CHAPTER 1

Introduction

1.1 Introduction

The main part of this research focuses on the stated problem of approaching dialog boxes when SW keyboard in UI is shown and hidden .As we all know that technology has witnessed great a advancement in the last decade . Smart phones for example ,they are very popular and now they are used for different purposes other than just making and receiving calls. In fact, smart phones are equipped with so many features and applications such as game applications ,wireless and Wi-Fi network connections and many more interactive applications. This requires more space for interaction on mobile screens and has become a real concern for IT experts worldwide. Due to the wide variety of inputs, small screens tend to interrupt tasks. Finally, mobile user interfaces present big challenges that are not present when using desktop systems.

Since designing mobile interfaces is a relatively new practice, but it is actually making progress and gaining popularity nowadays. This fact is urging manufactures and experts to find more effective and flexible tools to overcome many problems that might occur when designing user interface for modern mobile phones. We must be aware that mobile phones have built-in sensitivity and a wide variety of options and functions .One of the problems the designers encounter in the mobile phone technology is the range of variation of the capabilities among devices and platforms. For example, a specific type of phone can have a touch screen and no buttons, but with a virtual keypad; however another brand can have a number pad and a few arrows. These variances are equally different when it comes to computing power, screen capabilities, and more. To accommodate these differences, it likely requires building multiple designs for each platform which the interface needs to run on [1].

In this research work, we are using Model View Controller (MVC) architecture pattern for building a user interface for mobile devices , which is a fast and efficient way of creating different UI for mobile, This pattern is used as a methodology for designing the mobile user interface [21].

Generally, design patterns can help solve complex design problems if they are properly used, however the main advantage of using the MVC pattern is decoupling the business and the presentation layers [23].

MVC is defined as a common design pattern to integrate a user interface with the application domain logic. MVC separates the representation of the application domain (Model) from the display of the application's state (View) and user interaction control (Controller). The MVC design pattern is comprised of three major components[23] the Model (The Data Layer), the View (The User Interface Layer) and The Controller (The Business Logic Layer).

1.2 The Problem Statement

The main problem in this research is how to resize Text Box to avoid making some parts of these text dialog boxes invisible. Text Box or Dialog Box is defined as any message/text created or looking up a contact on the Contact Log saved in your phone. Therefore ,resizing practically prevents hiding some parts of the dialogs which eventually become invisible once a long text is being entered. Also, the severity of this problem largely depends on the type and style of user interface being used. To conclude, smart phones usually have small screens which do not allow for enough space which means less interaction for users when entering longer texts.

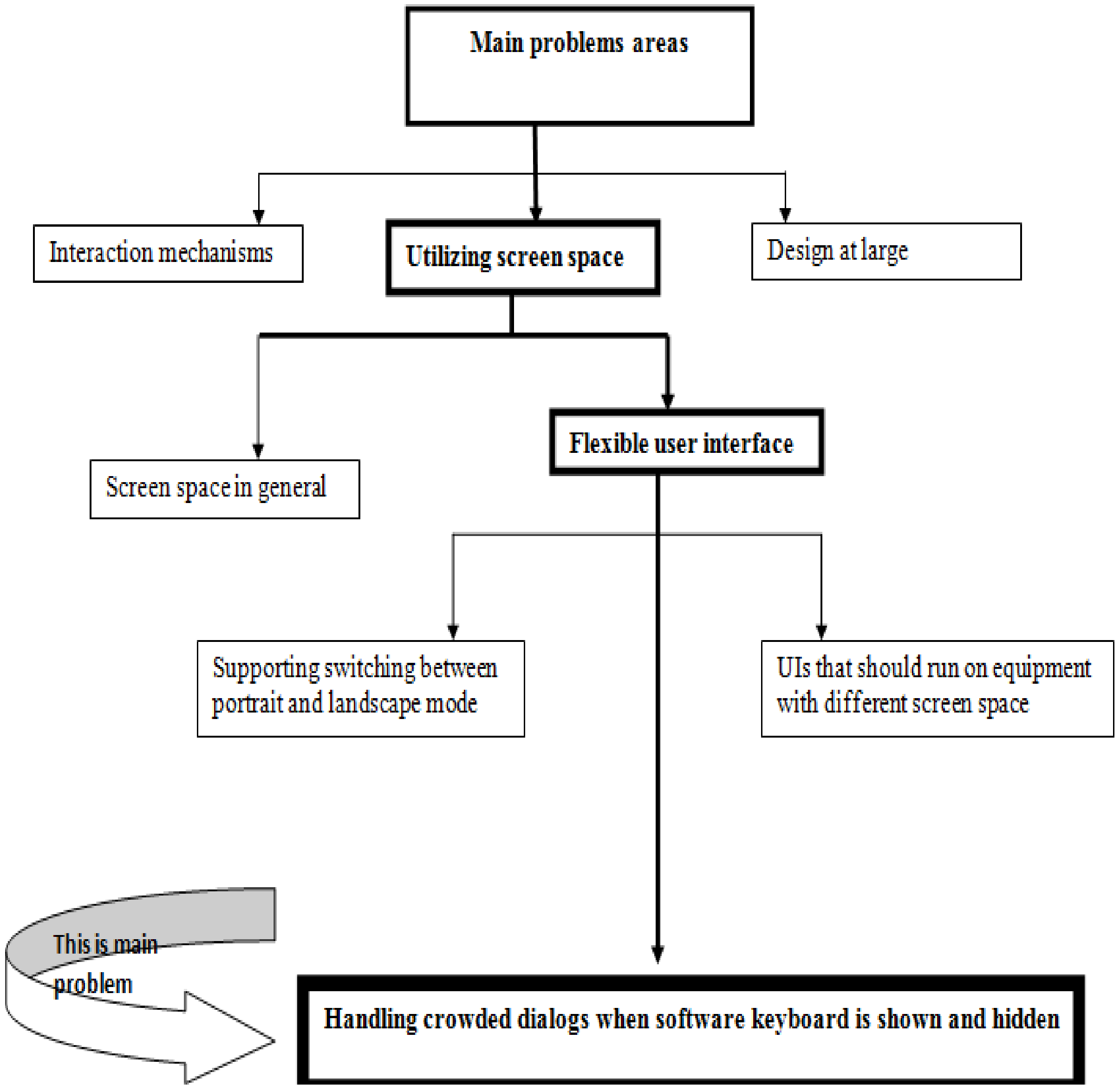
Main problems areas
Interaction mechanisms
Utilizing screen space
Design at large
Screen space in general
Flexible user interface
Supporting switching between portrait and landscape mode
UIs that should run on equipment with different screen space
This is main problem
Handling crowded dialogs when software keyboard is shown and hidden

Figure 1.1: The complexity of the problem statement adopted from [19]

1.3 Motivations and Objectives

1. The main objective of this study is to use an applicable and effective design pattern for better and more convenient use of screen space of smart mobile phones.
2. Design a user interface which can help users access the system easily and enjoy better and more interaction.
3. Understanding and analyzing the theory and practice behind using design patterns as solutions for UI problems in mobile applications.
4. Have an opportunity to see how we can encounter the challenges of modern technology.
5. Reassess and reuse the proposed design patterns such as Model View Controller which is believed to be one of the most popular among UI designers.

1.4 Solution Approach

The steps of the proposed solution in this research work can be summarized as follows:

1. Provide a clear and specific definition of the problem related to UI of mobile phone devices.
2. Present a description of the system requirements and specify an effective method to be used at the designing stage.
3. Illustrate in detail the tool to be used for designing the user to display information on the screen of the mobile device.
4. Determine a suitable design pattern to solve the stated problem of mobile UI. This is has to be done by providing clear definitions and explanation of these design patterns.
5. Choose a corresponding programming language such as J2EE or ASP.NET.
6. Finally, present a case study to examine how the proposed approach works when applied.

1.5 Structure of The Thesis

The remaining chapters of this thesis are organized as follows:

Chapter 1: Introduction

This chapter gives a short overview about design patterns , Model View Controller (MVC) and it also includes the motivation , objectives and the proposed solution approach.

Chapter 2:Background

This chapter presents a background and a general overview of Design patterns including families of MVC design pattern. It also introduces the ICONIX Methodology and the literature survey that are related to the proposed approach.

Chapter 3: The Proposed Solution for Designing a Mobile User Interface Using MVC Design Pattern

This chapter presents the proposed approach for solving the problem. It also explains in detail all the steps to be taken when using MVC design pattern and ICONX Methodology.

Chapter 4: implementation

This chapter presents the actual application of the proposed approach and the steps involved . it aims at solving the stated problem by designing a prototype with ASP.NET programming language.

Chapter 5: Conclusion

Presents findings, results and future work.

CHAPTER 2

Background

2.1 Introduction

This chapter is organized into five sections. In **section 2.2** presents an overview of design patterns , the definition of the essential elements of patterns , **Section 2.3** introduces the families of Model View * (MV*) design pattern .**Section 2.4** presents ICONIX Methodology and finally **Section 2.5** includes the literature survey that is related to the current research work.

2.2 Overview of Design Pattern

Design Patterns were first introduced as a defined concept by **Christopher Alexander** in his two books "A Pattern Language" in 1977, and "The Timeless Way of Building" in 1979 (**Seffah** 2010). **Alexander** is an architect who envisioned a way to capture all of the best aspects of architectural design in an easy-to-understand collection of what he termed "Patterns." Doing so enables engineers, architects, and even the laymen who would be using the buildings to communicate design ideas easily, and understand the problems facing each design [16].

Ultimately, **Alexander** wanted his pattern library to be used to help improve the quality of life for the people who would be living in or using the buildings. He hoped to capture what he refers to as the quality without a name, which he defined as follows: "there is a central quality which is the root criterion of life and spirit in a man, a town, a building, a wilderness. This quality is objective and precise, but it cannot be named" (**Wania** and **Atwood** 2009). Effectively, what this quality describes is those thoughtful designs that, whether obvious or subtle, make inhabiting a space a more pleasant, usable, or relaxing experience. What makes this emphasis so important is where the focus of the design is placed.

