Case Study: Design of a Web-Based Service Delivery System

Min Song and Il-Yeol Song
Drexel University, USA

songiy@Drexel.edu and Min.song@Drexel.edu

Abstract

In this paper, we present a case study on the design of a web-based online service delivery system using IBM's Net.Commerce system as well as the lessons learned in completing this project. The design specification is presented using the UML notation, while database schema is presented using the IDEF1X notation. Our design specification includes architecture, system components using package diagrams, system functions using use case diagrams, their processing logic using activity diagrams, and database design. We present a detailed database design and comment on design and customization considerations specific to e-commerce systems. Our experience shows that e-commerce tools still lack certain functionality such as processing back orders, allowing for customizable returns, and sending email notification to users, but overall can speed up the development of the system. Understanding the structure and transaction processing of e-commerce database systems will help system designers effectively develop and maintain these systems. Readers of this paper will understand and learn a typical design specification of a web-based service delivery system and various technical design issues.

Keywords: E-commerce, Database Design, UML, Net.Commece, Case Study

Introduction

In this paper, we present the design specification of a web-based online service delivery system. In general, an e-commerce system is built by following one of two approaches. The first approach is the customization approach using a suite of tools such as IBM's Net.commerce (Shurety, 1999), which was recently expanded into WebSphere Commerce Suite. For example, the Commerce Suite provides tools for creating the infrastructure of a virtual shopping mall, including catalog, and registration templates, a shopping cart, order and payment processing, and a customizable database. The second approach is the bottom-up development of an in-house system by experts of an individual company. In this case, the developer is manually building a virtual shopping mall with mix-and-match tools. In addition, a database supporting the business model of the e-commerce system must be manually developed. While the customization approach speeds up the development process, it is limited by the capacity of the tools used. On the other hand, even though the bottom-up approach takes a longer time than the former approach, it provides a flexible environment for customization at the expense of integration complexity. For a more detailed discussion on tools and approaches for developing web-based applications, we refer to the work by Fraternali (1999).

Whether a developer is using the customization or the bottom-up approach, understanding the structure and processing logic of the e-commerce system will help the developer effectively develop and maintain the system. Our paper is based on our experience of building a web-based online service delivery system using the customization approach.

The system was developed using *IBM's Net.Commerce* infrastructure to meet our business goals and needs. There were several critical reasons that *IBM's Net.Commerce* system was chosen. First, *Net.Commerce* supported multiple storefronts. Currently, we have built one storefront, but in the future our storefront will be migrated into a multiple storefront environment incorporating sister companies. Second, *Net.Commerce* provided us with flexibility in terms of integration with other business transaction sys-

tems. In spite of these well-featured functionalities, we had to customize *Net.Commerce* to accommodate certain business functions such as email notifications and returns processing. In this paper, we present system architecture and customized modules based on *Net.Commerce.* We then illustrate detailed design specifications, processing logics, and the database structure of our web-based service delivery system. We use UML (Booch, Rumbaugh, and Jacobson, 1999; Fowler, 1999) to present our design specifications, except for the relational schema, which uses IDEF1X notation (Song, Evans, and Park, 1995).

The design of the database structure for e-commerce systems requires different considerations than OLTP systems (Buchner Mulvenna, 1998; Ceri, Fraternali, and Paraboschi, 1999; Lohse and Spiller, 1998; Song and La-Van-Schultz, 1999; Song and Whang, 2000), such as:

Handling of multimedia and semi-structured data;
Translation of a paper catalog into a standard unified format and cleansing the data;
Supporting user interface at the database level (e.g., navigation, store layout, hyperlinks);
Schema evolution (e.g., merging two catalogs, category of products, sold-out products, new products);
Data evolution (e.g., changes in specification and description, naming, prices);
Handling meta data;
Capturing data for customization and personalization such as navigating data within the context.

For example, in our system, we used many variable-length data types to handle semi-structured data types. Our catalog has a lattice structure. We have designed the system so that further evolution of the system could be gracefully accommodated. Whenever possible, we took into account the structure growth, function growth, and volume growth of the system in our design considerations. Understanding the structure and transaction processing of e-commerce systems will help system designers effectively develop and maintain e-commerce systems.

The rest of this paper is organized as follows: Section 2 presents the architecture of the system and an overview of the system components. Section 3 presents the design specifications. In this section, we present requirements in terms of use case diagrams and their processing logic in terms of activity diagrams including personalizing displays, browsing catalogs, and placing orders. Section 4 illustrates a detailed database design and comments on design considerations specific to e-commerce database systems. Section 5 concludes the paper.

