Int J Digit Libr (2010) 11:155–168
DOI 10.1007/s00799-011-0065-9

Supplementing virtual documents with just-in-time hypermedia functionality

Li Zhang · Michael Bieber · Min Song · Vincent Oria · David E. Millard

Published online: 10 June 2011

Abstract Digital library systems and other analytic or computational applications 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 (links, comments, and bookmark anchors) must be generated "just in time"—automatically and dynamically. In addition, accessing the hypermedia features may cause target documents to be generated or regenerated. This article describes the specific challenges for virtual documents and dynamic hypermedia functionality: dynamic regeneration, and dynamic anchor re-identification and re-location. It presents Just-in-time Hypermedia Engine to support just-in-time hypermedia across digital library and other third-party applications with dynamic content, and discusses issues prompted by this research.

Keywords Dynamic hypermedia functionality · Just-in-time hypermedia · Digital libraries · Virtual documents · Dynamic regeneration · Re-location · Re-identification · Integration architecture

L. Zhang · V. Oria
Computer Science Department, College of Computing Sciences, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA

L. Zhang
e-mail: zhangli73@yahoo.com

V. Oria
e-mail: oria@cis.njit.edu
URL: http://web.njit.edu/~oria/

M. Bieber · M. Song (✉)
Information Systems Department, College of Computing Sciences, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA
e-mail: min.song@njit.edu
URL: http://web.njit.edu/~song/

M. Bieber
e-mail: bieber@njit.edu
URL: http://web.njit.edu/~bieber/

D. E. Millard
School of Electronics & Computer Science, University of Southampton, Southampton, SO17 1BJ, UK
e-mail: dem@ecs.soton.ac.uk
URL: http://users.ecs.soton.ac.uk/dem/

1 Introduction

Digital libraries often generate documents and screens dynamically in response to user searches. Furthermore, many animations and simulations stored in educational resource repositories generate documents and displays based on parameters that users input. Such "virtual documents" only exist when the user visits them. When the user closes the window, these virtual artifacts are gone.

Virtual documents require an entirely new level of "just-in-time" hypermedia support. When traversing links to them, they need to be regenerated. Regeneration requires the hypermedia system to recognize ("re-identify") them to place ("re-locate") link anchors, even if their contents have shifted (and thus their overall appearance has changed). When elements from one virtual document appear as components within another, the hypermedia system needs to re-identify the elements and to re-locate anchors within them.

Consider a researcher recursively refining a query within a digital library. Each successive search generates a refined list of journal articles. She or he may bookmark the names of potentially useful articles as well as authors who seem to share common research interests. Also, she or he may add a comment on authors who end up publishing research irrelevant to her or his interests or on articles with questionable results. Our researcher would want to see this comment on

every instance of the author's name or the article within any set of search results, any journal table of contents, any article's reference section, any digital library alert system, or any citation analysis. Following a bookmark on an author could either lead to the specific article where the bookmark was created or lead directly to a dynamically generated list of all articles the author has published. Following a bookmark to an intermediate or the final set of refined query results should re-execute that intermediate or final query dynamically without her or his having to remember the active set of parameters that generated either one of these originally.

Similarly, students may wish to bookmark, link to, or comment upon experts' answers within a question/answer digital library that is supported by an underlying database, such as AskNSDL [1]. Experts may add to or alter answers over time and users will want to access the latest results without re-entering parameters.

Many e-commerce and other analytic or computational applications display virtual documents with similar dynamic characteristics to digital libraries. As an alternate example, suppose an analyst wants to determine projected profits for different sales levels in her company. She or he performs the analysis within a sales support application, and makes comments (as hypertext annotations) on each resulting profit calculation. She or he knows that a few days later when preparing her or his final report, she or he will wish to return to them without having to remember the input parameter values for each and then manually re-performing each calculation, so she or he creates a bookmark to the results screen before closing the window. Invoking the bookmark later causes the sales support application to re-execute its calculations automatically, and the "just-in-time" hypermedia system to re-locate her or his comments in the application's newly re-generated display. This is analogous to a teacher preparing or students working with searches, animations, modeling, and other simulations in educational digital library systems.

Figure 1 presents a further example of a NASA system that accesses a geomagnetic field model (NSSDC [2]). Students or researchers may execute the model for several different sets of parameters. Following bookmarks placed previously on useful result sets should re-execute the model automatically with the corresponding set of original parameters. Suppose the student creates a comment on, or a link to or from a specific parameter on a specific analysis result. He or she may wish to see this comment or link anchor when that specific result is regenerated following a bookmark. Alternatively he or she may wish to see the comment or link anchor whenever that specific parameter has the same value, or perhaps in every display that contains that parameter regardless of its value.

The research we present can implement all of this dynamic hypermedia functionality within any digital library or analytic system that generates such virtual documents.

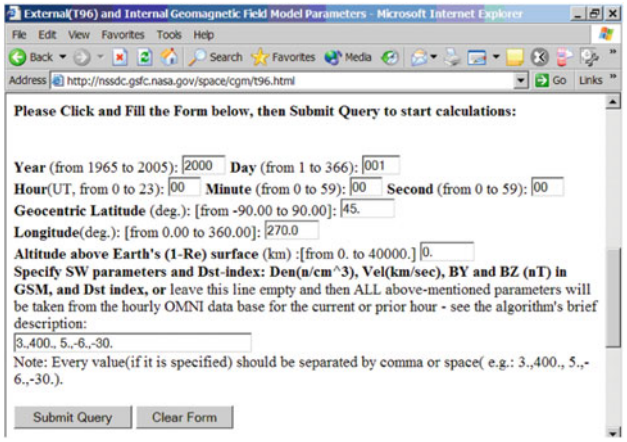

Fig. 1 NASA analysis input page

These examples raise several tricky issues. Consider NASA's modeling application in Fig. 1. If the user bookmarks the query results, how then will the hypermedia system know the parameters to regenerate it? One of our primary goals is that a digital library or analytic system should not be altered to integrate hypermedia support. This would be impractical; we cannot simply require developers of every digital library, business, or analytic system to re-engineer 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. However, many Web-based (as well as non-Web based) systems have no such mechanism. Other Web systems do not allow URLs with detailed parameters for security or other reasons, and may store the parameters in cookies instead. However, this solution would be local to a single workstation and storing all of the parameters for an active analyst over time would be impractical. This is just one of many problems with hypermedia support of virtual documents.