The first appearance of patterns at a conference was in 1997 at the CHI conference, sponsored by ACM SIGCHI. Since then, there have been numerous published articles and books furthering the discussion, refining the definition of what a user interface pattern consists of, and introducing new patterns to the community. In 2010, patterns were given center stage with the PEICS conference, which focused explicitly on issues surrounding designing and engineering user interfaces using design patterns. Amongst the things that have been discussed were how to define and structure patterns, and how to take the patterns and implement them in an automated fashion.

Design patterns represent a highly effective way to improve the quality of software engineering. Due to its ability to a capture the best practices and design knowledge based on real experience of software design, making it available to all software engineers [2]. It presents a generic proven solution to a common recurring design problem.

A design pattern in software engineering is a general repeatable solution to a commonly occurring problem in software design [3]. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Many patterns imply object-orientation or more generally mutable state, and so may not be as applicable in functional programming languages, in which data is immutable or treated as such.

A design pattern in architecture and computer science is a formal way of documenting a solution to a design problem in a particular field of expertise. The idea was introduced by the architect Christopher Alexander in the field of architecture[4] and has been adapted for various other disciplines, including computer science. An organized collection of design patterns that relate to a particular field is called a pattern language .The elements of this language are entities called patterns.

The patterns demonstrated Design Patterns focused around two keys attributes; they had to be reusable and they had to be flexible.

2.2.1 Design Patterns Definition

Christopher Alexander in (1977) defined the patterns as follows:

- Each pattern describes a problem that occurs repeatedly in our environment, and then presents the core of the solution to that problem in a way that you can reuse this solution a million times , without ever doing it the same way twice. [5]
- Design patterns gained popularity in Software engineering by the Gang of Four (GOF) book (1995).
- The design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context [6].

There are four essential elements of design patterns according to the GOF's structure.[7,16]:

1. **Pattern name**: The name of the pattern is the identity or the description of the design problem, the solutions and the consequences in a word or two. Giving names to patterns allows us to get a higher level of abstraction.
2. **Problem**: describes when the pattern should be applied including its context and what it solves.
3. **Solution**: describes the elements of the design, the responsibilities and collaborations. It does not describe a concrete solution. It is simply a template or a package that can be applied in different contexts.

4. **Consequences**: are the results expected after applying the pattern to solve problems. They expose the advantages and disadvantages of the solution proposed. They also include the impact on flexibility, extensibility and portability of the system.

2.2.2 Design Pattern Catalog

Design Patterns are described in graphical notation with Unified Modeling Language (UML) diagrams which capture the end product of design processes[34]. To reuse the design, record the designs, alternatives by describing them with a pattern name and classification, intent, motivation, applicability, structures, participants, collaboration, implementation and their uses in real systems. Design Patterns vary in their granularity and level of abstraction. Creational design Patterns concern the process of object creation, Structural patterns deal with composition of classes or objects while Behavioral patterns deal with the way in which classes or objects intent and distribute responsibilities [26].

The Gang of Four (GOF) patterns are ultimately considered as the foundation for all other patterns [8]. They are categorized in three groups: Creational, Structural, and Behavioral.

1- Creational Patterns

Creational design patterns deal with object creation mechanisms and to try create objects in a suitable manner to the situation. The basic form of object creation could result in design problems and increase the complexity to the design [35].

Recommend the way that objects should be created. In fact, these patterns are used when a decision must be made at the time a class is instantiated.

- **Abstract Factory**: Creates an instance of several families of classes.

- **Builder**: Separates object construction from its representation.
- **Factory Method**: Creates an instance o090f several derived classes.
- **Prototype**: A fully initialized instance -to be copied or cloned.
- **Singleton**: A class of which only a single instance can exist.

2- Structural patterns

Structural design patterns basically eases the designing process by identifying a simple way to realize relationships between entities [35].

These patterns are concerned with how classes inherit from each other or how they are composed from other classes[8].

- **Adapter:** matches interfaces of different classes.
- **Bridge**: separates an interface of an object from its implementation.
- **Composite**: a tree structure of simple and composite objects.
- **Decorator**: add responsibilities of objects dynamically .
- **Facade**: a single class that represents an entire subsystem.
- **Flyweight**: a fine-grained instance used for efficient sharing.
- **Proxy**: an object representing another object.

3- Behavioral patterns

Behavioral design patterns identify common communication patterns between objects and realize these patterns. These patterns increase flexibility in the communication among objects [35].

They also prescribe the way objects interact with each other. They help make complex behaviors manageable by specifying the responsibilities of objects and the way they communicate with each other.

- **Chain of Resp:** a way of passing a request between a chain of objects.
- **Command:** encapsulate a command request as an object.
- **Interpreter:** a way to include language elements in a program.
- **Iterator**: sequentially access the elements of a collection.
- **Mediator**: defines simplified communication between classes.
- **Memento**: captures and restores the internal state of an object.
- **Observer** : a way of notifying change/s to a number of classes.
- **State** :alters the behavior of an object when its state changes.
- **Strategy** : encapsulates an algorithm inside a class.
- **Template Method**: defers the exact steps of an algorithm to a subclass.
- **Visitor**: defines a new operation to a class without change.

The table below shows a description of the design pattern .

Creational Design Patterns		**Structural Design Patterns**		**Behavioral Design Patterns**	
Class	Object	Class	Object	Class	Object
Factory Method	Abstract Factory Builder Prototype Singleton	Adapter	Bridge Composite Decorator Façade Flyweight Proxy	Template Method Interpreter	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Table 2.1: Description of design pattern [26].

2.2.3 Essential Elements of Pattern

A pattern is considered as a basic tool of contact between designers, so describing patterns becomes a very important issue. The effectiveness of this contact leads to the usability and presentation of the template patterns. Also, the patterns are always documented in a template format, making the readers understand patterns easily [27].

The template below describes elements of a pattern template. The template does not display any details about the solution, but only describes information about the template and template implementation in general , which is dependent on the used programming language [27].

Content	Description
Pattern name	Describes the identity
Intent	Describes what the pattern does
Also known as	The list any synonyms for the pattern
Motivation	Motivation provides an example of a problem and how the pattern solves that problem
Applicability	Lists the situation where the pattern is applicable
Structure	Set of diagrams of the classes and objects related to the pattern
Participants	Describes the classes and objects their responsibilities that participate in the design pattern
Collaborations	Describes how the participants collaborate to carry out their responsibilities
Consequences	Describes the forces that exist with the pattern ,the benefits, and the variable that is isolated by the pattern

Table 2.2: Template of design patterns [27]

2.2.4 Benefits of Design Patterns

As we know that design patterns have many benefits. The following states the main benefits of design patterns. [9]:

1- They can speed up the development process by providing tested, proven development paradigms.

2- Effective software design requires considering issues that may not become visible until later in the implementation**.**

3- Reusing design patterns helps prevent subtle issues that can cause major problems. Therefore, design patterns improve code readability for coders and architects familiar with such patterns.

4- Design patterns provide general solutions, documented in a format that does not seek specific requirements to tackle a particular problem.

5- - Common design patterns can be improved over time, making them more robust than ad-hoc designs

2.3 Families of Model-View-*(MV*) Design Patterns

The main goal of this section is to introduce the different families of MV* design patterns and illustrate any differences between and also explore the various patterns within those families.

Generally speaking, MV* design patterns provide applicable and reusable solutions to the frequently emerging problem of synchronizing user interfaces with domain data, as in Widget-based user interfaces (which is not strictly a MV* pattern family).They are easy to implement for simple applications. However, as the approach does not separate domain and interface concerns, maintainability may become problematic when the complexity of the user interface grows [29].

MV* patterns are in fact, classified in three main categories: Model View Controller (**MVC**), Model-View-Presenter (**MVP**)[30] and Model-View-View Model (or Presentation Model) (**MVVM**). [31].Moreover with the ongoing evolution of programming languages and software technology, the MV* design patterns were changing too, and different families of patterns were born.

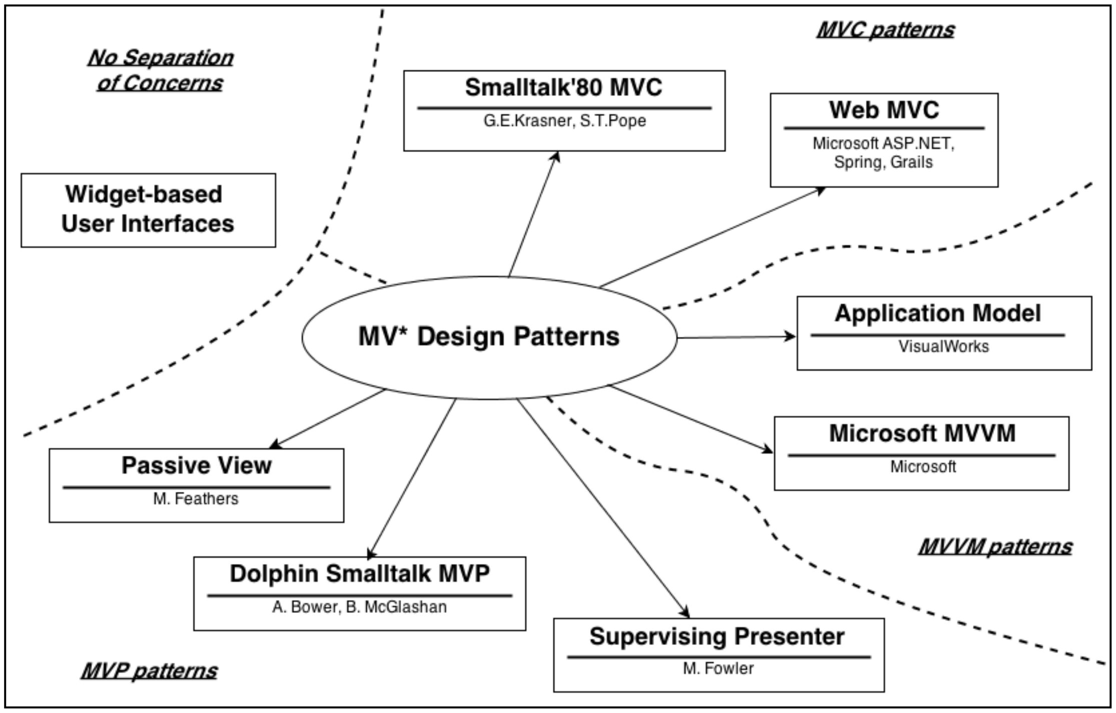

Figure 2.1: The Land of MV* Design Patterns [29]

2.3.1 Model View Controller (MVC) Design Pattern:

The term MVC has been in use since the late 1970s. It is made from Smalltalk, which is a programming language that was particularly designed to support the concepts of object-oriented programming. In the early 1970's, Alan Kay led a team of researchers at Xerox to invent a language that let programmers modify the data objects they intended to manipulate[45].

