Journal of International Technology and Information Management

Volume 19 | Issue 1 Article 5

2010

Identifying Entity Types for E-R Diagramming in Developing Data- Intensive Web Applications

Seung C. Lee
University of Minnesota at Duluth

Identifying Entity Types for E-R Diagramming in Developing Data-Intensive Web Applications

Seung C. Lee
University of Minnesota at Duluth
U.S.A.

ABSTRACT

Although the Web has become a major enabler for data-intensive business applications such as e-tailing, customer relationship management, and supply chain management, we find that most Web applications are built up in ad hoc fashion, raising the data-related issues of data definition and data integration. This paper proposes a new approach for identifying entity types for E-R diagramming in developing data-rich Web applications. The methodology is founded on elements of Web applications including pages, links, Web application architecture, and business logic modules.

INTRODUCTION

The Web has become a major enabler for new, data-intensive business applications such as e-tailing, customer relationship management, and supply chain management. Furthermore, new software service delivery models, including software as a service (SaaS) and Web services, have brought even traditional data-rich enterprise applications, like human resource management, within the reach of the Web (Dubey & Wagel, 2007; Carraro & Chong, 2006). The emergence of the Web as a powerful platform for such data-heavy applications seems to warrant a systematic approach to their development, and especially to the conceptual data modeling process. However, we still find that most Web applications are built up in ad hoc fashion, raising the data-related issues of data definition, data integration, and query construction (Florescu, Levy, & Mendelzon, 1998; Genero, Poels, & Piattini, 2008). Worse, unlike traditional standalone applications, Web applications have data dispersed everywhere because of their unique features (Myers, Hollan, Cruz, Bryson, Bulterman, Catarci, Citrin, Glinert, Grudin, & Ioannidis, 1996). These features include static or dynamic interactions between client and server via the request and response mechanism; link-based navigation facility; differentiations between client (Web) pages, server (Web) pages, and other resources; dynamic generation of user interface objects and contents; different degrees of exposure to the user (e.g., extranet or intranet); and data flows between Web pages, not between processes in the conventional SDLC sense. In this paper, the term page or pages, unless otherwise specified, refers to a Web page or Web pages that can be requested by a user agent or Web browser.

The problems caused by an ad-hoc development approach and by the unique features of Web applications have led to a significant body of research on data modeling and methodologies for Web applications development (Abiteboul, 1997; Atzeni, Mecca, Merialdo, & DI e Automazione, 1998; Buneman, 1997; Conallen, 2000; Isakowitz, Stohr, & Balasubramanian, 1995; Post & Kagan, 2005; Young, 2005). Although these studies made significant contributions to the Web applications development process as a whole, few addressed data-related issues,

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

especially the very first step of conceptual data modeling. This modeling includes identifying entity types of a problem domain and drawing an entity-relationship (E-R) diagram—a graphical representation of the Entity Relationship model, which remains the core approach for conceptual data modeling. As is clearly known, one of the key success factors for any data-intensive Web application (such as many data-heavy traditional applications) is optimally defined data in terms of data structure, data elements, and their organization. In modern database management, the starting point for defining the effective and efficient management of data is to thoroughly identify entity types, including their attributes and relationships, as part of the conceptual data modeling which is one of the most important tasks in developing applications (Kalczynski, 2005; Misic & Russo, 1996). Conceptual data modeling provides an overall picture of an application's most valuable resource. It can be costly to incorrectly represent data requirements at the conceptual level, and such errors can invalidate the completed application.

As already mentioned, the step in conceptual data modeling is to identify entity types and to draw an E-R diagram. Developing an E-R diagram has traditionally taken one (or both) of two non-normative approaches (Teorey, Yang, & Fry, 1986). With a top-down approach, the starting point is to prepare high-level descriptions of the application, including its functions, scope, and environment. This approach usually generates a high-level E-R diagram with only a limited set of major entities, attributes, and relationships. With a bottom-up approach, the designer studies documents, screens, and other data sources and pursues detailed discussions with users about a proposed application. This technique is necessary for producing a detailed E-R diagram. These two approaches have their own strengths and weaknesses when applied to Web applications. A top-down perspective is appropriate for providing an integrative view of multiple sub-domains of a proposed Web application. A bottom-up perspective is appropriate for conceptual data modeling for "private" Web applications because users and internal documents are well known beforehand. Regardless of applicability, identifying entity types for E-R diagramming using these two approaches may expose the designer to their pitfalls: a function or process can be confused with entity type (Batini, Ceri, & Navathe, 1991), and an entity can be confused with a relationship (Chen, 1976).

