Application of MVC Platform in Bank E-CRM

Liancai Hao

(School of Management, Harbin Institute of Technology, Harbin P. R. China 150001)
Haolc@hit.edu.cn

Abstract

Customer relationship management (CRM) has been as important to the banking industry at the start of the 21st century as it has been to any other industry. Banks have used E-CRM tools to acquire more customers and to improve relationships with them. Model-View-Controller (MVC) pattern is a software design pattern that is suitable for interactive system. This paper analyzes the CRM characteristics: interactive, expansibility, the complexity of data and integration. To these characteristics of CRM, it provides a system design scheme on MVC pattern, and describes the advantages of function and structure in the system.

Keywords: *Customer Relationship Management; E-CRM; MVC pattern; J2EE*

1. Introduction

Today, many businesses such as banks, insurance companies, and other service providers realize the importance of Customer Relationship Management (CRM) and its potential to help them acquire new customers retain existing ones and maximize their lifetime value. At this point, close relationship with customers will require a strong coordination between IT and marketing departments to provide a long-term retention of selected customers.

CRM is an integration of information technology and management method. Design and development of CRM is a key step of solution of enterprise's CRM. More competition and increased regulation made it more difficult for banks to stand out from the crowd. However, the development of CRM gave proactive banks access to technology that helped them improve customer retention by using customer feedback to offer conveniences like ATMs and online banking. Banks can also use CRM tools to improve customer loyalty by using data collected through customer sign-ups, transactions and feedback processes.

E-CRM is defined as a mean of selling, serving, or communicating with customer through the web. Further he takes E-CRM as a subset of CRM, which means that E-CRM is one of the channels that a company can use to deploy its CRM strategies [1]. The goal of E-CRM system is to improve customer service, retain valuable customers and to aid in providing analytical capabilities within an organization [2].

This paper deals with the role of Customer Relationship Management in banking sector and the need for Customer Relationship Management and provides a Design of CRM System Based on MVC Pattern according to the CRM characteristics.

2. The characteristics of E-CRM

CRM is a broad approach for creating, maintaining and expanding customer relationships. CRM is the business strategy that aims to understand, anticipate, manage and personalize the needs of an organization's current and potential customers. At the heart of a perfect strategy is the creation of mutual value for all parties involved in the business process. It is about creating a sustainable competitive advantage by being the best at understanding, communicating, and delivering and developing existing customer relationships in addition to

creating and keeping new customers. So the concept of product life cycle is giving way to the concept of customer life cycle focusing on the development of products and services that anticipate the future need of the existing customers and creating additional services that extend existing customer relationships beyond transactions. The basic CRM functions are collecting the market and customer data from external enterprise, processing the operational data from internal enterprise, analyzing all the information and providing the information and knowledge for managers and operators of the enterprise. So CRM system has interaction, expansibility, complexity of data and integration [3].

2.1. Interaction

Firstly, CRM system provides the touch points for customers in the business procedure in order to keep in touch with customers. Secondly, it transfers the customer information to staff of the enterprise to support the deciding and executing as soon as possible. The numerous users are decision-makers, administrators, and salesmen, retailers and customers who have various demands. The terminals of CRM system include PC, ATM, Web-self-service, PDA and wireless or mobile communicators. So CRM system is powerful interactive system.

2.2. Expansibility

Usually, the new modules are needed during CRM system running, the modifications come from the changing of the staff (or customers) and business procedure. Users can divide CRM system into particular modules that have special function, and assembled the modules for their own demands.

2.3. Complexity of data

The core function of CRM system is processing data. The data of CRM is dispersed, various and dynamic. The data that come from different channels and change with the customers changing include individual data, such as name, age, incoming, purchase list and personal habits. Different users need different content and vision of data.

2.4. Integration

CRM system can be integrated with internal and external IT systems to use existing resources adequately, share all the information, and maximize benefit net scale. So the interface of CRM system can fit for different kinds of OS and database.

According to these characteristics, MVC is a suited design pattern for CRM system development.

3. E-CRM in banking

Bank merely an organization it accepts deposits and lends money to the needy persons, but banking is the process associated with the activities of banks. It includes issuance of checks and cards, monthly statements, timely announcement of new services, helping the customers to avail online and mobile banking etc. Huge growth of customer relationship management is predicted in the banking sector over the next few years. Banks are aiming to increase customer profitability with any customer retention. This paper deals with the role of CRM in banking sector and the need for it is to increase customer value by using some analytical methods in CRM applications. It is a sound business strategy to identify the bank's most profitable customers and prospects, and devotes time and attention to expanding account

relationships with those customers through individualized marketing, pricing, discretionary decision making [4-6].

