ABSTRACT

JUST-IN-TIME HYPERMEDIA

by
Li Zhang

Many analytical applications, especially legacy systems, create documents and display screens in response to user queries "dynamically" or in "real time". These documents and displays do not exist in advance, and thus hypermedia must be generated "just in time" — automatically and dynamically.

This dissertation details the idea of "just-in-time" hypermedia and discusses challenges encountered in this research area. A fully detailed literature review about the research issues and related research work is given. A framework for the "just-in-time" hypermedia compares virtual documents with static documents, as well as dynamic with static hypermedia functionality. Conceptual "just-in-time" hypermedia architecture is proposed in terms of requirements and logical components. The "just-in-time" hypermedia engine is described in terms of architecture, functional components, information flow, and implementation details. Then test results are described and evaluated. Lastly, contributions, limitations, and future work are discussed.

JUST-IN-TIME HYPERMEDIA

by
Li Zhang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2005

APPROVAL PAGE

JUST-IN-TIME HYPERMEDIA

Li Zhang

Dr. Michael Bieber, Dissertation Advisor — Date
Associate Professor of Information Systems, New Jersey Institute of Technology

Dr. Vincent Oria, Dissertation Co-advisor — Date
Assistant Professor of Computer Science, New Jersey Institute of Technology

Dr. James Geller, Committee Member — Date
Professor of Computer Science, New Jersey Institute of Technology

Dr. Dimitri Theodoratos, Committee Member — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

Dr. Fabio Vitali, Committee Member — Date
Associate Professor of Computer Science, University of Bologna

Dr. Dave Millard, Committee Member — Date
Senior Research Fellow of Electronics and Computer Science
University of Southampton

BIOGRAPHICAL SKETCH

Author: Li Zhang

Degree: Doctor of Philosophy

Date: May 2005

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, NJ, USA, 2005

- Master of Science in Electrical Engineering
  Southeast University, Nanjing, China, 1997

- Bachelor of Science in Biomedical Engineering
  Xi'an Jiaotong University, Xi'an, China, 1992

Major: Computer Science

Publications:

Zhang, L., M. Bieber, D. Millard, and V. Oria. "Supporting Virtual Documents in Just-in-Time Hypermedia Systems", *Proceedings of the 2004 ACM, Symposium on Document Engineering*, pages 35-44, Milwaukee, Wisconsin, October 2004.

Catanio, J., N. Nnadi, L. Zhang, M. Bieber, and R. Galnares. "Ubiquitous Metainformation and the WYWWYWI Principle", *Journal of Digital Information*, Vol. 5(1), 2004.

Zhang, L., and M. Bieber. "Hypermedia in Dynamic Environments", *The 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 271-275, Orlando, Florida, July 2003.

Zhang, L., and M. Bieber. "Challenges for Just-In-Time Hypermedia", *Hypertext 2002 Advance Program: Demos & Posters*, College Park, Maryland, June 2002.

This dissertation is dedicated to

my family for their love and support

ACKNOWLEDGMENT

The author would like to express her sincere gratitude to her dissertation advisor, Dr. Michael Bieber, for his remarkable guidance, constant support and encouragement throughout this research. The author's deepest appreciation also goes to her co-advisor, Dr. Vincent Oria, for his help.

The author would like to thank Dr. James Geller, Dr. Dimitri Theodoratos, Dr. Fabio Vitali, and Dr. Dave Millard for their valuable suggestions to this dissertation and for serving as members of the committee. The author also would like to thank for Dr. Roberto Galnares, Mr. Anirban Bhaumik, and Ms. Nkechi Nnadi for their help on software design and implementation.

Finally, the author would like to express her deepest appreciation to her family, especially her parents and her husband, for their love, faith, patience and many sacrifices.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF TABLES

CHAPTER 1

JUST-IN-TIME HYPERMEDIA OVERVIEW

"Just-in-time" (JIT) hypermedia research introduces new concepts and encounters problems that are different from traditional hypermedia research. Section 1.1 motivates this research. Section 1.2 briefly discusses basic concepts and challenges met in this research area. (These issues will be discussed in Chapter 2 in detail.) Section 1.3 gives a brief description of the proposed "just-in-time" hypermedia engine. Section 1.4 presents the dissertation outline.

1.1 Motivation

Many analytical applications, especially legacy systems, create documents and display screens in response to user queries "dynamically" or in "real time". These documents and displays do not exist in advance, and thus hypermedia must be generated "just in time"—automatically and dynamically.

Suppose an analyst wants to determine projected profits for different sales levels in her company. She performs the analysis within a sales support application, and makes comments on each resulting profit calculation. A few days later when preparing her final report, she knows that she will need to include the calculation results and comments. She wishes to return to them without having to remember the input parameter values for each and then manually re-performing each calculation. Therefore, she creates a bookmark to the display screen containing each calculation result before discarding it. Invoking each bookmark later causes the sales support application to re-execute its calculations

automatically, and the hypermedia interface to relocate her comments in the application's newly re-generated display.

1.2 Challenges of "Just-in-time" Hypermedia

This section introduces basic concepts of "just-in-time" hypermedia and briefly describes challenges in this research area.

1.2.1 Basic Concepts

Documents and Elements: A hypermedia system usually enables users to add links, semantic information to documents (static or dynamic) and elements. An element in a document is an arbitrarily selected portion of the document.

Hypermedia Functionality: hypermedia systems include many features to help users to navigate or understand information well. These features are called hypermedia functionality.

Anchor: an area within the content of a document, which is the destination or source of a link.

Virtual Documents and Virtual Elements: A virtual document is a document for which no persistent state exists and for which some or all of each instance is generated at run time (Watters 1999). Elements in a virtual document are called virtual elements.

Dynamic Hypermedia Functionality: The hypermedia functionality associated with dynamically - generated documents and elements is called dynamic hypermedia functionality. Anchors associated with dynamic hypermedia functionality must be determined in virtual documents when the documents are generated. Virtual documents

that are the subjects of dynamic hypermedia functionalities do not exist except when the document is being displayed on the computer screen.

Dynamic Regeneration: A new invocation of the dynamically - generated virtual document is called dynamic regeneration.

Relocation and Re-identification: To identify a formally - identified element in a regenerated virtual document is called re-identification. To locate a formally - created anchor in a regenerated virtual document is called relocation.

1.2.2 Challenges

A JIT hypermedia system is different from a traditional hypermedia system in that it adds hypermedia functionality to dynamically - generated virtual documents, while traditional hypermedia systems usually deal with static documents. The major challenges encountered in JIT hypermedia research are dynamic regeneration, re-location and re-identification. An overview of these challenges is given in Section 1.2.2.1 and 1.2.2.2. Details will be discussed in Chapter 2.

1.2.2.1 Dynamic Regeneration. In analytical or computational applications, documents do not exist in advance, they are generated when users do queries or execute some commands. When the windows close, these documents no longer exist. If a user adds some comments to a document, next time when he tries to visit the comments, the virtual document that was annotated should be re-generated. Some Web systems regenerate analysis screens by storing all relevant parameters in the virtual document's URL (Uniform Resource Locator), but many Web-based and non-Web based legacy systems have no such mechanism. Other Web systems do not allow URLs with detailed parameters for security or other reasons. When the user bookmarks and returns to its

URL, the system returns the user to the home page or the initial input page. The JIT hypermedia system must provide automatic regeneration without asking the user to reenter parameters. Because a virtual document does not have a file name and a specific location to store the file content, dynamic regeneration requires a unique and persistent identifier to reference each virtual document. Dynamic regeneration also requires a criterion and procedure to decide whether the re-generated document is same as the previously marked document. Chapter 3 will discuss in detail the requirements for dynamic regeneration, the procedure to do regeneration and the information needed for regeneration.

1.2.2.2 Re-location and Re-identification. When a virtual document is regenerated, the virtual document's structure and content could be changed. It could also be possible that virtual elements in the document have changed. If a user has put some anchors on the virtual document, the position and content of the anchors could change. It is necessary to find the location of the predefined anchors in the virtual document, and decide whether the content is same as the previously marked one. This is called re-location and re-identification. Moreover, when analytical applications generate new query results, the JIT hypermedia system should relocate and re-identify the anchors for known elements in the old query results. Re-location requires flexible and efficient mechanisms to record the location and content of anchors. Re-identification requires a criterion to revalidate the virtual elements and some information about the virtual elements are needed. Chapter 3 discusses all the above aspects of re-location and re-identification in detail.

1.3 Solution: the Just-in-time Hypermedia Engine

The Just-in-time Hypermedia Engine (JHE) executes as a middleware between an application and its user interface, providing additional hypermedia navigational, structural and annotation functionality, with minimal modification to the application.

JHE should have the following functions to supplement hypermedia functionalities for the "just-in-time" environment:

(1) JHE assigns unique, persistent identifiers for documents, elements and anchors in the documents;

(2) JHE stores links, comments, etc., in an external linkbase;

(3) JHE does the regeneration by intercepting the messages between the user interface and the applications.

(4) JHE relocates and re-identifies the anchors in the documents efficiently.

"Wrappers" need to be written for each application in order to integrate them with the engine architecture.

1.4 Dissertation Outline

Chapter 2 gives a detailed literature review of hypermedia research areas. Chapter 3 discusses the "just-in-time" hypermedia framework. Chapter 4 describes related research areas including: virtual documents; WWW standards; unique identifiers; object identification and addressing mechanisms; and document structure recognition. Chapter 5 proposes the Just-in-time Hypermedia Engine (JHE), in terms of architecture, data formats, and flowcharts. Chapter 6 describes implementation details. Chapter 7 gives test results and evaluations. Chapter 8 discusses contributions of this research. Chapter 9 discusses the main issues and problems of this research, and future research work.

CHAPTER 2

HYPERMEDIA

Hypermedia is a concept that encourages authors to structure information as an associative network of nodes and interrelating links (Bieber 1997). This chapter describes basic concepts of hypermedia, hypermedia development history, open hypermedia systems and the World Wide Web (WWW), and hypermedia functionality. Section 2.1 gives an overview of hypermedia. Section 2.1.1 describes the basic concepts of hypermedia, including hypermedia, links and nodes. Section 2.1.2 describes the hypermedia development history from 1945 when Bush introduced the idea of hypermedia to the state of the art development progress. It describes the earlier hypermedia systems in 1960s and 1970s, which are mainframe-based text only systems. During 1980s, many hypermedia systems were built on workstations and hypermedia research groups started to work on interoperability of different hypermedia systems, which leads to the emergence of open hypermedia systems. Section 2.1.3 discusses benefits and drawbacks of hypermedia systems. Section 2.2 gives an overview of open hypermedia systems and the WWW. Section 2.2.1 describes concepts of open hypermedia systems, its progress and the Open Hypermedia System Working Group (OHSWG). OHS research has made significant progress since 1980s. Since early 1990s, the World Wide Web as the first publicly accepted hypermedia system changed people's lives greatly and activated much research work in WWW related fields. Because of the drawbacks of traditional WWW systems, many hypermedia systems were built based on the Web trying to enhance the traditional WWW systems' functionality.

As mentioned above, nodes and links are basic components of hypermedia systems, hypermedia systems should have more functions/features to enable users to view and manage documents. These features are called hypermedia functionality. Section 2.3 discusses hypermedia functionality. Section 2.3.1 gives overview on hypermedia functionality. Section 2.3.3 introduces basic concepts of hypermedia functionality, such as node, anchor, annotation and selection. Among them, links are the most important feature. Links represent relationships between nodes. Links can be static or dynamic; automatic or manual; embedded or external; unidirectional or multidirectional; and semantically typed. Section 2.3.3 discusses different aspects of links, pros and cons for different attributes. It also describes different link service systems and the link integrity problem.

A just-in-time hypermedia system is a hypermedia system that can supplement dynamically-generated virtual documents with hypermedia functionalities. A JIT system provides a Web interface for applications and it is built based on the WWW. Technologies in open hypermedia systems and the WWW can be helpful to a JIT system.

2.1 Overview

This section discusses the basic concepts of hypermedia, and its benefits and drawbacks. Then it presents a brief overview of hypermedia system history.

2.1.1 Basic Concepts of Hypermedia

Hypermedia is a concept that encourages authors to structure information as an associative network of nodes and interrelating links (Bieber 1997) Hypermedia researchers view the terms *hypertext* and *hypermedia* as synonymous and use them

interchangeably. Hypermedia nominally applies hypertext concepts to multiple media. A hypertext system is made of nodes (concepts) and links (relationships). A node usually represents a single concept or idea. It can contain text, graphics, animation, audio, video, images or programs. In most hypermedia systems, a node usually is a document. It can be semantically typed (such as detail, proposition, collection, summary, observation, issue) thereby carrying semantic information (Rao 1990). Nodes are connected to other nodes by links. Links represent relationships between two nodes.

2.1.2 Development History

The concept of hypertext first appeared in Vannevar Bush's article "As we may think" in 1945. Bush envisioned a "memex" machine, which, among many other things, could assist the scientist in developing associative indexes or "trails" through a collection of documents. The development of hypertext systems has evolved through many stages.

The earliest hypertext systems aimed to provide speedy access to information by cross- referencing and associative indexing. Most systems were mainframe-based text only systems, offering information frame by frame, such as NLS/Augment (Engelbart 1968), FRESS (van Dam 1968), and ZOG (McCracken 1984). NLS/Augment was the first hypertext system developed in 1960s by Douglas Engelbart's group. In NLS/Augment, nodes were organized into files. Reference links helped a user to move from one file to another. FRESS was developed by van Dam in 1968, which was a file retrieving and editing system running on an IBM/360 mainframe. ZOG was a large database consisted of frames, which was developed at Carnegie-Mellon University in 1975. A frame in ZOG consisted of a title, a description, a line with standard ZOG commands, and a set of menu items leading to other frames. These systems had support

for multiple users sharing the hypermedia information network. Nodes were un-typed, without supporting composites.

In the latter part of the 1980s, workstation and operating system technology had advanced to the stage that it was possible to start conducting research into the ideas originally conceived by Bush, Engelbart and McCracken. Research-oriented systems such as Notecards (Halasz 1988), KMS (Akscyn 1988), Neptune (Desisle 1986), gIBIS (Conklin 1988) and Intermedia (Yankelovich 1988 and Haan 1992) began to appear. Notecards (Halasz 1987) is a hypermedia system from the 1980s, aimed at easing the task of idea processing. Idea processing is divided up in three phases: asquisition, analysis and exposition. Notecards has four basic objects: notecards, links, browsers, and fileboxs. Navigating structure in Notecards can be done in either of two ways. It is possible to follow links from card to card, at each card deciding which link to follow. It is also possible to create an overview browser for some subnetwork, and then traverse the links from the browser to the referenced cards. In (Halasz 1988), experience with Notecards lead to suggestion of seven issues that later became a milestone in hypermedia. KMS (Knowledge Management System) was developed based on ZOG. Though KMS included a GUI (Graphic User Interface), it still remained a text-based system. It was intended to be a collaborative tool, in that users could modify the contents of a frame and the changes would be immediately visible to others through dynamically updated links. gIBIS is an Issue Based Information System that is designed to facilitate the capture of early design deliberations. gIBIS offers argumentation support through issue, position and argument objects. The system makes use of an overview picture of the complete structure, with an indicator showing where the user is. These systems are similar in concept to the earlier

mainframe-based systems. In addition, they supported graphics nodes and they had more advanced user interfaces.

Beyond these well-known systems there are many other research oriented prototype systems, some of which are especially well documented such as aTrellis (Stotts 1989) and ExperText (Rada 1991). Also there are some hypermedia reference models. The Dexter Model (Halasz 1994) is a well-documented hypermedia reference model. Prototype systems like DeVise (GrØnbak 1997) and DHM are based in Dexter model. Hypertext Abstract Machine (HAM) (Campbell 1987) is another high level hypermedia model.

In the late 1980s, the hypermedia research community developed, amongst others, two separate research threads: one focusing on Open Hypermedia Systems (OHS) and one on Adaptive Hypermedia (AH). A lot of work has been done in these areas since then, which will be discussed in detail later. In the early 1990s, the World Wide Web (WWW) appeared, as the first open and distributed hypermedia system. It gained its popularity due to its simplicity and generality. Much effort has been put into the WWW research areas and significant progress has been made since then. It will be discussed in Section 2.2.2.

2.1.3 Benefits and Drawbacks of Hypermedia

Hypermedia systems enable users to add links and comments to documents, in a way similar to human understanding. For example, a reader usually highlights or underlines some words, sentences or paragraphs in a book to help him/her to remember or understand the contents. A hypermedia system can help users to navigate documents by using guided tours (an ordered set of nodes or documents) or overviews (a overview map

of nodes). Hypermedia systems supply tools such as search to help users understand and organize the documents in information systems.

In summary, the benefits of adding hypermedia functionality to information system applications are that hypermedia provides contextual, navigational access for viewing information and that it represents knowledge in a form relatively close to the cognitive organizational structures that people use.

Conklin (1987) identifies two major dangers of free-formed hypermedia access within an associative network: *disorientation* and *cognitive overhead. Disorientation* occurs when readers lose track of their position within the hypertext web while traversing links. *Cognitive overhead* occurs when readers feel overwhelmed with too many choices of where to navigate next. While many hypermedia researchers believe that disorientation and cognitive overhead are problems inherent in hypermedia, proper implementations of advanced hypermedia features (such as guided tour or overviews) will alleviate these problems, as well as provide readers with a rich information environment. (Bieber 1997)

2.2 Open Hypermedia Systems and the WWW

Hypermedia research has many fields that address different aspects of hypermedia. It includes Open Hypermedia Systems (OHS), Adaptive Hypermedia (AH), spatial hypermedia, object-oriented hypermedia, World Wide Web (WWW), etc. A "just-in-time" (JIT) hypermedia system is a hypermedia system that can add hypermedia functionalities to dynamically-generated virtual documents. Also it can add a Web interface to analytical applications. OHS and WWW systems are discussed in this

chapter. Adaptive hypermedia systems generate dynamic documents based on user models. AH will be discussed in Chapter 4.

2.2.1 Open Hypermedia Systems

2.2.1.1 Overview. The first hypermedia systems were, from an architectural point of view, monolithic systems, effected as a single process, combining the user interface, linking mechanism and information stored in one program. This made the (mainly navigational) hypermedia facilities inflexible, since it forced the users to use the embedded user interfaces.

In the late 1980's, OHS research became an important area in hypermedia research. OHS research addresses the issue of integrating hypermedia functionality into existing applications in the computing environment. An open hypermedia system is typically a middleware component in the computing environment offering hypermedia functionality to applications independent of their storage and display functionality. Using the services of an OHS, existing applications in the computing environment can become "hypermedia enabled", thus supporting linking to and from information managed by the application without altering the information itself. To become "hypermedia enabled", applications must be able to communicate hypermedia requests to the OHS (OHSWG 1997).

2.2.1.2 Early Generation OHS Systems. Early generation OHS systems include Microcosm (Davis 1992), SP3 (Schnase 1994), Chimera (Anderson 1994), etc. Microcosm uses a "filter" model to support open hypermedia functionality. Anchor activation and link traversal are implemented as messages passing through a chain of filter processes, each of which can receive messages and take appropriate responses (such

as forwarding to the next filter in chain). Microcosm supports "generic links" anchored to any occurrence of a string in all documents, and "computed links", which uses text retrieval techniques to compute the link destination dynamically, thus supporting true content-based hyperlinking.

SP3 uses a process-based model. This system has a very general model in which anchors and links are themselves processes, and link operations correspond to communications between these processes and participating applications. Applications keep track of the locations of persistent selections in their documents.

2.2.1.3 OHSWG. Since different OHS systems used different architecture and data models, they lack presentation interoperability, storage interoperability and system interoperability. It was crucial to have common guideline and standards for interoperation between OHSs and applications that request hypermedia services. The Open Hypermedia Systems Working Group (OHSWG) was formed in 1996 to address three key issues: defining open hypermedia systems (the scenario subgroup), defining open link services (the protocol subgroup) and defining an OHS reference architecture (the reference architecture subgroup).

Since 1996, OHS systems and standards have made great progress in the aspects of the Open Hypermedia Protocol (Davis 1997), OHS system architecture and OHS data model (Grøbræk 1997). In 1998, a standard data model (which later became FOHM (Millard 2000)) for open hypermedia structures was proposed. It includes the OHSWG navigational data model, spatial data model and taxonomy data model.

In the late 1990s, OHS and WWW research became intertwined fields; most OHS systems built their systems to supply rich hypermedia functionality for the WWW, such as WebVise (GrØnbæk 1999).

Open hypermedia systems is now a relatively mature area making significant progress in the area of standardization and the support of interoperability between open systems, addressing some earlier concerns regarding the definition of intermediate formats. Difficulties remain in providing common interfaces across the wide range of applications and participating open hypermedia systems (Cunliffs 2000). Recent OHS work includes Web technology (using XLink as syntax to support Web services), hypermedia concepts (such as asynchronous linking) and open hypermedia infrastructure (security, authentication, etc) (OHS 2002).

A JIT system is a middleware between applications and the viewer. The ideas and methods of OHS systems, such as data models and link services, could be helpful in developing JIT systems.

2.2.2 World Wide Web

2.2.2.1 WWW History and Drawbacks. Tim Berners-Lee proposed the WWW research project (originally as an information system for the particle physics community) at CERN in Switzerland in 1989. The first Web server and browser were established on December 1990. The W3C consortium (W3C) was established in 1994 to improve WWW development and conclude WWW standards. Marc Andersen developed Mosaic for NCSA in 1994, which is the first Web browser to capture the public's imagination. In 1995, Digital Corp. made AltaVista (a search engine) available to the public. The World Wide Web (WWW) (Lee 1992) is the largest distributed hypermedia

system in use. It makes use of data formats (for example, HTML) and access protocols (for example, HTTP), which are open, extensible, and standard. WWW is a distributed hypermedia system: combining the concept of hyperlinks (associative browsing by following links to related information) with multimedia (text, image, audio, video, etc.). Through its URL (Universal Resource Locator) mechanism, WWW can represent links to any document on any Web, Gopher, or FTP server worldwide. The CGI (Common Gateway Interface) script interface allows WWW servers to start arbitrary application programs, for example linking into external databases or implementing complex search algorithms.

Despite its popularity and support for global distribution, the traditional WWW system has several weaknesses when examined from within the rich context of hypermedia system. (Andrews 1995)

(1) HTML uses embedded and one-way links. Embedded links have many disadvantages than external links, which is discussed in detail in Chapter 3.

(2) Standard hypermedia features such as guided tours and overviews are not provided automatically. Annotation was once a feature in Mosaic but taken away later.

(3) WWW has no native search facilities, but relies on external search engines.

(4) The flexibility provided by CGI is achieved at great cost: the uniformity of the interface disappears; different WWW servers behave differently – resulting in the "Balkanisation" (according to Ted Nelson) of the Web into independent "W3 Empires".

(5) The Web today is very much "read-only", in the sense that information providers prepare data sets in which information consumers can generally only browse.

(6) Although its URL mechanism endows WWW in terms of number of servers, it is not scalable in terms of number of users. When there are too many clients connecting to one server, traffic jam occurs on the Internet.

The World Wide Web gets its popularity due to its simplicity even though this simplicity brings many deficiencies. Since the Web does not have rich hypermedia functionality, some people even argue whether it is a real hypermedia system. A lot of efforts have been made to enrich hypermedia functionality for the WWW since its emergence.

2.2.2.2 Web Based Hypermedia Systems. To overcome the drawbacks brought up by the earlier WWW systems, other Web based hypermedia systems have been developed such as Hyper-G (Andrews 1994) (later HyperWave) and HyperDisco (Wiil 1996).

Hyper-G is a multi-user, multi-protocol, structured, hypermedia information system, which runs as a client-server application on the Internet. (Andrew 1994). To help alleviate the disorientation associated with becoming "lost in hyperspace", Hyper-G provides three closely coupled, orthogonal navigational mechanisms: structuring, hyperlinks, and search. The tight coupling allows clients to correlate search results, link maps, and structure overviews, providing a powerful navigation. Hyper-G groups documents into aggregate collections, which may themselves belong to other collections. Navigation may be performed down through the collection hierarchy, access rights assigned on a collection-by-collection basis, and the scope of searches restricted to a particular sets of collections. Links in Hyper-G connect a source anchor within one document to either a destination anchor within another document, an entire document, or a collection. Documents and collections have an associated set of attributes (author, title, keywords, etc.), which may be searched for, including Boolean combinations and term truncation. Full text (content) search facilities include vector and fuzzy Boolean queries.

HyperDisco (Wiil 1996) provides an extensible object-oriented hypermedia integration model supporting inter-tool linking, computation, concurrency control, notification control, version control, access control, search and query, and various other features. HyperDisco provides two distinct layers of hypermedia functionality: integration model layer and data model layer. HyperDisco provides an extensible object-oriented data model with a set of general hypermedia object types: anchor, node, link and composites. All instances in HyperDisco have a unique object identifier (OID). The link class supports a very general multi-headed (n-n) links. Links have a direction, but can be traversed in both directions. Links maintain two lists of endpoints (to and from). A link endpoint can be one of three types: static, dynamic (computed) or dangling. HyperDisco supports both content and structure based queries as a supplement to navigational information retrieval (browsing and link traversal). Boolean operators such as NOT, AND and OR and control structures such as IF, COND, and CASE are used to glue basic search operations together.

A JIT system is a Web-based hypermedia system, trying to provide hypermedia functionality for dynamically generated virtual documents. Hyper-G and HyperDisco provide many hypermedia functionalities for Web systems, technologies or methods from them or other Web-based open hypermedia systems could be helpful to JIT system design.

2.2.2.3 W3C and the Semantic Web. In October 1994, Tim Berners-Lee founded the World Wide Web Consortium (W3C) at the Massachusetts Institute of Technology, Laboratory for Computer Science, in collaboration with CERN, where the Web originated, with support from DARPA and the European Commission. Goals of W3C for

the Web include promoting universal access, semantic Web and Web of Trust. Universal access is to make the Web accessible to all by promoting technologies that take into account the vast differences in culture, languages, education, ability, material resources, access devices, and physical limitations of users on all continents. The semantic Web is the representation of the data with well-defined meaning on the World Wide Web, enabling computers and people to work in cooperation. Web of Trust is to guide the Web's development with careful consideration for the novel legal, commercial, and social issues raised by this technology. Since 1994, the W3C has made great progress in promoting and developing the vision of the WWW future, in designing Web technologies and standardizing Web technologies.

Tim Berners-Lee proposed the "Semantic Web" at the 1999 WWW8 meeting. The semantic Web is a Web that includes documents or portions of documents, describing explicit relationships between things and containing semantic information intended for automated processing by computers. The W3C has proposed a series of drafts about metadata standards on Web, such as XML (eXtensible Markup Language), RDF (Resource Description Framework), SVG (Scalable Vector Graphics) (Ferraiolo 2001), and SMIL (Synchronized Multimedia Integration Language) (Hoschka 1998). With the introduction of XML and RDF, and new developments such as RDF Schema and DAML+OIL (Harmelen 2001), the Semantic Web is rapidly taking shape. (Ossenbruggen 2002) discusses current Semantic Web infrastructure, which is composed of document representation language (such as XHTML, SVG and SMIL), knowledge representation language (such as RDF and RDF Schema) and the underlying XML language. Applications that use this infrastructure include W3C's Platform for Internet

Content Selection (PICS) (Miller 1996), Platform for Privacy Preferences Project (P3P) (Cranor 2000), and the Dublin Core (Weibel 1995).

One objective of JIT systems is to provide semantics for applications. Technologies in the semantic Web could help us develop JIT systems. For example, metadata are used to re-identify virtual documents and elements.

2.3 Hypermedia Functionality

2.3.1 Hypermedia Functionality Overview

Hypermedia systems include many navigation, annotation and structural features which take advantage of the node and link structure to support authors and readers. These features are called hypermedia functionality.

They could be classified as follows:

Hypermedia Structuring Functionality: Hypermedia structuring functionality includes global and local overviews; trails and guided tours; node, link and anchor typing; as well as keywords and other attributes on all of these.

Hypermedia Navigational Functionality: Navigational functionality encompasses access ranging from information retrieval to browsing. This includes structure-based query (content-based search); history-based navigation; bi-directional linking; dynamic and computed linking; and process enactment or execution through link traversal.

Hypermedia Annotation Functionality: Annotation functionality includes user-declared links, comments and bookmarks, etc.

2.3.2 Basic Concepts of Hypermedia Functionality (Bieber 1997)

Node: a unit of information. Also known as a frame (KMS) or card (Hypercard, Notecards).

Link Anchor: an area within the content of a node, which is the source or destination of a link.

Link Marker: The physical marking that shows the location of a link anchor when displayed. This is also called a "hotspot" or sometimes "button".

Link: the relationship between two anchors, stored in the same or different database.

Link Traversal: the activity to visit links.

Selection: an area in the screen, which contains the data (such as string, image) of an anchor.

Viewer: a program to display a document or media.

Annotation: the linking of a new commentary node to an existing node.

Path/trail: an ordered set of nodes or anchors.

Global and Local Overview: Overview diagrams provide pictures of neighborhood nodes. Global overview diagrams provide an overall picture and can also serve as anchors for local overview diagrams. Local overview diagrams provide a fine-grained picture of the local neighborhood of a node.

Guided Tour: A guided tour is a system-controlled path that can be entered and exited at the user's will.

Node Types: Node types provide semantics characterizing a node and its content.

Link Types: A link type represents the semantics of the relationship between the source and the destination of a link. (Allan 1996)

Anchor Types: the semantic types of an anchor.

Node/Link/Anchor Attributes: in addition to types, other semantics can be attached to nodes/links/anchors, such as labels, names, keywords or timestamps. (Bieber 1997)

Content-based Query: search a node or the web (an associative network) based on the node content, by using keywords (in conjunction with Boolean operation), or weighting of words based on their statistical properties.

Structure-based Query: different from traditional text string search on node content, structure-based query is based on node and link attributes.

Backtracking: Backtracking visits previously visited nodes. Rosenberg (1996) notes three reasons for backtracking: to review the content of a previously visited node, to recover from a link chosen in error; and as part of undoing. Conceptually, however, backtracking differs from *undoing* in that, backtracking returns the reader to a previously visited node in its current state.

History List: A history list is an ordered list of nodes that were examined in a particular session.

Bi-directional (Multi-directional) Linking: Links can be followed in either (multiple) direction by user choice.

Arbitrary Jump: Arbitrary jumps or gotos can be provided enabling users to go to any node in the system.

Landmark: "Landmarks" or prominent nodes are nodes that can always be accessed from anywhere in the system. Landmarks are usually provided by an author as opposed to bookmarks which are created by the end user (reader).

Bookmark: A bookmark list is similar to a history list except that a bookmark is placed by the reader. Also a bookmark can be seen as a single-ended link saved by the reader. Hence, a bookmark list is smaller, more manageable and more relevant to the user.

Embedded Menu: Embedded menus, as opposed to explicit menus, allow the user to select an item embedded within the text of a node and can be selected using a touch screen, cursor keys or mouse pointer.

Transclusion: Ted Nelson proposed transclusions as a mechanism for having the exact same object (document content) included in multiple places. A transclusion is not a copy, but rather the same exact instance (or a pointer to that instance).

Warm/hot Links: Warm and hot links are relationships that create a channel between the two endpoints, through which data flows from one document to the other. Warm and hot links are not pointers, but actual copies of the data, which can update automatically. With *warm links*, users explicitly ask to update the node content. With *hot links*, content is verified and updated automatically as soon as the data are displayed.

2.3.3 Links

Links connect related concepts or nodes. They can be bi-directional thus facilitating backward traversals. Links can also be semantically typed (such as specification link, elaboration link, membership link, opposition link and others) specifying the nature of relationship (Conklin 1987). Links can be either referential (for cross-referencing purposes) or hierarchical (showing parent-child relationships). Links have many different

aspects, (such as static, dynamic, automatic and external) which will be discussed in the rest of this section.

2.3.3.1 Static vs. Dynamic Links. Static links are direct, persistent connections from one node or anchor to another. Dynamic links on the other hand are computed each time an attempt is made to traverse them.

The static links suffer from a variety of problems (Mayfield 1997). These include:

- Dangling links: when a link points to a node that is subsequently moved or deleted, that link becomes stale; attempts to traverse it will fail.
- Inexpressiveness: some conventional systems lack link semantics information. Even in systems that do provide link types, if the number of link types is set in advance, then the information content that can be attached to a link is restricted to the defined set.
- Expensive construction: manually constructing links is time-consuming and expensive.
- Inflexibility: manual links are created once and are thereafter fixed; they are unable to rearrange themselves to suit the needs of the moment.
- Duplication: The semantic knowledge implicit in a particular link or link type cannot easily be reused or generalized.
- Irrelevance: When a node's content changes or other conditions change, the link may no longer valid.

Dynamic links are created at run-time rather than being generated earlier (pre-computed links). One approach to dynamism in hypertext focuses on computing links based on relationships or similarities between texts or passages of text. In this approach, the link is not defined as a pointer from one hypertext node to another, but rather as a query that leads to a different node (Bodner 1999). Pre-computed links can be constructed at any time, whereas dynamic links are computed at the moment they are required (Ashman 1997). A dynamic link can take into account the specifics of the

current user interaction. The query might be based on a combination of the browsing history, user profile, content of the current document, etc.

Compared to static links, dynamic links are more flexible than static links. Dynamic links are dynamically computed according to the underlying relationships between nodes or elements. Since dynamic links are computed at the time when users request to do so, there is no need to duplicate the links manually and it is not time consuming. On the other hand, dynamic links also can suffer from the dangling link problems. Dynamic link production, however, can detect a dangling link problem when it occurs and can guarantee that no dangling links are displayed. Dynamic links can allow links to be created from and to material that the link author does not control by overlaying the link within the document sent to a browser. When a node's content changes or other conditions change, dynamic link production has the potential to detect that the link is no longer valid.

Advantages of dynamic links include: a simplified interface to search functionality, reduced authoring effort, and greater opportunity for customization based on the current user's interaction history or specific task context. An additional benefit is decreased cost of maintenance. However, disadvantages include computation of each link at run-time (instead of using stored, pre-computed links), over-completeness and problems with false links. Computation links at run-time can be expensive in a large system with many users (Ashman 1997). Over-completeness occurs when the reader is presented with more links than he/she can comprehend (Thistlewaite 1997). False links may occur because of polysemy, i.e., multiple words and sentences that lead the search algorithm to incorrectly judge the similarity of text fragments. This may lead to unsound

links (Thistlewaite 1997) in the hypertext. JIT systems support both static and dynamic links.

2.3.3.2 Automatic Link Generation. In traditional hypermedia systems, links are created manually. Many of the existing links in the World Wide Web also are manually created. Two key advantages are that authors can provide carefully-defined specific relationships, and that users of the information have a context in which to understand information. However, these advantages can be difficult to realize as the size of the information space grows (Wilkinson 1999). Some of the problems are:

- The size of the collection may be simply too large to allow for human assigned relationships.
- The collection is sufficiently dynamic that human maintenance costs are too high (Thistlewaite 1997).
- User preference and interaction are unable to be considered if the links are created manually in advance.

To automatically generate links, relationships between pieces of information should be considered. Structural relationships, and semantic relationships could be used when generating links. Research work has been done in the relationship analysis area (Allan 1996 and Bieber 2000). Automatic links can be static or dynamic. Static automatic links are pre-computed links based on similarities between all pairs of related information. A similarity threshold is used to classify link types. Dynamic links are created on the fly possibly taking into account the user preferences or user interactions.

Fully automatic link generation has problems. If a similarity threshold is used, there tends to be a very uneven creation of links with huge numbers of links to and from some objects and no paths to others (Wilkinson 1999). JIT systems support both manual and automatically generated links.

2.3.3.3 Embedded vs. External Links. Davis (1995(2)) Hypermedia systems may be classified by how they store link and anchor information, and how they display link markers. Link information includes link ID, name, source and destination anchor ID (or name), link content (such as the destination file's URL). Anchor information includes anchor ID, name, physical position (such as the byte offset of the selected text in the source document) and content (such as the selected text). For example, if a user adds a link to some text in an HTML Web page, the link marker is highlighted as blue underlined text. The link information is associated with the selection. This entire blue underline text is the selection. There are three approaches to storing this information:

- **Embed the entire link and anchor information within node content at the link marker.**

Many of the earlier hypertext systems take this approach. The World Wide Web also embeds its links (as HTML mark-up). The advantage is that the document and its links form a self-contained object, which may be moved and edited without damaging any of the links the document contains. The disadvantages of embedded links and anchors include:

(1) Moving (or deleting) a document requires checking all references to this document within all other documents.

(2) It is not possible to update or insert mark-up in documents owned by another user or on a read-only media such as CD-ROM.

(3) Incorporating the same links in different documents requires to duplicate the anchor and link information.

- **Embed tags to store anchor information in source and destination nodes but store the link information externally.**

This approach was pioneered by the Sun Link Service (Pearl 1989) and is taken by many other hypermedia systems such as SP3. A link database holds information about which nodes are connected by the links, but the application is responsible for maintaining the anchor information. With all link structure held centrally, one can build tools, which navigate the links (such as browsers), as well as tools to identify dangling links.

- **Store both the link and anchor information externally.**

Intermedia, Microcosm, Hyper-G and other application take this approach, which has a number of advantages over the other approaches:

(1) Links may be made into and out of read-only material, since the pointers are stored externally.

(2) Anchors are easier to use with non-text materials.

(3) Bi-directional or multi-directional links could be supported.

(4) Users may select in which of a number of alternative webs (link databases) to store a particular link.

(5) Third party applications may be used as link service enabled viewers. Because document content contains no proprietary anchor mark-up (as in the previous approaches), any generally enabled viewer should be able to display it.

(6) A local map (fish-eye view) can be readily computed and visualized using the link database.

A major disadvantage to separating the structure from the data occurs when an editor changes a file, causing links to point to the wrong positions within the node.

How a system stores link anchors affects the media and tasks it can support, as well as the services it can provide. JIT systems store link and anchor information externally, but may embed links in Web pages immediately before display.

2.3.3.4 Semantically Typed Links. A link type (Allan 1996) is the description of the relationship between the source and the destination of a link. It has long been accepted that typed links have the potential to support a wider range of automated and user tasks within a hypertext system. One of the common criticisms of the traditional WWW is that the semantics of linking is typically highly localized. The types of links that are to be made available are typically tailored towards the specific application, considering the nature of the content, users tasks and so on. Authors have experienced difficulty in defining a manageable small set of link types that is also sufficiently rich to

capture link semantics, large sets of link types raise the issue of ambiguity. Systems that provide very few link types include KMS (Akscyn 1988) and HDM (Garzotto 1993). Systems with a small number of types often allow the source anchor to provide additional information about the link. For example, KMS has only two link types, but allows an anchor to express the semantics of the link to which it is attached. KMS also allows information about a link to be expressed in the mouse cursor that appears when the mouse is moved over an anchor. Other systems provide more link types, such as gIBIS (Conklin 1987) and TEXTNET (Trigg 1986). Whilst many of these link types are highly application specific, there have been suggestions that it might be possible to define a standard application independent core set of link types. A related issue is whether the set of link types should be defined by the development tool, by the authors of application built using that tool, or by the users themselves.

Allan (1996) presents a technique for automatically linking documents of related subject matter. Moreover, an extension of the technique, inspired by document visualization, allows automatic classification of the links into several types from a pre-specified taxonomy of links. Links can be divided into classes in several ways. Trigg (1987) provides the major divisions of internal "substance" links and external "commentary" links. Each of those is then broken down into sub-classes, and so on. In all, Trigg lists a set of 80 classes of link types. Parunak develops his own hierarchy of hypertext link types, including 27 types in three broad categories: revision, association and aggregation.

In Allan (1996), they present an amalgam of known link types and divide them into three major categories based upon whether or not their identification can be achieved automatically. The three categories are *manual, pattern-matching and automatic*. The approach presented for automatically typing a link between two documents is based upon straightforward statistical analysis of relationships (also statistical) between the sub-parts of the documents. However, statistical information retrieval techniques are imperfect and unable to handle word distinctions in meaning: the same subtleties of language that confuse a retrieval system will cause problems with the typing methods.

JIT systems can provide semantically typed links for users. JIT systems should allow users to manually create link types and also should be able to supply link types automatically based on the underlying relationships between documents and elements.

2.3.3.5 Link Service. A link service system stores links as independent objects in an external linkbase. A link service system has the ability to control the links in many aspects (Carr 1999):

- *Link inclusion*: in which way does the link embed with the requested documents.
- *Link presentation*: how are links marked in the display of a node? For example, plain links are usually marked as blue underlined text in HTML documents or black outlined buttons in PDF. Users could specify how the link presentation format would be.
- *Link prioritization*: how important the link is? It could be ranked by the user or the system based on some criteria. Link ranking is an important topic in hypermedia research.
- *Link semantics*: what kind of semantic information is associated with the link?

The Microcosm system provides open hypermedia services for early WWW browsers. It allows WWW documents to contain generic and computed links that are more flexible than the native HTML links. However, it led to the situation where the browser offered a porthole into the distributed data of the WWW, but the hypertext links were maintained locally on the PC. To make use of the links requires the user to obtain both the Microcosm linking software and the link databases themselves (Carr 1998[1]).

The Distributed Link Service (DLS) (Carr 1995) was developed to work in conjunction with existing WWW resources to support an additional underlying link service. It is composed of the server facilities that were accessed via the WWW and the client interface that worked in conjunction with a WWW browser. The server facilities are implemented as CGI scripts, and are accessed using a standard WWW server. The main scripts are those that allow the creation, following and editing of links which are stored in link databases. The client interface formulates requests for linking functions and dispatches these requests to a DLS link server for processing.