Architecture and Overview

Architecture

In this section, we briefly describe our e-commerce system architecture, which shares a lot of commonality with the generic *Net.Commerce* system. As illustrated in Figure 1, our system consists of the four major system components (Doemer et al., 1999):

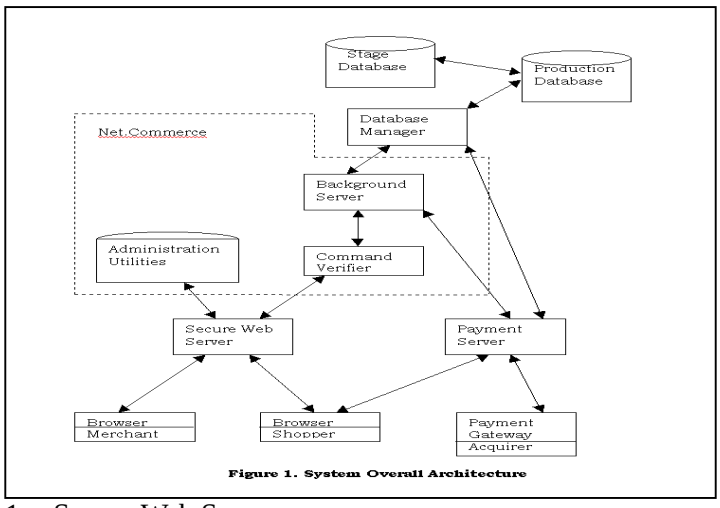


Figure 1. System Overall Architecture

1. Secure Web Server
2. Net.Commerce
3. Database Manager
4. Payment Server

The *Secure Web Server* component carries out functions such as receiving requests from users' browsers and forwarding them to the appropriate application. Since the customer's private information is captured and stored, this component is critical in terms of confidentiality of shopper transactions. In order to ensure that only authorized users perform system administration tasks, the web server implements the Secure Sockets Layer (SSL) protocol.

The *Net.Commerce* component plays a pivotal role in the overall system architecture. The major function of the component is to allow the system administrator to handle the system through the secure web-based administration facilities. In addition, it verifies commands invoked by the web server and process shopping requests.

The *Database Manager* component manages the *Net.Commerce* database which contains all the site data and store business transaction data. We use IBM DB2 Universal Database V5 as the database management system. Our *Net.Commerce* server and *Net.Data* use ODBC to connect to the database.

The *Payment Server* component manages payments using *CyberCash*. We use *CyberCash* as the method for processing credit card transactions. We first registered for the *CyberCash CashRegister* service, then downloaded and installed the *CyberCash* Merchant Connection Kit (MCK). The *CyberCash* function receives the shopper's credit card information and writes it to the database. A background server will periodically wake up and process new payments. It will connect to the *CyberCash* MCK and send the payment for processing and then update the database with the results.

High-Level Logical Components

In this section, we describe an overview of the functional components of our e-commerce system. The package diagram shown in Figure 2 enables us to see logical components of our e-commerce system.

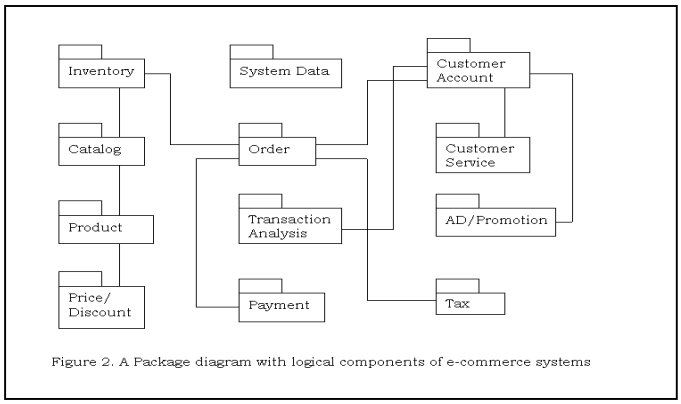


Figure 2. A Package diagram with logical components of e-commerce systems

A package in UML is a construct that groups inter-related modeling elements. Each package in Figure 2 contains one or more tables and their processing logic.

Package *Price/Discount* contains information that is needed for discount pricing and for determining which products and items are to be discounted at what rates for which shopper group. Package *AD and Promotion* involves tables that are related to advertising, promotion, and coupons. This package tracks which promotions are associated with which sessions, and which ads are displayed in which sessions. Package *Customer Service* is associated with customer feedback data for each user and order such as type, nature, status, and responses.

Package *System Data* holds various system parameters such as server configuration and access control parameters. Package *Order* keeps track of actual orders that are transferred from a shopping cart. This package keeps track of the products and items in shopping carts, the orders placed by shoppers, the suborders destined for different shipping addresses, the payment method, and the links between suborders and entries in the shoppers' address books. Packages *Inventory*, *Catalog*, and *Products* are closely related. *Products* holds the vendor data and products. *Inventory* lists all the items that are available for sale. *Catalog* is a standardized model that integrates various vendor-specific products in a domain. Since one vendor could use different terminology and format for the same product, *Catalog* protects the actual business model from the vendor-specific variations. *Catalog* acts as a buffer between *Products* and *Inventory. Catalog* also needs a dynamic catalog management module to accommodate new vendors and new products that were not considered in advance. Thus, actual implementation of the program is done against *Catalog* and *Inventory*. Our experience shows that developing the proper schema for *Catalog* for a specific domain is most time-consuming and complex.

