Supporting Virtual Documents in Just-in-Time Hypermedia Systems

Li Zhang[1]
lxz9848@oak.njit.edu

Michael Bieber[2]
bieber@oak.njit.edu
web.njit.edu/~bieber/

David Millard[3]
dem@ecs.soton.ac.uk
www.ecs.soton.ac.uk/~dem/

Vincent Oria[1]
oria@cis.njit.edu
web.njit.edu/~oria/

[1]Computer Science Department
[2]Information Systems Department
College of Computing Sciences
New Jersey Institute of Technology
University Heights, Newark, NJ 07102 USA

[3]School of Electronics & Computer Science,
University of Southampton,
Southampton, SO17 1BJ, UK

ABSTRACT

Many analytical or computational applications, especially legacy systems, create documents and display screens in response to user queries "dynamically" or in "real time". These "virtual documents" do not exist in advance, and thus hypermedia features must be generated "just in time" - automatically and dynamically. Additionally, the hypermedia features may have to cause target documents to be generated or re-generated. This paper focuses on the specific challenges faced in hypermedia support for virtual documents of dynamic hypermedia functionality, dynamic regeneration, and dynamic anchor re-identification and re-location. It presents a prototype called JHE (Just-in-time Hypermedia Engine) to support just-in-time hypermedia across third party applications with dynamic content, and discusses issues prompted by this research.

Categories and Subject Descriptors: H5.4 (Hypertext/hypermedia)

General Terms: Design

Keywords

Dynamic hypermedia functionality, Just-in-time hypermedia, Virtual documents, Dynamic regeneration, Re-location, Re-identification, integration architecture

1. INTRODUCTION

Analytical and computational applications, especially legacy systems, typically generate documents and screens dynamically in response to user queries. Such "virtual documents" are instances that only exist when the user visits them. When the user closes the window, these instances are gone. For these, hypermedia anchors and destination nodes must be generated "just-in-time"—automatically and dynamically. Furthermore, the hypermedia features leading to virtual documents often cause them to be generated or re-generated.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DocEng'04, October 28-30, 2004, Milwaukee, Wisconsin, USA.
Copyright 2004 ACM 1-58113-938-1/04/0010...$5.00.

Virtual documents require an entirely new level of hypermedia support. When traversing links to them, they need to be regenerated. When the user issues a query that creates them anew, the hypermedia system needs to recognize ("re-identify") them to locate old anchors over the documents or elements within them, even if the contents have shifted. When these elements appear as components within other virtual documents, the hypermedia system needs re-identify the elements and to re-locate anchors for them.

As an example, 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. She knows that she will need to include the calculation results and comments. A few days later when preparing her final report, she wishes to return to them without having to remember the input parameter values for each and then manually re-performing each calculation. So she creates a bookmark to the display screen containing each calculation result before closing the window. Invoking each bookmark later causes the sales support application to re-execute its calculations automatically, and the "just-in-time" hypermedia system to relocate her comments in the application's newly re-generated display.

Figure 1 illustrates such a parameter screen from our current prototype, which provides just-in-time hypermedia support to an analysis application at NASA. If the user bookmarks the query results, how will the hypermedia system know the parameters to regenerate it? A primary goal is that analytical systems not be altered to integrate hypermedia support, so we cannot simply require developers to reengineer their code to support us by storing and supplying these parameters on demand. Some Web systems regenerate analysis screens by storing all relevant parameters in the virtual document's URL, and this would solve the problem completely. 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. The parameters could be stored in cookies when permitted. But this solution would be local to a single workstation. Furthermore, storing all of the parameters for an active analyst over time would be excessive. This is just one of many problems with hypermedia support of virtual documents.

This research provides a general solution to supplementing virtual documents from third party applications with just-in-time hypermedia support, utilizing dynamic regeneration, re-

identification and re-location. We present the requirements and architecture for a Just-in-time (JIT) Hypermedia Engine (JHE), and discuss several of the interesting issues that it surfaces.

We begin in §2 with related research. §3 compares dynamic hypermedia functionality with static hypermedia functionality. In §4 and §5 we discuss some major challenges encountered in "just-in-time" hypermedia systems: dynamic regeneration, re-location and re-identification. In §6 we introduce our JHE architecture and prototype supporting "just-in-time" hypermedia for virtual documents and analytical applications. §7 concludes with a discussion of our future research and contributions.

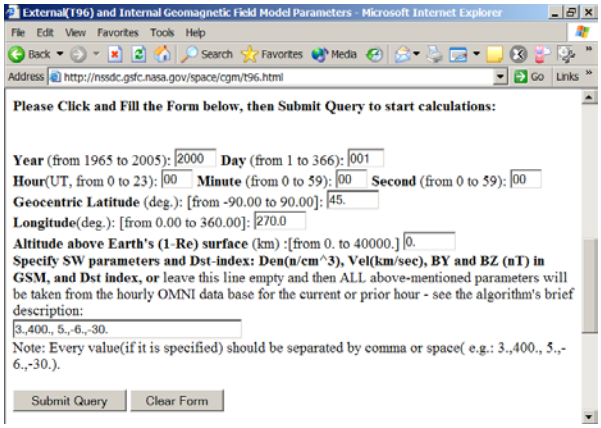

Figure 1: NASA query page.