In banking sector, relationship management could be defined as having and acting upon deeper knowledge about the customer, ensure that the customer such as how to fund the customer, get to know the customer, keep in tough with the customer, ensure that the customer gets what he wishes from service provider and understand when they are not satisfied and might leave the service provider and act accordingly.

CRM in banking industry entirely different from other sectors, because banking industry purely related to financial services, which needs to create the trust among the people. Establishing customer care support during on and off official hours, making timely information about interest payments, maturity of time deposit, issuing credit and debit cum ATM card, creating awareness regarding online and e-banking, adopting mobile request etc are required to keep regular relationship with customers.

The present day CRM includes developing customer base. The bank has to pay adequate attention to increase customer base by all means, it is possible if the performance is at satisfactory level, the existing clients can recommend others to have banking connection with the bank he is operating. Hence asking reference from the existing customers can develop their client base. If the base increased, the profitability is also increase. Hence the bank has to implement lot of innovative CRM to capture and retain the customers.

There is a shift from bank centric activities to customer centric activities are opted. The private sector banks in India deployed much innovative strategies to attract new customers and to retain existing customers. CRM in banking sector is still in evolutionary stage, it is the time for taking ideas from customers to enrich its service. The use of CRM in banking has gained importance with the aggressive strategies for customer acquisition and retention being employed by the bank in today's competitive milieu. This has resulted in the adoption of various CRM initiatives by these banks.

The idea of CRM is that it helps businesses use technology and human resources gain insight into the behavior of customers and the value of those customers. If it works as hoped, a business can: provide better customer service, make call centers more efficient, cross sell products more effectively, help sales staff close deals faster, simplify marketing and sales processes, discover new customers, and increase customer revenues. It doesn't happen by simply buying software and installing it. For CRM to be truly effective, an organization must first decide what kind of customer information it is looking for and it must decide what it intends to do with that information. For example, many financial institutions keep track of customers' life stages in order to market appropriate banking products like mortgages or IRAs to them at the right time to fit their needs. Next, the organization must look into all of the different ways information about customers comes into a business, where and how this data is stored and how it is currently used. One company, for instance, may interact with customers in a myriad of different ways including mail campaigns, Websites, brick-and-mortar stores, call centers, mobile sales force staff and marketing and advertising efforts. Solid CRM systems link up each of these points. This collected data flows between operational systems (like sales and inventory systems) and analytical systems that can help sort through these records for patterns. Company analysts can then comb through the data to obtain a holistic view of each customer and pinpoint areas where better services are needed.

4. MVC design pattern introduce

MVC (Model-View-Control) design pattern is for the system that presents several views for the same data. It separates the data lay and the expression lay, and divides the system

objects into three classes: Model class, View class and Control class. Model class implements the business and data logic; View class implements the display logic, and Control class implements the control processing. MVC keeps the three logic types independency to build the business and data logic oriented business and design the control and display logic oriented application. The changing of business processing does not modify the business and data logic. The changing of business principles and algorithms only modifies Model class. So MVC separates the data accessing and display and ensure the modules independency. MVC framework is as Figure 1.

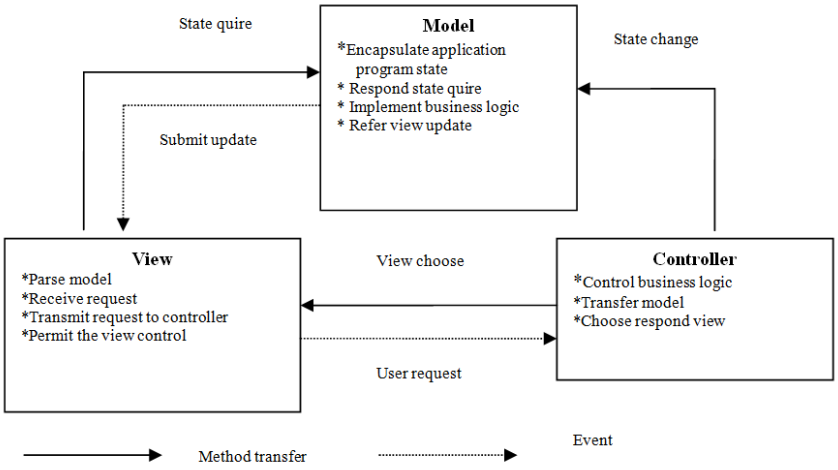

Figure 1. MVC models framework

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them [7].