This paper presents a methodology for identifying entity types for E-R diagramming in developing data-rich Web applications. The new methodology is radically different from the above two approaches. It incorporates the notion of a Web model where a proposed Web application can be broken into a hierarchy of tightly-cohesive and loosely-coupled business logic modules. Each business logic module, or each business logic piece of a proposed Web application, may be any mixture of business façades, workflow, and business rules. This implies that a business logic module is a superset of a function. The former may be a function or a unit of information to be delivered by a Web application. Delivering a unit of information (e.g., help on returning purchased items) may not involve any execution of code, which is always required by a function. The methodology consists of three major phases: business logic analysis phase, business logic design phase, and entity type identification phase. In the business logic analysis phase, business logic modules are arranged into a tree, organized from most abstract to the least abstract modules. In the business logic design phase, the three aspects of a business logic module are logically represented by presentation-layer Web pages (often called client pages), business-layer Web pages (frequently called server pages), other resources, and their logical associations (denoted by various semantic links). In addition to the presentation and business layers, another

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

layer is taken into account to logically identify one or more data stores that may interact with business layer pages. Altogether, the proposed methodology incorporates three-tier Web application architecture, a set of Web page types, resource types, and semantic link types.

This paper is organized into eight sections. Sections 2 and 3 lay the groundwork for the methodology by describing a Web model in terms of business logic, Web application architecture, and the unique features of Web applications. Section 4 suggests classification schemes for Web pages and links, followed by Sections 5 and 6 which present business logic analysis and design, respectively. Section 7 then explains how to use business logic diagrams to identify entity types for E-R diagramming for Web applications development. The last section presents some concluding remarks.

BACKGROUND

The Web is a large set of applications, called Web applications, running on the Internet. Implementation of a Web application in general employs a multi-tier Web application architecture. Application architecture is usually broken into logical chunks called "tiers", where every tier is assigned a role (Petersen, 2002). Traditional applications consist only of one tier, which resides on the client machine; but Web applications lend themselves to an n-tiered approach by nature. Web application architecture is important because it visualizes interactions among Web application elements and introduces a logical separation of client- and server-side pages (or simply client and server pages), as well as determines the actual level of application performance, resource utilization, and maintainability. This methodology assumes the common three-tier architecture, consisting of presentation layer, business layer, and data layer. The presentation or user-services layer includes things specific to the user interface. This layer often does all its work through interactions with the business layer. As stated previously, the proposed method is built around business logic and its components. (The terms "business logic" and "business logic module" will be used interchangeably throughout this paper). Each piece of business logic comprises business façades, workflow, and business rules. Business facades are interfaces that expose business services to the user in the presentation layer, while hiding how workflow and business rules have been implemented. Workflow is a sequence of steps that involve state transformations in terms of name-value pairs. Business rules control the implementation of autonomous transactions, such as constraints on acceptable data values, and conditions where data may be accessed. The business, or business-services, layer implements workflow and business rules. Finally, the data layer houses data and data store software, such as relational database management systems.

A Web application consists of a set of business logic modules, each of which is composed of interlinked pages and resources accessed via the client-server mechanism. The server stores the pages and resources of a Web application, serves as a nucleus where incoming instructions from one or more clients are processed, and sends the processing results back to the sources of the instructions. A client is, in general, a Web browser, and the origin of instructions for the server. The client displays business facades implemented by presentation-layer Web pages. They may be linked to other business façade pages or business-layer pages, which implement workflow and/or business rules of business logic. A page that has a link to another page is called a link page. A given link's endpoint is called an anchor page. Link and anchor pages may be

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

augmented by other resources such as components and stylesheets. Thus, a business logic module can be described by a set of client- and server-side link pages, client- and server-side anchor pages, client- and server-side resources, and links.