The package *Tax* handles information about the tax categories and codes, as well as information about the codes that are used to represent different states or tax jurisdictions. The package *Payment* is associated with the *CyberCash* and *CashRegister*, an application that enables secure online transactions.

The package *Transaction Analysis* keeps track of a shopper's request and shopping behavior by logging transaction information directly to a database table. This package contains customization data.

System Function Specifications

In this section, we present functional requirements employing use case analysis and the processing logic of these *use cases*, using activity diagrams. We show two class hierarchies, shopper and catalog to shed light on some distinct characteristics of our e-commerce system.

Shopper Hierarchy

Use case analysis begins by identifying *actors* of the system. An actor is a user group with a specific role with respect to the system. The primary actors of our system are shoppers. The major users of our system can be classified as shown in Figure 3 below: As shown in Figure 3, shoppers are classified into guest shoppers and registered shoppers. Registered shoppers are further broken down into general shopper groups and customized shopper groups. A guest shopper can browse the catalog in the system, but is not allowed to place an order. There could be counter points to this restriction. This decision was made due to the fact that our products are expensive, and thus it would be

rare for a one-stop shopper to place an order without registration.

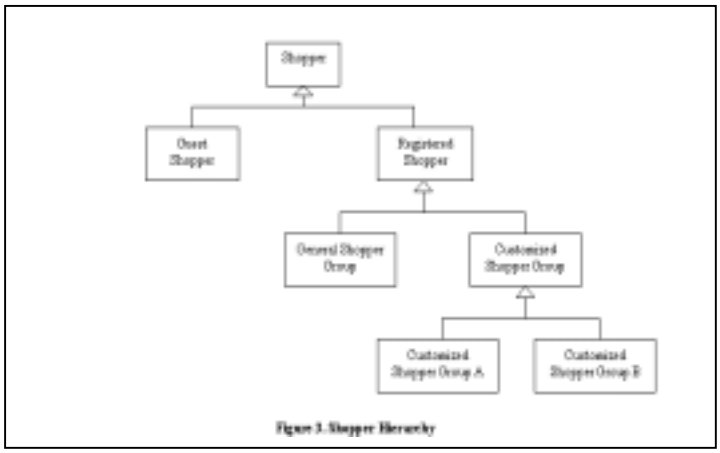


Figure 3. Shopper Hierarchy

For the purpose of personalization, we divide the registered shopper category into general shopper groups and customized shopper groups. Generalized shopper groups are not personalized, while customized shopper groups are personalized. A customized shopper group is assigned a shopper group ID and served with some personalized features in terms of different catalog templates and prices. A shopper group ID can be distributed to either a user or a group of users. A group of users can be defined by geography, types of business, or any criteria set up by the system administrator. For instance, we can set up the system in such a way that users who come from Japan will have a certain discount for a limited time period.

Catalog Hierarchy

As shown in Figure 4, our catalog system is a lattice structure. This structure is stored in the database and the web pages that display the catalog to the shopper can be built automatically using this data. *Catalog Home Page* is a root category, where *Catalog A* and *Catalog B* are subcategories. *Product A* and *Product B* are instances of commodities offered by our site, and can belong to multiple parent categories. Many products consist of multiple items that are variations of the base product.

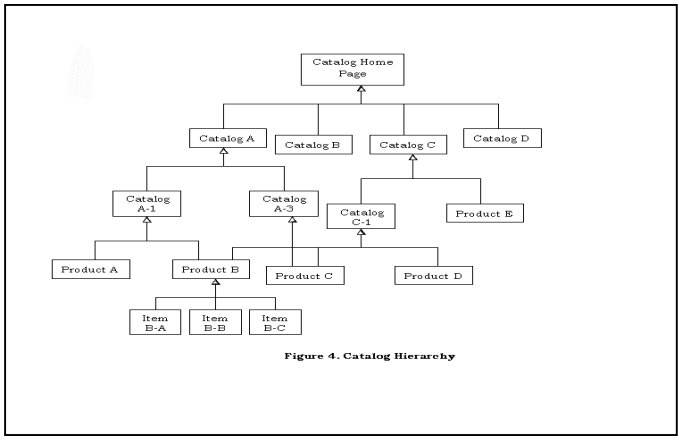


Figure 4. Catalog Hierarchy

Modeling System Functionality Employing Use Case Analysis

Use case analysis is widely used to capture high-level system functionality from each actor's point of view. During the development stage of the system, to identify all the actor activities and system requirements, we attempted to capture a set of use cases associated with the actors in our system design. The use case diagram shown in Figure 5 visualizes use cases of our e-commerce system.

