinkSpan**
extends Span
Version:
This is multiple-destination hyperlink.
Allow to defining one- to –many link relationship

$Revision: 1.8 $ $Date: 2004/02/02 13:16:26 $

See Also:
ActionSpan, ScriptSpan

Field Summary

static byte	**ACTIVE** Link state.
static java.lang.String	**ATTR_URI**
static java.lang.String	**ATTR_DEST**

static java.lang.String	**ATTR_LINK**
static byte	**HOVER** Link state.
static byte	**LINK** Link state.
static java.lang.String	**MSG_COPY_LINK** Copy link URI to clipboard.
protected byte	**state_** Current state of the link--normal, seen, cursor hovering above, mouse clicked on-- show we can show visually.
protected String	**dest_**
boolean	**delete_**
protected java.lang.Object	**target_** Target of link can be given as a String or URL.
static byte	**VISITED** Link state.
static List	**hyperlinks**

Fields inherited from class multivalent.Span
GI_END, GI_START, MSG_DELETE, MSG_EDIT, MSG_MORPH, MSG_UNATTACHED, pend, pstart

Fields inherited from class multivalent.Behavior
ATTR_BEHAVIOR, classname_, name_

Fields inherited from class multivalent.VObject
attr_

Fields inherited from interface multivalent.ContextListener
LITTLE, LOT, PRIORITY_LENS, PRIORITY_MAX, PRIORITY_MIN, PRIORITY_SELECTION, PRIORITY_SPAN, PRIORITY_STRUCT, SOME

Constructor Summary

MCHyperlinkSpan()

Method Summary

boolean	**appearance**(Context cx, boolean all) Spans are ContextListener's, which are behaviors that compose together to determine the Context display properties at every point in the document.
void	**event**(java.awt.AWTEvent e) Once we set the grab, subsequent events go directly to here, not to eventBefore/eventAfter.
boolean	**eventAfter**(java.awt.AWTEvent e, java.awt.Point rel, Node n) On a mouse button down, directly receive all furture low-level events by setting a grab in `Browser`, until a mouse button up.
byte	**getState**() Run-of-the-mill field getter.
java.lang.Object	**getTarget**() Run-of-the-mill field getter.
java.net.URI	**getURI**() Run-of-the-mill field getter.
void	**restore**(ESISNode n, java.util.Map attr, Layer layer) Restore almost always invokes its superclass, which when it chains up to `Behavior` sets the behavior's attributes and adds it to the passed layer.
ESISNode	**save**() Stuff instance state into attributes; if save buffer not null, write out corresponding XML.
boolean	**semanticEventAfter**(SemanticEvent se, java.lang.String msg) Catch Responds to the messages. Buildup dialogue window, add new hyperlink item, delete hyperlink, delete span.
boolean	**semanticEventBefore**(SemanticEvent se, java.lang.String msg) Add to the DOCPOPUP menu--the menu that pops up when the alternative mouse button is clicked over some part of the document (as opposed to the menubar) and the click is not short-

R

	circuited out by some behavior.
void	**setSeen**(boolean seen) Run-of-the-mill field setter.
protected void	**setState**(byte state) Run-of-the-mill field setter.
void	**setTarget**(java.lang.Object o) Sets target that isn't String or URI or URL.
void	**setURI**(java.lang.String txt) Computes full, canonical URL from a relative specification.
java.lang.String	**toString**() Subclasses should extend to check any attributes they add.

Methods inherited from class multivalent.Span
buildAfter, close, contains, contains, destroy, dumpPending, getEnd, getPriority, getStart, isSet, markDirty, move, move, move, moveq, moveq, moveq, moveqSwap, open, repaint, repaint, stretch, validate

Methods inherited from class multivalent.Behavior
buildBefore, clipboardAfter, clipboardBefore, createUI, eventBefore, formatAfter, formatBefore, getBrowser, getDocument, getInstance, getInstance, getLayer, getName, getPreference, getRoot, isEditable, paintAfter, paintBefore, putPreference, redo, restoreChildren, setName, stats, undo

