An Interface Specification method For E-Business Applications

Seung C. Lee
Department of Finance and MIS
Labovitz School of Business and Economics
University of Minnesota at Duluth
412 Library Drive
Duluth, MN 55812, USA
slee@d.umn.edu

Jinsoo Park
School of Business
Korea University
1, 5-ka, Anam-dong Sungbuk-ku
Seoul 136-701, South Korea
jinsoo.park@acm.org

Abstract

Specifying interface for e-business application is becoming complex because of its demanding functionality, a rapid advance in Web technology, and an increasing need of integration with legacy applications. All the recent developments call for a more methodical approach to e-business application interface. In this paper, we propose an interface specification method founded on the concepts of *meta-information structure* and *information structure*, as well as taking account of various page, link, and component types. They are rigorously utilized in the activities of the meta-information structure analysis and information structure analysis to arrive at a well-formed interface specification for e-business application.
Keywords: e-business application, meta-information, interface, systems development methodology

1. Introduction

Specifying the interface for Web applications are more complex than ever due to the three key developments in their design. First, these applications are now required to perform more and more functions than before because the Web as a business computing platform is gaining ground as companies race to transform their businesses into e-businesses [10]. A business function on the Web can be implemented by either writing code in a page or calling an existing software component. This implies that we should define what, where, and how functions should be performed. The answers to the question can be found by considering: (1) the information that must be delivered to the user regardless of the underlying functions; (2) the delivery process of the information that often involves many pages intertwined with each other; and (3) the enabling technology and available software components to implement a certain function.

Second, Web technology is rapidly advancing to meet the computing need that demands more interactive and dynamic delivery of information [2]. Although some Web pages are still static in nature, many of them are generated on the fly in response to a certain event. This means that a Web page can take shape of one or more of the following: browsable (visible to the user, e.g., a privacy page) or non-browsable (invisible but working behind the scenes to render the browsable, e.g., a page generating an order summary); base (a simple markup page, e.g., a pure HTML document or an application document, e.g., a spreadsheet) or derived (a document generated dynamically, e.g., an order summary); and interactive (containing an interactive element, e.g., a page with a form) or non-interactive. Third, Web-based applications often are integrated with existing non-Web applications [9, 10]. Companies should be able to leverage and extend critical existing business systems directly to customers, employees, suppliers, and distributors via the Web to improve time to market and reduce cost of development and deployment. This implies that we should take account of existing applications during the course of a Web application interface design.

According to Halasz and Schwartz (1994), a Web-based application consists of three layers: presentation, structure, and storage. The presentation layer deals with the user interface, which is essentially the reflection of the structure layer that defines the relationships among various building blocks such as pages, links, and software components. The user interface can be understood in terms of perceptive and cognitive aspects. The former is related to the use of colors, fonts, and other perceptive cues, while the latter is concerned with the flow of links, consistency in links, and other cognitive cues. Although an appropriate mix of the two can deliver higher usability of an application, the latter appears to be more important than the former in the case of e-business applications because the users are the general public rather than computer professionals. The cognitive aspect of the presentation layer first requires the identification of necessary links that are accessible by the user. This implies that some links are not accessible by the user. Then, the question is “Which links are accessible and which are not?” The answer to the question can be sought via an exploration of the structure layer because the presentation layer mirrors it. In other words, the identification of the cognitive aspect of the presentation layer inevitably involves a structural exploration of a Web-based application. This paper proposes an architectural design method for the cognitive aspect of the presentation layer by which we can define the structure layer and then identify necessary links and their organization.

Like other Web-based applications, an e-business application is a careful organization of meta-information and its elements, which often involves a variety of Web technologies to deliver unstructured as well as structured information to the user through an interface [6]. The interface of an e-business, therefore, involves both static and dynamic aspects of user interaction, which are basically enabled by means of hyperlinks that connect semantically related information chunks—generally called anchor pages—in a systematic way. The basic idea, however, behind virtually all interfaces specifically, and Web-based user interaction schemes in general, is simple: identify meta-information for all e-business domains (consisting of transaction units), work out in detail via an elaboration of the meta-information (possibly through such page types as visible and invisible), and then organize them into structures using hyperlinks (representing various semantics) in a way that makes it easier to manage the site later on. For example, suppose that an enterprise's e-business application for selling computers needs to include a user guide. The guide is a domain that might have several layers of meta-information. At the top layer are guides to the application, hardware, and operating systems. The next layer may include guides to individual applications, individual hardware for each platform, and various operation systems. These layers represent meta-information for the proposed domain. For each piece of the meta-information, we can then identify information elements in the form of pages. Some pages are static in the sense that they do not provide any interactive mechanism. Once a user clicks a link to a static page, the Web server searches only for the requested page and, if found, returns it to a user agent so that the requester can view the page. By contrast, some pages may contain dynamic elements such as a form. This requires the server system to process the request and to generate a certain page on the fly, occasionally with the help of software components. The whole scenario implies that we need to classify pages and hyperlinks into several categories to make the process of interface specification development systematic. It also requires that we take into account various software components including executable files, applets, Java class files, static libraries, and DLLs. Based on the building blocks, we can derive an orderly and methodical structure for the given collection of meta-information for each e-business domain.

