컨텍스트 기반의 웹 애플리케이션 설계 방법론

Context-based Web Application Design

박진수(Jinsoo Park)*

초 록

웹 기능의 향상과 웹 관련 기술의 발전, 레거시 시스템과의 통합 필요성 증대, 자주 변하는 웹 콘텐츠와 구조 등으로 인하여 웹 애플리케이션을 개발하고 관리하는 일이 과거보다 훨씬 더 복잡하게 되었다. 그러나 이러한 다양한 요인들을 고려하는 포괄적인 웹 애플리케이션 설계 방법론은 아직 존재하지 않고 있다. 따라서 본 연구에서는 이러한 요인들을 고려한 컨텍스트 기반의 웹 애플리케이션 설계 방법론을 제시하고자 한다. 본 연구에서 제시하는 방법론에서는 웹 정보를 전달하는 메커니즘에 따라 구분되는 9 종류의 웹 페이지 형태와 웹 페이지 간의 다양한 의미 관계를 정의하는 7 종류의 링크 형태 및 설계 과정 중에 사용되는 여러 종류의 컴포넌트 역할을 구별하는 소프트웨어 컴포넌트 형태 등 다양한 종류의 모델링 기법들을 소개하고 있다. 뿐만 아니라 이 방법론은 '콤펜디엄(compendium)' 이라 불리는 일단의 관련된 정보 클러스터들로 이루어진 독창적인 웹 애플리케이션 모델을 사용하고 있다. 하나의 콤펜디엄은 주제(theme), 컨텍스트 페이지, 링크 및 컴포넌트로 구성된다. 이러한 접근 방법은 모듈 방식의 설계에 유용할 뿐만 아니라 항상 변하는 웹 애플리케이션의 콘텐츠와 구조를 관리하는데도 도움이 된다. 본 연구에서 제시한 방법론은 의미적으로 응집력이 있고 구문적으로 느슨히 결합된 유연한 웹 디자인 산출물을 생성하는데 도움이 될 것이다.

ABSTRACT

Developing and managing Web applications are more complex than ever because of their growing functionalities, advancing Web technologies, increasing demands for integration with legacy applications, and changing content and structure. All these factors call for a more inclusive and comprehensive Web application design method. In response, we propose a context-based Web application design methodology that is based on several classification schemes including a Webpage classification, which is useful for identifying the information delivery mechanism and its relevant Web technology; a link classification, which reflects the semantics of various associations between pages; and a software component classification, which is helpful for pinpointing the roles of various components in the course of design. The proposed methodology also incorporates a unique Web application model comprised of a set of information clusters called *compendia*, each of which consists of a theme, its contextual pages, links, and components. This view is useful for modular design as well as for management of ever-changing content and structure of a Web application. The proposed methodology brings together all the three classification schemes and the Web application model to arrive at a set of both semantically cohesive and syntactically loose-coupled design artifacts.

Keywords : Web Application Design Methodology, Compendia, Access Scope, Link Types, Page Types

본 연구는 2004년도 한국학술진흥재단의 지원에 의하여 수행되었음(KRF-2004-041-B00154). 본 연구의 중간 결과물은 한국학술진흥재단의 규정에 의거하여 한국경영정보학회 2005추계국제학술대회(제주도)에서 발표하였음

* 서울대학교 경영전문대학원/경영대학 조교수

1. Introduction

Few people would disagree that the Web has become a major platform for complex and demanding enterprise applications in many domains, but many would agree that a vast majority of these applications have been being developed off the developer's head in an *ad-hoc* fashion, contributing to problems of user disorientation, content management, maintainability, and quality [1, 10]. For Web applications to become a successful business application model, however, they should be designed in the way that meets its functional requirements as well as nonfunctional ones such as usability, extendibility, and content manageability [21, 29]. In the recent years, there have been some developments towards addressing the gaps and the requirements [3, 9, 17, 29]. Although they have their own merits, they glossed over the differentiating properties of Web applications. They also appear to ignore incorporating ever-advancing Web programming specifications into their studies, most notably interface rendering mechanism that inevitably involves simple yet often complex interaction between the tiers of the application.

Deploying Web applications are often required to be integrated with exusting heterogeneous systems [9, 10]. Companies should be able to leverage and extend existing critical business systems directly to customers, employees, suppliers, and distributors via the Web to improve time to market and reduce the cost of development and deployment. This implies that we should take into account the existing applications during the course of Web application design. One of the emerging technologies is the concept of Web services that may serve as a middleware for the integration of Web applications with legacy systems (e.g., [8], [16]).

Moreover, ever-changing content of Web applications demands for effective content management from the outset of their development [13]. The term "content" in this paper means core deliverables, not including layout, styles, and other presentational enhancements. As Fingar [9] 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 structure. The evolving nature of Web applications inevitably involves updating the structure and presentation of the Web applications. This is particularly true for the Web applications for general public because of the ill-defined end users, and yet, their changing information needs and preferences. Traditionally, various process modeling techniques, such as use case driven modeling [18, 19] and data flow diagramming [6, 12], and design heuristics, such as loose-coupling and tight-cohesion, have been utilized as a means to cope with such dynamicity and reusability issues. They can equally apply to Web application development process-that is, by identifying all the business logic as loosely coupled, tightly cohesive separate modules for the target application domain and then by visualizing facade, workflow, and rules of individual business logic using the logic elements including pages, links, and components.

Following that wisdom, we should model a Web application as a collection of both semantically tight-cohesive and syntactically loose-coupled clusters of information. A semantically tight-cohesive information cluster can be obtained by assembling only highly relevant content around a given subject. Although semantically tight cohesiveness warrants syntactically loose-coupled information clusters, the latter can be fortified by parsimonious use of hyperlinks between information clusters based on natural flow of logic and rhythm of content.

Although the literature on Web application design is vast, the majority is centered on one or two aspects of Web application development. For example, the Relationship Management Methodology [17] focuses on static Web site design, and the work by Conallen [3] has a focal point on the functional aspect. Yet another study describes a higher-level integration of an e-commerce application with existing non-Web applications [9]. However, little research has been done on a comprehensive design method. The primary goal of this study is to provide a "sustainable" methodology for Web-based application design. In response, we propose a context-based Web application design method, which is founded on several classification schemes including: (1) a Web page classification, which is useful for identifying the information delivery mechanism and its relevant Web technology; (2) a link classification, which reflects the semantics of various links; and (3) a software component classification, which is helpful for pinpointing the roles of various components in the course of design. The method is also based on a unique view of the Web application as a set of semantically and syntactically cohesive information groupings called *compendia*, each of which consists of a theme or a subject, its contextual pages, and their contextual links. This view is useful for managing the evolving Web application. The method brings all together the classification schemes and the unique view to arrive at a set of design specifications.

In the next section, we present a literature review on Web application design principles followed by the Web application design model in which application architecture and design primitives will be explained in detail. After the foundational concepts, the main part of the proposed methodology including requirements gathering, analysis, and design will be described. The paper concludes with major contributions and suggestions for future research.

2. Design Principles

There are several fundamental issues in designing Web applications. To get some useful insights into the design of Web applications, this section reviews previous studies on Web application development focusing on design principles.

The first group of design issues is related to identifying, organizing, and managing information 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 the user of conventional applications, however, the user of public Web applications is 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 to provide comprehensive, up-to-date, and evolving content. Considering the above constraint, it seems natural that most of the previous studies on design center on content structuring and its presentation to the user with minimal attention to gathering user information requirements (e.g., [17]). Although a recent method proposed by Conallen [3] has suggested a gathering technique based on the use case approach [18], it basically assumes that the target application's users (and actors) are well defined. This implies that the method by Conallen [3] is more useful for Web applications that have clearly-defined end users.