2. RELATED WORK

Very little work has been conducted on dynamic regeneration and other aspects of "just-in-time" hypermedia. Related work includes virtual documents, open hypermedia, and object identification.

2.1 Virtual Documents

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 and Shepherd [1] give a number of interesting research issues about these virtual documents, including generation, search, revisiting, versioning, authentication, reference and annotation. Here we expand on some of these.

Reference: In our research, we generate and maintain a unique ID for each virtual document, or each version of the virtual document. The JHE maintains information about the virtual documents. When the virtual document is needed, the JHE uses the unique ID to look for the virtual document information, and get the commands and parameters to recreate it if regeneration is necessary.

Generation: 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 [3] define virtual documents as a non-organized collection of Information Bricks (IB), associated with methods allowing the generation of a finished IB sequence. In our 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 our research has similar purpose.

Versioning: Users want to be able to return to a bookmarked version of a virtual document, and to navigate forward and backward in time through changes to that virtual document. Some systems such as WikiWeb [14] can recreate Web pages by revisiting the same URL and stores page differences in database, so that it can track the Web page modifications.

Some research has been conducted on these issues Caumanns [4] deals with the creation of dynamic documents by predefined templates or knowledge. Iksal and Garlatti [5] describe an adaptive web application system, which generates adaptive virtual documents by means of a semantic composition engine based on user models. Our research differs from [5] in that, in our research the virtual documents are created by analytical applications; we have no way to control their generation, and the regeneration also depends on the analytical applications. What we particularly are interested in is how to dynamically regenerate the virtual documents, and how to determine that the regenerated one is the same one as before.

Our architecture, however, does not necessarily lead to the regeneration of old versions (unless the application system specifically offers this feature on its own). Just as hypermedia backtracking differs from "undoing" since it takes the user to the current state of a previously visited location, in this research following a bookmark, etc., leads to the current state of the target document (although viewed in a previous context, see §4.3).

2.2 Open Hypermedia Systems

Open Hypermedia Systems (OHS) traditionally keep links separate from content and combine them "just-in-time", to produce a viewable hypermedia document.

Separating links from content means that the links can be managed in a more sophisticated manner, but within OHS research there has also been an emphasis on supporting external applications [9], and dealing with structures beyond the traditional navigational link [19], including virtual documents].

However, OHSs that deal with external applications typically assume that those applications have static content. In addition, OHSs that do deal with dynamic content [7] tend to include virtual documents into their own extended set of supported structures. This has the advantage that link and virtual document structures are managed consistently, but bypasses many of the problems of regeneration that occur when dealing with dynamic content from third party applications (as described in §4).

Our JHE system is essentially an OHS that attempts to provide navigational link support to third party applications that have dynamic content. As such it is very similar in architecture to the Distributed Link Service (DLS) [9] but tackles problems more familiar to structural and contextual systems such as Auld Linky [16] and Construct [6].

2.3 Object Identification and Internal Document Structures

An anchor is a selected element (words, a sentence, a paragraph or anything else) in a document, including the entire document itself. After the user creates an anchor, the JHE should remember its selection in the document. The next time its underlying element appears, the JHE should be able to recognize it and associate

the proper hypermedia functionality with it. An anchor could have three different degrees of scope: specific, local and generic [9].

Specific: only applies to the particular element in the particular document.

Local: applies to all the elements with same name in the particular document.

Generic: applies to all elements with the same name in all documents.

There are several ways to express the location inside a text file. For example, the HyTime [8] standard allows users to express anchors within objects. An anchor may be expressed by:

- Naming - e.g., an SGML entity 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.

With text files, Microcosm's main approach for addressing an anchor is using byte offset [9]. Similarly, the Open Hypermedia Protocol (OHP) [15] uses byte offsets (both forward and backward count) to address locations. This can lead to the file editing problem, causing possible link inconsistency. When a static file is edited, links associated with it 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 re-locate 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 can not 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 re-located to this new location. This works well but can not guarantee 100% correctness.

Our JHE prototype uses XPath and XPointer for internal document addressing. XPath [16] is a language to address locations inside XML documents based on document structures. XPath models an XML document as a tree of nodes. XPointer [17] is the language to be used as the basis for a fragment identifier for any URI reference that locates an XML resource. Based on Xpath, it supports addressing into internal structures of XML documents. As XPath/XPointer is based on document structure, location expressions can be flexible and accurate if document content changes frequently as long as document structure does not change.

