A THEMATIC METHOD FOR WEB-BASED APPLICATIONS DESIGN

Seung C. Lee, University of Minnesota at Duluth, slee@d.umn.edu

ABSTRACT

The proposed methodology employs a multi-tier architecture and a unique Web application model in which a Web application is viewed as a collection of themes, meta-themes, and design primitives. A theme is defined as a set of semantically tight-cohesive and syntactically loose-coupled information clusters consisting of multiple design primitives. They include seven unique Web page types, eight link types, and four component types. The methodology also utilizes a central data repository called the link data dictionary.

Keywords: Web applications, design, methodology, theme

INTRODUCTION

For the Web application to be a successful business application model, it should be able to execute business logic and integrate itself with heterogeneous systems [17]. It should also support powerful processes composed from components as well as an increasing range of functions. It is required as well to be designed for high performance, scalability, and extendibility of the mode of usage. Such robustness can be accomplished by bridging the gap between available Web programming specifications and a Web application design methodology that also backs up content management. Robust Web applications should deliver the same functionality in the manner just as it has been done by conventional business applications [8]. This means that it is not feasible any more to develop Web applications off the top of developers' head. Rather, they need a well-defined methodical procedure for identifying informational and functional requirements and pinpointing data flows between the basic building blocks of Web applications. Furthermore, Web technologies have been rapidly advancing to enable organizations to deploy more sophisticated applications on the Web as it gains ground in business computing [1]. While new technologies contribute to broadening the scope of business applications on the Web, they also add a new level of tasks to Web application design. For example, today's advanced Web applications capitalize on the new technologies for data presentation, manipulation, representation, and integration at a very refined level.

Developing Web applications are often required to be integrated with existing heterogeneous systems [7, 8]. Web services are one of the emerging technologies that can be used for the integration of Web applications with legacy systems (e.g. [6, 13]). Besides, extending mode of usage of the Web application calls for its content management through its design process [7, 10]. As Fingar (2000) suggests, content management is equally important as developing an application. Indeed, it is believed that the Web application should always be "under construction" in terms of its content, presentation, and its structure. Traditionally, design heuristics such as loose-coupling and tight-cohesion concepts have been utilized as a way to cope with such dynamicity and reusability issues. We can achieve the goals for the Web application by applying the same heuristics to its design process—that is, by organizing both informational and functional requirements into semantically tight-cohesive and syntactically loose-coupled clusters, which in this paper are called *themes*. A theme is an abstract cluster of Web pages, links, components, and/or data stores, which is semantically and syntactically distinguishable from other clusters. Being abstract means that we are not concerned about the appearance details of themes; rather, it means that a theme is a main topic of discourse that has a name. For example, theme names might be "create your profile," "view course offers," and so forth for the "MBA Student Management System" deployed for a university. A "big" theme can be broken into smaller themes, and a group of themes can be promoted to a higher level theme called a *meta-theme*.

Most of the existing Web application design methodologies have glossed over the important factors aforementioned. This seems to be natural because Web applications development has evolved along with Web technologies. For example, the Relationship Management Methodology by Isakowitz et al. (1995) focuses on static Web site design, and recent work by Conallen (2000) has a focal point on the application of the new language paradigm (i.e. UML) to Web-based applications development. Yet another study by Fingar (2000) describes a higher-level integration of an e-commerce application with heterogeneous external systems. The lag between the

existing methodologies and new Web technologies, therefore, demands a comprehensive methodology.

In the next section, we present a review on the behavioral issues in the Web application design followed by the founding concepts for the proposed methodology in which the application architecture and design primitives will be explained in detail. After that, the two phases of the proposed methodology—analysis, and design—will be described. The paper concludes with major contributions and suggestions for future research.