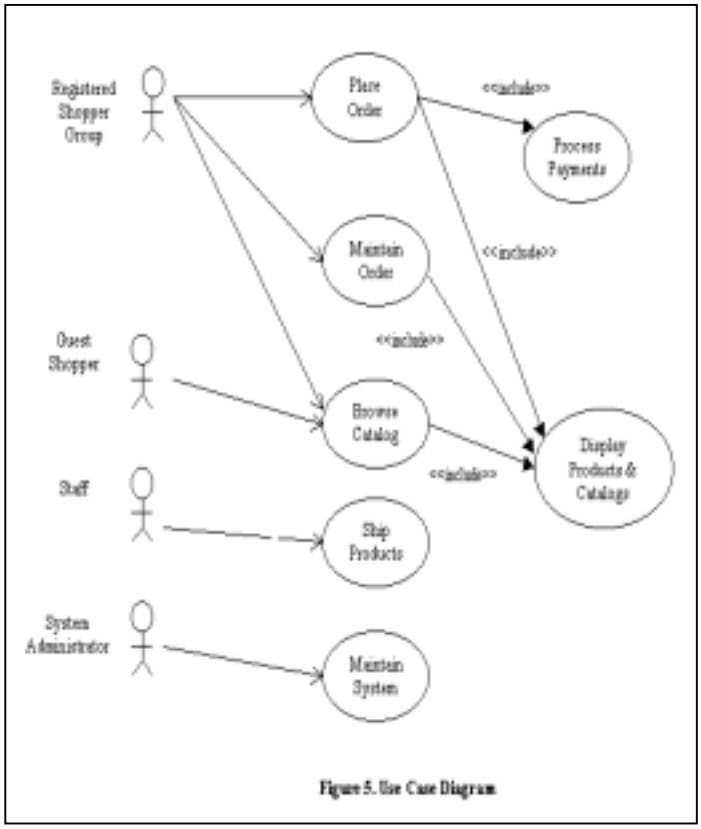


Figure 5. Use Case Diagram

The three *actors* that we identified are 1) Registered Shopper, 2) Guest Shopper, and 3) System Administrator. In addition, as illustrated in Figure 5, the relationship between use cases is shown in terms of the *include* relationship (Booch et al, 1999), which alleviates, otherwise, redundant steps describing a number of behaviors that are common across more than one use case.

To begin with, the *Browse Catalog* use case starts when either a registered shopper or a guest shopper selects the *Browse Catalog*. For the guest shopper, the regular catalog is displayed, but for the registered shopper group the personalized catalog is displayed. Since the *Place Order* use

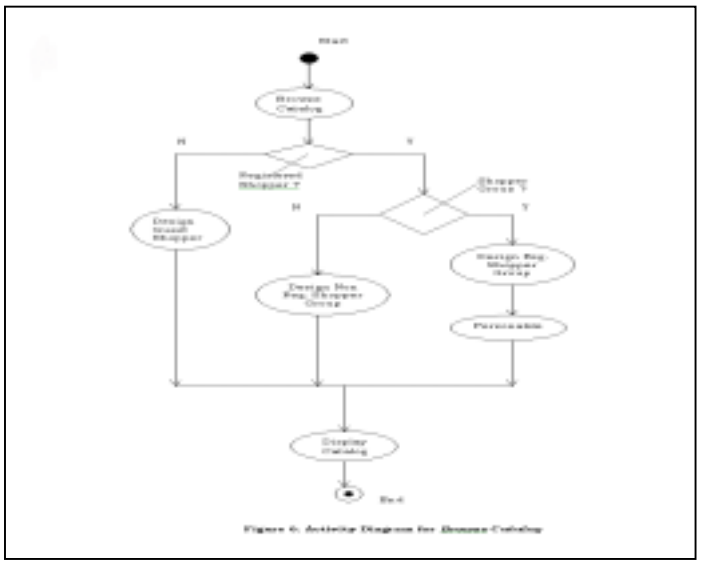
case is mandatory for any e-business system, we illustrate it in detail in the next section. The *Maintain Order* use case is associated with updating and canceling orders. When a user ends the shopping activity by completing an order or leaving the system without a purchase, the system updates the order status and the customer account. The *Ship Products* use case is performed by a staff actor after an order has been placed. The *Maintain System* use case is associated with a system administrator's tasks. A system administrator is involved in activities such as, staging copy and propagation between the product server and the staging server, monitoring user traffic, exercising data input, and enforcing access control.

Modeling Processing Logic Using Activity Diagrams

In this section, we illustrate the processing logic of use cases using activity diagrams. An activity diagram captures the workflow of a use case or a system function. Despite the fact that an activity diagram does not make the links among actions and objects very clear, it is suitable for modeling the functionality of the *use cases* of our system.