A controller can send commands to its associated view to change the view's presentation of the model (*e.g.*, by scrolling through a document). It can also send commands to the model to update the model's state (*e.g.*, editing a document).

A model notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.

A view requests from the model the information that it needs to generate an output representation.

It is important to note that both the view and the controller depend on the model. However, the model depends on neither the view nor the controller. This is one the key benefits of the separation. This separation allows the model to be built and tested independent of the visual presentation. The separation between view and controller is secondary in many rich-client applications, and, in fact, many user interface frameworks implement the roles as one object. In Web applications, on the other hand, the separation between view (the browser) and controller (the server-side components handling the HTTP request) is very well defined.

The active model is used when the model changes state without the controller's involvement. This can happen when other sources are changing the data and the changes must be reflected in the views. Consider a stock-ticker display. You receive stock data from an external source and want to update the views (for example, a ticker band and an alert

window) when the stock data changes. Because only the model detects changes to its internal state when they occur, the model must notify the views to refresh the display.

However, one of the motivations of using the *MVC* pattern is to make the model independent from of the views. If the model had to notify the views of changes, you would reintroduce the dependency you were looking to avoid. Fortunately, the *Observer* pattern provides a mechanism to alert other objects of state changes without introducing dependencies on them. The individual views implement the *Observer* interface and register with the model. The model tracks the list of all observers that subscribe to changes. When a model changes, the model iterates through all registered observers and notifies them of the change. This approach is often called "publish-subscribe." The model never requires specific information about any views. In fact, in scenarios where the controller needs to be informed of model changes (for example, to enable or disable menu options), all the controller has to do is implement the *Observer* interface and subscribe to the model changes. In situations where there are many views, it makes sense to define multiple subjects, each of which describes a specific type of model change. Each view can then subscribe only to types of changes that are relevant to the view.

Architecting the presentation layer around the *MVC* pattern results in the following benefits and liabilities:

Supports multiple views. Because the view is separated from the model and there is no direct dependency from the model to the view, the user interface can display multiple views of the same data at the same time. For example, multiple pages in a Web application may use the same model objects. Another example is a Web application that allows the user to change the appearance of the pages. These pages display the same data from the shared model, but show it in a different way.

Accommodates change. User interface requirements tend to change more rapidly than business rules. Users may prefer different colors, fonts, screen layouts, and levels of support for new devices such as cell phones or PDAs. Because the model does not depend on the views, adding new types of views to the system generally does not affect the model. As a result, the scope of change is confined to the view. This pattern lays the foundation for further specializations of this pattern such as *Page Controller*and *Front Controller*.

Complexity. The *MVC* pattern introduces new levels of indirection and therefore increases the complexity of the solution slightly. It also increases the event-driven nature of the user-interface code, which can become more difficult to debug.

Cost of frequent updates. Decoupling the model from the view does not mean that developers of the model can ignore the nature of the views. For example, if the model undergoes frequent changes, it could flood the views with update requests. Some views, such as graphical displays, may take some time to render. As a result, the view may fall behind update requests. Therefore, it is important to keep the view in mind when coding the model.

Although originally developed for personal computing, Model View Controller has been adapted as architecture for World Wide Webapplications. Several commercial and noncommercial application frameworks have been created that enforce the pattern. These

frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server [8].

Early web MVC frameworks took a thin client approach that placed almost the entire model, view and controller logic on the server. In this approach, the client sends either hyperlink requests or form input to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server [5]. As client technologies have matured, frameworks such as JavaScriptMVC and Backbone have been created that allow the MVC components to execute partly on the client (see also AJAX).

MVC separates the expression and content so well, for example, code of data lay such as SQL order and code of expression lay such as HTML, that the programs that provide users many types of data operations could repeat code very little. It supports several views to display many data vision, and separates data expression and control. MVC usually implements on the J2EE frame that can well interface with all the kinds of Relation-Database, Event-Process Server, Content Server, and Mail Server by API (JDBC、JMS、XML、JNDI, CORBA *etc*.). And J2EE supports the Web service that has integrative ability to protect existing investing and leave space for expansion [9].