Methods inherited from class multivalent.VObject
ASSERT, attrEntrySetIterator, attrKeysIterator, clearAttributes, getAttr, getAttr, getAttributes, getGlobal, getValue, hasAttributes, putAttr, removeAttr, setAttributes

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

ATTR_URI

```
public static final java.lang.String ATTR_URI
```

See Also:
Constant Field Values

LINK

```
public static final byte LINK
```

Link state.
See Also:
Constant Field Values

VISITED

```
public static final byte VISITED
```

Link state.
See Also:
Constant Field Values

HOVER

```
public static final byte HOVER
```

Link state.
See Also:
Constant Field Values

ACTIVE

```
public static final byte ACTIVE
```

Link state.
See Also:
Constant Field Values

MSG_COPY_LINK

```
public static final java.lang.String MSG_COPY_LINK
```

Copy link URI to clipboard.

`"copyLink".`

See Also:
Constant Field Values

---

target_

`protected java.lang.Object` **`target_`**

Target of link can be given as a String or URL.

---

state_

`protected byte` **`state_`**

Current state of the link--normal, seen, cursor hovering above, mouse clicked on--show we can show visually.

Constructor Detail

MCHyperlinkSpan

`public` **`MCHyperlinkSpan`**`()`

Method Detail

setSeen

`public void` **`setSeen`**`(boolean seen)`

Run-of-the-mill field setter.

---

setState

`protected void` **`setState`**`(byte state)`

Run-of-the-mill field setter. Checks `seen_` flag to see if link seen before.

---

getState

`public byte` **`getState`**`()`

Run-of-the-mill field getter.

---

getTarget

`public java.lang.Object` **`getTarget`**`()`

Run-of-the-mill field getter.

getURI

```
public java.net.URI getURI()
```

Run-of-the-mill field getter. Same as getTarget, but more convenient if target type known to be URL.

setTarget

```
public void setTarget(java.lang.Object o)
```

Sets target that isn't String or URI or URL.

setURI

```
public void setURI(java.lang.String txt)
```

Computes full, canonical URL from a relative specification. The canonical URL is used in the table of links already seen.

appearance

```
public boolean appearance(Context cx,
                          boolean all)
```

Spans are ContextListener's, which are behaviors that compose together to determine the Context display properties at every point in the document. For instance, a document will usually determine the font family, size, style, and foreground and background colors, among many other properties, of a piece of text with a combination of ContextListeners reporesenting the influence of style sheet settings, built-in span settings, and perhaps lenses. Here, the generic hyperlink hardcodes the action of coloring the text and gives it an underline, choosing either blue or red. The HTML media adaptor overrides this method to have no action, as the hyperlink appearance is dictated entirely by style sheets, either one linked to the particular web page or failing that the default HTML style sheet.

Specified by:

`appearance` in interface `ContextListener`

Overrides:

`appearance` in class `Span`

Parameters:

`all` - all attributes or exclude those that are not inherited

restore

```
public void restore(ESISNode n,
                    java.util.Map attr,
                    Layer layer)
```

Restore almost always invokes its superclass, which when it chains up to `Behavior` sets the behavior's attributes and adds it to the passed layer. Many behaviors also set default parameters and set fields from attributes in the attribute hash table that all behaviors have. See `Behavior`'s superclass, `VObject`, to examine the various attribute accessor methods.

Overrides:

`restore` in class `Span`

See Also:

`Location`

---

save

```
public ESISNode save()
```

Description copied from class: `Span`

Stuff instance state into attributes; if save buffer not null, write out corresponding XML. Subclass should override if have interesting content (can stuff content into attr then super.save()). If span is not attached to tree at save time, its old attachment points are retained. This way, spans that can't be attached presently can be tried again without degradation.