The second set of issues is related to presenting effective navigational cues to the user [14, 15, 22, 30], offering semantically cohesive content, and creating a syntactically loose-coupled structure for better user comprehension [11, 35, 36]. In cognitive science, comprehension depends on how a user constructs a mental model based on the visible objects and their semantic relations [37]. This implies that semantically cohesive and syntactically loose-coupled themes require less effort in modeling, and hence increase the comprehensibility of the user [35, 36]. In the modeling process, coherence has a positive influence while cognitive overhead has a negative influence on comprehension [4, 35, 36]. For example, a theme is coherent if a user can construct a mental model that corresponds to facts and relations in a real world [20]. The process of a mental model construction involves two types of coherence: local and global coherence. The former is to trees what the latter is to a forest. Local coherence enables a user to relate pieces of information locally while global coherence leads the user to a conclusion based on the set of local relations [37]. On the other hand, cognitive overhead is "the additional effort and concentration necessary to maintain several tasks or trails at one time" ([4], p. 40). This might be inevitable due to the limited capacity of human information processing [25].

These previous studies about Web application development provide 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 [24]. Second, to provide users with an effective navigation, a Web application should be characterized by (1) higher local coherence, i.e., providing appropriate indication of semantic relationships between contextual pages through careful use of hyperlinks within a given information cluster, (2) higher global coherence, i.e., providing adequate overview by aggregating contextual pages into a cohesive and loosely coupled information cluster, and (3) effective navigational facilities, i.e., providing support for navigation with respect to direction (breadth) and distance (depth) within a cluster as well as across clusters [36].

3. The Design Model

This section explains the fundamentals of the proposed methodology, including design architecture, page classification, component classification, compendium and context, and link classification.

3.1 Web Application Architecture

Web application architecture is important because it determines actual level of application performance, resource utilization, and maintainability. Like many other applications, Web applications begin with logical architecture and then move on to physical architecture. Logical architecture is based on a factoring of the application into logical layers by functions, while physical architecture is an actual implementation of the logical architecture. Physical architecture depends on implementation strategy and technological constraints. In this paper, we focus on logical architecture as shown in Fig. 1. The logical architecture consists of presentation, business, data access, and data layers.

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 others like MIME documents. The presentation layer does all its work through calls to the business layer.

Business layer includes 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 breadth of business logic can be addressed with three categories: business facades, workflow, and business rules. Business facades are interfaces that expose business services to consumers through the presentation layer while hiding implementation details. Workflow is a sequence of steps that involve state transformations. For example, business logic governing the sequence of steps of an order-from shopping cart to subtotal, tax, shipping, and grand total calculations, to credit card authorization, to packaging and shipping, to payment-is workflow. Business rules include logic controlling the implementation of autonomous transactions, such as rules that limit acceptable data values and conditions where data may be accessed. The business layer could be designed with only a Web server or a combination of a Web server and an application server. In the former case, a Web server alone provides the Web application's functionality. It takes a request and passes it to a server-side program that handles the request. The server-side program looks up, say, the pricing information from a data store and uses the retrieved information to formulate the HTML response. The latter case resembles the former case in that the Web server still delegates the response generation to a server page. However, the business logic for the pricing lookup is now put on the application

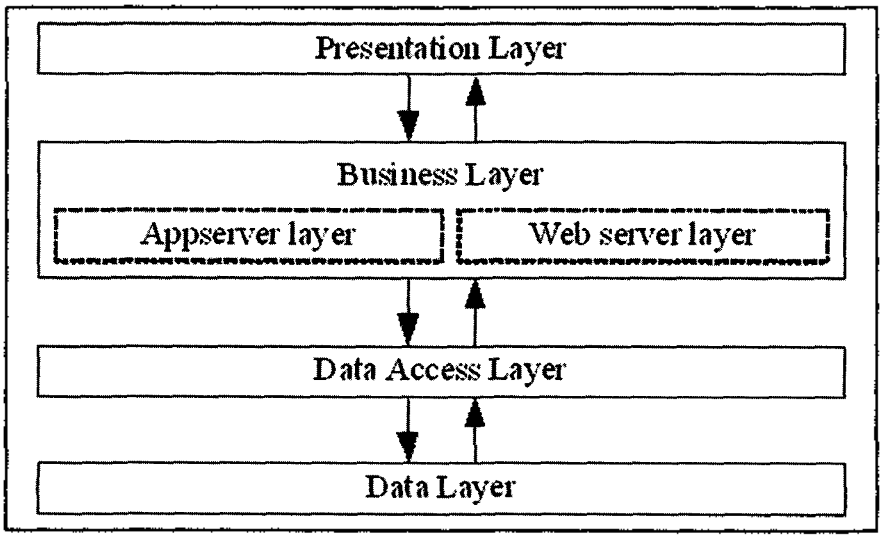


Fig. 1. Logical Architecture for the Methodology

server. We stick with the more general business layer design (i.e., business layer with two server sub-layers). The business layer would be implemented as reusable components and server pages.

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. The data access layer can also isolate the rest of the application from changes to the data store. The data layer includes data and data store software, including relational databases, e-mail stores, message queues, and directory services. Database may be comprised of catalog data, customer account data, order data, and session state data for an e-commerce application.

3.2 Page Classification

When we browse a Webpage, in general, we click on a link that is connected to the Webpage. At this moment, one of the two general Webpage rendering processes is involved depending on the page type: request/response and request/execution/response sequences. If the requested page is either a pure HTML page or any other MIME documents (e.g., a PDF document), the server simply responds with the requested page. This kind of page is called a *static* page whose exact content has already been determined by the page author at some time before any request for the page is made. Regardless of who requests the page and when and how the request is made, the content and appearance is always the same. In addition, depending on whether a static page contains a data capturing form (e.g., an HTML form), it is called either an *interactive* or a *non-interactive* page. In some cases, a Webpage's behavior is governed by a client-side control element such as a style sheet or a client-side script, which is called a *client preprocessor* page.

On the other hand, a *dynamic* page is generated on request by a page called a *server* page. Even though there are two different ways of generating a dynamic page-client-side and server-side models-we focus on the latter model due to some drawbacks of the former, such as poor security and slow download. Upon execution of a server page, a Web server returns an updated version of the same server page, or possibly on some occasions, a separate second page to the presentation layer. A server page returning updated itself is called the *interlayer* server page because it directly spans over the business layer and the presentation layer. Alternatively, a server page delegating a final response to a different page by passing a data string or state information is called the *withinlayer* server page. There is a mixed case, i.e., a server page could be a combined form of the interlayer and within layer server pages. For example, a server page could return a data capturing form generated by form generation logic in the page and write the received data via the form to a data store, and finally redirect the user to a different page with or without a data string or state information if the writing is successful. This sort of server page is called the *hybrid* server page. Just like a static page, a dynamic page may or may not include a dynamically generated data capturing form. In addition, some common part of business logic (e.g., data access specification) is often implemented as a preprocessor page for the given server page. This type of page is called a *server preprocessor* page.

Based on the above page types, let us consider a simple example. A user authentication process could involve the following pages: a login page (i.e., a static interactive page on the presentation layer), a style sheet page (i.e., a client preprocessor page on the presentation layer), a form validation script page (i.e., another client preprocessor page on the presentation layer), a login verification page (i.e., a withinlayer server page on the business layer), an include page specifying an access to login information stored in a data store (i.e., a server preprocessor page on the business layer), and a confirmation page (i.e., an interlayer server page on the business layer from which a non-interactive dynamic page is derived for the presentation layer). Fig. 2 shows the page classification, divided into two broad categories each of which is comprised of several context page types.

3.3 Component Classification

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 [10]. 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 [8, 9, 16].