Our method employs several new concepts under the notion that the basic unit of information delivery of an e-business application is a page, which may be the result of a simple interpretation of a pure markup document or a dynamically generated page through various underlying pages that embed functions and components. Regardless of the page delivery mechanism, the resulting page, if it needs to be accessible by the user, should be incorporated into a link page. Due to the vast amount of information a non-trivial e-business application delivers, it is not feasible to show all the anchor pages on a single link page. This implies that we need to break the body of information to be delivered to the user into a hierarchy of pages that contain various links to relevant anchor pages. Furthermore, as mentioned earlier, some pages are generated on the fly. This means that we also need to examine what page should be automatically generated in response to a certain event. The automatic generation of a page in turn makes it necessary to consider various underlying implementations that are used to create the page. This leads us to introduce *meta-information structure* and *information structure* along with various page, link, and component types.

There is some related work on the identification of meta-information, link types, and derivation of navigation structures (e.g., [3, 4, 12, 15]). They are based on several data models such as Entity Relationship and Unified Modeling Language and are indeed useful in developing Web-based applications. However, they lack the detailed process of developing an interface specification. The goal of the work presented in this paper is to ameliorate the above-mentioned limitation of interface specification by introducing the notions of meta-information structure and information structure. These notions incorporate two basic page types (visible and invisible pages), six link types (anchor, form, coordinate, trigger, redirect, and build), three software component types (client-side, server-side, and application components), and a structural view of the e-business application in which an e-business domain is stratified by its meta-information structure and information structure. The meta-information structure analysis is comprised of identifying three scopes: main scope, common scope, and aggregate scope. Those scopes are essential in representing the meta-level information of an e-business application. The information structure analysis is in essence an operationalization process of the meta-information identified in the meta-information structure analysis in which the comprising pieces of information are defined as a structure.

The reminder of the paper is organized as follows: a discussion of related work, followed by the description of the underpinning concepts used in this paper. The specification method for interface is presented in the subsequent section, and the last section presents conclusions.

2. Related Work

The explosive popularity of the Web has created a new interest in Web-based applications, including intranet, extranet, and various applications such as e-business and e-engineering. One of the problems, however, in developing an e-business application is an insufficiency of design methodologies [3, 4], not in terms of numbers but in terms of the richness that covers an important aspect of an e-

business design: the user interface. Much work has been published in the area of Web-based application development (e.g., [1, 3, 4, 5, 12, 15]. That work, in general, lacks a detailed procedure for the user interface specification.

Most of the design literature on Web-based applications has focused on the underlying data models, including the Dexter Model [13], ER [15], and UML [3, 4]. Although the data model is an important concern, user interface is an equally important matter that must be articulated and addressed. It is concerned with the visual presentation of content to users. Sometimes, the user interface has been the last issue to be considered. It is, however, at the presentation layer that the information in the underlying data model actually finds its way from abstract machine space into the user's perception [16]. In designing the user interface, the following should be considered: (1) global structure and controls such as spatial division and visualization of content; (2) local structure and controls such as text column and media stage, marking links, link tips, media stage and media browser, story header, and story footer; (3) the interface appearance of each navigational object seen by the user; and (4) other interface objects that activate navigation such as *go-to* buttons [14, 16, 18].

A Web-based application is not only a network of pages and links: it is a structured object [17]. If a Web application is ill-structured, the users will eventually suffer "cognitive overhead" [20]. Closely related to the structural problem is presentation of the structure to the users. A good structure does not guarantee a good presentation of the information contained in a Web-based application. Unlike conventional information systems, Web-based applications provide user interface objects by textual and graphical links with their contents. This means that, based on a good structure, there should be a mechanism to systematically reveal the structure to the users through systematic structure-revelation mechanisms. If not, the users will suffer "user disorientation" [20].

3. Underlying Concepts

In order to understand the interface specification described in the paper, it is first necessary to understand the underlying concepts. Accordingly, in this section, we provide a detailed description for them. The first part of this section explains page types which are fundamental units of Web applications, followed by details of software component types. The third part explains link semantics from which the six link types are derived.