Overrides:

`save` in class `Span`

---

eventAfter

```
public boolean eventAfter(java.awt.AWTEvent e,
                          java.awt.Point rel,
                          Node n)
```

On a mouse button down, directly receive all future low-level events by setting a grab in `Browser`, until a mouse button up. Also, take the synthesized events (that is, generated by a multivalent.* class as opposed to a java.* class) corresponding to entering and exiting the span, at which time change the cursor and perhaps the colors to indicate that the link is active. As a Span, the hyperlink receives low-level events in the region of the document it spans without additional registering. This is event*After* rather than event*Before* because the rule of thumb is to build in before so it's available to other behaviors, and to take action in after if some other behavior hasn't short-circuited you.

Overrides:

`eventAfter` in class `Behavior`

---

event

```
public void event(java.awt.AWTEvent e)
```

Once we set the grab, subsequent events go directly to here, not to eventBefore/eventAfter.
Specified by:
`event` in interface `EventListener`
Overrides:
`event` in class `Span`

---

semanticEventBefore

```
public boolean semanticEventBefore(SemanticEvent se,
                                   java.lang.String msg)
```

Add to the DOCPOPUP menu--the menu that pops up when the alternative mouse button is clicked over some part of the document (as opposed to the menubar) and the click is not short-circuited out by some behavior. Similarly to ClipProvenance, add `Span.MSG_EDIT` if the span is in an editable layer (e.g., if link comes from the HTML sent by a random server, it is not editable, whereas link annotations you added are), "copyLink", "open in new window", and "open in shared window".
Overrides:
`semanticEventBefore` in class `Span`

---

semanticEventAfter

```
public boolean semanticEventAfter(SemanticEvent se,
                                  java.lang.String msg)
```

Catch "copyLink" sent in `semanticEventBefore`. The pair of `Document.MSG_OPEN` are handled by another behavior. Many subclasses have various parameters or attributes, such as URL here or annotation text elsewhere, and the Span class supports editing by catching `Span.MSG_EDIT` and throwing up an associated HTML document with a FORM in a note window. When that window is closed, it sends `SystemEvents.MSG_FORM_DATA`. Semantic event with the name-value pairs of the form as a parameter. Respond to the messages MSG_DELETE and MSG_DELTETITEMS to delete the span or a special hyperlink.
Overrides:
`semanticEventAfter` in class `Span`

---

toString

```
public java.lang.String toString()
```

Description copied from class: `Span`
Subclasses should extend to check any attributes they add.
Overrides:
`toString` in class `Span`

---

multivalent.std.span

Class TargetAnchorSpan

```
java.lang.Object
  |
  +--multivalent.VObject
        |
        +--multivalent.Behavior
              |
              +--multivalent.Span
                    |
                    +--multivalent.std.span.TargetAnchorSpan
```

All Implemented Interfaces:

ContextListener, EventListener, java.util.EventListener

public class **TargetAnchorSpan**
extends Span

Intra-document destination of a hyperlink: a named, robustly located point in document.

Version:

$Revision: 1.4 $ $Date: 2002/05/11 08:29:21 $

Field Summary

public static final String	**ATTR_CREATEDAT**
Static Color	**Color**
String	**createdat**

Fields inherited from class multivalent.Span

GI_END, GI_START, MSG_DELETE, MSG_EDIT, MSG_MORPH, MSG_UNATTACHED, pend, pstart

Fields inherited from class multivalent.Behavior
ATTR_BEHAVIOR, classname_, name_

Fields inherited from class multivalent.VObject
attr_

Fields inherited from interface multivalent.ContextListener
LITTLE, LOT, PRIORITY_LENS, PRIORITY_MAX, PRIORITY_MIN, PRIORITY_SELECTION, PRIORITY_SPAN, PRIORITY_STRUCT, SOME

Constructor Summary

TargetAnchorSpan()

Method Summary

boolean	**paintBefore**(Context cx, Node node) A tree walk protocol, called before observed node has been painted.
void	**restore**(ESISNode n, java.util.Map attr, Layer layer) Given ESIS subtree, pluck class-specific information from attributes, call super.restore() for locations.
boolean	**semanticEventAfter**(SemanticEvent se, java.lang.String msg) Recognize "deleteSpan " and "morphSpan ".
boolean	**semanticEventBefore**(SemanticEvent se, java.lang.String msg) Navigate to referring links in same document, in span's popup menu.
void	**setAnchorName**(java.lang.String name)