Our work contributes a generally applicable solution to supplementing virtual documents from digital libraries and other third-party applications with just-in-time hypermedia support, utilizing dynamic regeneration, re-identification and re-location, as well as a prototype. Such functionality should benefit digital library users such as students, teachers, researchers, analysts, other knowledge workers using dynamic analysis systems, and in general also should be available to any web page.

In this article we present the requirements and architecture for a Just-in-time (JIT) Hypermedia Engine (JHE), and discuss several of the interesting issues that these raise. JHE is middleware running between the user interface (such as a browser) and the application (the digital library or other analytic system). JHE's key strength is that it provides links and other hypermedia functionality to virtual documents from

third-party applications, rather than documents controlled by the hypermedia system itself.

We begin in Sect. 2 by looking at related research and in particular at how existing hypermedia systems have to compromise to cope with dynamic content. Sect. 3 presents an analysis of how dynamic hypermedia functionality differs from static hypermedia functionality in order to define some requirements for "just-in-time" hypermedia systems. In Sects. 4 and Sect. 5 we discuss the challenges in realizing those requirements, in particular: dynamic regeneration, re-location and re-identification. In Sect. 6 we introduce our JHE architecture and prototype. Section 7 concludes with a discussion of our future research and contributions.

2 Related work

Related work includes systems that deal with virtual documents (content that is generated at runtime), open hypermedia systems (which keep links separate from content), and standards that deal with object identification (locating smaller parts within a larger media file).

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 [3] present a number of interesting research issues about virtual documents, including:

- Reference: How do you reference a virtual document? Does the reference refer to the process of generation, the parameters and process together, or a particular instance of a generated document? These last two may be different if some dynamic part of the document is not dynamic because of the parameters (rather, for example, such as something that depends on the time of viewing).
- 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 other application. Ranwez and Crampes [4] define virtual documents as a non-organized collection of Information Bricks (IB), associated with methods allowing the generation of a finished IB sequence. For our research we will mainly be considering virtual documents that are created by an application as the result of a user search or query.
- Revisiting: Users expect that documents found once will be available on a subsequent search. The notion of a bookmark does not apply to virtual documents in a normal, retrieval sense. Bookmarks for virtual documents need enough information to recreate the document as it was.
- Versioning: What does it mean to "version" a virtual document? Are you versioning the generation process, or storing generated pages over time? Some systems such as WikiWeb [5] visit a URL and store page differences in a database, so that the system can track the Web page modifications.

Some research has been conducted on these issues. Caumanns [6] deals with the creation of dynamic documents by predefined templates or knowledge. Iksal and Garlatti [7] describe an adaptive web application system, which generates adaptive virtual documents by means of a semantic composition engine based on user models. Tetchueng et al. [8] develop an adaptive composition engine, called SCARCE—SemantiC and Adaptive Retrieval and Composition Engine—to design a context-aware learning system for an adaptive virtual document. Qu et al. [9] use the RDF graph model to define description formulations and neighboring operations for constructing virtual documents. Our research differs from these in that we consider virtual documents created by digital libraries and analytic applications, where the hypermedia middleware system itself has no way to control their content or generation, but must determine that a regenerated document is the same one as before.

Our architecture does not necessarily lead to the regeneration of old instances (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 Sect. 4.1).

2.2 Open Hypermedia Systems (OHS)

OHS traditionally keep links separate from content and combine them "just-in-time" to produce a viewable hypermedia document. Systems such as Multicard [10] and HyperDisco [11] separated links from content, allowing exiting applications to support hypermedia functionality and enabling the links and document versions to be managed in a more sophisticated manner avoiding problems such as orphaned or dangling links [12].

Within OHS research there has also been an emphasis on supporting third-party applications [13], and dealing with structures beyond the traditional navigational link [14], including virtual documents.

The Dexter Reference Model for hypermedia created in 1994 modeled hypertext documents using an opaque 'within-component' layer, connecting content with link structures through an anchoring interface [15]. The DHM system (based on Dexter) extended this notion with more complex composite components, including computed and virtual documents,

and a more advanced anchoring interface to deal with them [16].

In later OHS work the tendency is to make this interface more transparent and see hyper-structure as something that permeates both content and navigation, with systems such as Construct and Callimachus defining fine-grained structural elements as the basic building blocks of their Hypermedia [17,18]. More recent OHSs such as Auld Linky have also viewed the navigational and content structure more equally and applied adaptive functionality consistently across both [19,20].

However, these systems include virtual documents within their own extended set of supported structures, which makes managing the documents much easier than if they were being generated and managed in third-party systems. For example, Goose et al. [21] propose an integrated architecture for open hypermedia systems with a distributed and collaborative model., but bypass many of the problems of regeneration that occur when dealing with dynamic content from third-party applications (see Sect. 4).

Our JHE system is essentially an OHS (as it stores and manages links separately from the content they relate); however, unlike previous systems, it attempts to provide navigational link support to *third-party dynamic content* from digital libraries and other applications. As such, it is very similar in architecture to the Distributed Link Service (DLS) [12] that applied OHS functionality to the broader Web, but tackles problems more familiar to structural systems that control their virtual documents more tightly.