Earlier link service systems include Proxhy (Kacmar 1991), Multicard (Rizk 1992), Sun's Link Service (Pearl 1989), and Chimera (Anderson 1994). Each of these systems expects client information systems to support anchor creation and selection by embedding hypermedia calls and handling minimal hypermedia functionality. They also provide multiple levels of hypermedia navigation and annotation support, based on the degree of hypermedia compliance the client non-hypermedia information system provides, as well as a limited set of support for client systems that are not hypermedia compatible at all.

In the late 1990s, link service systems based on open hypermedia and adaptive hypermedia were developed. COHSE (Carr 2001) is a Web-based open hypermedia link service that can offer a range of different link-providing facilities in a scalable and non-intrusive fashion. The conceptual open hypermedia services are integrated to form a conceptual hypermedia system to enable documents to be linked via metadata describing their contents and hence to improve the consistency and breadth of linking of WWW documents at retrieval time (as readers browse the documents) and authoring time (as authors create the documents). PAADS (Personal Agent information Acquisition and Delivery System) (Bailey 2000) is an adaptive hypermedia link service system using agents. There are three kinds of agents in the system, which are the personal user agent, the knowledgebase agent and the linkbase agent. The personal user agent runs at the client side, tracking the user activity and accumulating information about user interests. This information is stored in a user model and then sent to the server-side knowledgebase agent. The knowledgebase agent acts as a central repository for data accumulated by personal agents and handles queries about the user model data from clients. The linkbase agent maintains a linkbase and allows the user to add or remove links.

A JIT system is a link service that stores link information in an external database and provides semantic information for links.

2.3.3.6 Link Integrity. Hypertext links are connections between documents or parts of documents. Generally the ends of links are represented by some kind of a reference to a document or part of a document. When documents are moved or changed these references may cease to resolve to the correct places.

When the object at one end of a link is not present, or is not the object that was intended by the link author, then the link is said to be broken (Ingham 1996).

Link integrity is an important topic. Most of the earlier hypertext system developers assumed that it was part of the task of the hypertext system to ensure that link integrity was maintained, and this assumption was embodied in the Dexter Model (Halasz 1994). However, with the advent of distributed hypertexts it became difficult for the system to maintain this integrity. The World Wide Web suffers particularly from this problem as the entire link normally is embedded (as a URL) in the source document. It would be impossible for the remote site to inform all documents pointing to it of the changes, since there are no backward pointers; it would be necessary to interrogate every HTML document on every machine in the world to find out whether it pointed to this server. When the link is broken on the WWW, it terminates in the infamous 404 error (Davis 1999). Efforts have been made to ensure link integrity on the WWW, such as the URN system (which will be discussed in Chapter 4).

The content reference problem is another aspect of the link integrity problem. It occurs when a document is edited, but the references into the document are not updated. This problem does not happen in WWW HTML links, as the anchors are embedded in the text and are therefore moved as the text is edited. However, most modern hypertext systems do use some sort of pointer to reference link anchor positions in documents and as the Web community starts to adopt the new XPointer (XPointer) standards, the same problem will emerge there.

Link integrity is an important issue in JIT systems. Because dynamically-generated virtual documents could vary frequently, "dangling link" problems happen more frequently than with static documents.

2.3.3.7 Links in JHE System. An anchor in the proposed JHE system is a selection (which could be part of the document or the whole document) in a dynamically-generated virtual document. A link in the JHE system is the connection between two selections or multiple selections in one or more documents. JHE supports static and dynamic links. Static links in JHE are manual links created by users, which is similar to traditional systems. What JHE offers new are *re-location* and *re-identification* processing when previously-displayed documents are redisplayed (see Chapter 3 for details). Dynamic links in JHE system are computed, based on the inherent relationships between elements, metadata structure, document structure, relationships among application specific commands and parameters, user group relationships, etc. These relationships could be analyzed before the hypermedia interface is designed or at the time an anchor is selected, and JHE then dynamically determines the links. JHE can support node types by analyzing the dynamically-generated virtual document types, through classification, and through categorizing the commands and parameters.

CHAPTER 3

JUST-IN-TIME HYPERMEDIA FRAMEWORK

This chapter describes the just-in-time hypermedia framework. It first gives differences between dynamically - generated virtual documents and static documents in Section 3.1. These differences exist in many aspects, including status, storage, reference, generation, regeneration, template, metadata, editable, versioning and anchoring. Table 3.1 shows details. Section 3.2 compares dynamic hypermedia functionality with static hypermedia functionality. The major differences are that just-in-time hypermedia requires dynamic regeneration of virtual documents, re-location and re-identification of anchors in virtual documents. Section 3.3 discusses dynamic regeneration in terms of requirements, revalidation criteria, regeneration procedure and parameters required for regeneration. Section 3.4 discusses re-location and re-identification of virtual elements, in terms of requirements, procedure, re-identification criteria and parameters required. Static edited documents could have similar problems but could be handled in simpler ways than with virtual documents. Section 3.5 describes the just-in-time hypermedia framework. Section 3.5.1 lists requirements of a just-in-time hypermedia system. Section 3.5.2 describes the conceptual just-in-time hypermedia architecture. Logical components of the JIT architecture are depicted in Figure 3.1 and described in Section 3.5.2.

3.1 Dynamically-Generated Virtual Documents vs. Static Documents

The differences between dynamically-generated virtual documents and static documents are shown in Table 3.1.

Table 3.1 Dynamically-generated vs. Static Documents

	Dynamically-Generated Virtual Document	**Static Document**
Status	Dynamic and virtual: does not exist in advance, only exists when the user visits the virtual document.	Static and real, stored persistently in some physical location with a specific file name.
Storage	Only specifications or links and other information about the document are stored; requires much less room.	The entire content of the document is stored.
Reference	Could be referenced by a unique document identifier or some specifications about the document.	Referenced by file name, location or a unique document identifier.
Generation	(1) Dynamically generated by query, search, system commands or user actions, depending on parameters specified by user. The document contains results from the database or computation module; or dynamic information, such as date or time. (2) The creation date and time, file size and content could change with each generation.	(1) Often created manually. (2) Once created, the date, time, file size and content are consistent.
Regeneration	Regeneration is required every time the user revisits the document.	No regeneration is required.
Template	Usually some kind of document templates, composition algorithms, scripts or skeletons facilitate the document's composition, organization and components during dynamic regeneration.	Can follow a pre-defined format or be free-form.
Metadata	Usually it is not difficult to analyze the metadata. Because the document is the result of query, search or calculation, useful metadata should be available. For example, the query results from database have some definite metadata, the user actions (commands + parameters) are known, the dynamic information (date, time) has clear meanings, and the calculation has definite metadata.	Metadata may be provided. If not, then it often will be difficult to infer.
Editable	Unless the virtual document can be saved as a static document, the ability to edit it is irrelevant. For the purposes of dynamic hypermedia functionality, virtual documents are not editable and only can be generated by the underlying application.	Usually editable. The date, time and file size information could be used to indicate whether the file has been edited.
Versioning	In order to version the document: (1) History information about the old version should be kept and sometimes a minimal copy of the old version is required. (2) Date and time information is not very useful to indicate that this is another version. (3) The file size could be used to indicate that it is a new version to the extent that two versions are treated as the same (further discussion later). (4) Byte-by-byte comparison is not possible, because only a minimal copy of the document is kept (this also depends on the two versions are treated as the same, as discussed later).	Usually performed by a document versioning system. Usually the original version of the document is kept. Then for each version, the system does a file comparison and keeps the file differences ("deltas"). When displaying the new version, the file difference is added to the old version to recreate the new version. Date, time and file size information is very useful for versioning purposes.
Anchoring elements in a document	To record the anchor information, internal document addressing is required, possibly using the same methods as with static documents. Every time a user revisits the document, element relocation and re-identification are required (details will be discussed later).	As long as the document has not been edited, there is no need to re-locate and re-identify the anchors. However, if anchors are stored in an external base, they technically need to be relocated, but this is straightforward.

Table 3.1 gives the following conclusion:

(1) Dynamically-generated documents can be created in many flexible ways.

(2) It is more difficult to reference and version virtual documents. To reference a virtual document, a unique document identifier is required and a resolution scheme is needed. The "just-in-time" (JIT) hypermedia system should be able to generate unique identifiers and maintain them. Every time a virtual document is referenced, the JIT hypermedia system should resolve this ID to the specifications of the virtual document to regenerate.

(3) Maintaining hypermedia within dynamically-generated virtual documents leads to requirements that are different from static documents, including dynamic regeneration, relocation and re-identification, which will be discussed later.

3.2 Dynamic vs. Static Hypermedia Functionality

When the analytical application is supplemented with hypermedia functionality for this "just-in-time" environment, it is called dynamic hypermedia functionality. Hypermedia functionality includes: structuring (*global and local overviews; trails and guided tours; node, link and anchor typing*), navigational (*structure-based query, history-based navigation and bi-directional linking*), and annotation (*user-declared links, comments and bookmarks*) functionality (Bieber 1997). The dynamic functionalities would have to operate over virtual documents.

The dynamic hypermedia created in this "just-in-time" environment causes problems not found in static environments. A comparison of the dynamic hypermedia functionality with the hypermedia generated in static hypermedia system highlights that:

(1) The destinations of user-declared links, comments and bookmarks, nodes on the history list and overviews, stops along trails and guided tours, all are dynamic and virtual. When the user traverses to them, the JIT hypermedia system should regenerate the destinations, which normally means re-executing the commands associated with these destination nodes by using some regeneration rules.

(2) Once a node is regenerated, the JIT hypermedia system must relocate the anchors that users previously had placed within this node, and should re-identify any elements that the anchors cover as the same ones as before.

(3) When analytical applications generate new query results, the JIT hypermedia system should be able to relocate and re-identify the anchors for known elements appearing in other query results. For example, a user adds a comment on an element with global access permissions so that other users can see this comment everywhere that the same element appears. In this situation, every query result containing that same element should have an anchor associated with that global comment.

(4) Overviews and structure-based search (as well as content-based search) over specification links must operate over a hypertext web that does not exist. Instead these functionalities must infer the potential of node and link existence, and node content, based on any available specifications about the nodes and links. Historical record is not enough, for users will not want to see only the nodes that have previously been generated. Instead they want to explore what possibly could be generated by their target application.

The main differences between "just-in-time" hypermedia and static hypermedia is that "just-in-time" hypermedia operates on virtual documents and virtual elements within virtual documents, requires dynamic regeneration, re-location and re-identification. These three important issues will be discussed in more detail.

3.3 Dynamic Regeneration

Some Web systems regenerate analysis screens by storing all relevant parameters in the virtual document's URL, but many Web-based and non-Web based legacy systems have no such mechanism. Other Web systems do not allow URLs with detailed parameters for security or other reasons. When the user bookmarks and returns to its URL, such systems may return the user to the home page or the initial input page. The JIT hypermedia system must provide automatic regeneration without asking the user to reenter parameters. Dynamic regeneration has the following requirements:

(1) A unique and persistent identifier for each virtual document should be generated and recorded. Every time the user traverses a bookmark or other link, he or she should return to the same virtual document, otherwise, these bookmarks or links will no longer be valid. The JIT hypermedia system could keep a list of application commands and parameters that first generated the virtual document in a database keyed to the virtual document identifier. It also could store a document template if available or deducible. When the user first generates the document, the JIT hypermedia system should determine or create a unique ID for this document. When the user tries to revisit it, the system should use the same ID, retrieve the corresponding commands and parameters from its database, and re-execute the commands to dynamically regenerate the virtual document.

(2) Whenever the JIT hypermedia system receives a document for display, there should be some way to know whether the system has displayed it before. For dynamically-generated documents, the content and document structure could be changed without any notice; there should be some criteria to decide if this is the same document as before. Following are some "sameness" criteria for determining this, listed from very rigid to very flexible:

- The file content and structure should be exactly the same (very rigid).
- The file's structure remains the same, but the content could be different. For example, element values such as the current date or current stock price may differ from the last time the document was generated, but these elements will be in the same relative location within the document as before.
- Some critical sections of the document should not change; other sections may.
- As long as the query is the same one that generated it, the system treats it as the same document (very flexible).

Which criterion the application or the user uses depends on his or her requirements. In chapter 1's example, when the analyst did the query and put some comments on that screen, she felt interested in that particular query on that particular day, so she wants the identical analysis results. If next time when she revisits this comment and the newly-generated document is not same, the JIT hypermedia system should give an "invalid bookmark" or "stale document" warning and allow the user to remove the comment or keep it. On the other hand, sometimes a comment is valid for any content generated by the same query.

(3) A static document that has been edited also has the re-identification problem. Simpler ways exist, however, to indicate that this document has been edited. If a static document is edited, for example, the editing timestamp changes. While in a dynamically-generated virtual document, every time it is generated, the date and time varies. In this situation, even though the file size does not change, the JIT hypermedia system has to re-identify whether this document is the same one as before.

3.3.1 Regeneration Procedure

The regeneration procedure has the following steps:

(1) When a user makes a link to a virtual document (a manual link, a bookmark, a link as a step within a guided tour, etc.), the JIT hypermedia system records the link information.

(2) When the user traverses that link to revisit the virtual document, the JIT hypermedia system looks for the link information. From the link information, it finds the necessary command information (including any parameter information to re-execute the commands).

(3) The JIT hypermedia system sends the command information to the underlying analytical application.

(4) The underlying application executes the commands and generates a virtual document.

(5) The JIT hypermedia system receives the virtual document and revalidates it. Revalidation of a virtual document depends on which sameness criteria the JIT hypermedia system or the user chooses, which have been discussed above.

(6) If the regenerated virtual document is revalidated as the same as that generated previously, then the regeneration is successful, otherwise, the JIT hypermedia system gives a "stale document" warning to the user.

The commands which the JIT hypermedia system sends to the application often will be the same ones that the user executed when generating the document the first time. At that time, when the user, for example, issued a series of progressive queries, inputting a set of parameters for each, and bookmarked the final result, the JIT hypermedia system

recorded the steps and parameters. When regenerating, the system repeats these steps, filling in the parameters that the user originally entered. This occurs in the background, so the user only sees the resulting document (and is not required to reenter any input parameters). For example, suppose the user invoked a wizard that presented a series of dialog screens to gather input values before generating a document. The JIT hypermedia system could invoke the wizard in the background. Instead of displaying each dialog for the user to reenter the input values, the system fills each dialog in with the stored values originally entered and submits it. The user just sees the resulting document generated by the wizard, which (presumably) is the same one as she originally saw. (The JIT hypermedia system will have to use the sameness rules discussed earlier to validate this.)

Alternatively, when integrating an application with the JIT hypermedia engine (see Chapter 5), the integrator could specify "regeneration rules" for each class of virtual documents. The regeneration rule could provide a shortcut set of commands and parameters that can generate the virtual document more directly. If a regeneration rule is available, the JIT hypermedia system could pass all required parameters. For the wizard example, a regeneration rule might bypass the wizard and issue a single command to generate the document directly, using the parameters that had been stored the first time.

3.3.2 Parameters for Regenerating Virtual Documents

In order to regenerate a virtual document, the JIT hypermedia system should record the following information: dynamic link information, virtual document information, document generation information and document re-validation information. They are described in detail as follows:

Dynamic Link Information:

Dynamic links are those leading to a virtual document specification. These include hypermedia services or functionalities such as user-declared links, bookmarks and locations within a guided tour.

Link identifier: this should be unique and persistent.

Title: link name or label.

Description: brief description of this link.

Virtual document identifier: which document this link leads to.

User information: who created this link.

Date and time: the date and time this link is created.

Virtual Document Information:

Virtual document identifier: this should be unique and persistent.

Accessibility: Some groups of users are allowed to activate the commands that create the document; other groups do not have the right. For example, a system administrator should have access to all kinds of information: a manager can view the sales records of his employees, but has no rights to see other groups' records; a salesperson may have very limited information accessibility. All these access rights are read-only; because these results are from a calculation or query, they should not be changed and there is no way to write these documents back because they are generated dynamically and no longer exist when the window is closed. (An extension to the JIT hypermedia system would be allowing users to save static versions of virtual documents that could be edited, or retrieving existing static documents, in which case access permissions must be extended to them.)

Version: if the file content or structure changes enough, mark this as a new version.

Title: the name of this virtual document, stored as a display label for its link.

Description: brief description of its specification (e.g., the query or other commands that generate it).

Size: file size.

Structure: element list and the locations of the elements.

Metadata: the metadata of the virtual document and virtual elements within it.

Revalidation Information:

Virtual document identifier: this is the same identifier as in virtual document information above.

Criteria: which criteria the JIT hypermedia system or the user chooses to identify that the newly-regenerated virtual document is same as the virtual document visited previously.

Content: some critical sections of the file or some element values might need to be kept for revalidating the document. What kind of content to keep depends on which criteria the JIT hypermedia system or the user chooses.

Generation Information:

Virtual document identifier: this is the same identifier as in virtual document information above.

Application identifier: which application generates this virtual document.

Command: which commands are used to generate this virtual document.

Parameters: which parameters are used to generate this virtual document.

Shortcut: which replacement commands can generate the virtual document more directly.

Template: which document template is used to facilitate the generation, if any.

3.4 Relocation and Re-identification

Re-identification and re-locating nodes and anchors have the following complications:

(1) The JIT hypermedia system must recognize the re-generated node is the same one as it opened previously. This issue has been discussed in detail above.

(2) After a virtual document is regenerated, the JIT hypermedia system should be able to find those anchors that were marked in this document previously. This is called re-location.

(3) The JIT hypermedia system must recognize that some content within a newly generated (or regenerated) node is the same element as one marked as an anchor previously.

(4) When the user creates a link, she needs to decide whether it should appear every time the element appears in any node, every time it appears inside that particular node only (e.g., only for that particular query), or only on that particular instance within that particular node. For example, the user may place a comment on a person's name, but only mean it for one particular paragraph inside a document in which the name is mentioned several times. This is called *relocation granularity.*

(5) As with a static hypermedia system, the JIT hypermedia system must determine which anchors (and to which links they lead) are available for the re-identified and re-located elements in a (re)-generated node. The unique identifier is crucial here.

3.4.1 Procedure for Relocation and Re-identification

The relocation and re-identification procedure has the following steps:

(1) The first time a user selects and marks an anchor on the screen, the JIT hypermedia system records the anchor information in an external anchor database.

(2) After a virtual document is regenerated and re-identified as the same node, the JIT hypermedia system looks for the anchor information for this document in the external anchor database.

(3) For each anchor inside this document, the JIT hypermedia system finds the exact position inside the document. The byte offset of an element could change if the document's content changes between generations. For example, a simple change in the current date could shift byte positions. A flexible and smart internal document-addressing scheme such as XPATH should be used for re-location purposes.

(4) For the re-located anchor, the JIT hypermedia system should re-identify that the newly-generated element is same as that selected as an anchor. For flexible criteria, it would allow anchor value to change, for rigid criteria; it does not allow anchor value to change.

(5) After the anchored virtual elements are re-located and re-identified successfully, the JIT hypermedia system looks for those hypermedia functionalities associated with these elements in the database, and associates them with anchors.

3.4.2 Parameters for Relocation and Re-identification

For relocation and re-identification purposes, anchor information, virtual element information and re-identification criterion are needed.

Anchor Information:

Anchor identifier: this should be unique and persistent.

Title: anchor name or title.

Description: brief description of this anchor.

Granularity: this anchor could appear at a particular location in a particular document, on the same element anywhere in a particular document, or could appear in any document that has the same element.

Selection content: some content of the element or some context around this element.

Location: internal addressing in the document.

User information: who created this anchor.

Date and time: when this anchor was created.

Virtual Element Information:

Element identifier: this should be unique and persistent.

Metadata: which metadata this element uses and the JIT hypermedia system should use to revalidate the element.

Version: if the value of an element changes, this could possibly be another version.

Size: size of the element.

Accessibility: who has access rights to this element.

Re-identification Criterion Information:

Element identifier: this should be unique and persistent.

Criterion: which criterion the JIT hypermedia system or the user uses to re-identify the element.

Content: some parts of the element's value or metadata, which are used for re-identification purposes.

3.5 JIT Hypermedia Framework

In a sentence, a "just-in-time" hypermedia system is a hypermedia system that can supplement hypermedia functionality for dynamically-generated virtual documents. The JIT hypermedia framework is presented by discussing the requirements and the logical components for the JIT system.

3.5.1 Requirements

As discussed in Sections 3.1 to 3.4, the major requirements and differences between the "just-in-time" hypermedia and traditional hypermedia systems are that JIT hypermedia requires *regeneration, re-location* and *re-identification.* Besides this, other requirements include virtual document support; document generation; unique identifiers for virtual documents and elements; internal document addressing mechanisms for elements; application specific metadata support; hypermedia functionality; integration with viewers and integration with information systems. These requirements are discussed in detail as follows:

- Virtual document support:

Traditional hypermedia systems deal with static documents, which have persistent file names and locations. A JIT system should be able to reference a virtual document, record its information and enable hypermedia services for these virtual documents. Also it should enable hypermedia services for static documents.

- Document generation:

A JIT system should be able to generate a document when a user asks for it. Document generation may be handled by an independent underlying information system. However, the JIT system should have the ability to control the generation process, add semantic information and display the documents on viewers.

- Document regeneration:

When a user traverses to a link anchor or bookmark that was put on a virtual document, the JIT system should regenerate the document and then revalidate the regenerated document.

- Hypermedia functionality support:

Hypermedia functionality should work for the dynamically generated virtual documents with a similar effect as for static documents.

- Re-location and re-identification:

When a user traverses a link or comment that was put on a virtual element in a virtual document, the JIT system should be able to find the location of the anchor for this element in the document. This is called re-location. The JIT system also should be able to re-identify the anchored virtual elements in the virtual document.

- Unique, persistent identifiers for virtual documents and virtual elements:

Reference to virtual documents and virtual elements requires unique and persistent identifiers across the hypermedia system.

- Internal document addressing mechanisms:

Re-location and re-identification of virtual elements require flexible and efficient internal document addressing mechanisms.

- Application specific metadata:

The metadata of the applications should be analyzed and supported to the extent that it is used when the JIT system regenerates and revalidates the virtual documents and elements.

- Integration with viewers:

A hypermedia system could support multiple viewers (e.g., Web browsers and interfaces of non-Web applications). Because application-generated documents could be displayed by different viewers, this requires integration with viewers. Integration with viewer means the JIT system has an interface (middleware) between each viewer, which could transform the JIT data formats to data formats that different viewers could understand or display.

- Integration with information systems:

To supplement hypermedia functionality for the underlying independent information systems, integration with information systems is required. Different applications give out different source documents in different formats. Each requires an interface or middleware to transform its source documents into the universal documents that the JIT system can understand and process.

3.5.2 Logical Components

A conceptual JITHS architecture is proposed in Figure. 3.1. A brief description of the conceptual architecture is as follows:

Viewer Wrapper: integrated with viewers. It is composed of the Selection Manager; the Document Presentation Manager and the Communication Manager.

The *Selection Manager* gets the selections (content, location) from the screen.

The *Document Presentation Manager* handles the layout of the internal virtual documents and translates the virtual documents to the data format that the viewer can recognize. The *Communication Manger* manages the communication between the viewer and the hypermedia engine.

Application Wrapper: manages the communication between the application and the hypermedia engine. It passes the application outputs to identify any potential elements which JHE may wish to identify as link anchors. It then translates the application specific data formats to any internal document data format that the hypermedia engine requires.

Hypermedia Engine: Hypermedia Engine is composed of the *Hypermedia Gateway*, the *Regeneration Engine,* the *Hypermedia Service Module and* the *Virtual Document Manager.*

The *Hypermedia Gateway* deals with the communication between viewer wrapper and the hypermedia engine and the communication between the application wrapper and the hypermedia engine.

The *Regeneration Engine* deals with virtual document generation and revalidation.

The *Hypermedia Service Module* supplements virtual documents with hypermedia functionality.

The *Virtual Document Manager* manages unique, persistent identifiers and other related information for virtual documents and virtual elements. It also does re-location and re-identification for the regenerated virtual documents and virtual elements.

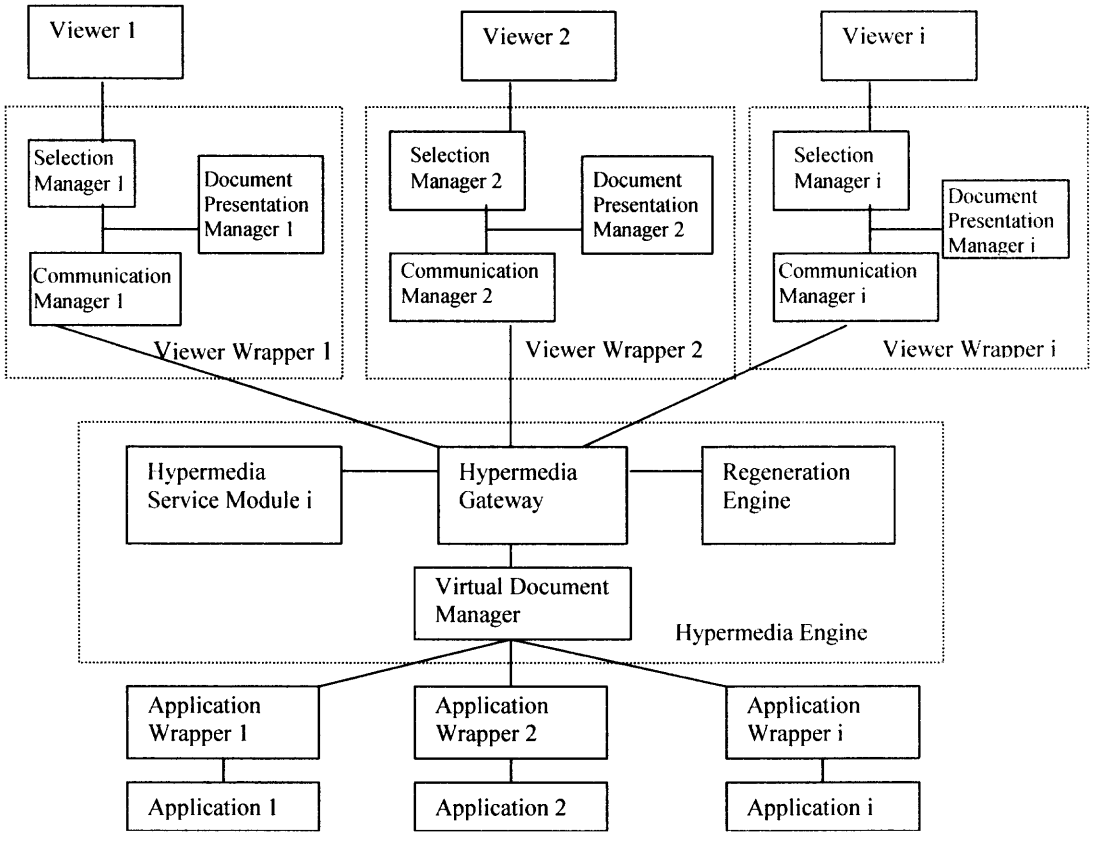

Figure 3.1 A conceptual JIT hypermedia architecture.

CHAPTER 4

RELATED RESEARCH

This chapter gives a fully detailed literature review on related research works. JIT hypermedia systems are related in following research areas: virtual documents, document structure, adaptive hypermedia, unique identifiers, and object identification and addressing mechanisms. Section 4.1 introduces concepts and lists a number of research issues on virtual documents. Existing research on virtual documents is reviewed in Section 4.1. Section 4.2 describes three types of document structure, markup languages for documents, document structure standards and document templates. Section 4.3 discusses adaptive hypermedia and reviews several existing adaptive hypermedia systems and research trends. Section 4.4 describes unique identifiers in detail. Unique identifiers are crucial to JIT systems in order to reference virtual documents, regenerate virtual documents, and re-locate and re-identified virtual elements. Some concepts and design requirements on identifiers are given in Section 4.4.1. Section 4.4.2 gives an overview on URI/URL/URN. The Uniform Resource Identifier (URI) is the most widely used identifier scheme on the Web. URLs (Uniform Resource Locator) and URNs (Uniform Resource Namespace) are subsets in URIs. The URL syntax is described and some examples are given in Section 4.4.2.2. Because URLs are not persistent, the PURL (persistent URL) and the URN standards are proposed to solve this problem. Section 4.4.3 describes the Digital Object Identifier (DOI), which is an important system for the publishing industry. Section 4.4.5 compares existing identifier systems and describes the identifiers in JIT systems. To re-locate anchors in documents, a flexible and efficient

internal document addressing mechanism is required. Section 4.5 describes types of anchors, different ways to express locations inside a document, and several existing systems or standards specifying locations. Section 4.5.1 gives overview for addressing mechanisms. Section 4.5.2 describes the HyTime standard, which has very flexible and complex ways to specify locations inside SGML documents. Section 4.5.3 describes addressing methods in Microcosm, the Open Hypermedia Protocol (OHP), and the Open Hypermedia Interchange Format (OHIF). Section 4.5.4 describes the XPath/XPointer languages, which are very useful to express locations for XML documents. Some examples are given to explain the languages in detail. Section 4.5.5 discusses object identification and addressing mechanisms used in JIT systems.

4.1 Virtual Documents

An electronic document consists of both the content and the links associated with that document. Such documents may be static and persistent or they may be generated dynamically and be virtual. A virtual document is a document for which no persistent state exists and for which some or all of each instance is generated at run time (Watters 1999).

A number of interesting research issues must be resolved surrounding these virtual documents on the Web. These issues include reference, generation, search, revisiting, versioning, authentication, and annotation (Watters 1999).

Reference: How do authors keep track of virtual documents or versions of virtual documents? In this research, a unique ID is generated and maintained for each virtual document, or each version of the virtual document. The JIT hypermedia system maintains information about the virtual documents. When the virtual document is needed, the JIT hypermedia system uses the unique ID to look for the virtual document information, and

gets the commands and parameters to recreate the virtual document if regeneration is necessary.

Generation: At what point in time is a virtual document defined? A virtual document can be defined by an author through the use of templates and links, or it can be defined as the result of a search or application. Ranwez and Crampes (Ranwez 1999) define virtual documents as a non-organized collection of Information Bricks (IB), associated with methods allowing the generation of a finished IB sequence. Methods that must take into account include predefined links between IBs if any, any parameters supplied, user actions, and a specific generation strategy. In this research, virtual documents normally are created by an application as the result of a user search or query.

Revisiting: Users have an expectation that documents found once will be available on a subsequent search. The notion of a bookmark does not apply to virtual documents in its normal, simplistic way. Bookmarks need enough information to recreate the document as it was. The dynamic regeneration in this research has a similar purpose.

Versioning: Version control has long been a concern of information retrieval research and is now a central issue for managing virtual documents. Users may want to return to an old version of a virtual document, and to navigate forward and backward in time through changes to that virtual document. Bookmarks usually take users to the current version. History information needs to be recorded in order to visit different versions of virtual documents.

Search: How do you search for virtual documents? The content normally is not available for a content-based search in JIT systems. What is the domain in which to perform the search? When will the document exist for the user to retrieve it?

Authentication: Who is responsible for the quality of the contents of a virtual document where components may come from a variety of sources and/or processes?

Annotation: The roles of information user and information supplier are merging. Readers expect to be able to add data, such as, comments, annotations, paths, and links, while they are reading.

Some research has been conducted on these issues. Martin and Eklund (Martin 1999) and Caumanns (Caumanns 1999) deal with the creation of dynamic documents by predefined templates or knowledge. Iksal and Garlatti (Iksal 2001) describe an adaptive Web application system, which generates adaptive virtual documents by means of a semantic composition engine based on user models. A virtual document is generated by computing components and applying a predefined template. For revisiting purposes,

history information is stored and the virtual document is bookmarked. Metadata information about document fragments is stored for versioning purposes. The JIT research differs from (Iksal 2001) in that, the virtual documents are created by analytical applications; the JIT system sends the user requests to applications asking for document generation, and the regeneration also depends on the analytical applications. What this research is particularly interested in is how to dynamically regenerate the virtual documents, and how to decide that the regenerated one is the same one as before.

Virtual documents in JIT systems are generated on demand in response to user inputs. The hypermedia interface gets user inputs and then sends commands and parameters to applications. Underlying applications generate source documents, the hypermedia interface intercepts the source documents and re-identifies the re-generated documents. As discussed in Chapter 3, *re-generation*, *re-location* and *re-identification* are required in JIT systems. JIT systems have similar problems encountered to virtual documents research issues described in the above sections. Existing methods in virtual documents research could be applied to JIT research. On the other hand, how JIT systems deal with virtual documents will provide additional insights concerning the above research issues in virtual documents.

4.2 Document Structure

This section describes the types of document structure, document templates, and standards for document structure.

4.2.1 Document Structure Overview

A generic document model describes the structuring rules of a document class. It defines in a generic way how specific documents are structured. It distinguishes three types of generic structures: semantic structure, logical structure and layout structure (Iksal 2001).

Semantic Structure: the semantic structure conveys the organization of the meaning of the content of a document.

Logical Structure: the logical structure reflects the syntactic organization of a document. A document can be broken down into components (chapters and titles), and subcomponents (sections, paragraphs).

Layout Structure: the layout structure reflects how the document should be displayed, such as color and font size.

4.2.2 HTML, SGML and XML

HTML, SGML and XML are markup languages for documents. A comparison of these three languages is as follows:

HTML (Hypertext Markup Language) as a widely supported document format in the World Wide Web can be characterized as being cheap, simple and intended for general use by a large audience; but too limited for more advanced hypermedia applications. HTML embeds hyperlinks in the documents; it supports both the logical and layout structure in the same HTML file. The way HTML documents need to be presented was initially hard-coded into a Web-browser; later this has been changed by the introduction of CSS (Cascading Style Sheets).

SGML (Standard Generalized Markup Language) is more customizable (thus flexible and more "powerful") at the expense of being more expensive to implement. An SGML DTD (Document Type Definition) defines the semantic structure, SGML defines the logical structure and DSSSL (Document Style Semantics and Specification

Language) [DSSSL] defines how logical documents are displayed. The purpose of a Document Type Definition (DTD) is to define the legal building blocks of an SGML document. It defines the document structure with a list of legal elements. While SGML does have some universal rules for interpreting documents, such as how a document type definition is specified, it has no inbuilt rules for how to interpret, act on or display hypertext links – that is up to the individual applications and their underlying markup language to specify.

XML (Extensible Markup Language) was developed to bridge this gap between HTML and SGML. XLink, XPointer and XSL (The Extensible Stylesheet Language) are other languages closely related to XML. XLink and XPointer define common link and addressing structures which can be used in XML documents, and XSL defines a style language for defining how XML documents should be presented to the user. XML defines the logical structure and data content of the document. A DTD or XML Schema defines the semantic structure of an XML document. XML has been widely accepted due to its simplicity, openness, standardized APIs to manipulate XML documents (such as parsing), and support of database.

4.2.3 DOM

4.2.3.1 Overview. The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. With DOM, programmers can build documents, navigate their structure,

and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using DOM.

The DOM core API allows software developers and Web script authors to access and manipulate parsed HTML and XML content inside conforming products. The DOM Core API also allows creation and population of a document object using only DOM API calls; loading a document and saving it persistently is left to the product that implements the DOM API.

4.2.3.2 The DOM Structure Model. The DOM presents documents as a hierarchy of node objects that also implement other, more specialized interfaces. Some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure for XML and HTML.

According to the DOM, a document is a hierarchical structure of nodes. Nodes are of several types, but there are three types that are generally used: element nodes, attribute nodes, and text nodes. Text nodes have no name but carry text, attribute nodes have both a name and carry text, and element nodes have a name. Element nodes may have children; attributes and text nodes are terminal. In addition the DOM specifies how to reach the children of an element node (such as the path from the parent to a child). Text and element children are held in what is essentially an array, the index in the array being determined by the order of the sub-elements in the document. Attribute children are held in a dictionary. The name of the attribute, which must be unique within an element, is used as the index.

4.2.4 Document Templates

Document templates are defined as sets of pre-linked documents that can be duplicated (Catlin 1991). Another definition of a hypermedia template states that it is a partially-created, properly formatted collection of document skeletons that can be filled in by the user (Rao 1990).

Templates automate the process of creating hypermedia collections by creating the "skeletons" of documents and linking them. They facilitate the design, organization, and presentation of a collection of knowledge in the form of hypertext.

The template can be considered as a composite object comprised of other objects such as nodes and links. The usage of a template will speed up the process of an average user's understanding of the underlying hypertext model or the metaphor. Without a template, a hypertext author will have to start constructing the hypertext collection of ideas from the beginning. Many applications such as collaborative writing, teaching aids etc., have some common basis that can be transformed into a hypertext template. Document templates are used to dynamically generate virtual documents, this includes templates of semantic, logical structure and layout structure. Some applications mix logical and layout content in one template, some uses them separately. Document templates are also important in Adaptive Hypermedia because AH generates dynamic documents based on user models, the hypermedia templates is very useful for the generation purpose.

4.2.5 Document Structure in JIT Systems

In JIT systems, the document template is for regeneration purposes. This includes templates for content and layout. The content template is for transforming the source documents, which are generated by the underlying application, into an XML document. The layout template is supplied to display the XML document on the Web browser. The semantic information is represented by a DTD or XML Schema file.

4.3 Adaptive Hypermedia

Adaptation/personalization is one of the main issues for Web applications since the late 1990s. The adaptive hypermedia (AH) community arose partly from the extensive work that had already been conducted into artificial intelligence (AI) and partly from Intelligent Tutoring Systems (ITS). AH researchers are primarily concerned with using pre-existing methods and techniques found in the fields of AI, ITS and user modeling, and extending, combining and merging these ideas to create systems that understand and aid the user in knowledge acquisition (Bailey 2002).

ITS promoted the development of educational server-side adaptive Web-based systems such as MANIC, INTERBOOK (Brusilovsky 1998) and AHA (Paul de Bra 1998). Other such server-side systems index Web sites or provide personalized interfaces to large hypermedia systems. MANIC is an open-source, cross-platform system to create and deliver courses over the WWW. MANIC allows instructors to create courses and deliver them to students as HTML slides shown in synchrony with the voice of the instructor in an audio system. AHA is a generic adaptive hypermedia system, which consists of an engine which maintains a user-model based on knowledge about concepts.

AH has also seen the development of client-side adaptive systems that follow users as they browse the WWW. Examples of these systems include WebMate (Chen 1997), Letizia and LiveInfo (Maglio 2000). WebMate keeps track of user interests in different domains (concepts) and these domains are automatically learned by WebMate. WebMate automatically extract keywords for refining document search. During search, the user can provide multiple pages as similarity/relevance guidance for the search. WebMate extracts and combines relevant keywords from these relevant pages and uses them for keyword refinement. Using these techniques, WebMate provides effective browsing and searching help and also compiles and sends to users personal newspaper by automatically spiting news sources. Letizia is a user interface agent that assists a user browsing the World Wide Web. As the user operates a conventional Web browser such as Netscape, the agent tracks user behavior and attempts to anticipate items of interest by doing concurrent, autonomous exploration of links from the user's current position. The agent automates a browsing strategy consisting of a best-first search augmented by heuristics inferring user interest from browsing behavior.

Brusilovsky (2000) divides adaptive hypermedia into two distinct areas: adaptive presentation and adaptive navigation support. Adaptive presentation systems rely on information chunks (or fragments) that can be processed and rendered in a variety of ways depending on user preferences. Adaptive navigation support includes direct guidance, adaptive link sorting, adaptive link hiding, adaptive link annotation, adaptive link generation, and map adaptation.

Adaptive systems have typically focused on the adaptive presentation of content and links based on user models (Cunliffe 2000). User modeling, the capture of information about the user such as their knowledge, tasks, attitudes and interests, could be integrated into adaptive hypermedia systems to improve user efficiency (Bailey 2001). (Iksal 2001) generates adaptive virtual documents for author-oriented Web applications providing several reading strategies to readers. An adaptive virtual document can be computed on the fly by means of a semantic composition engine using an overall document structure; an intelligent search engine, semantic metadata; and the user model. (Bailey 2001) presents a technique for cross-domain adaptive navigational support by combining link augmentation with a model of the user's spatial context. (Cannataro 2001) presents a graph-based layered model for describing the logical structure of the hypermedia. It also presents XML-based models for describing metadata about basic information fragments and "neutral" pages to be adapted. This adaptive hypermedia system is modeled with respect to three different dimensions describing user's behavior, technology and external environment.

Adaptive hypermedia systems dynamically generate documents based on user models. Although they are not dealing with dynamic regeneration, techniques about document template design and document structure recognition in these systems could be applied to dynamic regeneration in “just-in-time” hypermedia research.

4.4 Unique Identifiers

4.4.1 Overview

Unique identifiers are essential for digital objects and intellectual property. Brian Green and Mark Bide (Green 1997) give an overview of why the publishing industry needs identifiers and describe some existing standards of unique identifiers.

Unique identifiers are also crucial for object identification in hypermedia systems. As discussed in Chapter 3, unique identifiers are necessary for virtual documents and virtual elements in JIT systems. Section 4.4.1 describes some definitions about unique identifiers, requirements for designing identifiers. Section 4.4.2 and Section 4.4.3 describe existing unique identifier systems, such as URI and DOI. Section 4.4.4 compares comparisons among these identifiers and introduces unique identifiers in JIT systems.

