A Database and Web Application Based on MVC Architecture

Diana M. Selfa, Maya Carrillo, Ma. del Rocio Boone
Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla
14 Sur y Av. San Claudio, Edif. 135, Ciudad Universitaria
dselfa@hotmail.com, mayacar@siu.buap.mx, rboone@cs.buap.mx

Abstract

MVC architecture has had wide acceptance for corporation software development. It plans to divide the system in three different layers that are in charge of interface control logic and data access, this facilitates the maintenance and evolution of systems according to the independence of the present classes in each layer. With the purpose of illustrating a successful application built under MVC, in this work we introduce different phases of analysis, design and implementation of a database and web application using UML. As central component of the application, it has a database made up by fifteen relations and a user interface supported by seventeen web pages.

Key words: MVC architecture, database systems, analysis and system design.

1. Introduction

In software development the modeling and the standardized notations allow to express complex ideas in a precise way, facilitating the communication among the project participants that generally have different technical and cultural knowledge.

A notation should have a good defined semantics and should be dominated by the participants so that it facilitates the communication. The established standards allow overcoming bad interpretations and ambiguity [1].

Systems analysis and design produce models built by means of diagrams that use a specific notation. UML is mainly a graphic language to represent the concepts that are needed in the development of an object-oriented information system.

Models help to understand and to represent the problem that is solved, as well as the problem solution. Different approaches exist to model a problem and to express the system requirements and restrictions. If the problem model is considerably far from the solution model effort will be invested in trying to transfer the problem representation from a users’ understandable form to a designers and developers understandable one. This causes misunderstandings and difficulty to validate the solution with regard to the established problem [6].

On the other hand a software engineer applies potential solutions to development problems, monitors their success or failure and produces more effective solutions on the next occasion.

Different developers may expend a great deal of development time and effort on solving these problems from first principles each time they occur, and the solution that each produce may not be the most appropriate that could be achieved. Patterns provide a means for capturing knowledge about successful solutions in the software development.

One of these patterns is Model View Controller (MVC) architecture initially introduced for user interfaces in application implemented with the programming language Smalltalk. In this approach the system is divided in three components: model that express the domain knowledge, view that present the user interface, and control that manage the updates to views.

MVC has demonstrated its benefits for interactive applications allowing multiple representations of the same information, promoting the code reutilización, and helping developers to concentrate on a single application aspect. [2]

It is common that information systems should be tailored to different user needs according to their role. This means that if several users need to access assets information it is probable that a person in charge of inventory needs to see the amount of equipments, their distribution and who preserves them. However the responsible for budget needs to know how much it has been spent year to date in the acquisition of new assets. This situation is translated in having different views for
the same information that will be shown according to the user. If a view is updated, the results of this update should be reflected in the remaining views. To think about having an architecture based on subsystems like for example equipment subsystem, locations subsystem, budget subsystem can be expensive since several elements can be duplicated, making the software more complex and prone to errors. The system maintenance also gets complicated so that any change in the functionality of the system has to be changed in the subsystems interfaces. So the problems to be resolved are:

1. The same information should be shown in different formats in different views
2. Changes in a view should be reflected in the remaining ones
3. Changes in the user interface should be easy to make
4. The central functionality of the system should be independent of the interface to allow that multiple interfaces coexist [1]

The MVC architecture solves the mentioned points separating the system functionality from the system interface and it incorporates a mechanism for propagating the changes to the remaining systems views. Then their advantages with relationship to other architectures are:

1. Less coupling
2. Higher cohesion
3. The views provide bigger flexibility and agility: multiple model views can be created, to add, to modify, and to eliminate new views dynamically, you can nest views, you can change the way in which a view responds to the user without changing its visual representation, to synchronize the views, to have views for different devices according to their capacities, a view for Web and another for windows applications
4. More design clarity
5. It facilitates the maintenance
6. Bigger scalability [5, 10]