Methods inherited from class multivalent.Span
appearance, buildAfter, close, contains, contains, destroy, dumpPending, event, getEnd, getStart, isSet, markDirty, move, move, move, moveq, moveq, moveq, moveqSwap, open, repaint, repaint, save, stretch, toString, validate

Methods inherited from class multivalent.Behavior
buildBefore, clipboardAfter, clipboardBefore, createUI, eventBefore, formatAfter, formatBefore, getBrowser, getDocument, getInstance, getInstance, getLayer, getName, getPreference, getRoot, isEditable, paintAfter, paintBefore, putPreference, redo, restoreChildren, setName, stats, undo

Methods inherited from class multivalent.VObject
ASSERT, attrEntrySetIterator, attrKeysIterator, clearAttributes, getAttr, getAttr, getAttributes, getGlobal, getValue, hasAttributes, putAttr, removeAttr, setAttributes

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

TargetAnchorSpan

```
public TargetAnchorSpan()
```

restore

```
public void restore(ESISNode n,
                    java.util.Map attr,
                    Layer layer)
```

Description copied from class: Span

Given ESIS subtree, pluck class-specific information from attributes, call super.restore() for locations. Attributes named `start` and `end` are reserved to hold Robust Location data.
Overrides:
`restore` in class `Span`
See Also:
`Location`

---

semanticEventBefore

```
public boolean semanticEventBefore(SemanticEvent se,
                                   java.lang.String msg)
```

Navigate to referring links in same document, in span's popup menu.
Overrides:
`semanticEventBefore` in class `Span`

---

semanticEventAfter

```
public boolean semanticEventAfter(SemanticEvent se,
                                  java.lang.String msg)
```

Description copied from class: `Span`
Recognize "deleteSpan "
Overrides:
`semanticEventAfter` in class `Span`

---

paintBefore

```
public boolean paintBefore(Context cx,
                           Node node)
```

Description copied from class: `Behavior`
A tree walk protocol, called before observed node has been painted. Called in same coordinate space as node's painting. Can be used draw special background, but usual background setting is done by spans or style sheets.
Overrides:
`paintBefore` in class `Behavior`
Returns:
`true` to short-circuit to paintAfter at that node, bypassing painting of the subtree.

---

multivalent.std.span

Class InternalHyperlinkSpan

```
java.lang.Object
  |
  +--multivalent.VObject
        |
        +--multivalent.Behavior
              |
              +--multivalent.Span
                    |
                    +--multivalent.std.span.InternalHyperlinkSpan
```

All Implemented Interfaces:
ContextListener, EventListener, java.util.EventListener

public class **InternalHyperlinkSpan**
extends Span

This is the internal hyperlink. It allows the users to create a hyperlink that have destination in the document. Elaborately commented to serve as a simple example of translating Multivalent protocols into Java methods.

Version:
$Revision: 1.8 $ $Date: 2004/02/02 13:16:26 $

See Also:
`AnchorSpan`, `SetAsTargetSpan`

Field Summary

static byte	**ACTIVE** Link state.
static java.lang.String	**ATTR_TARGET**
static byte	**HOVER**

	Link state.
static byte	**LINK** Link state.
static java.lang.String	**MSG_COPY_LINK** Copy link URI to clipboard.
protected byte	**state_** Current state of the link--normal, seen, cursor hovering above, mouse clicked on-- show we can show visually.
static byte	**VISITED** Link state.