4.4.1.1 Some Concepts. Some basic concepts about identifiers are described as follows:

Identifier: a character or group of characters constituting a value which is used to distinguish one entity from another. (CCSDS 2000)

Namespace: a set of unique values which serves as identifiers for a corresponding set of entities. (CCSDS 2000)

Unique: unambiguous in a given namespace. An identifier must specify one and only one object in that space (Paskin 1999).

Persistent: identifiers must have an unlimited lifespan even though the objects they identify may not (Paskin 1999).

Granularity: to what level of detail does the content be identified (Green 1997).

Readability: identifier syntax in a way to aid interpretation by human inspection in an application (Paskin 1999).

Dumb (unintelligent) identifier: a purely random number which can only be interpreted by reference to a central database. The number itself does not contain any meaning or information about the object which it identifies (Green 1997).

Intelligent identifier: an identifier string which serves both as a unique label and also has some meaning or information related to the object itself (Paskin 1999).

Resolution: a process in which a unique identifier is the input (a request) to a system to specify some specific output (such as the resource location) (Paskin 1999).

4.4.1.2 Some Issues and Design Requirements. Several issues about designing unique identifiers are discussed in (Powell 1997), which are listed as follows:

Granularity: What level of granularity is required? For example, traditional publishers have worked at the book or journal level using the International Standard Book Number (ISBN) and International Standard Serial Number (ISSN) as identifiers, while the Digital Object Identifier (DOI) system provides identifiers for fragments of documents.

Dumb or intelligent identifier: What degree of intelligence does the identifier provide? Some software systems use dumb identifiers and users can not get related information about objects from identifiers. For example, the Microsoft Windows operating system creates unique identifiers for IDL (Interface Definition Language) projects. Each identifier is a combination of alphabets and numbers that carrying no meaning. However, most identifier systems have some degree of intelligence.

Identification and location: An identifier is a label while location is the physical position of the object. A true identifier must remain the same regardless of the location. The URL (Uniform Resource Locator) in this sense is not a real identifier.

Persistent: How long should an identifier need to last? Internet identifiers need to outlast current Internet technology and computer systems.

Besides these issues, Paskin (1999) discusses design requirements for unique identifiers such as uniqueness, international, neutral, persistent, ease of use in automated systems, granularity. Neutral means identifiers should not be tied to a specific application and should be able to function as a common identifier for a range of applications. Ease of use in automated systems requires identifiers to have a prescribed syntax to be able to be parsed by computers. These requirements should be used to evaluate existing identifier systems and to design unique identifiers in JIT systems.

4.4.2 URI (Uniform Resource Identifier)

4.4.2.1 URI Overview. Uniform Resource Identifiers (URI) are short strings that identify resources in the Web: documents, images, downloadable files, services, electronic mailboxes, and other resources. URIs make resources available under a variety of naming schemes and access methods such as HTTP, FTP, and Internet mail addressable in the same simple way.

A URI is a compact string of characters for identifying abstract or physical resources. URIs are characterized by the following definitions:

Uniform: Uniformity provides several benefits: it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ; it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers; it allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers.

Resource: A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources. The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process.

Identifier: An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax. Having identified a resource, a system may perform a variety of operations on the resource, such as “access,” “update,” “replace,” or “find attributes.”

A URI can be classified as a locator, a name, or both. The URL (Unique Resource Locator) refers to the subset of URIs that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attributes of that resource.

The URN (Unique Resource Namespace) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or become unavailable.

4.4.2.2 URL (Uniform Resource Locator). URLs are the most widely used identifiers in Internet. In the following, the URI syntax and examples are examined to show how URLs meet the identifier design requirements.

- **URL Syntax**

The URL syntax is dependent upon a scheme. In general, absolute URLs are written as follows:
<scheme>:<scheme-specific-part>
Most of the URLs share a common syntax for representing hierarchical relationships within the namespace. This "generic URL" syntax consists of a sequence of four main components:
<scheme>://<authority><path>?<query>
Each of which, except <scheme>, may be absent from a particular URL.

- **Specific Schemes**

The URI specification covers the following schemes:

ftp	File Transfer protocol
http	Hypertext Transfer Protocol
gopher	The Gopher protocol
mailto	Electronic mail address
news	USENET news
nntp	USENET news using NNTP access
telnet	Reference to interactive sessions
wais	Wide Area Information Servers
file	Host-specific file names
prospero	Prospero Directory Service

- **Example URL**

ftp://ftp.is.co.za/rfc/rfc1808.txt
ftp scheme for File Transfer Protocol services. "ftp:" is the schema name, the other part after "//" is the schema-specific part.
gopher://spinaltap.micro.umn.edu/00/Weather/California/Los%20Angels
gopher scheme for Hypertext Transfer and Gopher+ Protocol services. "gopher:" is the schema name, the other part after "//" is the schema-specific part.

From above syntax and examples, it can be conclude that URLs are intelligent, international, neutral, unique, easy to use in automated systems and have fine granularity. A URL is intelligent because it contains meaningful information about the object. In the first example, the object is a text file, which exists in the server "ftp.is.co.za" and the user should use ftp protocol to access the file. URLs are international and neutral because they incorporate many different schemas and they are used widely all over the world. URIs are unique, there is one-to-one relationship between the URI and the object. URIs are easy to use in an automated system such as the WWW. URIs use file system concepts like hierarchy to achieve granularity at the document level. However, URLs are not persistent, which causes the "dangling link" problem on Web. This leads the URN and PURL efforts to achieve persistence.

4.4.2.3 URN (Uniform Resource Namespace). URNs are intended to serve as persistent, location-independent, resource identifiers.

- **Syntax**

<URN>::= "urn:"<NID>":"<NSS>
<NID> is the namespace identifier, such as ISBN and ISSN, which needs to register in the URN system to get a NID first. This NID is used to resolve the location (URL) by the resolution system, such as Handle. <NSS> is the namespace specific string, which is unique in the namespace. The leading "urn" is to announce that this is a URN identifier.

- **Example**

urn:isbn:123456789X
urn:telecom:61733654310
The first URN shows support for the existing ISBN scheme used by the publishing industry. The second URN shows an (hypothetical) example of an international telecommunications carrier and a particular individual's telephone number.

- **URN Resolution Service**

The URN resolution service provides the functionality that allows clients who have a URN to locate the resource it refers to, or to find the associated URC (Uniform Resource Characteristics) (Iannella 1996). A URC is a set of attribute pairs describing a resource. The client contacts the resolver service, asking it to resolve the URN. The resolver finds the URC corresponding to the URN and returns that information to the client. The information in the URC may include the location of the resource itself, which the client can use to retrieve the resource.

URNs are intelligent, international, neutral, unique, persistent, easy to use in automated systems and have fine granularity. The URN system has incorporates some application systems but is still under development and not widely used.

4.4.2.4 PURL (Persistent Uniform Resource Locator). The Persistent Uniform Resource Locator (PURL) was developed and implemented by the Online Computer Library Center Inc. (OCLC) as a naming and resolution service for general Internet resources. It is intended as an interim system to be used until the URN framework is well established. A PURL looks just like a URL, except that it points to a resolution service instead of the actual location of the digital resource. Below is a PURL example:

http://purl.oclc.org/OCLC/PURL/FAQ

"http" is the protocol, "purl.oclc.org" is the resolver address, "OCLC/PURL/FAQ" is the resource name. The client sends the PURL address to the PURL server, and the PURL server sends the real URL back to the client. Then the client can complete the URL transaction in the normal fashion.

4.4.3 DOI (Digital Object Identifier)

4.4.3.1 Overview of DOI 2001. The Digital Object Identifier (DOI), originated in 1996, is a system for:

- Identifying and exchanging intellectual property in an interoperable digital environment;
- Providing an extensible framework for managing intellectual content in any form at any level of granularity and linking customers with content suppliers;
- Facilitating electronic commerce and;
- Enabling automated copyright management for all types of media.

The DOI provides a stable, persistent link between content and a directory on the Internet to which the content owner wishes the DOI to point. As a persistent identifier, the DOI has advantages over a non-persistent Uniform Resource Locator (URL). Unlike a URL, which points to a piece of content based on its location on a computer connected to the Internet, the DOI identifies a piece of content by a permanent number that is independent and that never changes once it is assigned to the content. The DOI points to a central directory on the Internet, which in turn redirects the user's browser to the current location of the specified content. It is also possible for a DOI to reference content that does not exist in static form, i.e., does not exist as a specific file on an Internet server. In addition to its characteristics of persistence, the DOI has a syntax approved by the American National Standards Institute that can facilitate the creation of a unique, unambiguous ID for each element in a work at any level of granularity. For example, the DOI can identify an entire book or magazine, individual chapters or sections, illustrations, or tables.

The DOI provides this persistent link by providing a resolution-and-routing system similar in concept to the Internet's Domain Name System (DNS). ADNS system maintains a table recording the IP (Internet Protocol) address (which is using unique 32 digits, later 128 bits in IPv6) and domain name. Usually a domain name is represented by a URL. A user uses the domain name to visit the Web site. The DNS resolves domain names to unique IP addresses. Then the DNS takes the user to the right location (IP address). A domain's location could change, for example, a company may use a different server (which has a unique IP address) to host the Web information but users still can visit the same company's homepage through the DNS resolution procedure.

4.4.3.2 The Structure of DOI. The DOI system has two parts – the "DOI agency" and the "DOI client". The DOI agency assigns publisher IDs, issues guidelines for DOI usage and works with the relevant standards bodies to maintain the integrity of the system as a whole. The DOI clients form a distributed system that resolves any DOI to its associated URL.

The DOI system is based on the proposal that publishers should uniformly adopt formal or standard numbering schemes for their online products; the current feeling is that existing legacy systems such as the ISBN, ISSN or ISWC (International Standard Work Code) should be used as a basis for this numbering. Such numbered products and their online addresses will be registered in an online directory or database which will automatically route queries about a product to the correct location of that product, where publishers' response screens will automatically offer the enquirer a variety of commercial transaction options, such as viewing, purchasing, licensing or further routing, for example, from an article abstract to full article contents.

The DOI number will be embedded in the product itself, and becomes the chief retrieval mechanism for that object, by means of clicking on a " DOI button" in a publisher's list, in a bibliographical reference, in the DOI directory itself, or in a library if the library has purchased or licensed access to the document. The product can be as large as an encyclopedia or as small as an abstract or a graph in an illustrated scientific research article, since the publisher may choose to trade on any level of document granularity, and may therefore number as many works and 'fragments' of works as desired. Any change in the location of a registered online product or object must be forwarded by the copyright owner of the object to the online DOI directory database so that it may be updated. Enquiries will be routed seamlessly by the directory to the latest online address of an object by this method of indirect resolution. The underlying technology for this central DOI directory is called the Handle system (Handle 1997). When it receives a DOI request from a user's browser, it translates or "resolves" that DOI to the specific location which the publisher has specified for it, and then automatically re-routes the user's browser to that location.

4.4.3.3 DOI Numbering (Maree 1998). The DOI numbering system leans heavily on current print numbering schemes, with added digits indicating the registration agency and the copyright owner.

The DOI has two components, known as the prefix (publisher ID) and the suffix (content ID). These are separated by a forward slash. A typical DOI number may read as follows:

10.1234/[ISBN]0-333-45678-X

This could indicate the original DOI registration agency (10), the original copyright owner or registrant (1234), the numbering system that was used [ISBN], and the product itself (0-333-45678-X).

An expanding DOI numbering string for a literary text, if based on the ISWC approach, could read as follows:

10.540/142756081/[ISWC]-L-054.000.015.4

This would indicate the DOI issuing authority (10), the prefix assigned by the DOI agency to the rights holder (540), the CAE number (142756081) (which is an author's number), the numbering system code (ISWC) and an ISWC number for a literary text (-L-054.000.015.4).

4.4.4 Unique Identifiers in JIT Systems

According to (Paskin 1999), basic requirements for identifiers are uniqueness, persistence, intelligence, fine granularity, global scope, etc. (see Section 4.4.1.2). URLs are unique, not persistent, intelligent, but have fine granularity and global scope. PURLs are persistent. However, PURLs integrate the resolver's address with the PURL. This allows only one resolver for the system. PURLs only serve as an intermediate tool when URNs are still in the framework process. URNs and DOIs are unique, persistent, intelligent, and have fine granularity and global scope.

Identifiers in JIT systems should be unique, persistent, intelligent, and easy to use in automated systems and have fine granularity. Virtual or static documents and elements in documents should have unique, persistent identifiers for reference purposes. Hypermedia objects (such as anchors, links, bookmarks, annotations) also should have

unique, persistent identifiers. Because hypermedia objects are stored in an external database, identifiers are necessary for reference purposes. JIT systems integrate multiple applications; therefore each application could be treated as a namespace. The JIT hypermedia engine assigns a unique ID for each application. This is similar to URN and DOI systems; the URN system assigns a NID for each namespace; the DOI system assigns a publisher ID (prefix) for each publishing organization. In a URN, the <NSS> section denotes a unique ID inside the namespace, which is assigned by the registered application or organization itself. The DOI system has a similar method. In contrast, in JIT systems, IDs for documents, elements, and hypermedia objects are all assigned by the hypermedia engine; not by applications. IDs in JIT systems should have fine granularity because elements and hypermedia objects are based on documents, and elements could be any part of the document. Intelligence and ease of use are important because this makes it easier and efficient to create and maintain IDs in JIT systems.

4.5 Object Identification and Addressing Mechanisms

4.5.1 Overview

Object identification is for locating the predefined anchors and identifying the displayed contents as the same one as one previously encountered. An anchor is a selected element (words, a sentence, a paragraph or anything else) in a document, including the entire document itself. An anchor could have three different degrees of scope: specific, local and generic (Davis 1995[1]).

Specific: only applies to that particular instance of the element in that particular document.

Local: applies to any instance of that particular element in that particular document.

Generic: applies to all the instances of the same elements in all documents.

Usually the anchor information contains a unique identifier (or name) and a location specifier. The location specifier indicates the position and content of the selected area in the node. For example, in a text file, if a user selects a string from byte offset 100 to byte 110, the content is the 11 characters from byte offset 100 to byte offset 110. The location is 100, if the system uses byte offset to record locations. This anchor could have name assigned by the user.

How a hypermedia system assigns the anchor information and where this information is stored varies among different systems. HTML documents in the traditional WWW system embed anchor information. Anchors specifying outgoing links normally do not have identifiers or names; the content is surrounded by tags at the exact place of the selection. Therefore, there is no need to record the content and location information. In most hypermedia systems, such as DHM (GrØnbæk 1994) and MultiCard (Rizk 1992), anchor information is stored in an external database; when a user traverses a link, the anchor information needs to be resolved. The hypermedia system maintains a table of IDs and location specifiers (content and location). When there is a request to resolve the anchor information, the hypermedia system uses the anchor ID and searches the corresponding location specifier information for that anchor.

There are several ways to express locations inside a text file. These include:

- Naming -- e.g., an element name or id;
- Counting - e.g., the 234th byte in this file, or the 2nd item in this list;
- Querying - e.g., the first item with a type attribute whose value is 100.

There are many standards and protocols using different methods to record locations. In the following sections, several addressing schemes in detail are described, such as HyTime, OHP and XPath.

4.5.2 HyTime

The Hypermedia/Time-based Structuring Language (HyTime) provides facilities for representing static and dynamic information that is processed and interchanged by hypertext and multimedia applications. HyTime is an application of the SGML (Standard Generalized Markup Language). HyTime provides standardized mechanisms for specifying interconnections (hyperlinks) within and between documents and other information objects, and for scheduling multimedia information in time and space.

The HyTime location address module allows the identification of objects that cannot be addressed by SGML unique identifiers, and objects that are in external documents. Three basic types of address may be supported: *name*, *semantic* location and *coordinate* location. Addressing of multiple locations (a list of locations) is also possible. The syntax and semantics of these addressing mechanisms are independent of the data content notations of the data being addressed.

4.5.2.1 Locations in HyTime (Reich 1997). HyTime distinguishes the following types of location addressing: name space locations, coordinate locations, and semantic locations.

- **Name Space Locations**

Name space locations in HyTime can either be done by a) referencing a local element by its ID using IDREF (a tag in SGML to indicate that this is the unique ID in a SGML file), or b) by referencing a named object. For the latter the element-type *nameloc* is used. Instead of a single name, a name list can be used. Thus, multiple end points are possible.

- **Coordinate Locations**

Coordinate locations are used for addressing one or more sequences of data by a relative position. HyTime supplies for the following architectural forms (basic building blocks which can be re-used by different document type definitions):

1) Bit combinations, being addressed by *dataloc*, for example, representing characters in a character string.

2) Tokens, such as words, names, numbers, within a token list using *dataloc*.

3) Nodes in a tree structure, being addressed either by a *treeloc* or a *pathloc*.

- **Semantic Locations**

HyTime distinguishes the semantic locations as three types: *property* location address, *notation-specific* location address and *bibliographic* location address. *Property* locations can be used for addressing the value of a named property of an object. *Notation-specific* addressing is done by issuing a query in a specific (not HyTime defined) notation. Lastly, *bibliographic* addressing means addressing objects that the system cannot access automatically, for example, a printed book on a shelf.

- **Querying**

A dynamic method of addressing is through queries, where the exact location is not known but resolved through matching against properties in real time. HyTime defines a query language (HyQ) which has since been superseded by the DSSL query language (SDQL).

The HyTime standard was introduced as a general format to cover hypermedia structures as well as synchronization mechanisms for time based content. However, due to its complexity, these formats have not been put into widespread use.

4.5.3 Microcosm, OHP and OHIF

4.5.3.1 Microcosm. Microcosm (Davis 1995[1]) is an open hypermedia system which can generate "computed links" in the early 1990s. Microcosm defines the difference between an *anchor* (a hypertext object which describes one end of a link) and a *persistent selection* (that object within the node data which is the physical manifestation

of the link anchor, such as a colored text string). With text files, Microcosm's main approach for addressing an anchor is using byte offset (Davis 1995[1]). This can lead to the file-editing problem, which causes possible link inconsistency. When a static file is edited, links associated with this file could become invalid. Microcosm uses date and time stamps to indicate that the file content has been changed, and warns users about the possible link inconsistency. To relocate the anchors, the forward offset and reverse offset of the anchor are used. Some context (usually 10 characters surrounding the anchor) is stored. When the anchor cannot be found in the previous location, Microcosm searches the file for all occurrences of the context. If only one occurrence is found, Microcosm assumes it is the same anchor and the link is relocated to this new location.

4.5.3.2 OHP (Open Hypermedia Protocol) (Davis 1996). The Open Hypermedia Protocol (OHP) is a protocol for communications between applications and hypermedia link services. It defines the format of anchors, communication protocols and messages from and to different applications. An anchor is described as a binding between a component identifier and a *location specifier* (LocSpec). A LocSpec contains the details which describe a persistent selection, and the semantics of these details are decided by the application at the time that the anchor is created and resolved using the same algorithm when the anchor is loaded again at a later time.

The difficulty in defining a standard LocSpec is that if it is to be sufficiently flexible to meet all possible ways of expressing positions in a document as intended by the HyTime standard (see Section 4.5.2) then it will be difficult to parse and thus puts considerable onus on the application. OHP defines a LocSpec as follows:

LocSpec ::=
{\ContentType MimeType
\Content Mime encoded text string
\Count [Comma separated list of numbers]
\ReverseCount [Comma separated list of numbers]
\Name [Mime encoded text string]
\Script [Viewer Executable Script]}

The important thing to realize about the LocSpec is that although the link service may be required to store this information, it is not usually required to interpret the information in any way. The semantics of the various fields within the LocSpec are decided by the viewer at the time that it creates the LocSpec, and then stored in the link service for later retrieval and interpretation by the same viewer. Thus different viewers are not required to keep to any particular convention for representing these fields.

LocSpec is not optional. The protocol requires that the LocSpec contains at least one method for the viewer to store information about the position of an anchor.

The comma separated list of numbers used in the count and reverse count will depend on the method used by the viewer to represent positions in documents, e.g. 3, 2, 5, 423 might be used by a structured document to mean the 423rd character in the 5th paragraph of Section 3.2. Alternatively it might represent the top left and bottom right co-ordinates of a rectangle in a bitmap.

The reverse offset allows us to represent a position by counting from the end of a document. This overloading can be useful when trying to fix LocSpecs that have become broken due to document editing. The script defines an executable command interpreted by the viewer in order to locate a portion of the contents. The name may be any object understood by the viewer, which is unique to the current document, such as a bookmark name or a drawing object.

4.5.3.3 OHIF (Open Hypermedia Interchange Format). (GrØnbæk 2000) introduces an approach to utilize open hypermedia structures such as links, annotations, collections and guided tours as metadata for Web resources. This XML based data format is called the OHIF (Open Hypermedia Interchange Format), for such hypermedia structures. OHIF stores anchor information externally in an XML file based on the OHP addressing structure.

Each OHIF file represents a "context" object of the data model. A context is a (indexed) collection of other hypermedia objects. OHIF represents nodes, links, anchors, endpoints and other hypermedia objects by XML format. Each object has attributes of "id", "type" and "name". The identifier is unique.

OHIF links are general many-to-many relationships as supported first in the Dexter model and HyTime. For multidirectional links, the OHIF link carries a collection of endpoints. OHIF endpoints are responsible for addressing the anchor which is responsible for keeping the LocSpec for a specific location inside a node's content. Endpoints also hold a direction attribute, with information about whether the endpoint is considered as a source, a destination or both.

4.5.4 XPath and XPointer

4.5.4.1 XPath. XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer. XSL is the stylesheet language to specify the display layout of a XML document. XSLT is the language to transform one XML document to different layout according to different XSL file. The primary purpose of XPath is to address parts of an XML document. In support of this primary purpose, it also provides basic facilities for manipulating strings,

numbers and booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax.

XPath models an XML document as a tree of nodes. There are different types of nodes, including *element* nodes, *attribute* nodes and *text* nodes. XPath defines a way to compute a string-value for each type of node. Some types of nodes also have names. XPath fully supports XML Namespaces (an entire set of XML elements). Thus, the name of a node is modeled as a pair consisting of a local part and a possibly null namespace URI. This is called an expanded-name.

- **Examples**

The basic XPath syntax is similar to file system addressing. If the path starts with /, then it represents an absolute path to the required element. Similar to HyTime, XPath has naming, counting and querying methods to specify the elements in an XML file.

The document below (Figure 4.1 Admin.xml) represents a user who has a userid and a password. Moreover, a user can have multiple roles, in this case two: *Domain Administrator* and *Help Desk Administrator*:

```
<user>
    <userid>someone</userid>
    <password>somewhere</password>
    <roles>
        <role id="admin">Domain Administrator</role>
        <role id="helpdesk">Help Desk
Administrator</role>
    </roles>
</user>
```

Figure 4.1 Admin.xml.

The following are some XPath expressions that specify items inside admin.xml. The left side is the XPath expression, the right side is the explanation for the expression.

/user/userid/text() *select 'someone'*
/user/roles/role *select all 'role' elements*
//role *select all 'role' elements*
//role[@id='admin']/text() *select 'Domain Administrator'*
/user/roles/role[1] *select the first role element, i.e., 'Domain Administrator'*
/user/roles/role[last()] *select the last role element, i.e., 'Help Desk Administrator'*

4.5.4.2 XPointer. XPointer is the language to be used as the basis for a fragment identifier for any URI reference that locates an XML resource. It is based on the XML Path Language (XPath), and supports addressing into the internal structures of XML documents. It allows for examination of a hierarchical document structure and choice of its internal parts based on various properties, such as element types, attribute values, character content, and relative position.

- **Examples**

XPointer uses some commands such as *id, range-to,* and *string-range* to specify arbitrary strings or characters in an XML file. The following are some examples of XPointer expressions and the explanations.

XPointer(id("chap1")/range-to(id("chap2"))):

- locates the range from the start-point for the element with ID "chap1" to the end-point for the element with ID "chap2".

String-range(/,"!",1,2)[5]

- selects the fifth exclamation mark in any text node in the document and the character immediately following it.

4.5.5 Object Identification in JIT Systems

HyTime is a complex and flexible addressing mechanism based on SGML. While SGML has not been widely accepted due to its complexity, HyTime has the similar situation. OHP is a pretty simple addressing mechanism. It uses the forward and backward offsets in the file to record the location. But it cannot reflect the document structure information.

OHIF uses an XML format to describe the hypermedia composites based on the OHS data model. The advantage is it gives meta-information about different hypermedia composites and it can address to non-XML file. However, the disadvantage is that it uses the absolute offset to record the anchor position in the file. Then if the editing problem occurs, or if the document content changes, this will become a dangling link.

In "just-in-time" environments, the virtual documents are the objects being dealt with, in which the re-location and re-identification problems are solved based on the document structure. For example, if an element's value changes, it could still be treated as the same element based on the revalidation criterion discussed in Section 3.4.2. In this case, semantic information about documents and elements in documents is needed. The XPath/XPointer supplies a flexible addressing mechanism for XML documents based on the document semantic structure. Also it is possible to address non-XML documents by this standard. Thus the author considers the XPath/XPointer standard to be the appropriate addressing mechanism for "just-in-time" hypermedia systems.

CHAPTER 5

THE JUST-IN-TIME HYPERMEDIA ENGINE

To implement and evaluate the just-in-time hypermedia framework, the JIT hypermedia engine (JHE) is designed based on the Dynamic Hypermedia Engine (DHE). A hypermedia engine is a hypermedia interface that can supplement applications with hypermedia services. Section 5.1 describes several hypermedia engine systems including Microcosm, Freckles, OO-Navigator, WebVise, SFX and InfiniTe. Section 5.2 describes the Dynamic Hypermedia Engine (DHE), in terms of its architecture and component functionality. DHE is a hypermedia system that can supply hypermedia services for analytical applications by intercepting documents between application computation and the user interface. DHE can automatically generate links based on the application's underlying relationships. JHE is an extension of DHE to supply hypermedia functionality for virtual documents. Section 5.3 describes the JIT Hypermedia Engine in terms of idea, architecture, information flow, and implementation details. Section 5.3.1 describes the JHE architecture, which is the extension of the DHE architecture. New components are added to the DHE architecture to support dynamic regeneration, re-location, and re-identification. These components are defined in Section 5.3.2 (and implementation details given in Section 6.2). Section 5.3.3 uses a flowchart to depict the information flow for different situations. Section 5.4 describes implementation details for each new module in JHE.

5.1 Hypermedia Engine Overview

To supplement applications with hypermedia functionality, a hypermedia engine is needed to model and maintain application-specific hypermedia structures. Hypermedia engines execute independently of an application, and provide the application's users with hypermedia support. Notable projects include Microcosm's Universal Viewer (Davis 1994), Freckles (Kacmar 1995), the OO-Navigator (Garrido 1996), WebVise (Grønbæk 1999), SFX (Van de Sompel 1999) and InfiniTe (Anderson 2001). Details about these hypermedia engines and evaluations for them will be described in Chapter 7.

5.2 Dynamic Hypermedia Engine (DHE)

5.2.1 Analytical Applications

Analytical or computational applications logically can be split into two parts: the computational part that performs calculations and generates screens and documents for display; and the user interface — the users' viewer or browser that displays the screen or document (Bieber 1995). Microcosm's Universal Viewer and Freckles seamlessly integrate with applications but provide only manual linking. OO-Navigator provides a seamless hypermedia support for computational systems that execute within a single Smalltalk environment. The Dynamic Hypermedia Engine (DHE) (Galnares 2001) supplies dynamic linking that applies to any Web-based service (including, but not exclusively, "Web services"). The Just-in-time Hypermedia Engine (JHE) extends the concepts in DHE for just-in-time hypermedia and fully supports hypermedia for virtual documents.

5.2.2 DHE Architecture

To supplement analytical applications with hypermedia functionality, the Dynamic Hypermedia Engine (DHE) (Galnares 2001) intercepts documents and screens as they are about to be displayed on the browser, adding link anchors dynamically over elements it can recognize. When the user selects one of these supplemental anchors, DHE generates a set of relevant links. Choosing one of these links prompts DHE to send a command (e.g., a query) to the target analytical application that will cause it to generate a virtual document containing the calculation results. The target application can be the same one that generated the original display or a different one. DHE can often provide this supplemental hypermedia functionality with minimal or no changes to the analytical applications through the use of application wrappers (Galnares 2001and Bhaumik 2001). A wrapper is an interface between the hypermedia engine and the application, which mediates between them. Wrappers are described in more detail later.

DHE provides the following features in a dynamic environment:

- A Web interface for legacy systems.
- Automatically generated meta-information and hypermedia functionality.
- Support for virtual documents. Virtual documents in DHE are source documents from analytical applications. The application wrapper has the ability to parse elements in the source documents and dynamically generates links based on underlying relationships. However, DHE does not provide rich hypermedia functionality for elements and does not support dynamic regeneration, re-location and re-identification.
- Mapping rules to represent an application's internal structure. Mapping rules provide a mechanism to add commands to a link. When a user selects a link from the list of links generated, its mapping rule defines the commands for that link and the application wrapper will send those commands to the application to execute.
- Wrappers to integrate applications with minimal changes.

The current DHE architecture consists of several well-defined and separate processes, each possibly running on different platforms. It is shown in Figure 5.1.

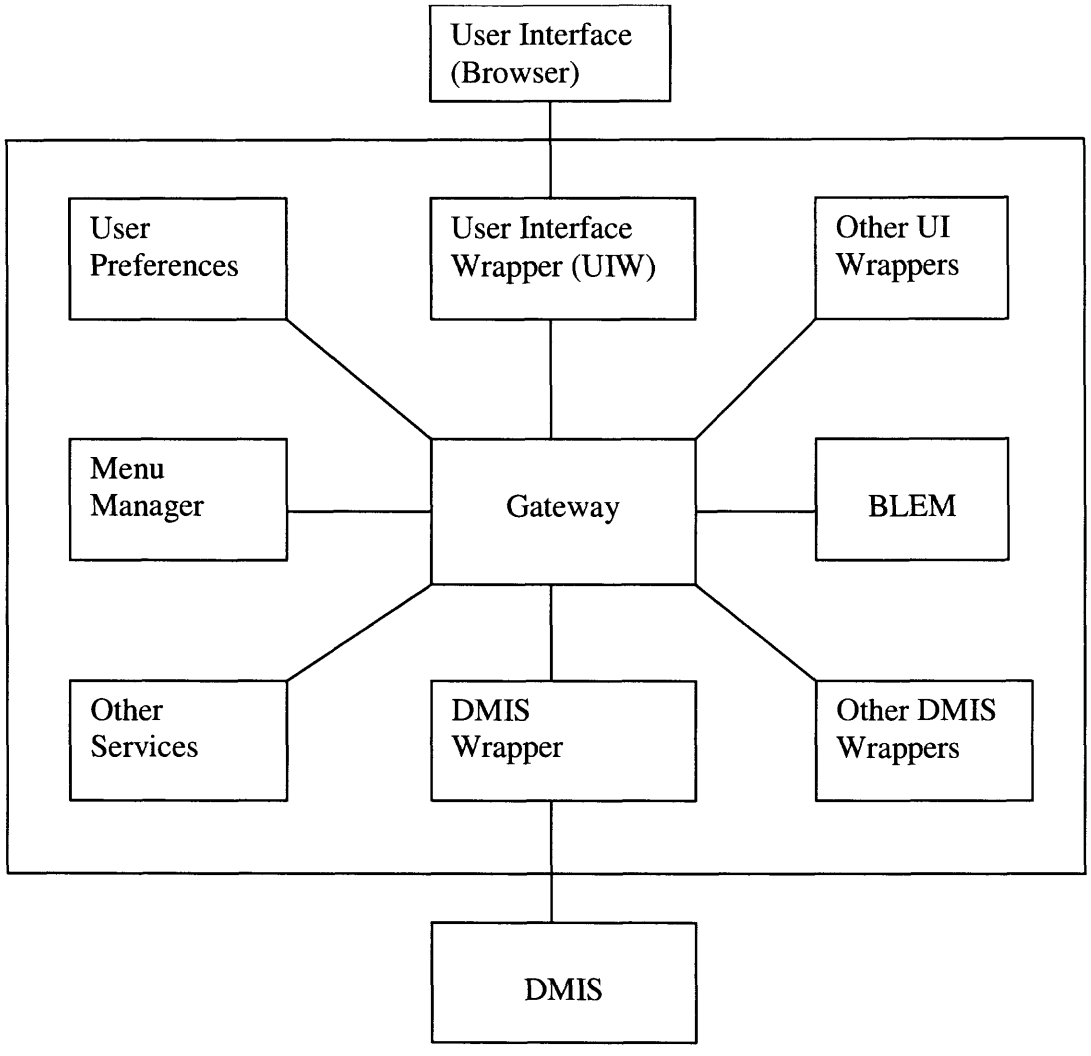

Figure 5.1 DHE architecture.

5.2.3 Component Functionality

Component functionality is described as follows:

- **User Interface (UI):** This usually runs on the user's computer (Web browser).
- **User Interface Wrapper (UIW):** This serves three important functions: First, it translates the displayable portion of the internal messages, from DHE's standard format to a format the UI can process, and vice versa; second, it handles the

communication between the UIW and the UI; and third, it implements any functionality the DHE requires of the UI, which the UI cannot provide itself.

- **Gateway (GW):** This enables the communication among the DHE modules and works as the router for all the DHE internal messages. All DHE messages pass through the GW, which then redirects them to the appropriate module.

- **Bridge Law Element Wrapper (BLEM):** This maps the application data and relationships to hypertext objects at run-time. Bridge laws are another name for mapping rules (Bieber 1992). BLEM maps the element instances in the virtual document to the global element types, and finds the links for them.

- **Dynamically Mapped Information System (DMIS):** This is an application system that dynamically generates the data requested by the user.

- **Dynamically Mapped Information System Wrapper (DMISW):** This manages the communication between the DMIS and the application system, translates the user requests from the DHE internal format to the application API, receives the output from the DMIS and converts it to the DHE format. The DMISW also identifies and marks the elements within the DMIS output to which hypermedia components are mapped.

5.3 The Just-in-time Hypermedia Engine (JHE)

While DHE dynamically generates link anchors and links for virtual documents, it currently does not support re-identification, relocation or regeneration. The "Just-in-time Hypermedia Engine" (JHE) is an extension to DHE. It can supplement analytical applications with hypermedia functionality in this "just-in-time" environment. JHE's architecture is shown in Figure 5.2. The following sections describe identifiers, the architectural components and the information flow within JHE.

5.3.1 JHE Architecture

The JHE architecture is shown in Figure 5.2.

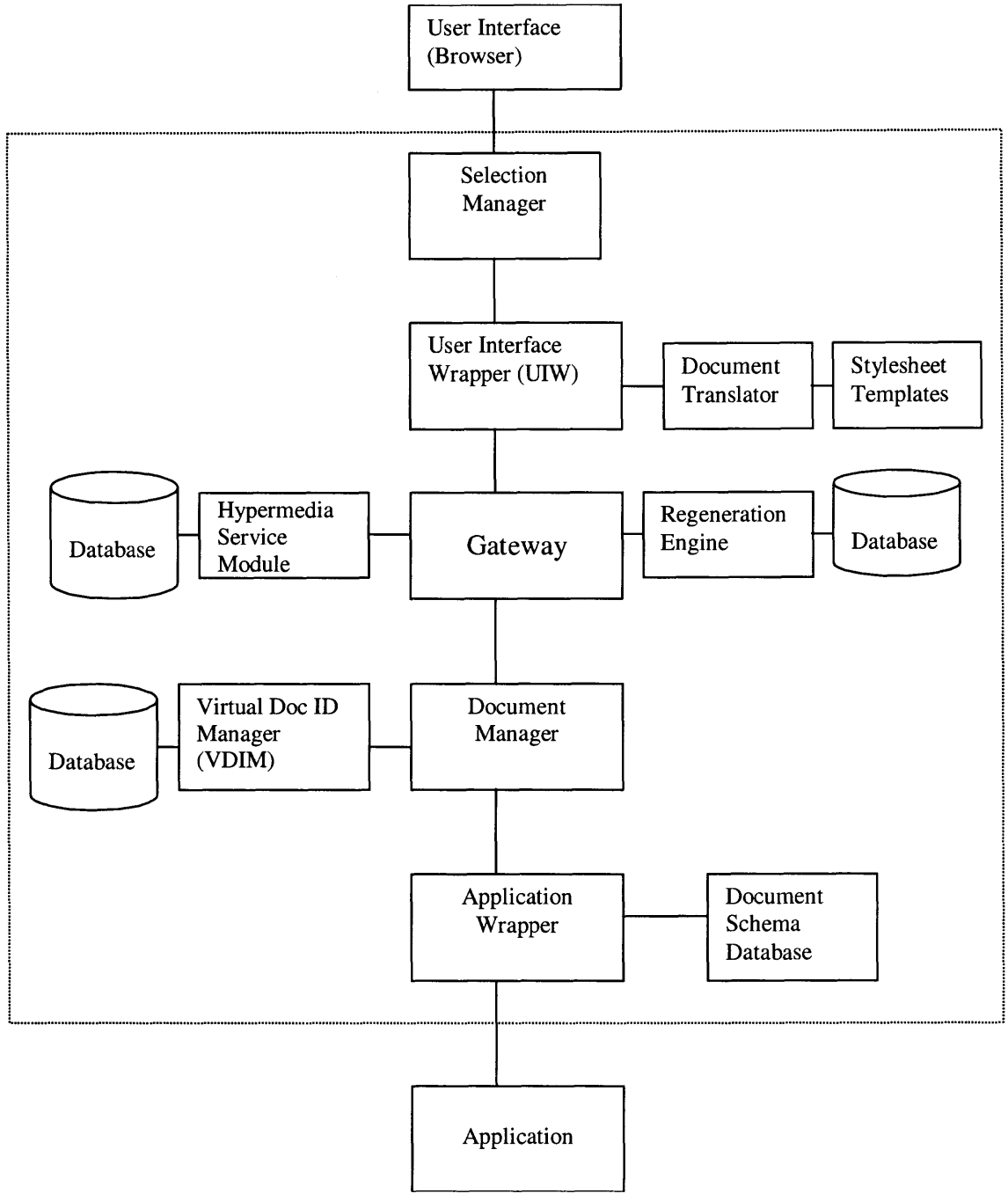

Figure 5.2 JHE architecture.

5.3.2 Component Functionality

JHE's architecture from Figure 5.2 uses many of the same component modules as DHE. Underlined components are entirely new to JHE and support re-identification, relocation and dynamic regeneration.

User Interface (UI): This usually runs on the user's computer (e.g., a Web browser providing a Web interface for the underlying analytical system). JHE displays virtual documents, anchor markers, and lists of comments and links on the UI.

Document Translator (DT): This translates a page in JHE's internal XML format to an HTML page for display if required according to the stylesheet template file. How virtual documents display varies on different Web browsers. Some Web browsers need stylesheet templates only; others may only be able to display HTML pages. In the latter case, a Document Translator is needed.

StyleSheet Templates: A stylesheet template is supplied for a virtual document or a set of virtual documents for display on the Web browser.

Selection Manager (SM): When the user selects a span of content on the UI in order to create an anchor, the SM gets the selection, and generates the anchor location information.

Hypermedia Service Module (HSM): This receives the hypermedia construct information from the user, then stores hypermedia construct information into the database. When user visits the hypermedia construct, HSM retrieves hypermedia construct information into the database and sends back to the user.

User Interface Wrapper (UIW): This serves three important functions: First, it translates the displayable portion of the internal messages from the JHE's standard format to a format the UI can process, and vice versa; second, it handles the communication between the UIW and the UI; and third, it implements the hypermedia functionality to allow users add hypermedia constructs (comments, links, bookmarks).

Application: A computer application external to the hypermedia system. This research focuses on analytical applications that dynamically generate virtual documents as the result of user queries. These may be any kind of user requests, not only database queries.

Application Wrapper (AW): This manages the communication between JHE and the application system, translates the user requests from the JHE internal format into the protocols the application requires and receives documents and screens for display from the application and converts each to an XML document and embeds it in an internal JHE message.

Document Schema Database (DSS): Application specific metadata for virtual documents and virtual elements are generated and maintained by the Application Wrapper and stored into the Document Schema Database.

Regeneration Engine (RE): This serves three important functions: First, the RE gets the necessary commands and parameters from the JHE database according to the virtual document ID. Second, to regenerate documents it sends commands to the appropriate AW for execution and gets back resulting virtual documents from the AW in XML page format. Third, it compares the newly-generated virtual document with the history information stored in RE database to revalidate it.

Document Manager (DM): This looks for the hypermedia components (links, comments, etc.) associated with a re-generated virtual document and virtual elements within it, marks the pre-existing anchors as elements, and generates a table of the hypermedia components for the virtual document. Re-locates and re-identifies the virtual elements to decide if they are the same as previously marked ones.

Virtual Document Identifier Manager (VDIM): This generates and maintains a table of identifiers for virtual documents, virtual elements and hypermedia components (such as anchors, links, and comments).

Gateway (GW): This enables the communication between the JHE modules and works as the router for JHE internal messages.

5.3.3 Information Flow

Information flows through the architecture are shown as flowcharts in Figure 5.3 and Figure 5.4. Figure 5.3 shows how to visit a virtual document. Figure 5.4 shows how to add or display a hypermedia construct. Details are described as follows.

5.3.3.1 Visit a Virtual Document. There are two ways to visit a virtual document, either by creating a new document or revisiting a bookmark. In the first case, the user should enter parameters on the application's home page (which is already integrated into JHE), and then submits it to UIW. In the latter case, he chooses the bookmark from the bookmark list displayed on the UI menu, and submits it to UIW. UIW gets the user commands from the UI, translates the necessary information into an internal JHE message and passes this message to the Gateway. The Gateway forwards it to the

AW, which passes the commands to its application for execution. The application performs the request and sends the resulting screen or document to the AW for display.

