ISSN 22773061

IDENTIFICATION AND IMPLEMENTATION OF DESIGN PATTERNS IN MOBILE BANKING

M.Hemalatha , A.Vikneshraj, N. Adithya Prasanna

Assistant Professor, Sri Manakula Vinayagar Engineering College Pondicherry
hemalatha.smvec@gmail.com

Student, Department of IT, Sri Manakula Vinayagar Engineering College

vikneshrj005@gmail.com

Student, Department of IT, Sri Manakula Vinayagar Engineering College

n.adithyaprasanna@gmail.com

ABSTRACT

The aim of this paper is to develop a mobile banking system using design patters that provides customers with the facility to check their accounts and to do online transactions using mobile phones. There are various number of software design patterns that have been identified and used by software developers in various domains such as navigation systems, e-commerce, data mining, construction of operating systems, e-business, games and website designing and development purpose. In order to achieve our aim we are going to follow three steps. First step is the identification process, in this process we are going to study the various design patterns and the existing architecture of several mobile banking systems and will identify the design patterns based on the cross cutting concerns that occur in flow of the banking process. The second step is to implement the identified design patterns using the patterns skeleton code. We need to improve the reusability using object oriented programming. Finally we are going to implement the identified patterns and evaluate the patterns using already available methods such as SAAM, ATAM, ALAM (Architecture-Level Maintainability Analysis).

Keywords: Identification, Implementation, Evaluation, SAAM, ALAM.

ISSN 22773061

1.0 INTRODUCTION

The process of designing reusable object oriented software is very difficult as it involves steps like finding the related objects, grouping them into several classes at right granularity, defining the class interfaces and the hierarchies and establishing key relationship among them.The design should be specific to the problem to be solved and must address the future problem and requirements. Making a reusable and flexible design is difficult for the experienced object oriented designers. The designers will not try to solve every problem from its first principles rather than using solution from the past. Whenever a designer finds a good solution, he will use it again and again. Hence the designer will find the recurring patterns of classes and communicating object in several object oriented systems. These patterns are used to solve specific design problems on creating flexible, stylish and reusable object oriented designs. When a designer is familiar with such patterns, then the designer can apply them immediately to the design problems without having to rediscover them. Once the designer knows the patterns, then automatically a lot of design decisions occur. A design records the experiences in designing object oriented software has design patterns. Each design pattern consistently names, explains and evaluates a recurring design in object oriented systems. A design pattern helps to reuse designs and architectures and to select the design alternatives that construct a reusable system. Design patterns also improve the documentation and maintenance of existing systems by providing a specification of class and object interaction explicitly. The design patterns are not a new one, but the tracing is given in a new way. There are no application or domain specific design patterns .Hence the patterns have been used in applications such as reservation systems, e-commerce, navigation systems, operating system constructions, gaming and website development etc. Though patterns have been implemented in various domains but there are no identified patterns for the mobile banking system. The existing mobile banking system does not make use of design patterns and so if we use patterns, the reusability of the mobile banking system can be improved.

2.0 REASONS FOR USING MOBILE BANKING SYSTEM

The main reason for choosing the mobile banking system to implement the design patterns is that mobile banking is a real time application which is being used by millions of people all over the world. Mobile banking provides many advantages, such as good security, easy access and plentiful applications for smart phones. Another advantage is that there is more control of user's money. Each and every real time application strives hard to provide services to its users. The banking is one such system which involves several complex processes and these processes are confined and compacted into a mobile application which in turn simplifies the banking process and therefore providing services to fulfil the requirements of the customer. Thus it increases the customer satisfaction and his privacy. Hence we go for mobile banking in order to implement design patterns.

3.0 PATTERN ORIENTED SOFTWARE ARCHITECTURE

Pattern-Oriented Software Architecture is a methodology or technique which is used for the construction of software architecture. Pattern oriented software architecture is a system of patterns. It is software architecture study based on the usage of patterns. It shows us not only to group individual patterns into various kinds of structures but also to provide an effective environment for the construction of interactive and adaptable real time systems.