3. DYNAMIC VS. STATIC HYPERMEDIA FUNCTIONALITY

Analytical or computation applications currently can take little advantage of hypermedia functionality on the Web. In large part this is due to the predominantly read-only nature of Web applications today, in that most applications do not facilitate functionality that requires the user to add link anchors. But even if this were facilitated, most hypermedia functionality only supports static documents for embedding link anchors and target nodes that do not require parameters to generate (unless all parameters are held within the URL - see §4).

Hypermedia functionality includes: structuring functionality (*global and local overviews; trails and guided tours; node, link and anchor typing*), navigational functionality (*structure-based query, history-based navigation and bi-directional linking*), and annotation functionality (*user-declared links, comments and bookmarks or favorites*) [2]. Dynamic hypermedia functionality applies these over the virtual analytical space of a computational system and the virtual documents generated within it.

Any dynamic hypermedia created in a "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:

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

(b) Once a node is regenerated, a JIT hypermedia system must re-locate 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.

(c) When analytical applications generate new query results, a JIT hypermedia system should be able to re-locate and re-identify the anchors for the same elements appearing in other query results.

(d) 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, and requires dynamic regeneration, re-location and re-identification.

We now discuss these three important issues in more detail, and describe some of the specific challenges that are faced by our JHE system when dealing with independent (third party) content published dynamically on the web.

4. DYNAMIC REGENERATION

It has long been thought that managing virtual structures is an important part of hypermedia system functionality [20]. Whenever dynamic hypermedia links leads users to a virtual document, a hypermedia system should be able to arrange for its regeneration without relying on the user to reenter any parameters.

4.1 Regeneration and Context

Virtual documents are regenerated using some user specific metadata that we call *context* (for example, parameters located in a web user's cookies). Other elements of the documents may also be dynamic but independent from user context (e.g. the current price of stock). The hypermedia system does not effect the regeneration of these elements, but does have to deal with them when placing anchors (this is discussed in Section 5)

When regenerating a virtual document, there are two contexts to consider: the user's original context when they first flagged the document as a destination, but also the user's current context, which could also influence the regeneration. Three general cases are possible [23].

(a) The system could keep the old context, and simply regenerate the pages using the original parameters. This is almost equivalent to displaying a cache of the original page, however the generation processes are re-invoked, which might be important for logging and diagnostics

(b) The system could adapt the page to the new user context. For example, a bookmarked analysis of a violent news event should not include graphics of a sensitive nature if the new user context is a minor. This might be of particular importance when data in the document is sensitive in some way and security is thus a consideration.

(c) The system could merge contexts, determining which settings and parameters of the original user context should "override" those of the current user context. For example, the original analyst may believe a particular photo to be so central to the piece that it should be viewed by minors.

In our current research we are concentrating on the first case, focusing on the challenge of maintaining parameters and other aspects of the original context. We are not developing adaptive applications; any content adaptation is the responsibility of the analytical application our JIT hypermedia engine supports. Customizing the set of links generated is the focus of other research underway but outside the scope of this paper [11]. More flexible support of contexts in general is an issue for future research.

4.2 Dynamic Regeneration Requirements

Given this assumption about consistent context, our JHE hypermedia system has the following requirements for dynamic regeneration:

(a) A unique and persistent identifier for each virtual document should be generated and recorded. Every time the user traverses a bookmark or other link, she should return to the same virtual document, otherwise, these bookmarks or links will no longer be valid. The JHE 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 JHE should create a unique ID for this document. When the user tries to revisit it, the application system should use the same ID, retrieve the corresponding commands and parameters from its database, and re-execute the commands with these parameters to dynamically regenerate the virtual document.

(b) Whenever the JHE receives a document for display, there should be some way for it to reorganize whether the application 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 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 §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.

(c) A static document that has been edited also has the re-identification problem [9]. 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 JHE has to re-identify whether this document is the same one as before.

4.3 Regeneration Procedure

The regeneration procedure of our JHE system 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 JHE records the virtual document information and the link information. The virtual document information contains a unique identifier for each document, generation command/parameters, etc. If a one-way link is from a virtual document A to a virtual document B, then the link information contains the virtual document identifier A as the source and identifier B as the destination.

(2) When the user traverses that link to revisit the virtual document, the JHE retrieves the link information. From the link information, it finds out the destination identifier B, which points to the virtual document B's information. From this, the hypermedia system gets the necessary command information for its original application to regenerate it (including any parameter information to re-execute the commands).

(3) The JHE sends the command information to the underlying analytical application.

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

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

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

The commands which JHE sends to the application are 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 JHE 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 JHE invokes 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 JHE uses the sameness rules discussed earlier to validate this.)