IDEMPOTENCY AND WEB APPLICATIONS

Web applications have some unique characteristics compared to traditional counterparts. They include ill-defined users in the case of public Web applications, mode of information delivery, statelessness of pages, and multi-faceted interface functionality (Lee, 2008; Molly, 2001). Unlike traditional applications, Web applications can deliver information to a user in two different modes: static mode and dynamic mode. Static delivery mode involves a simple rendition by a browser of the content that has already been marked up at design time in a markup document such as an HTML page (e.g., census data marked up in an HTML document). The term "static" is used in the sense that the process results of a page is idempotent by both interpretation and execution—that is, duplicate requests to a static page received by an application have the same effect as a single request. In an idempotent state by interpretation, each request involves an interpretation of the markup elements for the same content. In an idempotent state by execution, the same content always results through the execution of program code. For example, a Web page may contain executable code, not just markups, to deliver the same login form on each request. Thus, the input to a request for a static page is a simple descriptive message (e.g., I want this page), not a prescriptive message such as parameters or user input for functions. The latter is the case for the dynamic delivery mode. Dynamic delivery mode involves an execution of program code to generate certain output on the fly by processing user data. This mode could be idempotent by input, as well as by execution, but not persistently. Such dual delivery modes of information are possible because of the unique coding scheme of Web programming. Traditional applications, either standalone or distributed, are written in programming languages that are used to convert functional requirements into executable code. By contrast, Web applications employ two different language types: markup languages such as HTML and XML, and ordinary programming languages. This unique feature is the direct result of the original purpose of HTML, which is information exchange over a network in a platform-independent manner (Berners-Lee, 2000). However, the programming aspect of Web applications is the result of the extended functionality of the Web. The dual-language coding for Web applications carries an important implication in terms of Web page types and of interactions and associations among them.

The statelessness of Web pages by HTTP makes a Web application more scalable because it does not have to store state information between page requests. However, real world Web applications rely mostly on stateful services, which require special mechanisms like query strings. In addition to this sort of stateful link between pages, many other types of links can be defined, such as the ordinary hyperlinks you can create using the HTML anchor tag (i.e., <a>). Others would include a link connecting a page containing a user form (e.g., a registration form) to a page processing the form data upon being submitted, a link calling a component, a link associating a page with a style sheet, and even a link redirecting a user to a different page. One thing, however, is clear. Not all the link types listed above would carry meaningful or observable data in the context of application data. Some of the links simply carry a request string or a flag. This is where we encounter difficulties in applying well-established process modeling techniques such as use case modeling and data flow diagramming to Web applications development. The

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

former is based on the notion of binding data and its processing together, and its diagramming tools employ sequences of function calls and data flows. Use case modeling also assumes that users and actors of applications are well defined around functional requirements. On the other hand, in data flow diagramming every process must have observable data inflows and outflows, which is not always true for Web applications. Above all, both modeling techniques do not explicitly take into account the unique characteristics of Web applications and the various page linkages that have different semantics. Therefore, it appears to be a natural development that the previous studies were centered on specifying navigational arrangements and functional requirements (e.g., Conallen, 2000; Isakowitz et al., 1995).

Multi-faceted interface functionality is about offering business logic and its elements (i.e., façade, workflow, and business rules) in a cohesive and loosely-coupled way for better user comprehension. Thus, the interface functionality pertains to presenting effective navigational structure (Hardman & Sharrat, 1990; Kahn, 1995; Rivlin et al., 1994; Treiblmaier, Pollach, Floh, & Kotlowski 2004), and rendering interfaces to the user (Thüring, Hannemann, & Haake, 1996). In cognitive science, a user's level of comprehension is a function of the mental model the user would construct, based on visible objects and their semantic relations, in which coherence has a positive influence while cognitive overhead has a negative influence on comprehension (Conklin, 1987). This suggests some useful insights into Web application development: to provide users with an effective navigation, a Web application should be characterized by higher coherence through careful accommodation of various link semantics, effective structure with appropriate breadth and depth, and efficient static and dynamic interface rendition. To realize the insight, a Web application should replicate the real building blocks of an application domain—business logic modules—in a cohesive and loosely-coupled way for better user comprehension.

DESIGN PRIMITIVES

We have discussed the unique characteristics of Web applications and also introduced three-tier Web application architecture. In this section we explain design primitives: page and link types.