Browse Catalog

Figure 6 shows the processing logic of the *Browse Catalog* use case. The activity of *Browse Catalog* is first branched depending upon whether a user is registered or a guest. If the user is a guest shopper, only minimal benefit is provided to the user, and the user is not allowed to place an order. If a user was registered before, the activity is further branched depending on whether the registered shopper has any valid shopper group IDs. If so, personalized modules are generated so that the user can view a personalized catalog and prices. If the shopper does not belong to any shopper group at the moment, regular modules are shown to the shopper. Regardless of shopper type, all shopper groups may have equal access to a special offer for a limited amount of time.



Personalize

Figure 7 illustrates how activities associated with the *Personalize* common use case take place. The activity of *Assign Shopper Group ID based on purchase history* and *Send it to the targeted user* plays a critical role in creating a shopper group. Currently our criteria for the selection of the shopper group depends on the previous purchase records. For users who cancel the order that they intended to purchase, we store this information in our independent order system. Since we do not integrate the present order system into the e-commerce site at this time, we carry out this process manually. The activity of *Enter Shopper Group ID* is branched depending on whether a user has a valid shopper group ID. If the user does not, the system displays the regular catalog. If the user has a valid ID, the activity is further branched by whether or not the user successfully enters the shopper group ID. In the case that the user forgets his shopper group ID, he can provide the system with his e-mail address and user ID which is identical to the one he used for registration. If the e-mail and user ID are not the same, the user must contact us by phone or another communication tool to verify their identity. If the email and user ID match, the user will receive an e-mail containing the shopper group ID assigned to him.

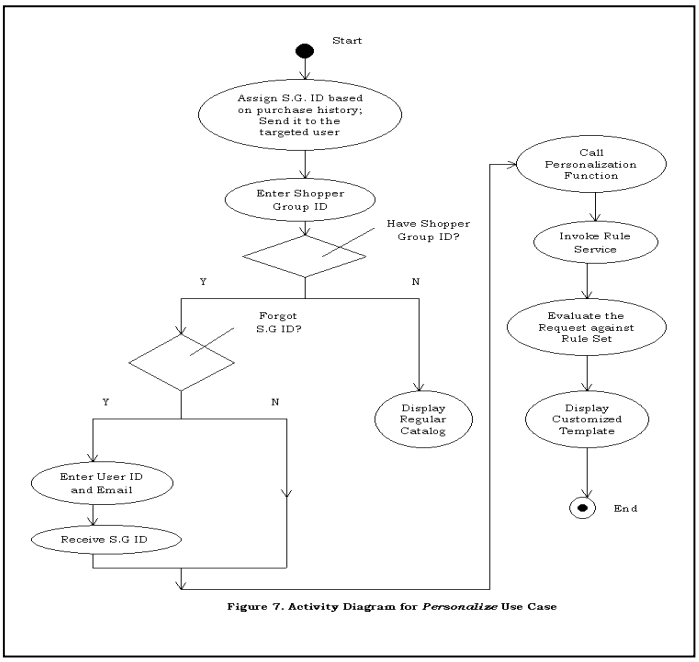


Figure 7. Activity Diagram for *Personalize* Use Case

If the user enters the system with the correct shopper group ID, the system generates a personalized catalog based on a predetermined set of rules in our personalization module. The rules are used to determine what products and categories are assigned to which shopper groups under what conditions. In general, each shopper group ID is assigned to at least one product or category. One shopper group ID can be assigned to many products and categories, but only one ID may be applied per session.

The personalization module is based on a set of rules. By using rules derived from the business policies of our company, the personalization module can evaluate a particular shopper or their purchases, and generate a list of products to recommend to them. The rules take the simple form of *<if condition, then action>*. For example, the following statement represents of an example of a personalization rule:

- If the customer has an interest in the subject field A, then recommend a product A1.

The rule processor evaluates whether the condition is true or not. If the condition is true, then it performs a particular action. Since maintaining rules and combining the results of rule processing are complex, we are in the process of migration of the system to WebSphere Commerce Suite, which provides all of the flexibility associated with an object-oriented approach.

The business logic behind our personalization module is straightforward as our module does not require sophisticated techniques such as data mining and collaboration filtering. As pointed out by Aggarwal and Yu (2000), personalization can be fine-tuned by explicit methods such as user feedback and ratings, or implicit methods, such as the observation of user buying behavior. In the next generation of our system, we will implement a personalization module with data mining techniques so that the statistics of the users' previous shopping behaviors are reflected in personalization.