Table 1. Related Works

Paper Title	Description Of the paper	Solution inferred	Technique	Patterns used
Design Patterns for Games	It designs an OO model for a two-person game that can be used to identify a game model to work out the finest moves and also enables us to convey all dimensions of the games at the highest level of notion.	The OO design process involves identifying the variant behaviours of the system, encapsulating them into abstract subsystems and decoupling them from the Invariant core. Design patterns are used extensively to achieve the proper abstractions and decoupling.	Min-max algorithm and mvc patterns are used to decompose the overall architecture of the program.	State ,Visitor and Strategy
Patterns for E-commerce applications	It provides a conceptual framework for reasoning on design reuse in Web Applications. It deals with applications that have an object model, a navigational view and an interface. E-commerce applications will	These patterns focus mainly on ways to solve usual problem where customers have to find and buy products in the shop. They provide solutions to the Web application designer in order to make these applications more usable and effective both from the point of customers and owners of the store. By	Pattern mining and Pattern advising	Opportunistic Linking, Advising, Explicit Process, Easy Undo and Push Communication.

	engage particular problems at the object level	showing non-trivial extension of the basic Web model (based on nodes and links) these patterns help to improve the navigation topology and some aspects of the customer-store communication 'styles.		
A Navigation Pattern for Scalable Internet Management	This paper introduces the echo pattern, a scheme for distributing management operations, which addresses some difficulties. Management operations based on this pattern do not need knowledge of the network topology, they can dynamically adapt to changes in the topology, and they scale well in very large networks.	As the pattern dynamically adapts to changes in the network topology, it does not require global network information .	Wave-propagation algorithms, traversal algorithms	Echo pattern, navigation patterns, star pattern

4.0 PATTERN

Pattern in software architecture is the way of extracting architectural design ideas as predictable and reusable descriptions. A pattern is a solution to the problem that arises within a specific context. Pattern falls into the family of similar problems. It is the process of distilling common factors from the system. Pattern is said to be as the relation between the context, problem and solution. A pattern has four essential elements namely Pattern Name, Problem, Solution, Consequences

Pattern name is used to describe a design problem, its solutions and consequences in a word or two. It makes us easier to think about design patterns and to communicate them and their trade-offs to others.

Problem describes when and how to apply the design patterns at any situation. It includes a list of constraints that must be satisfied before it makes sense to apply patterns.

Solution describes the elements that frame the design, their relationships, responsibilities and collaborations. Consequences are the results and trade-offs applying the patterns.

5.0 Types of Patterns

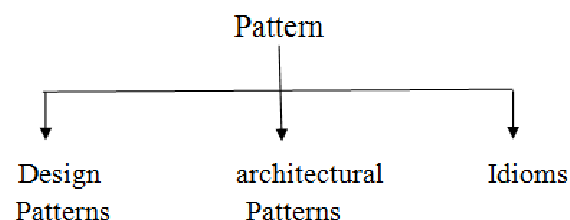


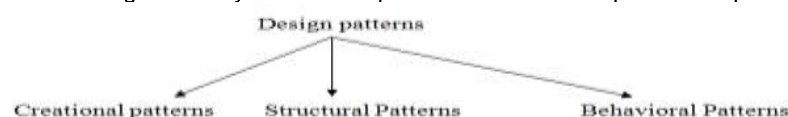
Figure 1. Different types of patterns

5.1 Architectural Pattern

It is highest level of patterns which helps us to define the basic structure of an application by specifying the responsibilities. It also includes the rules and guidelines.

5.2 Design Patterns

It is the medium level Patterns refining the subsystem or components. It is the blue print of the particular solution.



ISSN 22773061

Figure 2. Categories of design patterns

5.3 Idioms

An idiom is a low level pattern that is specific to a programming language. An idiom describes how to implement particular aspects of components or the relationship between them using features of the given language. It addresses both design and implementation.