BEHAVIORAL ISSUES IN WEB APPLICATION DESIGN

There are technical and behavioral issues in Web application design. The former is largely related to implementation (see [17]). The first group of behavioral issues is related to identifying, organizing, and managing informational requirements of a target Web application. It is well recognized that user information needs should be thoroughly analyzed prior to developing an effective application [2]. Unlike users of conventional applications, however, the counterparts of public Web applications are not clearly defined because they are general public. This could make it difficult to apply conventional requirements gathering techniques such as interviews and may require designers to have deep knowledge about the users of the target Web application. Thus, it seems natural that most of the previous work on design center on content structuring and its presentation to the user with minimal attention to gathering user information requirements (e.g. [14]). Although a recent method proposed by Conallen (2000) has suggested a gathering technique based on the use case approach [15], it basically assumes that the target application's users (and actors) are well defined. This implies that the method by Conallen (2000) is more useful for Web applications that have clearly-defined end users.

The second set of the issues is related to presenting effective navigational cues to the user [11, 12, 18, 25], offering semantically cohesive content, and creating a syntactically loose-coupled structure for better user comprehension [9, 30, 31]. In cognitive science, comprehension depends on how a user constructs a mental model based on the visible objects and their semantic relations [32]. This implies that semantically cohesive and syntactically loose-coupled themes require less effort in modeling, and hence increase the comprehensibility of the user [30, 31]. For example, a theme is coherent if a user can construct a mental model that corresponds to facts and relations in a real world [16]. On the other hand, cognitive overhead is "the additional effort and concentration necessary to maintain several tasks or trails at one time" [21, p. 40]. This might be inevitable due to the limited capacity of human information processing.

The previous studies about Web application design indicate some useful design principles. First of all, a Web application should be designed in a way that reduces cognitive overhead, which is primarily related to user disorientation and user-interface adjustments [22]. Second, to provide users with an effective navigation, a Web application should be characterized by: (1) higher local coherence—that is, providing appropriate indication of semantic relationships between contextual pages through careful use of hyperlinks within a given theme; (2) higher global coherence—that is, providing adequate overview by aggregating contextual pages, links, components, and/or data stores into a cohesive and loosely coupled theme; and (3) effective navigational facilities—that is, providing support for navigation with respect to direction (breadth) and distance (depth) within a theme as well as across themes [30, 31].

THE FOUNDATION CONCEPTS

This section explains the foundation of the proposed methodology, including a multi-tier Web application architecture, page types, component types, themes and meta-themes, and link types.

Web Application Architecture

In this paper we adopt a logical architecture comprised of the four layers: presentation, business, data access, and data. The presentation layer includes everything specific to the user interface. It isolates the rest of the application from changes to the presentation layer. This layer would be implemented as HTML, graphics, style languages, and other MIME documents that involve no execution of business logic on the server side. The business layer hosts business logic that is deeply integrated from the user interface to the data store rather than being contained solely within a package of code. The data access layer supports all data access requirements of the business layer and includes all components used to access data. Data access interfaces simplify data access by hiding implementation details. Finally, the data layer includes data and data store software, including databases, e-mail stores, file stores, XML documents, message queues, and directory services.

Web Page Types

If a page is rendered by the request/response process, the page is called the *primary page*. It is primary because the page content is not formulated or computed from program code and is rendered the same on every request unless changed by the page author. The request/execution/response process, on the other hand, is employed when a requested page contains program logic that the Web server must execute before it responds to the request. The server executes the logic, formulates an HTML response, and returns it to the client. If a page is rendered in this manner, the page in this paper is called the *derived page*. For example, a search process in general involves the three steps: submitting a keyword, executing the search logic, and responding with a search result. The search result page is an example of derived page. A derived page is, therefore, always the result of execution of a *server page*. A server page is a Web page that includes program logic that is executed on a Web server before any response to a request.

Meanwhile, a primary page may or may not render a form (e.g. HTML form). If it does, it is called the *primary interactive page (PIP)*. It is interactive in the sense that a user can submit input to the Web server (in fact, a server page) for its processing and the server responds in some manner. If a primary page does not render a form, it is called the *primary noninteractive page (PNP)*. A primary page may be associated with zero or more preprocessing pages (e.g. style sheet pages or client-side script pages that are often reusable). They are embedded in one or more primary pages by explicit instructions. Such a page is called the *client preprocessor page*. It is named so because it is processed by the client (i.e. a browser) *before* an associated primary page is processed. A derived page can be generated from a server page either statically or dynamically. In addition, it may or may not have a form. We, therefore, can identify four different types of the derived page: *statically-derived interactive page (SIP)*, *statically-derived noninteractive page (SNP)*, *dynamically-derived interactive page (DIP)*, and *dynamically-derived noninteractive page (DNP)*. The presentation-layer page types are in fact business facades.