Place Order

Figure 8 shows how activities in the *Place Order* use case occur. This use case represents the most critical component of the entire e-commerce system. The activity of *Select Order Detail* is branched depending upon whether the shipping address is the same as the address entered during registration. If the user prefers to receive the ordered product at an address different from the one in system, the user will be asked to enter this shipping information. Otherwise,

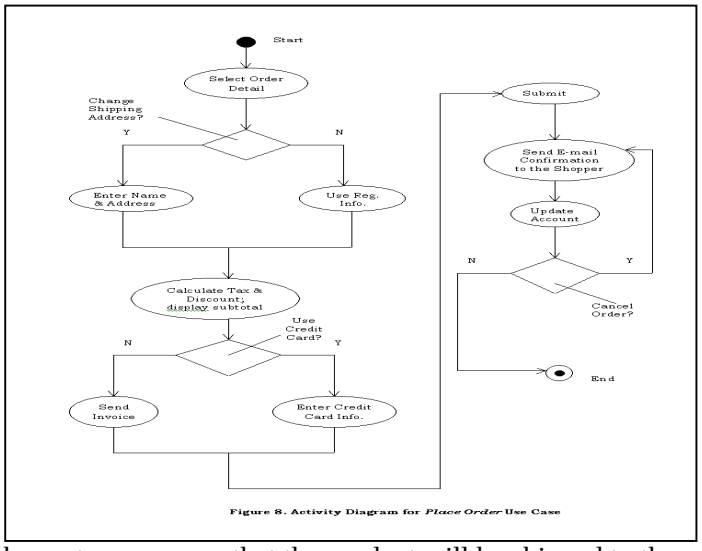


Figure 8. Activity Diagram for *Place Order* Use Case

the system assumes that the product will be shipped to the address specified during registration.

Once the shipping address is captured, the system calculates discount and tax, and displays the subtotal of the order. Even though a discount was applied to the product before, we still need a discount feature that allows us to get a shopper discount after tax is calculated for some products.

If the user wants to pay by credit card, the user is assisted while entering his credit card information. Also, if the user would like to receive an invoice, he can choose the option of *Send Invoice*. If the credit card information is incorrect, the system displays an error message so that the user can take proper action. Once the activity of *Submit* is finished, the system takes the user to the order confirmation page, and an e-mail order confirmation is simultaneously sent to the user at the e-mail address specified in the registration process. The *Update Account* activity is associated with the order status and the order history. If the users want to cancel the order, they will need to access a sub-module of our order system, called refund system, by entering their user ID, password, and credit information used to place the order. Upon completion of the cancellation process, the user's account is updated and an e-mail with a cancel confirmation is sent to the user accordingly.

Database Structure

In this section, we present a database schema for our e-commerce transaction processing (ECTP) system that sells inventory items. An overview of the ystem components was presented as a package diagram in Figure 2. Figure 9 shows the ECTP schema slightly modified from *Net.Commerce* (Doemer et al, 1999; Shurety, 1999). The schema includes some portions from the Catalog, Product, Order, Payment, and Price/Discount packages. Although

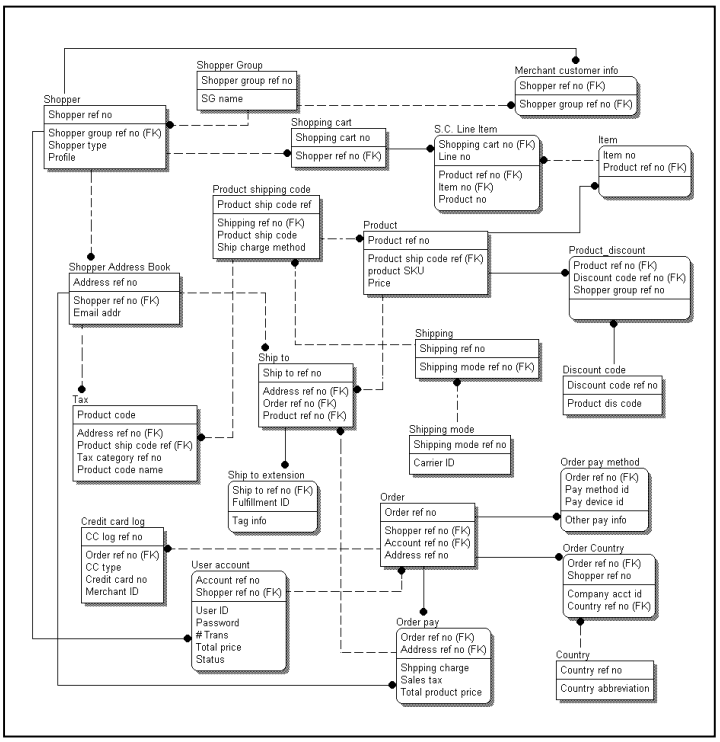


Figure 9. A database schema for an e-commerce transaction processing system

we intended to reflect the real world e-commerce database in Figure 9, the actual schema would vary and be more complicated at the time of system implementation. In addition, depending on the underlying business model and needs, the detailed schema will differ from case to case. Note that Figure 9 uses the IDEFIX notation (Song, Evans, and Park, 1995).

For most tables, we use a system-generated (surrogate) primary key (PK) to avoid dependence on data change. Smart keys, which have embedded meanings, create dependencies on the data. Thus, changes to the data causes changes to the primary keys. All tables in Figure 9 use surrogate keys. PKs are in the first compartment whereas non-PKs are in the second. FK represents a foreign key.