3.1 Page Types

We believe that our approach is significantly different from, and more descriptive than, current interface design specification methods available in the literature. In particular, our approach divides application pages into two categories based on whether or not they are visible to the user. A visible page is a page that is browsable by the user, while an invisible page (such as a server pages) is non-browsable. The latter works behind the scenes to render a visible page. This classification plays an important role in the information structure analysis. As we will see shortly, the information structure analysis is performed solely based on visible pages.

A visible page can be either a *base* page or a *derived* page, both of which in turn are classified as either an *interactive* or a *non-interactive* page depending on whether the page contains an interactive element other than plain content. In other words, a visible page, either base or derived, may be comprised of any combination of its non-link content, one or more links to other pages, and one or more interactive elements. The non-link content could be very simple such as a message, a prompt, a confirmation, and/or a heading, which works like status cues in a conventional program. For example, one case might be a table of contents containing links to other pages and a simple heading. This means, of course, that the content of a visible page includes not only its description but also links and other elements.

Like visible pages, invisible ones are divided into three categories: *server page*, *client-side control page*, and *server-side control page*. A server page is a page that contains a server-side script that is processed by its engine before a storage system (i.e., Web server) returns its processing result to the client. A server page may contain logic in the form of a statement, a function, or a subroutine for a desired output. A component or an existing business application may be involved in the course of executing a server page. A client-side control page such as an external cascading style sheet (CSS) has several characteristics. First of all, it is hidden from the user of an e-business application (i.e., not directly browsable). Second, it controls the way the user agent displays a refined and consistent content. Third, it is executed or triggered by the client with no intervention other than the "response" of the server. Fourth, it may not require *explicit* user intervention such as clicking a link. Once a user requests a page that is associated in some way with a client-side control page, the target page automatically sends another request for the client-side control page to get a designated control. A server-side control page such as an *include* file has several characteristics that are similar to those of a client-side control page. First, it is also hidden from the user. Second, it controls the way the server performs a certain function and renders a desired output to the user. Third, it is triggered by the server, not by an intervention of the client other than the "request" of the user agent. Fourth, it does usually not require explicit user intervention such as clicking a link. Just like on the client side, once a user

requests a page that is associated with a server-side control page, it automatically includes or triggers the server-side control page for a designated control. Unlike visible pages, however, invisible pages are not divided into base or derived pages. Figure 1 portrays such a page classification.

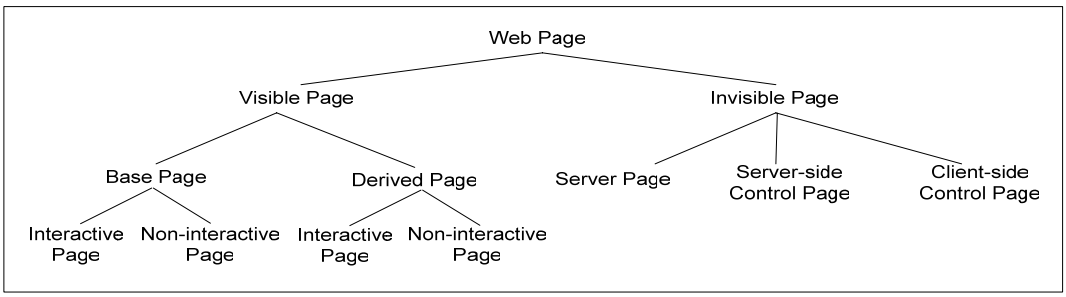

Figure 1. Page types

3.2 Component Types

A component here means a software component including executable files, applets, Java class files, static libraries, and DLLs. They are usually embedded in pages via a certain type of link. Core business processes residing in existing business applications can also be considered components which can be integrated into an e-business application through some published interfaces. Regardless of the form of a component, in the context of an e-business application, it plays an important role in delivering visible pages, supporting invisible pages, and/or performing necessary functions. In this paper, components are broken down into three categories: client-side component, server-side component, and application component. The client-side component could be divided into two types: host-dependent components, such as a client-side script function, and client system components, such as an autonomous media player. In a strict sense, an internal client-side script function—such as calculating lease payment, providing a calendar, or validating a form—may not be a component. But for a design purpose, we consider it a component, although it is not a separate entity from the "hosting" page. A client-side component can be triggered either implicitly (e.g., by a file extension) or explicitly (e.g., embedding a specific media player using a tag). In a similar sense, the server-side components are any components that provide a functional support to a server page (e.g., file uploading component or database access component). The application components are core business processes of existing business applications wrapped as components to expose interfaces to a Web application. For the sake of simplicity, we might treat whole core business processes as a component. This kind of component is especially useful when legacy or existing systems are integrated into an e-business application.

