A NOVEL APPROACH FOR IDENTIFYING ENTITY TYPES FOR CONCEPTUAL DATA MODELING IN DEVELOPING DATA-INTENSIVE WEB APPLICATIONS

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

ABSTRACT

Creating effective and efficient databases requires that we thoroughly identify and clearly define entity types, attributes, and relationships. That is why conceptual data modeling is one of the most important tasks in developing applications. This paper introduces a new method for identifying entity types for entity-relationship conceptual data modeling in developing data-intensive web applications. The method is founded on the very elements of web applications including pages, links, web application architecture, and business logic modules.

Keywords: Entity types, conceptual data modeling, web applications, web page types, web link types

INTRODUCTION

We now all would agree that the Web has become a major platform for complex and demanding enterprise applications in such data-intensive domains as e-commerce, customer relationship management, and supply chain management. Even traditional data-rich enterprise applications like human resource management are now within the reach of the Web via new software service delivery models of, for example, software as a service (SaaS) and Web services [5, 7]. However, many would agree that a vast majority of web applications are being developed off the top of a developer's head in an ad-hoc fashion, contributing to the problems, among others, of data— defining data, integrating data, and constructing queries [8, 10]. Worse, web applications are characterized by a unique mix of features—interaction between client and server involving request and response, associations via links between web pages, relationships between client pages and server pages, dynamic generation of web pages for presentation, internal structure and contents of web pages, interactivity of web pages, nature of web applications (which is often public not private ones like Intranets), data flows between web pages, not between processes, and finally, contents of web applications in finer granularities.

These features make web applications radically different from traditional applications of information technology [15]. The salient difference between the two kinds of applications is that data are dispersed everywhere in the case of web applications. These unique features have called for new ways of managing and querying data and hence led to a significant body of research on data models and methodologies for web applications in an effort of addressing the data-related problems as well as other problems such as user disorientation [e.g., 1, 2, 4, 6, 9, 12]. However, the data models and methodologies for web applications hardly mentioned that the differences warrant a new technique for identifying entity types for a web application's entity-relationship (E-R) diagram, a graphical representation of an E-R model that remains the core approach for conceptual data modeling.

As it is well known, creating effective and efficient databases requires that we thoroughly identify and clearly define entity types, attributes, and relationships. That is why conceptual data modeling is one of the most important tasks in developing applications [14]. It provides an overall picture of an application's most valuable resource. It can be too costly to fail to capture and represent the data requirements at the conceptual level because the failure could invalidate the completed application. The first thing to do in conceptual data modeling is to identify entity types to draw an E-R diagram. Traditionally, developing an E-R diagram has taken one (or both) of the two non-normative approaches [17]. With a top-down approach, the starting point is to prepare or get high-level descriptions of the application, including its functions, scope, and environment. This perspective usually ends up generating 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 along with detailed discussions with users about proposed web applications. This technique is necessary for producing a detailed E-R diagram. These two approaches have their own strengths and weaknesses. A top-down perspective appears to be appropriate for providing an integrative view of multiple sub-domains of a proposed web application. A bottom-up perspective seems rather to be appropriate for conceptual data modeling for "private" web applications because their users and

internal documents to study are well known beforehand. However, the designer is still exposed to one of the pitfalls of the two approaches in which functions or processes can be confused with entity types [3].

This paper presents an entity type identification method for conceptual data modeling in developing data-rich web applications. The method is new and radically different from the above two approaches but has the most of the advantages of them. The method adopts the notion of a web model within which a web application can be broken hierarchically into tightly-cohesive and loosely-coupled business logic modules. Each business logic module is represented by a set of business façades, workflow, and business rules. Business facades are user interfaces represented by presentation-layer web pages and workflow and business rules are operationalized by business-layer web pages (often called server pages). The business layer pages may or may not access a data store logically residing in data layer. The web pages in two different layers may interact with each other via various link types. The interactions may carry data depending on the link types. Therefore, the proposed method also incorporates three-tier web application architecture, web page types, and link types.