Alternatively, when integrating an application with the JHE (see §6), 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 JHE could pass all required parameters

Figures 2 and 3 show screenshots from the JHE prototype. When the user clicks on the "add bookmark" button in the lower left screen, the system allows him or her to create a bookmark, and as part of this, specify the regeneration criteria in the lower right screen. Choosing that bookmark anywhere (from the bookmark list shown in Figure 3) will cause the document in Figure 2 to be regenerated.

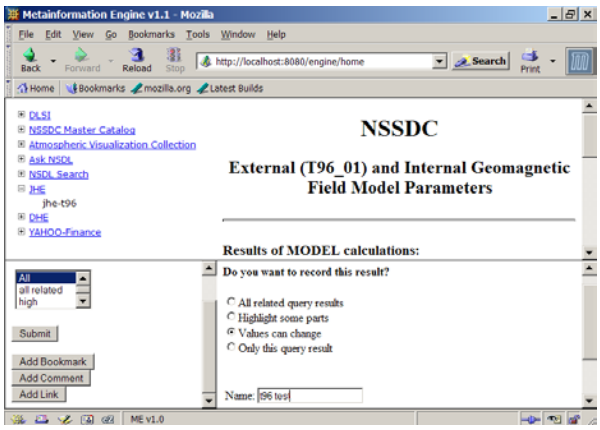

Figure 2: Regeneration Criteria

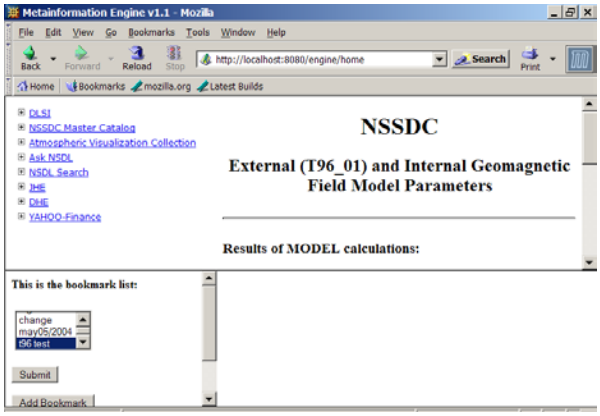

Figure 3: Choosing a Bookmark

4.4 Parameters for Regenerating Virtual Documents

In order to regenerate a virtual document, the JIT hypermedia system should record: dynamic link information, virtual document information, document generation information and document revalidation 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.

Source anchor identifier: the anchor identifier from which the one-way link starts.

Destination type: the link destination could have three types: URL, anchor and document.

Destination identifier: depending on the destination type, this identifier may contain URL, anchor identifier and virtual document identifier.

User information: who created this link.

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

Virtual Document Information

Virtual document identifier: should be unique and persistent.

Version: if the file content or structure changes, 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 the newly-regenerated virtual document is same as the virtual document visited previously.

Content: some critical sections of the file or some element values should be kept for revalidating the document. What kind of content to keep depends on which criteria the JIT hypermedia system or the user chooses. Corresponding to the "sameness" criteria described in §4.2, JHE keeps the following content information from level 1 to level 4:

Level 1: Keeps an exact copy of the document.

Level 2: Keeps exact structure information of the document.

Level 3: Keeps the critical section content.

Level 4: Does not need to store any content of the document.

Generation Information

The following parameters are maintained as part of the *regeneration rules*.

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.

5. RE-LOCATION / RE-IDENTIFICATION

Whenever an application generates or regenerates a document, the JHE needs to determine whether it has encountered that document before (i.e., re-identify it). This is because users may have declared hypertext anchors over the document as a whole (as opposed to over a fragment within the document's content), which the hypermedia system should re-locate so users again can access these anchors. In addition, the virtual document's content could contain elements over which users may have declared hypertext anchors—either within a former appearance of this document or within different documents that contained some of the same elements

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

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

(b) 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.

(c) 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. How it decides that an element is the same one depends on some criteria, which will be discussed later in detail.

(d) When the user creates a link, she needs to specify 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 [9]. This is called *re-location granularity*.

(e) As with a static hypermedia system, the JHE 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.

(f) Re-location and context: As with dynamic regeneration, users may have "assumed" their current context when creating anchors and not thought about whether anchors should appear for users with different contexts. While we allow users to specify whether an anchor only counts for the current document or for any document in which its content appears, we do not currently allow the user to specify the anchor's context in more detail. This is a subject of future research.

5.1 Procedure for Re-location and Re-identification

The re-location and re-identification procedure of our JHE system has the following steps:

(1) The first time a user selects and marks an anchor on the screen, the JHE records the anchor information in an external anchor database. Anchor information includes virtual document identifier, location, selection content, granularity, etc. Granularity means different degrees of anchor scopes. An anchor could appear at a particular location in a particular document (specific), on the same element anywhere in a particular document (local), or could appear in any document that has the same element (global).