Model View Controller design pattern is also an architectural design that helps in making a user interface mobile applications modify-able with future requirements by splitting the whole application into three components, model, view and controller [42]. Each of these components handles discrete set of tasks.

- **Model:** is the core of the application. This maintains the state and data that the application represents. When significant changes occur in the model, it updates all of its views
- **Controller:** basically takes the role of a vocal point between the model and the view.
- **View:** The user interface which displays information about the **Model** to the user. Any object that needs information about the **Model** needs to be a registered **View** with the **Model**.

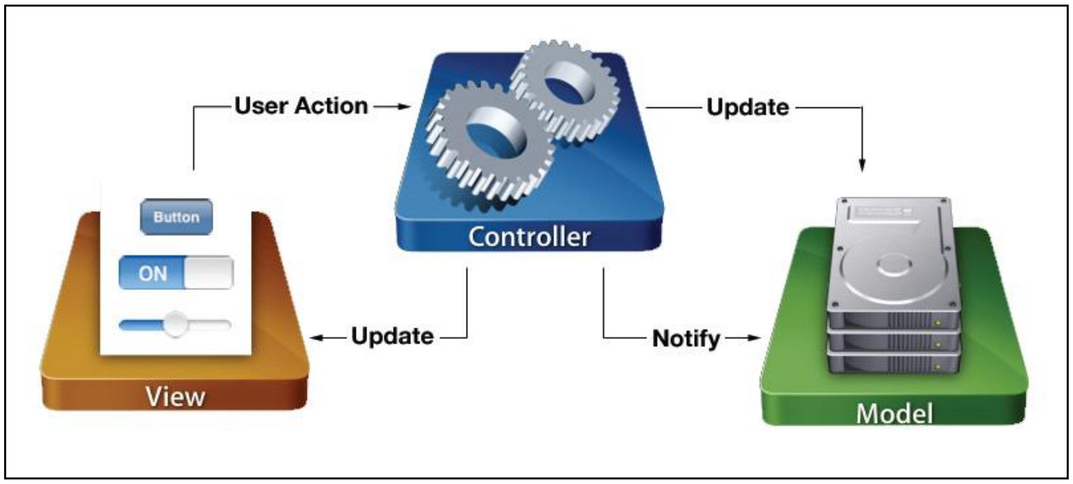

Figure 2.2: Component of MVC design pattern[57]

MVC is the most influential family of design patterns for synchronizing a user interface with the state of the application domain. The approach was first introduced in the 1980s, even before widget-based user interfaces were used [32]. Initially, MVC was used for designing and building desktop applications with rich graphical user interfaces. Over time, the original MVC pattern evolved and variants emerged driven by technological evolutions and new needs. Nowadays, MVC is used for integrating interface logic with domain logic in development of various domains, such as Web applications and Mobile systems [33].

Central to MVC is the separation of the representation of the application domain (the model) from the display of the application’s state (the view) and the user interaction processing (the controller)[29**]**

Since the late 1980s when MVC was documented, numerous new MV* design patterns emerged that aimed to eliminate the drawbacks of their predecessors.

The table below shows a brief description of the main characteristics of two different programming languages for MVC.

	Intent	Motivation	Structure	Collaborations	Consequences
Smalltalk’80 MVC	Separates the concerns of the application domain and its representation in three modules, each handling a specific task	Support the design and development of highly maintainable applications with rich user interfaces by maintaining a strict separation between domain logic and presentation logic	The three key components of the MVC pattern are Model, View and Controller.	The cooperation between Model, View and Controller relies on observer synchronization	The division of responsibilities of the MVC pattern has proven to be very effective.
Web MVC	Separates the domain logic from the presentation logic for the domain of Web applications in three components with distinct responsibilities	Due to specifics of the way the Web works, it matches well with the principles of the MVC design pattern.	The general principles of the MVC family apply to the MVC pattern: Model stores data, View displays data, and Controller handles user input.	The Web is stateless and operates as a set of requests and responses, so there is no need of strong synchronization.	The MVC design pattern supports clear separation of responsibilities of web application logic, which leads to better-organized code that is easy to understand and maintain.

Table 2.3: Smalltalk’80 MVC &Web MVC adapted from [29]

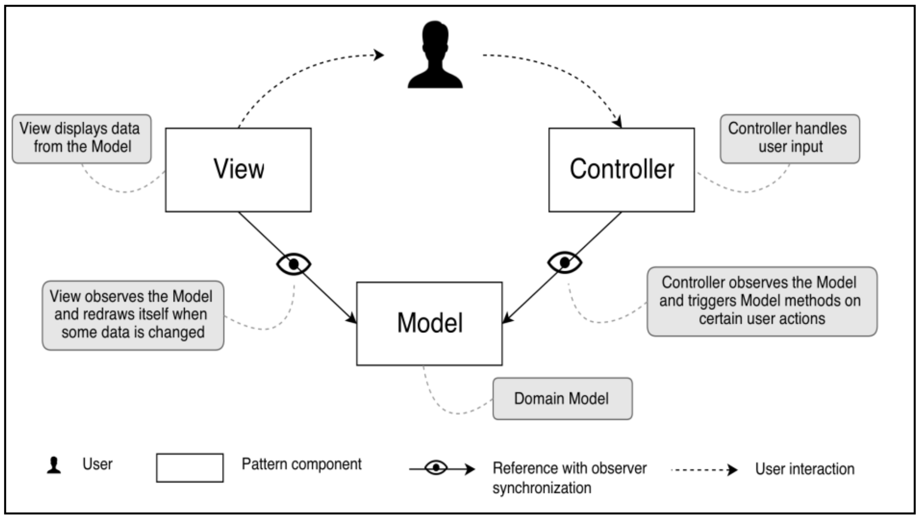
View displays data from the Model
View
Controller
Controller handles user input
View observes the Model and redraws itself when some data is changed
Model
Controller observes the Model and triggers Model methods on certain user actions
Domain Model
User
Pattern component
Reference with observer synchronization
User interaction

Figure 2.3: Smalltalk'80 MVC pattern [29]

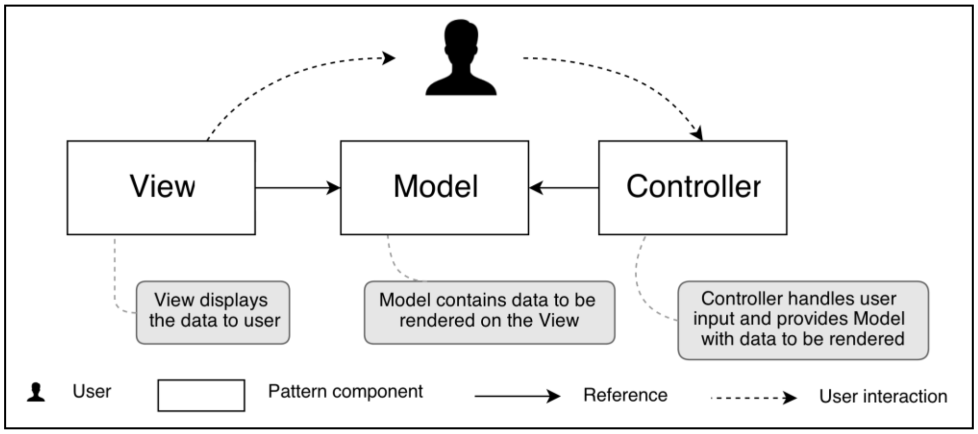
View
Model
Controller
View displays the data to user
Model contains data to be rendered on the View
Controller handles user input and provides Model with data to be rendered
User
Pattern component
Reference
User interaction

Figure 2.4: Web MVC pattern [29]

2.3.1.1 MVC Interaction Cycle

In this section we illustrate the MVC Interaction Cycle in four steps [43][44]:

- **The first step**: The user interacts with the view through a user input such as clicking a button or a link on a user interface. The view sends the user input event to the controller. The controller handles this request.
- **The second step**: The controller sends calls to the model to modify its state according to the request
- **The third step**: The controller sends calls to the view to modify its state. In fact, when the controller receives a request from the view, it may need to modify the view state; for example, the controller could enable or disable certain buttons or menu items in the user interface.
- **The fourth step:** The model updates the view representation when its state is changed. Actually, something changes in the model. This change is based on some requests by user input, such as clicking a button, or some other internal changes. The model updates the view that makes its display and eventually the user interface changes. This means that the view updates its state directly from the model**.**