5.2.1 Types of Creational Patterns

Design Pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not 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

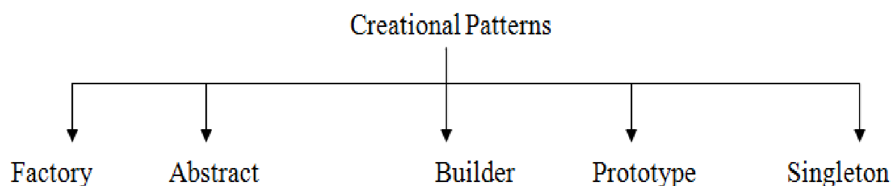


Figure 3.Kinds of creational patterns

5.2.1.1 Factory Method

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. (Gamma et al., 1994)

5.2.1.2 Abstract Factory

Provide an interface for creating families of related or dependent objects without specifying their concrete classes. . (Gamma et al., 1994)

5.2.1.3 Builder

Separate the construction of a complex object from its representation so that the same construction process can create different representations. (Gamma et al., 1994)

5.2.1.4 Prototype

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype. (Gamma et al., 1994)

5.2.1.5 Singleton

Ensure a class only has one instance, and provide a global point of access to it. . (Gamma et al., 1994)

5.2.2 Types of Structural Patterns Structural Pattern

Structural patterns deal with the composition of several classes and objects in forming larger software architectures. They make use of the concept of inheritance in order to combine the interfaces and the implementations. So that the properties of the main class or parent class can be combined to form the resultant class.

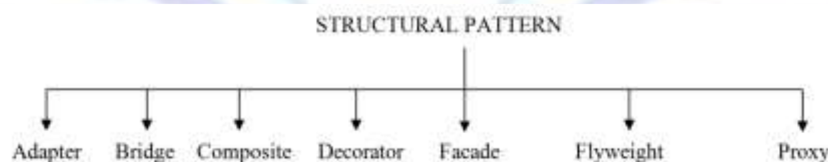


Figure 4. Kinds of structural patterns

5.2.2.1 Adapter Pattern

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces. (Gamma et al., 1994)

5.2.2.2 Bridge Pattern

Decouple an abstraction from its implementation so that the two can vary independently. (Gamma et al., 1994)

5.2.2.3 Composite Pattern

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. (Gamma et al., 1994)

5.2.2.4 Decorator Pattern

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality. (Gamma et al., 1994)

5.2.2.5 Facade Pattern

Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. (Gamma et al., 1994)

5.2.2.6 Flyweight Pattern

Use sharing to support large numbers of fine-grained objects efficiently. (Gamma et al., 1994)

5.2.2.7 Proxy Pattern

Provide a surrogate or placeholder for another object to control access to it. (Gamma et al., 1994)

5.2.3 Types of Behavioural Patterns

Behavioural patterns are design patterns that recognize familiar communication patterns between objects and comprehend these patterns. Thus by performing so, these patterns boost flexibility in transmitting out this communication.

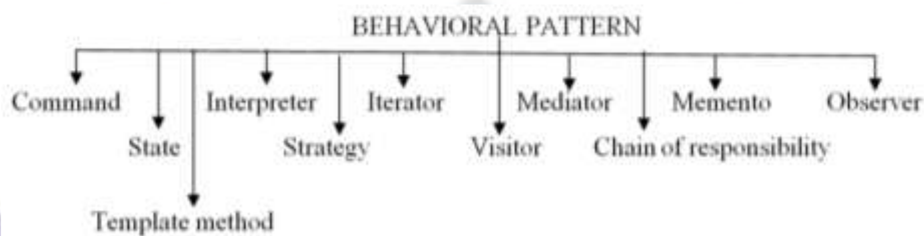


Figure 5. Kinds of behavioural patterns

5.2.3.1 Chain of responsibility

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle. (Gamma et al., 1994)

5.2.3.2 Command

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. (Gamma et al., 1994)

5.2.3.3 Interpreter

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language. (Gamma et al., 1994)

5.2.3.4 Iterator

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. (Gamma et al., 1994)

5.2.3.5 Mediator

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently. (Gamma et al., 1994)

5.2.3.6 Memento

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later. (Gamma et al., 1994)

5.2.3.7 Observer

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. (Gamma et al., 1994)

5.2.3.8 State

Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class. (Gamma et al., 1994)

5.2.3.9 Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. (Gamma et al., 1994)

5.2.3.10 Template method

ISSN 22773061

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. (Gamma et al., 1994)

5.2.3.11 Visitor

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates. (Gamma et al., 1994)

6.0 IDENTIFIED PATTERNS FOR MOBILE BANKING

In mobile banking various operations is similar so the similar operations are created as the patterns.