When the AW receives the resulting document, it translates the source document into an XML document. If it is a regenerated document, the Regeneration Engine revalidates the document by applying the criteria information recorded in database. If it is a new document (i.e., there is no bookmark information about this document), the Document Manager generates a unique identifier for the virtual document and records the virtual document information (document identifier, application command, parameters, etc.) into database. Then the Document Manager looks for the hypermedia objects associated with this virtual document from the database. The Document Manager also relocates and re-identifies virtual elements in this virtual documents. Then this virtual document is sent back to the browser for display.

Details about regenerating virtual documents, relocation and re-identification will be described in Chapter 6. For a new virtual document, it still needs to do relocation and re-identification. Because "global" anchors are valid for all documents, those hypermedia objects that have "global" granularity should be associated with the new virtual document.

After the virtual document is generated and displayed on the screen, users can add new hypermedia constructs for this document. Users also can browse the hypermedia construct information. Following explains how to add or display a hypermedia construct. Figure 5.4 shows the information flow.

5.3.3.2 Add and Display Hypermedia Constructs. When a user adds a hypermedia construct, he chooses a function ("add bookmark", "add comment", "add link", etc.) from the UI menu list. To add a comment or a link, he needs to select some texts from the screen. The Selection Manager generates anchor location information for the selected texts. Then the user enters other anchor information, such as name, granularity and re-identification criteria on the screen. Then UIW packs the commands into a message and sends them to Gateway. To add a bookmark, the user does not need to select some texts from the screen. He just enters the bookmark information, such as name and revalidation criteria. Then UIW packs the commands into messages and sends them to Gateway. Gateway receives the message, and calls the Hypermedia Service Module (HSM) to process the commands. For a bookmark, HSM stores bookmark information into database. To add a comment or a link, HSM stores the anchor information into database, and then it stores comment or link information into database. After all the information has been stored into database successfully, Gateway sends a message to UIW. Finally, UIW refreshes the document and inserts signs at the side of the newly created anchors for UI to display.

When a document is generated, it is displayed with anchors. Each anchor has a small icon inserted at its side. When the user clicks on the icon, JHE pops a new window to display a list of comment and link titles on the screen. This is called "display hypermedia constructs" in Figure 5.4. In this case, HSM retrieves all comment and link information for the anchor from database. When the user clicks on the comment or a link title on the screen, HSM retrieves the comment and link information from database. For a comment, UI displays comment title and content on the screen. For a link, if the link

destination is an anchor or a bookmark, Gateway calls AW and RE to do regeneration and UI displays the destination virtual document. If the link destination is a URL, UI displays the page content on the URL.

For a bookmark, it is shown in the UI menu. When JHE starts up, Gateway retrieves all bookmark information and all bookmark titles are listed in the UI menu.

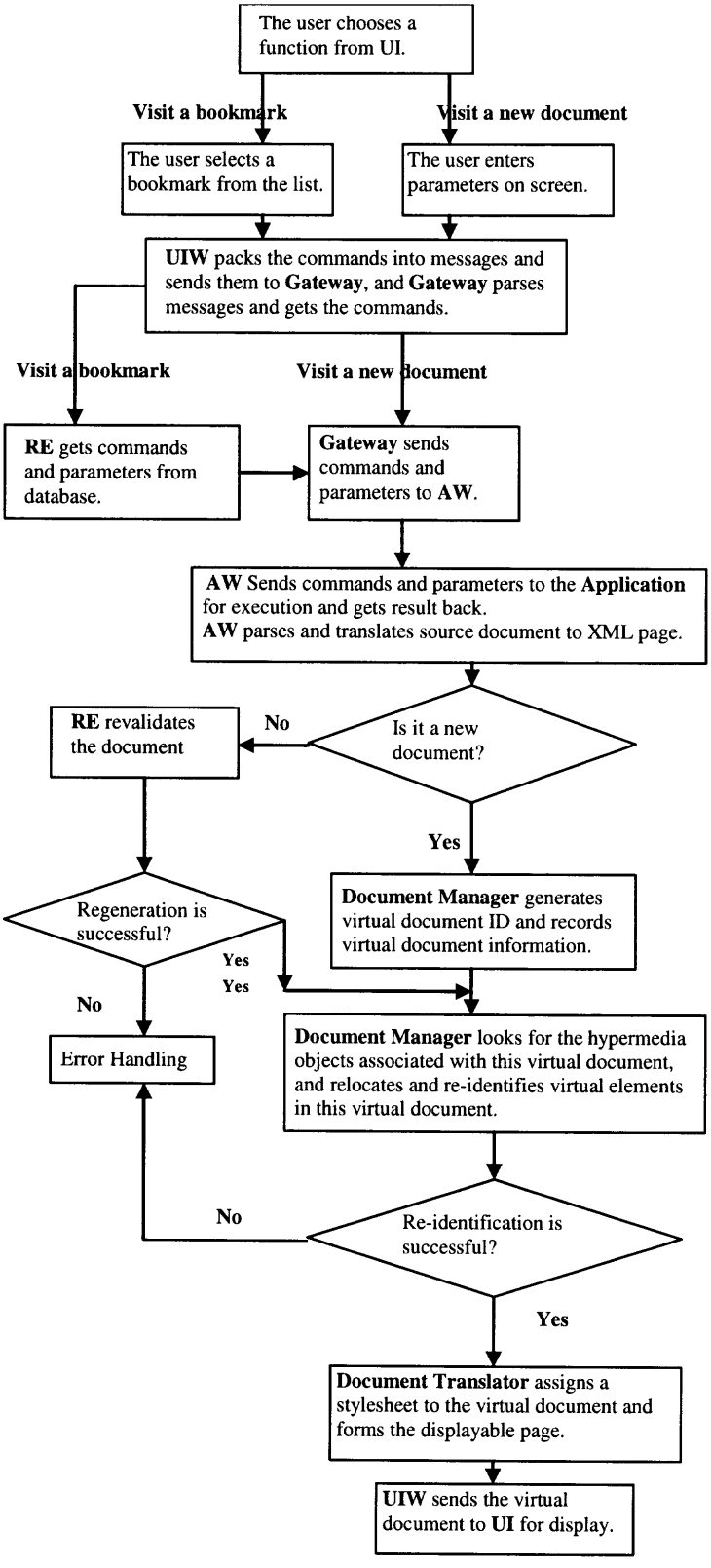

Figure 5.3 Visit a document.

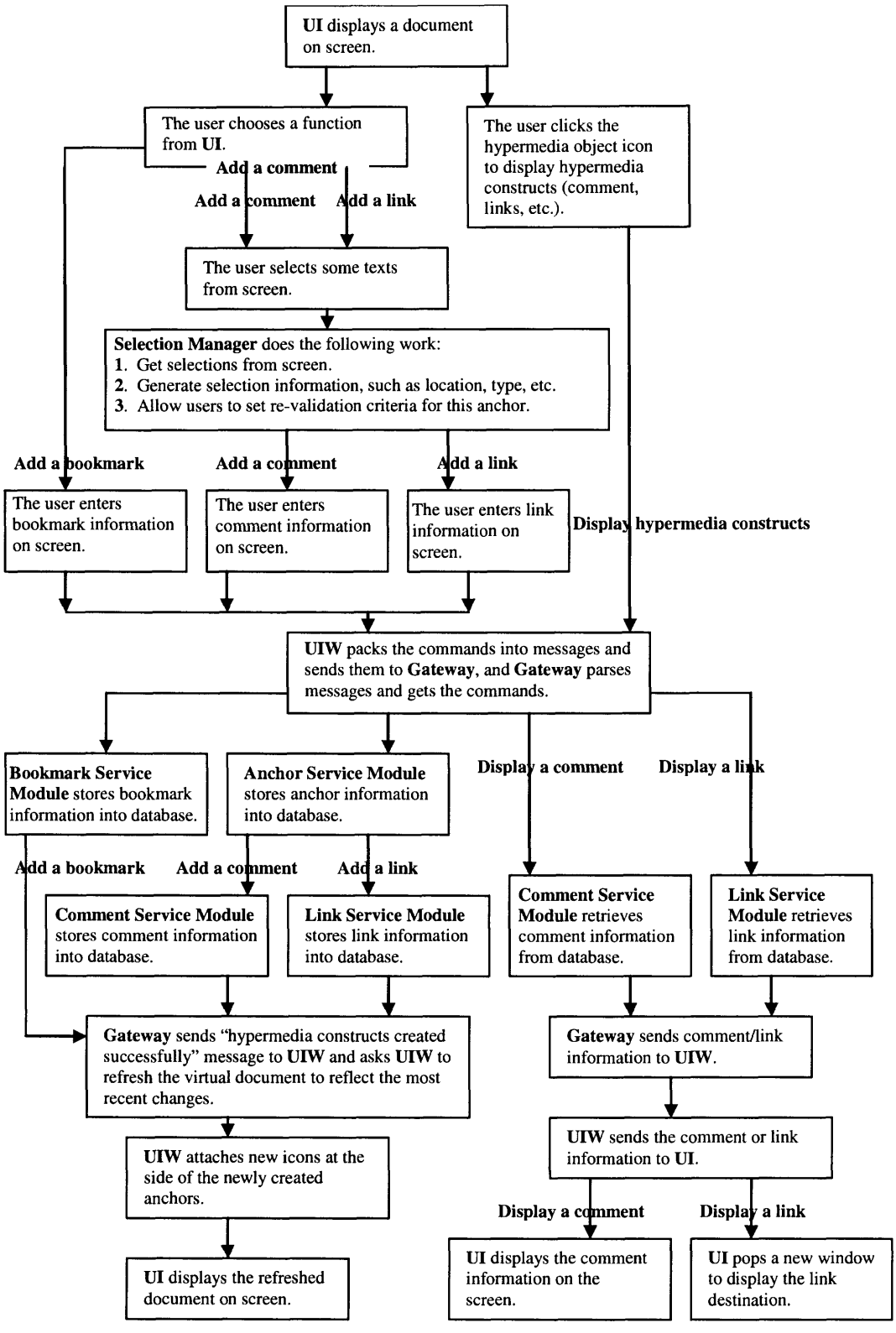

Figure 5.4 Add and display hypermedia constructs.

CHAPTER 6

IMPLEMENTATION

This chapter describes the JHE implementation details. Section 6.1 describes the application integrated in the JHE system, including the analysis of the application module and its output. Section 6.2 describes implementation details for each module in the JHE architecture.

6.1 Application

Currently, NASA's NSSDC (National Space Science Data Center) Web system is used as the test bed. The NSSDC system provides space science data and many scientific models to calculate and describe space data. Section 6.1.1 describes the NSSDC system. Section 6.1.2 introduces the earth's magnetic filed models. Section 6.1.3 gives description of the T96 model as an example of the magnetic field models. It also discusses the input parameters and output documents of the T96 model. The T96 application produces calculation results based on user input parameters. The documents are generated dynamically, which varied by different input parameters. This is a good test module for JHE.

6.1.1 NSSDC

The National Space Science Data Center serves as the permanent archive for NASA space science mission data. "Space science" means astronomy and astrophysics, solar and space plasma physics, and planetary and lunar science. As permanent archive, NSSDC

teams with NASA's discipline-specific space science "active archives" which provide access to data to researchers and, in some cases, to the general public.

NSSDC provides online information bases about NASA and non-NASA data as well as spacecraft and experiments that generate NASA space science data. NSSDC also provides information and support relative to data management standards and technologies. The web site is http://nssdc.gsfc.nasa.gov/.

6.1.2 Magnetosphere Field Models

The Earth's magnetosphere is a very dynamic system. Its configuration depends on internal and external factors. The first factor is the orientation of the Earth's magnetic axis with respect to the Sun-Earth line, which varies with time because of both the Earth's diurnal rotation and its yearly orbital motion around the Sun. Another important factor is the state of the solar wind, in particular, the orientation and strength of the interplanetary magnetic field, "carried" to the Earth's orbit from Sun, owing to the high electrical conductivity of the solar wind plasma.

Modeling of the global geomagnetic field has a unique place in the Sun-Earth connection studies, since that field underlies all processes in the near-Earth space environment:

- It links the interplanetary medium with the upper atmosphere and ionosphere;
- Guides energetic charged particles;
- Channels low-frequency electromagnetic waves;
- Confines the radiation belts and controls the auroral plasma;
- Directs electric currents;

- Stores huge amounts of energy, intermittently dissipated in the course of magnetospheric sub storms.

Magnetic fields determine key properties of the geospace plasma, in particular, its anisotropy, which makes it possible to compare observations made in different regions of space by mapping them along the magnetic field lines.

There are many magnetic field calculation models, such as T96 model, IGRF/DGRF model, and IGS model. Section 6.1.3 describes T96 model, which is integrated in the JHE system as a test application.

6.1.3 T96 Model

The T96 model code calculates dipole and Corrected GeoMagnetic (CGM) coordinates and several other geomagnetic field parameters for geographically specified points at the Earth's surface (6371.2 km) or in near-Earth space.

Figure 6.1 shows the homepage (http://nssdc.gsfc.nasa.gov/space/cgm/t96.html) of the T96 model. It has a query table, which allows users to enter parameters, and then gives out the calculation result. Figure 6.1 shows the required input parameters, described as follows:

- Year between 1945 and 2005 (in one year increments);
- Day of the year, (from 1 to 366), hour, minute, and second in UT;
- Geocentric latitude and longitude of a given point;
- Altitude of the point above the Earth's surface (40,000 km is an upper limit);
- Solar wind parameters: Density (n/cm^3), Velocity (km/sec), IMF By (nT) and Bz (nT) components in the GSM coordinates, and the Dst-index.

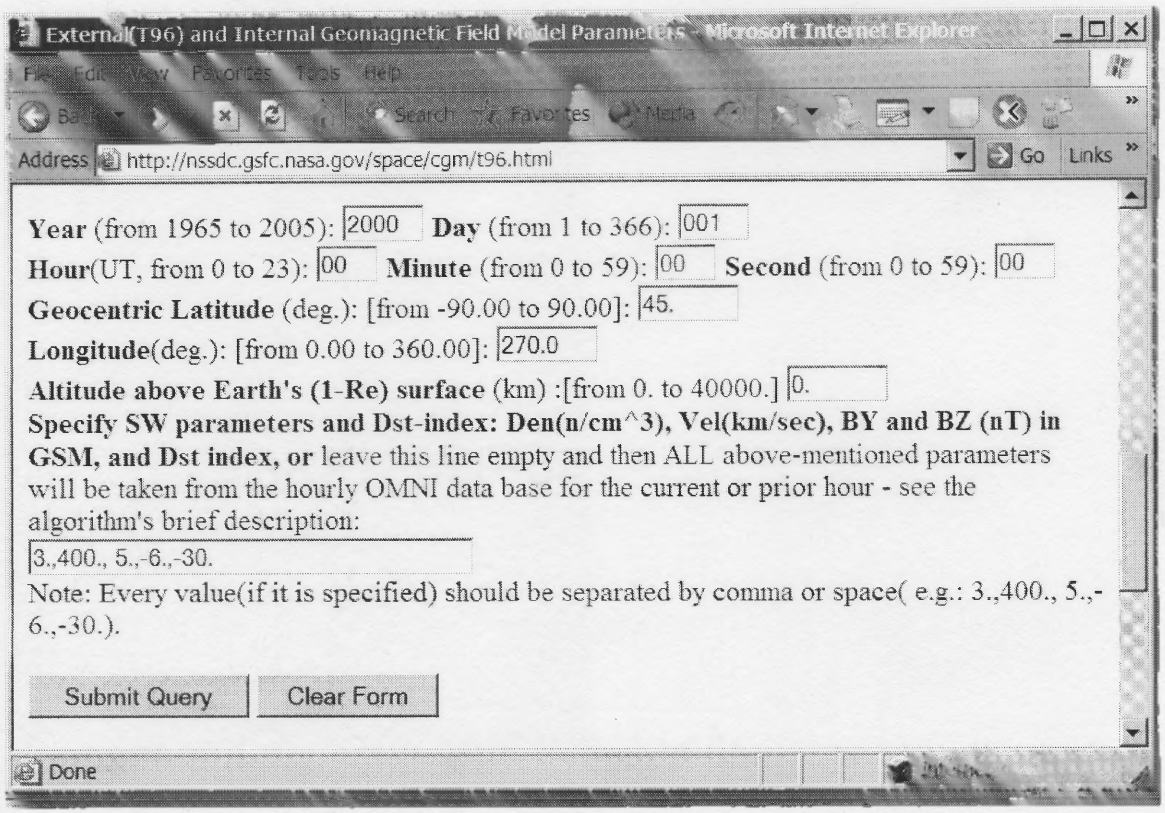

Figure 6.1 T96 model query table.

Figure 6.2 shows an example of calculation result. The output has two parts: input parameters and computed output parameters for a given point. The output parameters are:

- Dipole and CGM coordinates;
- DGRF/IGRF magnetic field component values H (nT), D (deg.), X (nT), Y (nT), and Z (nT);
- External sources, geomagnetic field component values X (nT), Y (nT), and Z (nT) at the given point;
- Apex of the magnetic field line and corresponding B-min field value which resulted from the internal and external sources;
- Geocentric coordinates of the B-min point radially projected to the Earth's surface;

- Geocentric coordinates of the realistic, magnetically conjugate point and its dipole and CGM coordinates, as well as the internal and external geomagnetic field component values at this point.

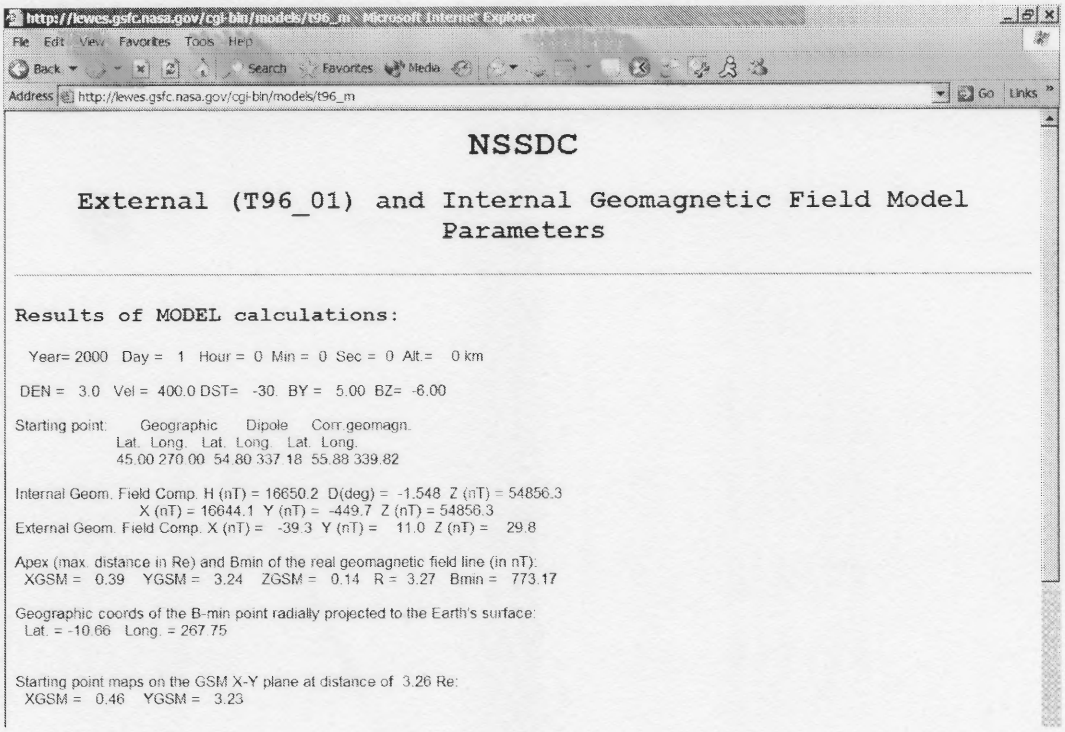

Figure 6.2 Calculation result.

The source document is an HTML file. When a user enters in different parameters, the calculation results are different, however, in similar format. Figure 6.3 shows another calculation result. It shows that many elements have different values, such as "month", "day", "minute", etc. The absolute location for each element has changed from the original one. In this case, recording byte offset for each element is not a good way to record an anchor.

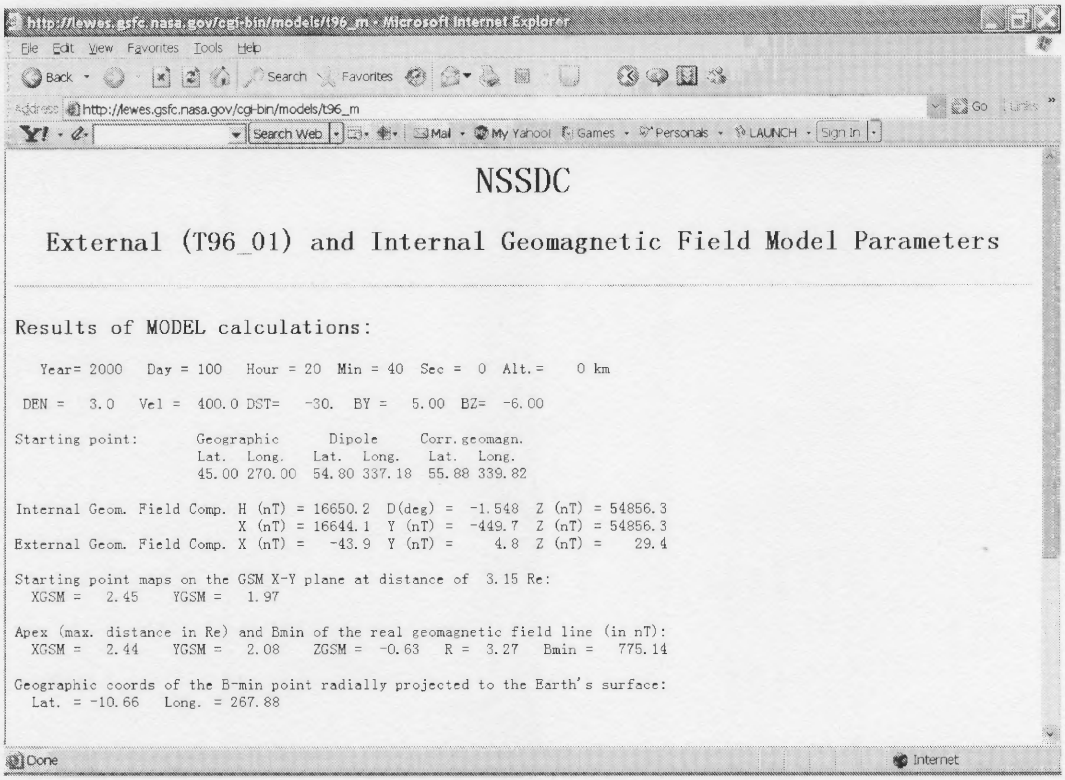

Figure 6.3 Another example of T96 model calculation result.

When a user enters some parameters into the T96 table, and click "submit query", the NSSDC server will get the query parameters and calculate the T96 magnetic field model, then sends it to the browser. Then suppose the user bookmarks the calculation result. The next time the user revisits the bookmark, it should give the bookmarked result to the user. However, the Web browser gives out the error message, shown in Figure 6.4.

Unlike many web sites whose query parameters are added after the URL, this T96 model web site hides query parameters in background HTTP requests. In this case, general bookmarks by current Web browsers cannot do dynamic regeneration.

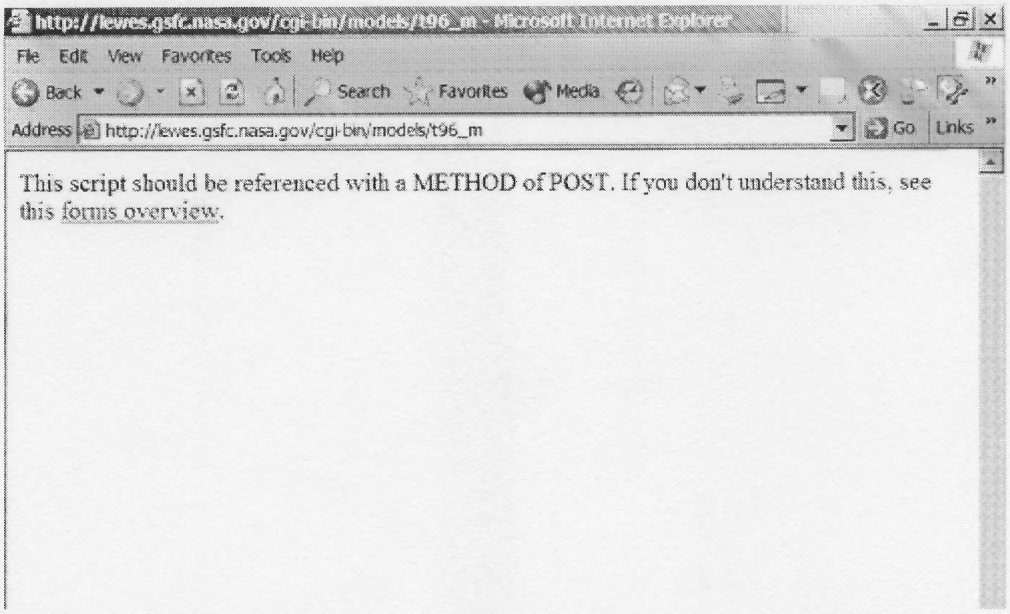

Figure 6.4 T96 error.

6.2 Implementation Details

This section describes implementation details for modules. JHE is a Web server that can integrate many external applications and supply hypermedia functionality for them. JHE uses Apache Tomcat Web server (http://jakarta.apache.org/tomcat/) and Java language to do programming. Most JHE modules are programmed in Java to run together with the Apache package. JHE uses JavaScript language to program on user interface design, which is part of the User Interface Module. The Selection Manager also uses some JavaScript functions to get selections from window.

The Application Wrapper (AW), Hypermedia Service Module (HSM) and the Regeneration Engine (RE) deal with regeneration. The AW intercepts the application document and parses the document into an XML document. Section 6.2.4 describes the AW. The HSM stores and retrieves bookmark information. Section 6.2.3 describes the HSM. The RE revalidates the document. Section 6.2.5 describes the RE.

The Document Manager (DM) and the Selection Manger (SM) deal with relocation and re-identification. The SM generates anchor information. Section 6.2.6 describes the SM. The DM finds all anchors related to a virtual document, and find the exact positions of the anchors, and re-identify the anchor content. Section 6.2.7 describes the DM.

The User Interface Wrapper (UIW) contains several User Interface (UI) menus to allow users to add bookmarks, comments and links to a virtual document. The HSM stores and retrieves bookmark, comment and link information into database. Section 6.2.7 describes the UIW.

Unique identifiers are crucial for virtual documents, anchors, comments and links. Section 6.2.1 describes identifiers. JHE has a database system to store all data and information for virtual documents and hypermedia constructs. Section 6.2.2 describes the JHE database system and database descriptions of hypermedia constructs.

6.2.1 Identifiers

The unique identifiers in the JHE system include document, anchor, and hypermedia object identifiers. They are unique numbers generated by the JHE system. The generation algorithm uses the system time stamp. In Java language, the command is:

```
long t1 = System.currentTimeMillis();
```

The virtual document identifier is the absolute value of the system time stamp. The anchor identifier is a label "A" plus the system time stamp. Following shows some identifiers examples:

Document identifier: 1101174376250
Anchor identifier: A1101176844468
Comment identifier: C1101580932171

Link identifier: L1101176948060

All identifiers are generated by the Identifier Generator Module and stored in a database.

6.2.2 JHE Database

The JHE database stores information about virtual document, bookmark, comment, link and application commands. The Java language has the JDBC interface for the developers to store and retrieve information in database. The JDBC interface needs a bridge driver and a data warehouse depending on different operating systems. The bridge driver is a middleware between the native database driver and the JDBC driver. A data warehouse is an application that allows users to create table and store records in some database file. For example, in the Microsoft Windows system, the native database driver is ODBC, the data warehouse could be Access, Excel, dBase or Visual Fox Pro. Currently JHE is running on Windows XP and it is using the Access database system. The data is stored in some .mdb file. The ODBC driver can use the Access database system, but JDBC can not use the Access database system directly, thus an ODBC/JDBC bridge driver needs to be installed. JHE uses the RmiJDBC (http://rmijdbc.objectweb.org) bridge driver. Implementing the JHE database module has following steps:

(1) Download the RmiJDBC driver from:
http://rmijdbc.objectweb.org/download.html.

(2) Compile the source codes and install the RmiJDBC driver.

(3) Use the Access application to create an empty Access database file called jhe.mdb.

(4) Setup the ODBC datasource to use jhe.mdb according to the following web page:
http://rmijdbc.objectweb.org/Access/access.html.

(5) Use the Access application to create or modify database tables. JHE has "Anchor", "Bookmark", "Comment", "Link" and other tables.

(6) Create a JHEDatabase Java class to use the JDBC interface. This Java class can use the JDBC driver interface to store and retrieve information into the database. It uses the SQL (Standard Query Language) to store and retrieve data.

6.2.3 Hypermedia Service Module (HSM)

The HSM is a Java class that stores and retrieves hypermedia construct information into the JHE database. There are three hypermedia functionality menus displayed on the UI. They are the Bookmark Menu, Comment Menu and Link Menu. These Menus are parts of the UIW. They are HTML pages that allow users to enter parameters and submit the information to the Gateway. The Gateway forwards the hypermedia functionality command and parameters to the HSM. HSM parses the command and stores the hypermedia construct information into the JHE database. When the user clicks the hypermedia construct icons on the screen, the HSM receives the hypermedia functionality command and parameters from the Gateway. The HSM retrieves the hypermedia construct information from the JHE database and then sends it to the Gateway. The UIW receives the information from Gateway and displays the information on the UI. Section 6.2.3.1 describes the bookmark, comment and link definitions in the database, and gives example records in the database. Section 6.2.3.2 explains how to store and retrieve the hypermedia construct information from database.

6.2.3.1 Hypermedia Construct Information in the JHE Database. JHE supports anchor, bookmark, comment and link.

- Anchor Information

Here is the database table of the anchor.
Anchor = (AnchorID, QueryDocID, Location, Granularity, Selection, Criteria);
"**AnchorID**" is the unique identifier for this anchor.
"**QueryDocID**" is the virtual document identifier of the document containing the anchor.
"**Location**" is the XPointer expression for the anchor.
"**Granularity**" is the anchor's scope in the system, which has three optional values 'global', 'local' and 'specific'. See detail explanations in Section 3.3.
"**Selection**" is the selected text from the screen.
"**Criteria**" is the re-identification rule for the anchor. See detail in Section 3.3.

Following is an example of anchor information:
('A1083795853460', '1083794959299',
'XPointer(start-point(stringrange(
/html[1]/body[1]/calculationresult[1]/inputparams[1]/minute[1]/name[1], "", 1, 1))
/range-to(end-point(string-range(
/html[1]/body[1]/calculationresult[1]/inputparams[1]/minute[1]/value[1],"", 1, 1))))',
'Global', 'Min = 9', 'change');

- Bookmark Information

Following is the database description of bookmark and one example of the record:
Bookmark = (QueryDocID, Name, Criteria, Content)
('1082666227732', 'all related', '2', null)
"**QueryDocID**" is the unique identifier for the virtual document, which is also the identifier for the bookmark, it's "1082666227732" in the above example.
"**Name**" is the user declared bookmark name, it's "all related" in the above example.
"**Criteria**" is the revalidation criteria, see details in Section 3.3. The value is "0" in the above example, which indicates all related query all acceptable for regeneration.
"**Content**" is the recorded document content for revalidation purpose. The value is "null" in the above example.

- Comment Information

Following is the database description of comment and one example of the record:
Comment = (commentID, AnchorID, Name, Content);
"**commentID**" is the unique identifier for a comment.
"**AnchorID**" is the unique identifier for the selected anchor.
"**Name**" is the user declared comment title.
"**Content**" is the comment content.

Following is an example of the comment information:
('C1083794113173', 'A1083794113142', 'year', 'this is year 2004');

- Link Information

Following is the description of a link table in the database, and an example of link information:
Link = (linkID, Name, Type, srcAnchorID, dstAnchorID);
"linkID" is the unique identifier for a link.
"Name" is the user given link name.
"**Type**" specifies the destination types: anchor, bookmark, URL.
"**srcAnchorID**" is the unique identifier of the source anchor.
"**dstAnchorID**" is the unique identifier of the destination. If the destination is an anchor, then it's an anchor ID; if the destination is a virtual document, then it's a document ID; if the destination is an external URL, then it's an URL.

Following shows some examples of link information:
('L1105916271296', 'test link', 'anchor', 'A1105916271265', 'A1105916191328')
('L1105998707421', 'virtual doc link', 'bookmark', 'A1105998707359', '1105919082234')
('L1105999515671', 'external link', 'URL', 'A1105999515265', 'www.nasa.gov')

The first example shows the destination is an anchor in a document, thus the destination type is "anchor", the destination ID is an anchor ID. The second example shows the destination is a virtual document, since the destination type is "bookmark". Therefore, the destination ID is a virtual document ID. The third example shows the destination is an external URL, thus the destination type is "URL", the destination ID is a URL.

6.2.3.2 Store and Retrieve the Hypermedia Construct Information. HSM receives messages from the Gateway, and then parses the command and parameters in the request. In JHE the message is a Java Paramtable class which stores pairs of key and values in the Paramtable data structure. Figure 6.5 shows the source codes to parse and process a hypermedia functionality request from UIW. For example, UIW sends a request to add a link. The Paramtable contains the key "addLink", the key "linkID" with the value of the link identifier, anchor information, and also other link information stored as key, value pairs. HSM calls the function addLinkInfo(pTable). In the function addLinkInfo(pTable), HSM gets anchor and link information from pTable then calls the Java class JHEDatabase to insert anchor and link information into database.

```
Paramtable pTable = request.getParamtable();

if(pTable.containsKey("addComment"))
{
addCommentInfo(pTable);
}
else if(pTable.containsKey("addLink"))
{
addLinkInfo(pTable);
}
else if(pTable.containsKey("addBookmark"))
{
addBookmarkInfo(pTable);
}
else if(pTable.containsKey("hyperConstruct"))
{
objectID = pTable.getParamAsString("hyperConstruct");
hyperSize = getHyperConstructs(objectID);
}
else if(pTable.containsKey("comment"))
{
commentID = pTable.getParamAsString("comment");
ArrayList commentInfo = new ArrayList();
getCommentInfo(commentID, commentInfo);
}
else if(pTable.containsKey("link"))
{
linkID = pTable.getParamAsString("link");
ArrayList linkInfo = new ArrayList();
getlinkInfo(linkID, linkInfo);
}
```

Figure 6.5 Store or retrieve hypermedia construct information.

6.2.4 Application Wrapper Module

The Application Wrapper (AW) is a Java class that intercepts the application commands and catches the source documents from applications. Then it parses the source document and translates it into a well-structured XML document. The AW performs following steps:

(a) Catch the underlying parameters

The application's command and parameters are analyzed. In most cases, it can be inferred from the query table that the Web site supplies. Usually the HTML "form" entity has the information about the parameters and the application commands. For example, in the T96 model query table (Figure 6.1), the HTML file has a "form" shown in Figure 6.6. The HTML document in Figure 6.6 shows the application destination is http://lewes.gsfc.nasa.gov, the command is "cgi-bin/models/t96_m". There is a list of parameters needs to be filled in, such as "year", "day" and "altitude". Users can enter parameters into the text boxes on the screen, then clicks "submit" button on the screen. The Web browser will send the command "cgi-bin/models/t96_m" and parameters ("year", "day", "altitude", etc.) to the destination "http://lewes.gsfc.nasa.gov".

JHE intercepts the request by replacing the application destination with JHE's module name and the command with JHE's command. Figure 6.7 shows the intercepted query form. Here the destination application in the first line is changed to "jhe", and the command to "sc". But parameters are same as the original ones. Then the request will go through the JHE Gateway.

```
<form action="http://lewes.gsfc.nasa.gov/cgi-bin/models/t96_m" method="POST">
        <h4>Please Click and Fill the Form below, then Submit Query to
   start calculations:</h4>
        <p>
          <br />
          <b>Year</b>
           (from 1965 to 2005):
          <input type="text" size="4" maxlength="4" name="year" value="2004" />
          <b>Day</b>
           (from 1 to 366):
          <input type="text" size="3" maxlength="3" name="day" value="331" />
          <br />
          <b>Hour</b>
          (UT, from 0 to 23):
          <input type="text" size="2" maxlength="2" name="hour" value="14" />
          <b>Minute</b>
           (from 0 to 59):
          <input type="text" size="2" maxlength="2" name="minute" value="12" />
          <b>Second</b>
           (from 0 to 59):
          <input type="text" size="2" maxlength="2" name="second" value="30" />
          <br />
          <b>Geocentric Latitude</b>
           (deg.): [from -90.00 to 90.00]:
          <input type="text" size="6" name="Latitude" value="45." />
          <br />
          <b>Longitude</b>
          (deg.): [from 0.00 to 360.00]:
          <input type="text" size="6" name="Longitude" value="270.0" />
          <br />
          <b>Altitude above Earth's (1-Re) surface</b>
           (km) :[from 0. to 40000.]
          <input type="text" size="6" name="Altitude" value="0." />
          <br />
          <b>Specify SW parameters and Dst-index: Den(n/cm^3), Vel(km/sec),
   BY and BZ (nT) in GSM, and Dst index, or</b>
           leave this line empty
   and then ALL above-mentioned parameters will be taken from the
   hourly OMNI data base for the current or prior hour - see the
   algorithm's brief description:
          <br />
          <input type="text" size="40" name="field" value="3.,400., 5.,-6.,-30." />
          <br />
           Note: Every value(if it is specified) should be separated by
   comma or space( e.g.: 3.,400., 5.,-6.,-30.).
        </p>
        <p>
          <input type="submit" value="Submit Query" />
          <input type="reset" value="Clear Form" />
        </p>
      </form>
```

Figure 6.6 Original HTML "form".

```
<form action="/jhe/sc" method="post">
        <h4>Please Click and Fill the Form below, then Submit Query to
   start calculations:</h4>
        <p>
          <br />
          <b>Year</b>
           (from 1965 to 2005):
          <input type="text" size="4" maxlength="4" name="year" value="2004" />
          <b>Day</b>
           (from 1 to 366):
          <input type="text" size="3" maxlength="3" name="day" value="331" />
          <br />
          <b>Hour</b>
          (UT, from 0 to 23):
          <input type="text" size="2" maxlength="2" name="hour" value="14" />
          <b>Minute</b>
           (from 0 to 59):
          <input type="text" size="2" maxlength="2" name="minute" value="12" />
          <b>Second</b>
           (from 0 to 59):
          <input type="text" size="2" maxlength="2" name="second" value="30" />
          <br />
          <b>Geocentric Latitude</b>
           (deg.): [from -90.00 to 90.00]:
          <input type="text" size="6" name="Latitude" value="45." />
          <br />
          <b>Longitude</b>
          (deg.): [from 0.00 to 360.00]:
          <input type="text" size="6" name="Longitude" value="270.0" />
          <br />
          <b>Altitude above Earth's (1-Re) surface</b>
           (km) :[from 0. to 40000.]
          <input type="text" size="6" name="Altitude" value="0." />
          <br />
          <b>Specify SW parameters and Dst-index: Den(n/cm^3), Vel(km/sec),
   BY and BZ (nT) in GSM, and Dst index, or</b>
           leave this line empty
   and then ALL above-mentioned parameters will be taken from the
   hourly OMNI data base for the current or prior hour - see the
   algorithm's brief description:
          <br />
          <input type="text" size="40" name="field" value="3.,400., 5.,-6.,-30." />
          <br />
           Note: Every value(if it is specified) should be separated by
   comma or space( e.g.: 3.,400., 5.,-6.,-30.).
        </p>
        <p>
          <input type="submit" value="Submit Query" />
          <input type="reset" value="Clear Form" />
        </p>
      </form>
```

Figure 6.7 Intercepted HTML "form".

(b) Call the application to execute

When a user clicks the button, the UIW will send the command "/jhe/sc" together with the parameters to the JHE Gateway first, then the Gateway forwards it to the destination Application Wrapper (AW). The AW parses the query result, maps the JHE destination and command to the real application destination and command. For example, "jhe" maps to "http://lewes.gsfc.nasa.gov", "sc" maps to "cgi-bin/models/t96_m". Then the AW sends an HTTP request to the destination application with parameters (the real URL of the application). The real application (the "lewes.gdfc.nasa.gov" Web server) executes the command (the CGI command on the Web server) and parameters (the long parameter list). Then the real application sends back the document to the AW.

(c) Parse the source document

The AW parses source document according to the document structure, then translates the elements into XML elements and forms an XML document. This is different for each application. Each application has a document template. For example, in the source documents shown in Figure 6.2 and Figure 6.3, they have a common document structure, just have different element values. Parsing this document is not difficult. For example, for element "year=2000", it can be translated as an element "YEAR", its name is "year", value is "2000". The "DAY" element in Figure 6.2 has a value of "1", while in Figure 6.3, its value is "100".

(d) Translate the parsed document into an XML document

After the AW parses the source document and gets elements' values, the document is restructured and translated into an XML document based on the document structure. While the current instantiation of a document is parsed automatically, usually the

document structure is analyzed manually in advance. The document structure is described as an XML Schema. Figure 6.8 shows the document schema for T96 model.