The server page has three types as well and resides in the business layer. If a server page produces one of the four derived page types, it is called the *interlayer server page*. Otherwise, it is called the *withinlayer server page*. The former is named so because it crosses over the boundary between the business layer and the presentation layer. For example, a server page for order summary is an interlayer server page. The latter is named in a similar context. A withinlayer server page works behind the scenes. It normally passes data string, state information, an execution result (e.g. a search result), or a flag to another server page for additional processing for HTML response formulation. A withinlayer server page, therefore, has no direct rendering targeted for the presentation layer. Instead, it is confined within the business layer (e.g. setting a session variable). Then, what if a server page plays the dual roles? Such a server page is called the *hybrid server page*. In addition, like a primary page may have zero or more preprocessor pages, a server page can have its counterparts, which are called the *server preprocessor page*. Again, like the client preprocessor page, the server preprocessor page is first processed by the Web server *before* an associated server page is processed. Thus, the business layer has the following page types: interlayer server page (ISP), withinlayer server page (WSP), hybrid server page (HSP), and server preprocessor page (SPP).

Component Types

The Web is becoming a critical business computing platform as organizations move their everyday tasks to the Web. The migration could begin with identifying legacy applications, services, and data for a certain form of integration [8]. One way is to make the core business processes running on the legacy applications or third-party software components available for a Web application by object wrapping or Web services [7, 6, 13]. The proposed methodology explicitly takes components into account in four ways: *presentation layer* (PLC), *webserver layer* (WLC), *appserver layer* (ALC), and *data access layer* (DAC) components. We rule out any components triggered automatically such as plug-ins or add-ons because the inherent access to such components is implied by semantics of the corresponding design element.

Link Types

If a click on a link causes user input to be written to a data store via a server page, it plays both semantic and syntactic roles because it passes the data to the server page (i.e. passing data is semantic in that it can be interpreted as a specific instruction), which in turn triggers a data access component (i.e. syntactically associate the two elements, the server page and the component). Accordingly, we classify the link into eight link types based on their semantic and syntactic

context: *anchor link* (<A>), *build link* (<B>), *call link* (<C>), *directive link* (<D>), *enjoin link* (<E>), *form link* (<F>), *germinate link* (<G>), and *intermediate link* <I>. The anchor link type is a "generic" hyperlink created by the anchor tag. A bookmark could be considered to be an anchor link type. The directive link type is used to associate a preprocessor page to a Web page. The call link type is used to trigger a component from a Web page. The build link type connects either an interlayer server page or a hybrid server page to a dynamically-derived page—that is, it applies to the case where a server page dynamically produces a page for the presentation layer. Unlike the build link type, the enjoin link type is used to request another page from the page that has been requested. The form link type denotes a submission of a page containing a data-capturing from to its process page. The germinate link type connects either an interlayer server page or a hybrid server page to a statically-derived page (not a dynamically-derived page, which is handled by a build link type). Finally, the intermediate link type is a placeholder used in the analysis phase. It will be replaced with one of the seven other link types in the design phase.

Themes and Meta-themes

A Web application can be viewed as a collection of themes and meta-themes, which in turn are a set of Web pages, links, components, and/or data stores. Highly cohesive and loosely coupled information clusters and hence their delivery logic can make content management easier [7, 24]. A theme has the following characteristics:

- It has a meaningful, abstract name.
- It is highly self-contained in the sense that a theme is syntactically loose coupled and semantically tight cohesive.
- It can be decomposed into lower level themes and promoted to a meta-theme as well.
- Every theme has its own *theme page*.