In a Web application, software components are called during execution of relevant code inside a page. They provide various services to the calling page, such as data access and file manipulation, to eventually render information to end users. They can be called from the presentation or business layer. The proposed methodology, therefore, explicitly takes components into account in four context component types: (1) *Presentation layer components* include browser components (e.g., components available through Internet Explorer Administration Kit) as well as operating system components (e.g., XML Document Object). (2) *Webserver layer components* are provided by Web server software and its related technologies such as

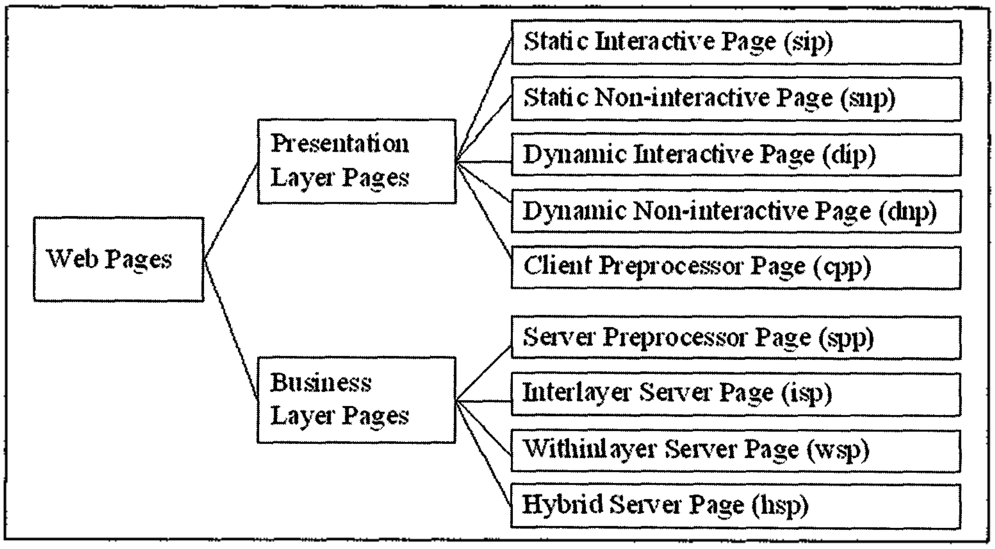

Web Pages
Presentation Layer Pages
Business Layer Pages
Static Interactive Page (sip)
Static Non-interactive Page (snp)
Dynamic Interactive Page (dip)
Dynamic Non-interactive Page (dnp)
Client Preprocessor Page (cpp)
Server Preprocessor Page (spp)
Interlayer Server Page (isp)
Withinlayer Server Page (wsp)
Hybrid Server Page (hsp)

Fig. 2. Page Classification

SMTP for e-mail. (3) *Appserver layer components* are those related to workflow and business rules. They could be newly developed, be available from existing business applications, or be purchased from third-parties. (4) *Data access layer components* support all data access requirements of the business layer (e.g., ODBC, OLEDB or JDBC). These components are defined in the business and data access layers during the design phase. Note that we rule out any components triggered automatically such as plug-ins or a MIME application because the access to such components is implied by semantics of the corresponding design element. In other words, we consider only the access to components that requires explicit calls in coding design.

3.4 Link Classification

A link is an associative connection between Web application elements. Conversely, an anchor identifies the precise endpoint of a link. It can be understood in terms of both semantic and syntactic context. For example, if information is presented in a sequence of pages, the links between the pages play syntactic roles. If a click on a link causes user data to be stored in 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 has semantics that 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 seven context link types based on their semantic and syntactic context: *anchor link* (〈a〉), *directive link* (〈d〉), *call link* (〈c〉), *build link* (〈b〉), *enjoin link* (〈e〉), *form link* (〈f〉), *and intermediate link* (〈i〉). Note that a links does not always mean a hyperlink created by the anchor tag (i.e., 〈a〉) of the HTML. It could be an event as a form of link that triggers the generation of a dynamic page or a link denoting a redirection.

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 directs a browser or a server to apply a specific setting to either a client preprocessor page or a server preprocessor page. For example, a page may contain a directive that instructs a browser to apply a style sheet to the 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. A good example is a link to a server page that generates a billing statement upon completion of an order transaction. Unlike the build link type, the enjoin link type directs a withinlayer server page to request a different page. During the redirection, a server page normally passes a data string or state information to the redirected page. The form link type is used for a connection between an interactive page and a server page to process data submitted by a data capturing form. Finally, the intermediate link type is a placeholder used during the analysis phase. It will be replaced with one of the six other link types during the design phase.

3.5 Compendium and Context

A Web application can be viewed as a collection of information clusters. Highly cohesive and loosely coupled clusters can make change management easier [9, 28]. A resulting semantically cohesive and syntactically loose-coupled information cluster is called a *compendium*, which has the following characteristics: (1) it is comprised of one or more context pages, zero or more context components, one or more context links, and zero or one theme page; (2) the theme of a compendium is also its name; (3) the tight semantic cohesiveness and loose syntactic coupling of a compendium are achieved by connecting context pages and components necessary to complete a business activity via context links around the theme of the compendium. Fig. 3 shows a view of a Web application in terms of compendium and context elements. The theme page is optional and could be any presentation layer page types excluding the client preprocessor page.

4. Methodology

The proposed Web application design methodology primarily consists of analysis and design phases (see Fig. 4). We will also provide some general ideas for the requirements gathering and implementation phases.

4.1 Requirements Gathering

As mentioned earlier, the users of the *public* Web applications are vaguely defined. This may hamper the requirements gathering process. We suggest a requirements gathering idea 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 [23]. Although the rules are changing as more and more Internet-related patents and legal battles are looming [27], getting ideas from existing Web applications would be acceptable behavior. There are numerous existing Web sites, which are gradually becoming parts of people's lives. This implies two things: (1) the vast number of currently running Web applications could provide designers with the inspiration for gathering requirements; and (2) they may also provide designers with clues for "sustainable" requirements because the deployed Web sites have evolved over time and been massively tested with real users.

Therefore, designers might consult similar Web sites to learn what should be the essential content for the application. This sort of technique may be called

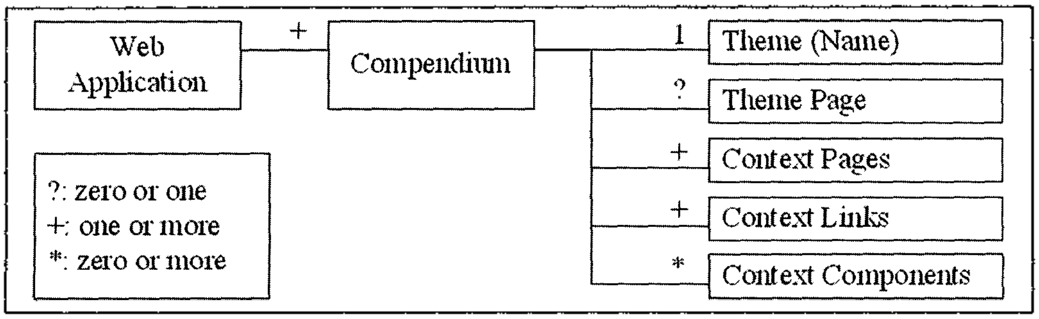


Fig. 3. Compendium and Context View

"inspired assimilation." It could have some advantages over a simple dependence on designers' knowledge and/or conjecture. First, they could identify relatively complete requirements if the consulted Web sites are fairly stable. Second, they could identify the requirements that have been widely accepted. This would make it much easier to find the application-specific requirements once designers have recognized the general requirements. The suggested idea could be augmented by the model-based approach [33, 34]. Its goal is to construct application models that show the structure, processes, and resources of a business as simply and directly as possible. Based on the approach, developers may be able to gather requirements by applying the conventional information gathering techniques (e.g., [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.