Figure 9 shows two types of relationships for specifying the cardinality of the relationships in IDEF1X; (1) one-to-many *identifying* relationship represented by a solid line; (2) one-to-many *non-identifying* relationship represented by a dashed line. For example, the relationship between SHOPPER and USER ACCOUNT is a one-to-many identifying relationship because the primary key of SHOPPER, SHOPPER REFERENCE NO#, becomes a part of the primary key of USER ACCOUNT. The relationship between SHIPTO and PRODUCT is a one-to-many non-identifying relationship because the primary key of PRODUCT, PRODUCT REFERENCE NO#, becomes a foreign key of SHIPTO, but not a part of the primary key of SHIPTO.

With regard to referential integrity, the following three types of deletion actions are considered: 1) Delete Cascade, 2) Delete No Action, and 3) Delete Set Null. The Net.Commerce system implements *delete cascades* to preserve the referential integrity of the database. When data is deleted from a table, the *delete cascade* causes the same data from related tables to be deleted. For example, ORDER and ORDERPAY is maintained by the Delete Cascade action. All tables are influenced by *delete cascade*.

The second type of the deletion action, *delete no action*, is used when the foreign key value needs to be preserved even when the primary key is deleted. For example, PRODUCT and SHIP TO relations use this type of deletion action. When a PRODUCT REF NO in PRODUCT is deleted, the foreign key at SHIP TO remains the same. This ensures that no information on any shipping order is lost when a PRODUCT REF NO in table PRODUCT is deleted . USER ACCOUNT and ORDER also use *Delete No Action.* Since only a registered shopper can place an order and a registered shopper cannot be deleted from the database, deletion of the shopper reference number is not allowed. However, when a shopper moves, the shopper can provide new contact address information, and a new entry is added to SHOPPER ADDRESS BOOK. The old address information is not discarded, but it is flagged as a temporary address in by the StatusFlag attribute, the temporary address can then be deleted by the *Cleanupafter 60 days* utility provided by Net.Commerce.

The third type of deletion action, *set value to null*, is used when the foreign key value does not have any meaningful value and when we do not want to delete the row with the foreign key. For example, TAX PRODUCT CODE in SHIP TO uses this deletion action. In these cases, if the unique TAX PRODUCT CODE is deleted, the product code assigned to the product for taxation purposes as defined by the Taxware Sales/Use Tax system is set to null.

There are a few custom-defined tables such as CREDIT CARD LOG, SHIP TO EXTENSION, and COUNTRY. The CREDIT CARD LOG table was created because it is needed for the refund procedure used by payment. The SHIP TO EXTENSION table was created to satisfy the need for storing extra values required by the company's tagged data files, which is used later for reporting to the fulfillment service department. The COUNTRY table was created for storing the company's country abbreviation from the Global conversion table,

which is the company's accounting system. ORDER COUNTRY table matches orders coming from a country that has an agency for the company.

Lesson Learned

In this section, we summarize the lessons learned in developing our system with Net.Commerce.

1. A project plan for customization

The original plan of developing our e-commerce site was to adapt an "out of box" model provided by the Net.Commerce software. We experienced difficulty when using the "out of box" model to directly meet the requirements of the internal client, which led to the need to customize Net.Commerce. Since Net.Commerce was supposed to be used for any domain, the tool was quite complicated in terms of understanding and customization. We had to simplify certain modules such as single storefront and order processing. Due to the complexity, the customization required more time to complete than we expected. One of the obstacles was a lack of consensus between a development team and the internal client in the early design phase. For instance, we were to develop a system function of blocking customers from certain countries where agents got contracts to sell our products. The decision was made not to block the customers, but instead, to take them to the agent sites when they wanted to order products through our system. Other customization efforts included email notifications and handling of returns based on company procedures.

2. Deficient information about the system architecture

The Net.Commerce system architecture was not clearly explained in the software package. In the first design phase, we had difficulty understanding the system due to this deficiency. In particular, the lack of detailed understanding of the software architecture made it harder for us to customize Net.Commerce to meet our business needs. In addition, Net.Commerce did not come with adequate documentation for customization, forcing us to rely on costly consultants.

3. Lack of communication among the development team

We realized an effective communication channel between development team members must be developed. Several developers wrote different pieces of the C++ classes based on Net.Commerce libraries , simultaneously. As a result, we experienced some inconsistency and redundancy in the writing of the program. In addition, the project suffered from ineffective communication among the internal clients, project mangers, and developers.

4. Integration with the legacy systems

We experienced difficulty integrating the e-commerce system with the existing business systems such as the accounting and tax systems. Due to the lack of automated validation and reporting capabilities in Net.Commerce, we had to build an interface module that automatically handled transactions between our e-commerce system and the legacy system.

5. Educating all related stakeholders

We also observed that some members resisted the change caused by the new e-commerce system and experienced project delay as a result. This reconfirms the importance of educating all related stakeholders regarding the development of the project from the beginning.