This paper is organized into eight parts. Section 2 describes a web model in terms of business logic and web application architecture. Sections 3 and 4 classify and explain web page types and link types, respectively, which is followed by two other sections for business logic identification, organization, and drawing business logic diagrams incorporating the pages types and link types. The section followed explains how to use business logic diagrams to identify entity types for conceptual data modeling for web applications development. The last section concludes this paper with some remarks and suggestions for future research.

WEB MODEL, BUSINESS LOGIC, AND WEB APPLICATION ARCHITECTURE

The structure of a web application resembles "hub-and-spokes" of a wheel. The hub serves as a nucleus where incoming traffic from a source is switched to the outgoing traffic bound for its destination. A web application consists of many interconnected hubs, each of which supports a set of connections (i.e., spokes). The nature of a hub is determined by the characteristics of its connections, sources, and destinations. We apply this structural metaphor to the proposed method. A source corresponds to a link page (i.e., a business façade web page), a destination to an anchor page (i.e., a server page or another business façade web page), a spoke to a link, and a hub to a business logic module that is represented by a set of link pages, anchor pages, and links. A web application can, therefore, be considered as a collection of business logic modules. They can be better explained in the context of web application architecture.

Web application architecture is important because it visualizes interactions among web application elements and logical separation of client and server pages, as well as it determines actual level of application performance, resource utilization, and maintainability. This paper adopts 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. Aforementioned, the proposed method is built around business logic and its components (The terms "business logic" and "business logic module" will be used interchangeably throughout the paper). Individual 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. 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. Now the question is how the concepts described above can help the designer thoroughly identify and clearly define entity types. To answer the question, page types, link types, identification and organization of business logic modules, and business logic diagrams should be introduced.

PAGE TYPES

Before we describe each of the page types used to construct this method, we need to understand how a web page is rendered. To open a web page in a user agent (in general, a web browser) a user types in a URL and hit an enter key on keyboard or press a button on a browser. Right after the key stroke, the

Table 1. Page types.

Architectural Layer	Page Type	Description
Presentation layer	R/R non-interactive page (RRNIP)	A pure HTML page that *has no* form for, say, registration. This is often called a client page and is for a business façade.
	R/R interactive page (RRIP)	A pure HTML page that *has* a form for, say, registration. This is also often called a client page and is for a business façade.
	R/E/R non-interactive page (RERNIP)	A page resulted from an execution of a server page. This page type *has no* form. This is often called a derived page and is for a business façade. This type of page shows an instance of physically one server page but logically two different pages.
	R/E/R interactive page (RERIP)	A page resulted from an execution of a server page. This page type *has* a form. This is also often called a derived page and is for a business façade. This type of page shows an instance of physically one server page but logically two different pages.
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.

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 pure HTML document or contains HTML elements only, the server copies the page and puts it as part of its 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 stripping the HTML tags away through an interpretation process. This kind of rendition involves a simple request/response (R/R) procedure while utilizing the static service of a web server. The other page rendition mode involves one more step than the R/R procedure does. When a requested page contains server-side scripts written in a scripting language (e.g., ASP.NET or PHP), then the web server executes the page logic specified by the scripts and builds a HTTP response message with the execution results. This more sophisticated rendition, thus, requires a request/execution/response (R/E/R) procedure. These two distinct procedures lead to a classification of web pages for both presentation layer and business layer (Table 1 above).