5. CRM system design based on MVC

Table 1. CRM System Modules and Part of Functions

	Sale Automatic	Marketing	Customer Service	Call Center	E-commerce
Users Management Function	Salesman Account, Order, Price Sales, Report	Marketing staff Promotion, Customer Segmentation	Service staff Service, E-mail, Appeal	Receiver Sales & Service On real time	Customer, staff Integrate all function
Including information	Product & price list, purchase & stock recorder	Customer, product, marketing knowledge	Customer, product, order, linkman	All information	All information

The system function modules are as Table 1. An implement of order management explains the design. System is developed on the J2EE, and MVC is mapped to J2EE (Figure 2).

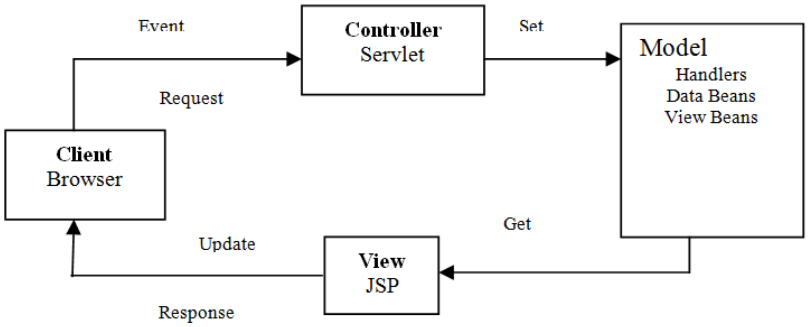

Figure 2. System Structure on J2EE

In the system, Java is used to develop the application program at the server, apache tomcat is used as application program, SQL Server2000 is DBMS and JDBC2 is the interface of database. The order management function is implemented as follows:

5.1. Database Design

The table 3 order information table for orders in database:

Table 2. Order information table

Field name	OrderNo	CustNo	PrdtNo	Price	Amnt	TotPrc	PayAmnt	PayDate
Describe	Order number	Customer number	Product Number	Per price	Order amount	Total Price	Paid amount	Paying date

5.2. Function Design

Many Java Beans compose model lay. And according to these functions, Java Beans are defined as three classes: Handlers\Data Beans\View Beans. Handlers transfer one or more Data Bean according to request parameter, and assemble them into View Bean and send it to Control lay. Data Beans describe and define the object models that are abstracted from reality. View Beans include page display information and user access information, and they are container of the data packages, which encapsulate the object instances and return to the client terminal. And a series of static SQL functions (Data Manager) is designed to access data and produce Data Beans.

Servlet implements control lay. When it receive a user request, servlet calls Handlers for offering a Handler that processes this request according to the parameter, and require the Handler process the request.

JSP pages implement View lay. Each JSP corresponds one View Bean.

5.3. System Processing

For example, user inputs order number and inquire customer information and payment of the order, the event triggers system to process. (these classes implement part of function of order request, Table 2)

(1) When Servlet receives the request from brows, it searches Handler in the Handlers list, which processes this request especially according to the parameter, and requires Handlers offer QueryClientHandler that processes customer information of order query and QueryPayAmntHandler that processes the payment query.

(2) QueryClientHandler and QueryPayAmntHandler deal with the information by business logic, which comes from Servlet. Firstly, in order to obtain the queried information, they transfer the SQL functions from Data Manager to access database, and then encapsulate the return recorders as object instance (ClientInfo and PayAmntData). Secondly, they encapsulate the Data Beans into QueryResultViewBean that has been defined already. After the process, Handlers return QueryResultViewBean to Servlet.

(3) Servlet transfers the request and returns parameters to QueryResultViewBean, and control the corresponding JSP to obtain all information from QueryResultViewBean. The JSP is returned to the terminal user as HTML page.
The process as Figure 3:

Table 3. System classes list (part)

Object name	Function
Handlers：	
.QueryAmntHandler	Query order quantity
QueryClientHandler.	Customer query
QueryPriceHandler.	Order price query
QueryPayAmntHandler.	Payment query
Data Beans：	
ClientInfo	Customer information
AmntData	Order quantity
PriceInfo	Price information
PayAmntData	Paid amount
View Beans	
OrderHomeViewBean	The content of order main page
OrderQueryViewBean	The content of order query page
QueryResultViewBean	The content of query result page