4.2 Analysis

Analysis is the process of examining the requirements and developing the blueprint of an application to be built, which manifests higher global and local coherence. It begins with organizing the themes into higher-level hierarchies, which are, then, gradually refined into lower-level details. Complexity frequently takes the form of a hierarchy, which is a major facilitating factor enabling us to understand and describe complex objects and their parts [5, 32]. The following subsections illustrate

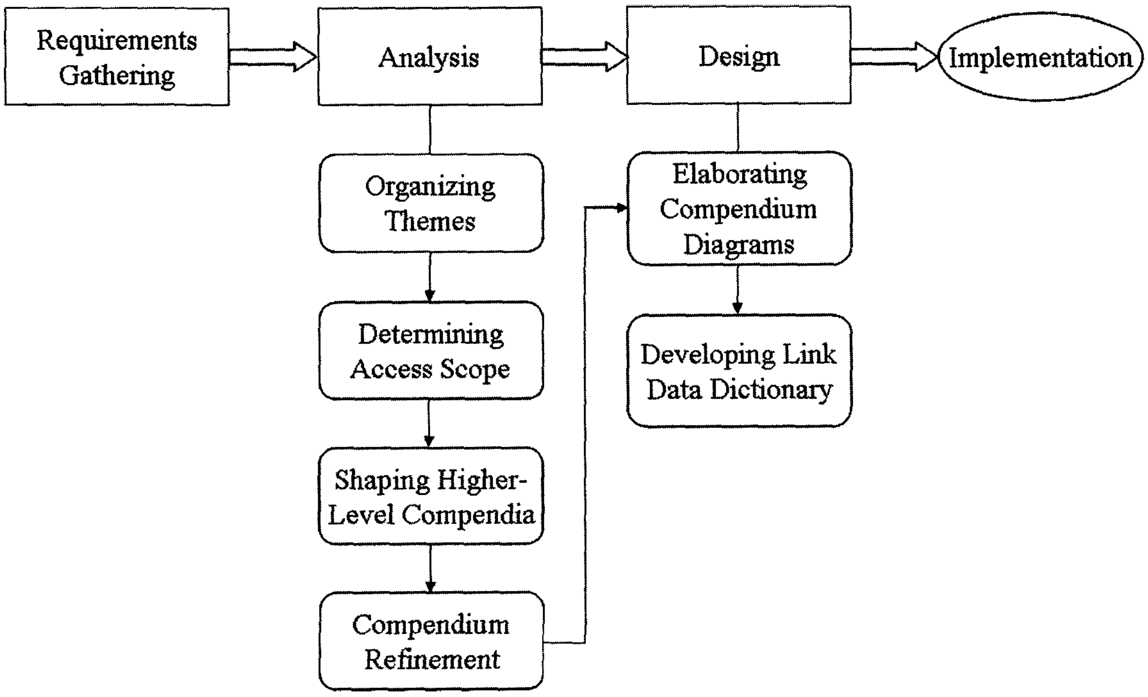


Fig. 4. Context-based Web Application Design Process

the analysis process.

4.2.1 Organizing Themes

First of all, it is possible for a target Web application to have one or more domains (sub-applications). For example, an online academic conference manager application may consist of three domains: Author, Administration, and Conference information domains. One of them is designated as a default domain (e.g., in this paper, the Author domain). Once we clarify domains, three different tasks are performed on each domain: (1) identifying top-level themes, (2) decomposing them into lower-level themes, and (3) organizing the themes into trees.

A theme can be either descriptive (factual/definitional) or prescriptive (process/procedural). A site consisting of only descriptive themes can be called a Web site because it does not invoke business logic [7]. By contrast, the prescriptive themes normally involve workflow and business rules. A Web application is comprised of both types of themes [3]. Once the top-level themes are decomposed into lower levels, they are located by dot separators (e.g., Coauthor.Registration) within a tree. On some occasions, a theme such as "Registration" may repeat in a tree since it is required for both the primary author and coauthor. In such a case, we simply repeatedly show it in the tree. Fig. 5 shows a theme tree for Author domain of the aforementioned conference manager.

Although an authentication process is an integral part of many themes, it is implicitly shown in the tree (e.g., Initial Sub(mission) requires authentication). Note that the tree shows only themes. It does not show context pages, components, and link types. It does not have to be balanced, either. Some themes are about a definition or fact, while others are about a process or procedure, implying that in actual implementation, we need various context page types. We may continue to refine the identified themes for a given domain until further refinements are impractical, i.e., until a leaf node can be described by a single page that will be the same as the theme page. For example, we might have had the 'Submission Guideline' theme under the 'Initial Sub(mission)' or 'Final Sub(mission)' theme in Fig. 5. We can do so if the guideline is comprised of more than one page. However, we assume that it is a single page description, which will be the same as the theme page of the would-be theme.

The overall depth and breadth of the tree depends

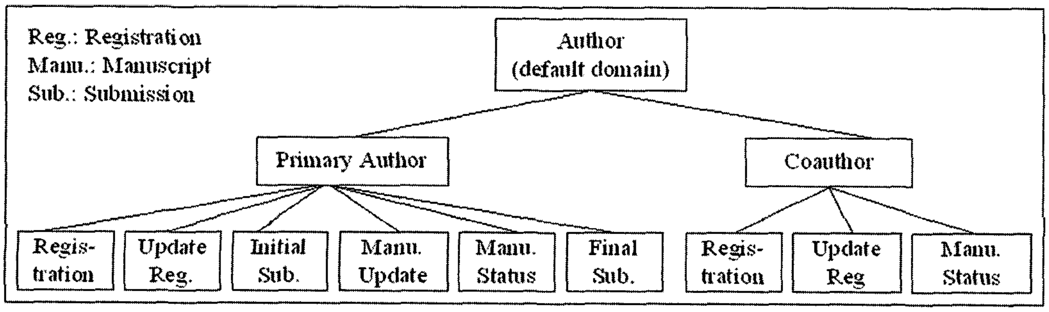


Fig. 5. Theme Tree for the Default Author Domain (Default Domain)

on the level of complexity and the size of a domain as well as the implementation strategy. For example, if a designer wants to minimize the number of clicks, she might want to make a tree wider rather than deeper. The degree of information granulation also determines, as Fingar [9] 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 the subsequent lower levels. We suggest that a designer come up with a tree as fine-grained as possible and then combine nodes in line with her implementation strategy.

4.2.2 Determining Access Scope

This step determines the accessibility to various themes from within a page in each domain based on its theme tree. Broader access scope normally makes a page crowded with links. However, it can bring a shallow structure, which may require smaller number of clicks to get to a destination page. We determine the three kinds of access scope: the *homepage scope*, *common scope*, and *expansion scope*.