2.3 Element identification and internal document structures

An anchor is a link endpoint—an identifiable element in a document. This can include a selected word, paragraph, image, field in a web form, term (e.g., key phrase), identifiable object (e.g., an author or article title, educational simulation parameter, NASA model parameter, or a product price in an online book store), or the entire document itself. After the user creates an anchor, JHE should remember its selection in the document. The next time its underlying element appears, 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 [12].

- Specific: only applies to one particular element in one particular document (e.g., an author's name of one specific article within another particular article's reference list).
- Local: applies to many elements with same identifying aspect (e.g., key phrase, object ID, or some specified object parameter) in one particular document (e.g., every instance of an author's name within one specific article).
- Generic: applies to many elements with the same identifying aspect in many documents (e.g., every instance of an author's name in any article, search result listing, or other virtual document).

The manner of describing an anchor's location varies according to the type of media. For example, the HyTime [22] standard allows users to express anchors within SGML (text) files. 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 particular attribute (key phrase or author name with the value "Douglas Engelbart").

With text documents a byte count is commonly used. Microcosm's main approach for addressing an anchor is using byte offset [12] and similarly, the Open Hypermedia Protocol (OHP) [23] uses byte offsets (both forward and backward from the start and end of the file) to address locations. However, this can lead to the file editing problem (the file changes and the byte counts are no longer valid), causing possible link inconsistency. Microcosm uses date and time stamps to indicate that the file content has been changed, and warns users about possible link inconsistencies. To help prevent this problem some anchor context (usually ten 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 re-located to this new location. This works well but cannot guarantee 100% correctness.

Our JHE prototype uses XPath and XPointer for internal document addressing. XPath [24] is a language to address locations inside XML documents based on document structures. XPath models an XML document as a tree of nodes. XPointer [25] 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 Comparisons between hypermedia support for dynamic and static documents

Table 1 summaries the many differences between dynamically generated virtual documents and static documents, which affect the hypermedia support each requires.

Table 1 Major differences between dynamically generated and 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, links and other information about the document are stored, requiring much less room.	The entire content of the document is stored.
Reference	A query: Could be 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. It 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, unless the content is updated.
Regeneration	Regeneration is required every time the user revisits the document.	No regeneration is required.
Template	Usually some kind of document templates, query language specification, 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	Metadata is often used when generating the document's content (metadata becomes, is input to, or matches parameters used in document generation). Usually, it is not difficult to analyze the metadata. Because the document results from a query, search, or calculation; the query results from database have some embedded 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 is saved as a static document (virtual document materialization), 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 we treat two versions as the same (further discussion later). (4) Byte-by-byte comparison is not possible, because we only keep a minimal copy of the document. (This also depends on how we treat the two versions as the same, as we discuss later).	Usually performed by a document versioning system. In general, 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 the document	To record 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 (which we shall discuss in detail 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, then they technically need to be re-located, but this is straightforward.

From Table 1 we conclude that

(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. JHE should be able to generate unique identifiers and maintain them. Every time a virtual document is referenced, JHE 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 [26], including dynamic regeneration, re-location and re-identification, which we will discuss later.

Digital library and analytic applications currently can take little advantage of hypermedia functionality on the Web. This is, in large part, due to the predominantly read-only nature of Web applications today, in that most applications do not facilitate functionality that allows the user to add link anchors or comments, and bookmarks generally are limited to a particular page URL or internal anchor defined by the author. However, even if this were facilitated, most hypermedia functionality only supports static documents for embedding link anchors and target nodes (destination documents) that do not require parameters to generate (unless all parameters are held within the URL).

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) [27]. Dynamic hypermedia functionality applies these over the virtual analytic space of a computational system and the virtual documents generated within it. While the current research does not implement all of this functionality, doing so in each case would require consideration of the following aspects.

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

(1) Link destinations, comments, bookmarks, history items, stops along trails, and guided tours are dynamic and virtual. When the user traverses them, JHE should regenerate the destinations, which normally requires re-executing the commands associated with these destination documents (nodes).
(2) Once a document is regenerated, JHE must re-locate the anchors that users previously had placed within it, and should re-identify any elements that the anchors cover as the same ones as before.
(3) When analytic applications generate new query results, JHE should be able to re-locate and re-identify the anchors for the same elements originally identified in other query results.
(4) Overviews and structure-based search (as well as content-based search) over specification links must operate over a hypertext "network" or "web" that does not permanently 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 [28].

The main differences between "just-in-time" hypermedia and static hypermedia is that "just-in-time" hypermedia can operate on virtual documents and virtual elements within virtual documents, but to do this with digital libraries and other third-party applications requires processes of dynamic regeneration, re-location and re-identification.

In the next section we 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 [29]. Whenever a dynamic hypermedia link leads users to a virtual document, the hypermedia system should be able to arrange for its regeneration without relying on the user to reenter any parameters. Regeneration should occur, for example, when a student follows a bookmark to a simulation result or a researchers follows a bookmark to a search result.

Two major aspects are the extent to which regenerated documents should reflect the current state of the user ("context") and are expected to reflect the current status of the document content ("sameness criteria").

4.1 Regeneration and context

Virtual documents are regenerated using some parameters—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 latest articles published or current price charged for a book). The hypermedia system does not affect the regeneration of these elements, but does have to deal with them when placing anchors.

When regenerating a virtual document, there are two contexts to consider: the context of the user who first flagged the document as a destination, but also the current user's context, which could also influence the regeneration. Three general cases are possible [11].

(1) 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 instead are re-invoked (re-executed), which might be included in current logging and diagnostics.
(2) The system could adapt the page to the new user context. For example, a bookmarked article concerning a violent news event should not include graphics of a sensitive nature if the new user context is a minor. This also might be of particular importance when data in the document is sensitive in some way and security is thus a consideration.
(3) 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. Since we are not developing adaptive applications, any content adaptation is the responsibility of the digital library or analytic application that JHE supports. Customizing the set of links generated is the focus of other research underway but outside the scope of this article [30]. More flexible support of contexts, the second and third cases, is an issue for future research.