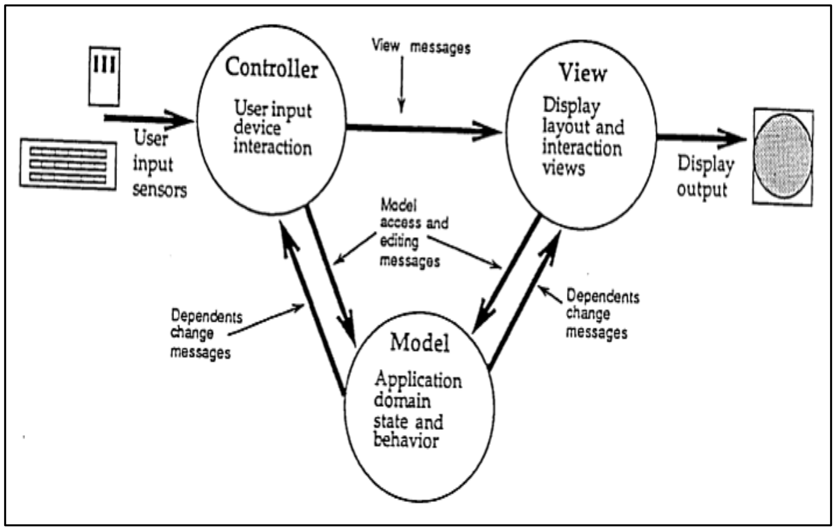
Controller
User input device interaction
View messages
View
Display layout and interaction views
User input sensors
Display output
Model access and editing messages
Dependents change messages
Dependents change messages
Model
Application domain state and behavior

Figure 2.5: Interaction between components of MVC [38]

2.3.1.2 Advantage of MVC Design Pattern

1- The biggest advantage of the MVC design pattern is that it separates the model from the view. As stated earlier that the model represents the data and the business rules and the view represents elements of the user interface such as texts, images, and form inputs.
    - This separation allows for easy changes for each object without affecting each other. It also leads to easier maintenance and modification of the UI[46].
    - Separates the three objects that lead designers to work on the UI of mobile devices without worrying about the underlying data .It also helps developers focus on the data instead of being too concerned about data presentation and avoid code repetition[44].

2- MVC has the ability to reduce designing time because programmers who focus on the controller object can work independently while designers are responsible only for the view object or model object.

3- MVC has the ability to bring about changes in the view object without recompiling the code of the model objects or the object of the controller[46].

2.3.1.3 MVC Architectural Pattern of Mobile Web Application

It was already mentioned in the introduction, that mobile technologies is one of the swiftly evolving areas in information technology. Mobile technologies are a perspective and a well suited investment for many reasons. Most electronic devices are becoming smaller, requiring less energy and a lower data transfer rate.

Nowadays, more and more people start to use mobile devices because they are simply very useful tools in for a wide range of purposes and fields. At the same time, the performance of these devices increases rapidly and extends the possibilities of using such devices. Based on innovations, new and more smart devices are produced , which means using mobile devices is becoming inevitably prevailing and handy in so many fields.[41]

As we all know that there are so many types and brands of mobile devices and different ways to present website content to them. In 1997 ,there was a Wireless Application Protocol (WAP) Forum established and one year later , WAP 1.0 standard was introduced , which described complete software stack for mobile internet access [25].

Since 2004, WAP disappeared from handsets as there is now support for full HTML even in low-end market phones. Before WAP in Europe there were similar technologies, most notably was Japanese i-Mode which also used cut-down version of HTML back in late 90' [24].

The market of mobile devices will be increasing in the next few years. So using an effective method to easily transfer existing applications into the new market will be very valuable as   mobile web application using MVC architectural pattern ,which is a very fast and efficient way to build different end-user sites without the need of redeveloping the core application .

2.3.2 Model View Presentation (MVP) Design Pattern

MVP was first described by **Mike Potel** from Taligent (IBM) in 1996. **Potel** in his work on MVP [36] questioned the need for the Controller class in MVC. He noticed that modern Operating System user interfaces already provide most of the Controller functionality in the View class and therefore the Controller seems slightly redundant [29].

MVP patterns provide flexibility for designers who can allocate responsibilities in different ways, so the patterns can be adjusted into a wide range of application scenarios. On the counter side, MVP patterns are not very strict regarding separation of concerns, which may increase the complexity of the code and hamper maintainability.

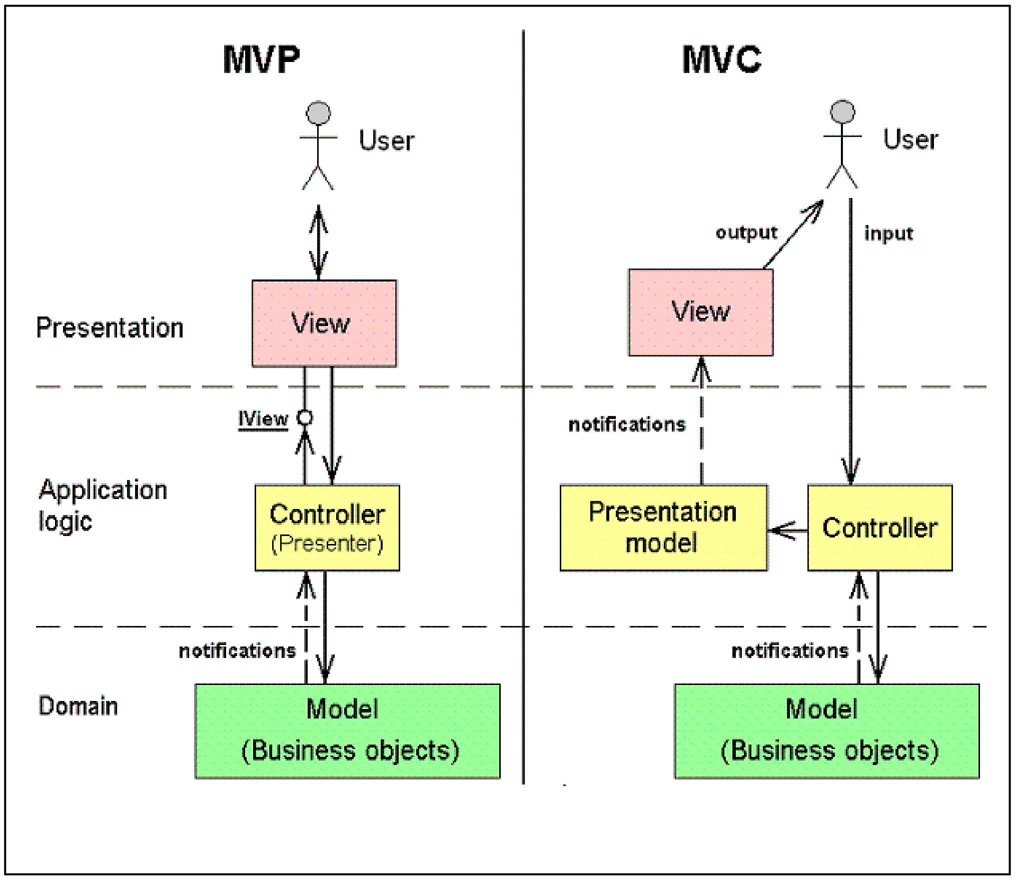

Figure 2.6: Differences between MVC & MVP [29].

2.3.3 Model View View Model (MVVM) Design Pattern

The term MVVM was first introduced by the WPF Architect, **John Gossman**, on his blog in 2005 [37]. It was then described in depths by **Josh Smith** in his MSDN article "WPF Apps with the Model-View-View Model Design Pattern" [38].

MVVM patterns support simultaneous representation of multiple views on the same data. State of the art frameworks that support MVVM provide support for declarative specification of parts of the synchronization and its automatic execution. MVVM emphasizes separation of concerns, which Support understandability and maintainability. On the other hand, extensive use of observer synchronization combined with multiple views can have a negative effect on system [29].

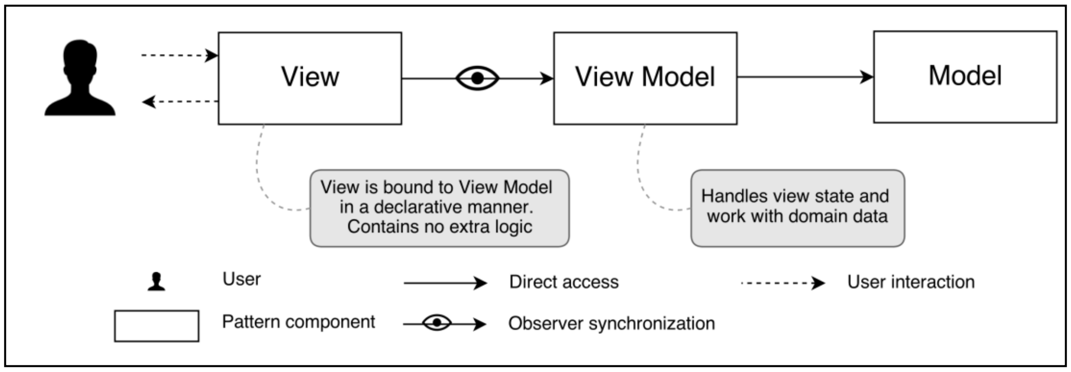
View
View Model
Model
View is bound to View Model in a declarative manner. Contains no extra logic
Handles view state and work with domain data
User
Direct access
User interaction
Pattern component
Observer synchronization

Figure 2.7: MVVM pattern [29]

The table below illustrates the main differences between **MVC** and **MVVM.**

MVC	**MVVM**
The controller determines the Application Flow	View Model encapsulates presentation logic and state
Controller is a must	View Model is an optional pattern
User hits the controller first	User hits the view first
The view knows about the Model	View can not see the Model
View obtains an instance of the Model	View has an instance of the View Model

Table 2.4: Differences between MVC & MVVM

2.4 ICONIX Methodology

ICONIX is an object oriented software development methodology, consists of dynamic and static workflows [47], and it uses UML diagrams in a four-step process that transfers from use case to code.

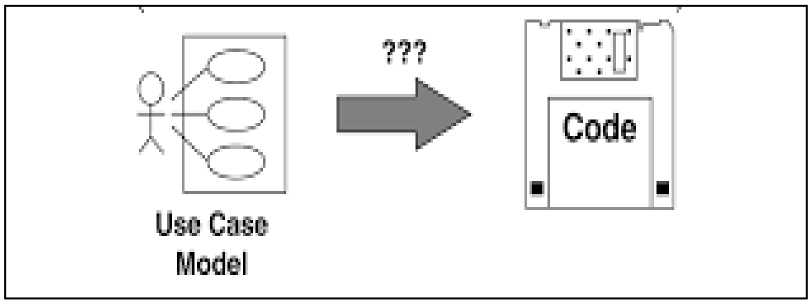

Figure 2.8: ICONIX Methodology [48]

ICONX process is very suitable for MVC and focuses on the area that lies in between use cases and code. It also describes the core logical analysis and design process.