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

(3) For each anchor inside this document (local and specific anchors with same document identifier), the JHE 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. The JHE should also locate all 'global' anchors that may exist in this document no matter what document identifier it has.

(4) For the re-located anchor, the JHE should re-identify that the newly-generated element is the same one that was selected as an anchor. For virtual elements, we allow users to specify the "sameness" criteria. If a user's comment depends on the exact value of the element, then the system should store and compare the element's value. Otherwise, the system does not need to store and compare it.

(5) After the anchored virtual elements are re-located and re-identified successfully, the JHE, upon demand, can look for those hypermedia functionalities associated with these elements in the database, and associates them with anchors.

5.2 Parameters for Re-location and Re-identification of Virtual Elements

In order to re-locate and re-identify virtual elements (anchors), the JHE should record anchor information and re-identification criteria, described as follows:

Anchor Information:

Anchor identifier: this should be unique and persistent.

Title: name or label.

Granularity: name scope of the anchor, including whether it is specific, local and global.

Virtual document identifier: in which document this anchor originally resided.

Location: an expression of the anchor location according to the document structure.

Selection: the selected and marked content from the document.

Re-identification criteria: "sameness" criteria, whether the user allows the value to change.

6. The JIT Hypermedia Engine (JHE)

In this section we present our architecture and prototype for a JIT Hypermedia Engine (JHE), as well as how information flows between applications and JHE. JHE extends our prior work with the Dynamic Hypermedia Engine (DHE).

6.1 The Dynamic Hypermedia Engine

To supplement analytical applications with hypermedia functionality, the Dynamic Hypermedia Engine (DHE) [10] intercepts documents and screens as they are about to be displayed on the user interface 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 prompts DHE to send a command (e.g., a query) to the target analytical application, causing it to generate a virtual document containing the calculation results. The target applica-

tion 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 [10, 11], described below.

6.2 Just-in-time Hypermedia Engine Architecture

While DHE dynamically generates link anchors and links for virtual documents, it currently does not support re-identification, re-location or regeneration. The "Just-in-time Hypermedia Engine" (JHE) extends DHE's architecture to supplement analytical applications with hypermedia functionality in this "just-in-time" environment. JHE's architecture is shown in Figure 4. In what follows we describe identifiers, the architectural components and the information flow within JHE.

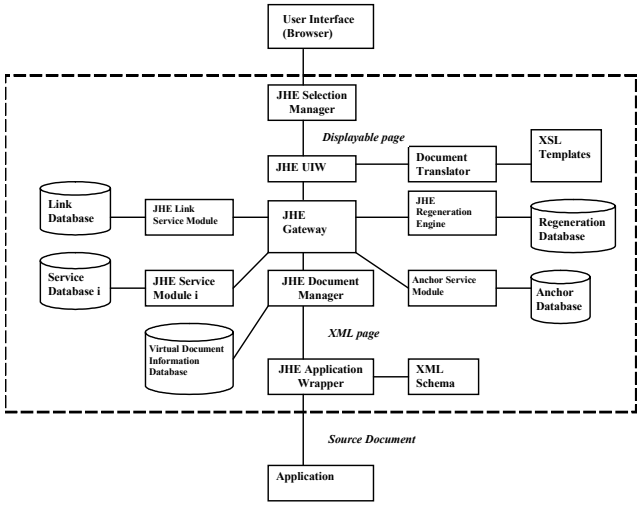

Figure 4: Just-in-time Hypermedia Engine (JHE) Architecture

6.2.1 Identifiers

We use XPATH expressions to address the anchors inside the document, and XPOINTER to express arbitrary selections. The unique identifiers (ID) of virtual documents and anchors are generated and maintained by JHE using the following format:

Virtual document identifier = (application ID, command ID, parameter set ID)

Anchor identifier = (document ID, location ID)

Parameter set = (parameter1, parameter 2, … parameter n, version number)

The parameter set ID is a unique number which represents the parameter set. The location ID is a unique number which represents the location inside a document. The parameter set carries a list of parameter name and value information, and is unique in an application. However, in practice, this parameter list is too long to use as an identifier, thus we use a unique number as the ID for the parameter list. Anchor location is generated dynamically when it is placed in the virtual document. It is unique inside the document. The location expression is usually a long string, so we use the location ID instead of the location expression itself in the anchor identifier.

JHE groups together the identifiers within each application and within documents they generate. The actual command and parameter values will be filled in at execution time by JHE, but the command format or skeleton is predefined by the application developers at the time they declare their application wrappers (see below). JHE assigns each un-instantiated (not filled-in) command "skeleton" its own ID. For example, a database application may support SQL queries, but each query follows a well-defined format (that can be instantiated in an infinite number of ways).