Then, what is not a theme? There are no cut-and-dry criteria but we suggest some heuristics. An entity is not a theme if it:

- Consists of a single primary non-interactive page that has no outgoing links.
- Has no much meaning or becomes impractical when decomposed further into lower levels.
- Violates the design heuristics of tight cohesiveness and loose coupling beyond necessary compromises.

On the other hand, a theme of themes is called a meta-theme, which has the following characteristics:

- Its name can be used as either a "menu item" name or a placeholder during design and hence implementation.
- It is in general a collection of two or more themes, but it may be allowed to have only one theme when a meta-theme is used for either a non-hyperlinked menu name or a placeholder.
- A "parent" theme is automatically qualified as a meta-theme.
- It has also a theme page. If its name is used for either a non-hyperlinked menu name or a placeholder, a meta-theme has no theme page.

METHODOLOGY

The proposed Web application design methodology primarily consists of analysis and design phases. However, we will also propose some general ideas for the requirements gathering and implementation.

Requirements Gathering

As mentioned earlier, users of a *public* Web application are vaguely defined. This may hamper the requirements gathering process. We suggest an idea for requirements gathering that could mitigate the difficulty. The Internet was originally intended to grow on open standards and open sources. This has inevitably resulted in the relatively lower barriers to imitation [19]. Although the rules are changing as more and more Internet-related patents and legal battles are looming [23], getting insights from existing Web applications would be acceptable behavior. This may be called "inspired assimilation." It could be augmented by the model-based approach [28, 29]. Its goal is to construct application models that show the structure, processes, and resources of a business as simply and as directly as possible. Based on the approach, developers may be able to gather requirements by applying the conventional information gathering techniques [2] to those who are directly involved in the business processes and practices. Although they may not be the direct users of the target application, designers could at least get some insights into requirements gathering.

Analysis

It begins with organizing the themes identified via a requirement gathering process into a higher-level hierarchy, which is then gradually refined into lower-level details. Complexity frequently takes the form of

a hierarchy, which has been a major facilitating factor enabling us to understand and describe complex objects and their parts [4, 27]. The following subsections illustrate the analysis process.

Organizing themes

First of all, it is possible for a target Web application to have more than one sub-application. For example, an "MBA Student Management System (MMS)" may consist of two sub applications for, say, student and administrator. Once we identify sub-applications, four different tasks are performed on each of them: (1) identifying higher-level themes, (2) decomposing them into lower-level themes, (3) promoting or adding meta-themes, if necessary, and (4) organizing all the themes into a hierarchy. Figure 1 shows a simplified theme hierarchy for the student sub-application of the MMS.

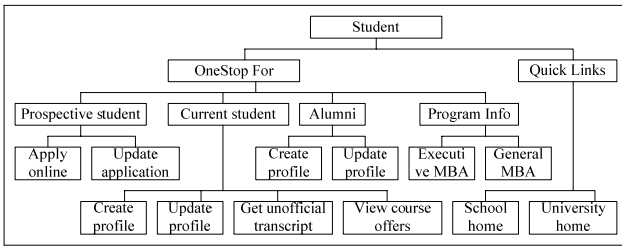

Figure 1. A theme hierarchy for the student sub-application of the MMS.

The degree of information granulation also determines, as Fingar (2000) points out, the degree of changeability of structure and presentation. In fact, the breath and the depth have a trade-off relationship: a deeper (shallower) structure would require a narrower (wider) breadth and vice versa. To enhance the degree of changeability, we should achieve tightly cohesive themes both at the top level and their lower levels. We suggest that a designer come up with a hierarchy as fine-grained as possible and then combine themes (i.e. nodes in the hierarchy) in line with her implementation strategy.