Page types

Before we describe each of the page types used to construct this methodology, we need to understand how a Web page is rendered. To open a Web page in a Web browser a user types in a URL to the page or clicks a link to the page. Right after the key stroke, the browser prepares a request for the page designated by the URL with the help of HTTP. The request arrives at the Web server identified by the URL (i.e., a destination IP address specified in the HTTP request message header). As soon as the web server receives a request, it begins searching the page requested. If it is a static page (e.g., HTML document), the server copies the page and puts it into an HTTP response message. The message travels back to the browser that has requested the page. Then the browser displays the contents of the page after interpreting the HTML tags. This kind of page rendition procedure involves request/response (R/R) steps. Another page rendition procedure needs one more step in addition to the R/R steps. When a requested page contains executable code, then the web server executes the code and builds an HTTP response message with the execution results. Thus, this more sophisticated rendition procedure requires request/execution/response (R/E/R) steps.

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

Some Web pages rendering business facades may be idempotent, either by interpretation (i.e., facades are delivered in a static delivery mode by interpretations of markups specified in Web pages at design time), or by execution (i.e., facades are rendered in a static delivery mode by execution of some program code contained in Web pages). Other Web pages may not (i.e., facades are delivered in a dynamic delivery mode, but corresponding Web pages are not idempotent by interpretation; they could be idempotent by input and by execution, but not persistently). Regardless of being idempotent or not, business facades are results of HTML documents interpreted by a browser, which could be augmented by other resources, including style sheets, client scripts, client components (e.g., plug-ins) and various media types defined by the RFCs for MIME. Furthermore, a markup document may or may not carry a user form for catching data from users. If a rendered page contains a form, it is said to be an interactive page. Otherwise, it is called a non-interactive page. In summary, page augmentation schemes, two distinct page rendition procedures (R/R and R/E/R), and two information delivery modes (static delivery mode and dynamic delivery mode) lead to a classification of Web pages and resources (Table 1).

Table 1: Page and resource types.

Architectural Layer	**Page Type**	**Description**
Presentation layer	R/R non-interactive page (RRNIP)	A pure HTML page that *has no* form (e.g., a product description page). This is often called a client page or base page and is for a business façade. Idempotent by interpretation.
	R/R interactive page (RRIP)	A pure HTML page that *has* a form (e.g., a registration page). This is also often called a client page or base page and is for a business façade. Idempotent by interpretation.
	R/E/R non-interactive page (RERNIP)	A page resulted from an execution of a server page containing only markups. This page type *has no* rendered form. This is often called a derived page and is for a business façade. This type of page is an example of a server page that is physically one (i.e., a single server page) but logically two different pages (a page for the presentation layer and another for the business layer). Could be idempotent by input and by execution, but not persistently.
	R/E/R interactive page (RERIP)	A page resulted from an execution of a server page containing only markups. This page type *has* a form. This is also often called a derived page and is for a business façade. This type of page is also an example of a server page that is physically one but logically two different pages. Could be idempotent by input and by execution not persistently.
	CSR	Client-side resources that augment web pages on the client side, including style sheets, client scripts, and

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

		various media types defined by the RFCs for MIME.
	CSC	Components that work on the client side of websites, including plug-ins, add-ons, and applets.
Business layer	R/R server page (RRSP)	This is a type of server page needed to process form data submitted via R/R interactive pages (RRIPs). This type of page handles workflow and business rules of a business logic module.
	R/E/R server page (RERSP)	This is the other type of server page required to handle interactions with R/E/R non-interactive pages (RERNIPs) and R/E/R interactive pages (RERIPs). This type of page also handles workflow and business rules of a business logic module.
	SSR	Server-side resources that augment web pages on the server-side, including global routines.
	SSC	Components that work on the server side of websites, including custom components and Web services

Link types

A link is an associative connection between pages, which can be understood via descriptive and prescriptive *semantics*. A standard hyperlink, which is created by using the HTML anchor tag, that simply links a page to another, plays a descriptive role. If a link, upon clicked, conveys data between pages, it plays a prescriptive role because the data also include an instruction (e.g., process form data being submitted or read as a cookie file). The instruction can be implicit or explicit. When a server page programmatically generates a presentation-layer page (either RERNIP or RERIP) on the fly (e.g., generating a billing summary at the end of an online order process), the semantic relationship between the server page and the presentation-layer page is called a "build" link. Assume that a page passes a piece of information to another page to maintain a state. State management is the process of keeping, in terms of name/value pairs, state and page data over multiple requests for the same or different web pages (Fraternali, 1999). This sort of association is dubbed a "state" link. Imagine that a user submit a registration form on a RERIP to a RERSP. This kind of relationship is named a "form" link. In some situations a page redirects a user to a different page, or a page of a business logic module delegates a task to a page of another business logic module. This type of association could be called "redirect" link. There could be more link types. For example, a page is often augmented by a client-side or a server-side resource page like a style sheet. This sort of connection could be named a "directive" link because it directs the resource page to be processed before the other page is processed (see W3C website for link-related issues). Table 2 summarizes the link types.