The analysis of robustness introduced by Ivar Jacobson [7] can be easily moved to MVC architecture. In this approach the text of the use cases is used to identify the participant objects and they are classified according to the role that they play in the use case. This classification locates the objects in a natural way inside the components of MVC. Ivar classifies the objects in:

1. Entity objects: objects with long life times related with persistent data, they can be databases tables, files, and cache or session data (MVC model)
2. Boundary objects: objects that communicate to the system with their environment, they can be screens, windows, menus, or any graphic interface element (MVC view)
3. Control objects: objects that carry out the use case actions are used to filter out the data to be presented to the user (MVC control)

This analysis imposes the following rules:

1. Actors can only talk to boundary objects.
2. Boundary object can only talk to controllers and actors
3. Entity objects can only talk to controllers
4. Controllers can talk to boundary objects and entity objects, and to others controllers, but not to actors [11].

In this work, we present like case study a database and Web application (SAF) to control the assets information owned by an educational institution. Its modeling was made using UML (Unified Modeling Language), it was used analysis of robustness and MVC architecture.

In this paper we present MVC architecture. Section three describes SAF specification aspects. Sections four, five, and six describe SAF view, control, and model components respectively. Conclusions and perspectives of this work are in the last section.

2. MVC

The Model View Controller (MVC) paradigm was first designed for user interfaces in applications implemented with Smalltalk, but from then it has become a design paradigm for user interfaces without caring the implementation language and for web applications whose control components frequently change.

MVC architecture is illustrated in Figure 1, it divides the interactive system in three components each one specialized in a task. The model contains the application data and manages the core functionality. The view manages the visual display of the model and the feedback to the user. The controller interprets the mouse and keyboard inputs from the user, commanding the model and the view to change appropriately [3].
The pattern may be passive, which means that it does not know about the existence of the view or the controller. For example if the model is a text that can only be changed by the user. However in most of the cases the model must have a link to the view to inform it on changes made to its state caused by internal procedures. The view and the controller are always connected. The controller communicates with the view to determine which objects are being manipulated by the user and to call model methods to make changes on these objects. The model carries out the changes and it notifies the view to update.

The application view generally includes several nested MVC views. The controllers of these views should cooperate to assure that the appropriate control is interpreting the user entrance. For this purpose they form a hierarchical tree where the messages pass from controller to controller through the branches of the tree. Only the controller that has the focus takes an action.

Each view is associated with a unique controller and vice versa, but the model can have at the same time more than one view-controller pair. Every time that the model changes each dependent view must be notified so that they change accordingly. The possibility to have multiple synchronized views is a significant benefit of the MVC architecture [4].

3. SAF

We present the case study SAF an application to manage assets information for illustrating the use of the MVC architecture.

UML was used as modeling language, since it has become an industry standard, we use it to carry out the models of the system.

Trying to diminish the breach between the problem model and the solution model use cases were used for eliciting SAF requirements. Figure 2 shows the main use cases: 1. Validate access requests user and password to enter the system, 2. Manage assets information administers assets information, 3. Manage system catalogs administers information related to the assets: accounts, people responsible for assets, locations, assets classes, assets types, assets models, organizations among other, and 4. Generate reports presents reports by classification, location, asset type, people in charge of assets, serial number, inventory label and accounts as well as combinations of the listing attributes.

Once defined the use cases in detail we use the robustness analysis introduced by Ivar Jacobson [7].

Robustness analysis rules were applied in SAF analysis and design. Figure 3 shows a simplified diagram for Delete Account use case related to Manage System Catalogs by a relation << uses >>. The view is represented by an HTML page named Cuentas. When the user selects the delete option of the datagrid (grdCuentas) on the HTML page to eliminate an element a message is sent to PrvControl who sends the message prvBorrar to MdPrincipal. This last one send the message execute to the control SqlCliente that is the .NET framework Data provider [13] who by means of SAF_SP_CUENTA stored procedure erases the selected element.