6. Catalog design, construction and maintenance

Given various types of products, it was critical for us to come up with an appropriate catalog structure. Although we spent a good deal of time designing the catalog and constructing the catalog system, we continually had to update the catalog structure due to lack of complete understanding of the products. This problem resulted from miscommunication between the system developing team and the internal system users. This issue reconfirms the importance of thorough requirements modeling in minimizing costly updates and maintenance.

7. Training the Storefront Administrator

In our case, the marketing group was responsible for managing the storefront through the web-based administration tool. Due to their lack of technical knowledge, they were afraid of using the administration tool and tended to depend on the developers to complete their tasks. Although Net.Commerce provides the web-based administration tool, called NCADMIN, it was not designed so that the novice user could utilize it, which led to many unnecessary mistakes.

8. Handling Referential Integrity Constraints

We learned that not every referential integrity constraint could be automatically enforced. Depending on the semantics of the relationships, different types of referential ac-

tions such as DELETE CASACDE, DELETE NO ACTION, or DELETE SET NULL must be carefully selected.

The overall lesson learned is that the project team should take an iterative design approach in building the system. During project planning, software demonstration and deployment should be integrated into the schedule. Less emphasis should be placed on the number of technologies involved, but rather on the impact these technologies will have and the value they will add to the business. Project team members should have a firm understanding of innovative and large-scale from the beginning of the project.

Net.Commerce is full-featured and powerful but requires experienced programmers to implement its capabilities. Its development and maintenance tools, however, are still fairly rudimentary, and it needs more GUI interfaces and wizards, as well as better integration of existing modules into a more coherent package.

Conclusion

In this paper, we presented a case study on the design of a web-based service delivery system using IBM's *Net.Commerce* system. We have presented technical design specifications and lessons learned from the project. We presented architecture, system components using package diagrams, system functions employing use case diagrams, their processing logic using activity diagrams, and database design. With the power of Net.Commerce and some help from consultants, we were able to successfully customize the system for our business needs.

We also presented a detailed database schema for our e-commerce transaction processing system. Most real-world e-commerce database schema for a service delivery system will have a similar framework as we presented in this paper. Understanding the structure of e-commerce systems and its processing logics will help to effectively develop and maintain the system, regardless of the approach taken and tools used. Our experience shows that e-commerce tools still lack certain functionality such as processing back orders, allowing for customizable returns, and emailing notification to users, but overall can speed up the development of the system.

The lessons learned from this project show that the development of a web-based system should also be based on a typical system development methodology. Hurdles met included a thorough understanding on system architecture, integration with legacy systems, a thorough requirement modeling, educating all the stakeholders, training users, and careful handling of referential integrity constraints.

Reference

Booch G., Rumbaugh, J. and Jacobson, I. (1999) UML User's Guide, Addison Wesley.

Buchner, A and Mulvenna, M. (1998) Discovering Internet Market Intelligence through Online Analytical Web Usage Mining. SIGMOD Record, 27(4): 54-61.

Charu C. Aggarwal and Yu, Philip S. (2000) Data Mining Techniques for Personalization, IEEE Data Engineering, 23(1): 4-9.

Ceri S., Fraternali P. & Paraboschi, S. (1999) Design Principles for Data-Intensive Web Sites. SIGMOD Record, 28(1): 84-89.

Doemer R., Ezers P., Gustavsson P., May C., and Shull G. (1999) Building e-commerce Solutions with Net.Commerce: A Project Guidebook, IBM.

Fraternali P. (1999) Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. ACM Computing Surveys, 33(3): 227-263.

Fowler, M. (1999) UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley: Harlow, England.

Lohse, G. L. and Spiller P. (1998) Electronic Shopping. Communications of the ACM, 41(7): 81-88.

Shurety S. (1999) E-Business with Net.Commerce. Prentice Hall.

Song I., Evans M, and Park E. (1995) A Comparative Analysis of Entity-Relationship Diagrams. Journal of Computer and Software Engineering, 3(4): 427-459.

Song I and LeVan-Schultz K. (1999) Data Warehouse Design for E-Commerce Environments. Lecture Notes in Computer Science, 1727: 374-388.

Song I and Whang K (2000) Design of Real-World E-commerce Systems. IEEE Data Engineering Bulletin, 23(1): 23-28.

Treese, G.W. and Stewart, L.C. (1998) Designing Systems for Internet Commerce, Addison Wesley.

Biographies

Min Song is a Software Engineer at ISI Thomson. He is a certified IBM Net.Commerce developer and currently a PhD student of College of Information Science and Technology at Drexel University.

Il-Yeol Song is a professor of Drexel University since 1988. Prof. Song has authored over 75 papers on the subject of database design, object-oriented analysis & design, and data warehousing. Prof. Song co-chaired ACM CIKM ’99, DOLAP ’98 and DOLAP ’99