Fields inherited from class multivalent.Span
GI_END, GI_START, MSG_DELETE, MSG_EDIT, MSG_MORPH, MSG_UNATTACHED, pend, pstart

Fields inherited from class multivalent.Behavior
ATTR_BEHAVIOR, classname_, name_

Fields inherited from class multivalent.VObject
attr_

Fields inherited from interface multivalent.ContextListener
LITTLE, LOT, PRIORITY_LENS, PRIORITY_MAX, PRIORITY_MIN, PRIORITY_SELECTION, PRIORITY_SPAN, PRIORITY_STRUCT, SOME

Constructor Summary

InternalHyperlinkSpan()

Method Summary

boolean	**appearance**(Context cx, boolean all)

	Spans are ContextListener's, which are behaviors that compose together to determine the Context display properties at every point in the document.
`void`	**`event`**`(java.awt.AWTEvent e)` Once we set the grab, subsequent events go directly to here, not to eventBefore/eventAfter.
`boolean`	**`eventAfter`**`(java.awt.AWTEvent e, java.awt.Point rel, Node n)` On a mouse button down, directly receive all furture low-level events by setting a grab in `Browser`, until a mouse button up.
`byte`	**`getState`**`()` Run-of-the-mill field getter.
`void`	**`go`**`()` jump to the location of the target anchor when internal hyperlink is clicked.
`void`	**`restore`**`(ESISNode n, java.util.Map attr, Layer layer)` Restore almost always invokes its superclass, which when it chains up to `Behavior` sets the behavior's attributes and adds it to the passed layer.
`ESISNode`	**`save`**`()` Stuff instance state into attributes; if save buffer not null, write out corresponding XML.
`boolean`	**`semanticEventAfter`**`(SemanticEvent se, java.lang.String msg)` Catch "copyLink" sent in `semanticEventBefore`.
`boolean`	**`semanticEventBefore`**`(SemanticEvent se, java.lang.String msg)` Add to the DOCPOPUP menu--the menu that pops up when the alternative mouse button is clicked over some part of the document (as opposed to the menubar) and the click is not short-circuited out by some behavior.
`void`	**`setSeen`**`(boolean seen)` Run-of-the-mill field setter.
`protected void`	**`setState`**`(byte state)` Run-of-the-mill field setter.

Methods inherited from class multivalent.Span
`buildAfter, close, contains, contains, destroy, dumpPending, getEnd, getPriority, getStart, isSet, markDirty, move, move, move, moveq, moveq, moveq, moveqSwap, open, repaint, repaint, stretch, validate`

Methods inherited from class multivalent.Behavior
buildBefore, clipboardAfter, clipboardBefore, createUI, eventBefore, formatAfter, formatBefore, getBrowser, getDocument, getInstance, getInstance, getLayer, getName, getPreference, getRoot, isEditable, paintAfter, paintBefore, putPreference, redo, restoreChildren, setName, stats, undo

Methods inherited from class multivalent.VObject
ASSERT, attrEntrySetIterator, attrKeysIterator, clearAttributes, getAttr, getAttr, getAttributes, getGlobal, getValue, hasAttributes, putAttr, removeAttr, setAttributes

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

ATTR_URI

`public static final java.lang.String` **`ATTR_URI`**

See Also:

Constant Field Values

LINK

`public static final byte` **`LINK`**

Link state.

See Also:

Constant Field Values

VISITED

`public static final byte` **`VISITED`**

Link state.

See Also:

Constant Field Values

HOVER

```
public static final byte HOVER
```

Link `state`.

See Also:
Constant Field Values

ACTIVE

```
public static final byte ACTIVE
```

Link `state`.

See Also:
Constant Field Values

MSG_COPY_LINK

```
public static final java.lang.String MSG_COPY_LINK
```

Copy link URI to clipboard.

`"copyLink"`.

See Also:
Constant Field Values

target_

```
protected java.lang.Object target_
```

Target of link can be given as a String or URL.

state_

```
protected byte state_
```

Current state of the link--normal, seen, cursor hovering above, mouse clicked on--show we can show visually.

Constructor Detail

InternalHyperlinkSpan

```
public HyperlinkSpan()
```

Method Detail

setSeen

```
public void setSeen(boolean seen)
```

Run-of-the-mill field setter.

---

setState

```
protected void setState(byte state)
```

Run-of-the-mill field setter. Checks `seen_` flag to see if link seen before.

---

getState

```
public byte getState()
```

Run-of-the-mill field getter.

---

getTarget

```
public java.lang.Object getTarget()
```

Run-of-the-mill field getter.