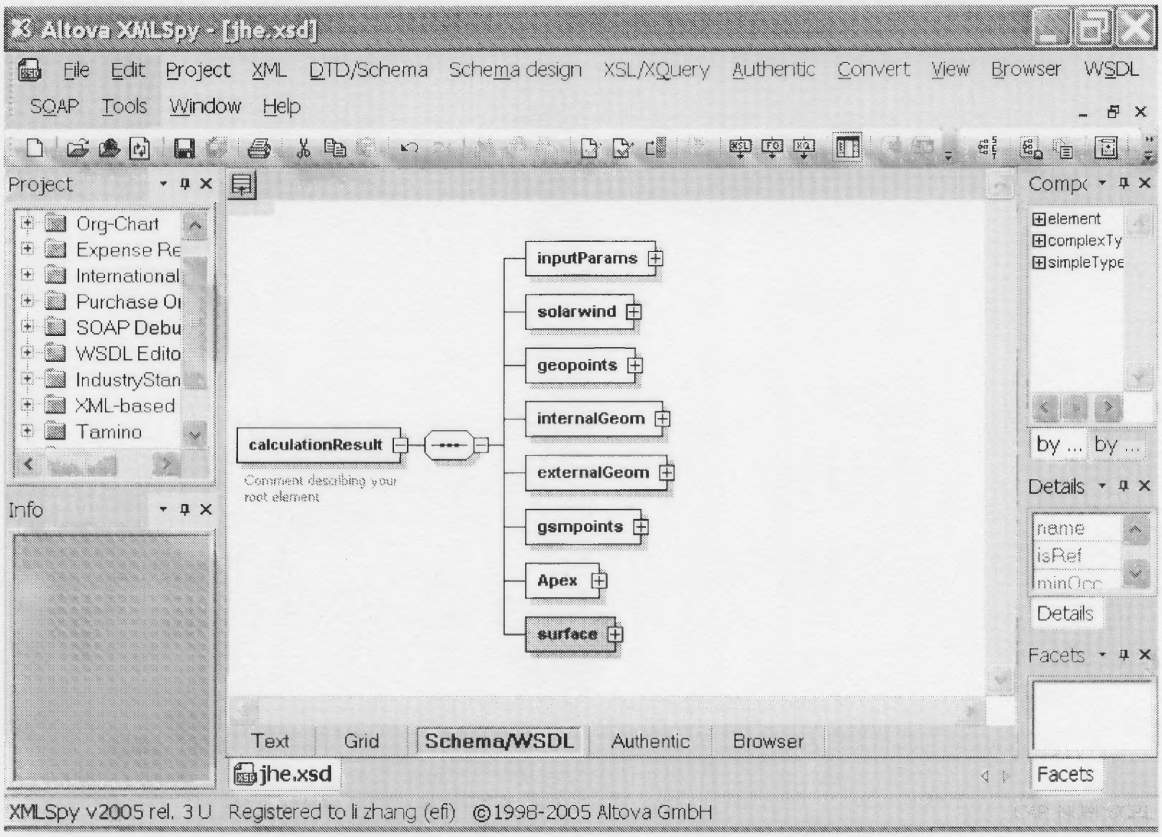

Figure 6.8 A Document schema for T96 model.

6.2.5 Regeneration Engine (RE)

The Regeneration Engine is a Java class that performs the following steps to regenerate a virtual document:

(1) Get query command and parameters from database by the unique document identifier;

(2) Send a request to the AW and then get document back;

(3) Validate the virtual document by different criteria.

RE receives the source document, then revalidates for different criteria.

Criteria 0: Exact copy. This means it needs to do byte comparison between the new document and the stored document.

Criteria 1: Values can change. This means it needs to compare the document structure.

Criteria 2: All related queries. This means do not need to compare with history information.

When the user creates a bookmark, if the "Criteria" is "0", then the document will be stored in the JHE server. The "content" value in the bookmark record is the file name. The file is stored in the same directory that the JHE server locates. In Figure 6.9, the variable "document" in the function is the regenerated document. If the "Criteria" is "0", RE calls the function compareVDoc. In the function "compareVDoc", it retrieves the file name from database, and reads the original document from the specific directory that stores documents. Then it compares the original document with the new document byte by byte.

If the "Criteria" is "1", RE calls the function "parseVDoc". In the function "parseVDoc", RE calls the AW to parses the regenerated document according to a stored document template. For example, in the T96 calculation result, the word "year=2005" indicates this is a "year" element which has the value "2005". This document follows a specific document template which defines the names and orders of the keywords and their values. For example, the "day" keyword follows the "year" keyword with some values. The order is important: if it changes, this means the document structure changes. The parsing process will fail. If the "Criteria" is "2", then RE does not compare anything.

```
public boolean validateDoc(Document document, String QueryDocID, String criteria)
{
        if(criteria.equalsIgnoreCase("0"))              // exactly same
        {
                return (compareVDoc(document, QueryDocID));
        }
        else if(criteria.equalsIgnoreCase("1"))         // value can change
        {
                return (parseVDoc(document, QueryDocID));
        }
        else                                            // do not compare anything
        {
                return true;
        }
        return true;
```

Figure 6.9 Revalidation procedure.

If revalidation is successful, then the Document Manager will do relocation and re-identification for all related anchors for the virtual documents, otherwise it will give warning messages to users.

6.2.6 Selection Manager

When a user selects some texts from screen, then clicks a button, the Selection Manager will generate an XPointer expression and then sends it to the Gateway. The Selection Manager is not a Java class. Besides, JHE users need to install the Mozilla (http://mozilla.org) browser and a certificate in order to use the Selection Manager. The implementation has following steps:

(a) Enable XPointer Lib service

The Selection Manager is implemented by Mozilla's XPointer Lib (http://xpointerlib.mozdev.org/). XPointer Lib is one of Mozilla's XPCOM (http://www.mozilla.org/projects/xpcom/) libraries. XPointer Lib generates XPointer expressions when users select some texts from the screen. To enable XPointer Lib's

service, the JHE developer needs to download and install the XPointer Lib package first.

The installation has following steps:

(1) Open Mozilla web browser.

(2) Go to http://xpointerlib.mozdev.org/installation.html.

(3) Click "installation" button on the above web page.

(4) Install the XPCom viewer from http://www.hacksrus.com/~ginda/cview/.

(5) Click Mozila's menu: "Tools" -> "Web Development" -> "Component Viewer".

(6) If the XPointer Lib service is shown in the left column of the viewer, then the XPointer Lib is installed correctly. Figure 6.10 shows the XPCom Component Viewer.

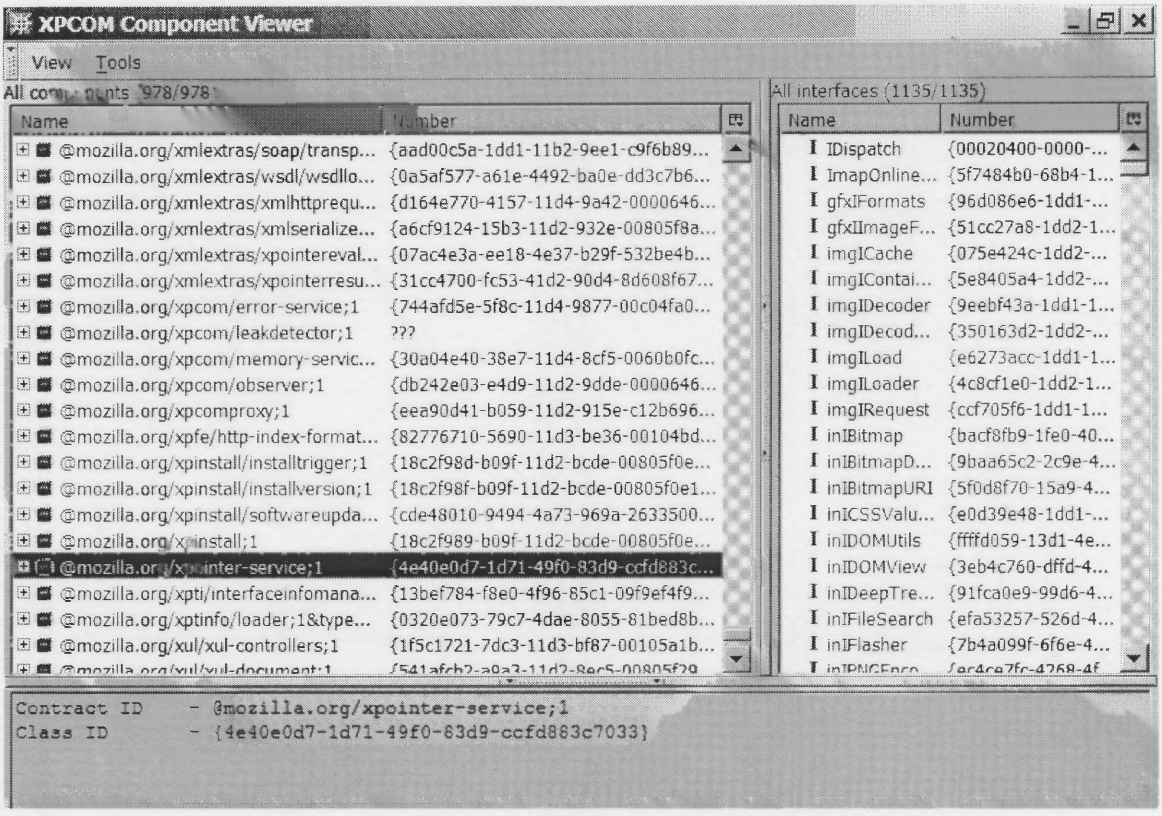

Figure 6.10 XPCom component viewer.

(b) Add a function to get XPointer expression from the desired HTML page

Then the JHE developer adds the following Javascript function to an HTML file where JHE wants to select some texts and gets XPointer expressions:

```
function getXptr()
{
netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
var xptrService = Components.classes["@mozilla.org/xpointer-service;1"].getService();
xptrService = xptrService.QueryInterface(Components.interfaces.nsIXPointerService);
var xptrString =
xptrService.createXPointerFromSelection(window._content.getSelection(),
window._content.document);
return xptrString;
}
```

The UIW automatically adds the above Javascript function for every document displayed on the screen.

(c) Install the Mozilla certificate

JHE users should install a Mozilla certificate on the Mozilla browser. JHE users can not use other Web browsers such as Internet Explorer or Safari to access JHE, because JHE uses Mozilla's XPointer Lib service which is not available in other browsers. The certificate is an authority file from the Web server to allow users to use some functions that it supports. The JHE developer or the Web administrator creates a certificate, then JHE users install it on their own computers which have the Mozilla browser. This certificate is not part of the JHE module, thus it's not a Java class or some Javascript functions. It is a file that contains digital signatures and encrption information.

Creating a certificate has following steps:

(1) Download and install NSS (Netscape Security Service) 3.7.8 from http://ftp2.de.freebsd.org/pub/misc2/mozilla/security/nss/releases/NSS_3_7_8_RTM.

(2) Create and distribute a certificate, see http://books.mozdev.org/html/mozilla-chp-12-sect-5.html for details. Actually there are two certificates: the root certificate and the distribute certificate. The JHE administrator creates these two certificates, then installs the root certificate on the JHE server. He also sends the distribute certificate to JHE users. Then every JHE user installs the distribute certificate on his Mozilla browser.

(3) If a user or the administrator wants to check whether the Mozilla browser has the certificate from JHE, he opens the Mozilla browser and selects Mozilla browser's menu: "Edit" -> "Preferences" -> "Privacy & Security" -> "Certificates", clicks "Manage Certificate". Then the "Certificate Manager" window shows a new certificate is installed. Figure 6.11 shows the JHE Mozilla certificate (the highlighted line) is installed. Now, the JHE engine can use the XPointer Lib service.

(d) Send XPointer expressions to JHE gateway

This is done by using HTML's form capability. When a user clicks the "submit" button, the browser will send an HTTP "POST" request together with some parameters (such as selected text content, location expressions and the virtual document ID) to the destination module – the Gateway.

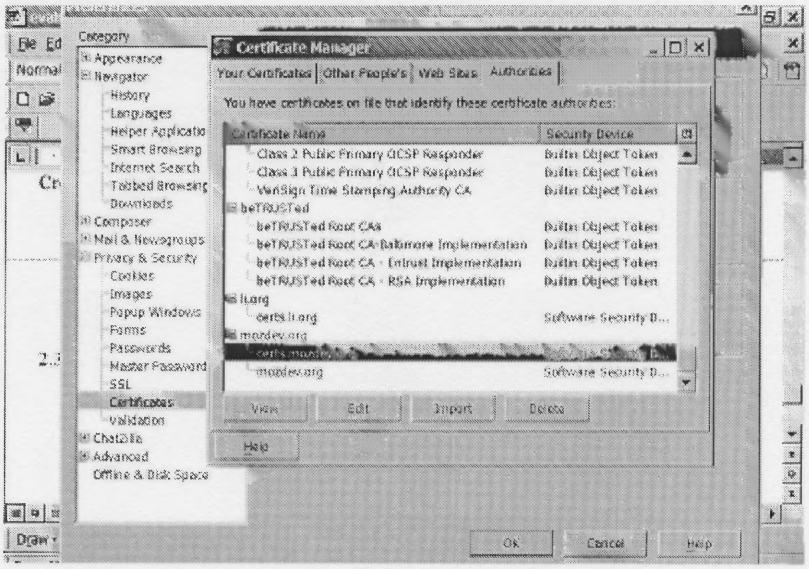

Figure 6.11 Mozilla certificate manager.

6.2.7 Document Manager

The Document Manager is a Java class that gets all related anchors from the JHE database, then relocates and re-identifies anchors according to different criteria. The Document Manager performs the following steps (see Figure 6.12):

(1) Get an anchor list from database, this includes all global anchors and other local or specific anchors that has the same document identifier with this virtual document.

(2) Evaluate each of these anchors according to the criteria. First the Document Manager finds the anchor location according to document structure, then it compares the element value with the history information if the criterion does not allow element value to change; it does not compare the element value if the criterion allows element value to change. The Document Manager uses Apache's XPath tool (which is a Java class package that evaluates XPath expressions in a JDOM document) to find the anchor locations. The highlighted codes in Figure 6.13 show how the XPointer expression is evaluated and the element value is retrieved from the document. Then the function compares the original value with the new value.

(3) The Document Manager finds all related hypermedia constructs for each anchor (comment, links, etc.), then it attach hypermedia objects to this document by inserting icons at the side of the elements.

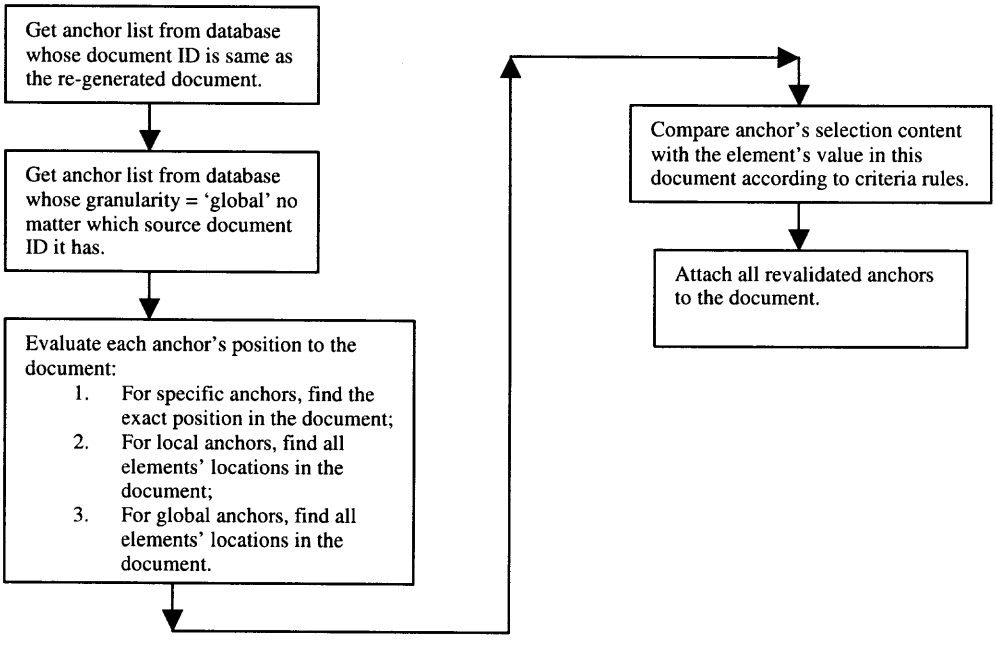

Figure 6.12 Flowchart for anchor relocation and re-identification.

```
private boolean evaluateElementValue(String xpointer, String origValue, Object doc)
{
        try
        {
        List results = new ArrayList();
        if ( doc instanceof org.jdom.Document)
        {
                XP[illegible]
                resu[illegible]path.selectNodes((org.[illegible]ument) doc);
        }
                Iterator rItx = results.iterator();
                int rsize = results.size();
        while (rItx.hasNext())
        {
                org.jdom.Element elem = (org.jdom.Element) rItx.next();
                String newValue = elem.getText();
                if(newValue.equalsIgnoreCase(oldValue))
                        return true;
        }
```

Figure 6.13 Evaluate an XPointer expression.

6.2.8 User Interface Module (UIW)

After a user logged in JHE, it will take him to the user interface (shown as Figure 6.14). The user interface is composed of four parts: application list, main window, menu list, and menu window. The upper left screen shows the application list that JHE supports. Currently JHE only supports the NSSDC application, but will support more applications in the future. The upper right screen shows the virtual document from applications. The lower left screen shows the hypermedia functionality that JHE supports, this include bookmark list, add new bookmark, add new comment, and add new link menu. The lower right screen shows the menu content, which allows users to enter some information or select some information from menu. There is no menu content in Figure 6.14 because the user does not select any menu after he loges in JHE. Figure 6.15 shows the Bookmark Service Menu content at the lower right corner of the window. The UIW contains

Bookmark Service Menu, Link Service Menu and Comment Service Menu. Each service menu is an HTML page containing some input parameters and JavaScript functions.

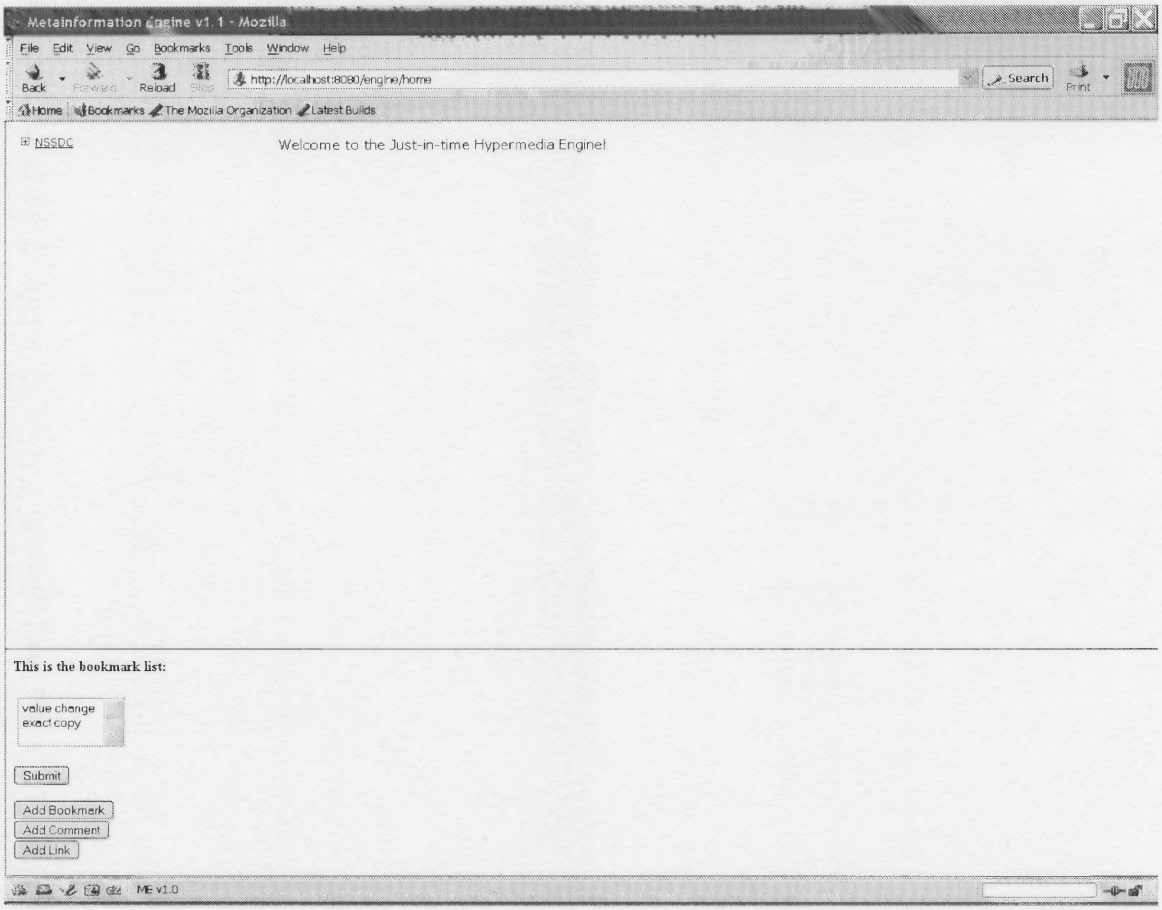

Figure 6.14 JHE main menu.

- **Bookmark Service Menu**

When a user wants to add a bookmark, he clicks the "add bookmark" button on the screen, then the bookmark service menu will appear at the right corner of the window, shown as Figure 6.15. The Bookmark Service Menu is an HTML page with some JavaScript functions. The HTML page has a "form" to allow users to input parameters from screen, such as the bookmark name and revalidation criteria.

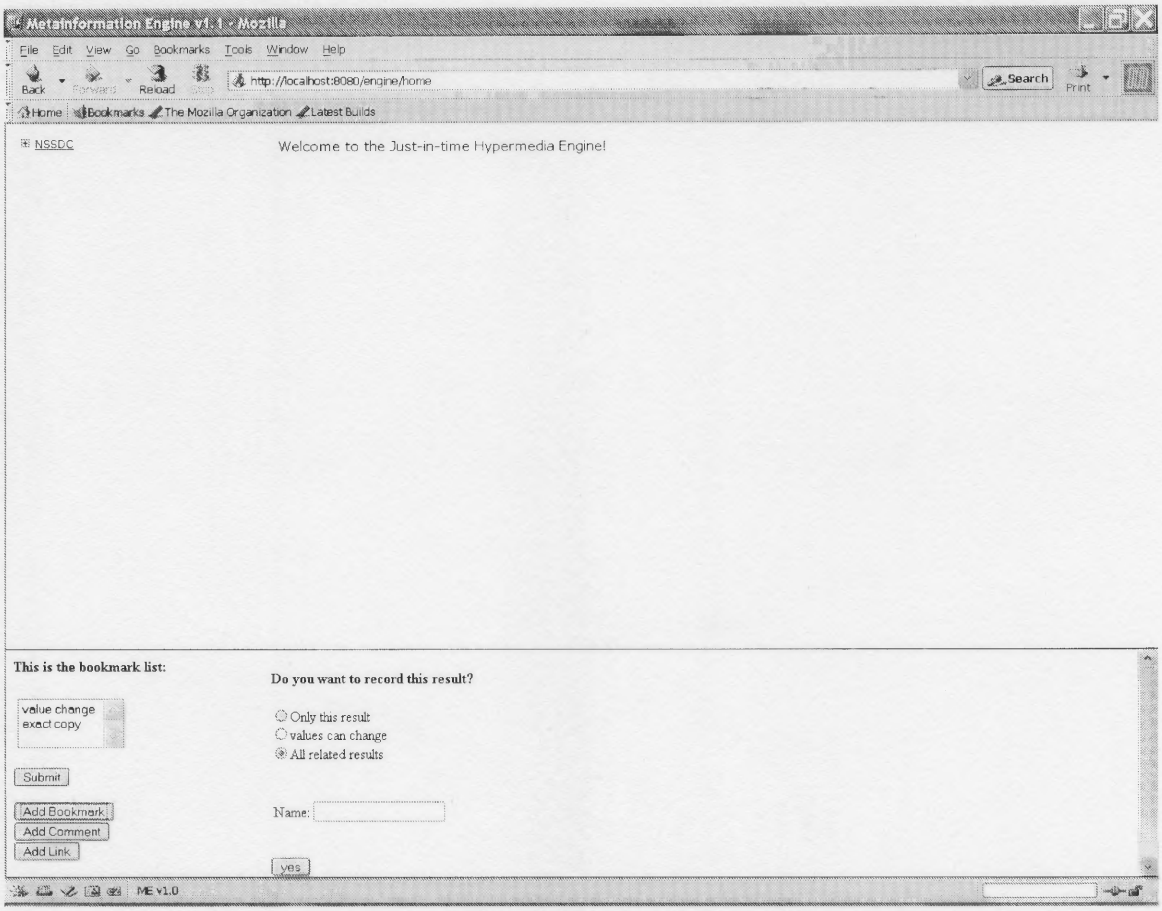

Figure 6.15 Bookmark service menu.

- **Link Service Menu**

When a user wants to add a link, he clicks the "add link" button on the screen, then the link service menu will appear at the right corner of the window, shown as Figure 6.16. The Link Service Menu is an HTML page with some JavaScript functions. The HTML page has a "form" to allow users to input parameters from screen, such as the link name and link destination. When the user clicks the "submit" button, UIW packs the information and sends it to Gateway. A link destination could be a URL entered by the user, an anchor or a bookmark. When UIW composes the Link Service Menu, it retrieves all anchors and bookmarks and lists them in the menu. If a user wants to add a link to a

bookmark or an anchor that has not existed, he should create it first and then makes a link to it.

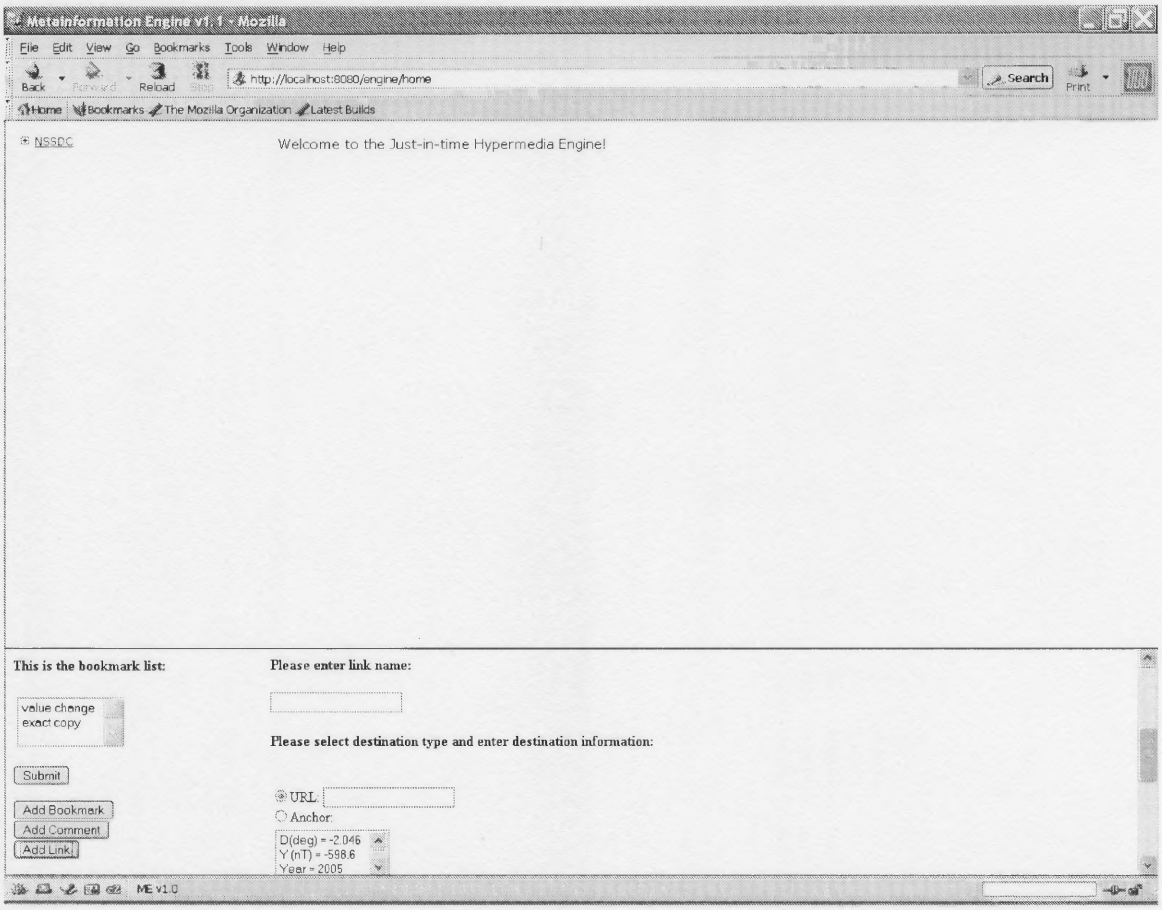

Figure 6.16 Link service menu.

- **Comment Service Menu**

When a user wants to add a comment, he clicks the "add comment" button on the screen, and then the comment service menu will appear at the right corner of the window, shown as Figure 6.17. The Comment Service Menu is an HTML page with some JavaScript functions. The HTML page has a "form" to allow users to input parameters from screen, such as the comment name and content.

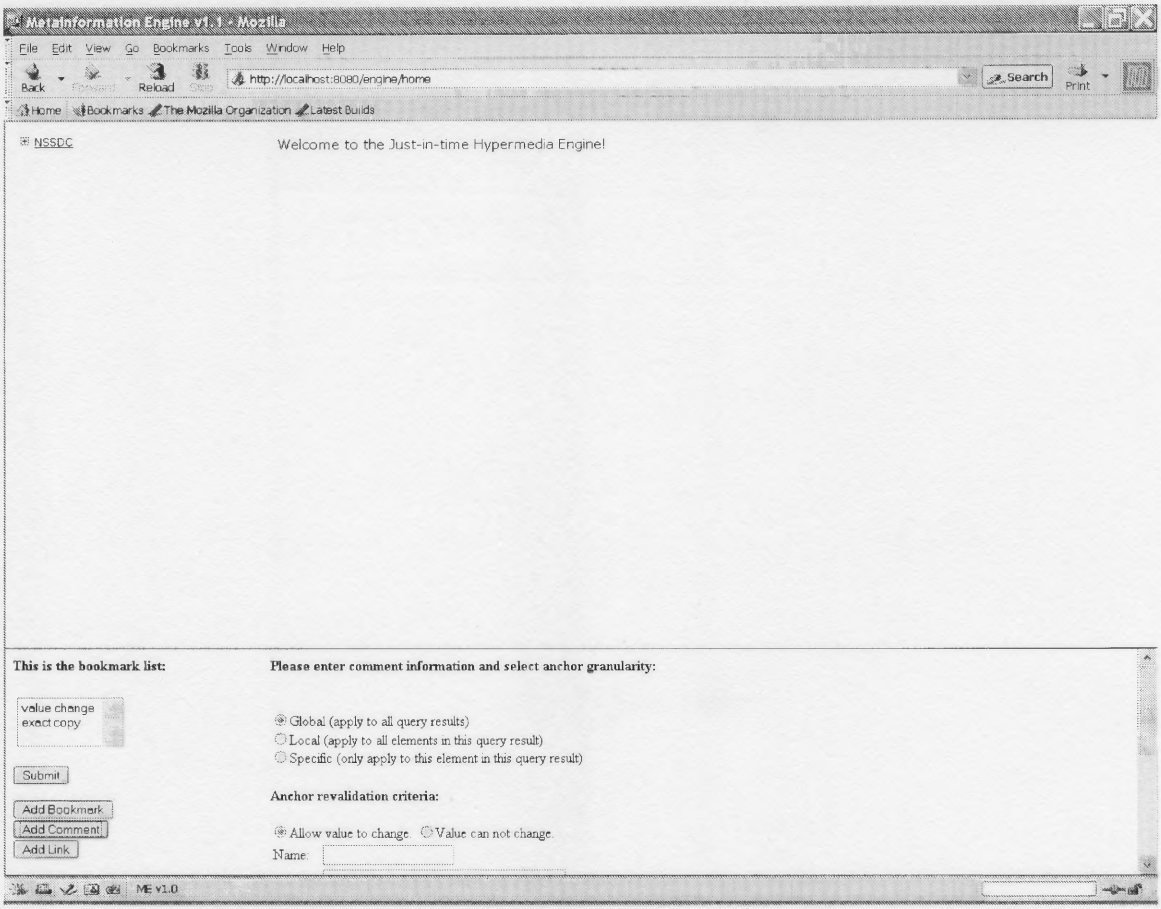

Figure 6.17 Comment service menu.

6.2.9 Document Translator

The Document Translator transforms the XML document into a displayable HTML format. In JHE, the HTML file also includes displayable parts (source document from application) and non-displayable parts. The non-displayable parts include system data (e.g. document identifiers and bookmark information list) and script functions. The script functions implement some of user interface functions. The Document Translator is a Java class running on the JHE server.

6.2.10 Tools Used In The JHE Project

The JHE project is a Web based project which uses the following tools:

(1) Java and Javascript: (http://java.sun.com/). Java is a platform independent language, which is widely used in Web applications. Java is usually programmed to run on server side. Javascript is a script language running on client side.

(2) JDOM.(http://www.jdom.org/docs/apidocs/). JDOM is a Java API that can access, manipulate, and output XML data.

(3) Mozilla and related utilities (http://www.mozilla.org). Mozilla is an open source Internet client suite designed for standards compliance, speed and portability. Mozilla is not only a Web browser, but also provides much functionality for Web applications.

(4) Apache Tomcat Web server (http://jakarta.apache.org/tomcat/). Apache Tomcat is an open source Web server, which supports Java Servlet (http://java.sun.com/products/servlet/index.jsp) and Java ServerPage (http://java.sun.com/products/jsp/) technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process (http://java.sun.com/aboutJava/communityprocess/).

CHAPTER 7

RESULTS AND EVALUATIONS

This chapter gives some test results and then does evaluation based on the just-in-time hypermedia framework described in Chapter 3. Section 7.1 describes dynamic regeneration results and evaluation. Section 7.2 describes dynamic hypermedia functionality results and evaluation. Section 7.3 describes dynamic relocation and re-identification results and evaluation. Section 7.4 compares JHE with other similar systems. Section 7.5 discusses system extensibility. Finally, Section 7.6 describes WWW standards used in JHE.

7.1 Dynamic Regeneration

JHE integrates application modules into the hypermedia system, and then gets the user commands from UI. It sends the user requests to the destination application for execution and receives the source documents. When a user revisits a bookmark, JHE will revalidate the generation results according to different criteria. Section 7.1.1 gives some test results on dynamic regeneration. Section 7.1.2 evaluates the test results.

7.1.1 JHE Regeneration Results

To get a regeneration result, the user creates a bookmark then revisits it later.

7.1.1.1 Create a Bookmark. Figure 7.1 shows the T96 application module is integrated into JHE. When user selects the "T96 Model" command (which is read from the system setting file when JHE starts up) from the UI menu, it gives users the T96 homepage, which has the query table. This allows users to enter in parameters. Before the

user creates a new bookmark, he needs to enter and submit parameters to JHE to create a new calculation result. After he adds a new bookmark for the result, he will not need to enter parameters again the next time. JHE will do regeneration automatically.

Figure 7.2 shows the calculation result after the user enters some parameters. When the user selects the "add bookmark" command from the menu at the right corner of the window, the Bookmark Service Menu will prompt the user to enter in bookmark information, such as name ("exact copy" in the example) and criteria ("Only this query result"). Then after the user clicks "submit", a new bookmark named "exact copy" is added into database.

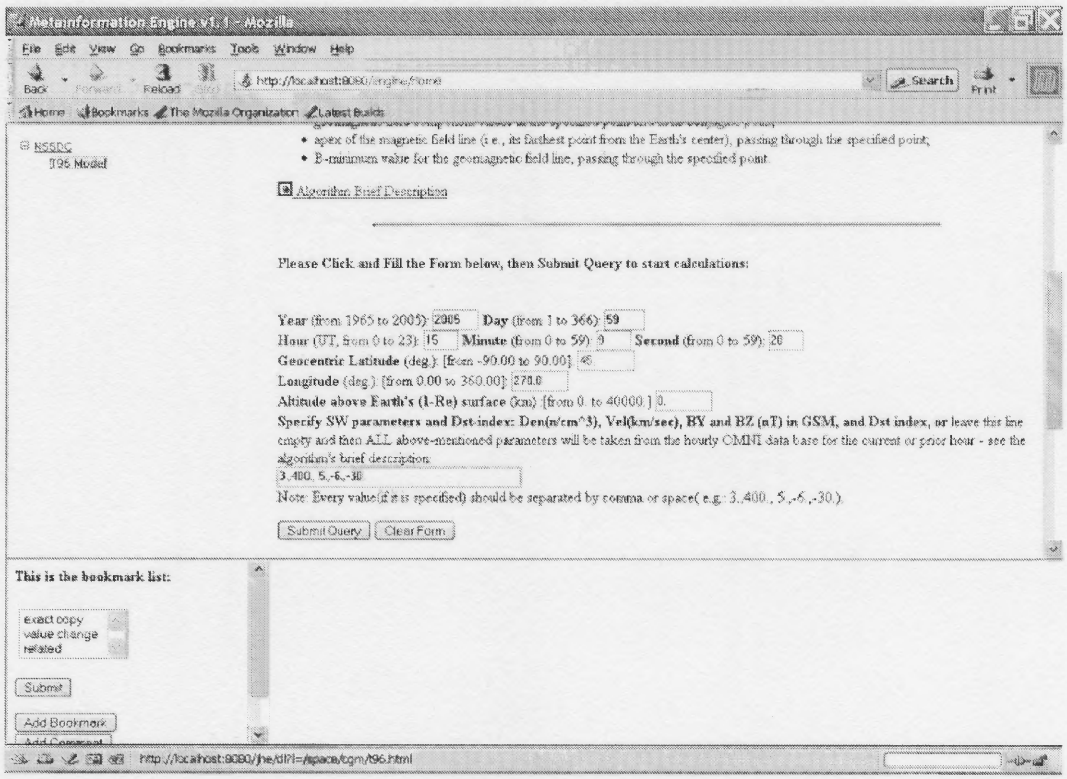

Figure 7.1 T96 home page integrated into JHE.

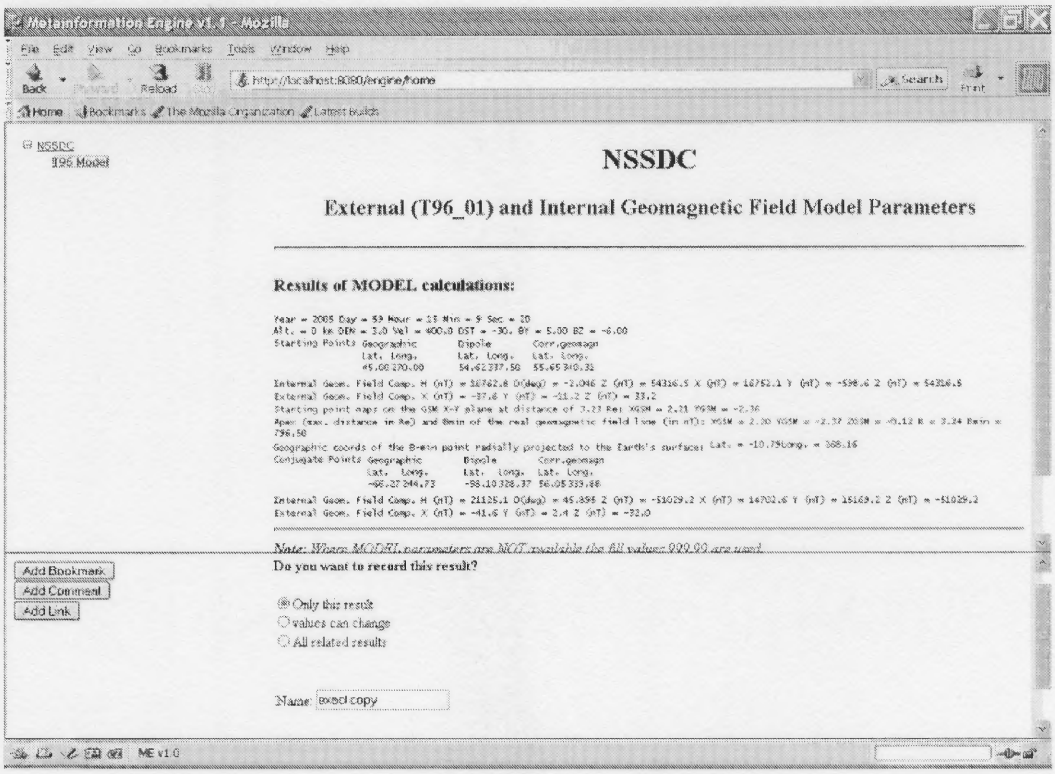

Figure 7.2 Add a bookmark for a calculation result.