Usually a parameter set is composed of multiple parameters and a version number. When one of the parameter's metadata changes or when a new parameter appears, depending on user settings this virtual document can be treated as a new document (this corresponds to "sameness" criteria level #3 from §4, and a new version number is added to the parameter set) or treated as the same virtual document (this corresponds to "sameness" criteria level #4).

6.2.2 Component Functionality

JHE's architecture from Figure 4 uses many of the same component modules as DHE. JHE is middleware that integrates many applications by application wrappers. The dotted box in Figure 4 contains all JHE components. All components (except the JHE selection manager) run on the server side. Components underlined below are entirely new to JHE and support re-identification, relocation and dynamic regeneration.

User Interface (UI): Usually runs on the user's computer to display documents, links and JHE commands (e.g., a Web browser providing a Web interface for the underlying analytical system).

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 records location information. Many Web browsers allow users to select text from the screen, and the browsers record the location information for each selection. Our prototype uses Mozilla [21] as the default Web browser, utilizing its XpointerLib [22] interface as the Selection Manager.

Document Translator (DT): translates a page in JHE's internal XML format to an HTML page for display according to the XSL template file.

User Interface Wrapper (UIW): handles communications between the UI and Gateway, and implements the application commands displayed in UI.

Application: A computer application external to the hypermedia system. In this research we focus on analytical applications that dynamically generate virtual documents as the result of user queries.

Application Wrapper (AW): Manages the communication between the Gateway and the application system. It parses the application's screens and documents to identify the "elements of interest" that JHE will make into link anchors [11]. It also translates these screens and documents into XML pages for JHE internal processing.

Regeneration Engine (RE): Serves three important functions: First, RE gets the necessary commands and parameters from the RE database for regeneration 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): Looks for the hypermedia components 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.

Service Module (SM): Enables users to add and retrieve hypermedia functionality (such as links, annotations, guided-tours) for selected anchors and stores hypermedia information into the database. Each type of hypermedia functionality has its own SM and service database (such as the **Anchor Service Module**, which stores anchor information in a database and parses anchor information to find an anchor's exact and absolute byte offset within a document).

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

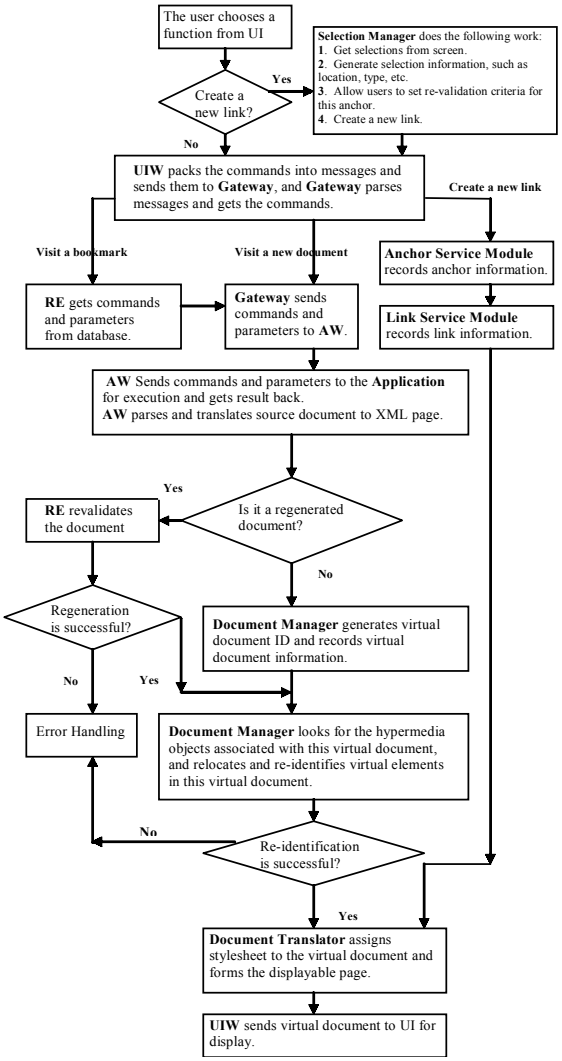

Figure 5: JHE Information Flow

6.2.3 Information Flow

Information flows through the architecture as follows (shown in figure 5).

(a) Displaying a Document for the First Time

When the user chooses an application function within its display, the JHE 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.

After the AW successfully translates the source document into an XML document, the Document Manager generates a unique identifier and records virtual document information into the database. Then this virtual document is sent back to the browser for display.

(b) Creating a Hypermedia Construct

Several differences exist between creating a new hypermedia construct (e.g., an anchor, user-declared link or a bookmark) in JHE over the many other hypermedia and open hypermedia systems. Constructs such as anchors are placed within virtual documents. Anchors, for example, will potentially need to be re-identified and re-located within any virtual document displayed. Any document or screen the construct leads to must be potentially re-generated and its endpoint within that document then re-located. Also, when users declare a construct, they can set re-validation (sameness) criteria. Figure 6 shows an example of adding a comment for the selected text "2004" on the screen. When a user clicks the "add comment" button, the lower left screen displays information on how to create a comment. Then the user selects some text from screen, enters information about the comment, selects the anchor granularity and submits the comment information to the JHE system. All related information will then be stored in the database.