3.3 Link Semantics

A link is an associative connection defined between information elements. Links are established between information elements within a page as well as across pages. On the other hand, an anchor identifies the precise endpoint of a link. The endpoint can be a page or a bookmark within a page. A page that contains a link is called the link page, and a page for which a link is destined is called the anchor page. A link can appear to have multiple endpoints especially when a link is conditional. For example, a link can point to a different page depending on a conditional situation (e.g., rotating ad banners). From the user's point of view, however, a link with any number of endpoints appears to be one.

A link may have a meaning depending on the link reference. For example, if a link refers to a base page, it simply requests the Web server to retrieve and send the requested page back to the client. This is different from a link that triggers a certain component or that executes logic contained in a page. This implies that, based on the semantics of a link, we may classify the link into many types as does the World Wide Web Consortium (http://www.w3.org/DesignIssues/LinkTypes.html), which defines 15 link types. Additional link types are possible if we further elaborate the semantics (e.g., [4, 15]). Various links types can be useful in representing detailed semantics of the link and, hence, the precise context. They might, however, introduce complexity to the design of an e-business application because the designer should identify semantics of every occurrence of a link in depth and also keep track of all the link types employed. In any case, link types would be more meaningful if they were used to show global and local structures, and to enhance global and local coherence of a Web-based application, and, thus, to reduce the cognitive overhead of the user. The proposed method collapses possible link types into six context link types, including *anchor* (<a>), *coordinate* (<c>), *form* (<f>), *redirect* (<r>), *trigger* (<t>), and *build* (<b>).

The anchor link type is created by an anchor tag and used to represent a connection between elements of an e-business application. It may simply cause a server to retrieve a page or fire other link types. A bookmark within the same page is a different form of the anchor link type, while a bookmark to a different page can be considered simply an anchor link type. The coordinate link type connects a page to a client-side control page or a server-side control page to make the two pages work effectively as a whole for a well-controlled rendering. In other words, two or more pages act together to bring in content in a smooth, concerted way. The trigger link type is necessary to accommodate components. As Web-supportable functions expand, more and more components are intertwined with pages through links. Clicking on such a link passes data to and triggers a component to be executed (e.g., playing an audio clip). The trigger link type is used to represent such link semantics, which are implemented as either hyperlinks or instantiations. The build link type fits into a situation where a requested page generates another page on the fly responding to an event. In today's dynamic and interactive Web application environment, many e-business application pages are built in the runtime to take into account certain conditions. A good example is creating a billing statement or a personalized page based on personal preferences. Unlike the build link type, the redirect link type brings up another page without intervention by the user. Finally, the form link type is created by a form tag and represents cases where a link page contains a form with a submit button that is associated with an anchor page.

4. Interface Specification Method

In addition to the underlying concepts described above, we should introduce a couple of new concepts: meta-information structure and information structure. The meta-information structure is used to organize meta-information regarding an e-business application, while the information application), we would have a corresponding number of meta-information structures, but the number of information structures for each meta-information structure depends on the number of *leaf* elements of the aggregate as well as of main scope in some cases (the meaning of scope will be explained later). A meta-information structure represents a global interface structure, while an information structure defines the local interface structure of each leaf element of an aggregate. To show the entire process of interface specification development, we use an imaginary user-guide domain of an e-business application.

According to Simon [1962], hierarchical structure is a major facilitating factor enabling us to understand, to describe, and even to see complex objects and their parts. The notion of structure (usually a tree-like network) has been a part of most hypertext systems since the time of the Non-Linear Systems [7]. In line with the well-understood tree-like network, we also follow the common practice for interface specification. In meta-information structure analysis, we use the three access scopes: main scope, common scope, and aggregate scope. To understand the three access scopes, we begin the meta-information structure analysis with information decomposition.

4.1 Step 1: Meta-information Structure Analysis — Information Decomposition

There are three activities at this step: (1) breakdown of the target domain into top-level meta-information, (2) decomposition of the top-level meta-information into lower levels, and (3) stratification of the decomposed elements. To illustrate these activities, let us consider a simple e-business application design. Figure 2 portrays an information decomposition of the "User Guide" domain that is placed in the domain layer along with other domains. As we can see, it is not a full-blown decomposition. However, it should be enough to understand the process of information decomposition.