This essential logical analysis is designed to move the user from requirement analysis to implementation in a quick and efficient manner.

A very essential element of the ICONIX Process is the use of **Jacobson's Robustness Analysis** technique to bridge the gap between requirements analysis and detailed design. In fact, this analysis approach is the most convenient for MVC.

2.4.1 ICONX Process

The ICONIX process is divided into four milestones or founding steps .At every stage; all steps are carefully reviewed and updated.

Milestone 1: Requirements Review

This step is considered as requirements analysis, which is performed by identifying a problem statement and real-world domain objects in a domain model. It also includes identified functions requirement by Use Case diagram, which generates some prototypes for each use case. From this analysis, use cases can be identified, a domain model is produced and some prototype GUIs are made.

Milestone 2: Preliminary Design Review

Another very important milestone is Robustness Analysis. It is considered as a middle ground between analysis and design as it discovers objects for each use case and updates the domain model according to the objects discovered.

Once use cases are identified, texts can be entered to see how users and the system will interact. Then, robustness analysis is done to find any potential errors in the use case text which means the domain model is updated accordingly.

The use case text is important to observe how users will interact with the proposed system.

Milestone 3: Detailed Design Review

This step is mainly concerned with the design. We use objects which are discovered from the robustness analysis to make sequence diagrams, and we use the domain model as explained in the previous step to design the class diagrams.

During this stage of the ICONIX process, the domain model and use case text from milestone 2 are used to design the system . In this step ,class diagrams are produced from the domain model and the use case texts are used to make sequence diagrams.

Milestone 4: Deployment

The final step is the execution of all the desired system .Unit tests are written to verify if the system matches to the use case text and sequence diagrams. Finally, the code is written using the class and sequence diagrams as a guide.

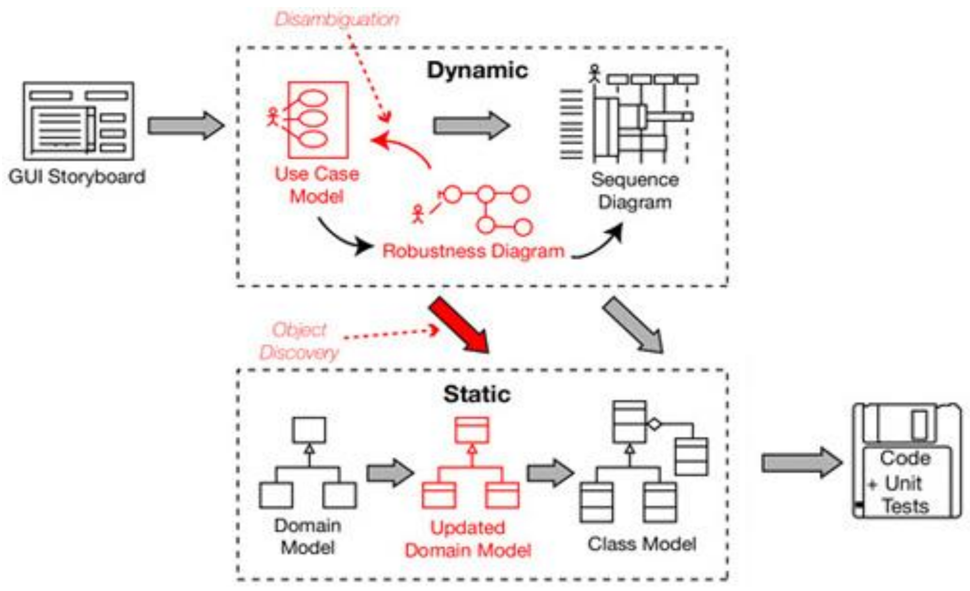

Figure 2.9: The phases of the ICONIX process [48]

2.4.2 Robustness Analysis

Robustness analysis intends to fill the gap between analysis (the what) and design (the how). Actually, robustness analysis is considered as a preliminary design when designers make assumptions on the design structure and start thinking of any possible technical solutions.

For supporting robustness analysis, they use robustness diagrams. it uses UML concepts. and also It is a specialized communication diagram that uses stereotyped objects.

It was Introduced by Jacobson and it basically analyzes use cases and estimates the first set of objects that participate with those use cases. It also classifies objects according to the roles that use cases play. Robustness analysis helps discover objects and identify the main domain classes before design or implementation [49].

Robustness Analysis consists three of elements

- **Entity objects**: describe objects dealing with persisting states.
- **Boundary objects**: describe links between the system and environment.
- **Controller objects**: describe use-case specific behavior

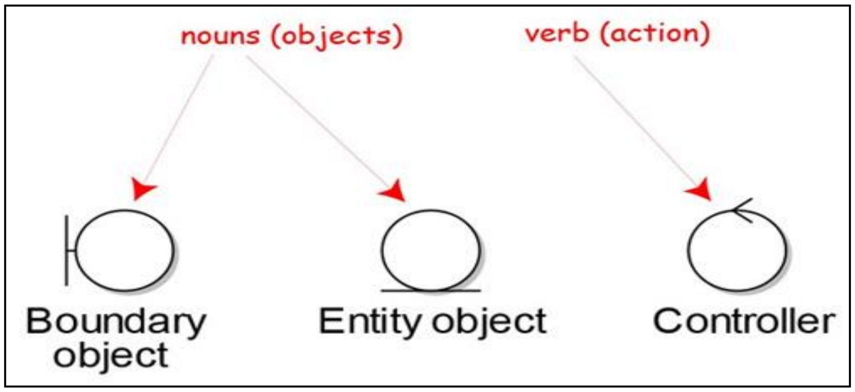

Figure 2.10: Robustness diagram symbols (EBC) [49]

2.4.2.1 Robustness Analysis in MVC Design Pattern

MVC objects are related to EBC objects in one-to-one mapping. Thus, entity object maps onto model object, boundary object maps onto view object, and controller is the same in MVC and EBC [50].

The MVC and EBC are techniques that separate responsibilities in software to avoid potential coupling [51].

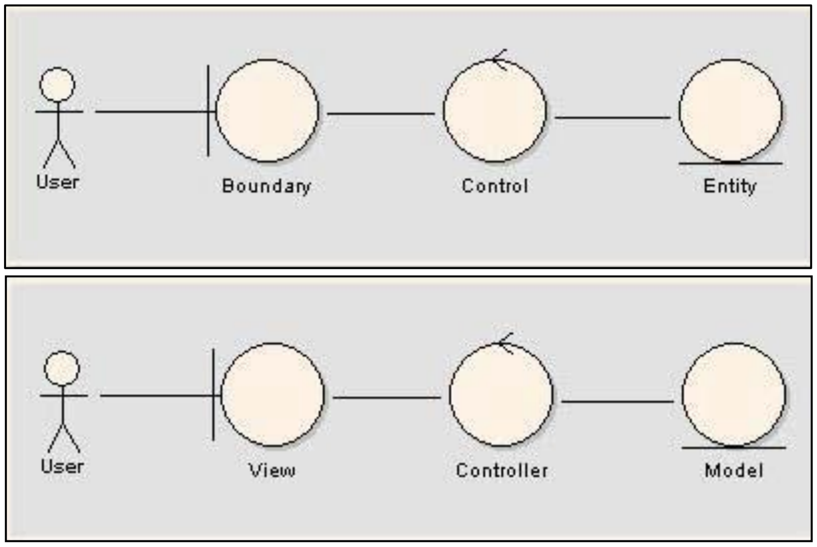

Figure 2.11: Robustness analysis in MVC [50]

ICONIX is a methodology approach that uses entity, boundary, and controller objects that presents a fundamental approach for modeling software systems , and also it is the most convenient for GUI-based Object-Oriented (OO) applications.

The figure below shows Entity Boundary Controller pattern:

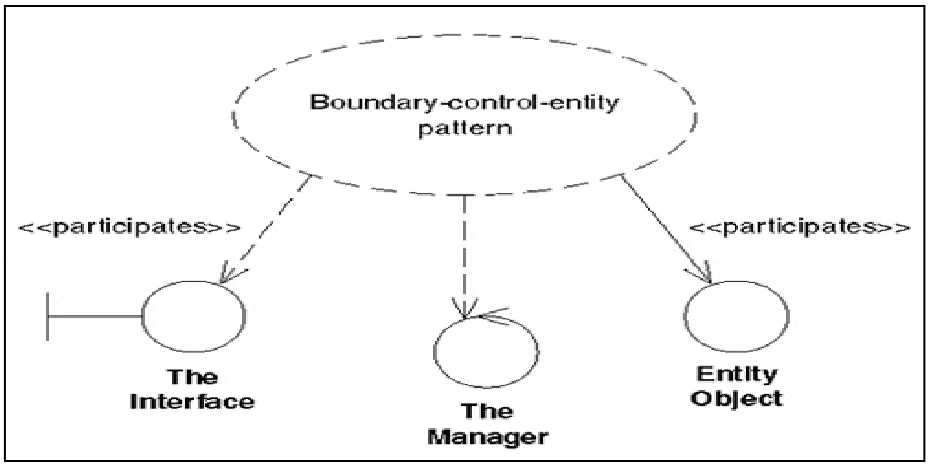

Figure 2.12: EBC pattern

Since the analysis use cases consider matters such as “what”, and design “how,” robustness analysis is therefore really preliminary design[49].