Starting from the robustness diagrams classes, sequence, collaboration and states diagrams were built. SAF was implemented in ASP.NET [8] that is coupled to the MVC paradigm separating the interface (view) in a natural way in file *.aspx and code behind page (control) in files *.aspx.yy, where yy depends on the programming language, in our case it was Visual Basic .NET. SQL Server 2000 was used as relational database [13, 14] and Crystal Report to generate reports. The main functions of the system are:

![Figure 2. SAF main use cases](image)

![Figure 3. Conceptual robustness diagram for Delete Account use case](image)
1. Validate access, 2. Manage catalogs: assets type, accounts, classifications, locations, organizations, measure units, principals, people in charge of assets, acquisition types, marks, models, conservation states and users. 3. Manage assets 4. Generate reports, and 5. Maintain assets historical.

Figure 4 shows SAF distribution: an ASP pages server, a database server, and clients that will be able to use the application according to their authorization level using a browser.

4. View

The system interface consists of 17 web pages 13 of them for catalogs administration. They allow homogenizing the assets information. Figure 5 shows the page for administering account catalog. In each line of a datagrid you can see the registered accounts if X is selected the element of the line is deleted. If E is selected the key account and the description account are placed in the corresponding text box and you are allowed to modify them. If the new button is pressed the text boxes of key account and description account are activated to be edited with information of a new account. This information is save to the database when the accept button is pressed.

5. Control

Controls are constituted by the *.aspx.vb files one for each web pages developed, in them there are 12 methods on average that contain the logic for event handling and an additional control to communicate with the model and to returns data to page control. Like example part of the PrvControl class code is shown. This code allows eliminating a catalog account. It can be observed the call to the ejectua method of the model layer with SAF_SP_CUENTA a stored procedure as parameter.

```vbnet
Private Sub prvBorrar()
If Me.grdCuentas.SelectedIndex <> -1 Then
    Me.txtClave.Text = Me.grdCuentas.SelectedItem.Cells(1).Text
    Me.txtCuenta.Text = Me.grdCuentas.SelectedItem.Cells(2).Text
    If Ejecuta("SAF_SP_CUENTA 3, " &
        Me.grdCuentas.SelectedItem.Cells(0).Text & "," &
        Me.txtClave.Text & ","," &
        Me.txtCuenta.Text & ",") Then
        prvEstadoInicial()
    Else
        Me.lblResultados.Text = "Ha ocurrido un error al intentar..."
    End If
End If
End Sub
```

5. Model

In the model we find the class MdPrincipla with 3 main methods: ejecuta, consulta, and insertaConClave and 17 stored procedures. Next the ejecuta method code is shown. Ejecuta method was called by the Control using the prvBorrar method. It can be observed that until this layer the application makes use of specific classes to communicate with the selected database.

```vbnet
Public Function Ejecuta(ByVal vsInstruccion As String) As Boolean
    If gnNivelUsuario > 0 Then
        Try
            goConeccion = New SqlClient.SqlConnection("DATA
SOURCE=localhost;initial
catalog=saf;uid=saf;pwd=userSAF;")
       (goConeccion.Open()
            goComando = New
            SqlClient.SqlCommand(vsInstruccion, goConeccion)
        GoComando.ExecuteNonQuery()
            Ejecuta = True
        Catch e As System.Exception
            Return False
        Finally
            goComando = Nothing
            goConeccion.Close()
        End Try 'de Try
        Else
            Return False
        End If
    End Function
```

As it was previously mentioned in SQL Server 2000 17 store procedures were created to add, to update, to delete, and to search on the database tables. The store
procedure SAF_SP_CUENTA operates on the SAFCCUENTA table. It can be stood out that until this level a specific SQL syntax is used.

```
CREATE PROCEDURE SAF_SP_CUENTA
@VNOPCION TINYINT,
@VNCLAVEUNICA INTEGER,
@VNCLAVE INT,
@VSDESCR VARCHAR(20)
AS
IF (@VNOPCION = 1)
  INSERT INTO
    SAFCCUENTA(DNCLAVE, DSDESCR)
  VALUES
    (@VNCLAVE, @VSDESCR)
END
IF (@VNOPCION = 3) BEGIN
  DELETE
  SAFCCUENTA
  WHERE
  PNCLAVE = @VNCLAVEUNICA
END
GO
```

Some of our experiences when developing this system were that use case diagrams used to specify the system functionality from the user perspective together with robustness analysis are a powerful tool to track each requirement since its definition up to its implementation and tests. The use of programming standards to write code reflected in the names of web pages, variables, tables, stored procedures, triggers and methods gave as a result an understandable and simple code. Also the code organization in three layers has allowed other developers quickly understand its structure facilitating the changes implementation.