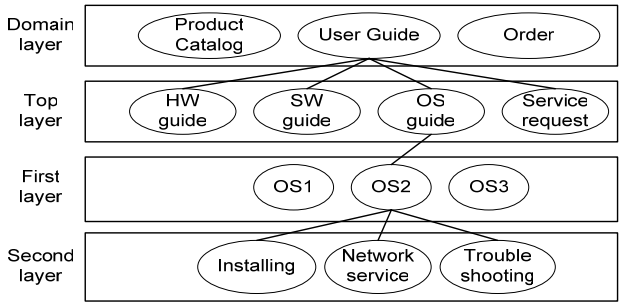

Figure 2. Information Decomposition

First of all, we identify the top-level meta-information of a given domain "User Guide" from the domain layer. When we decompose a target domain, we may think of two separate groups of meta-information: descriptive (factual/definitional) and prescriptive (process/procedural) meta-information. The former fits into the hyperdocument [8] that already exists as a knowledge product such as a written manual. By contrast, the latter requires the user to follow given steps for more refined results—and often some user input as well. For example, since the meta-element of "Service request" requires explicit user inputs rather than simple clicks, it can be considered prescriptive meta-information. For the second part of this step, we decompose all the top-layer meta-information further down into their lower-levels and arrange them into hierarchies. Note that some meta-elements may be dependent on a different meta-element, an event, or appear repeatedly. For example, the "Service request" may appear more than once because it is reusable meta-element. It may appear under the "hardware" user guide, "application" user guide, and

"operating system" user guide. The exact location should be determined by the implementation of the user interface.

We continue to refine the identified meta-elements for a given domain until further refinements are impractical. Once we stop refinement, the next step is to define access scopes followed by information structure analyses for each leaf meta-element of all the aggregate scopes (see the following subsection for details). The final shapes of hierarchies would vary depending on the two factors. The overall depth and breadth depend on the level of complexity and the size of the proposed e-business application, both of which could be measured by the depth and breadth of the top layer. For example, the number of clicks required to get to a particular meta-information could affect the depth of a hierarchy. In addition, as Fingar (2000) points out, the degree of changeability of content and structure also determines the breadth and depth. In fact, they 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 loosely coupled meta-information both at the top layer and the subsequent layers. We suggest that a developer may start with fine-grained hierarchies and then modify depending on the implementation strategy in terms of the breadth and the depth of a proposed e-business application. Note that hierarchies don't have to be balanced.

4.2 Step 2: Meta-information Structure Analysis — Defining Access Scopes

When the meta-information structure is implemented, each meta-element, in general, will be detailed by its overview (e.g., overall purpose of the User Guide domain), one or more links to its supporting pages (e.g., especially for the leaf meta-information elements of the aggregate scope), and some links to other meta-elements (e.g., as in Figure 2, links to the four meta-elements in the top layer from the User Guide in the domain layer). This means that the name of each meta-element actually represents an abstract for semantically related content and may become a link text—that is, each meta-element in the hierarchies would be operationalized as a link together with its overview, links to supporting pages, and links to other meta-elements. Then, how can we determine which meta-elements should appear on the default page, what should be included in every page, and what should have links to supporting pages? To answer these questions, we need to define three access scopes in the context of meta-elements, not of supporting pages and links. The three access scopes include *main scope*, *common scope*, and *aggregate scope*. They are special kinds of meta-information.

The main scope is necessary to accommodate the default page. It works as a gateway to other meta-elements. It may include the elements in all layers of the hierarchy if an e-business application is relatively small or some of the layers in the hierarchy if it deals with a large number of elements. The common scope defines a common set of meta-elements that would appear as hyperlinks on every page including the default page. It contains a subset of them of the main scope. For a small e-business application, the meta-elements included in the main scope might be the same as those in the common scope. The aggregate scope also defines a collection of meta-elements but in a different context. The aggregate scope determines the total number of links that can be clickable within a link page. It may not be practical to represent a higher-level meta-element as a very deep structure (i.e., deeply stratified structure for a meta-element). For example, the aggregate scope of "OS2" could contain only the meta-elements in the second layer or those in the entire lower-layers if we had more layers. This implies that a *big* meta-element may have many lower-level aggregate scopes in addition to a top-level aggregate scope. For instance, suppose that a user reached a page that contains a link to "OS2." Depending on the aggregate scope, the only choices he/she may have are "Installing," "Network service," and "Troubleshooting" or, in addition to those choices, he/she could have choices of, say, "Configuration problem," "Network problem," and "Miscellaneous problem" available under the "Troubleshooting" meta-element in Figure 2. Note that the number of aggregate scopes also depends on the main scope, common scope, and implementation strategy.

Figure 3 partially portrays an example of a main, a common, and an aggregate scope, respectively, and their relationships derived from Figure 2.