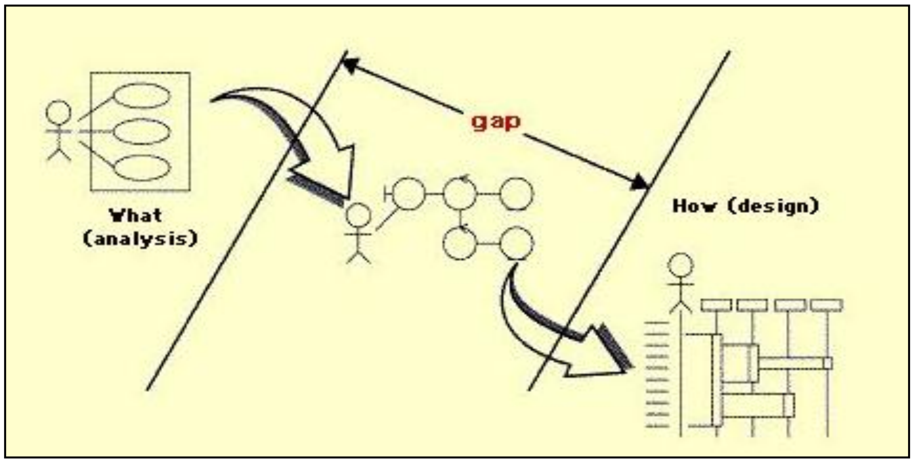

Figure 2.13: Robustness analysis mediates between analysis and design [48]

2.4.2.2 Robustness Diagrams Rules:

Robustness analysis describe how actors use boundary (interface) objects to communication with the system[52] .

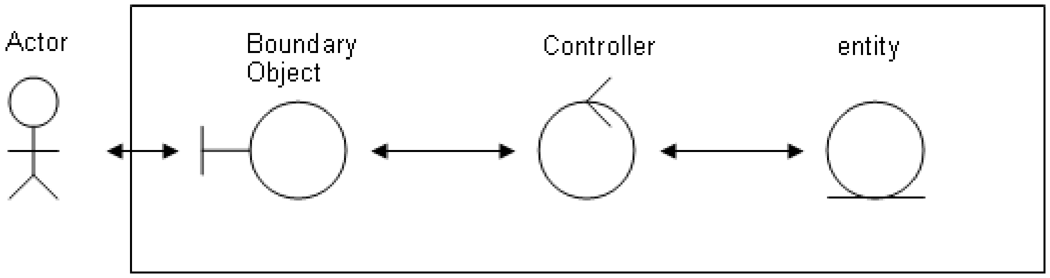

Figure 2.14: Actors communication with the system

When we start extracting objects, analyze use cases and attempt to ignite interaction among diagrams through these objects, there are four primary rules that must be followed:

- **Actors** are allowed only to interact with the boundary objects.
- **Boundary** objects are allowed only to deal with controllers and actors.
- **Entity** objects controllers are allowed to engage in the same interaction.
- **Controllers** basically interact with boundary objects and entity objects, and to other controllers, but not with actors.

Generally, there is one basic map between the actor and the boundary objects. In this context, The controller objects can merely interact with all the objects, but not allowed to access the actor. Also, the entity objects can interact with each other through the controller object. Thus, the controller object is considered the route of communication between objects [50].

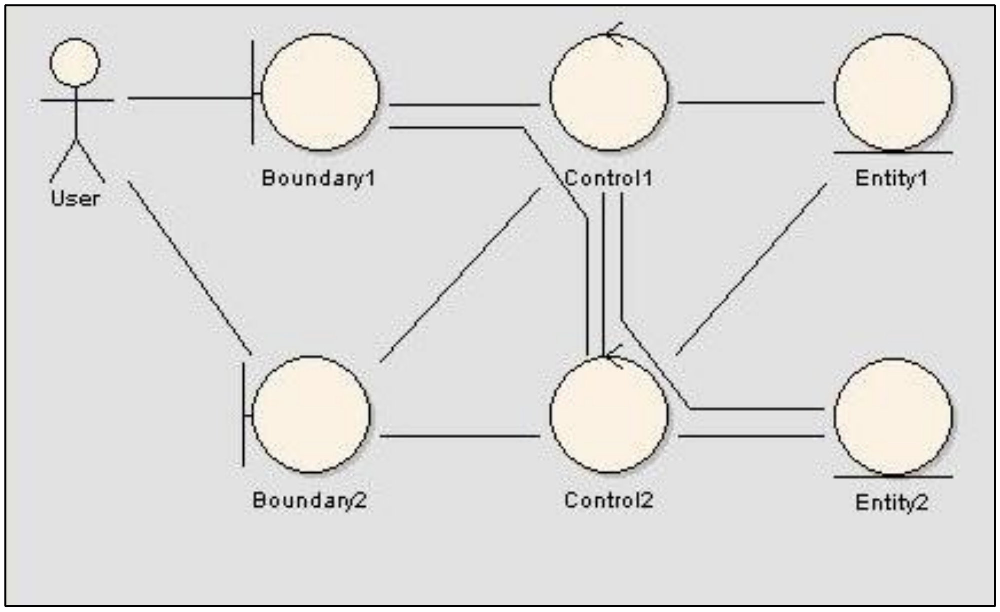

Figure 2.15: Robustness diagram rules [50]

2.5 Related work

Erik G. Nilsson [15]: has proposed a set of important guidelines on how to solve various and complex problems. This set was mainly used as means for facilitating development of more user friendly applications on mobile devices (PDAs/Smart Phones).His work has explicitly provided practical advice for each single problem in different contexts.The main focus was given to three problems which were categorized as follows : utilizing screen space ,interaction mechanism and design at large .Each was described in more detail.

Eric Magnuson[16]: attempted to provide a clearer definition for design patterns, find out how effective they are and why they have not been utilized to their full potential .Additionally, the use of design patterns in practice is demonstrated by using patterns to design a user interface. The interface was then implemented in a prototype application for Nokia N 900 mobile as a model. Examples for the prototype were the Two Panel selector , Card Stack and Hub and Spoke applications for the mentioned device .In his detailed work , there was a great effort to demonstrate the full process involved in developing user interfaces with design patterns.

Amin A.Rasooli[17]: has clearly presented the real-world challenges faced by users of mobile devices like Visibility of system status, Flexibility and efficiency of use and many more. It was suggested in his work that designing a HCI pattern is a solution to solve our usability problem that occurs in different contexts of use. The way the problems were categorized was similar to Erick G.Nillson .

Astahovs Ilja[18]: has adopted MVC design pattern for game development and conducted a number of case studies .It was aimed at assessing the functionality and efficiency of a variety of design patterns on a game project. His findings were that Some design patterns (e.g. MVC and Observer, State Machine and Singleton) are best suited when used together. Both MVC and State (State Machine) patterns have shown its great potential as a foundation for small game applications. The MVC is a decent choice for overall game structure and this pattern is adopted by other middleware. The State Machine can be used to split the Model further into smaller modules. He also emphasized that Even though the mentioned design patterns date back to 1994, they are still used by the popular frameworks. Such DPs as MVC and State Machine are best suited for building a game engine or a small game from scratch because they define the whole structure of the application.

CHAPTER 3

The Proposed Solution for Designing a Mobile User Interface Using MVC Design Pattern

3.1 Introduction

This chapter describes in detail the proposed solution that we used to solve the problem. The proposed approach is based on using Model View Controller design pattern (MVC) and ICONX methodology for mobile devices. **Section 3.2** gives an overview on the proposed prototype. **Section 3.3** introduces the framework of Model View Controller pattern. **Section 3.4** explains the ICONX methodology, including the stages, the models and the techniques of each stage of the solution approach.

3.2 Overview of the Solution Approach

The proposed solution to the stated problem is mainly based on utilizing MVC pattern (see Section 2.3.1 in Chapter 2) and ICONX methodology (as given in Section 2.4 in Chapter 2). The methodology used for solving this problem is ICONX. It involves many steps including the selection of a proper design pattern. The ICONIX process is a streamlined approach to software development. One of the main advantages of this process is that helps extract code from use cases quickly and efficiently. This process is done by using a concentrated subset of the UML and related tools and techniques. This methodology has also the ability to solve complex problems related to mobile user interface. Thus, MVC proves to be the most effective and convenient design pattern to fix the stated problem.

3.3 Framework of MVC Design Pattern

MVC pattern is usually implemented by ASP MVC.NET and J2EE, but in this research we use ASP.NET MVC because it is easier than J2EE because J2EE has more details that need to be well studied .The following describes the major differences between ASP.NET MVC and J2EE MVC:

- The Model in ASP.NET MVC is business-based logic, while in J2EE MVC is simple java bean classes.
- The View in ASP.NET MVC is ASP files ,but in J2EE MVC is JSP pages.
- ASP.NET MVC has one controller for each possible view, but in J2EE MVC has only one controller.

ASP.NET is a development framework for building web pages and web sites with HTML, CSS, JavaScript and server scripting. This application framework was developed by Microsoft for building desktop applications. It is based on Common Language Runtime (CLR) which gives developers the freedom to develop applications in multiple languages like Visual C#, VB.NET, Visual J#, Visual C++ and other languages that are supported by the .NET framework [39].

ASP.NET supports three different development models: Web Pages, MVC , and Web Forms.

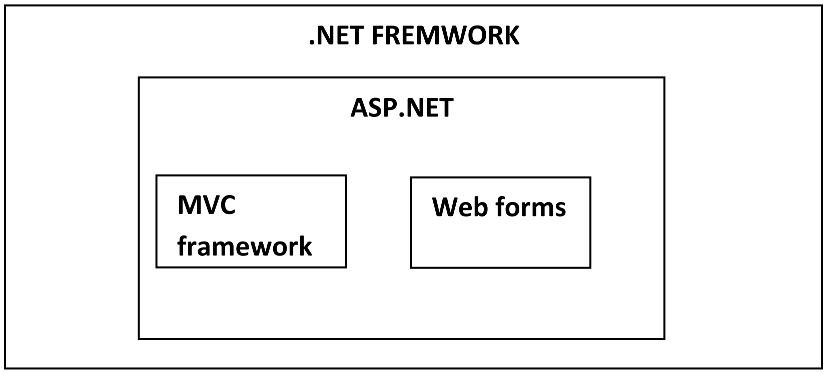

Figure 3.1: .NET framework and MVC[55]

MVC is actually one of three ASP.NET programming models, which include framework for building web applications using the Model View Controller design. The MVC model defines web applications with three logic layers: the data access layer (Controller logic), the display layer (View logic) and the business logic layer (Model logic), (for more detail on MVC design, see chapter 2 section 2.3.1)

In this research, we used a specific programming language with a framework such as C#, which is an object-oriented programming language by Microsoft. It mainly aims at combining the computing power of C++ with the programming ease of ASP.net [36].