4.2 Revalidation "sameness" criteria

Whenever JHE receives a document for display, there should be some way for it to recognize 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 "validate" whether this is the same document as before. Following are some of the "sameness" criteria for determining this, listed from very rigid to very flexible:

(1) The file content and structure should be exactly the same (very rigid).
(2) The file's structure remains the same, but the content could differ. For example, element values such as the set of documents satisfying specific search criteria, simulation model calculations, or the current product 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.
(3) Some critical sections of the document should not change (e.g., the actual query results); other sections may (e.g., whether the results display full article abstracts, partial abstracts, or just the article citation).
(4) As long as the identifying parameters are the same that generated it (e.g., the search query), 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 Sect. 1's sales support example, when the analyst did the query and put some comments on that screen, she or he felt interested in that particular query on that particular day, so she or he wants the identical analysis results. If next time when she or he revisits this comment and the newly generated document is not same, then JHE should give an "invalid bookmark" or "stale document" warning and allow the user to remove the comment or keep it. Similarly, a researcher might want to confirm which journal articles were available at the time she or he did the original query, instead of which ones are available today. On the other hand, sometimes a comment is valid for any content generated by the same query.

A static document that has been edited also has the re-identification problem [12]. Simpler ways exist, however, to indicate that this document has been edited. If a static document is edited, for example, the editing timestamp changes. Every time a dynamically generated virtual document is generated, the timestamp varies. In this situation, even though the file size does not change, JHE has to re-identify whether this document is the same one as before.

4.3 Regeneration and revalidation procedure

The regeneration procedure of our JHE system has the following steps:

(1) Regeneration occurs when the user traverses some kind of link to a virtual document (a manual link, a bookmark, a link as a step within a guided tour, etc.) When a user creates this link, 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

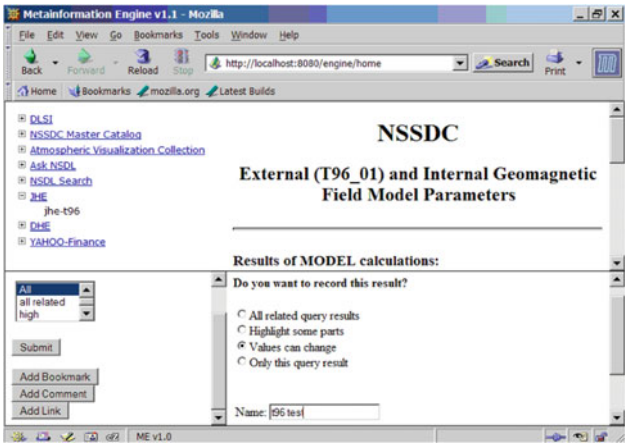

Fig. 2 Regeneration criteria

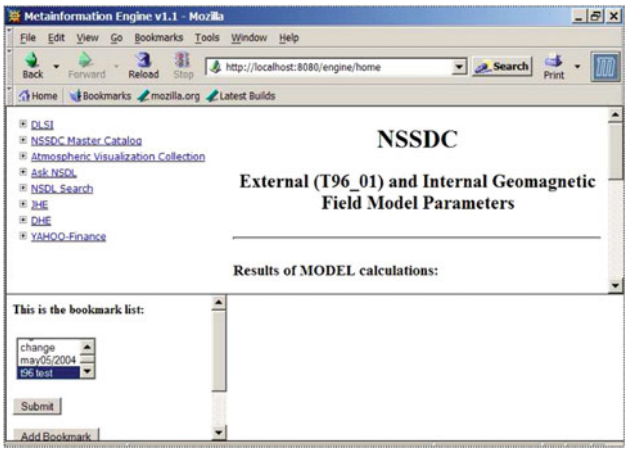

Fig. 3 Create a bookmark

virtual document B, then the link information contains the virtual document identifier A as the source and the identifier B as the destination.

Figures 2 and 3 show screenshots from the JHE prototype that follow from the example shown in Fig. 1. 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/revalidation criteria in the lower right screen. Choosing that bookmark anywhere (from the bookmark list shown in Fig. 3) will cause the document in Fig. 2 to be regenerated.

(2) When the user traverses that link to revisit the virtual document, 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) JHE sends the command information to the underlying digital library or other analytic application.

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

(5) JHE receives the virtual document and "revalidates" it. Revalidation of a virtual document depends on which "sameness" criteria JHE or the user chooses. For level 1 (very rigid), JHE compares it with the stored exact copy of the original document. For level 2, JHE translates the virtual document to an XML document according to the original document structure. For level 3, JHE finds the location of the critical section and then compares it with original stored content. For level 4 (very flexible), JHE does not compare the document content with the original document.

For levels 1 and 2, the regenerated virtual document is parsed according to the original document structure.

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

The commands which JHE sends to the application in step (3) are the same ones that the user executed when generating the document the first time. For example, at that time when the user issued a series of progressive queries, inputting a set of parameters for each, and bookmarked the final result, 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 that during his or her classroom preparation, a teacher invoked a simulation from a digital library course material repository that presented a series of dialog screens to gather input values before generating an animation, which he bookmarked. In the classroom the next day, when he follows the bookmark, JHE invokes the input dialog screens in the background. Instead of displaying each dialog for him or her to reenter the input values, the system fills in each dialog with the stored values originally entered and submits it. He or she just sees the resulting animation it generates, which (presumably) is the same one as he originally saw. (JHE uses the sameness rules discussed earlier to validate this.)