When the user adds a hypermedia construct that uses an anchor, she first makes a selection on the screen and then chooses the hypermedia feature. The Selection Manager gets the selection and generates a unique identifier for the selection and records the details of the selection (position, content, type, etc.). The UIW passes appropriate details about the anchor and hypermedia feature in a message to the Gateway.

The Gateway asks the Anchor Service Module to generate a unique identifier for the anchor and records the anchor information in the database. The Document Manager asks the appropriate Service Module (e.g., Link Service Module) to generate a unique identifier for the feature and records appropriate parameters. Then the anchor on the screen is highlighted.

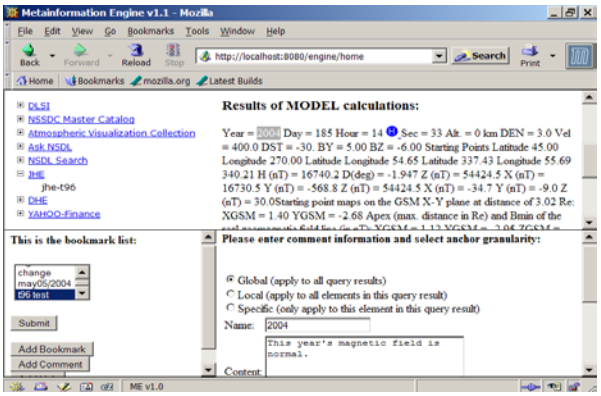

Figure 6: Create a JHE comment

(c) Revisiting a Document

Suppose that later, after the document has been closed, the user wishes to revisit it, e.g., from a bookmark. The UIW sends the bookmark information back to the Gateway, which asks the Regeneration Engine to recreate the execution result. The Regeneration Engine will do the following work. It retrieves the virtual document ID from the bookmark using this virtual document ID. It gets the regeneration information from the database. Then the RE asks the AW to direct the application to re-execute the commands and RE receives the execution result as a XML document (translated by the AW). Using the re-identification criteria and virtual document history information, RE compares the information of the newly-generated virtual document with the history information to revalidate the new document, incorporating some of the techniques used by Davis [9]. If this document is same as the previous one according to the revalidation criteria, then this regeneration is successful. The virtual document sameness criteria are applied here. After this, the Gateway will retrieve all the related hypermedia information for this document, re-locating each anchor in the document, and re-identifying it if it is the same as before. For those anchors that are successfully re-located and re-identified, the Gateway sends the related information to the UIW and forms the displayable HTML page. Figure 7 shows a regenerated document with some selected anchors highlighted. A small icon 'H' marks each anchor. When the user clicks the icon, it will pop up a window to display related hypermedia construct information, such as comments and links.

6.2.4 Prototype Status

Currently, our JHE prototype provides just-in-time hypermedia support over two computational applications, AskNSDL [12] and NSSDC [13]. The AskNSDL service is a system where users can post questions, which experts in the question domain will answer. One needs to be registered to ask questions but anyone can browse the database of questions and answers later. The NSSDC (NASA's National Space Science Data Center) is the permanent archive for most NASA astrophysics data. It is housed at the Goddard Space Flight Center in Maryland. In these two systems, users can enter parameters (such as the date) or select from a list of parameters and choose commands or menus, then submit the request. Documents are generated dynamically and then sent back to end users. Document contents vary according to different input parameters.

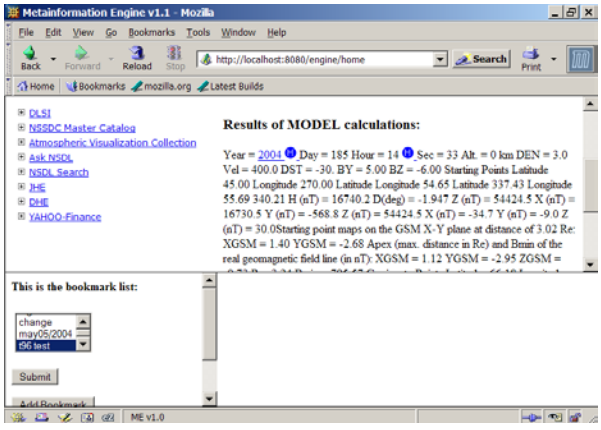

Figure 7: Regenerated document with relocated and re-identified anchors

7. CONCLUSION

This paper examines research issues for "just-in-time" hypermedia, in particular with regard to supporting dynamic third party applications; this includes the dynamic regeneration of virtual documents, re-identification of re-generated virtual documents, and re-location and re-identification of virtual elements in virtual documents. We have designed and are refining a "just-in-time" hypermedia engine sitting between the browser and the underlying analytical application to supplement hypermedia functionality for applications. We believe our work will also benefit other research fields which share some of the challenges, such as virtual documents and adaptive hypermedia.