3.4 ICONX Methodology

The ICONX Methodology is defined in detail. (see chapter 2 section 2.4). It mainly consists of four stages. Figure 3.2 shows the main stages of ICONX methodology and Figure 3.3 shows the sub-stages of each main stage.

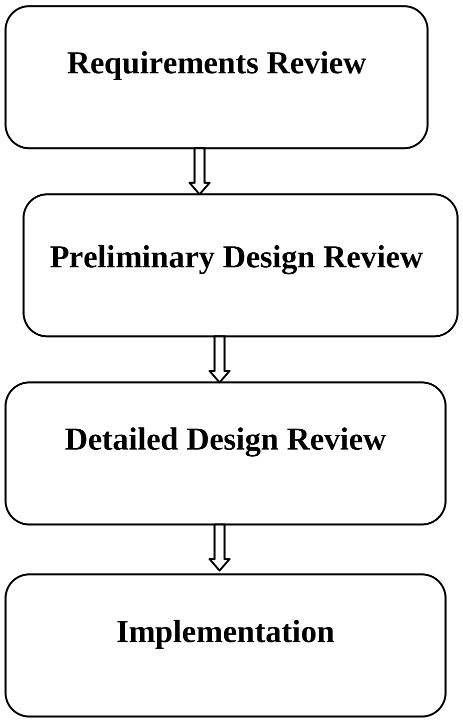

Figure 3.2: The main stages of ICONX methodology

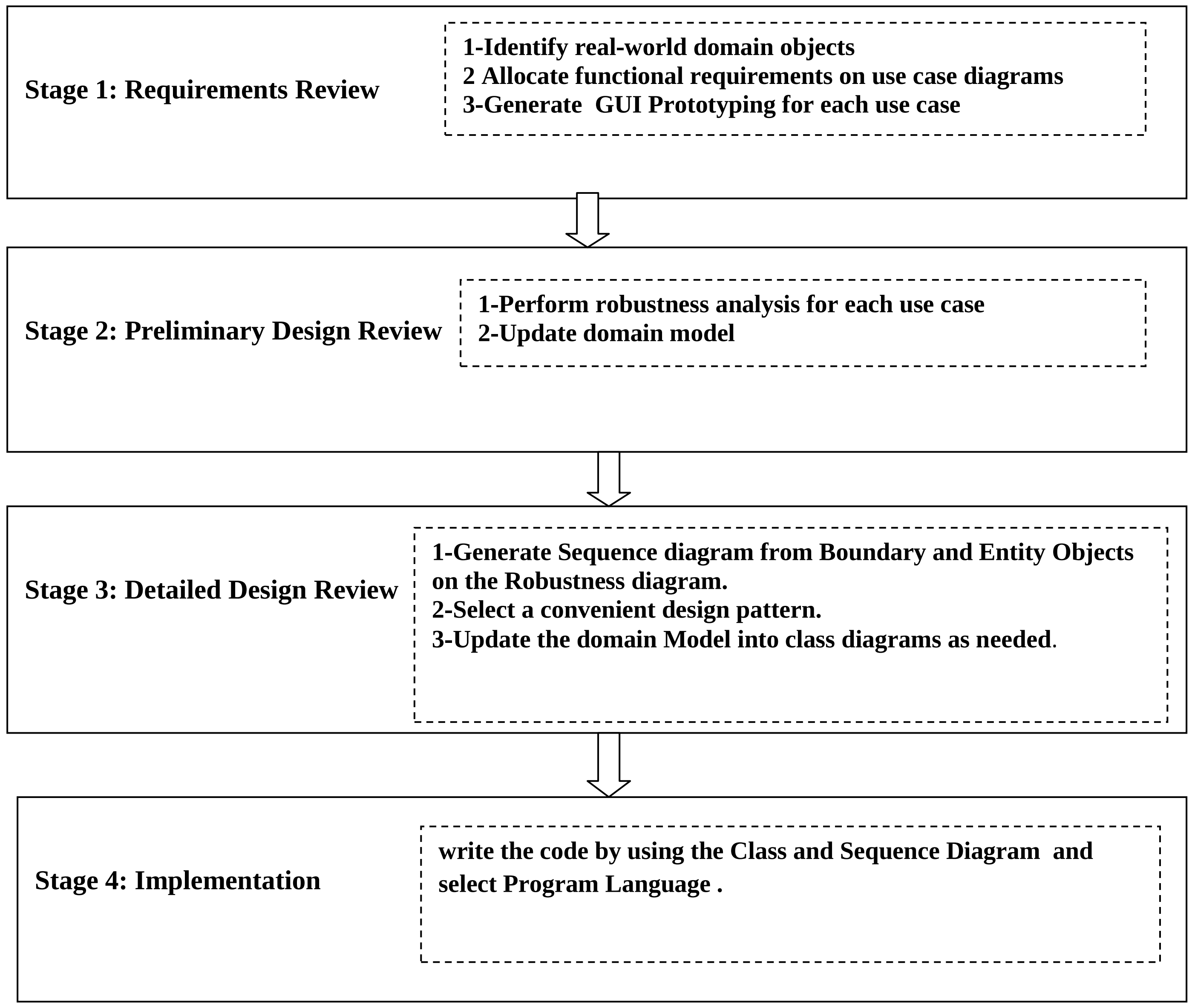
Stage 1: Requirements Review
1-Identify real-world domain objects
2 Allocate functional requirements on use case diagrams
3-Generate GUI Prototyping for each use case
Stage 2: Preliminary Design Review
1-Perform robustness analysis for each use case
2-Update domain model
Stage 3: Detailed Design Review
1-Generate Sequence diagram from Boundary and Entity Objects on the Robustness diagram.
2-Select a convenient design pattern.
3-Update the domain Model into class diagrams as needed.
Stage 4: Implementation
write the code by using the Class and Sequence Diagram and select Program Language .

Figure 3.3: The sub- stages of ICONX methodology

3.4.1 Stage One : Requirements Review

Once we think about implementing the ICONIX process, we must bear in mind some requirements for analysis. This analysis enables use cases to be identified, a domain model can be produced and some prototype GUIs are created as a result. This step includes identifying domain model, use cases and generating GUI prototyping for each use case.

3.4.1.1 Identifying Real-World Domain Objects

This is the first and the most important step. It simply focuses on the real world and describes the problem of user interface. A clear definition of the problem leads to better understand the next stages and to explicitly understand the scope of the problem.

3.4.1.2 Allocate Functional Requirements on Use Case Diagrams

The use case diagrams are usually defined during the requirements activities to capture the requirements of the functionalities of the system. It also presents the requirements that need to in explained in more detail. The scenario of the use case is to describe the interaction of the user with the system.

The use case illustrates all actors (actor is any person who has interaction with the user interface) .Another function of the use case is to describe how the user interface responds to those actors . The representation is used to extract the functional requirements of the system.

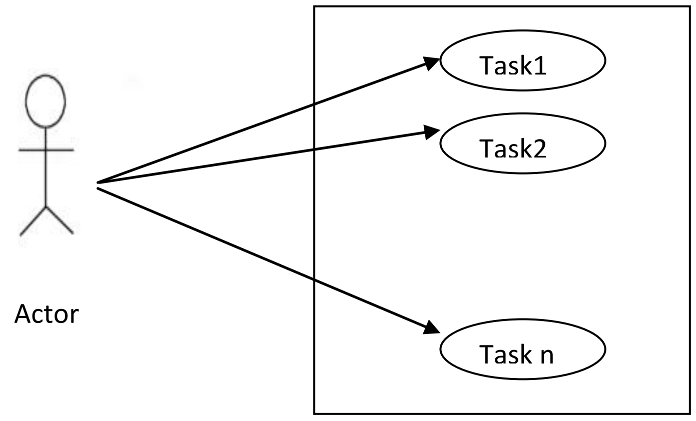

Figure 3.4: Use case diagram

3.4.1.3 Generate GUI Prototyping for each Use Case

At this stage, the final prototypes of the main components are placed. The user interface is the main component and user interface prototypes will be generated for each use case of the system being developed to illustrate their actions.

For examples, the elements of the proposed GUI include:

- **Button** : a control that can be clicked to perform an action
- **A text box**: allows the user to enter text information which is to be used by the program.
- **A list box**: a type of box within a collection of graphical user interface widgets that can be grouped.

3.4.2 Stage Two :Preliminary Design Review

This step is mainly concerned with using a robustness analysis, which is the best for visually describing the MVC (discussed in more detail in chapter 2 section 2.4.2).This analytical procedure has two sub-steps : Perform Robustness Analysis and Update Domain Model.

3.4.2.1 Perform Robustness Analysis for each Use Case

The *Robustness diagram* includes multiple elements. It actually consists of a class diagram and an activity diagram. It visually represents behavior of use case , showing both participating classes and software behavior. A robustness diagram is probably easier to read than an activity diagram since objects speak to each other.

- **Boundary**: is the interface between the system and the outside world. Boundary objects are typically screens or web pages (i.e., the presentation layer that the actor interacts with).
- **Entity**: Entities are usually objects from the domain model.
- **Control**: Control objects are the “glue” between boundary and entity objects.