The homepage scope chooses the number of themes that will be appeared as links on the (default) domain homepage. It may include all or a subset of the themes identified in a tree. For example, you may access all themes from within the homepage if you put every theme in Fig. 5 onto the homepage. The *common* scope is a subset of the homepage scope and defines a fixed number of themes that will be common to every page as links, including the (default) domain homepage. The common scope and the homepage scope could be identical, especially when the application is small. The *expansion* scope is affected by the two previous types of scope. For example, let's assume that the homepage scope and the common scope contain only 'Primary Author' and 'Coauthor' themes. Then, the expansion scope for the 'Primary Author' can include the themes up to the third. This means that,

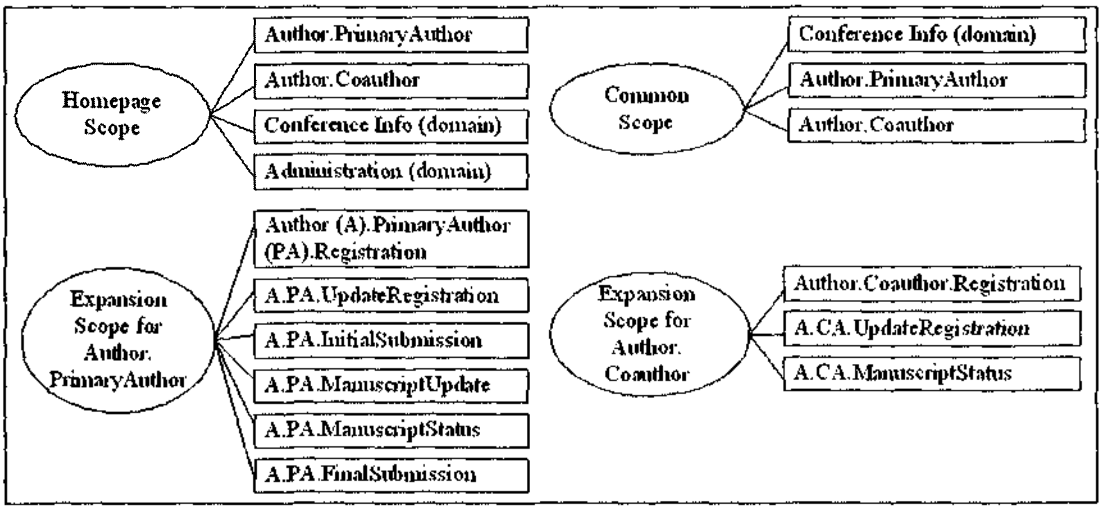


Fig. 6. Sample Scope Diagrams for the Default Author Domain

upon deployment, once a user gets to a 'Primary Author' theme page, she can directly access the lower-level themes without jumping to another page. Fig. 6 shows each sample of the homepage, common, and expansion scope for the default 'Author' domain. The 'Administration' and 'Conference Information' are other domains. They are shown to imply a relation with the default 'Author' domain, although, upon implementation, a link to the 'Administration' domain doesn't have to be accessible by non-administrators. In any case, actual scope determination should take into account implementation strategy.

4.2.3 Shaping Higher-Level Compendia

The three types of scope can be called "meta-themes," meaning that each of the homepage and expansion scope may have a theme page (i.e., "homepage scope theme page" and individual "expansion scope theme pages," respectively). These two types of scopes consist of one or more themes, each of which may have its own theme page. The common scope is *excluded* because it simply represents a group of themes common to each page in a given domain as well as a subset of the homepage scope. Recall that a compendium is comprised of an *optional* theme page and one or more context pages, one or more context links, and zero or more components (see Fig. 3). We assume each theme has its own theme page.

This step shapes high-level compendia out of the scope diagrams for each domain. In a higher-level compendium diagram, any page or theme is denoted as a rectangle with its descriptive name in it. There are several major principles in drawing a higher-level compendium. First, we must take account of only the presentation layer pages (see Fig. 2) that are semantically related to a given theme (one exception is that the compendium diagram for the homepage scope may show the presentation layer pages and themes that may not be semantically cohesive). Second, the diagrams for the expansion scope and "non-leaf" themes must show all semantically-related presentation

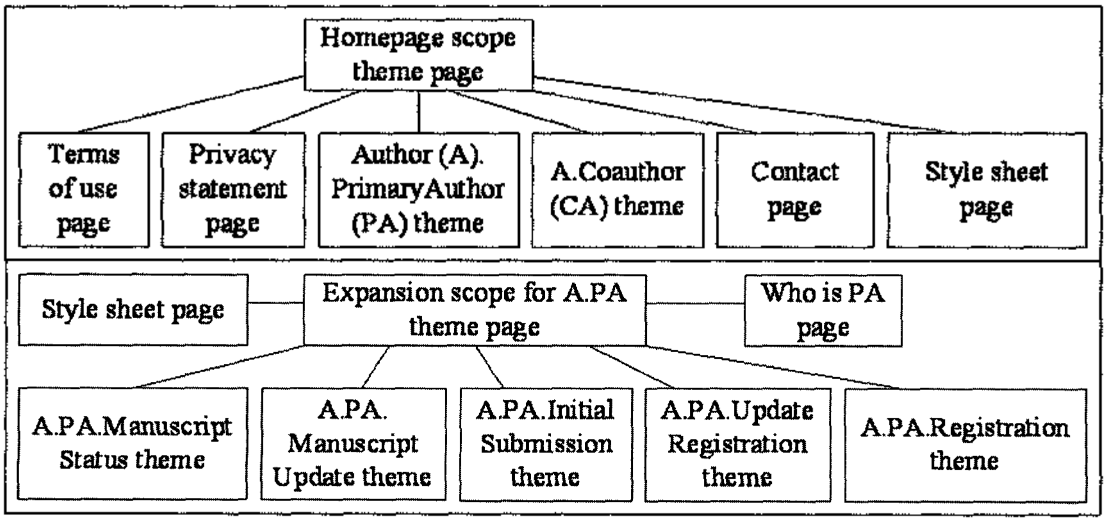

Homepage scope theme page
Terms of use page
Privacy statement page
Author (A). PrimaryAuthor (PA) theme
A.Coauthor (CA) theme
Contact page
Style sheet page
Style sheet page
Expansion scope for A.PA theme page
Who is PA page
A.PA.Manuscript Status theme
A.PA. Manuscript Update theme
A.PA.Initial Submission theme
A.PA.Update Registration theme
A.PA.Registration theme

Fig. 7. Sample Higher-level Compendium Diagrams

layer pages and themes. Third, the compendium diagrams for "leaf" themes will show only semantically-related presentation layer pages. It also shows other related theme pages (not themes) to show a logical relationship between the leaf theme and other themes except for the themes in the common scope. Fourth, we draw diagrams in a top-down manner. Therefore, we first come up with compendium diagrams for meta-themes. Fig. 7 shows a series of sample compendium diagrams drawn based on the above major principles. Note that, in the diagram, the non-directed lines used to connect pages and themes are not links, rather they simply show syntactic relationships.

As illustrated in Fig. 7, there is no indication about page, component, and link types. At this stage, we are just concerned about only themes and the presentation layer pages. There are several important points regarding higher-level compendia diagrams. First, they show only the presentation layer pages (along with themes). This is necessary to separate "what" from "how" aspects of an application design. Second, pages should be as fine-grained as possible in a higher-level compendium diagram. The fine granulation of pages gives designers a higher flexibility when they later need to collapse them into a smaller number of pages based on their implementation strategies. It also makes customization and/or reuse easier. Third, if a certain page must have multiple versions, it can be denoted as overlapped rectangles. Finally, the order of pages and themes in a higher-level compendium diagram does not carry a meaning. It will be taken into account later.