Having a prototype JIT hypermedia engine that supports third party dynamic content will allow us to pursue several different strands of research involving dynamic hypermedia functionality. We intend to evaluate the effectiveness of dynamic hypermedia support. We shall conduct experiments, in which users perform assigned tasks with and without the capacity of dynamic hypermedia functionality. We shall investigate extending our approach to incorporate user profiles and document context, for regeneration, re-identification and re-location.

We intend to extend dynamic hypermedia functionality and just-in-time hypermedia support to virtual documents with non-textual contents.

We plan to explore automatically generating the XML templates for and performing an automatic metadata analysis over virtual documents, in order for wrappers to parse and (re-)identify elements. This will ease the job of writing a wrapper and registering every kind of document in advance. Related research on document templates and document structure analysis will be very helpful to this effort.

ACKNOWLEDGEMENTS

We gratefully appreciate funding support for this research by the United Parcel Service, the New Jersey Institute of Technology, and the National Science Foundation under grants IIS-0135531 and DUE-0226075.

REFERENCES

[1] Carolyn Watters and Michael Shepherd, "Research Issues for Virtual Documents", Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference, May, 1999, Toronto, Canada

[2] Michael Bieber, Fabio Vitali, Helen Ashman, V. Balasubramanian and Harri Oinas-Kukkonen, "Fourth Generation Hypermedia: Some Missing Links for the World Wide Web", Int. J. Human-Computer Studies, World Wide Web Usability Special Issue, (Eds.) S. Buckingham Shum & C. McKnight, 47(1): 31-65.

[3] Sylvie Ranwez and Michel Crampes, "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, May, 1999, Toronto, Canada,

[4] Jörg Caumanns, "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, May, 1999, Toronto, Canada

[5] Sebastien Iksal and Serge Garlatti, "Revisiting and Versioning in Virtual Special Reports", Third Workshop on Adap-

tive Hypertext and Hypermedia, 12th ACM Conference on Hypertext and Hypermedia, Arhus, Denmark, August 2001

[6] Construct, 2001.http://www.cs.aue.auc.dk/construct

[7] Christopher Bailey, Samhaa R. El-Beltagy and Wendy Hall, "Link Augmentation: A Context-Based Approach to Support Adaptive Hypermedia", 12th ACM Conference on Hypertext and Hypermedia, Arhus, Denmark, August, 2001

[8] HyTime, "Information Technology - Hypermedia/Time-based Structuring Language (HyTime)". http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html

[9] Hugh Davis, "Data integrity problems in an open hypermedia link service", Ph.D. thesis, Southampton University, 1995.

[10] Roberto Galnares, "Augmenting Applications with Hypermedia Functionality and Metainformation", PhD thesis, New Jersey Institute of Technology, Newark, NJ 07102.

[11] Catanio, Joseph, Nkechi Nnadi, Li Zhang, Michael Bieber and Roberto Galnares, "Ubiquitous Metainformation and the 'What You Want When You Want It' Principle," Journal of Digital Information, 2004.

[12] AskNSDL, http://www.asknsdl.org

[13] NASA's National Space Science Data Center (NSSDC), http://www.nssdc.nasa.gov/

[14] WikiWeb – Web Based Corporation Tools, http://www.wikiweb.com

[15] Davis, H.C., A. Lewis, and A. Rizk. "OHP: A Draft Proposal for a Standard Open Hypermedia Protocol." 2nd Workshop on Open Hypermedia Systems.

[16] "XML Path Language (XPath)", http://www.w3.org/xpath

[17] "XML Pointer Language (XPointer)", http://www.w3.org/xptr

[18] Jon Griffiths, David E. Millard, Hugh Davis, Danius Michaelides, and Mark J. Weal, "Reconciling Versioning and Context in Hypermedia Structure Servers." In Nürnberg, P. J., Eds. Proceedings of Metainformatics International Symposium, pp. 118-131, Esbjerg, Denmark, 2002.

[19] U. K. Wiil, P. J. Nürnberg, Evolving Hypermedia Middleware Services: Lessons and Observations, in: Proceedings of the 1999 ACM Symposium on Applied Computing (SAC '99), San Antonio, TX, 1999, pp. 427–436.

[20] Frank G. Halasz, Reflections on NoteCards: Seven issues for the next generation of hypermedia systems, in Proceedings of the ACM Conference on Hypermedia, North Carolina, USA, 1987, pp 345-365

[21] Mozilla Web Browser. http://www.mozilla.org

[22] XpointerLib. http://xpointerlib.mozdev.org/