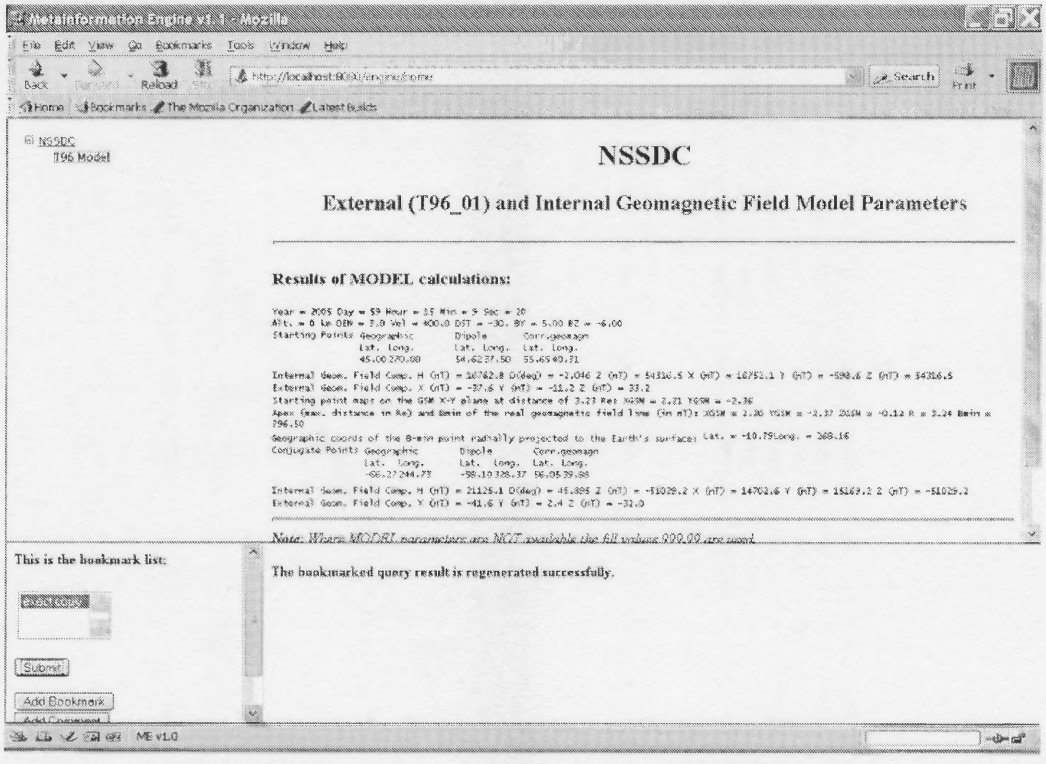

Figure 7.3 Regenerated document with criterion 0.

7.1.1.2 Revisit the bookmark and get the regenerated result. Next time when the user revisits the bookmark, he can select the bookmark from the list (in left corner of the window). The JHE would then regenerate the result without asking the user to reenter parameters. Figure 7.3 shows the regenerated document and the message shows the regeneration is successful.

7.1.1.3 Revalidate Virtual Documents with Different Criteria. According to Chapter 3, there are three different levels of re-validation criteria. Level 0 requires the regenerated document to be exactly same as the original one. Figure 7.2 shows the original virtual document. Figure 7.3 shows the regenerated virtual document. Comparing these two documents, they would be found to be exactly the same. Level 1 allows the elements' values to change. Figure 7.4 shows the original virtual document. Figure 7.5 shows the regenerated virtual document. There are some values changed, such as the element "year". Its original value is "2000", but the current value is "2005". According to Level 1 criterion, it is still a valid regeneration. Level 2 is very flexible; it allows most elements' values to change. Also it allows the document structure to change. Figures 7.6 to 7.7 show the original virtual document and the regenerated virtual document. Many elements' values have changed, but it's still valid regeneration. Criterion 2 allows the document structure to change; it gives more flexibility than criterion 1.

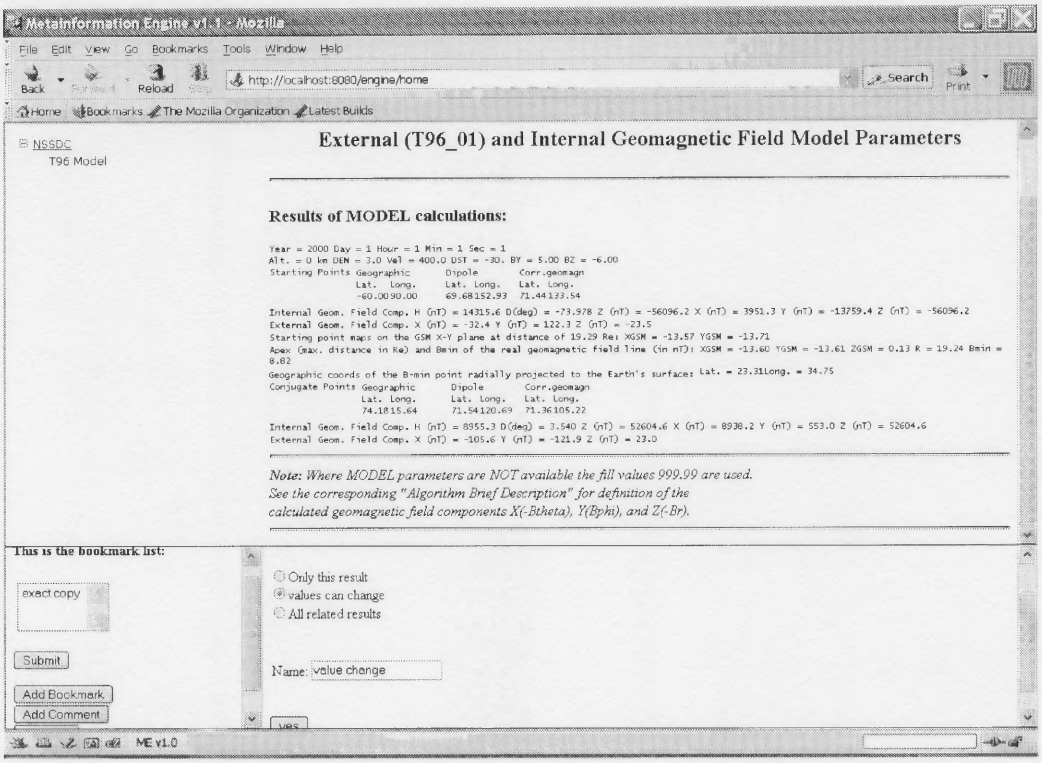

Figure 7.4 A bookmark with criterion 1.

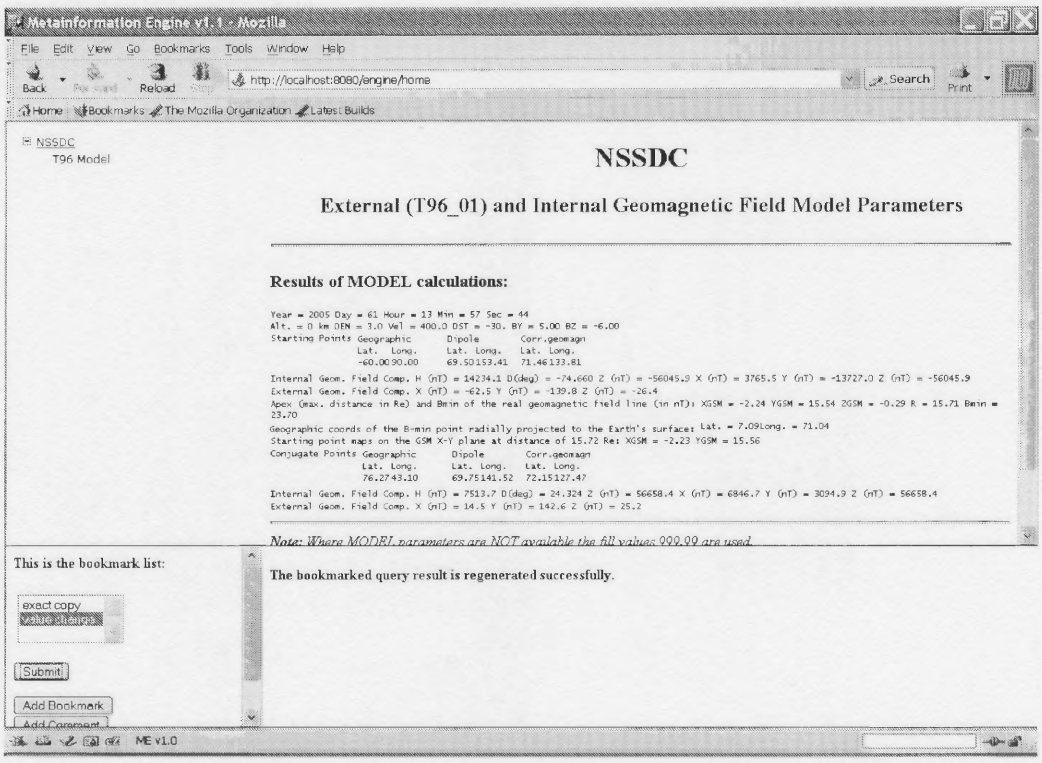

Figure 7.5 Regenerated virtual document with criterion 1.

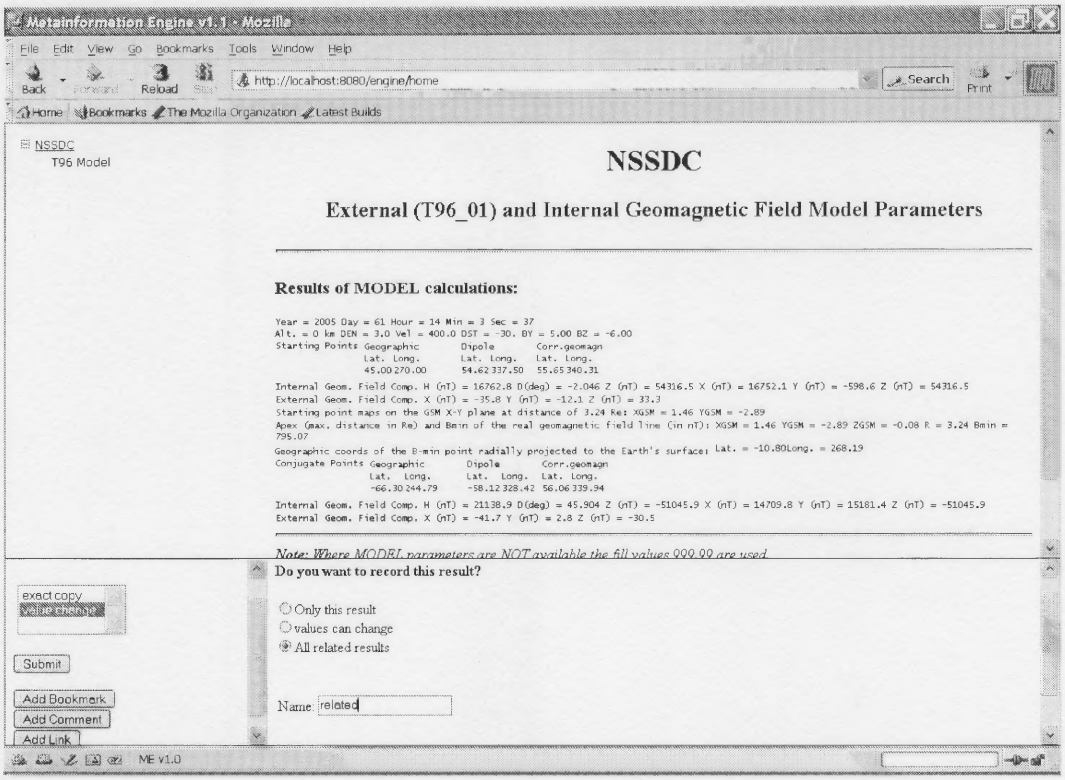

Figure 7.6 A bookmark with criterion 2.

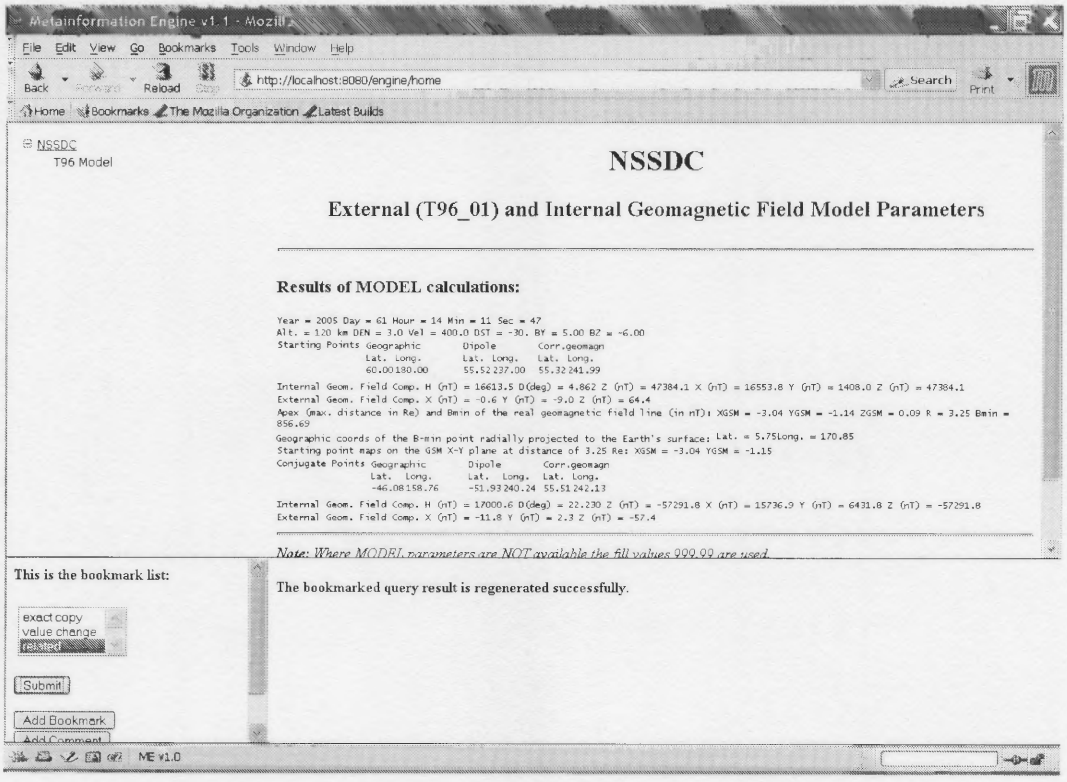

Figure 7.7 Regenerated virtual document with criterion 2.

7.1.2 JHE Regeneration Evaluation

When JHE does dynamic regeneration, RE calls AW to generate the virtual document and then RE does revalidation. This section evaluates the generation and the revalidation.

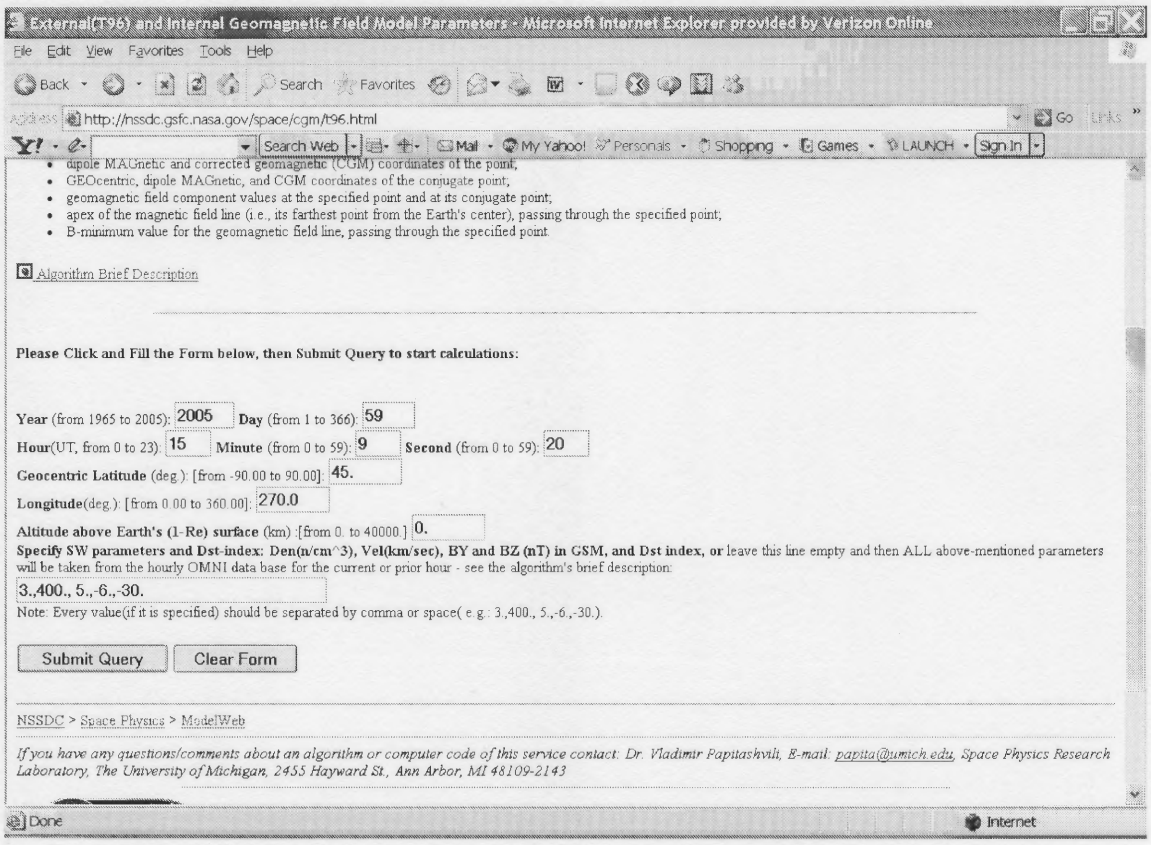

Figure 7.8 T96 home page.

(1) Compare the JHE generated virtual documents with the original virtual documents generated by the application. JHE integrates applications into the hypermedia system and it intercepts the source documents from applications. Since the original virtual document is translated into a well-structured document, it is important that data from application be the same as the original. For example, the calculation result should be exactly the same as the original although the format may be a little different. Also, users are expecting to get a similar layout document, such as font size, line width, and background. JHE translates the original document into an XML document, and then generates some style file for this document to have a similar layout as the original. Figure 7.1 shows the T96 model home page that has been integrated into JHE. Figure 7.8 shows the original T96 model home page. Comparing the two figures, they are the same.

The integration does not affect the application's outlook, although the "submitted" query will go to JHE gateway first, and then redirected to the destination application. Figure 7.3 shows the regenerated virtual document. Figure 7.9 shows the calculation result at the application. Comparing these two documents, the contents are exactly same. However, the layouts have some small differences. These are due to the limitations of current stylesheet languages. When transforming between different styles, they do not work perfectly.

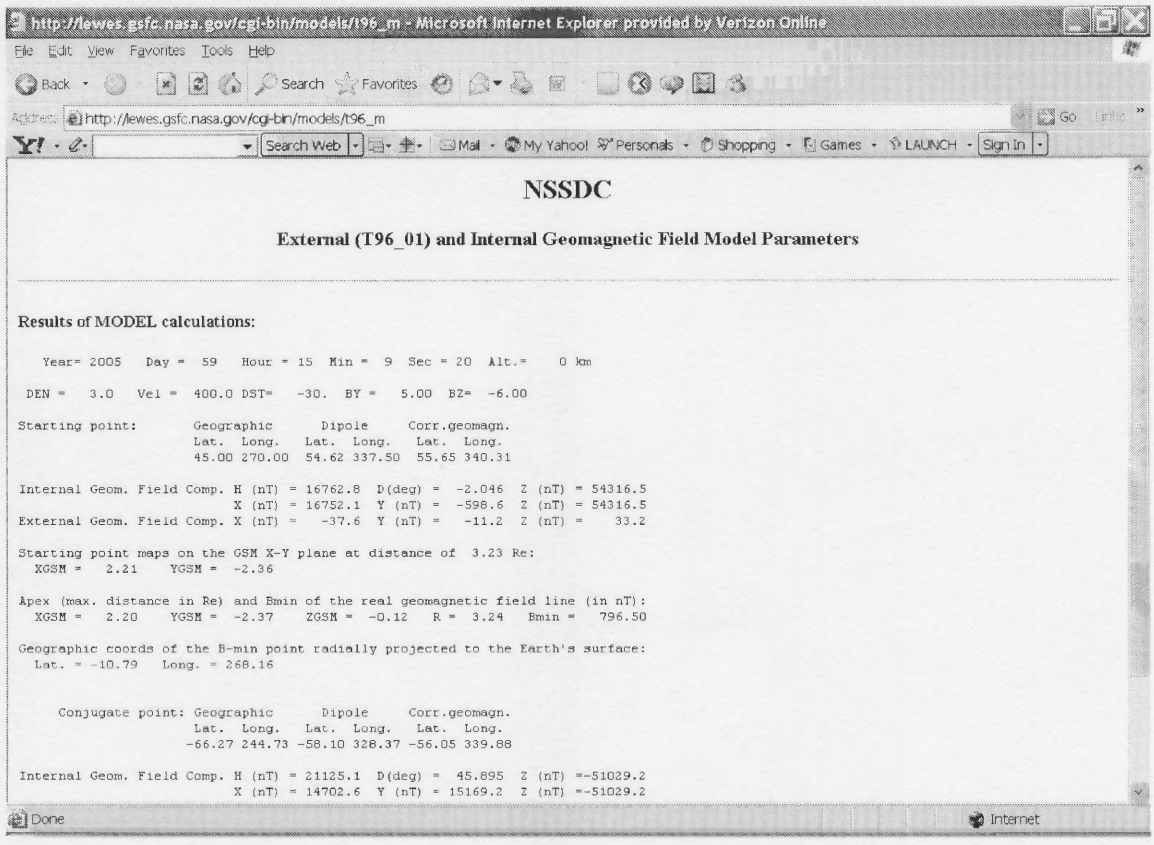

Figure 7.9 T96 calculation result.

(2) When doing regeneration, there is a question whether the re-generated document is the same as the original. Depending on the dynamics of the application systems, a virtual document could change frequently or rarely. The revalidation process gives users some information about the comparisons between the newly generated document and the original. The three levels of revalidation give users flexibility to perform hypermedia functionality on a virtual document. If the revalidation is not successful, this does not mean the virtual document is wrong. It just gives some warning message to the user that the newly-generated virtual document has changed according to the user's criterion. It may remind the user that the data is out-of-date, and he may need some changes. Unlike a traditional bookmark, which just remembers the address of the document, it does not have any history

information about it. By doing this kind of revalidation, it can help the user to compare the regenerated virtual document with the original and remind him to do some changes, which is really better than traditional bookmark.

7.2 Dynamic Hypermedia Functionality

JHE provides bookmark, user declared comment and user declared link functionalities for the virtual documents. *Add bookmark* functionality is described in Section 7.1. Section 7.2.1 describes *user declared comment* and Section 7.2.2 describes *manual link.*

7.2.1 User Declared Comment

(1) Add a User Declared Comment

Creating a user declared comment has following steps:

- Click the "add comment" button
- Select some texts from screen
- Specify granularity
- Edit comment content
- Edit comment title
- Click "submit" button

Figure 7.10 shows how to create a comment from UI.

(2) Comment Results

After the user clicks "submit" button, JHE stores comment information into database. Next time, when the user revisits the virtual document, JHE finds marked anchors in the document and re-identifies the anchors according to different criteria, and then inserts an icon at the side of the element, show in Figure 7.11. In Figure 7.11, the anchor "year=2005" is relocated and re-identified after regeneration. When the user clicks on the (H) icon, it pops up a window shows a list of hypermedia constructs, such as comments and links, show in Figure 7.12 (there are two comments and one link). When the user clicks on the comment title in Figure 7.12, the window will display corresponding comment contents, shown in Figure 7.13.

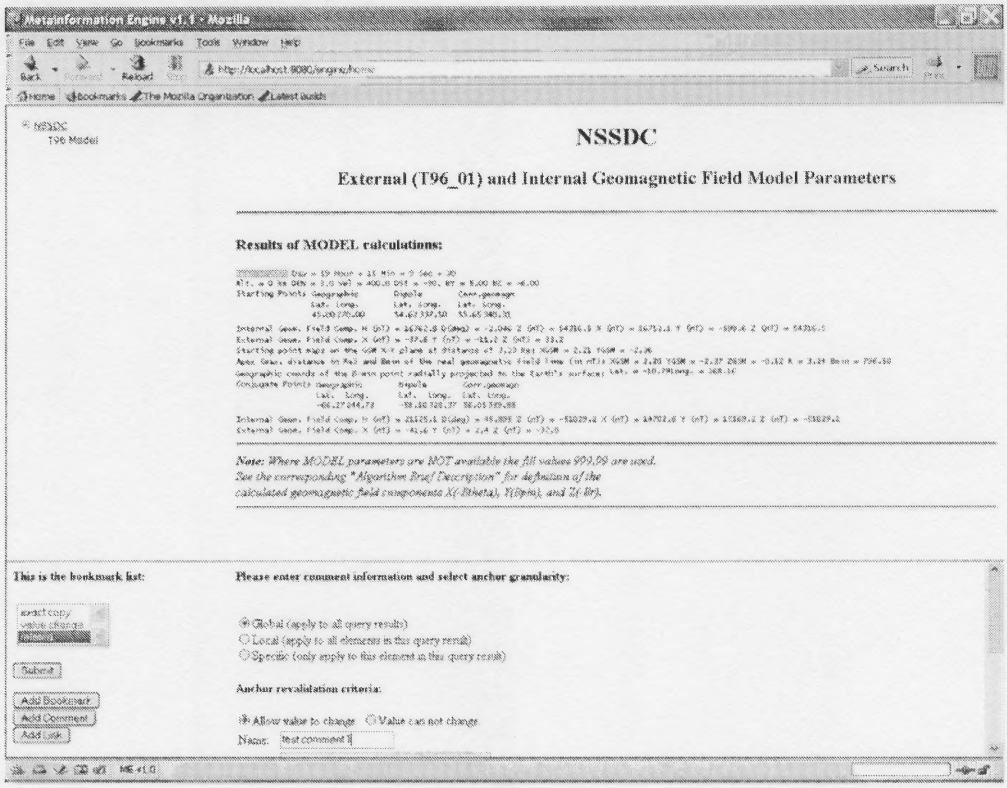

Figure 7.10 Create a comment.

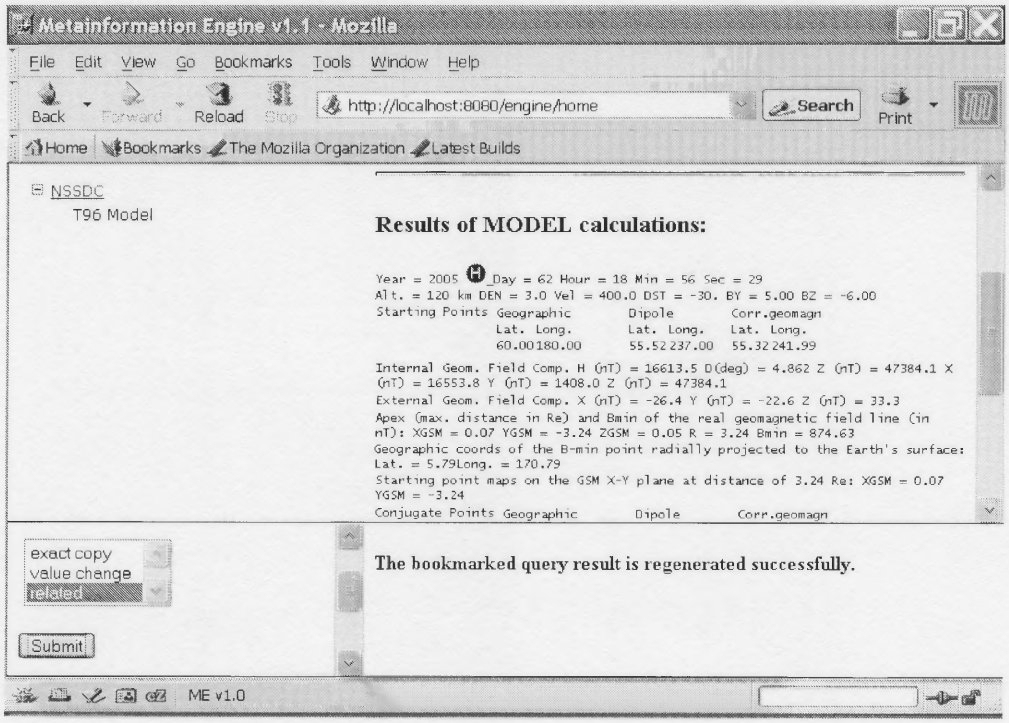

Figure 7.11 Regenerated virtual document with hypermedia constructs attached.

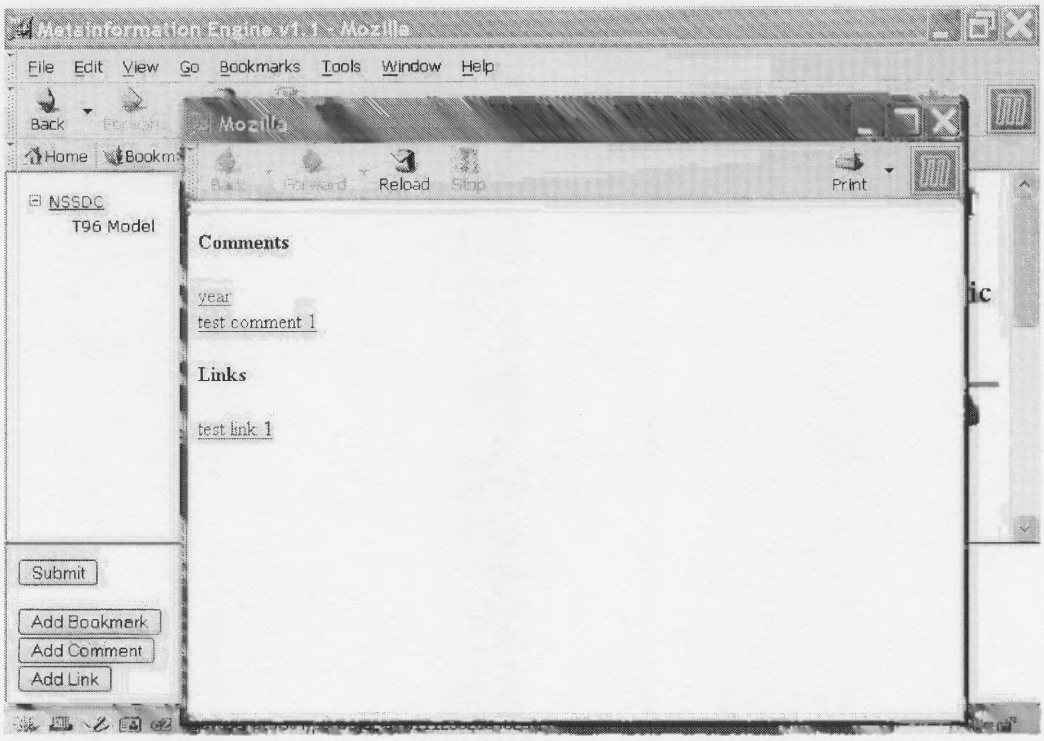

Figure 7.12 List of comments and links.

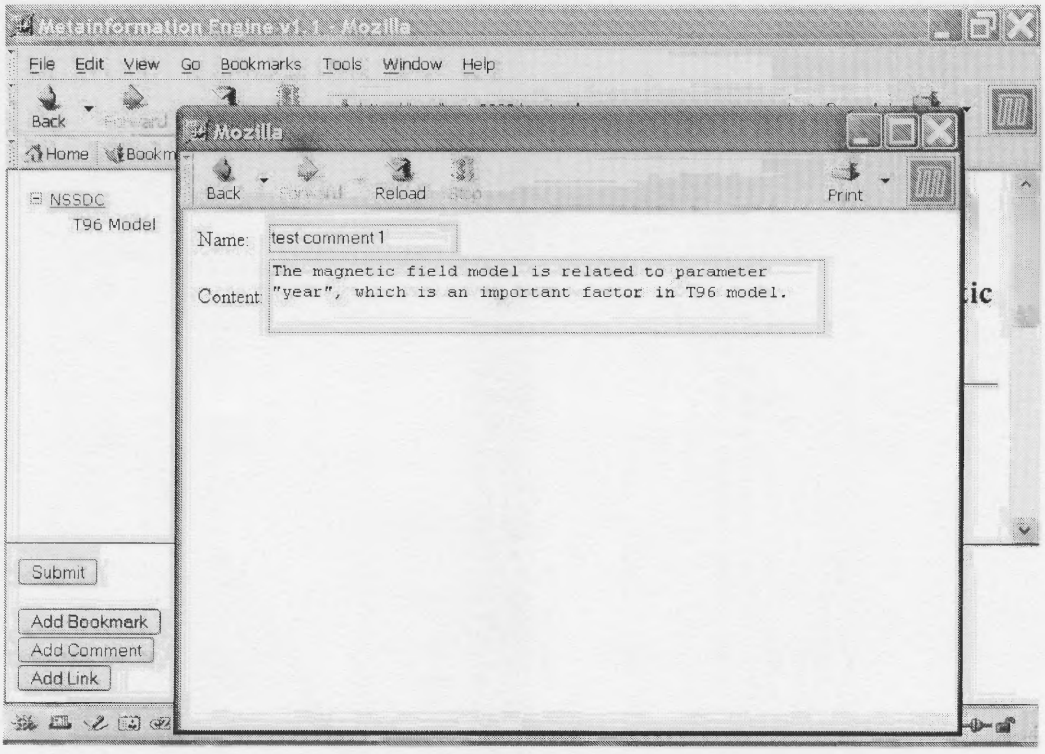

Figure 7.13 Displaying the comment contents.

(3) Evaluation

Figures 7.11 to 7.13 show that adding a new user declared comment is quite easy for users to use. The JHE system can store and get the comment information from database. Also, the user interface to add comment and display comment information is quite good.

7.2.2 Manual Link

A user selects some texts from screen then enters parameters in the Link Service Menu and submits the request to JHE. JHE adds a link into database. Next time the user clicks the link title in the hypermedia constructs window to traverse a link.

7.2.2.1 Add a Manual Link. Creating a manual link has following steps:

- Click the "add link" button;
- Select some texts from screen;
- Select destination;
- Specify granularity;
- Edit link title;
- Click "submit" button.

If the type is "URL", he enters URL in the menu; if the type is an anchor, he selects an anchor from the anchor list; if the type is a bookmark, he selects a bookmark from the bookmark list. The anchors and bookmarks are preexisted before the user makes a link to them. If the anchor or the bookmark does not exist, then the user should create it before adding a link to it. Figure 7.14 shows how to create a link from UI.

7.2.2.2 Link Results. Figures 7.14 to 7.16 show the link results whose destination is an external URL. Figures 7.17 to 7.19 show the link results whose destination is an anchor inside a virtual document. Figures 7.20 to 7.22 show the link results whose destination is a virtual document. There are three types of link destinations: a URL, an

anchor, and a virtual document in JHE. An anchor is a selected anchor in a JHE supported virtual document. A virtual document is a JHE supported virtual document that can be generated by JHE integrated applications. Depending on different types of link destination, the link traversal is different. For a link that has a specific URL, clicking on the link will take the user to another Web page. For an anchor, clicking on the link will take the user to the virtual document which JHE regenerates, and the destination anchor is highlighted. For a virtual document, clicking on the link will take the user to the virtual document which requires JHE to do regeneration. Also, for the destination document, hypermedia objects are attached on it.

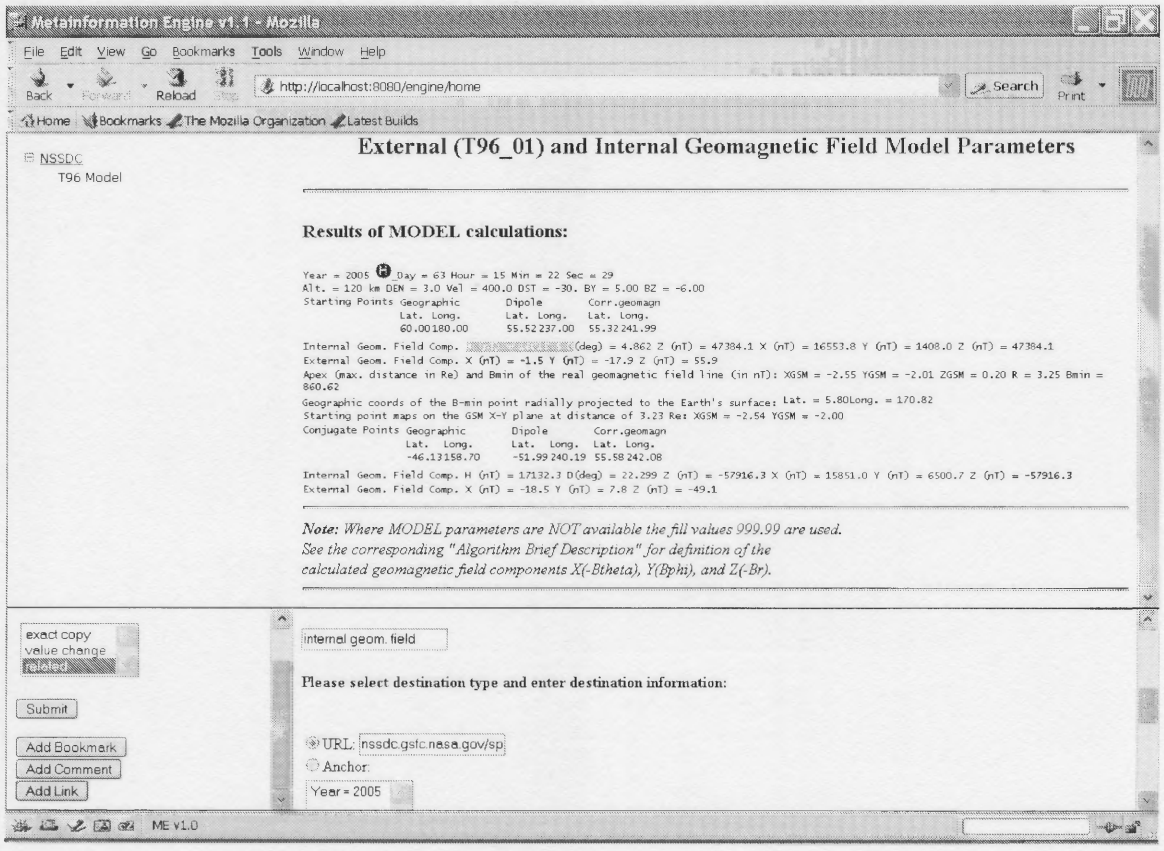

Figure 7.14 Add a new link whose destination is a URL.

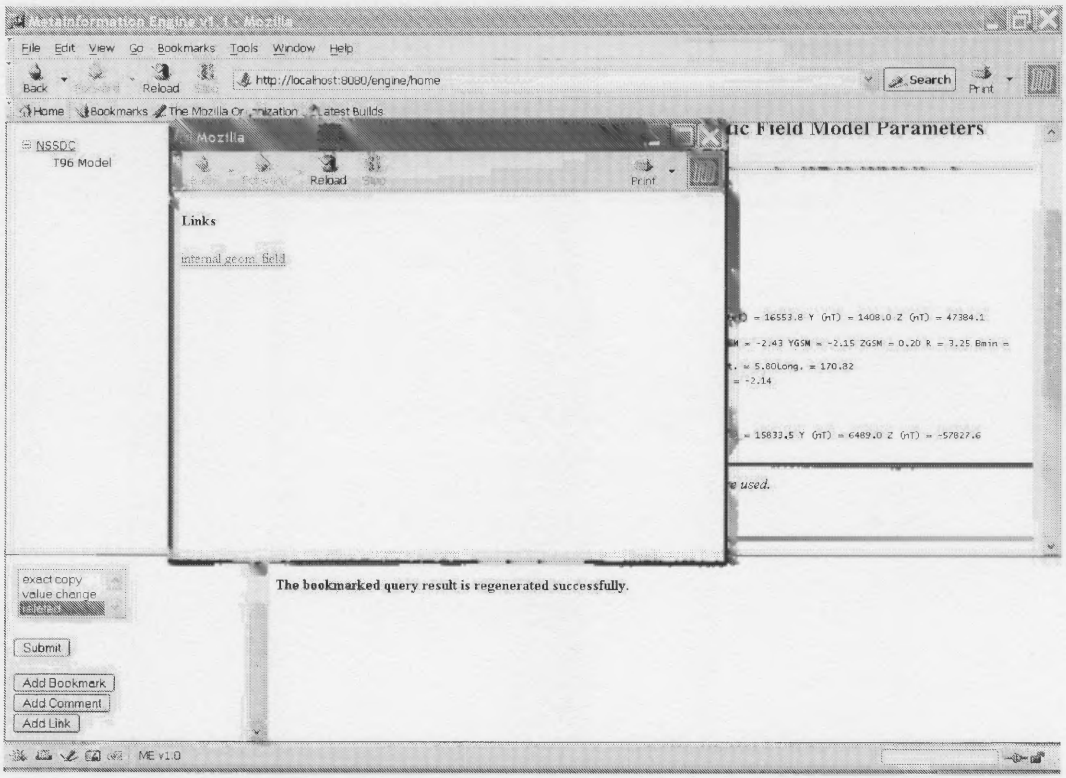

Figure 7.15 List of link information.