4.2.4 Compendium Refinement

The higher-level compendium diagrams are refined by incorporating the presentation layer context page types (i.e., abbreviated as 'sip' for static interactive page, 'snp' for static non-interactive page, 'dip' for dynamic interactive page, 'dnp' for dynamic non-interactive page, and 'cpp' for client preprocessor page) and three context link types (i.e., abbreviated as ⟨a⟩ for anchor link, ⟨d⟩ for directive link, and ⟨i⟩ for intermediate link). The remaining types of pages, links, and components will later be taken into account in the design phase. Note that a link can be unidirectional, bidirectional, or conditional. It is a design issue whether we should show a bidirectional link between two pages

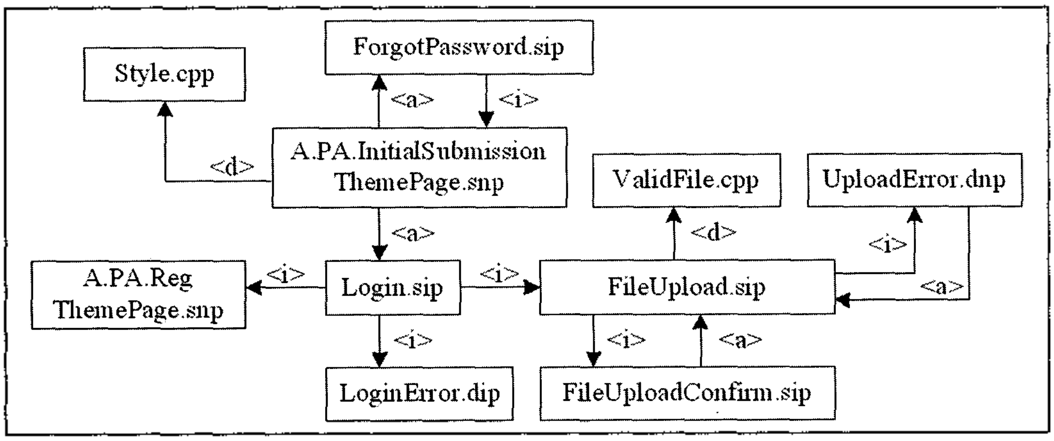


Fig. 8. A Refined Compendium Diagram

or let the user press the backward button. A conditional link is determined by the context of a given compendium. In addition, we may have a situation where two or more pages must be sequentially accessed. This means that not all context pages are directly linked to the theme page of a compendium. We will use unidirectional links to cover all the cases and for orderly linking to the pages and component. Fig. 8 shows a refined compendium diagram for the leaf theme 'Initial Submission.' Although the higher-level compendium diagram for the theme is not shown in Fig. 7, the refined diagram off the page types and link types becomes the higher-level diagram.

Regarding Fig. 8, several things should be noted. First, the meta- and non-leaf themes shown in higher-level compendium diagrams must be shown as theme pages in the refined compendium diagrams (e.g., A.P.A.RegistrationThemePage.snp). Any affiliated or third-party Websites can also be handled in this way. Second, a style sheet page for a given theme should appear only once and linked to the theme page to avoid clutter (e.g., Style.cpp). Third, the intermediate link type is used when the connection between two pages should be mediated by one or more business layer pages and/or components. This means that one or more of the other six link types will replace an intermediate link in the design phase. For example, we need more pages and components to accomplish the file upload task, which implies the intermediate link between the 'FileUpload.sip' and the 'FileUploadConfirm.sip' must be replaced with some other link types (which will be explained in detail in a later section). The two pages are connected in both directions because a primary author may submit more than one manuscript. Fourth, the "external" theme page (i.e., A.P.A.RegistrationThemePage.snp) is shown because a primary author should register after a couple of failed login attempts. Fifth, we assume that the 'LoginError.dip' will show an error message as well as the exact login form contained in the 'Login.sip' page. As such, the pages and their links are affected by implementation envisioning and strategy in the designer's mind.

4.3 Design

The design phase adds more details to the analysis artifacts of a Web application to make them realizable in software by performing the two tasks: (1) elaborating the refined compendium diagrams with the consideration of all the page types, link types, component types, and the application architecture, and (2) developing the link data dictionary (LDD) that details data flows along the links across pages and components as well as application layers.

4.3.1 Elaborating Compendium Diagrams

This step only shows the rendering process of the presentation layer pages. We defer the underlying logic to the next step. Aforementioned, there are five types of presentation layer pages, six main link types, four component types, and four business layer page types. We place all the design primitives in an appropriate layer of the application architecture. At this step, we exclude any related theme pages (e.g., A.P.A.RegistrationThemePage.snp in Fig. 8) as well as affiliated or third-party Websites. Each design primitive must be denoted by the corresponding given suffix (e.g., 'snp' for static non-interactive page) or notations (e.g., ⟨f⟩ for form link). Any components that will be automatically triggered should be excluded. Fig. 9 shows an elaborated compendium diagram for Fig. 8.

4.3.2 Developing Link Data Dictionary

A data dictionary is a central repository of all data definitions for the target application. There are many advantages of using it for application development [31]. Developing a *link data dictionary* for the target application is the final step before its implementation. Fig. 10 shows elements of the link data dictionary: (a) a sample link data flow dictionary, (b) a sample page/component dictionary, (c) a sample link data structure dictionary, and (d) a sample link data element descriptions. The *link data flow dictionary* identifies data flows and structures over the link and their origins and destinations. The *link data structure dictionary* contains detailed definitions for data structures, which are built on four data relationships (for more details, see [31], pp.184-204). Data elements in each data structure are described by its name, description, alias, length, and value in the *link data element description*. The *page/component dictionary* provides, if necessary, logic summaries for either pages or software components, especially, application components.

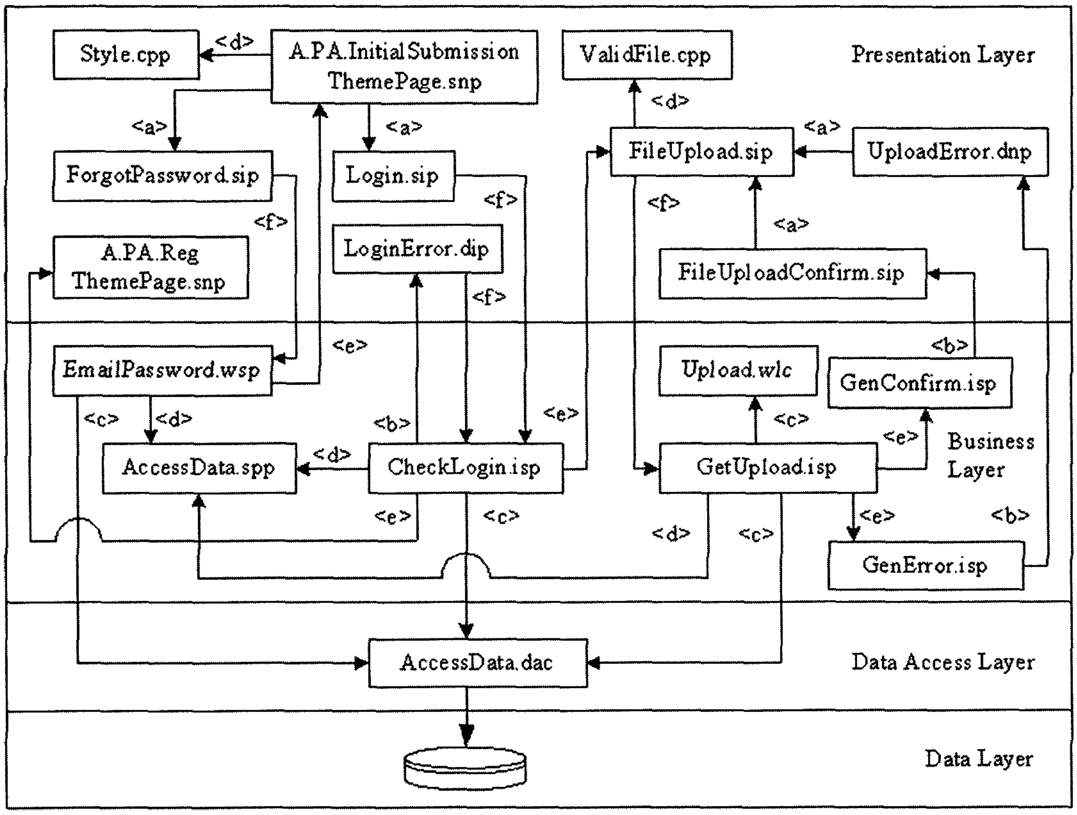
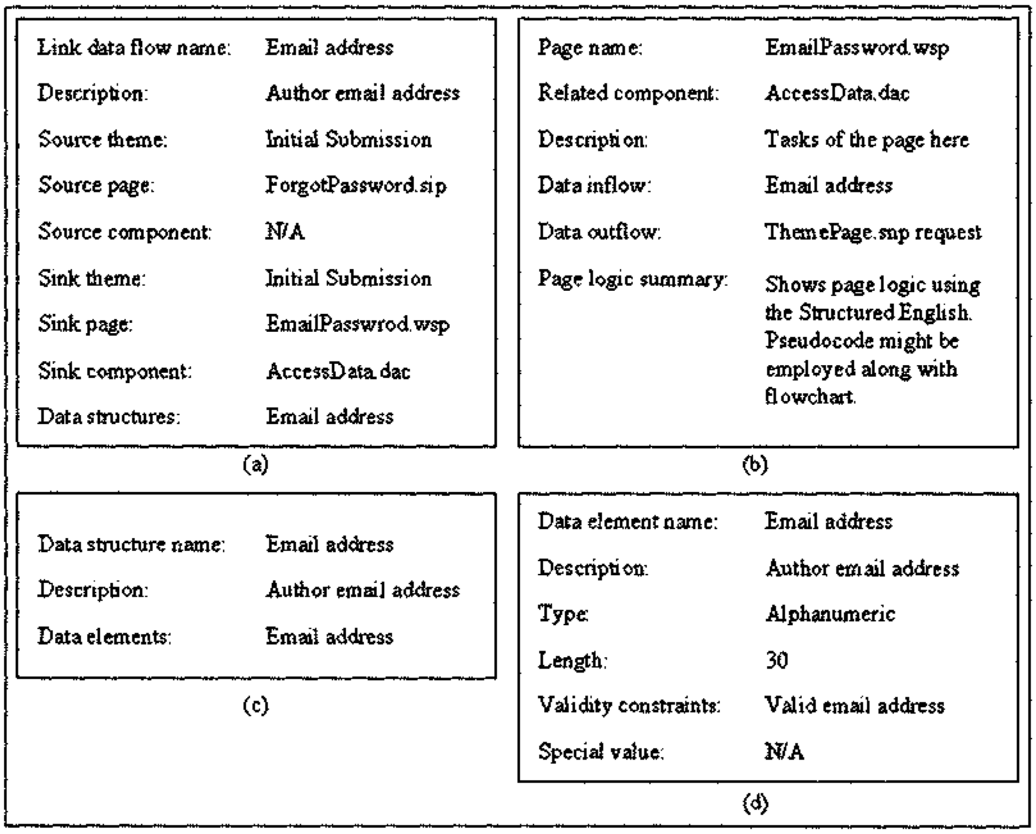