Alternatively, when a developer integrates an application with JHE (see Sect. 6), she or he 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, JHE could instantiate it with all of the required parameters.

4.4 Parameters for regenerating virtual documents

In order to regenerate a virtual document, JHE should record: dynamic link information, virtual document information, document generation information and document re-validation information. Parameters include:

- Link (linkID, title, description, source, destination, userID, time_created)
- Document (docID, version, title, description, size, structure, metadata)
- Revalidation (docID, criteria, content)
- Generation (docID, applicationID, command, parameters, shortcut, template)

4.5 Document schema

We use XML schema to express the document structure. Figure 4 shows the original document.

For the document schema in Sect. 4.5, the XML schema is shown in Fig. 5.

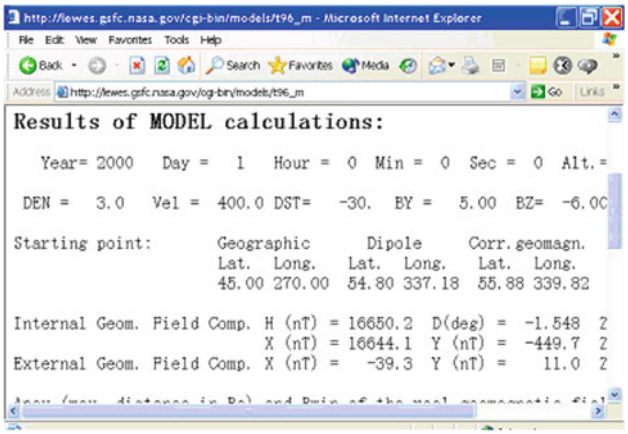

Fig. 4 Original query result

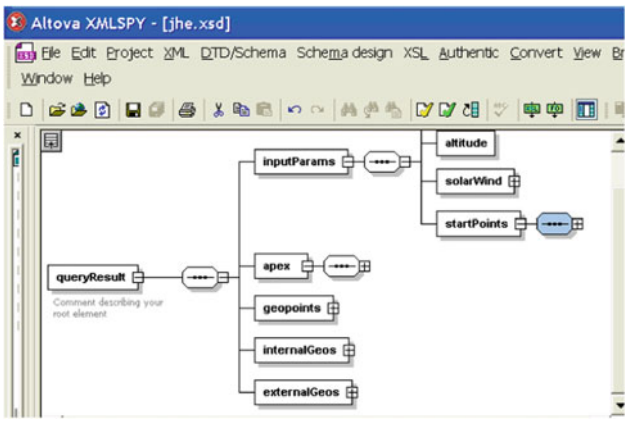

Fig. 5 XML schema for virtual document

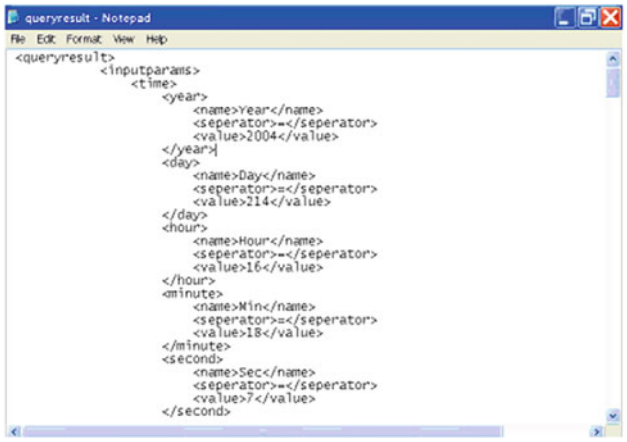

Fig. 6 Translated virtual document

After parsing and translating the document, we have an XML document shown in Fig. 6.

Currently, for each type of virtual document, the developer manually analyzes the document structure and generates the XML schema offline. Each application could have several schema based on different document types. This is a one-time activity for each document type, unless its structure changes later, requiring an update. The development of automatic generation of document schema is in progress [31].

5 Re-identification and re-location

Whenever an application generates or regenerates a document, JHE needs to determine whether it has encountered that document before (i.e., re-identify it). For example, has JHE encountered this simulation result or stock analysis before? Next JHE analyzes the document contents both to "re-locate" any anchors that users had placed within the document, and also to "re-identify" any content that matches other anchors that had been created independent of this document, but happen to be found within its content (in accordance with sameness criteria). For example, has a student or teacher placed a comment on a simulation parameter, the researcher linked to an article title, or the analyst added a guided tour step to this stock value, when each element appeared originally in a different document (virtual or static)? If so, these elements should be re-identified as those anchors.

Re-identification and re-locating documents (aka hypermedia nodes) and anchors have the following complications:

(1) JHE must recognize the re-generated document as the same one as it opened previously. Sects. 4.2 and 4.3 discussed this "revalidation."

(2) After a virtual document is regenerated, JHE should be able to find those anchors that were marked in this document previously.
(3) JHE must recognize when some content within any newly generated (or regenerated) document is the same element as one marked as an anchor previously. How it decides that an element is the same one depends on element-level sameness criteria (analogous to the document-level sameness criteria discussed above).
(4) When the user creates a link, she or he needs to specify whether it should appear every time the element appears in any document, every time it appears inside that particular document only (e.g., only for that particular query), or only on that particular instance within that particular document [12]. This is called "re-location granularity."
(5) As with a static hypermedia system, 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 document. The unique identifier is crucial here.
(6) As with dynamic regeneration, users may have "assumed" their current context when creating anchors and not thought about whether anchors should appear (be "re-located") 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-identification and re-location

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

(1) The first time a user selects and marks an anchor on the screen, JHE records the anchor information in an external anchor database. Then it is available for re-identification and re-location when documents are generated subsequently. Anchor information includes virtual document identifier, location, selection content, granularity, etc. Granularity means different degrees of anchor "scopes" (see Sect. 2.3). An anchor could appear at a particular location in a particular document (specific scope), on the same element anywhere in a particular document (local scope), or could appear in any document that has the same element (generic scope).
(2) After a virtual document is regenerated and re-identified as the same node, JHE looks for the anchor information for this document in the external anchor database.
(3) For each anchor inside this document (local and specific scope with same document identifier), 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. JHE should also locate all generic scope anchors that may exist in this document no matter what document identifier it has.
(4) For the re-located anchor, 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 element "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, JHE, upon demand, can look for those hypermedia functionalities associated with these elements in the database, and associate them with anchors.