Determining access scope

This step determines the access scope from a page to various themes based on a theme hierarchy. Broader access scope normally makes a page crowded with themes. However, it can bring a shallow structure that requires fewer clicks to get to a destination theme. At this step, we determine the three kinds of access scope: a single *homepage scope* (HS), a single *common scope* (CS), and possibly multiple *extension scope* (ES). An HS includes the themes to be appeared on a default homepage either *with or without being hyperlinked*. It may include all or a subset of the themes identified in a theme hierarchy. A CS defines a number of themes that will commonly appear *as being hyperlinked* on every page, probably including a default homepage. The themes included in a CS and an HS may be identical, especially when the application is small. An ES is constrained by the two previous types of scope. For example, let's assume that an HS contains "onestop for," "quick links," "prospective student," "current student," "alumni," and "program info," and a CS has the last four themes in the HS above. Then, an ES for the "current student" may include all or a subset of the lower-level themes (nodes) below the "current student" theme. Figure 2 shows examples for the homepage, common, and extension scope. We use the namespace aforementioned to pinpoint any specific node within a hierarchy. In any case, actual scope should reflect a designer's implementation strategy.

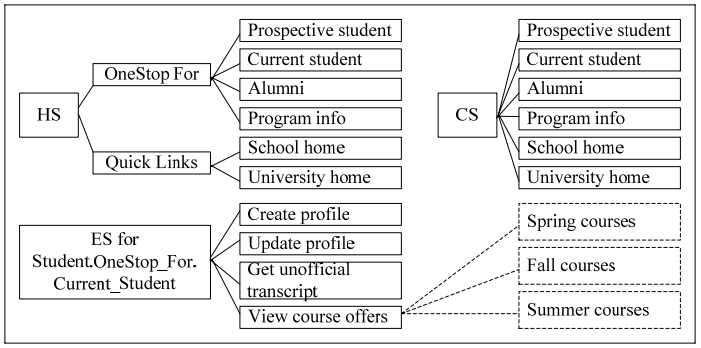

Figure 2. Example scope diagrams based on Figure 1 (the dotted not shown).

Drawing higher-level theme diagrams

This step is for drawing higher-level theme diagrams for the homepage scope, each extension scope, and other themes and meta-themes identified in the three types of scope by mainly identifying non-theme pages and foreign themes. While drawing, we explicitly consider theme pages. The three types of scope, in fact, are special meta-themes. We first draw a theme diagram for each access scope except for the common scope, and then move to individual meta-themes. The *homepage scope theme diagram* has a theme page (i.e. a homepage), and each *extension scope theme diagram* has also a theme page. A theme diagram for any scope non-leaf node (i.e. meta-theme) of any access scope *may* include a theme page, while a theme diagram for any scope leaf node (i.e. non-meta-theme) includes a theme page. Thus, there would be as many higher-level theme diagrams as the sum of the number of scope leaf nodes, the number of scope non-leaf nodes, the number of

extension scope, and one homepage scope. (see Figure 3).

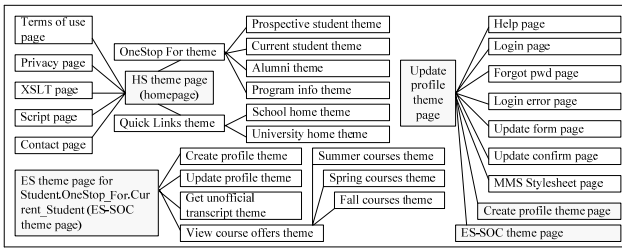

Figure 3. Sample higher-level theme diagrams based Figure 2.

Refining the higher-level theme diagrams

The refinement of the higher-level theme diagrams involves: (1) specifying all of the presentation-layer page types by their suffixes and (2) using the three link types out of the eight link types with an arrow: <A> for an anchor link, <D> for a directive link, and <I> for an intermediate link. The remaining pages types, link types, all component types, data stores, logic flows, and the application architecture will be taken into account in the design stage. Figure 4 shows a refinement of the "update profile" theme diagram in Figure 3.

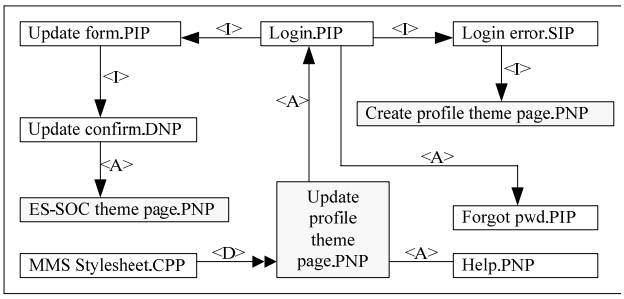

Figure 4. A refinement of the "update profile" theme in Figure 3.