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

Table 2: Link types.

Symbol	Meaning	Description
<h>	Hyperlink	The standard hyperlink created by <a> HTML element.
<b>	Build link	A semantic link that associates a business-layer page with a presentation layer page, where the former builds the latter on the fly.
<i>	Invoke link	A semantic link that associates a presentation- or business-layer page with either a CSC or SSC, where the former invokes the latter.
<s>	State link	A semantic link that associates a page to another, where the former passes state information to the latter.
<d>	Directive link	A link that associates a page to a CSR or SSR. This link can be created by <link>, <script>, or a processing instruction.
<r>	Redirect link	A semantic link that shows a task delegation from a page to another or that literally redirects a user to another page.
<f>	Form link	A semantic link that associates an interactive page to a server page that processes the form data submitted.

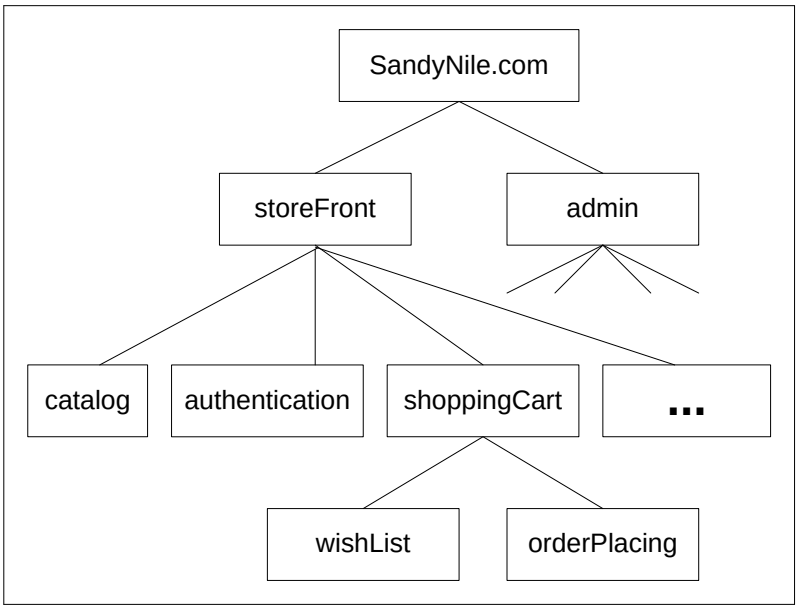
BUSINESS LOGIC ANALYSIS

As mentioned earlier, a Web application can be seen as a collection of loosely-coupled and tightly-cohesive business logic modules (BLM). A business logic analysis phase is for the process of identifying BLMs for a target Web application and then organizing them into a hierarchy, from the most abstract to the least abstract business logic modules. Identification of BLMs should be done in the manner that manifests higher coherence and loose coupling among BLMs while considering their hierarchical relationships. Hierarchy has been a major facilitating factor enabling us to understand and describe complex objects and their parts (Simon, 1962). Arranging BLMs as nodes of a tree depends on the scope of a target Web application. It may be divided into multiple sub-domains. For instance, we could think of "storefront" and "administration" sub-domains for a typical online store. During the requirements gathering, we first focus our effort in identifying highly abstract, loosely-coupled, and highly-cohesive BLMs for each sub-domain. For example, we can think of "catalog" and "shoppingCart" BLMs for the storefront sub-domain. The next step, as in the established process modeling techniques, is to decompose the abstract BLMs until we reach reasonable, non-trivial, but smaller BLMs. To secure succinct and logical relationships between BLMs, we should organize them into a hierarchy, where each BLM would become a node of the tree. In this methodology, nodes represent BLMs and arcs represent logical relationships between BLMs. Child nodes should be finer and more cohesive than parent nodes in terms of the number of functions and logical relationships between them. For example, for an online store, a parent BLM labeled "shoppingCart" could be decomposed into two child BLMs, which could be named "wishList" and "orderPlacing." The former would include functions like storing item information a customer wishes to buy, notifying the wish list upon customer login, converting the list into orders, etc. The functions seem to be all logically cohesive, and further decomposition may be

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

unnecessary. Figure 1 shows a partial BLM tree example for a hypothetical online store named SandyNile.com.