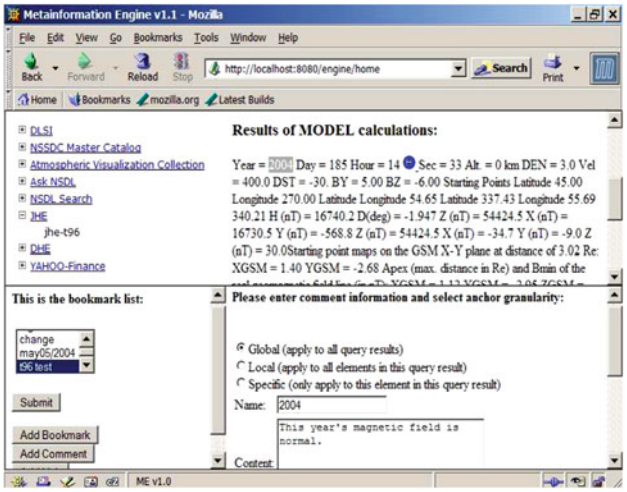

Fig. 7 Creating a JHE comment

Figure 7 shows how to create a JHE comment. The user clicks on the "Add Comment" button on the left side of the window and then selects the comment text from the main window. He or she enters some information in the lower right of the window and then submits it to JHE.

Figure 8 shows the regenerated document with relocated and re-identified anchors. The small blue icon at the side of the text indicates that JHE identified some hypermedia functionality (comments, links, guided tours, etc.) associated with the element. If the icon is red, that indicates the content of the anchor has changed based on the "sameness" criteria.

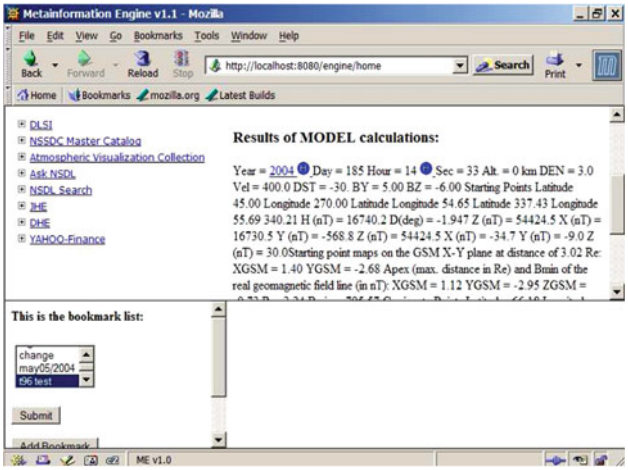

Fig. 8 Regenerated document with relocated and re-identified anchors

5.2 Parameters for re-identification and re-location of virtual elements

In order to re-identify and re-locate virtual elements (anchors), 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 contents from the document.
- Re-identification criteria: "sameness" criteria, whether the user allows the value to change.

6 The JHE

In this section, we present our architecture and prototype for JHE, as well as the functionality for each component. JHE extends our prior study with the Dynamic Hypermedia Engine (DHE).

6.1 The DHE

To supplement digital libraries and analytic applications with hypermedia functionality, the DHE [27,30,32] 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 prompts DHE to send a command (e.g., a query) to the target digital library or other analytic applications, causing it to generate a virtual document containing the query or computed 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 analytic applications through the use of application wrappers described below.

6.2 JHE architecture

While DHE dynamically generates link anchors and links for virtual documents, it currently does not support re-identification, re-location, or regeneration. JHE extends DHE's architecture to supplement digital libraries and other analytic applications with hypermedia functionality in this "just-in-time" environment. JHE's architecture is shown in Fig. 9. In what follows, we describe JHE identifiers and the architectural components.

To integrate an application into the JHE infrastructure, the developer must specify the identifying information for each *kind* of document the application generates (e.g., search input screen, search results, table of contents, article details), as well as each *kind* of element which is specified through structural location or parameters (e.g., authors, article titles, and journal titles) instead of user selections within the content. JHE's document manager will instantiate or fill in the element values upon generating the virtual document. The developer also must write an application wrapper that interfaces the application to the JHE communication Gateway module.

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)
- Parameter set = (parameter1, parameter 2, … parameter *n*, version number)
- Anchor identifier = (document ID, location ID)

Each command is an uninstantiated expression that, once filled in during execution, can instruct an application to perform some computation, which results in generating a document. Each parameter is an uninstantiated placeholder for a value passed to and used by the command when executed.