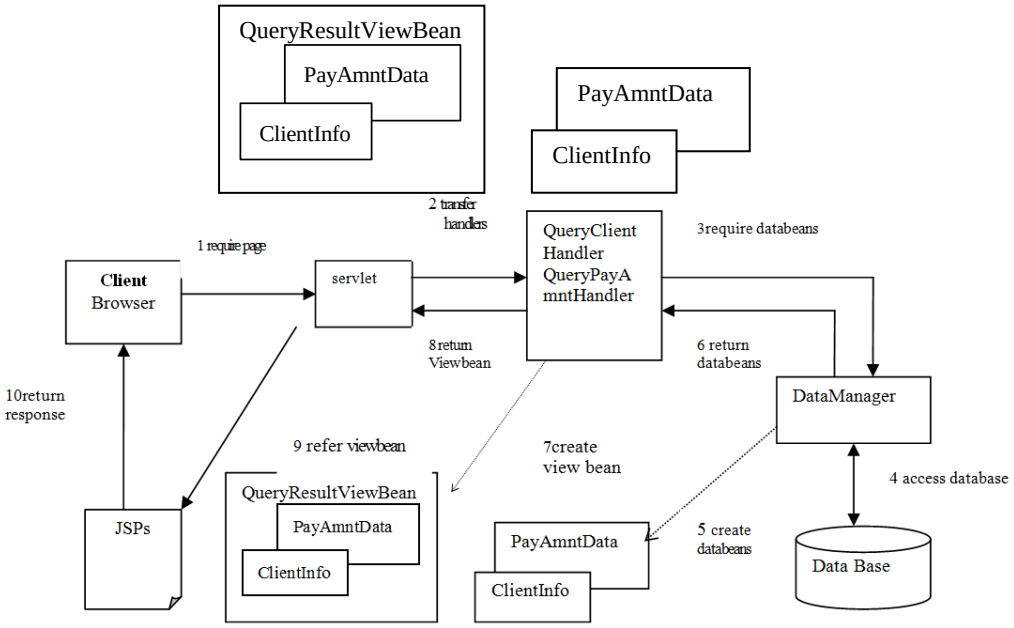

Figure 3. System Process Illustrations

5. Conclusion

The MVC design in the paper suits CRM system very well, as follows:

1. In the design, JSPs associate with servlet to implement dynamic content service: JSPs create expression content, and servlet completes task of deeper level. This method separates expression and content clearly. Data Manager accesses database and create Data Beans, which encapsulates the detail of data access and separates business and data level well. The extension and alternation of CRM system functions only reorganize the corresponding business logic without modifying the whole system. The design can reuse existing components well.

2. Each handler of the system only implements one function, all Handlers are controlled and transferred by servlets. The arrangement defines role authority of different users clearly in order that CRM system easy to support new customer styles.

3. Servlets can find corresponding JSP pages by the information about user's session that every View Bean takes. So system can display user's individual interface to be suitable for complexity and variety of data operation in the CRM system.

4. The system on the J2EE is easy to integrate with other system.

So the system design based on MVC fits the multi-user CRM system that is extensible, high interactive, and easy to maintain.

References

[1] J. Dyche, "The CRM Handbook: A Business Guide to Customer Relationship Management", **(2001)**.
[2] J. Fjermestad and N. C. Romano Jr., "Electronic customer relationship management: revisiting the general principles of usability and resistance: an integrative implementation framework", Business Process Management Journal, vol. 9, no. 5, **(2003)**, pp. 572-591.
[3] Y. Qi and W. Jia-yang, "The Design and Implementation of a Kind of Customer Relation Management Information System Based on J2EE Architecture", Application Research of Computers, vol. 11, **(2002)**, pp. 153-154
[4] C. Perng, S. -L. Wang and W. -C. Chiou, "A Conceptual Framework of Library Reader Service from Customer Relationship Management Perspective", International Journal of u - and e - Service, Science and Technology, vol. 2, no. 1, **(2009)**, pp. 11-20.
[5] P. Nagarajan and G. W. Jiji, "Online Educational System (e- learning)", Journal of u - and e - Service, Science and Technology, vol. 3, no. 4, **(2010)**, pp. 37-48.
[6] E. Asfoura, G. Kassem and R. Dumke, "Characterization of business model for federated ERP systems", Journal of u - and e - Service, Science and Technology, vol. 3, no. 4, **(2010)**, pp. 19-36.
[7] F. Buschmann, "Pattern-Oriented Software Architecture", **(1996)**.
[8] A. Leff and J. T. Rayfield, "Web-Application Development Using the Model/View/Controller Design Pattern", IEEE Enterprise Distributed Object Computing Conference, **(2001)** September, pp. 118-127.
[9] E. Gamma, R. Helm and R. Johnson, "Design Patterns: Elements of Reusable Object-Oriented Software", New York: Addison Wesley Professional, **(1995)**.

Author

Liancai Hao, PH. D. student of Harbin Institute of Technology.