Figure 1: A partial BLM tree.

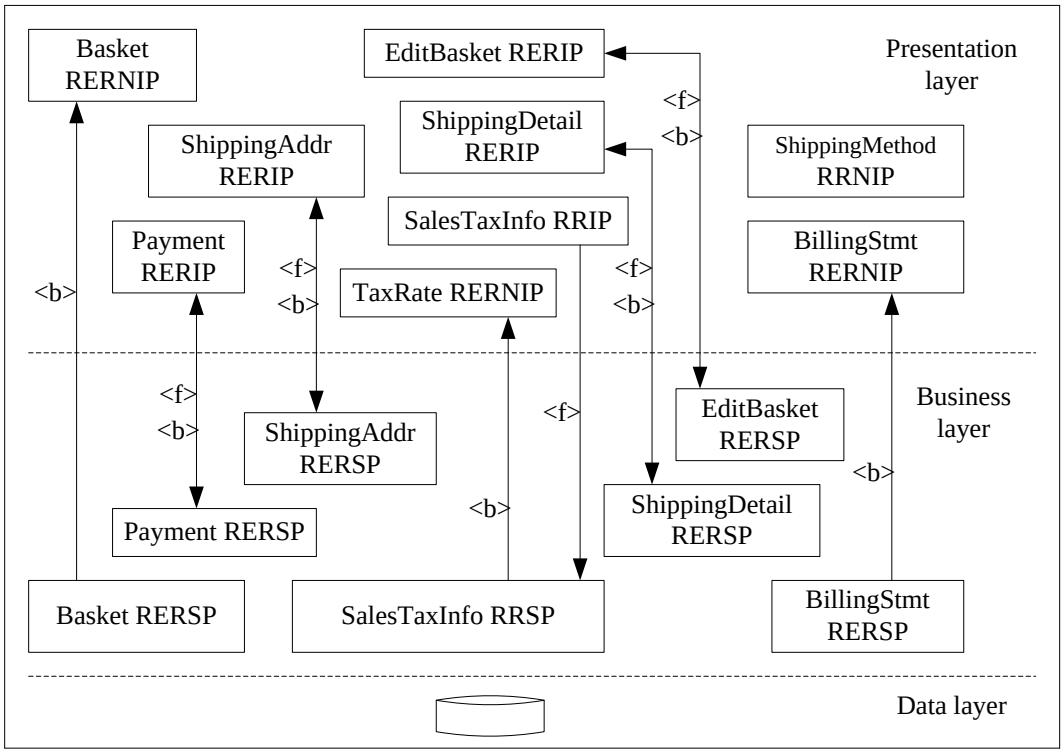


BUSINESS LOGIC DESIGN

Once we get a hierarchy of BLMs, the next step is to elaborate the identified BLDs by drawing a business logic diagram (BLD) for each *leaf* node in the hierarchy and by incorporating the architecture, page and resource types, and link types. Although all of the seven link types can be used for drawing BLDs, to avoid cluttering, the figure does not show associations among pages *within* the presentation and business layers. When we identify entity types from BLDs, we are mainly interested in the build link (<b>) and form link (<f>). The former, in general, carries data structures or data elements between pages (e.g., generating a billing summary for an order), and the latter always transfers data structures or data elements between pages (e.g., processing registration data). Data can also flow over the state link (<s>) (e.g., a query string attached to a URL may carry one or more name/value pairs) and invoke a link (<i>) (e.g., a page calling a directory access component gets directory data from the component). For the sake of simplicity, but without losing generality, we shall illustrate how to draw a BLD and how to identify entity types from the BLD. Figure 2 shows an example BLD for "orderPlacing" of the online store.

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

Figure 2: An example business logic diagram.