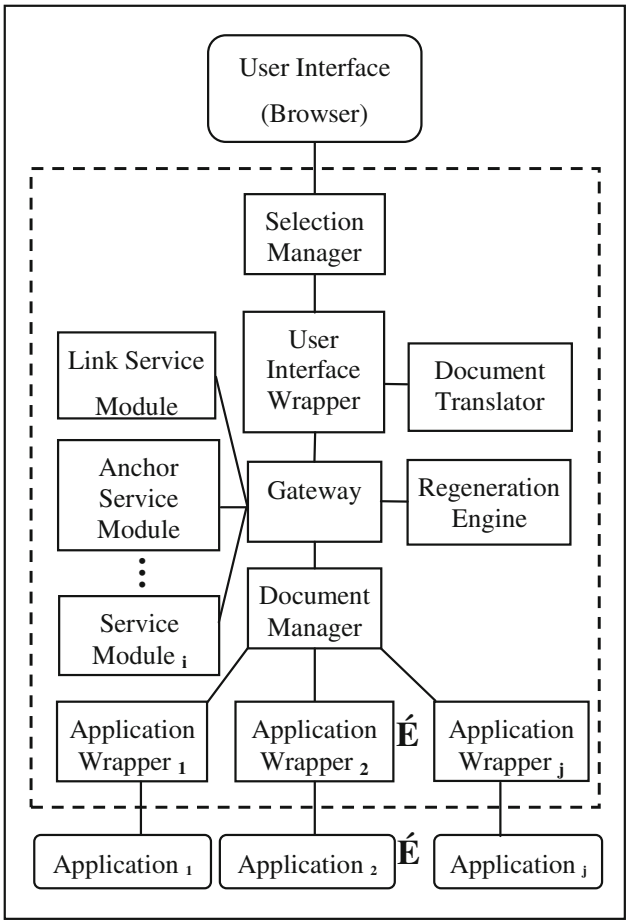

Fig. 9 Just-in-time Hypermedia Engine (JHE) architecture (in-between the user interface and applications such as a digital library)

Each location is an XPath/XPointer expression to a particular anchor within a document. While JHE will fill in the actual command and parameter values at execution time, the command format or skeleton is predefined by the application developer at the time she or he integrates her or his application. JHE assigns each un-instantiated (not filled-in) command "skeleton" its own ID. For example, a specific digital library or database application may support search or SQL queries that follow a well-defined format (which can be instantiated in an infinite number of ways).

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. Because, in practice, this parameter list is too long to use as an identifier, we use a unique number as the ID to represent it. Anchor location is generated dynamically when it is placed in the virtual document. It is unique within the document. Similarly, because the location expression is usually a long string, we use the location ID to store and retrieve the location expression.

Often 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 may be revalidated as a new document (corresponding to "sameness" criteria level 3, and a new version number is added to the parameter set) or as the same virtual document (corresponding to "sameness" criteria level 4).

6.2.2 Component functionality

JHE is middleware that integrates many applications through application wrappers. JHE resides between the user interface (normally a web browser) and the applications that users access (e.g., a digital library). All JHE components (except the JHE selection manager) run on the server side. Inspired by the notion of Structural Computing [33], we developed the Regeneration Engine and Document Manager modules to be structure servers. These modules are entities that build upon the basic structure store services to implement a specific type of abstraction, and to make that abstraction available to client applications. Figure 9 shows the JHE architecture. Inside the dotted line box are the JHE components. We describe each component functionality in the following:

Gateway: Enables the communication between the JHE modules and serves as the router for JHE internal messages.

Application Wrapper: Manages the communication between its application system and the Gateway. It passes commands and parameters to the application to execute. It parses the resulting screens and documents to identify any "elements of interest" that JHE will make into link anchors [30]. It then translates these virtual documents and elements into XML pages for JHE internal processing.

Selection Manager (SM): When the user selects a span of content on the browser 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 compatible browsers [34], utilizing its XPointerLib [35] interface as the Selection Manager.

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

User Interface Wrapper: Handles communications between the User Interface (and Selection Manager) and Gateway.

Regeneration Engine (RE): It 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 Application Wrapper for execution and gets back resulting virtual documents from the Application Wrapper 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: 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): It manages (enables users to add and retrieve) hypermedia functionality such as links, annotations, guided-tours for a content selection, or existing anchor selected by the user, and stores hypermedia information into the database. Each type of hypermedia functionality has its own Service Module and service database. This includes the Anchor SM (which stores anchor information in a database and parses anchor information to find an anchor's exact and absolute byte offset within a document), Link SM, Comment SM, etc.

7 Conclusions and future work

This article examines research issues for "just-in-time" hypermedia, in particular with regard to supporting digital libraries and other dynamic third-party applications. This environment requires 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 implemented a "just-in-time" hypermedia engine residing between the browser and the underlying dynamic applications to provide previously unavailable hypermedia functionality for applications. JHE provides services for users to add bookmarks, comments, and links for virtual documents generated by an application system without altering the internal code of the external applications. We believe our study will also benefit other research fields which share some of these challenges, such as virtual documents and adaptive hypermedia.

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. This includes extending dynamic hypermedia functionality and just-in-time hypermedia support to virtual documents with non-textual contents. We shall investigate incorporating user profiles and document context, for regeneration, re-identification and re-location.

We also shall explore automatically generating the XML templates for and performing an automatic metadata analysis over virtual documents, for wrappers to parse and (re-)identify elements. This will ease the job of writing an application 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.

Dynamic hypermedia functionality is still greatly limited on the Web. We envision a much more utilitarian Web environment in which the full hypermedia feature set becomes available to and for being broadly used by the users of every digital library and other dynamic application.

Acknowledgements The authors gratefully appreciate the funding support for this research provided by the United Parcel Service, the U.S. National Science Foundation (under grants IIS-0135531, DUE-0226075, DUE-0434581 and DUE-0434998), and the Institute for Museum and Library Services (under grant LG-02-04-0002). The authors are grateful to our reviewers and the journal editors for their excellent suggestions with regard to revisions of an earlier version.