Fig. 9. An Elaborated Compendium Diagram



Link data flow name:	Email address
Description:	Author email address
Source theme:	Initial Submission
Source page:	ForgotPassword.sip
Source component:	N/A
Sink theme:	Initial Submission
Sink page:	EmailPasswrod.wsp
Sink component:	AccessData.dac
Data structures:	Email address

(a)

Page name:	EmailPassword.wsp
Related component:	AccessData.dac
Description:	Tasks of the page here
Data inflow:	Email address
Data outflow:	ThemePage.snp request
Page logic summary:	Shows page logic using the Structured English. Pseudocode might be employed along with flowchart.

(b)

Data structure name:	Email address
Description:	Author email address
Data elements:	Email address

(c)

Data element name:	Email address
Description:	Author email address
Type:	Alphanumeric
Length:	30
Validity constraints:	Valid email address
Special value:	N/A

(d)

Fig. 10. A Link Data Dictionary

5. Discussion

The proposed methodology is an attempt to incorporate the environmental changes in Web application development into its design, including growing functionalities [10], advancing Web technologies [1], increasing demands for integration with non?Web applications [9, 10], and ever-changing content and structure [9, 13]. Each area of change lends itself to its own issue. The growing functionality introduces the issue of how to separate the "what" from the "how" aspects in designing Web applications. Addressing the issues is our first contribution. The former aspect is addressed via the requirement gathering and the analysis phase, while the latter by the design phase. The proposed classifications of Web pages, components, and links as well as the concept of the link data dictionary play critical roles in this separation view. The presentation layer pages constitute the "what" aspect because they specify "what information content" should be delivered to the user. The business layer pages and components along with the link data dictionary signify the "how" aspect because they represent the logic behind the rendering process of the presentation layer pages. The link types, then, work as the glue that associates the pages and components.

Our second contribution comes from the reflection on the rapid advances in Web technologies. This raises a different issue of how to incorporate the technology elements into designing Web applications. We address the issue with the notion of granulation of Web pages and components. The concepts of dynamic pages, server pages, client and server preprocessor pages, and the four types of components encompass all the available technology elements for the design. They enable us to take into account the various markup languages, scripting technologies, and component technology in designing Web applications. Related to the second contribution is an explicit consideration of core business processes running on non-Web applications as components for the integration with Web applications.

Another important issue ensues from the ever-changing content and structure of a Web application. Those changes inevitably bring up changes in its navigational structure and presentation [9]. The way of managing changes in the structure and presentation affects the degree of the global and local coherence, and, hence, the level of the user cognitive overhead and disorientation. Therefore, the issue can be reduced to the matters of global and local coherence and effective change management. A higher global and local coherence and its resulting better user comprehensibility can be achieved by presenting effective navigational structure [14, 15, 22, 30], while the manageability and modifiability of a Web application can be enhanced by maintaining semantically cohesive content and syntactically loose-coupled information clusters [35, 36]. In other words, an effective navigational structure requires an effective arrangement of navigational cues (i.e., links) in both global and local levels, while an efficient change management requires loosely coupled and tightly cohesive information clusters. Indeed, "modules" with those characteristics make it easier to manage changes [28].

The proposed methodology makes another contribution by addressing the two critical matters via the concepts of compendium and access scope. A compendium is semantically cohesive because it is all about a theme that is detailed by its context pages, links, and components. The individual compendium is also syntactically loose-coupled because each of them deals with a different theme. The characteristics of a compendium should make it easier to update, add, or delete pages and links without serious "ripple effects." On the other hand, the three types of access scopes are useful tools for enhancing the global and local coherence because they present the navigational structure given by the internal and external interconnections of compendia. Changes in compendia can be easily reflected by corresponding alterations in the access scope without affecting the overall level of user comprehensibility.

The methodology also has implications for managers. It suggests a user information requirements gathering technique and encompasses the recent environmental changes in Web application design. This would give managers insights into what should be involved in a Web application development project in terms of information requirements and Web application infrastructure. The methodology also takes explicitly into account the integration of Web applications with existing non-Web counterparts as well as business partners' and customers' business processes by allowing them to be connected to appropriate compendia. This would give managers an idea about what core business processes of non-Web applications, business partners, and customers should be integrated with the target Web application.

6. Conclusion

Developing a Web application is just beyond converting documents into markup counterparts. It involves various Web technologies and often requires integration with other technologies. It frequently implements fairly complex logic either through components or within pages. Moreover, due to an inherent characteristic of the Web, the size of an application can grow indefinitely. The designers of Web-based applications seem to get a sizable amount of pressure on delivering high-coherence and low-cognitive-overhead applications along with "sustainable" contents. As a consequence, developing Web application becomes complex and time-consuming [10]. The primary goal of this paper is to provide a "sustainable" methodology for Web-based application design. We conclude with some remarks on implementation and future research.

As for the implementation, there are several issues that should be addressed during design. First, the designer should determine appropriate page sizes in accordance with the intended breadth and depth of an application. The decision will lead to a proper organization of highly-granulated pages identified in the analysis process. Second, the common scope is comprised of only themes but can include links to pages repeating on every Webpage such as 'contact' page or 'help' page. Third, although the proposed methodology seems to be a "waterfall" model, an iterative approach should be employed.

