CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 INTRODUCTION

System implementation is a process of developing a system based on the requirements. In a software project, the requirements of analysis, system design and implementation phases do not have a clear boundary. Each phase tends to overlap one another. This phase at times involves some modification to the previous design in the requirement documentation. System implementation is well defined as the integration of the physical, conceptual and constructed resources that produce a working system. Therefore, system implementation is the physical realization of the database and application designs. In the system implementation, a database will be created while the source code will be written, which will then produce and deliver a functional system.

The major works in system implementation are the construction of the system, installation of the constructed system as well as the conversion of the system. The purpose of the system construction is to develop a functional system that fulfills the design requirements. The system construction of this project involves system coding. Coding is generally recognized as a major aspect of the system construction and it is a process of turning
program logic into specific instructions that the computer system could execute. In doing so, programming languages are used to transform program logic into code statements. The given algorithm, coding and description were the implementation for functions discussed in Chapter 4 (Requirement Analysis).

6.2 DEVELOPMENT APPROACHES

T-UCD is developed modularly using the top-down approach. Top-down approach is a design method in which the modules to be accomplished is broken down into sub modules and each sub modules is further decomposed into smaller sub modules and so forth. This approach is a good way of constructing a program because the sub program is small enough to be written in functions. An advantage of using this approach is that important modules are ensured to be developed and tested.

Developing T-UCD with top-down approach involves building the high-level software modules that are refined into functions and procedures. The advantages of using this approach in T-UCD are:

- Testing can begin on some of the modules while others are still being coded.
- Critical functions can be coded first to test their efficiency.
- Code is easier to follow, since it is written methodically and with purpose.

By using this approach, the first technique for writing a T-UCD program is writing a main procedure that names all the needed major functions. Later, the requirements of each of those functions are looked and the process is repeated. These compartmentalized sub-
routines eventually will perform actions so simple they can be easily and concisely coded.

When all the various sub-routines have been coded, the program is done. By defining how the application comes together at a high level, lower level work can be self-contained. By defining how the lower level objects are expected to integrate into a higher level project, interface become clearly defined.

6.3 MODULE DESIGN

In this section the coding for each module for T-UCD and the interfaces related to the coding will be discussed in detail. The overall system can be separated into several modules. The overall modules of T-UCD are as follows:

i. Notes Module  
ii. Case Study Module  
iii. Elements Module  
iv. Relations Module  
v. Diagram Module

6.3.1 Notes Module

This module consists of notes for the user to view before they begin with the tutorial to draw a use case diagram. The included notes are on UML, Use Case Diagram, Actor, Use Case, Association, Relationship and System Boundary. Guideline in drawing use case diagram is also available. Under the Guideline section, detailed explanation on how to draw a correct use case diagram is given. User is allowed to view the notes by clicking on the target notes.
To develop the Notes interface, the contents of the notes is initially saved in the .html format before they recalled in Visual basic.Net using the AxWebBrowser function. The coding is shown in Figure 6.2.
Figure 6.2: Coding for select notes

6.3.2 Case Study Module

This page contains two text boxes. The smaller text box allows the user to key-in the name of the system or the case study. This name will appear on the use case diagram generated by the system at the Diagram page. The bigger text box allows the user to enter the case study. After a case study is entered, T-UCD will identify the actors and use cases in the given case study and these two elements will be highlighted by different color scheme. Figure 6.3 shows the Case Study interface.
In order to implement the function of identifying actors and use cases, firstly, the name of actors are saved as noun and use cases as verb in the database. Once the user enters the case study, the system will match the entered data with the list of actors and use cases in the database.

As mentioned in Chapter 4 (Requirement Analysis), T-UCD provides interactive functions for the user. By applying this function, T-UCD will highlight the identified actors as red in color and identified use cases as blue in color. The Case Study module is useful for users who have limited knowledge in specifying and classifying use case elements. The coding of this function is shown in Figure 6.4.