References

1. AskNSDL, http://www.asknsdl.org (2010). Accessed 12 March 2010
2. NASA's National Space Science Data Center (NSSDC), http://www.nssdc.nasa.gov/ (2010). Accessed 30 Aug 2010
3. Watters, C., Shepherd, M.: Research issues for virtual documents. In: Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference, Toronto (1999)
4. Ranwez, S., Crampes, M.: Conceptual documents and hypertext documents are two different forms of virtual document. In: Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference. Toronto, May (1999)
5. WikiWeb—Web Based Corporation Tools. http://www.wikiweb.com (2011). Accessed 12 May 2011
6. Caumanns, J.: A Modular framework for the creation of dynamic documents. In: Workshop on Virtual Documents, Hypertext Functionality and the Web at the 8th International World Wide Web Conference, Toronto (1999)
7. Iksal, S., Garlatti, S.: Revisiting and versioning in virtual special reports. In: Third Workshop on Adaptive Hypertext and Hypermedia, 12th ACM Conference on Hypertext and Hypermedia, Arhus (2001)
8. Tetchueng, J.L., Garlatti, S., Laube, S.: A context-aware learning system based on generic scenarios and the theory in didactic anthropology of knowledge. Int. J. Comput. Sci. Appl. **5**(1), 71–87 (2008)
9. Qu, Y., Hu, W., Cheng, G.: Constructing virtual documents for ontology matching. In: Proceedings of the 15th International Conference on World Wide Web, Edinburgh, pp. 23–31 (2006)
10. Rizk, A., Sauter, L.: Multicard: an open hypermedia system. In: Proceedings of the ACM European conference on Hypertext, Milan, pp. 4–10 (1992)
11. Wiil, U.K., Leggett, J.J.: The HyperDisco approach to open hypermedia systems. In: Proceedings of the 7th ACM Hypertext Conference, Washington, pp. 140–148 (1996)
12. Davis, H.: Data integrity problems in an open hypermedia link service. Ph.D. Thesis, Southampton University, Southampton (1995)
13. Whitehead, E.J., Jr.: An architectural model for application integration in open hypermedia environments. In: Proceedings of the eighth ACM conference on Hypertext, Southampton (1997)
14. Wiil, U.K., Nürnberg, P.J.: Evolving hypermedia middleware services: lessons and observations. In: ACM Symposium on Applied Computing, San Antonio, pp. 427–436 (1999)
15. Halasz, F., Schwartz, M.: The Dexter hypertext reference model. Commun. ACM **37**(2), 30–39 (1994)

16. Grønbæk, K., Trig, R.H.: Design issues for a Dexter-based hypermedia system. Commun. ACM **37**(2), 40–49 (1994)
17. Wiil, U.K.: Hypermedia technology for knowledge workers: a vision of the future. In: Proceedings of the Sixteenth ACM Conference on Hypertext, Hypertext 2005, Salzburg, Sep, pp. 4–6 (2005)
18. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., Schraefel, M.C., Vaitas, M., Christodoulakis, D.: Structuring primitives in the Callimachus component-based open hypermedia system. J. Netw. Comput. Appl. **26**(1), 139–162 (2003)
19. Bailey, C., El-Beltagy, S.R., Hall, W.: Link augmentation: a context-based approach to support adaptive hypermedia. In: 12th ACM Conference on Hypertext and Hypermedia, Arhus, pp. 239–251 (2001)
20. Griffiths, J., Millard, D., Davis, H., Michaelides, D., Weal, M.: Reconciling versioning and context in hypermedia structure servers. In: Nürnberg, P.J.(ed.) Proceedings of metainformatics international symposium, Esbjerg, pp. 118–131 (2002)
21. Goose, S., Lewis, A., Davis, H.: OHRA: towards an open hypermedia reference architecture and a migration path for existing systems. J. Digit. Inform. **1**(2), 45–61 (1997)
22. HyTime: Information Technology—Hypermedia/Time-based Structuring Language (HyTime). http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html (2011). Accessed 12 May 2011
23. Davis, H.C., Lewis, A., Rizk, A.: OHP: a draft proposal for a standard open hypermedia protocol. In: 2nd Workshop on Open Hypermedia Systems, Washington (1996)
24. XML Path Language (XPath). http://www.w3.org/xpath (2010). Accessed 1 Jan 2010
25. XML Pointer Language (XPointer). http://www.w3.org/xptr (2011). Accessed 30 May 2011
26. Karadkar, U.P., Francisco-Revilla, L., Furuta, R., Shipman, F., Arora, A., Dash, S., Dave, P., Luke, E.: Metadocuments supporting digital library information discovery. Int. J. Digit. Libr. **4**(1), 25–30 (2004)
27. Ho, S.M., Song, M., Bieber, M.: IntegraL: the effectiveness of a link-based federated search infrastructure. In: iConference, University of Illinois, Urbana-Champaign, pp. 109–114 (2010)
28. Montero, S., Díaz, P., Dodero, J., Aedo, I.: AriadneTool: a design toolkit for hypermedia applications. J. Digit. Inform. **5**(2), 214–217 (2004)
29. Halasz, F.G., Reflections on NoteCards: seven issues for the next generation of hypermedia systems. In: Proceedings of the ACM Conference on Hypermedia, Chapel Hill, pp. 345–365 (1987)
30. Catanio, J., Nnadi, N., Zhang, L., Bieber, M., Galnares, R.: Ubiquitous metainformation and the 'what you want when you want it' principle. J. Digit. Inform. **5**(1), 1–37 (2004)
31. Ho, S.M., Song, M., Bieber, M.: Shaping user's information seeking behavior: a Link-based federated search infrastructure. Inform. Sci. (in preparation)
32. Galnares, R.: Augmenting applications with hypermedia functionality and metainformation. Ph.D. Thesis, New Jersey Institute of Technology, Newark (2001)
33. Nürnberg, P.J., Schraefel, M.C.: Relationships among structural computing and other fields. J. Netw. Comput. Appl. **26**(1), 11–26 (2003)
34. Mozilla Web Browser. http://www.mozilla.org (2011). Accessed 30 May 2011
35. XpointerLib. http://xpointerlib.mozdev.org/ (2011). Accessed 30 May 2011