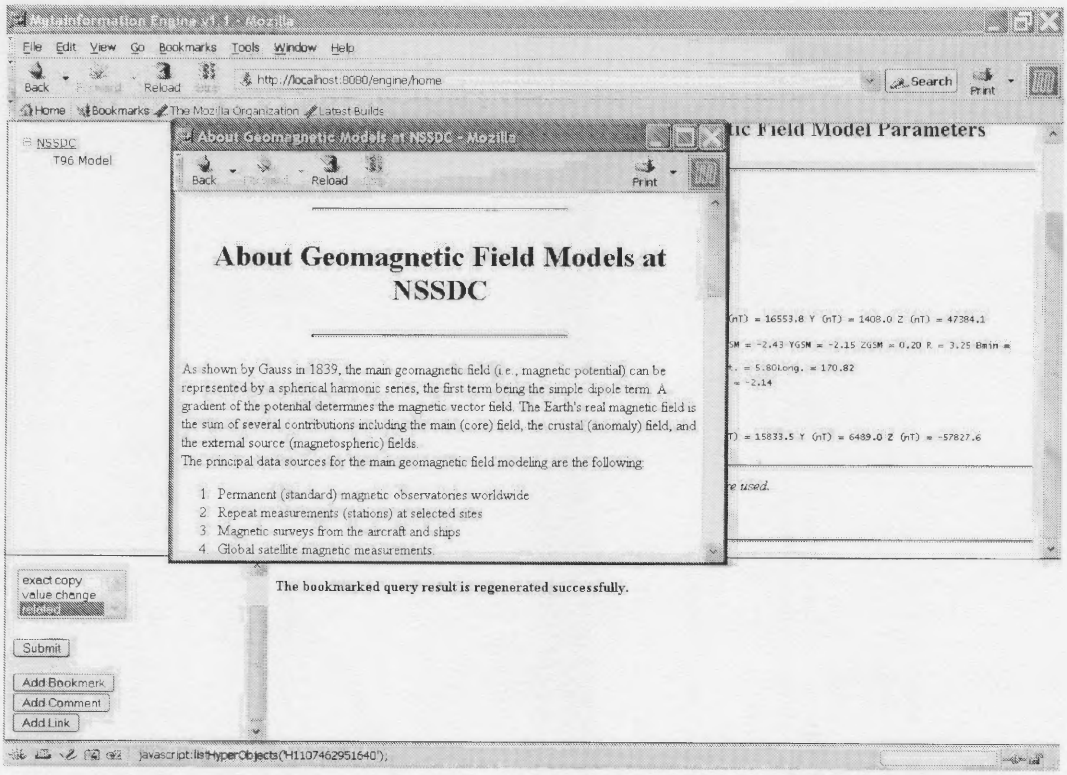

Figure 7.16 Traverse to the link destination URL.

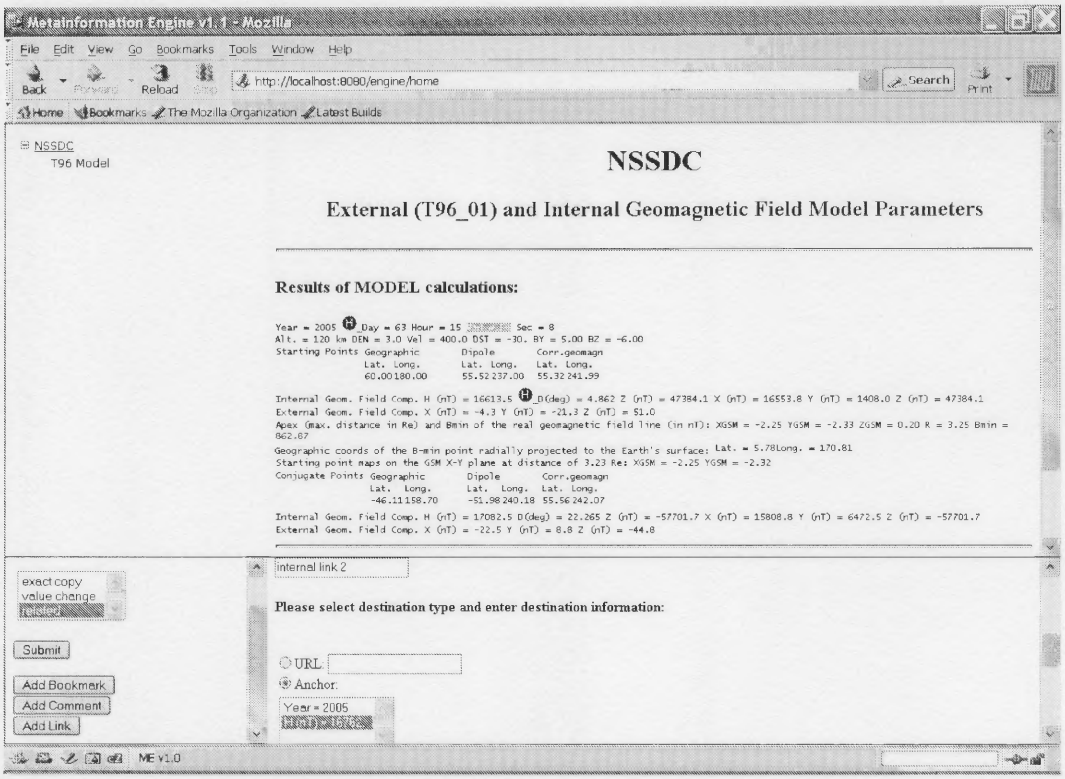

Figure 7.17 Add a link whose destination is an anchor.

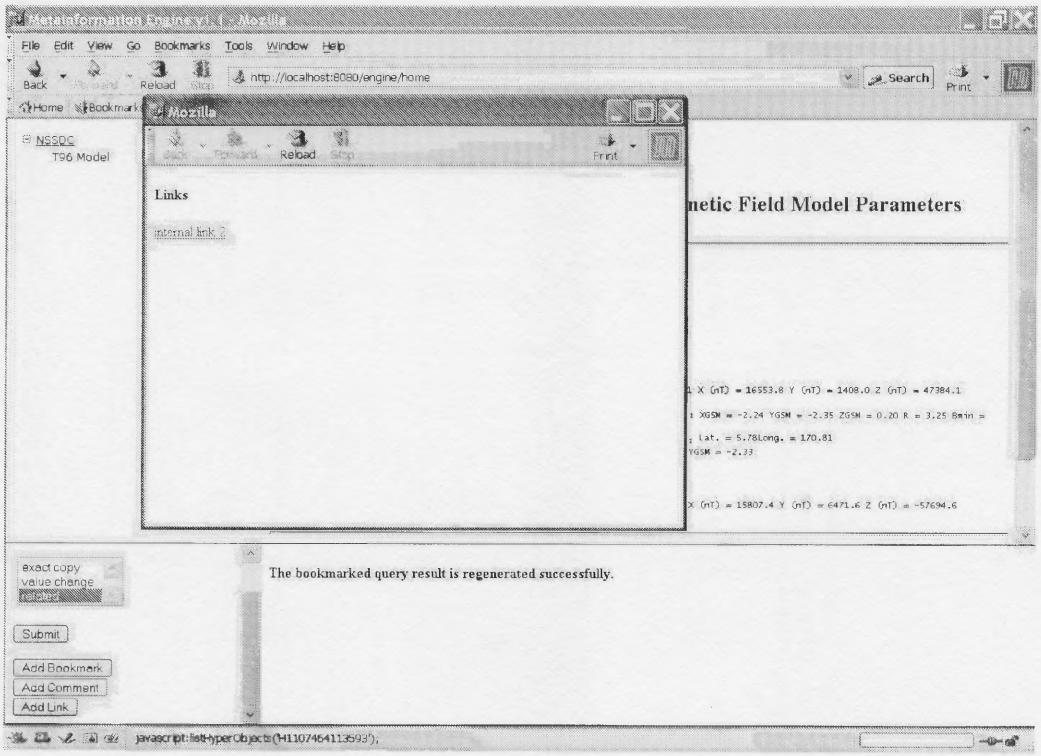

Figure 7.18 List of link information.

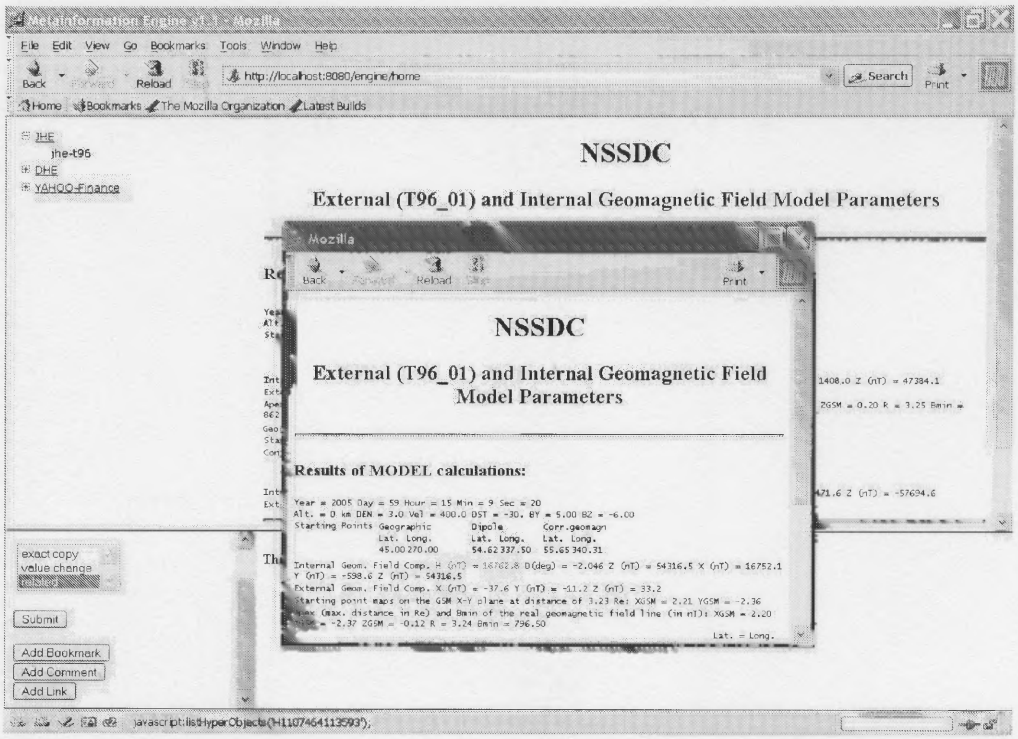

Figure 7.19 Traverse to link destination (to the anchor highlighted in red).

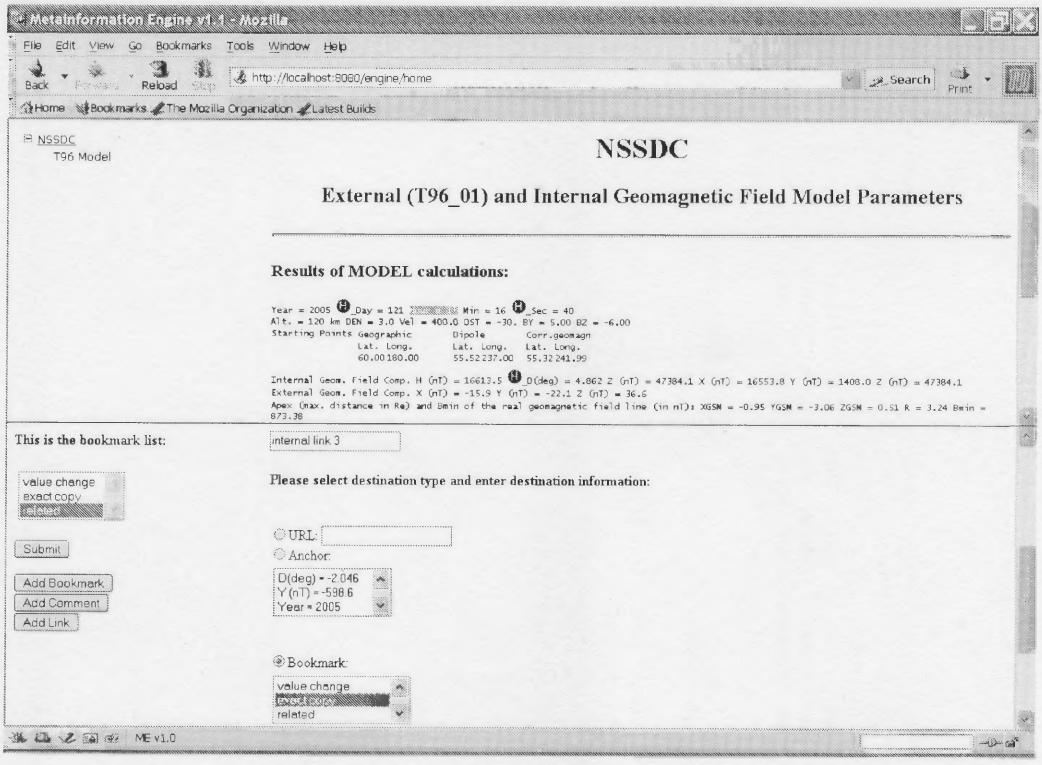

Figure 7.20 Add a link whose destination is a bookmark.

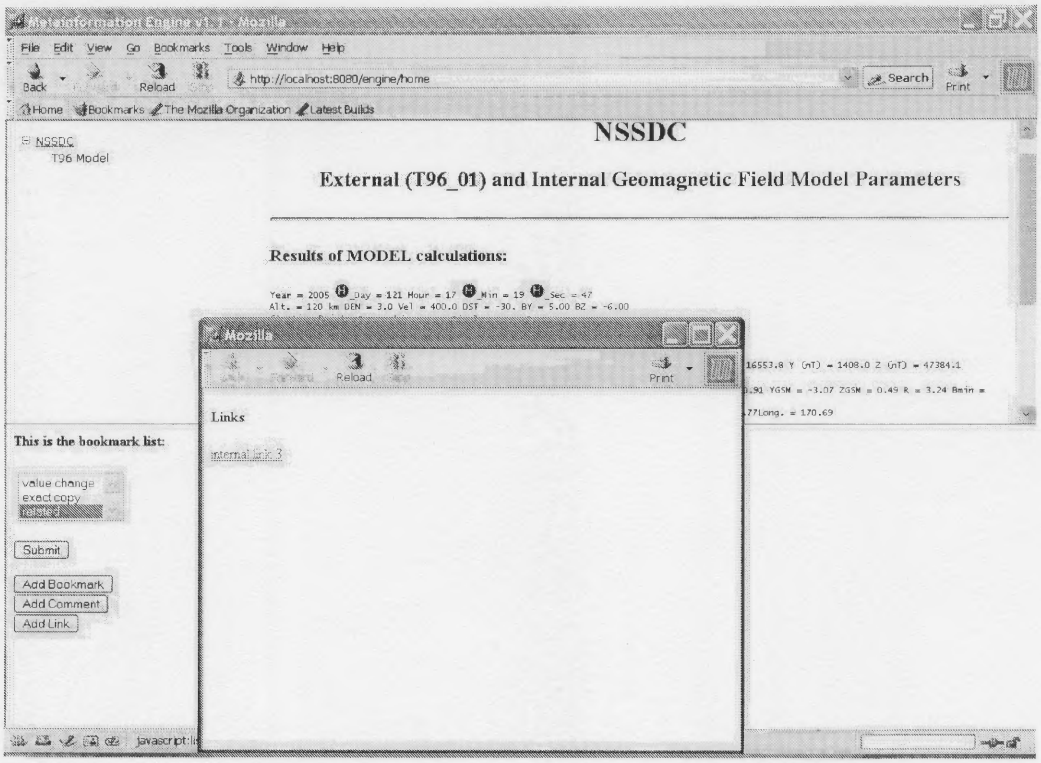

Figure 7.21 List of link information.

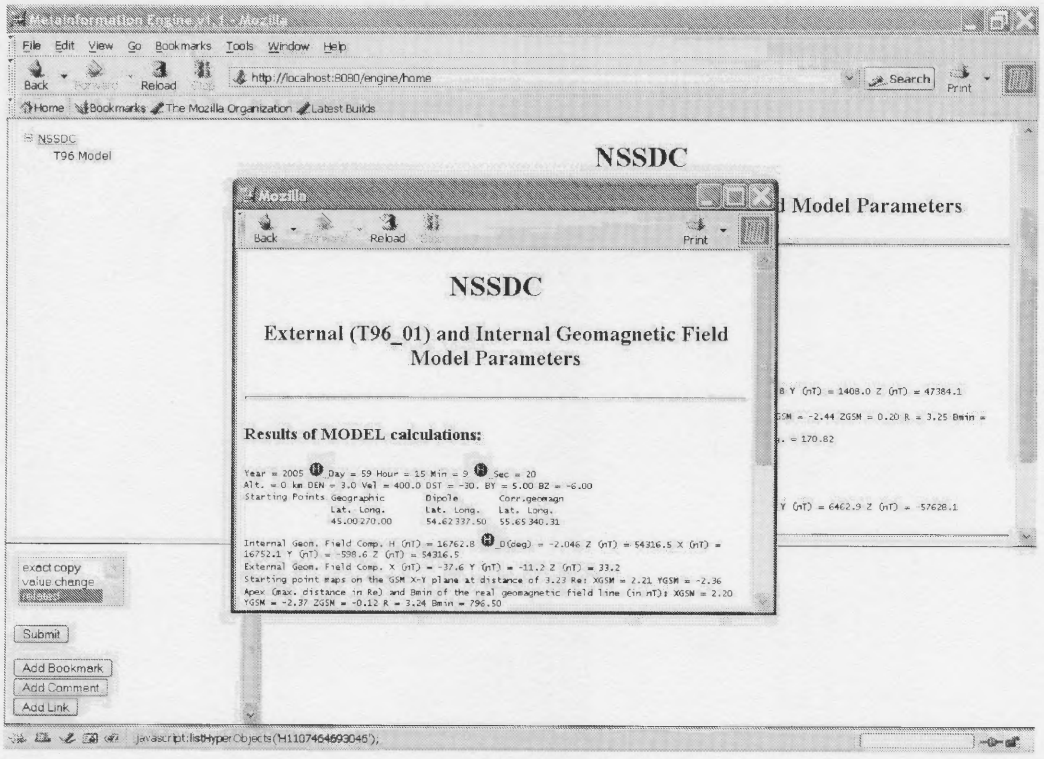

Figure 7.22 Traverse the link to a virtual document.

7.2.2.3 Evaluation. Figures 7.14 to 7.22 show that adding a new manual link is quite easy for users to use. The JHE system can store and get the link information from database. Also the user interface to add comment and display link information is quite good. There are three types of link destinations, which include a URL, a selected anchor and a bookmarked virtual document. This allows users to link to either resource outside JHE or resource inside JHE, which is quite flexible. Though currently JHE only supports manual link, it would be not too difficult to automatically create these links for anchors in virtual documents, as DHE supports.

7.3 Relocation and Re-identification

This section gives test results on relocation and re-identification. Relocation means to find the location of the anchor when the byte offset of the anchor has changed. This also relates to anchor granularity because it defines the scope of the anchor. Section 7.3.1 gives test results and evaluations on relocation. Re-identification is to identify whether the located anchors are same as before. This depends on the re-identification criteria that the user wants to use. Section 7.3.2 gives test results on re-identification and also does evaluations on re-identification.

7.3.1 Anchor Granularity and Relocation

There is no independent menu for creating an anchor. The anchor menu is part of the Comment Service Menu (when the user adds a comment) or the Link Service Menu (when the user adds a link). The anchor menu allows users to specify anchor granularity and re-identification criteria. Relocation is based on the anchor location and the granularity. This section discusses relocation test results and evaluations.

7.3.1.1 Test Results. There are three types of anchor granularity: global, local and specific. "Global" means the anchor can appear in any document that has the same element. "Local" means the anchor can appear in the same element anywhere in a particular document. "Specific" means the anchor can only appear at a particular location in a particular document.

Figure 7.23 shows how to create a "local" anchor on the selection "degree" in the document entitled "value change" when the user adds a comment on the anchor. The "degree" element is a sub element under element "Internal Geom. Field". It appears twice in this document. Figure 7.24 shows revisiting the document entitled "value change" by clicking on the "value change" bookmark. Since it is a "local" anchor, all "degree" elements in this document are attached with hypermedia constructs. By clicking on the icons besides same elements, users can see the same comment content.

Figure 7.25 shows how to create a "specific" anchor "horizontal". The "horizontal" element is a sub-element of element "Internal Geom. Field", which appears twice in this document. Figure 7.26 shows revisiting the document entitled "value change". Since "horizontal" is a "specific" element, the attached comment only appears at the original location of the element.

Figure 7.27 shows how to create a "global" anchor "year". The "year" element only appears once in this document. Figure 7.28 shows revisiting the document entitled "value change". The "year" element is attached with an icon. Figure 7.29 shows revisiting document "exact copy". The "year" element is attached with an icon. Since it's a "global" anchor, it appears in all documents that have the same element. But the other anchors "degree" and "horizontal" only appears at the document "value change".

Because they are not "global", they should only appear at the document that has same identifier when they are created.

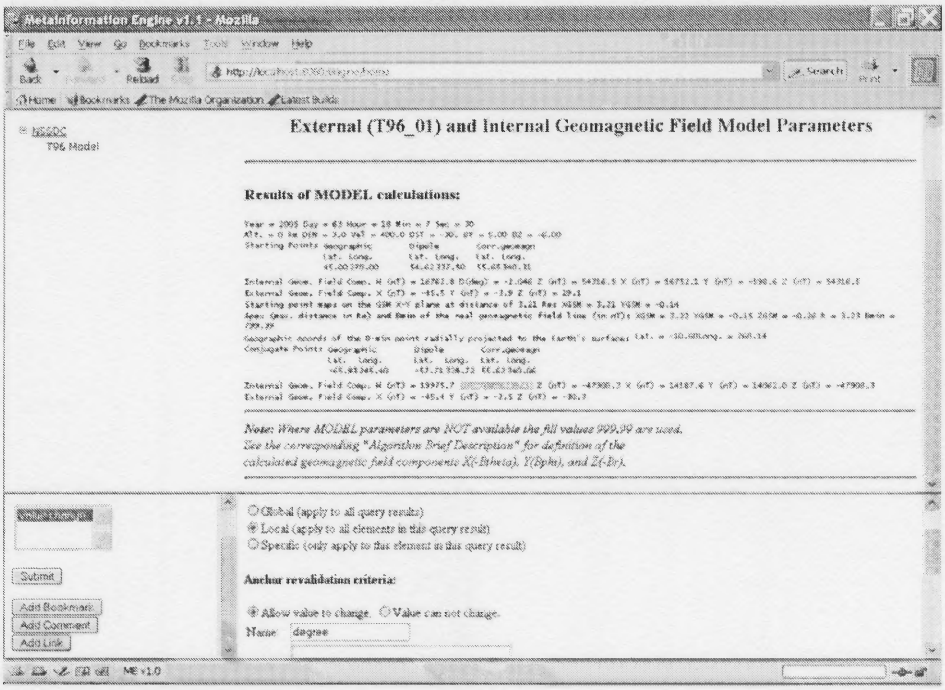

Figure 7.23 Create a "local" anchor.

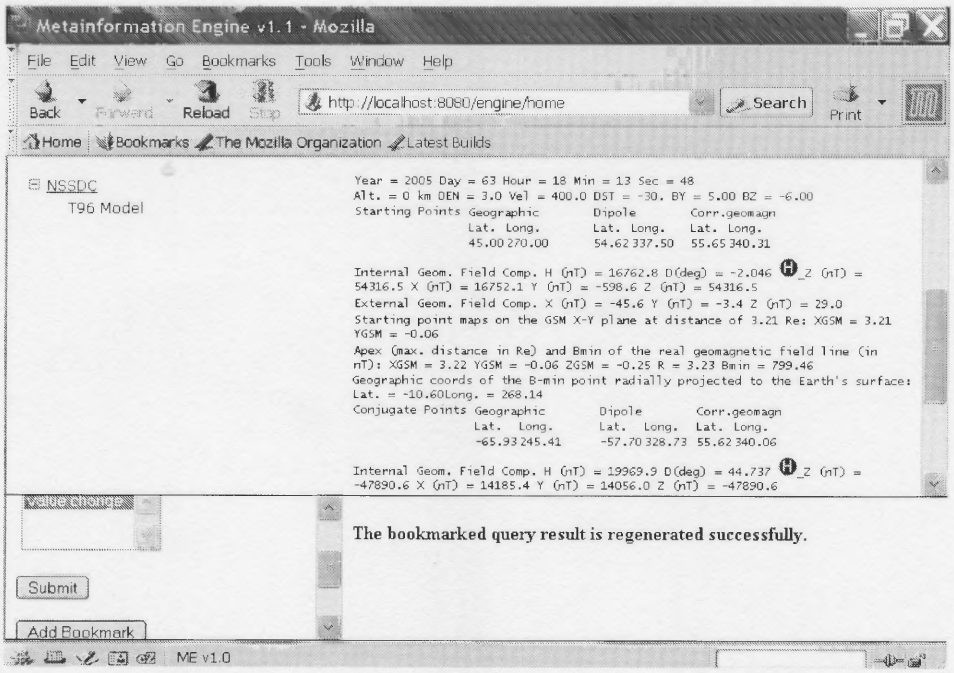

Figure 7.24 Relocate "local" anchors.

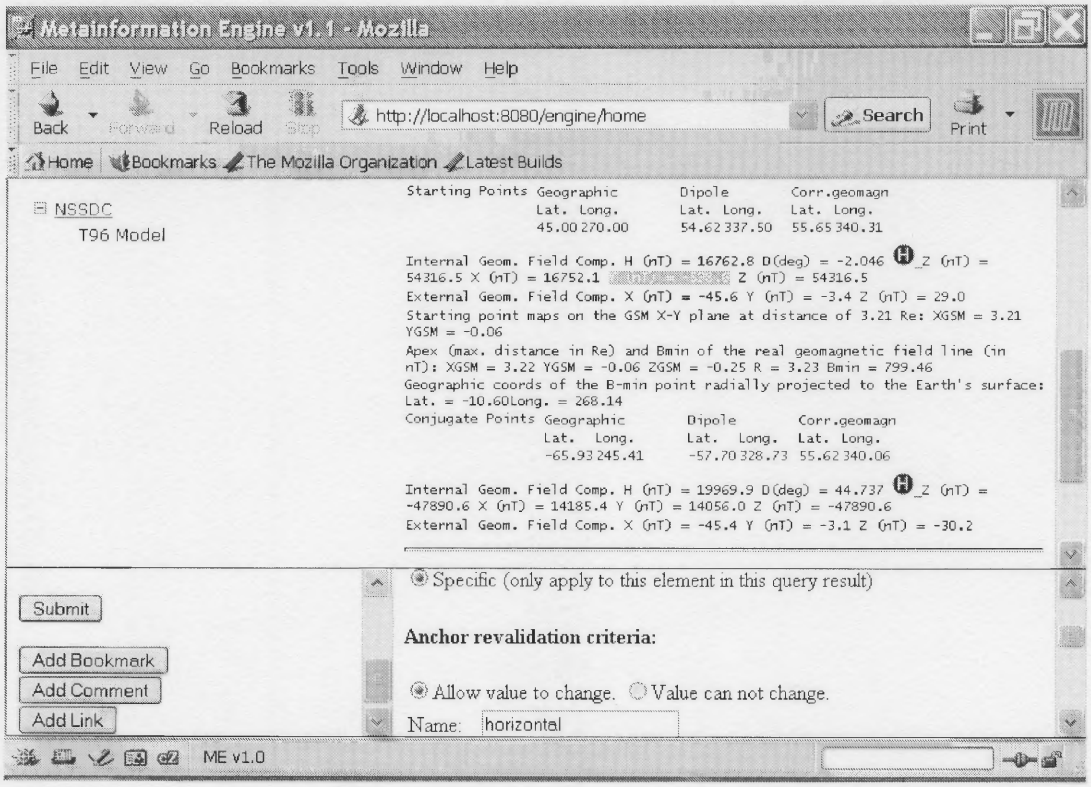

Figure 7.25 Create a "specific" anchor.

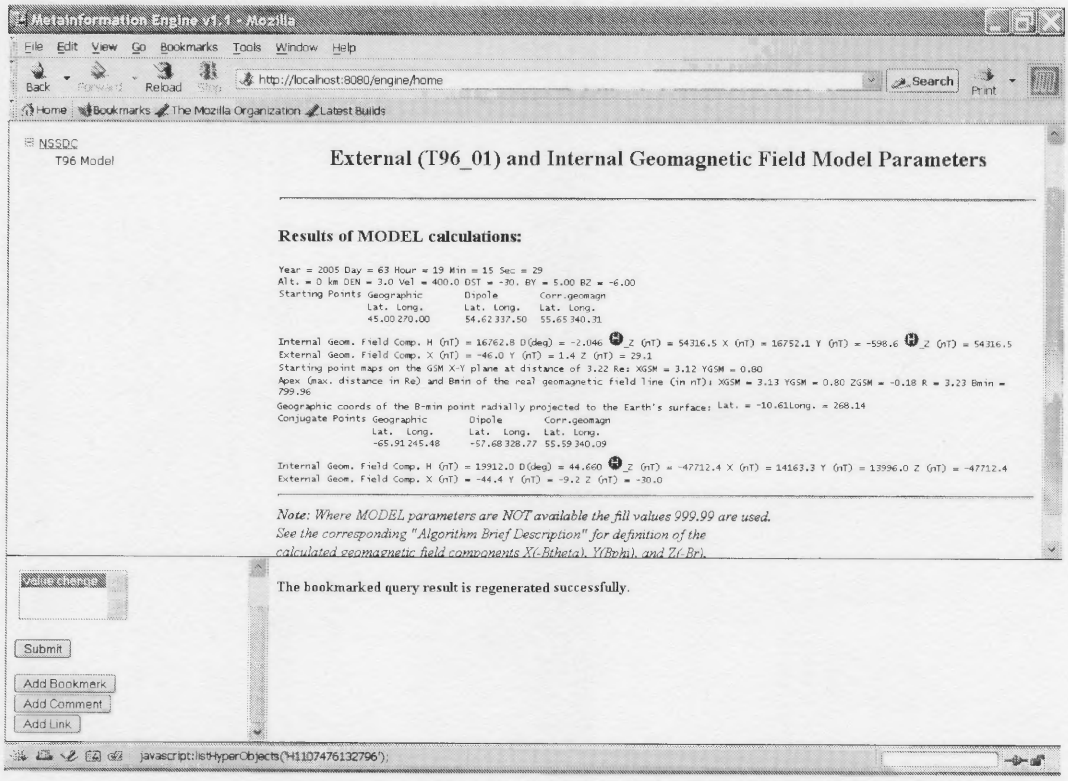

Figure 7.26 Relocate a "specific" anchor.

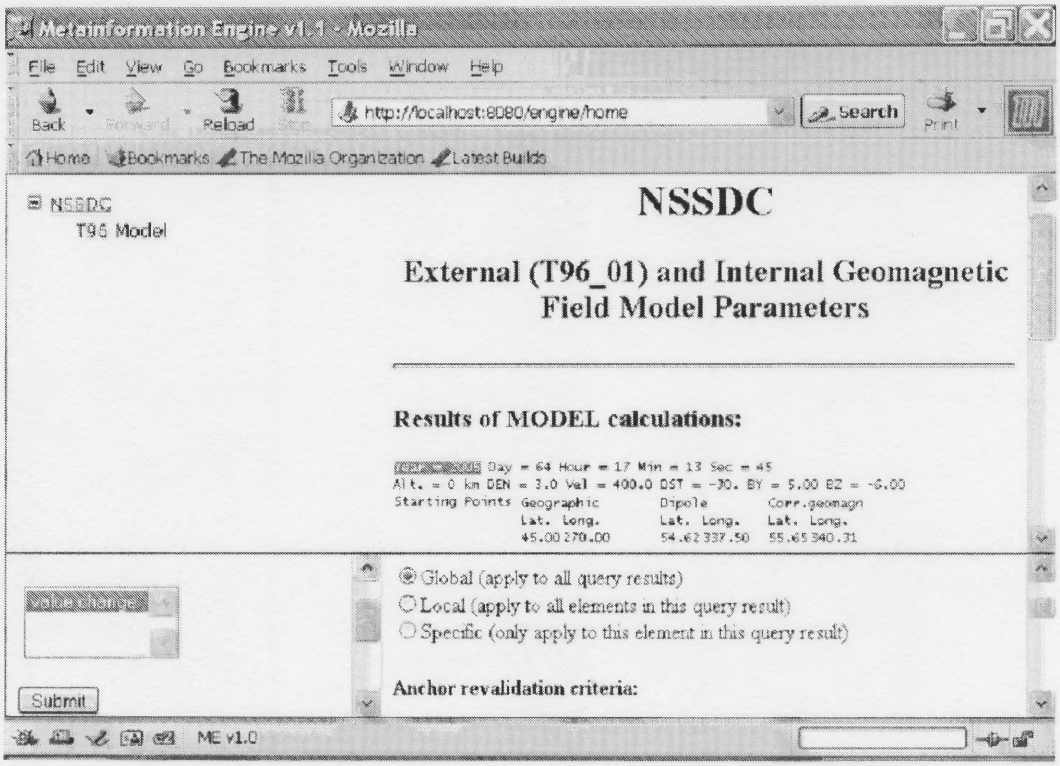

Figure 7.27 Create a "global" anchor.

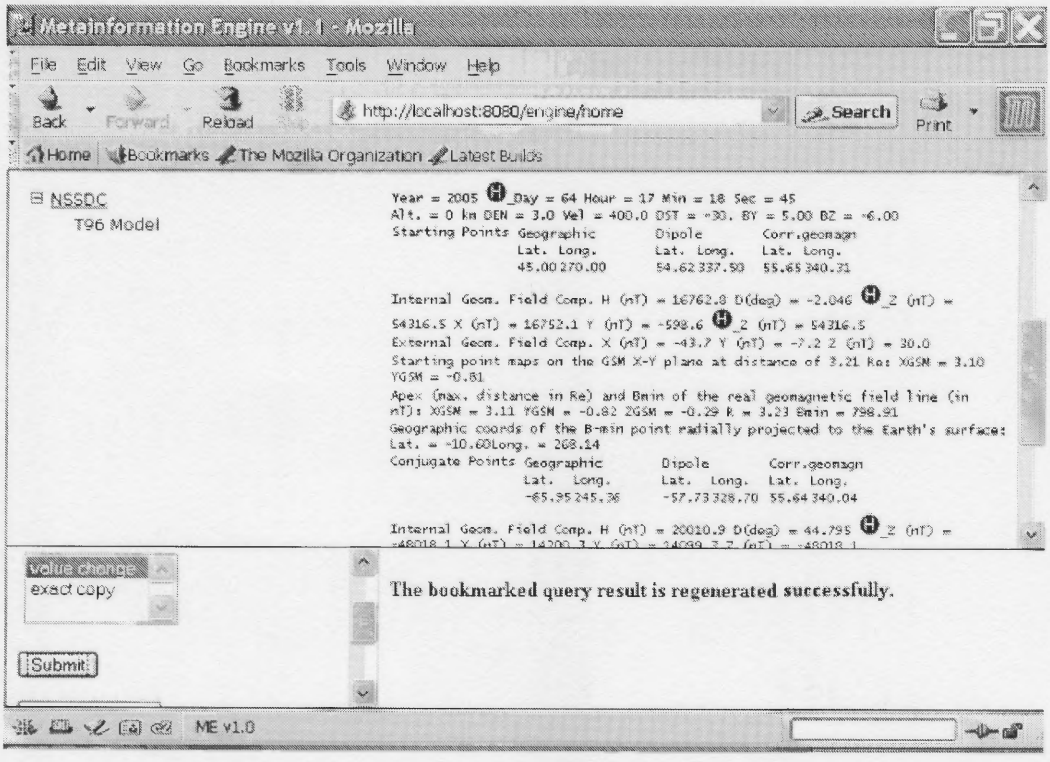

Figure 7.28 Revisit the same document.

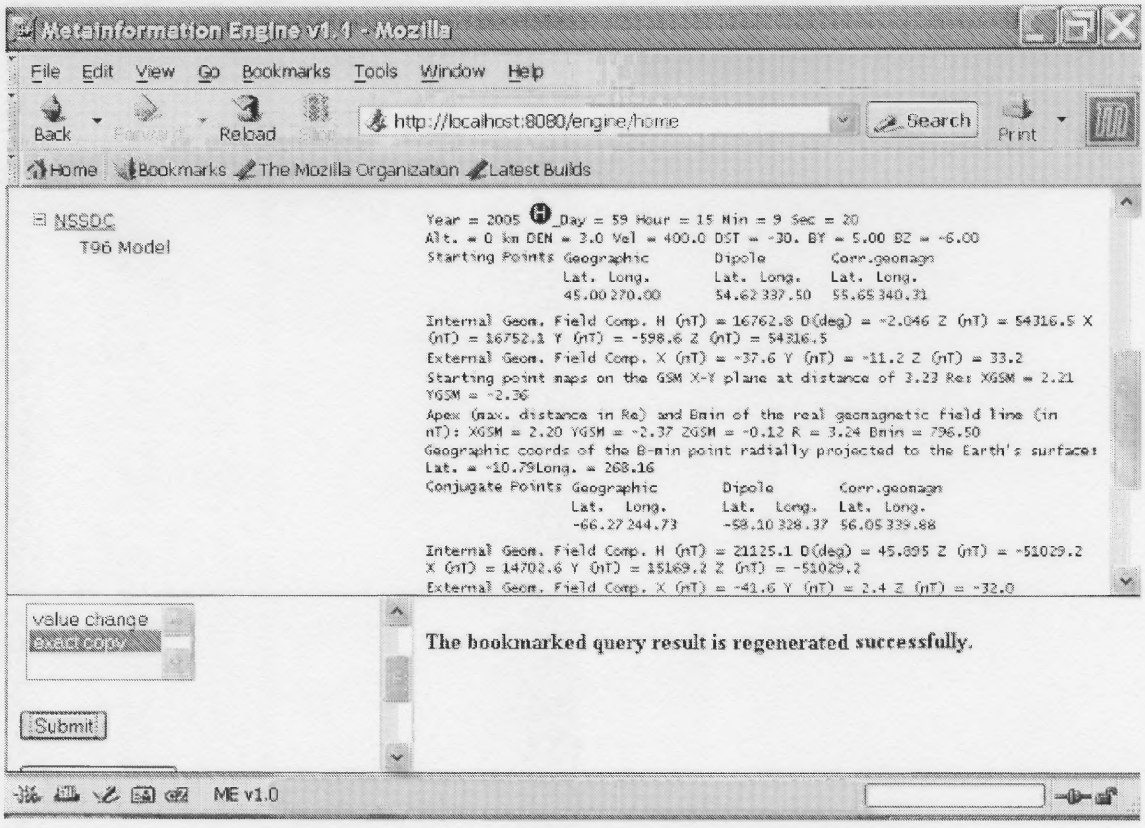

Figure 7.29 Revisit another document.

7.3.1.2 Evaluation. The above test results prove that JHE can successfully find the anchors' locations. The test results prove that JHE gives users ability to specify anchor granularity, which provides flexibility for users. For example, if a user adds a comment on an anchor and wants the comment to appear in all the related virtual documents, he can specify the granularity as "global". He does not need to add the anchor for all the other virtual documents for multiple times. If a user wants to add a comment that only relates to a very particular location, then he can specify the type "specific". If he wants to add comment once without retyping the same information for the same element appears at different locations, then he can give the type "local".

7.3.2 Re-identification

7.3.2.1 Test Results. After finding the anchor's location, JHE needs to identify that the relocated anchor is the same one as the originally marked one. There are two criteria for re-identification: element's value can change; element's value can not change. From Figures 7.23 to 7.29, all elements' values are allowed to change.

Figure 7.30 shows an anchor's re-identification criteria is "value can not change". Figure 7.31 shows revisiting the document. Since the original value is "2", the new value is "0", they are not same. JHE gives a warning message by adding a pink icon at the side of the element. For those valid elements, the icons attached are blue.

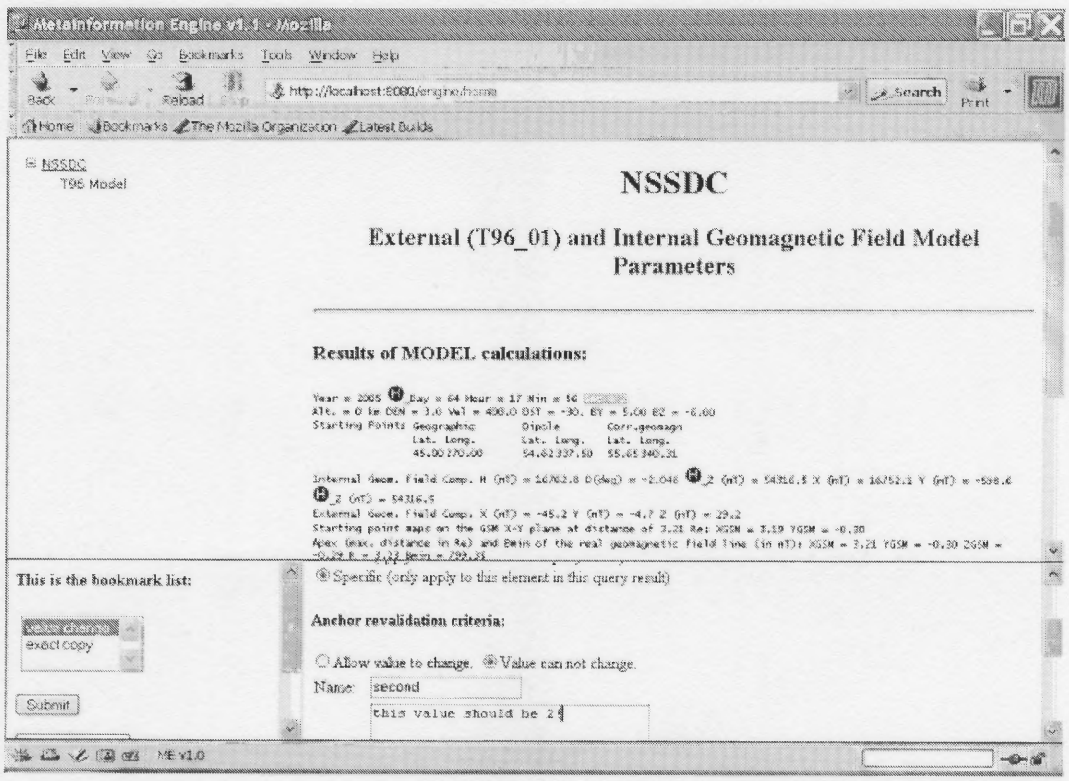

Figure 7.30 Add a "no value change" anchor.