**Figure 6.3:** Case Study interface
6.3.3 Elements Module

The Elements page is used to list down the actors and the use cases that have been identified in the case study entered by user in the previous Case Study page. In order to implement this page, T-UCD applies the user friendly function which has been explained in non-functional requirement section (Chapter 4 – Requirement Analysis). As a result of applying this function, the user is allowed to add or delete the elements by using the “Add” or “Delete” buttons. Besides, a step-by-step description in doing a tutorial is provided for

Figure 6.4: Coding for identify elements from Case Study
the users as a guide for them to complete a tutorial in T-UCD. Figure 6.5 shows the Elements interface.

![Elements Interface](image)

**Figure 6.5: Elements interface**

All actors and use cases which were identified from the Case Study page will be listed down in the list boxes. In this case, the system will copy the blue and red text into the database. The coding for drop down list for the elements is shown in Figure 6.6.
Figure 6.6: Coding for list down the elements

Figure 6.5 shows that only verbs are displayed in the use case list. This is because the use case names have been set as verb in database. By applying the user friendly function, the Elements page allows the user to edit the elements name. Figure 6.7 shows the Elements page after a user performs the editing to the elements.

Figure 6.7: Elements interface where editing for the elements have been made by user
Users are able to edit the elements by using the “Add” and “Delete’ button. Once the user adds or deletes an element, data of the particular element in database will also changed. In this case, the latest elements entered by users are saved in the database. The coding for add and delete the elements is shown in Figure 6.8 and Figure 6.9.

(i) Add Elements

```vbscript
Private Function AddAct()
    If txtActor.Text = "" Then
        MessageBox.Show("Please put an item in the form", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxIcon.DefaultButton.Button1)
        GoTo stopp
    End If
    currentID = CreateNewRecordActor()
    UpdateRecordActor(currentID, txtActor.Text)
    Dim myRS As New ADODB.Recordset
    myRS.Open("Select Actor from Actor", myDB)
    lstActor.Items.Clear()
    While Not myRS.EOF
        lstActor.Items.Add(myRS.Fields("Actor").Value)
        myRS.MoveNext()
    End While
    myRS.Close()
    txtActor.Text = ""
   (txtActor.Focus())
    stopp:
    End Function

Private Function AddUse()
    If txtUseCase.Text = "" Then
        MessageBox.Show("Please put an item in the form", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information, MessageBoxIcon.DefaultButton.Button1)
        GoTo stopp
    End If
    currentID = CreateNewRecordUseCase()
    UpdateRecordUseCase(currentID, txtUseCase.Text)
    Dim myRS As New ADODB.Recordset
    myRS.Open("Select UseCase from UseCase", myDB)
    lstUseCase.Items.Clear()
    While Not myRS.EOF
        lstUseCase.Items.Add(myRS.Fields("UseCase").Value)
        myRS.MoveNext()
    End While
    myRS.Close()
    txtUseCase.Text = ""
    txtUseCase.Focus()
    stopp:
    End Function
```

Figure 6.8: Coding for add elements
(ii) Delete Elements

```vbnet
Private Function DelAct()
' check if something is selected in the list
If lstActor.SelectedIndex <> -1 Then
    'delete the record with current ID
    DeleteRecordActor(currentID)
    Dim myRS As New ADODB.Recordset
    ' match the selected name to one in the database
    myRS.Open("SELECT Actor from Actor", myDB)
    ' save the ID in case we want to update
    'currentID = myRS.Fields("ID").Value
    ' clear old results
    lstActor.Items.Clear()
    ' place results in the list
    While Not myRS.EOF
        lstActor.Items.Add(myRS.Fields("Actor").Value)
        ' go to next record in the results
        myRS.MoveNext()
    End While
    myRS.Close()
End If
End Function
```

```vbnet
Private Function DelUse()
' check if something is selected in the list
If lstUseCase.SelectedIndex <> -1 Then
    'delete the record with current ID
    DeleteRecordUseCase(currentID)
    Dim myRS As New ADODB.Recordset
    ' match the selected name to one in the database
    myRS.Open("SELECT UseCase from UseCase", myDB)
    ' save the ID in case we want to update
    'currentID = myRS.Fields("ID").Value
    ' clear old results
    lstUseCase.Items.Clear()
    ' place results in the list
    While Not myRS.EOF
        lstUseCase.Items.Add(myRS.Fields("UseCase").Value)
        ' go to next record in the results
        myRS.MoveNext()
    End While
    myRS.Close()
End If
End Function
```