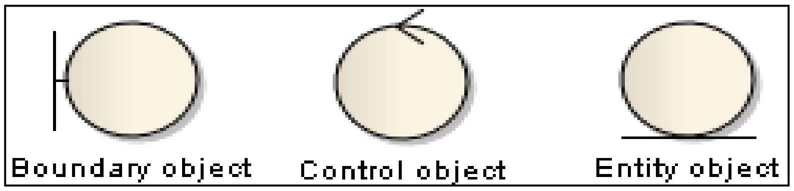

Figure 3.5: Elements of robustness diagram

3.4.2.2 Update Domain Model

This sub-task is performed through extracting entity classes from the domain model, then adding any missing entities discovered during the robustness analysis.

3.4.3 Stage Three: Detailed Design Review

During the stage of ICONIX process, the domain model and use case text from stage 2 are used to design the system being built. A class diagram is produced from the domain model and the use case text is used to make sequence diagram. This step is categorized into: Sequence diagram, Design patterns ,and Class diagram.

3.4.3.1 Generate Sequence Diagram from EBC on the Robustness Diagram.

This step explains the sequence model, which is one of the UML models .We must show interaction between the set of objects(e.g. boundary and entity) ,messages being sent and received by those objects and demonstrate the behavior of objects[53].

In fact ,the boundary and entity classes in a robustness diagram will generally become object instances in a sequence diagram, while controllers will become messages[49].

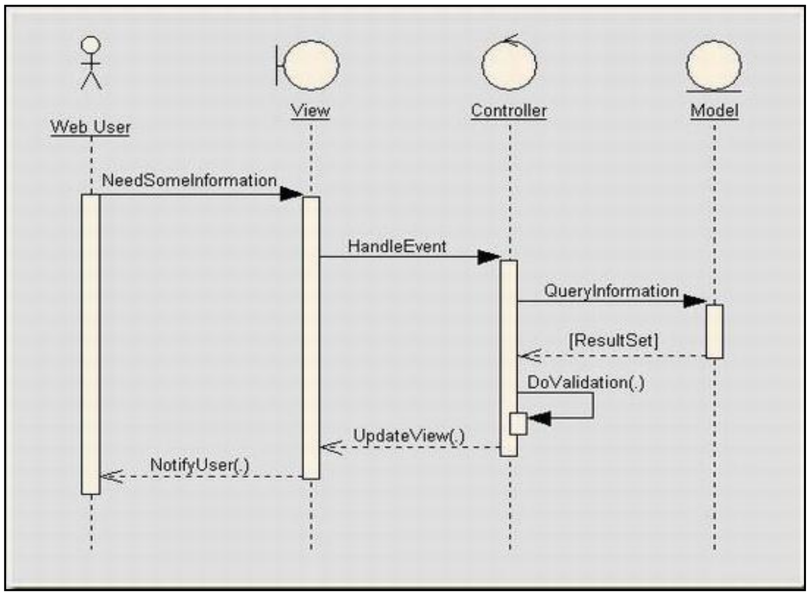

Figure 3.6: Sequence diagram of MVC[49]

3.4.3.2 Select Suitable Design Pattern

As we know, there are 23 different types of design pattern (explained in detail in chapter 2 section 2.2) However, we still need to define what we actually mean by *MVC design pattern*.

MVC is basically a set of classes to build a user interface. Those classes that define the main MVC relationship are Observer and Strategy .The diagram below illustrates the three essential types of objects in the Observer: the model is the application data, the view is the screen and the controller defines the way the View reacts to user input. As shown, this comprehensive Observer process allows us to attach multiple Views to the same model.

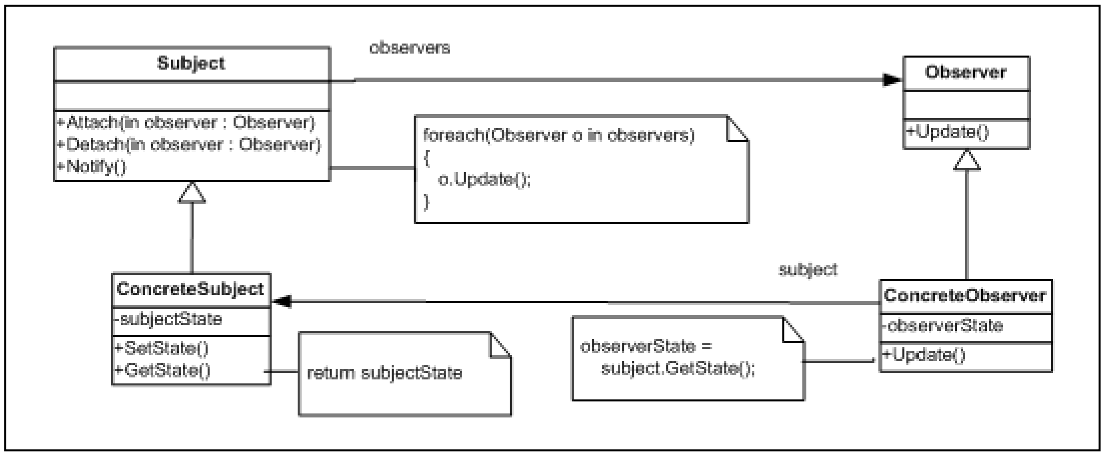

Figure 3.7: Observer design pattern[59]

This Observer pattern aims at defining the one-to- many relationships between the subject and the observers. This means that if the Subject is altered, then all Observers are updated .The Subject here keeps the list of the Observers and can attach and detach objects to the list. Another component of MVC is the View –Controller relationship.

The Controller is used by the View to implement a certain type of response .It also allows the View to respond differently to user input .This View-Controller connection is an example of the strategy design pattern.

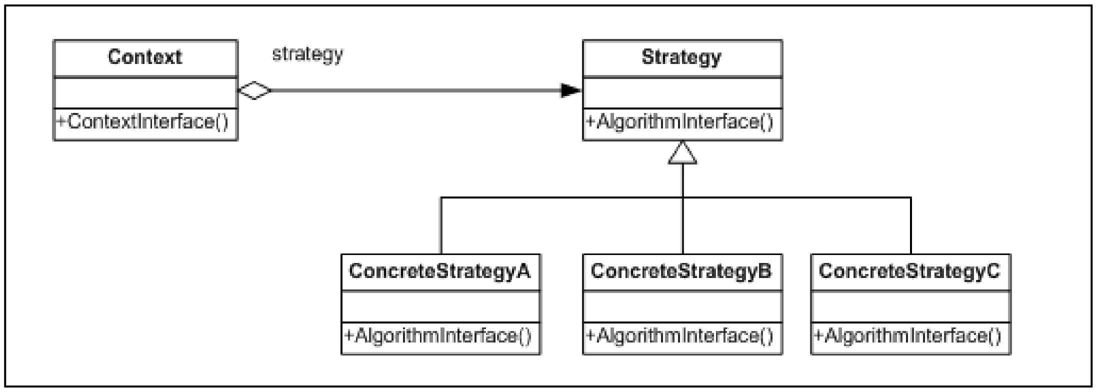

Figure 3.8: Strategy design pattern [58]

Now we can state that the View takes on the role of the Observer object and the Model acts as a Subject from the Observer pattern as shown in the two diagrams below:

The View is a Context and the Controller is a Strategy object. From this combined knowledge we can draw the MVC UML class diagram.

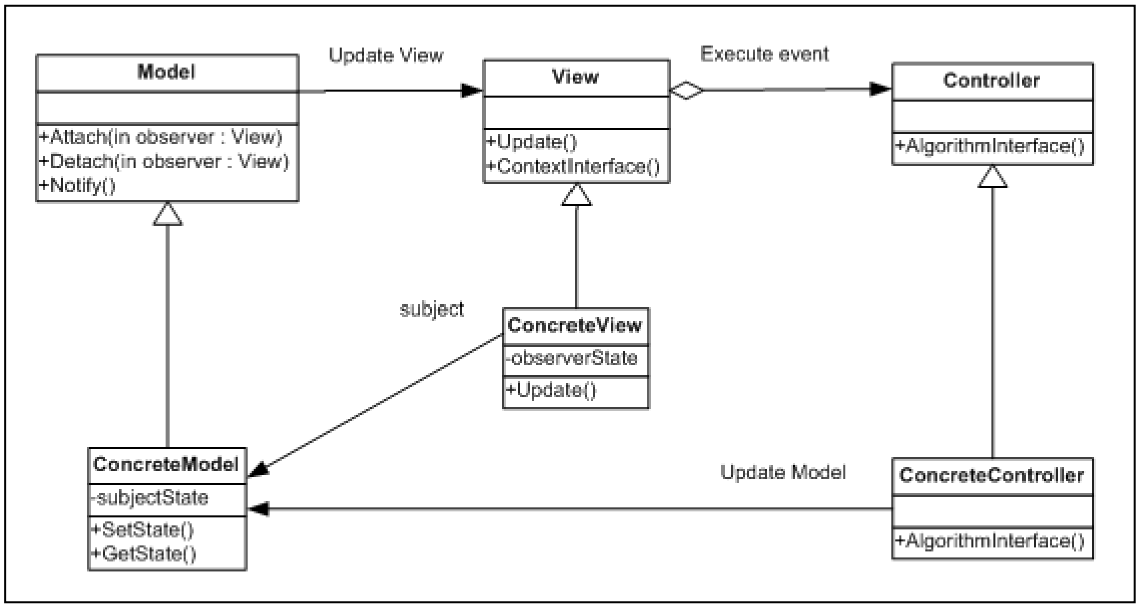

Figure 3.9: MVC design pattern

3.4.3.3 Update the Domain Model into Class Diagrams as needed.

Class diagrams indicate the set of classes and relationship between them. Every class contains three elements: Class name, Attributes and Method or Operation.

Class Name
Attributes
Operation

Figure 3.10: Class diagram

3.4.4 Stage Four : Implementation

This is the final stage of the ICONEX methodology. It verifies the system which will match up with the Use case, Text and Sequence diagrams. Finally, Code is written by using the Class and Sequence diagrams .In this research we used the ASP.NET MVC framework for designing user interface of mobile.