Design

This phase performs the two tasks: (1) finalizing the refinement process and (2) developing the link data dictionary (LDD).

Finalizing theme diagrams

This step defines the detailed rendering process for each presentation-layer page identified in the previous step and incorporates the Web application architecture into the process. Figure 5 shows a finalized theme diagram for the "update profile" theme of the MMS that has been refined in the previous step.

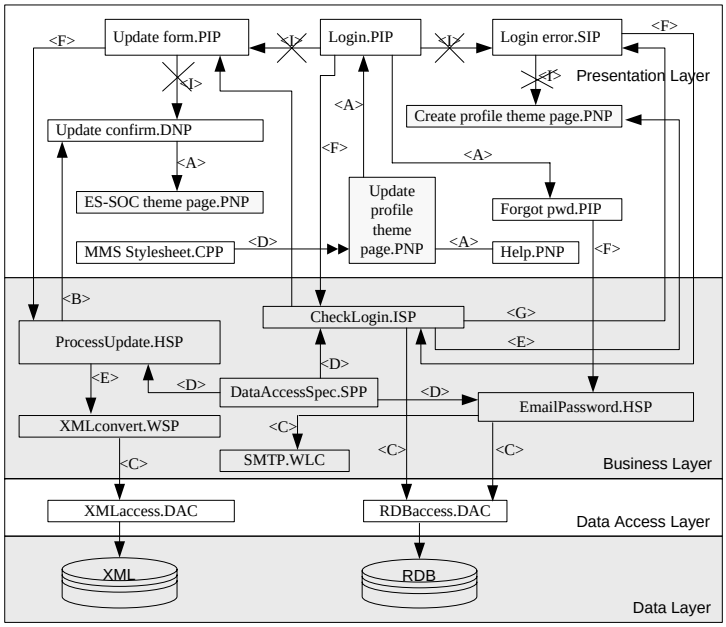

Figure 5. A finalized theme diagram for the "update profile" shown in Figure 4.

Developing the link data dictionary

A data dictionary has been used as a central repository of all data definitions for a conventional business application, which supports the development process in designing data stores [26]. Similarly, a *link data dictionary* (LDD) in this paper is a central repository of data for a Web application. It helps the development of data stores, the specification of input controls of a form, and the identification of necessary data flow details between pages.

CONCLUSIONS

As for the implementation, the first issue is that, the designer should design Web pages. Second, the common scope contains themes to be hyperlinked as common menu items repeating on every Web page. As for future research, there are some suggestions. First, we assume that software components are out there. We do not consider component-based application development tasks [20]. Second, we did not specifically deal with page layout design issues. Despite the limitations, we believe that the method should provide a consistent and manageable way for the development of Web applications.

REFERENCES

1. Britton, K. H., Li, Y., Case, R., Seekamp, C., Citron, A., Topol, B., Floyd, R. and Tracy, K., 2001. Transcoding: Extending e-business to new