The presentation-layer page Basket RERNIP is displayed with an item a customer just added, plus a list of suggested books. The page is built dynamically, based on the added item by the business-layer page Basket RERSP, but does not contain any form. This is why the link between the two pages is labeled a "build" link, and the page type is non-interactive. Once the customer clicks on a "Proceed to checkout" button, he will be asked to log in or register, depending on the nature of the customer. The authentication process should be shown by a separate BLD for an authentication business logic module, which is not shown here. Once the customer is authenticated, another presentation-layer page, ShippingAddr RERIP, is presented. He can add a new shipping address or edit an existing address(es). To add a new address, the page should have a form; to edit an address, the page should display the existing address. Adding a new address requires a "form" link to the business-layer page ShippingAddr RERSP, editing an existing address requires a "build" link from the same page. Both actions require interactions with the server page. That is why the presentation-layer page is marked interactive. The same logic applies to the EditBasket, ShippingDetail, Payment, SalesTaxInfo pages in the two layers, and similarly to the BillingStmt pages across the two layers. Note that the SalesTaxInfo is an example of a pure HTML page with a form whose data are processed by the SalesTaxInfo RRSP. This server page also builds the TaxRate RERNIP page, showing a tax rate and related information based on the form data submitted through the SalesTaxInfo RRIP. Although we have identified the ShippingMethod RRNIP page, it is not our concern because there are no data flows into or out of the page. This type of page is in general written in HTML only, and probably contains client-side scripts and style rules such as JavaScript scripts and Cascading Style Sheet. The most appropriate link type, from and to this type page, would be standard hyperlinks, which

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

do not carry any substantial data. Note that, as we can see in Figure 2, any possible links between presentation-layer pages and between business-layer pages are not of interest. Rather, our agenda should be navigation design. Furthermore, although not shown, we should assume there are connections between business-layer pages and a database shown in the data layer. How then do we identify entity types from a BLD?

IDENTIFICATION OF ENTITY TYPES

As implied by Figure 2, links between pages may carry one or more data abstractions (structures), or one or more data elements. For example, a presentation-layer page for resume posting would convey several data abstractions like "personal information," "education," and "experience" over a form link. A presentation-layer page for user login would transfer "username" and "password" data elements, which probably are part of a data abstraction. Similarly, when a customer queries sales tax rates of a state, a business-layer page (e.g, SalesTaxInfo RRSP) would build a presentation-layer page (e.g., SalesTaxInfo RRIP) based on a sales tax rate data abstraction. It would carry data elements such as state name and tax rate over a "build" link. As such, we can easily identify such data abstractions and data elements, carried by links, from Figure 2. The data abstractions we can identify from the figure would include "book," "payment," "shipping_address," "sales_tax_rate," "order," "shipper," and "customer." After identifying all the data abstractions from all of the BLDs, duplicate data abstractions should be removed to define a set of unique data abstractions for a web application. Note that it is possible to have syntactically different but semantically identical data abstractions. The names of data abstractions then become entity types. If a link carries one or more data elements, we can infer the data abstractions to which they should belong. In this manner, we can thoroughly identify and clearly define entity types for conceptual data modeling in developing data-rich web applications.

CONCLUSION

E-R modeling, as the mainstream approach for conceptual data modeling, is perhaps the single most important facet of database development, and according to Misic and Russo (1996), is one of the most critical tasks that determine the quality of an application. Failure to capture and represent the data requirements of a Web application at the conceptual level could invalidate the completed application. To warrant a successful Web application development, we should secure correct and complete data requirements. At the heart of such an important task is the requirement to identify entity types completely before we begin drawing E-R diagrams. This paper has introduced a novel, yet clean, method for entity type identification, for conceptual data modeling. It has built upon the very elements of Web applications: pages, links, and business logic, and we therefore argue that the method is robust and easy to follow. One final thing to note is that the method, although it is not explicitly described, could readily be extended to cover the attribute identification of, and relationships between, entity types.

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

REFERENCES

Abiteboul, S. (1997). Querying semi-structured data. *Proceedings of the International Conference on Database Theory (ICDT)*, Delphi, Greece.

Atzeni, P., Mecca, G., Merialdo, P., & DI e Automazione. (1998). Design and maintenance of data-intensive web sites. *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, 436-450, Valencia, Spain.

Batini, C., Ceri, S., & Navathe, S. B. (1991). Conceptual database design: an Entity-relationship approach. Redwood City:Benjamin-Cummings.