- Proxy pattern
- Composite pattern
- Singleton pattern
- Decorator pattern
- Façade pattern

6.1 Proxy Pattern

The proxy pattern is used in situations where one object is used instead of another object, by acting as a placeholder for that object to control references to it. While reserving online tickets using mobile banking, money is debited directly from the user account. Invoking unsought objects is a burden on the utilisation of resources. Therefore we do not want to instantiate the objects unless they are requested by the client. In this scenario, we are using EJB interfaces and there are only going to interact with the client side. Hence the objects will be instantiated upon the client's request.

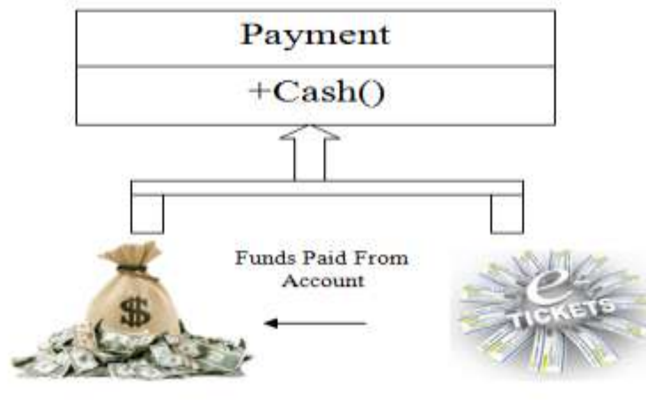


Figure 6.Proxy in online ticket reservation

6.2 Composite Pattern

Composite pattern allows the clients to treat individual objects and compositions of objects homogeneously. It arranges objects into tree structures to symbolize entire hierarchy in general. The below figure shows us an example of an enquiry operation, which is a composition of several sub processes and are represented in the form of a tree. A banking application needs to manipulate this hierarchical collection of operations in enquiry into "primitive" and "complex" objects. Processing of a simple FD enquiry operation object is handled one way, and processing of a loan enquiry is handled differently, since there are so many classifications of loan enquiries. Hence we use this composite pattern whenever we use operations that contain more sub operations or components, each of which could be a composite.

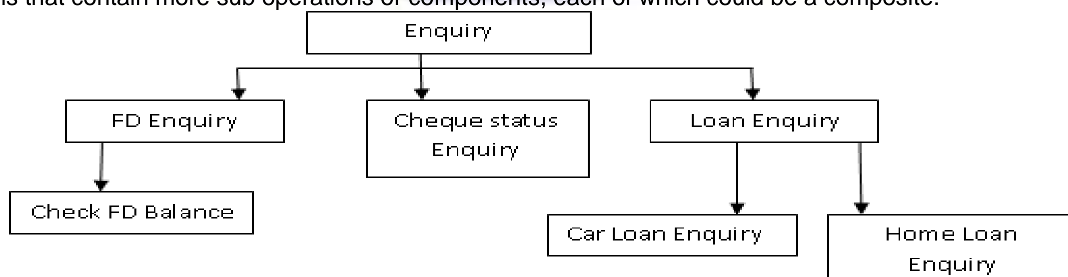


Figure 7. Visitor pattern used in Enquiry operations

6.3 Singleton Pattern

A bank must ensure that an account is accessed by only one customer at a particular duration and the customer must be able to access the account from anywhere around the globe. A banking application may not be able to provide access to an account, when two customers trying to access the same account and for this lazy initialization and global access are

necessary. To provide high level security for a user account and to provide a global point of access a bank must facilitate the customer with a single user name and password and only that authorized user can access the account at any time (No other person can access the same account at that time). For this purpose we make use of the singleton pattern.

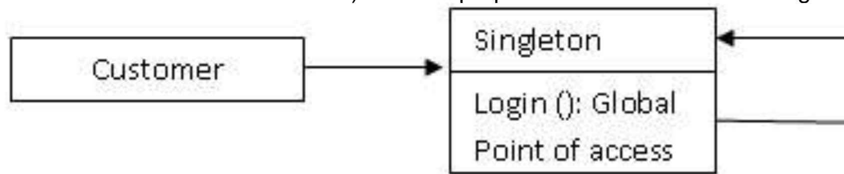


Figure 8. Singleton pattern using point of access

6.4 Decorator Pattern

Mobile banking must provide additional facilities such as SMS alerts and advertisement. Decorator pattern provide a flexible alternative to sub operations for extending functionality. Inheritance of additional functionalities is not feasible because it is possible only for static features and changes made in any feature affects the entire working. We are adding so many services to the existing system such as SMS alerts to the user to indicate the transaction (withdrawal and deposit), purchase alerts for M-Shopping. To give these additional services for the user we make use of decorator pattern.