**Figure 6.9:** Coding for delete elements
6.3.4 Relations Module

Figure 6.10: Relations interface

Figure 6.10 shows a Relation interface. The Relations page allows the user to select the elements to be related in the Diagram page. Users are also needed to select the relationship that should be included in the use case diagram. On this page, T-UCD applies a simple function for the user by providing all identified actors and use cases names in the drop down list boxes. Furthermore, suitable relationships for the selected elements are also listed in the drop down list box. The user is only required to select and add the necessary relations. Relations that have been selected by the user will be listed down in the column and will be saved in the database. Relationship drop down list box will only activate if the user selects the relation between actor and actor or use case and use case. The coding for the Relations interface is shown in Figure 6.11 to Figure 6.13.
(i) Select Elements

Private Sub comRel31_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles comRel31.SelectedIndexChanged
    comRel41.Items.Clear()
    If comRel31.Text = "Actor" Then
        lblRel41.Text = "Actor name"
    Else
        lblRel41.Text = "Use case name"
    End If
    If comRel11.Text = "Use case" And comRel31.Text = "Use case" Then
        comRel51.Items.Clear()
        comRel51.Items.Add("<<extend>>")
        comRel51.Items.Add("<<include>>")
        comRel51.Text = "<<extend>>"
        comRel51.Enabled = True
    ElseIf comRel11.Text = "Actor" And comRel31.Text = "Actor" Then
        comRel51.Items.Clear()
        comRel51.Text = "generalization"
        comRel51.Items.Add("generalization")
        comRel51.Enabled = True
    Else
        comRel51.Enabled = False
    End If
End Sub

Figure 6.11: Coding for select elements

(ii) Add Relation

Private Function CreateNewRecordRelation() As Integer
    Dim myRs As New ADODB.Recordset
    myRs.Open("Relation", myDB, ADODB.CursorTypeEnum.adOpenKeyset, ADODB.LockTypeEnum.adLockOptimistic)
    ' add a new empty record
    myRs.AddNew()
    ' update the recordset to include the new entry
    myRs.Update()
    myRs.Requery()
    myRs.MoveLast()
    CreateNewRecordRelation = myRs.Fields("RelID").Value
    myRs.Close()
End Function

Figure 6.12: Coding for add relation
(iii) Delete Relation

Once the users delete the relation, record of the relation is deleted from the relation database.

Public Function DeleteRecordRelation(ByVal currentID As Integer)
    ' delete a record from the database
    ' create a string variable that will hold an SQL statement
    Dim mySQL As String

    Dim item As String = CStr(ListRelID.SelectedItem)
    mySQL = "Delete * from Relation where RelID='" & item & "'

    ' execute the SQL command
    myDB.Execute(mySQL)
End Function

Private Function delrel()
    Dim index As Integer
    Dim item As String = CStr(ListRelID.SelectedItem)
    Dim mySQL As String

    Try
        ' assign index from selected item
        index = ListBox1.SelectedIndex()

        ' clear selected item
        ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)

        mySQL = "Delete * from relation where RelID= " & item & ""
        ' run SQL command
        myDB.Execute(mySQL)
        ' remove selected item
        ListRelID.Items.RemoveAt(index)

        'DeleteRecordRelation(currentID)
        Catch ex As Exception
    End Try
End Function

Figure 6.13: Coding for delete relation

168
6.3.5 Diagram Module

The automatically generate diagram will be displayed in the Diagram page. The diagram is generated based on the case study entered by the user in the Case Study page as well as the selected relation in the Relation page. In order to provide a clarity concept, the generated diagrams provided by T-UCD are apparent for the users to view. If the users are not satisfied with the arrangement of the elements, users are able to adjust the diagram by dragging it around. Figure 6.14 is shows an interface of the Diagram module.