During application development we received several changes from the user; one of them was the possibility to have a historical registration for people in charge of an asset and asset characteristics at the assignment moment. To implement this requirement it was only necessary to modify the model and the view of the application. A trigger associated to SAFDBIEN assets table was added in the model to validate the moment when the person responsible of the asset, asset characteristics or asset state is updated to insert a new record into SAFDHIST historical table. Trigger code to insert a record into SAFDHIST when the asset characteristics are changed is shown:

```
CREATE TRIGGER SAF_TG_HISTORIAL ON
dbo.SAFDBIEN
AFTER UPDATE
AS
  IF EXISTS (SELECT * FROM INSERTED
              JOIN DELETED ON INSERTED.DSCARAC <>
                            DELETED.DSCARAC)
    INSERT INTO SAFDHIST
    SELECT INS.PNCLAVE, 1,
           INS.DSCARAC, INS.FNRESP,
           GETDATE(), INS.FNUSUARIO,
           INS.FNEDO
    FROM INSERTED INS
    JOIN DELETED ON
    INS.PNCLAVE = DELETED.PNCLAVE
```

Similar code was written to insert a record if the person in charge of the asset or the asset state is updated. The easiness to implement changes like the described has shown the advantages of MVC architecture.

To have a better idea about the objects in model layer, table 1 shows the system database tables and their description.

### 6. Conclusions

Through the development of the application that we have presented in this work, we confirm the advantages of MVC architecture for database and web application development. It was also evident that the construction of clear and understandable models for the users and software engineers allows the construction of quality systems with predictable development times. With regard to the case study presented in this work we will incorporate new components to allow the capture of computer assets in an automatic way to feed and update the database, this means that using the network infrastructure computer hardware and software characteristics will be captured. These characteristics will be used to obtain performance and space distribution graphics.
Table 1. SAF database tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFCCLAS</td>
<td>Asset Classification</td>
</tr>
<tr>
<td>SAFCCUENTA</td>
<td>Accounts</td>
</tr>
<tr>
<td>SAFCDIR</td>
<td>Principals</td>
</tr>
<tr>
<td>SAFCEDO</td>
<td>Asset State</td>
</tr>
<tr>
<td>SAFCMARCA</td>
<td>Brands</td>
</tr>
<tr>
<td>SAFCMOD</td>
<td>Models</td>
</tr>
<tr>
<td>SAFCORG</td>
<td>Organizations</td>
</tr>
<tr>
<td>SAFCRESP</td>
<td>People in charge of assets</td>
</tr>
<tr>
<td>SAFCTIPO</td>
<td>Asset Type</td>
</tr>
<tr>
<td>SAFCTIPOAD</td>
<td>Acquisition Type</td>
</tr>
<tr>
<td>SAFCUBI</td>
<td>Locations</td>
</tr>
<tr>
<td>SAFCUNIDAD</td>
<td>Departments</td>
</tr>
<tr>
<td>SAFCUSU</td>
<td>Users</td>
</tr>
<tr>
<td>SAFDBIEN</td>
<td>Assets</td>
</tr>
<tr>
<td>SAFDHIST</td>
<td>Asset historical</td>
</tr>
</tbody>
</table>

7. References

[2] Colin Moock, Object-Oriented Development with ActionScript 2.0, O’Reilly
[4] Hanna Dobos, Separable User Interfaces and Interaction Controls, Master’s Thesis, University of Jyväskylä, Department of Mathematical Information Technology, 2002