environments, IBM Systems Journal, 40, 1, 153-178.
2. Byrd, T. A., Cossick, K. L. and Zmud, R. W., 1992. A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques, MIS Quarterly, 16, 1, 117-138.
3. Conallen, J., 2000. Building Web Applications with UML, Reading, Massachusetts: Addison-Wesley. Conklin, J., 1987. Hypertext: An Introduction and Survey, IEEE Computer, 20, 9, 17-40.
4. Cook, M. A., 1996. Building Enterprise Information Architectures: Reengineering Information Systems, Upper Saddle River, NJ: Prentice Hall.
5. Engelbart, D. C., 1995. Toward Augmenting the Human Intellect and Boosting Our Collective IQ, Communications of the ACM, 38, 8, 30-33.
6. Ferris, C. and Farrell, J.,2003. What are Web Services, Communications of the ACM, 46, 6, 31.
7. Fingar, P., 2000. Component-based frameworks for e-commerce, Communications of the ACM, 43, 10, 61-66.
8. Flurry, G. and Vicknair, W., 2001. The IBM application framework for e-business, IBM Systems Journal, 40, 1, 8-24.
9. Fraternali, P., 1999. Tools and approaches for developing data-intensive Web applications: A survey, ACM Computing Surveys, 31, 3, 227-263.
10. Guenther, K., 2001. What is a Web content management solution? Online, 25, 4, 81-84.
11. Halasz, F. and Schwartz, S., 1994. The Dexter Hypertext Reference Model, Communications of the ACM, 37, 2, 30-39.
12. Hardman, L. and Sharrat, B., 1990. User-centered Hypertext Design: The Applications of HCI Design Principles and Guidelines, In R. Mcaleese and C. Green (Eds.), Hypertext State of the Art, Intellect, 252-259.
13. Huang, Y. and Chung, J., 2003. A Web services-based framework for business integration solutions, Electronic Commerce Research and Applications, 2, 1, 15-26.
14. Isakowitz, T., Stohr, E. A. and Balasubramanian, P., 1995. RMM: A Methodology for Structured Hypermedia Design, Communications of the ACM, 38, 8, 34-44.
15. Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G., 1992. Object Oriented Software Engineering: A Use Case Driven Approach, Wokingham, England: Addison-Wesley.
16. Johnson-Laird, P. N., 1989. Mental Models, In M. I. Posner (Ed.), Foundations of Cognitive Science, Cambridge, MA: MIT Press, 469-499.
17. Johnson, R. D. and Reimer, D., 2004. Issues in the development of transactional Web applications, IBM Systems Journal, 43, 2, 430-440.
18. Kahn, P., 1995. Visual Cues for Local and Global Coherence in the WWW, Communications of the ACM, 38, 8, 67-69.
19. Makadok, R., 1998. Can first-mover and early-mover advantages be sustained in an industry with low barriers to entry/imitation? Strategic Management Journal, 19, 7, 683-696.
20. Mili, H., Mili, F. and Mili, A., 1995. Reusing Software: Issues and Research Directions, IEEE Transactions on Software Engineering, 21, 6, 528-561.
21. Miller, G. A., 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capability for Processing Information, The Psychology Review, 63, 2, 81-97.
22. Marchionini, G. and Schneiderman, B., 1988. Finding Facts and Browsing Knowledge in Hypertext Systems, IEEE Computer, 21, 3, 70-80.
23. O'Reilly, T., 2000. The Internet patent land grab, Communications of the ACM, 43, 6, 29-31.
24. Page-Jones, M., 1988. Practical Guide to Structured Systems Design, Englewood Cliffs: Yourdon Press.
25. Rivlin, E., Botafogo, R. and Schneiderman, B., 1994. Navigating in Hyperspace: Designing a Structure-based Toolbox, Communications of the ACM, 37, 2, 87-96.
26. Senn, J. A., 1989. Analysis and Design of Information Systems, New York: McGraw Hill.
27. Simon, H., 1962. The Architecture of Complexity, Proceedings of the American Philosophical Society, 106, 6, 467-482.
28. Taylor, D. A., 1995. Business Engineering with Object Technology, New York: John Wiley & Sons.
29. Teng, J. T. C., Grover, V. and Fiedler, K. D., 1994. Business Process Reengineering: Charting a Strategic Path for the Information Age, California Management Review, 36, 3, 9-31.
30. Thüring, M., Haake, J. M. and Hannemann, J., 1991. What's ELIZA Doing in the Chinese Room Incoherent Hyperdocuments - and How to Avoid Them? Proceedings of Hypertext '91, 161-177.
31. Thüring, M., Hannemann, J. and Haake, J. M., 1995. Hypermedia and Cognition: Designing for Comprehension, Communications of the ACM, 38, 8, 57-66.
32. van Dijk, T. A. and Kintsch, W., 1993. Strategies of Discourse Comprehension, Orlando, FL: Academic Press.