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, simply interconnecting pages 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 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 "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 [11]). This sort of association is dubbed "state" link. Imagine that a user submit a registration form on a RERIP to a RERSP. This kind of relationship is named "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 "directive" link because it directs the resource page to be processed before the other page is processed [see 13 for issues and types of links]. However, the method concerns only the

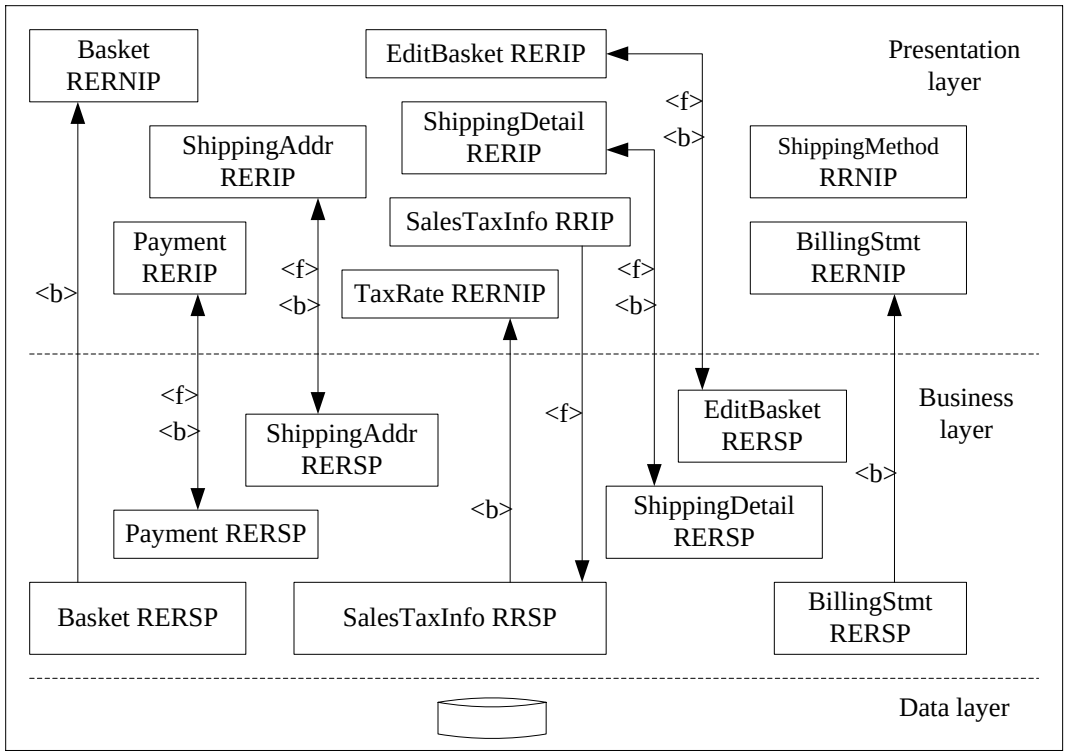

Figure 1. An example of business logic diagram.

build link (<b>), state link (<s>), and form link (<f>) types because they would carry substantial data between pages across the presentation and business layers.

IDENTIFICATION OF BUSINESS LOGIC MODULES

As mentioned earlier, a web application can be seen as a collection of loosely-coupled and tightly-cohesive business logic modules (BLMs). For example, a web application for selling certain products could be divided into two distinct sub-domains such as "storefront" and "administration." For the storefront sub-domain, we would need principal BLMs such as "authentication," "shopping cart," and "catalog browsing." This example implies two important things. First, a target web application, depending on its scope and nature, can consist of multiple sub-domains or even a set of heterogeneous applications if the proposed web application is supposed to integrate them. Second, depending on its size and cohesiveness in terms of the number of functions a BLM performs and the logical relationships between the functions, respectively, a BLM may need to be decomposed into smaller and more cohesive ones. To identify all the necessary BLMs for a web application, the designer may use a high-level description of the application, documents, other data sources, and user interviews about proposed web applications. For certain web applications, even currently running web applications may be considered to get general ideas [16].

BUSINESS LOGIC DIAGRAMS

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 method, 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, a parent BLM labeled "shopping cart" of an online bookstore could be decomposed into two child BLMs, which could be named "shopping cart for prospective customers" and "order placement." The former would include functions like storing customer wish lists, notifying the lists upon customer log-ins, converting the lists into orders, etc. The functions seem to be all logically cohesive so no further decomposition may be necessary. Once we get a hierarchy of BLMs, the next step is to draw a business logic diagram (BLD) for each of *leaf* nodes in the hierarchy by incorporating the architecture, page types, and link

types. Figure 1 shows an example of BLD for "order placement" of an online bookstore.