Berners-Lee, T. (2000). Weaving the web: the original design and ultimate destiny of the world wide web. Collins.

Buneman, P. (1997). Semistructured data. *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 117-121, Tucson, Arizona.

Carraro G., & Chong, F. (2006). Software as a service (saas): an enterprise perspective. Retrieved March 19, 2009, from http://msdn2.microsoft.com/en-us/architecture/aa905332.aspx.

Chen P. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9-36.

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

Conklin, J. (1987). Hypertext: an introduction and survey. *IEEE Computer*, 20(9), 17-40.

Dubey, A., & Wagel, D. (2007). Delivering Software as a Service. *The McKinsey Quarterly*, www.mckinseyquarterly.com/article_page.aspx?ar=2006&l2=4&l3=43&srid=17&gp=0.

Florescu, D., Levy, A., & Mendelzon, A. (1998). Database techniques for the World-Wide Web: a survey. *ACM SIGMOD Record*, 27(3), 59-74.

Fraternali, P. (1999). Tools and approaches for developing data-intensive Web applications: A survey. *ACM Computing Surveys*, 31(3), 227-263.

Genero, M., Poels, G., & Piattini, M. (2008). Defining and validating metrics for assessing the understandability of entity–relationship diagrams. *Data & Knowledge Engineering*, 64(3), 534-557.

Hardman, L., & Sharrat, B. (1990). User-centered hypertext design: the applications of HCI design principles and guidelines. In R. Mcaleese and C. Green (Eds.), *Hypertext State of the Art*, Intellect, 252-259.

  ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

Isakowitz, T., Stohr, E. A., & Balasubramanian, P. (1995). RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 38(8), 34-44.

Kahn, P. (1995). Visual cues for local and global coherence in the WWW. *Communications of the ACM*, 38(8), 67-69.

Kalczynski, P. J. (2005). Time dimension for business news in the knowledge warehouse. *Journal of International Technology and Information Management*, 14(3), 21-32.

Lee, S. (2008). Collocation and collation of business logic for web application development. *Journal of Computer Information Systems*, 49(1), 57-66.

Misic, M., & Russo, N. (1996). Educating systems analysts: a comparison of educators' and practitioner' options concerning the relative importance of systems analyst tasks and skills. *Journal of Computer Information Systems*, 36(4), 86-90.

Molly, H. C. (2001). Designing User-Centered Web Applications in Web Time. *IEEE Software*, 18(1), 62-69.

Myers, B., Hollan, J., Cruz, I., Bryson, S., Bulterman, D., Catarci, T. Citrin, W., Glinert, E., Grudin, J., & Ioannidis Y. (1996). Strategic directions in human-computer interaction. *ACM Computing Surveys*, 28(4), 794-809.

Petersen, J. (2002). Benefits of using the n-tiered approach for web applications. *ColdFusion Developer's Journal*. Retrieved March 19, from http://www.adobe.com/devnet/coldfusion/articles/ntier.html

Post, G. V., & Kagan, A. (2005). Systems development tools and the relationship to project design: cost and budget implications. *Journal of International Technology and Information Management*, 14(1), 1-14.

Rivlin, E., Botafogo, R., & Schneiderman, B. (1994). Navigating in hyperspace: designing a structure-based toolbox. *Communications of the ACM*, 37(2), 87-96.

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

Teorey, T. J., Yang, D., & Fry, J. P. (1986). A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2), 197-222.

Thüring, M., Hannemann, J., & Haake, J. M. (1995). Hypermedia and cognition: designing for comprehension. *Communications of the ACM*, 38(8), 57-66.

  ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy

Treiblmaier, H., Pollach, I., Floh, A., & Kotlowski, M. (2004). A conceptual framework for e-branding strategies in the non-profit sector. *Journal of International Technology and Information Management*, 13(3), 143-156.

World Wide Web Consortium (W3C). Link Types. Retrieved Mar 19, from http://www.w3.org/DesignIssues/LinkTypes.html.

Young, D. (2005). Best practices and web practices: comparing corporate supplier diversity programs with web-based minority supplier content. *Journal of International Technology and Information Management*, 14(1), 41-52.

ISSN: 1543-5962-Printed Copy ISSN: 1941-6679-On-line Copy