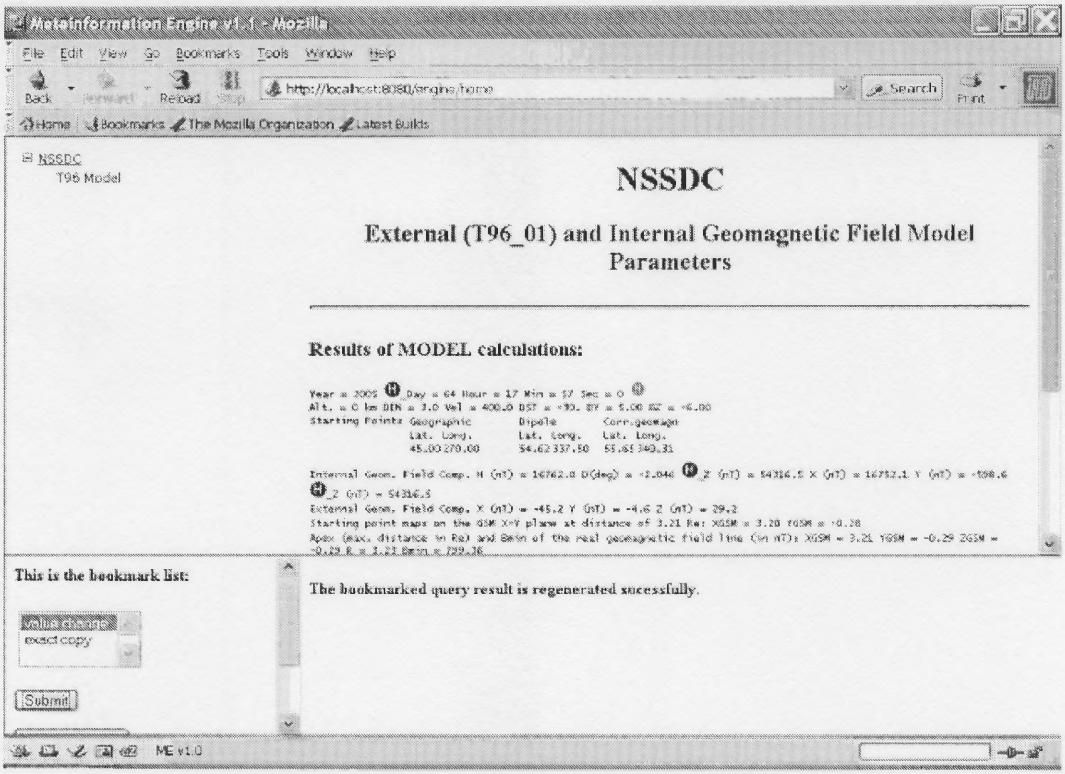

Figure 7.31 Revalidate anchors.

7.3.2.2 Evaluation. The above test results prove that JHE can re-identify re-located anchors successfully. JHE gives users ability to specify anchor re-identification criteria. If users require the anchor value to be not changeable, then JHE compares the new value with the original value. If they are not same, JHE gives a warning message to users but still attaches hypermedia constructs to the elements. This gives the user flexibility to decide whether he should remove the out-of-date anchor or just leave it there.

7.4 Comparisons with Other Similar Systems

To supplement applications with hypermedia functionality, a hypermedia engine is needed to model and maintain application-specific hypermedia structures. Hypermedia engines execute independently of an application with minimal modifications to it, and provide the application's users with hypermedia support. Notable projects include Microcosm's Universal Viewer (Davis 1994), Freckles (Kacmar 1995), the OO-Navigator (Garrido 1996), WebVise (Grønbæk 1999), SFX (Van de Sompel 1999), InfiniTe (Anderson 2001).

7.4.1 Microcosm

Microcosm is an open hypermedia system developed in University of Southampton which can support generic links and computed links. Microcosm supports viewers with different levels of awareness of the full set of Microcosm protocols. The fully aware viewers are applications adapted at the source code level. These fully aware viewers can participate in the full range of protocols. The semi-aware viewers are applications adapted in some way to participate in some subset of the full communication protocol. Typically this is achieved using the application's own macro language. Some application programs that one might wish to use offer no access to source code and have no macro programming language. Such applications can be hypermedia-enabled by using the Universal Viewer which sits on top of unaware program, and handles the communications with Microcosm.

The Microcosm Universal Viewer handles all communications with Microcosm for a list of applications without any modifications for the applications. The Universal Viewer maintains a list of applications and monitors Microcosm events in order to

identify when a program on this list is launched. When it identifies such an event it attaches a small Microcosm icon to the application's title bar. Clicking on this icon produces a menu, identical to the action menu on any other viewer. The Universal Viewer is a "proxy" which is responsible for all communications with Microcosm, and handling any communications that are possible with the application. The application communicates with the Universal Viewer through windows DDE (Dynamic Data Exchange), clipboard or a named swap file.

7.4.2 Freckles

"Freckles" uses an autonomous process that is completely external to the application. The facility monitors application activity and provides hypermedia services to applications running on a user's display.

These approaches can minimize the interactions between the application component and hypermedia engine thus reducing the extent of modification necessary to retrofit an existing application with hypermedia services provided by their engines. This also allows these facilities to support applications that cannot or do not need to be extensively modified to provide some level of hypermedia activity.

Freckles presents a model based on a process model of hypermedia and on the premise that a very "sparse" set of messages pass between the application and hypermedia components. It can be accomplished with no modification or a trivial modification of the application component; it requires no modification of the application's backend or data store.

The hypermedia facility monitors the activities in windows on the user's workstation. Whenever a window is opened, closed or moved, the hypermedia facility is notified of the event by the window manager. The window manager is an operating system specific process that can monitor window events (such as an XWindow mouse movement or activity). The hypermedia facility must interact with the window manager in order to obtain information about the existence of windows, specifically the title of a window, and the location of each window on the display. The window title serves as the identifier for the window's contents, allowing the hypermedia facility to create and maintain links based on the value of the title string. Links, anchors and navigation requests are maintained by the hypermedia facility. Link markers are determined and displayed by the hypermedia facility. The link destination is delivered to the application through the window manager or a file.

7.4.3 OO-Navigator

OO-Navigator is an object-oriented implementation of a domain-specific architecture for hypermedia. This framework comprises a set of class hierarchies representing well-known hypertext concepts: nodes, links, access structures and a collaboration model implemented in abstract classes that allows an object-oriented application to be extended with hypertext functionality. OO-Navigator views applications as having three layers: the objects layer, the hypermedia layer and the interface layer. The objects layer contains the application data and behavior, the hypermedia layer implements navigation, and the interface layer presents nodes and links. A communication model is defined to interact between the objects layer with hypermedia layer, and between hypermedia layer with interface layer. The OO-Navigator is implemented in SmallTalk environment.

7.4.4 WebVise

WebVise is an open hypermedia system to provide structures such as contexts, links, annotation, and guided tours stored in hypermedia databases external to the Web pages. This includes the ability for users collaboratively to create links from parts of HTML Web pages they do not own and support for creating links to parts of Web pages without writing HTML target tags.

The hypermedia structures are stored in a hypermedia database, developed from the DeVise hypermedia framework, and the service is available on the Web via an ordinary URL. The WebVise system is composed of WebVise clients and structure servers. A structure server maintains information about hypermedia structures (such as nodes, links and guided tours).

The WebVise client enables the end user to take advantage of open hypermedia services when working with arbitrary applications. It does so by being a kind of "middleware" between various applications running on the user's workstation and one or more structure servers.

The communication between the WebVise client and the structure servers is compliant to the Open Hypermedia Protocol – Navigational Interface (OHP-Nav). For each application that is extended with open hypermedia services, the WebVise Client implements a corresponding application wrapper. An application wrapper handles the communication with its corresponding application. This includes launching the application and requesting the application to open and display a document.

7.4.5 SFX

SFX is a generic link service in a hybrid library environment that can generate dynamic links. SFX is a link service system built in Ghent University in hybrid library environment.

SFX provides dynamic and open links for users; static links are not considered. SFX anticipates potential links for users. Links are presented to users only upon user request to reduce link generation delays. An SFX button is inserted into each record in the library.

For each link-source, an identifier is hidden behind an SFX-button. This identifier holds the following information:

- ID of the server from which the link-source originates
- database ID of the database where the link-source originates
- unique record ID of the link-source within that database
- the SFX server process that is executed by clicking this button

A user must explicitly request links for a link-source by clicking the SFX-button. Clicking transfers the identifier to the local target that uses it to pull the link-source into its environment. Next, the document is parsed into a generic format and essential parameters are extracted. Then the server does conceptual verification of potential links from the Colli (a collection of anticipated conceptual links). In order to prevent irrelevant links from being presented to the user, the SFX-base is introduced. The SFX-base describes the relationship between the conceptual links from the Colli and the parameter values of link-sources for which the conceptual links are valid. Matching parameters of a link-source with the SFX-base filters out irrelevant links.

7.4.6 InfiniTe

InfiniTe is an open hypermedia system that can store and maintain relationships for software artifacts. InfiniTe is designed to be integrated with the existing open hypermedia system Chimera (Anderson 1994), such that relationships created in InfiniTe can be viewed and traversed over the original software artifacts with an engineer's favorite set of open hypermedia aware tools. InfiniTe imports text documents into the InfiniTe environment, running integrators to generate keyword and index information. InfiniTe uses this information to generate anchors and links. Anchors and links information are imported to the Chimera open hypermedia system for display on the original text document.

InfiniTe uses the XPointer standard to specify locations in documents. A mapping must exist between XPointers in InfiniTe and LocSpecs in an open hypermedia system that refer to the original document. The InfiniTe integrator translates text documents into InfiniTe, performs a keyword search and then exports relationships as XLinks to Chimera system. Then the Chimera system views relationships from InfiniTe. N-ary relationships are supported by using XLinks.

7.4.7 Comparisons

Another hypermedia engine system is the Dynamic Hypermedia Engine (DHE). DHE is a hypermedia system that can provide hypermedia functionality for analytical applications, by intercepting documents or messages between the application's computational and user interface. Details are described in Section 5.2.

Hypermedia engines differ in how applications integrate with hypermedia systems; what kind of hypermedia functionality the hypermedia engine can supply, and how applications communicate with the hypermedia engine. For example, Microcosm's Universal Viewer handles all communications without modification to application's source codes; WebVise and DHE use application wrappers to integrate with hypermedia systems. Early hypermedia engines (Microcosm and Freckles) only support first-generation hypermedia functionality (such as links and nodes); while WebVise and InfiniTe supports more hypermedia functionality and are Web-based. Many hypermedia engines communicate with applications by message passing using existing protocols (e.g, WebVise uses OHP); some use operating system specific tools to handle communications (such as Freckles uses window event manager).

The above hypermedia engines are evaluated by using the JIT hypermedia framework described in Chapter 3. Table 7.1 shows evaluations of existing hypermedia engines. These hypermedia engines have many characteristics in common, such as integration with viewers, integration with information systems, support for identifiers of objects, internal document addressing and hypermedia functionality. None of them supports virtual documents, except DHE and SFX. DHE and SFX can analyze the underlying relationships based on the application information, then parse the source documents and insert links to these documents. Links are dynamic and automatically generated. However, they do not support regeneration, re-location and re-identification.

Table 7.1 Evaluations of Hypermedia Engines

System Requirements	Microcosm	Freckles	WebVise	DHE	OO-Navigator	SFX	InfinTe
Integration with viewer	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Integration with IS	No	No	No	Yes	No	Yes	No
Virtual Doc support	No	No	No	Yes	No	Yes	No
Document generation	No	No	No	Yes	No	No	No
Document regeneration	No	No	No	No	No	No	No
Re-location	Yes	No	Yes	No	No	No	No
Re-identification	Yes	No	Yes	No	No	No	No
Unique IDs for objects	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Internal document addressing mechanisms	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Application specific metadata	No	No	No	Yes	No	Yes	No
Hypermedia functionality	Yes	Yes	Yes	Yes	Yes	Yes	Yes

7.5 System Extensibility

Another example of NASA's model is T89C. T89C is similar to T96 model. The difference is that, T89C does not use solar wind as one of the parameters needed. Figure 7.32 shows the homepage (http://nssdc.gsfc.nasa.gov/space/cgm/t89.html) of T89C model. Figure 7.33 shows the T89C calculation result.

Adding a new module into this project has following steps: (1) Analyze document structure; (2) Markup document; and (3) Add T89C as a new module to JHE. Actually, adding a new application to JHE does not require too much work to do. Most components (Selection Manager, UIW, Regeneration Engine, Service Module, and Document Manager) do not need to change. A new Document Translator and Application Wrapper for the new application need to be added. This proves JHE has good system extensibility.

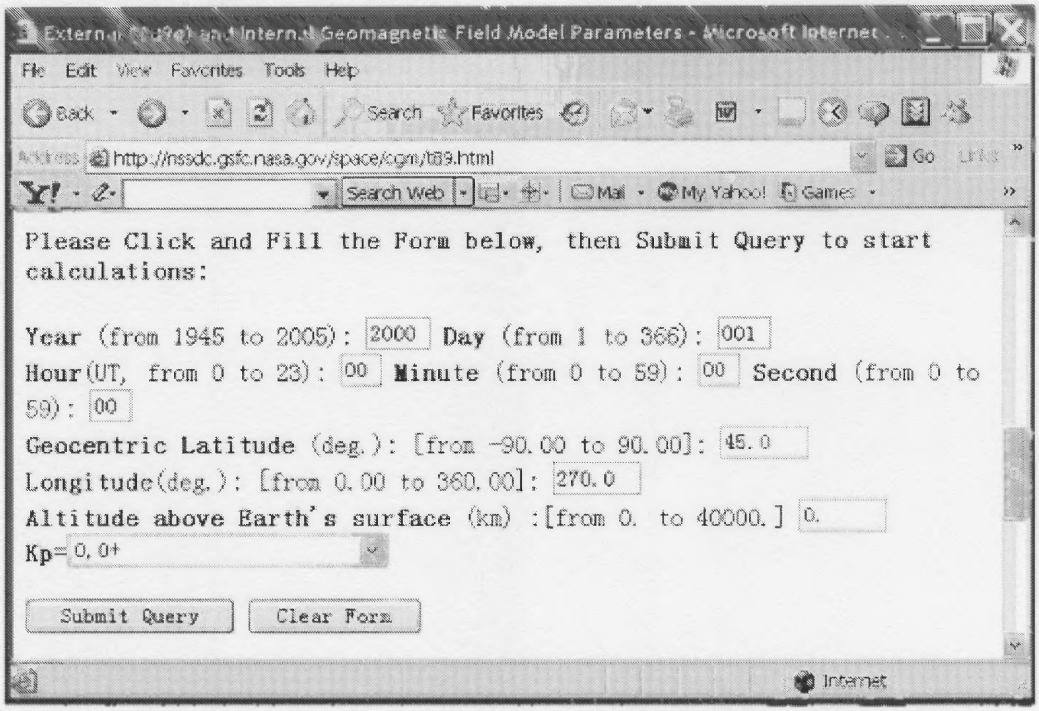

Figure 7.32 T89C model query table.

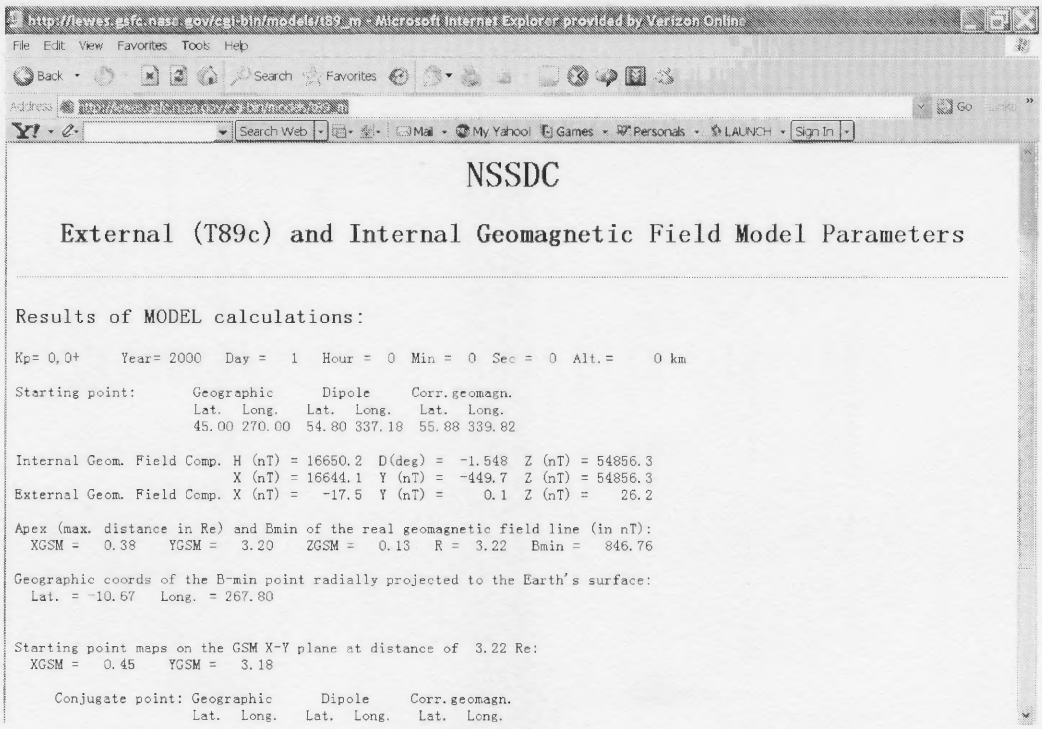

Figure 7.33 T89C calculation result.

7.6 Standards Used In JHE

JHE uses XML, XML Schema, XPointer, XPath and HTML standards. JHE translates the source documents into XML documents and represents the document structure with XML Schema. JHE generates XPointer expressions to represent anchor location information and uses XPath tools to relocate anchors in the documents. JHE uses HTML to display the source documents and implement some User Interface functions.

7.7 Reflections on the Implementation

Due to the current technologies, there are some limitations of the current JHE system.

(1) Currently the document structure is inferred and created manually by the JHE developer. If JHE can automatically infer the document structure, it will make application integration into JHE more efficient.

(2) Current JDOM tools cannot handle very big and complicated document well. Many existing Web sites generate documents that contain multimedia objects and script functions, which are difficult to parse.

(3) The Document Translator currently is using manually created stylesheet files to form the display Web pages. Due to the limitation of the current stylesheet language, it is difficult to mimic the appearance of the translated documents to be exactly sane as the source documents from the application.

(4) If the element's value is changed, JHE gives users warning messages by inserting a pink icon at the side of the element. Currently JHE does not allow users delete the invalid anchor or comment. It is not difficult to extend for the future system.

(5) JHE does not specify user access rights. Users can view comments from other persons. In the future JHE should have access right control functionality.

(6) JHE does not identify whether a newly created document looks similar to some document that the user has created previously.

(7) The sameness criteria that allows parts of the document's content to change is not implemented due to the complexity of storing and identifying parts of the document structure. However, it should not be very difficult to implement in the future JHE system.

CHAPTER 8

CONTRIBUTIONS

This research makes the following contributions:

(1) It gives clear descriptions for differences between virtual and static documents from many aspects. The comparisons provide considerations for designing a JIT hypermedia system framework and developing a JHE system as a conceptual proof of the framework.

(2) It introduces the concept of dynamic hypermedia functionality, which provides traditional hypermedia functionality for dynamically-generated virtual documents. The JHE system implements several forms of dynamic hypermedia functionality: dynamic bookmarks, manual links and user-declared comments. Each works for virtual documents and elements within them that can change between generations.

(3) It introduces the concept of dynamic regeneration of virtual documents, which is a requirement for providing virtual documents with dynamic hypermedia functionality. It discusses the requirements, information needed in a JIT system, and the procedure to do dynamic regeneration. This information serves as guidelines to solve the dynamic regeneration problem and helps the developer to build a practical software implementation. JHE facilitates analytical applications to regenerate documents pointed to by users' bookmarks and links. JHE re-validates the regenerated documents according to the sameness criteria, which gives the user some flexibility to manipulate hypermedia functionality.

(4) It introduces the concept of element relocation and re-identification for virtual documents. It discusses requirements, information needed in a JIT system, and the procedure to do relocation and re-identifying for virtual elements. This information helps the developer to choose appropriate methods to solve the relocation and re-identification problem. JHE relocates and re-identifies previously identified objects within (re)generated virtual documents using the document's structure and based on flexible criteria for revalidation.

(5) It designs a framework for JIT hypermedia systems. The framework explores the challenges in JIT hypermedia systems, analyzes the system requirements and proposes a comprehensive overview. It could help one build and evaluate a JIT hypermedia system.

(6) It designs the JHE system in terms of identifiers, implementation architecture, and information flow. The JHE system provides practical solutions to solve problems encountered in the JIT research area and meets all the requirements of the JIT hypermedia system framework.

CHAPTER 9

DISSERTATION BOUNDARIES AND FUTURE WORK

This section presents some boundaries of this dissertation, reviews the deliverables and describes future research.

9.1 Dissertation Boundaries

This research does not deal with: (1) For binary and multimedia formats, addressing into objects has more complex ways than text-only documents, and selections from objects require tighter integration with the viewer. Therefore, JHE will only handle the textual content of documents. (2) In some applets and highly interactive systems, some computation is coded directly into the screen and executed directly on the client viewer. In these cases, a hypermedia engine may need to be more tightly integrated into the interface computation, in order to parse the contents of displays as they are created on the client. The current JHE will not address this situation. The JHE implementation has many limitations due to current technologies' limitations (see Section 7.7). Several future extensions are also envisioned (see Section 9.3).

9.2 Deliverables

This project has several procedural and technical results. All are well-documented and available free of charge.

- The JHE infrastructure.
- Full documentation for JHE and integrating applications (how to analyze application metadata, how to write application wrappers).

- The hypermedia services (add bookmarks, comments, links, etc.) JHE created to prove the infrastructure works.

- Java and Javascript source codes to implement JHE modules.

- Documentation for JHE system settings and configurations.

9.3 Potential Future Work

Future work includes:

- Implementing more hypermedia functionality for the JHE system, such as automatic linking and guided tours. Since DHE already supports automatic linking, it is not difficult to add an automatic linking hypermedia service. To build a guided tour, the JHE developer should have a through understanding of the node relationships inside the application. For example, which document is the root, and which documents are the leaves or subsections.

- Integrate more applications to JHE. Adding a new application into JHE requires the JHE developer to analyze the document structure in the application and to build a new Application Wrapper for the application.

- Improving and extending the JIT hypermedia framework. The JIT hypermedia system can support more functionality for users. For example, it can add access permission modules based on user groups.

- Support virtual documents with binary formats. JHE needs a new Selection Manager to get selections from binary objects (such as an image in the Web page) and a different format to express locations in a binary object.

- Investigating more granularities for determining when documents and elements are same. These could include: same "context"; same/related "semantic meaning" from a thesaurus or some concept on the same ontology tree; same document structure/template; same document cluster; from the same application.

- Using different methods to generate unique identifiers instead of the system timestamp in current implementation to make identifiers more intelligent.

- Add more functionality to allow users edit or delete user declared bookmarks, comments or links.

- Improve JHE's implementation by applying more efficient and intelligent tools.

REFERENCES

Akscyn, R., D.L. McCracken, and E. Yoder. 1988. "KMS: A Distributed Hypertext for Sharing Knowledge in Organizations." *Communication of ACM*. Vol. 31(7): 820-835.

Allan, J. 1996. "Automatic Hypertext Link Typing." *Proceedings of the 7th ACM Conference on Hypertext*: 42-52.

Anderson, K.M., R.M. Taylor, and Jr. E.J. Whitehead. 1994. "Chimera: Hypertext for Heterogeneous Software Environments." *Proceedings of the ACM Conference on hypertext*: 94-107.

Anderson, K., S. Sherba, and W. Lepthien. 2002. "Towards Large-scale Information Integration." *Proceedings of the 24th International Conference on Software Engineering*.

Andrews, K., F. Kappe, and H.A. Maurer. 1995. "The Hyper-G Network Information System." *Journal of Universal Computer Science*. Vol. 1(4): 206-220.

Ashman, H., and J. Verbyla. 1993. "Dynamic and Externalized Linking: Requirements of Large-Scale Hypermedia Management Systems." *Hypertext '93 Workshop on Hyperbase Systems*.

Bailey, C., and W. Hall. 2000. "An Agent-Based Approach to Adaptive Hypermedia Using a Link Service." *Proceedings on Adaptive Hypermedia and Adaptive Web-Based Systems International Conference*: 260-263.

Bailey, C., S.R. El-Beltagy, and W. Hall. 2001. "Link Augmentation: A Context-Based Approach to Support Adaptive Hypermedia." *12th ACM Conference on Hypertext and Hypermedia*.

Bailey, C., W. Hall, D. Millard, and M. Weal. 2002. "Towards open Adaptive Hypermedia." *Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*.

Berners-Lee, T., R. Cailliau, J.F. Groff, and B. Pollermann. 1992. "World Wide Web: the Information Universe." *Electronic Networking: Research, Applications and Policy*. Vol. 2(1): 52-58.

Berners-Lee, T., R. Fielding, and L. Masinter. 1998. "Uniform Resource Identifiers (URI): Generic Syntax." http://www.ietf.org/rfc/rfc2396.txt, retrieved on April 2002.

Bhaumik, A., D. Dixit, R. Galnares, M. Tzagarakis, M. Vaitis, M. Bieber, V. Oria, A. Krishna, Q. Lu, F. Aljallad, and L. Zhang. 2001. "Towards Hypermedia Support for Database Systems." *Proceedings of the 34th Hawaii International Conference on System Sciences.*

Bieber, M. 1992. "Automating Hypermedia for Decision Support." *Hypermedia.* Vol. 4(2): 83-110.

Bieber, M., F. Vitali, H. Ashman, V. Balasubramanian, and H. Oinas-Kukkonen. 1997. "Fourth Generation Hypermedia: Some Missing Links for the World Wide Web." *Int. J. Human-Computer Studies, World Wide Web Usability Special Issue.* Vol. 47(1): 31-65.

Blake, M. 2002. "Implementation of the OpenURL and the SFX Architecture in the Production Environment of a Digital Library." http://lib-www.lanl.gov/lww/articles/OpenURL_vala.pdf, retrieved on May 2002.

Bodner, R., and M. Chignell. 1999. "Dynamic Hypertext: Querying and Linking." *ACM Computing Survey.* Vol. 31(4).

Brusilovsky, P.L. 1998. "Adaptive Navigation Support in Educational Hypermedia: An Evaluation of the ISIS-Tutor." *Journal of Computing and Information Technology.* Vol. 6: 27-38.

Bush, V. 1945. "As We May Think." *The Atlantic Monthly.* Vol. 176(1): 101-108.

Cannataro, M., and A. Pugliese. 2001. "XAHM: an XML-based Adaptive Hypermedia Model and its Implementation." *Third Workshop on Adaptive Hypertext and Hypermedia, 12th ACM Conference on Hypertext and Hypermedia.*

Carr, L., D.D. Roure, W. Hall, and G. Hill. 1995. "The Distributed Link Service: A Tool for Publishers, Authors and Readers." *The Web Journal.* Vol. 1(1): 547-656.

Carr, L.A., W. Hall, and S. Hitchcock. 1998. "Link Services or Link Agents?" *Proceedings of Ninth ACM Conference on Hypertext.*

Carr, L.A., D.D. Roure, W. Hall, and G. Hill. 1999. "Implementing an Open Link Service for the World-Wide Web." *ACM Computing Surveys.* Vol. 31(4).

Carr, L.A., S. Bechhofer, C. Goble, and W. Hall. 2001. "Conceptual Linking: Ontology-based Open Hypermedia". *Proceedings of Tenth World Wide Web Conference.*

Caumanns, J. 1999. "A Modular Framework for the Creation of Dynamic Documents." *Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference.*

Chen, L., and K. Sycara. 1998. "WebMate: A Personal Agent for Browsing and Searching." *Proceedings of the 2nd International Conference on Autonomous Agents.*

Conklin, J. 1987. "Hypertext: An Introduction and Survey." *IEEE Computer.* Vol. 20(9): 17-41.

Conklin, J., and M.L. Begeman. 1988. "gIBIS: A Hypertext Tool for Exploratory Policy Discussion." *ACM TOIS*: 140-152.

Cranor, L., M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. 2000. "Platform for Privacy Preferences Project 1.0 (P3P1.0) Specification." http://www.w3.org/TR/REC20001215, retrieved on March 2002.

Crossref. http://www.crossref.org, retrieved on May 2003.

Cunliffe, D. 2000. "Trailblazing: trends in hypermedia." *The New Review of Hypermedia and Multimedia.* Vol. 1: 19-46.

Davis, H., W. Hall, I. Heath, G. Hill, and R. Wilkins. 1992. "Microcosm: an Open Hypermedia Environment for Information Integration." *University of Southampton CSTR*: 92-15.

Davis, H.C., S. Knight, and W. Hall. 1994. "Light Hypermedia Link Services: A Study of Third Party Application Integration." *Proceedings of the 1994 European Conference on Hypermedia Technology.*

Davis, H.C. 1995(1). "To Embed or Not to Embed?" *Communications of the ACM (CACM).* Vol. 38(8): 108-109.

Davis, H.C. 1995(2). "Data integrity problems in an open hypermedia link service." *Ph.D. Dissertation, Southampton University.*

Davis, H.C., A. Lewis, and A. Rizk. 1996. "OHP: A Draft Proposal for a Standard Open Hypermedia Protocol." *2nd Workshop on Open Hypermedia Systems.* http://diana.ecs.soton.ac.uk/~hcd/protweb.htm, retrieved on April 2003.

Davis, H.C. 1999. "Hypertext Link Integrity." *ACM Computing Surveys.* Vol. 31(4).

De Bra, P., A. Aerts, G.J. Houben, and H. Wu. 2000. "Making General-Purpose Adaptive Hypermedia Work." *Proceedings of the AACE WebNet Conference*: 117-223.

Delisle, N.M., and M.D. Scharwtz. 1986. "Neptune: A Hypertext System for CAD Applications." *Proceedings of ACM SIGMOD'86*: 132-142.

DOI. 1998. "The Digital Object Identifier System and its applications in the Online Information Industry: an Overview for Southern African Information Professionals." http://www.unisa.ac.za/library2/afdeling/tdienste/doi.html, retrieved on May 2003.

DOM. 2001. "Document Object Model". http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913/DOM3-Core.html, retrieved on April 2002.

DSSSL. 1996. International Standards Organization, "Document Style Semantics and Specification Language (DSSSL) (ISO 10179)." *Geneva: ISO.* 1996

Dublin Core, http://www.dublincore.org, retrieved on June 2002.

Duhig, A.J. 2001. "Separating Links from Content Using XML, XLink and XPointer." http://www.gca.org/papers/xmleurope2001/papers/pdf/s16-2.pdf, retrieved on June 2003.

Engelbart, D.C. 1963. "A Conceptual Framework for the Augmentation of Man's Intellect." *Vistas of Information Handling.* Vol. 1.

Ferraiolo, J. 2001. "Scalable Vector Graphics (SVG) Specification." http://www.w3.org/TR/SVG, retrieved on June 2003.

Galnares, R. 2001. "Augmenting Applications with Hypermedia Functionality and Metainformation." *Ph.D. Dissertation, New Jersey Institute of Technology.*

Garrido, A., and G. Rossi. 1996. "A Framework for Extending Object-Oriented Applications with Hypermedia Functionality." *The New Review of Hypermedia and Multimedia.* Vol.2: 25-41.

Garzotto, F., D. Schwabe, and P. PaoLini. 1993. "HDM-A Model Base Approach to Hypermedia Application Design." *ACM Transaction on Information Systems.* Vol. 11(1): 1-26.

Campbel, B., and J.M. Goodman. 1987. "HAM: A General-Purpose Hypertext Abstract Machine." *Proceedings of ACM Hypertext'87 Conference*: 21-32.

Grønbæk, K., N. Bouvin, and L. Sloth. 1997. "Designing Dexter-based hypermedia services for the World Wide Web: Experiences with Walden's Paths." *Proceedings of the ACM Hypertext '97 Conference*: 167-176.

Grønbæk, K., and U.K. Wiil. 1997. "Towards a Reference Architecture for Open Hypermedia." *Proceedings of ACM Hypertext '97 Conference.*

Grønbæk, K., L. Sloth, and P. Qrbek. 1999. "WebVise: Browser and Proxy Support for Open Hypermedia Structuring Mechanisms on the WWW." *Proceedings of the 8th International World Wide Web Conference*: 232-267.

Grønbæk, K., L. Sloth, and N.O. Bouvin. 2000. "Open Hypermedia as User Controlled Meta Data for the Web." *Proceedings in 9th International World Wide Web Conference.*

Haan, B. J., P. Kahn, V.A. Riley, J.H. Coombs and N.K. Meyrowitz. 1992. "IRIS Hypermedia Services." *Communications of ACM.* Vol. 35(1): 35-51.

Halasz, F. 1988. "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems." *Communication of the ACM.* Vol. 31(7): 836-855.

Halasz, F., and S. Mayer. 1994. "The Dexter Hypertext Model." *Communication of ACM.* Vol. 37(2).

HyTime. "Information Technology - Hypermedia/Time-based Structuring Language (HyTime)." http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html, retrieved on June 2003.

Iannella, R., H. Sue, and D. Lenong. 1996. "BURNS: Basic URN Service Resolution for the Internet." http://archive.dstc.edu.au/RDU/reports/Apweb96, retrieved on June 2003.

Iksal, S., and S. Garlatti. 2001. "Revisiting and Versioning in Virtual Special Reports." *Third Workshop on Adaptive Hypertext and Hypermedia, 12th ACM Conference on Hypertext and Hypermedia.*

Ingham, D., S. Caughey, and M. Little. 1996. "Fixing the 'Broken-Link' Problem: the W3Objects Approach." *Proceedings of Fifth International World Wide Web International Conference.*

Kacmar, C.J., and J.J. Leggett. 1991. "Proxhy: a Process-oriented Extensible Hypertext Architecture." *ACM Transaction on Information Systems.* Vol. 9(4): 399-419.

Kacmar, C. 1995. "A Process Approach for Providing Hypermedia Services to Existing." *Non-hypermedia Applications, Journal of Electronic Publishing: Organization, Dissemination and Design.*

Maglio, P., and S. Farrell 2000. "LiveInfo: Adapting Web Experience by Customization and Annotation." *Proceedings of the First International Conference on Adaptive Hypermedia and Adaptive Web-based Systems.*

Manic. http://manic.cs.umass.edu/~stern/manic.html, retrieved on July 2002.

Martin, P., and P. Eklund. 1999. "A Key for Enhanced Hypertext Functionality and Virtual Documents: Knowledge." *Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference.*

Mayfield, J. 1997. "Two-Level Models of Hypertext." *Intelligent Hypertext: Advanced Techniques for the World Wide Web* 1326: 90-108.

McCracken, D. 1984. "Experience with the ZOG Human-Computer Interface System." *International Journal of Man-Machine Studies*. Vol. 21: 293-310.

Millard, D.E., L. Moreau, H.C. Davis, and S. Reich. 2000. "FOHM: a Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains." *Proceedings of Hypertext 2000 Conference*: 93-102.

Miller, J., P. Resnick, and D. Singer. 1996. "Rating Services and Rating Systems (and the Readable Descriptions), Version 1.1." http://www.w3.org/TR/REC-PICS-services, retrieved on May 2003.

Nelson, T. 1965. "A File Structure For the Complex, the Changing, and the Indeterminate." *Proceedings of the ACM National Conference*: 84-100.

OHSWG. 1997. http://www.csdl.tamu.edu/ohs/, retrieved on May 2003.

OHS.2002. http://www.cs.umd.edu/ht02/workshops/openhypermedia.shtml, retrieved on May 2003.

OpenURL. The OpenURL Standard. "The OpenURL Framework for Context-Sensitive Services." http://library.caltech.edu/openurl/PubComDocs/StdDocs/Part1-PC-20030513.pdf, retrieved on May 2003.

Ossenbruggen, J., L. Hardman, and L. Rutledge. 2002. "Hypermedia and the Semantic Web: A Research Agenda." *Journal of Digital Information*. Vol. 3(1).

Paskin, N. 1999. "Toward Unique Identifier." http://www.ieee.org/portal/cms_docs/pubs/proceedings/proc071999.pdf, retrieved on April 2003.

Pearl, A. 1989. "Sun's Link Service: A Protocol for Open Linking." *Proceedings of Hypertext'89*: 137-146.

Rada, R. 1991. *Hypertext: From Text to Expertext*. McGraw Hill Publishers.

Ranwez, S., and M. Crampes. 1999. "Conceptual Documents and Hypertext Documents are two Different Forms of Virtual Document." *Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference*.

Rao, U., and M. Turoff. 1990. "Hypertext Functionality: A Theoretical Framework." *International Journal of Human-Computer Interaction*.

RDF. http://www.w3.org/RDF/, retrieved on July 2003.

Rizk, A., and L. Sauter. 1992. "Multicard: an Open Hypermedia System." *Proceedings of ECHT'92, ACM Press*: 4-10.

Rosenberg, J. 1996. "The Structure of Hypertext Activity." *Proceedings of ACM Hypertext '96.*

Schnase, J.L., J.J. Leggett, D.L. Hicks, P.L. Nurnbery, and J.A. Sanchez. 1994. "Open Architecture for Integrated, Hypermedia-based Information Systems." *Proceedings of the 26th Annual Hawaii International Conference on System Sciences.* Vol. 3: 386-395.

SMIL. http://www.w3.org/TR/REC-smil/, retrieved on May 2003.

Stotts, P.D., and R. Furuta. 1989. "Petri-net-based Hypertext: Document Structure with Browsing Semantics." *ACM Transactions on Information Systems (TOIS).* Vol. 7(1): 3-29.

SVG. http://www.w3.org/TR/SVG11/, retrieved on July 2003.

Thistlewaite, P. 1997. "Automatic Construction and Management of Large Open Webs." *Information Processing and Management.* Vol. 33(2): 161-173.

Trigg, R., and M. Weiser. 1986. "TEXTNET: A Network-Based Approach to Text Handling." *ACM Transactions on Information Systems.* Vol. 4(1): 1-23.

URN. http://www.ietf.org/rfc/rfc2141.txt, retrieved on May 2003.

Van de Sompel, H., and P. Hochstenbach. 1999. "Reference Linking in a Hybrid Library Environment, Part 2: SFX, a Generic Linking Solution." *D-Lib Magazine.* April, 1999

Van de Sompel, H., and O. Beit-Arie. 2001(1). "Generalizing the OpenURL Framework beyond References to Scholarly Works: the Bison-Fute Model." *D-Lib Magazine*, March 2001.

Van de Sompel, H., and O. Beit-Arie. 2001(2). "Open Linking in the Scholarly Information Environment Using the OpenURL Framework." *D-Lib Magazine.* August 2001.

Van Harmelen, F., P.F. Patel-Schneider, and I. Horrocks. 2001. "Reference Description of the DAML+OIL." http://www.w3.org/TR/daml+oil-reference, retrieved on March 2003.

W3C. http://www.w3.org/, retrieved on May 2002.

Watters. C., and M. Shepherd. 1999. "Research Issues for Virtual Documents." *Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference.*

Weibel, S. 1995. "Metadata: The Foundations of Resource Description." http://www.dlib.org/dlib/July95/07weibel.html, retrieved on August 2002.

Wiil, U. K., and J.J. Leggett. 1996. "The HyperDisco Approach to Open Hypermedia Systems." *Proceedings of the 7th ACM Hypertext Conference*: 140-148.

Wilkinson, R. 1999. "Automatic Link Generation." *ACM Computing Survey*. Vol. 31(4).

XHTML. "XHTML: The Extensible Hypertext Markup Language." http://www.w3.org/TR/xhtml, retrieved on March 2003.

XLink. "XML Linking Language (Xlink)." http://www.w3.org/TR/xlink, retrieved on March 2003.

XML. "XML: the eXtensible Markup Language." http://www.w3.org/XML/, retrieved on March 2003.

XML-Schema. "XML Schema." http://www.w3.org/TR2000/CR-xmlschema-1-20001024, retrieved on March 2003.

XPointer. "XML Pointer Language (XPointer)." http://www.w3.org/TR/xptr, retrieved on May 2003.

XSL. "XSL: the eXtensible Style Language." http://www.w3.org/Style/XSL/, retrieved on May 2003.

XSLT. "XSL Transformation(XSLT)." http://www.w3.org/TR/xslt, retrieved on May 2003.