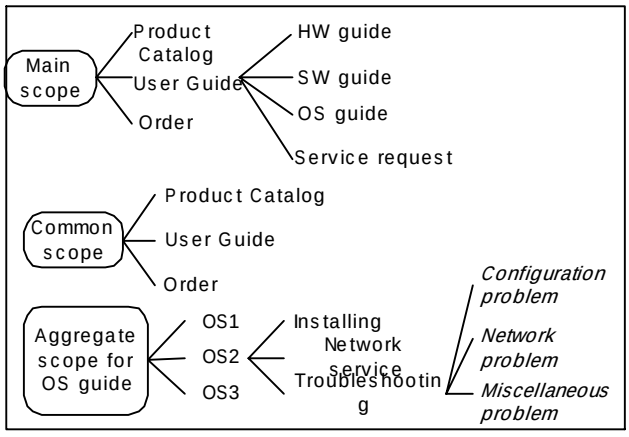

Figure 3. A Main, a Common, and an Aggregate Access Scope

Note that the "Service request" could be located in every scope or only in the main scope, depending on the implementation plan. The italicized meta-elements mean that they can be included in the "Aggregate scope for OS

help" or in a lower-level aggregate scope if elected. In the latter case, the lower-level scope would be called "Aggregate scope for troubleshooting." In fact, the number of levels in each scope has been arbitrarily determined for description purpose. The exact number of levels should be dependent upon the implementation strategy (e.g., depth and breadth). Although not shown, we might need more aggregate scopes for other leaf meta-elements in the main scope as well as in the aggregate scopes if necessary, depending on the meta-information structure. In most cases, the leaf meta-elements in a main scope would be expanded via aggregate scopes. It is, however, quite possible for a leaf meta-element in a main scope not to expand through an aggregate scope. In that case, we should take it into account when we perform the information structure analysis. For example, the leaf meta-element "Service request" in Figure 3 may not need any further breakdown, which means we may not need an aggregate scope for the element. The meta-elements in the main scope will be shown as hyperlinks on the default page, while those in the common scope will be shown on every page.

4.2 Step 3: information Structure Analysis

After deriving a meta-information structure with information decomposition and defining the three access scopes, our next activity moves on to the information structure analysis. The overall purpose of the meta-information structure analysis performed earlier is to identify information chunks that would appear as menu-like links on the default page and other pages. We are not concerned much about the supporting pages of the leaf meta-elements and links to those pages. In fact, any meta-elements should have links to others based on the hierarchical meta-information structure. Any leaf meta-elements in the aggregate scopes and, occasionally, in the main scope should also have links to supporting pages. In the information structure analysis, we take account of the pages types, link types, and component types that were discussed earlier. Before embarking on our information structure analysis, we provide a reference to the notation used throughout this section. Figure 4 contains this information.

When performing the information structure analysis, we consider only the visible page. We pay attention only to the visible page because when we develop an interface specification, we only take care of what users can actually view or browse. However, at this phase, we do not have to worry about the detailed visible page rendering process, how invisible pages are intertwined with each other to deliver visible pages, or how components are used by the invisible page to render the visible page or perform certain functions. We take two pieces of meta-information from Figure 3 to illustrate the overall process of the information structure analysis: "Network problem" and "Service request" as examples of definitional/factual meta-information and process/procedural meta-information, respectively. Figure 5 illustrates the output of the information structure analysis on the two meta-information elements.

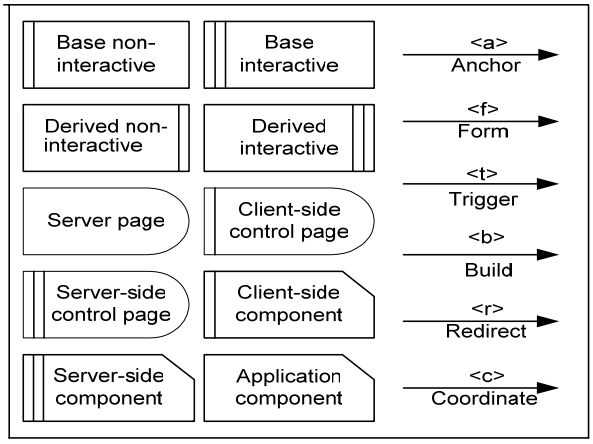

Figure 4. Notation for the Information Structure Analysis

Although we do not need layers other than "leaf meta-information" and "visible" layers, in Figure 5 we show all the layers with appropriate page types, link types, and component types to provide an overall context for the "leaf meta-information" elements and "visible" layers. The left side of Figure 5 depicts a factual/definitional meta-information element.