Related to future research, there are several suggestions. First, we assume that software components are out there. We do not consider component based application development tasks: building, searching, customizing, and composing [26]. Developing an effective method that deals with the tasks is another topic. Second, we did not specifically deal with page layout design issues, which may include identifying individual content elements, positioning the elements, determining page sizes, and incorporating other interface elements. Finally, this paper lacks detailed implementation and testing procedures. Although, in most cases, implementing the design specifications proposed in the methodology would be a straightforward mapping process, it would be a complete method provided there are detailed implementation and testing procedures. Despite these factors, we believe that the method that has been developed by rigorously applying a number of new concepts should provide a consistent and manageable way for Web-based application design.

References

[1] Britton, K.H., Y. Li, R. Case, C. Seekamp, A. Citron, B. Topol, R. Floyd, and K. Tracy, "Transcoding: Extending e-Business to New Environments," *IBM Systems Journal*, Vol. 40, No. 1, 2001, pp. 153-178.

[2] Byrd, T.A., K.L. Cossick, and R.W. Zmud, "A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques," *MIS Quarterly*, Vol. 16, No. 1, 1992, pp. 117-138.

[3] Conallen, J., *Building Web Applications with UML*, Addison-Wesley, Reading, 2000.

[4] Conklin, J., "Hypertext: An Introduction and Survey", *IEEE Computer*, Vol. 20, No. 9, 1987, pp. 17-40.

[5] Cook, M.A., *Building Enterprise Information Architectures: Reengineering Information Systems*, Prentice Hall, Upper Saddle River, 1996.

[6] DeMarco, T., *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, 1979.

[7] Engelbart, D.C., "Toward Augmenting the Human Intellect and Boosting Our Collective IQ," *Communications of the ACM*, Vol. 38, No. 8, 1995, pp. 30-33.

[8] Ferris, C., and J. Farrell, "What are Web Services," *Communications of the ACM*, Vol. 46, No. 6, 2003, p. 31.

[9] Fingar, P., "Component-based Frameworks for E-commerce," *Communications of the ACM*, Vol. 43, No. 10, 2000, pp. 61-66.

[10] Flurry, G., and W. Vicknair, "The IBM Application Framework for e-Business," *IBM Systems Journal*, Vol. 40, No. 1, 2001, pp. 8-24.

[11] Fraternali, P., "Tools and Approaches for Developing Data-intensive Web Applications: A Survey," *ACM Computing Surveys*, Vol. 31, No. 3, 1999, pp. 227-263.

[12] Gane, C., and T. Sarson, *Structured Systems Analysis*, Prentice-Hall, Englewood Cliffs, 1979.

[13] Guenther, K., "What is a Web Content Management Solution?" Online, Vol. 25, No. 4, 2001, pp. 81-84.

[14] Halasz, F., and S. Schwartz, "The Dexter Hypertext Reference Model," *Communications of the ACM*, Vol. 37, No. 2, 1994, pp. 30-39.

[15] Hardman, L., and B. Sharrat, "User-centered Hypertext Design: The Applications of HCI Design Principles and Guidelines," *in Hypertext State of the Art*, R. Mcaleese and C. Green (eds.), Intellect, 1990, pp. 252-259.

[16] Huang, Y., and J. Chung, "A Web Services-based Framework for Business Integration Solutions," *Electronic Commerce Research and Applications*, Vol. 2, No. 1, 2003, pp. 15-26.

[17] Isakowitz, T., E.A. Stohr and P. Balasubramanian, "RMM: A Methodology for Structured Hypermedia Design," *Communications of the ACM*, Vol. 38, No. 8, 1995, pp. 34-44.

[18] Jacobson, I., M. Christerson, P. Jonsson and G. Overgaard, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Wokingham, 1992.

[19] Jacobson, I., G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley Longman, Reading, 1999.

[20] Johnson-Laird, P.N., "Mental Models," in *Foundations of Cognitive Science*, M.I. Posner (ed.), MIT Press, Cambridge, 1989, pp. 469-499.

[21] Johnson, R.D., and D. Reimer, "Issues in the Development of Transactional Web Applications," *IBM Systems Journal*, Vol. 43, No. 2, 2004, pp. 430-440.

[22] Kahn, P., "Visual Cues for Local and Global Coherence in the WWW," *Communications of the ACM*, Vol. 38, No. 8, 1995, pp. 67-69.

[23] Makadok, R., "Can First-Mover and Early-Mover Advantages be Sustained in an Industry with Low Barriers to Entry/Imitation?" *Strategic Management Journal*, Vol. 19, No. 7, 1998, pp. 683-696.

[24] Marchionini, G., and B. Schneiderman, "Finding Facts and Browsing Knowledge in Hypertext Systems," *IEEE Computer*, Vol. 21, No. 3, 1988, pp. 70-80.

[25] Miller, G.A., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capability for Processing Information," *The Psychology Review*, Vol. 63, No. 2, 1956, pp. 81-97.

[26] Mili, H., F. Mili and A. Mili, "Reusing Software: Issues and Research Directions," *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, 1995, pp. 528-561.

[27] O'Reilly, T., "The Internet Patent Land Grab," *Communications of the ACM*, Vol. 43, No. 6, 2000, pp. 29-31.

[28] Page-Jones, M., *Practical Guide to Structured Systems Design*, Yourdon Press, Englewood Cliffs, 1988.

[29] Papazoglou, M.P., and W.V.D. Heuvel, "Service-oriented Design and Development Methodology," *International Journal of Web Engineering and Technology*, Vol. 2, No. 4, 2006, pp. 412-442.

[30] Rivlin, E., R. Botafogo, and B. Schneiderman, "Navigating in Hyperspace: Designing a Structure-based Tpolbox," *Communications of the ACM*, Vol. 37, No. 2, 1994, pp. 87-96.

[31] Senn, J.A., *Analysis and Design of Information Systems*, McGraw Hill, New York, 1989.

[32] Simon, H., "The Architecture of Complexity," in *Proceedings of the American Philosophical Society*, Vol. 106, No. 6, 1962, pp. 467-482.

[33] Taylor, D.A., *Business Engineering with Object Technology*, John Wiley & Sons, New York, 1995.

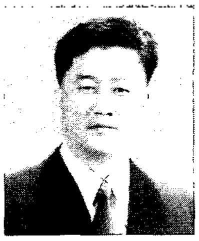
[34] Teng, J.T.C., V. Grover and K.D. Fiedler, "Business Process Reengineering: Charting a Strategic Path for the Information Age," *California Management Review*, Vol. 36, No. 3, 1994, pp. 9-31.

[35] Thuing, M., J.M. Haake and J. Hannemann, "What's ELIZA Doing in the Chinese Room Incoherent Hyperdocuments - and How to Avoid Them?" in *Proceedings of the 3th Annual ACM conference on Hypertext '91*, San Antonio, TX, December 15-18, 1991, pp. 161-177.

[36] Thuing, M., J. Hannemann and J.M. Haake, "Hypermedia and Cognition: Designing for Comprehension," Communications of the ACM, Vol. 38, No. 8, 1995, pp. 57-66.

[37] van Dijk, T.A., and W. Kintsch, *Strategies of Discourse Comprehension*, Academic Press, Orlando, 1993.

저 자 소 개



박진수	(E-mail : jinsoo@snu.ac.kr)
	The University of Arizona 경영정보시스템 (경영학박사)
	University of Minnesota(Carlson School of Management)조교수
	고려대학교 경영대학 조교수
현재	서울대학교 경영전문대학원/경영대학 조교수
관심분야	정보시스템모델링, 웹 정보시스템, 온톨로지, 정보시스템 통합, 지식공유, 에이전트