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 by 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, say, for authentication business logic, 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 existing address(es). To add a new address the page should have a form, and 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 purely written in HTML 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 do not carry any substantial data. Note that, as we can see in Figure 1, any possible links between presentation-layer pages and between business-layer pages are not our interest. It should be agenda for navigation design. Furthermore, although not shown, we should assume there are connections between business-layer pages and a database shown in the data layer. Then how to identify entity types from a BLD?

IDENTIFICATION OF ENTITY TYPES

As implied by Figure 1, links between pages may carry one or more data abstractions 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 1. 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 same data abstractions. The names of data abstractions then become entity types. If a link carries one or more data elements, we can infer data abstractions 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 [14], 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 important task is 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, so we argue that the method is robust

and easy to follow. One thing to note is that the method, although it is not explicitly described, could readily be extended to cover identification of attributes of and relationships between entity types.

REFERENCES

1. Abiteboul, S. (1997). "Querying semi-structured data," *Proceedings of the International Coference on Database Theory (ICDT)*, Delphi, Greece.
2. Atzeni, P., Mecca, G., Merialdo, P. and 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*, pp. 436-450, Valencia, Spain.
3. Batini, C., Ceri, S. and Navathe, S. B. (1991). Conceptual database design: an Entity-relationship approach. Benjamin-Cummings, Redwood City, CA.
4. Buneman, P. (1997). "Semistructured data," *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 117-121, Tucson, Arizona.
5. Carraro G. and Chong, F. (2006) "Software as a Service (SaaS): An Enterprise Perspective," Microsoft, http://msdn2.microsoft.com/en-us/architecture/aa905332.aspx.
6. Conallen, J. (2000). Building Web Applications with UML, Addison-Wesley, MA:Reading.
7. Dubey, A. and 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.
8. Florescu, D., Levy, A. and Mendelzon, A. (1998). "Database techniques for the World-Wide Web: a survey," *ACM SIGMOD Record*, Volume 27, Issue 3, pp. 59-74.
9. Garzotto, F., Paolini, P., and Schwabe, D. (1993). HDM: A model-based approach to hypertext application design," *ACM Tranactions on Office Information Systems*, Volume 11, Issue 1 pp. 1-26.
10. Genero, M., Poels, G. and Piattini, M. (2008). "Defining and validating metrics for assessing the understandability of entity–relationship diagrams," *Data & Knowledge Engineering*, Volume 64, Issue 3, pp. 534-557.
11. Introduction to Web Forms State Management, Microsoft, http://msdn2.microsoft.com/en-us/library/75x4ha6s(VS.71).aspx.
12. Isakowitz, T., Stohr, E. A., and Balasubramanian, P. (1995). "RMM: A Methodology for Structured Hypermedia Design," Communications of the ACM, Volume 38, Number 8, pp. 34-44.
13. Link Types, World Wide Web Consortium (W3C), http://www.w3.org/DesignIssues/LinkTypes.html.
14. Misic. M. and 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*, Volume 36, Issue 4, pp. 86-90.
15. Myers, B. Hollan, J., Cruz, I., Bryson, S., Bulterman, D., Catarci, T. Citrin, W., Glinert, E., Grudin, J. and Ioannidis Y. (1996). "Strategic directions in human-computer interaction," *ACM Computing Surveys*, Volume 28, Issue 4, pp. 794-809.
16. O'Reilly, T. (2000). "The Internet patent land grab," *Communications of the ACM*, Volume 43, Issue 6, pp. 29-31.
17. Teorey, T. J., Yang, D. and Fry, J. P. (1986). "A logical design methodology for relational databases using the extended entity-relationship model," *ACM Computing Surveys*, Volume 18, Issue 2, pp. 197-222.