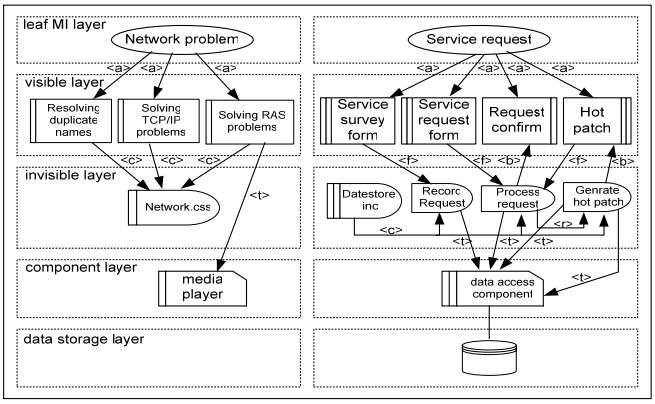

Figure 5. Outputs of the Information Structure Analysis on Two Meta-Information Elements

We assume it does not require a data storage layer but uses a style sheet file (i.e., network.css) and a client-side component (i.e., media player). They are all invisible pages but require links to "host" pages. The right side of Figure 5 is more complex than the left side because it shows the output of an information structure analysis on a process/procedural meta-information element (i.e., "Service request"). We assume that, when a user submits a service

request, the server page "Process request" is designed to generate a hot patch to reduce the workload of the User Guide Department. Thus, it redirects the request to the "Generate hot patch." The service requester will first attempt to fix his/her problem based on the hot patch solution. If he still cannot fix the problem with the hot patch, then he should resubmit the service request using the "Hot patch" page. Upon receiving the resubmission, the "Process request" page processes the request and generates a confirmation page. The details of all the service requests will be recorded in the data storage, which is accessed by the data access component. Before we mention the implementation and testing aspects of the interface specifications based on the meta-information structure and information structure analyses, we should answer the following question: What is the link type that connects meta-information elements with each other? It can be the "anchor" link type or the "build" link type. If an e-business application should generate all the pages, except for the default page, dynamically we are going to connect them using the "build" link type.

5. Implementation and Testing

The interface design specifications can be implemented in a straightforward manner by mapping the scopes and creating links to supporting pages identified in the information structure analysis phase. We admit that this interface specification method lacks the details of content of each page. The actual items and their layout are largely dependent on the developer and on the implementation strategy. Note that some scripting efforts should be made if pages have dynamic contents and are dynamically generated. The developer may create a prototype based on the specification and let the user evaluate and give feedback on the preliminary system as a testing procedure. Needless to say, testing a system before deployment is very important. In the case of an intranet, the target organization may employ the same user agent, and hence the testing might be easier. A prototype system is currently under development. The system will allow Web application designers to perform meta-information structure and information structure analyses as well as generate the site topology and application page templates, based on our proposed methodology.

6. Concluding Remarks

Currently, many advanced technologies (e.g., scripting, software components, general programming languages, and constantly evolving markup languages) are incorporated into the e-business application and other types of Web-based applications. To incorporate such rapid technological advances in the area of Web-based application development, we propose an interface design methodology for e-business applications. Developing a Web-based application such as an e-business application is simply beyond converting a document into a markup document. It involves more and more Web-related technologies and often requires integration with other technologies. It frequently implements fairly complex logic either through components or within pages or both. Moreover, due to an inherent characteristic of the Web, the size of an application can grow infinitely. The developers of any Web-based application seem to be pressured to deliver high-coherence and low cognitive-overhead applications along with "sustainable" contents. As a consequence, developing a Web application is becoming complex and time-consuming [10]. This paper is the result of a contemplative effort whose primary goal is to provide a sustainable method for the interface design of the e-business application.

This paper makes three key contributions. First, as the Web-based applications become complex in terms of structure and interface [11], it becomes important to employ an effective technique for reducing complexity both at a higher level and a lower level. Viewing an e-business application at a higher level using the meta-information structure concept provides a framework for the application, and refining the higher-level framework into lower-level details using an information structure facilitates completeness and consistency of the interface implementation. We can achieve a high-level architecture for the interface of a proposed e-business application through the main, common, and aggregate scopes that give insights into what an overall structure and a high-level presentation of the application should look like.

Second, refining the three scopes derived from the meta-information structure analysis using the information structure analysis enhances modifiability and maintainability because deliverables of the analysis enable us later on to connect the pages to each other based on semantic links. Maintaining separate scopes makes it easier to change their content as well as add a new information structure as an application evolves over time. The modularity obtained by applying the concepts of meta-information structure and information structure can insulate us from the "ripple effect." The consistent global and local views of an application defined by the three access scopes can also enhance global and local coherence [20], while a complete context and its semantic associations with other contexts can reduce the cognitive overhead.

Third, classifying Web application pages into the visible and the invisible and treating components as separate entities enables us to distinguish "what" from "how," which plays an important role in meta-information structure and information structure analyses. The users of an e-business application do not care about what is working behind the scenes, but they really do care about what they

actually see. The granulation of pages and components make it possible to accelerate the idea. It enables us to identify "what" by considering only the visible pages and then move to "how" through the incorporation of the invisible pages and various components.