![Diagram Interface](image)

**Figure 6.14:** Diagram interface

The diagram has been set in coding by using a coordinate. The coding for drawing a use case diagram is shown in Figure 6.15 to Figure 6.17.
(i) **Draw Actor**

```vbnet
Dim body, lhand, rhand, lfoot, rfoot As Point
Dim rec, rec1 As Rectangle
Dim int As Integer

g.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
rec = New Rectangle(c.X - 13, c.Y - 26, 26, 26)
rec1 = New Rectangle(c.X - 23, c.Y - 30, 46, 90)
g.FillRectangle(New SolidBrush(Color.AliceBlue), rec1)
int = str.Length

g.DrawString(str, dFont, New SolidBrush(Color.Black),
             c.X - int * 3, c.Y + 47)

body.X = c.X
body.Y = c.Y + 25
lhand.X = c.X - 15
lhand.Y = c.Y + 15
rhand.X = c.X + 15
rhand.Y = c.Y + 15
lfoot.X = c.X - 15
lfoot.Y = c.Y + 45
rfoot.X = c.X + 15
rfoot.Y = c.Y + 45
```

**Figure 6.15:** Coding for draw the actor

(ii) **Draw Use Case**

```vbnet
Public Function usecasepic(ByVal c As Point, ByVal str As String)
    Dim rec As Rectangle
    Dim int As Integer
    int = str.Length
    g.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

    rec = New Rectangle(c.X - 75, c.Y - 40, 150, 80)

    g.FillEllipse(New SolidBrush(Color.White), rec)
g.DrawEllipse(p, rec)
g.DrawString(str, dFont, New SolidBrush(Color.Black),
             c.X - int * 3, c.Y - 12)
End Function
```

**Figure 6.16:** Coding for draw the use case
(iii) Draw Line

Public Function drawline(ByVal pfirst As Point, ByVal plast As Point, ByVal str As String)
    Dim pos As Point
    Dim p1 As New Pen(Color.Blue, 2)
    Dim p2 As New Pen(Color.Black, 2)
    g.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
    pos.X = (pfirst.X + plast.X) / 2
    pos.Y = (pfirst.Y + plast.Y) / 2
    p1.StartCap = Drawing2D.LineCap.ArrowAnchor
    p1.EndCap = Drawing2D.LineCap.ArrowAnchor
    g.DrawString(str, dFont, New SolidBrush(Color.Black), pos.X, pos.Y)
    Select Case str.ToUpper
        Case "<<EXTEND>>"
            p2.DashStyle = Drawing2D.DashStyle.Dash
            g.DrawLine(p2, pfirst, plast)
        Case "<<INCLUDE>>"
            p2.DashStyle = Drawing2D.DashStyle.Dash
            g.DrawLine(p2, pfirst, plast)
        Case "GENERALIZATION"
            p1.DashStyle = Drawing2D.DashStyle.Dash
            g.DrawLine(p1, pfirst, plast)
        Case ""
            g.DrawLine(p, pfirst, plast)
    End Select
End Function

Figure 6.17: Coding for draw the line
6.4 SUMMARY

System implementation and coding is a step to turn the designed system process flow and data flow to a reality system. In order to change from design to reality, the algorithm must be well defined to convert the data flow diagram into a text format. Based on the algorithm, the codes have been written using Visual Basic .Net programming language.