---

getURI

```
public java.net.URI getURI()
```

Run-of-the-mill field getter. Same as getTarget, but more convenient if target type known to be URL.

---

setTarget

```
public void setTarget(java.lang.Object o)
```

Sets target that isn't String or URI or URL.

---

setURI

```
public void setURI(java.lang.String txt)
```

Computes full, canonical URL from a relative specification. The canonical URL is used in the table of links already seen.

---

appearance

```
public boolean appearance(Context cx,
                          boolean all)
```

Spans are ContextListener's, which are behaviors that compose together to determine the Context display properties at every point in the document. For instance, a document will usually determine the font family, size, style, and foreground and background colors, among many other properties, of a piece of text with a combination of ContextListeners representing the influence of style sheet settings, built-in span settings, and perhaps lenses. Here, the generic hyperlink hardcodes the action of coloring the text and gives it an underline, choosing either blue or red. The HTML media adaptor overrides this method to have no action, as the hyperlink appearance is dictated entirely by style sheets, either one linked to the particular web page or failing that the default HTML style sheet.

Specified by:

`appearance` in interface `ContextListener`

Overrides:

`appearance` in class `Span`

Parameters:

`all` - all attributes or exclude those that are not inherited

---

restore

```
public void restore(ESISNode n,
                    java.util.Map attr,
                    Layer layer)
```

Restore almost always invokes its superclass, which when it chains up to `Behavior` sets the behavior's attributes and adds it to the passed layer. Many behaviors also set default parameters and set fields from attributes in the attribute hash table that all behaviors have. See `Behavior`'s superclass, `VObject`, to examine the various attribute accessor methods.

Overrides:

`restore` in class `Span`

See Also:

`Location`

---

save

```
public ESISNode save()
```

Description copied from class: `Span`

Stuff instance state into attributes; if save buffer not null, write out corresponding XML. Subclass should override if have interesting content (can stuff content into attr then super.save()). If span is not attached to tree at save time, its old

attachment points are retained. This way, spans that can't be attached presently can be tried again without degradation.
Overrides:
`save` in class `Span`

---

eventAfter

```
public boolean eventAfter(java.awt.AWTEvent e,
                          java.awt.Point rel,
                          Node n)
```

On a mouse button down, directly receive all furture low-level events by setting a grab in `Browser`, until a mouse button up. Also, take the synthesized events (that is, generated by a multivalent.* class as opposed to a java.* class) corresponding to entering and exiting the span, at which time change the cursor and perhaps the colors to indicate that the link is active. As a Span, the hyperlink receives low-level events in the region of the document it spans without additional registering. This is event*After* rather than event*Before* because the rule of thumb is to build in before so it's available to other behaviors, and to take action in after if some other behavior hasn't short-circuited you.
Overrides:
`eventAfter` in class `Behavior`

---

event

```
public void event(java.awt.AWTEvent e)
```

Once we set the grab, subsequent events go directly to here, not to eventBefore/eventAfter.
Specified by:
`event` in interface `EventListener`
Overrides:
`event` in class `Span`

---

go

```
public void go()
```

Jump to the location of the target anchor corresponding to this internal hyperlink when the link is clicked. It finds the target span by the code of the span and move to the span

---

semanticEventBefore

```
public boolean semanticEventBefore(SemanticEvent se,
                                   java.lang.String msg)
```

Add to the DOCPOPUP menu--the menu that pops up when the alternative mouse button is clicked over some part of the document (as opposed to the menubar) and the click is not short-circuited out by some behavior. Similarly, to ClipProvenance, add `Span.MSG_DELETE` if the span is needed to delete.

Overrides:

`semanticEventBefore` in class `Span`

---

semanticEventAfter

```
public boolean semanticEventAfter(SemanticEvent se,
                                  java.lang.String msg)
```

Catch "copyLink" sent in `semanticEventBefore`.

Overrides:

`semanticEventAfter` in class `Span`

---

toString

```
public java.lang.String toString()
```

Description copied from class: `Span`

Subclasses should extend to check any attributes they add.

Overrides:

`toString` in class `Span`