Nevertheless, we might not be able to conclude without some caveats that eventually suggest future research directions. First, we intentionally omitted some details in the course of the meta-information structure and information structure analyses. Specifically, we missed incorporating external pages and/or meta-information elements into an e-business application. Nowadays, the majority of Web-based applications have links to external resources. Handling external links with proper security, for example, might not be an easy task. We, however, purposely glossed over this item under the assumption that there are few external resources for the e-business application. We also did not deal with the explicit embodiment of the underlying data model. We did not adopt a specific data model in the course of the method development. The method, however, is implicitly based on the procedural data model of data flow diagramming.

Second, we did not consider how to find and compose appropriate components and integrate them into the e-business application because we have focused on the interface specification. We believe that subsequent research on the development method should address the issues. Finally, this paper lacks detailed navigation design steps. It is a partial methodology, not a full-blown one, since our proposed methodology also lacks elaborated implementation and testing guidelines. In most cases, however, implementing the interface design specifications resulted from the application of this method would be a straightforward mapping process. If we had details on the implementation and testing specifications, the methodology would not be a partial one. Despite these factors, we believe that the interface specification method that has been developed by rigorously applying a number of new concepts should provide consistent and manageable interface specification design for the e-business application development.

References

[1] Bajaj, A. and R. Krishnan (1999) "CMU-WEB: A Conceptual Model for Designing Usable Web Applications," *Journal of Database Management*, 10(4), pp. 33-43.

[2] 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), pp. 153-178.

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

[4] Conallen, J. (1999) "Modeling Web Application Architectures with UML," *Communications of the ACM*, 42(10), pp. 63-70.

[5] De Troyer, O. (1998) "Designing Well-Structured Websites: Lessons to be Learned from Database Schema Methodology," *Proceedings of the 17th International Conference on Conceptual Modeling (ER '98)*, T. W. Ling, S. Ram, and M.-L. Lee (eds.), Singapore, November 16-19, pp. 51-64.

[6] Dias, C. (2001) "Corporate Portals: A Literature Review of a New Concept in Information Management," *International Journal of Information Management*, 21(4), pp. 269-287.

[7] Engelbart, D.C. (1968) "Authorship Provisions in Augment," *Proceedings of Fall Joint Computer Conference*, San Francisco, CA, December 1968, Vol. 33, pp. 395-410.

[8] Engelbart, D.C. (1995) "Toward Augmenting the Human Intellect and Boosting Our Collective IQ," *Communications of the ACM*, 38(8), pp. 30-33.

[9] Fingar, P. (2000) "Component-based Frameworks for E-Commerce," *Communications of the ACM*, 43(10), pp. 61-66.

[10] Flurry, G. and W. Vicknair (2001) "The IBM Application Framework for E- Business," *IBM Systems Journal*, 40(1), pp. 8-24.

[11] Fraternali, P. (1999) "Tools and Approaches for Developing Data-intensive Web Applications: A Survey," *ACM Computing Surveys*, 31(3), pp. 227-263.

[12] Garzotto, F., P. Paolini, and D. Schwabe (1993) "HDM - A Model-based Approach to Hypertext Application Design," *ACM Transactions on Information Systems*, 11(1), pp. 1-26.

[13] Halasz, F. and M. Schwartz (1994) "The Dexter Hypertext Reference Model," *Communications of the ACM*. 37(2), pp. 30-39.

[14] Hardman, L. and B. Sharrat (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*, Bristol, England: Intellect, pp. 252-259.

[15] Isakowitz, T. E. A. Stohr, and P. Balasubramanian (1995) "RMM: A Methodology for Structured Hypermedia Design," *Communications of the ACM*, 38(8), pp. 34-44.

[16] Kahn, P. (1995) "Visual Cues for Local and Global Coherence in the WWW," *Communications of the ACM*, 38(8), pp. 67-69.

[17] Nanard, J. and M. Nanard (1995) "Hypertext Design Environment and the Hypertext Design Process," *Communications of the ACM*, 38(8), pp. 49-56.

[18] Rossi, G., D. Schwabe, C. J. P. Lucena, and D. D. Cowan (1995) "An Object-Oriented Model for Designing the Human-Computer Interface of Hypermedia Applications," *Proceedings of the International Workshop on Hypermedia Design(IWHD '95)*, Montepellier, France, June 1-2, pp. 123-143.

[19] Simon, H. (1962) "The Architecture of Complexity," *Proceedings of the American Philosophical Society*. 106(6), pp. 467-482.

[20] Thuring, M., J. Hannemann, and J. M. Haake (1995) "Hypermedia and Cognition: Designing for Comprehension," *Communications of the ACM*, 38(8), pp. 57-66.