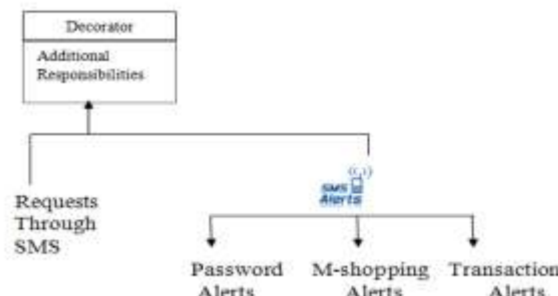


Figure 9. Decorator pattern used in mobile alerts

6.5 Façade Pattern

In a bank, services to the customer are provided through a customer service representative for each area or zone. The customer service representative acts as a mediator between the customer and the bank. We need to use only a subset of a complex system and not the entire system and each user wants to interact with the system in a particular way. The bank manager cannot able to provide the service directly instead of manager; a customer service representative is employed to provide the service. The Facade presents a new interface for the user of the existing system to use. We use a facade pattern which hides the complex database access interface behind a few easy to understand and maintainable interface.

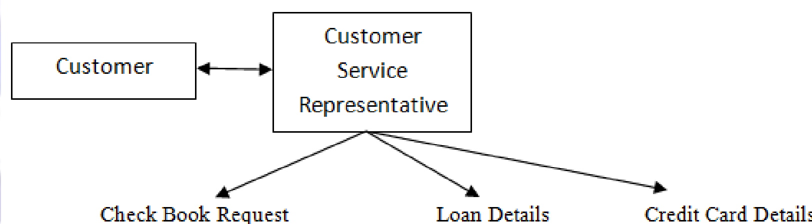


Figure 10. Facade pattern used in customer service interface

7.0 IMPROVING REUSABILITY USING DESIGN PATTERNS

After the identified patterns are implemented in mobile banking, we are going to prove that the proposed mobile banking using design patterns is more reusable than the existing banking systems (Without usage of patterns). In order to measure the reusability, we make use of the existing reusability formula for object oriented programming.

8.0 CONCLUSION

Although patterns are available for various real time applications so far there are no patterns identified for mobile banking applications. Banking is a huge application consisting of many tedious operations that are branched one below another. So it consumes more time for the developer to develop a banking application from the beginning. In order to overcome this issue we use design patterns, which provides more reusability to the developer. Our proposal will support the software developers to develop a more reusable mobile banking application in future. We also plan to evaluate our architecture using patterns with the existing architecture.

REFERENCES

[1] , Andrés Neyema,, Sergio F. Ochoa, José A. Pinob, Rubén Darío Franco ,A reusable structural design for mobile collaborative applications.The Journal of Systems and Software 85 (2012) 511– 524.
[2] Erik G. Nilsson, Design patterns for user interface for mobile applications. Advances in Engineering Software 40 (2009) 1318–1328.
[3] Saifullah M Dewan, Issues in M-Banking: Challenges and Opportunities. Proceedings of 13th International Conference on Computer and Information Technology (ICCIT 2010)23-25 December, 2010, Dhaka, Bangladesh.
[4] Parul Gandhi & Pradeep Kumar Bhatia, Reusability Metrics for Object-Oriented System: An Alternative Approach.
[3] Identifying, Relating, and Evaluating Design Patterns for the Design of Software for Synchronous Collaboration Claudia Iacob University of Milan Via Comelico, 39/41Milan, Italy.
[4] K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra, Software Reuse Metrics for Object-Oriented Systems.
[5] Mr.A.Meiappane, Ms.J.Prabavadhi, r.V.Prasanavenkatesan "STARTEGY PATTERN: PAYMENT PATTERN FOR INTERNET BANKING" International Journal of Information Technology and Engineering (IJITE), ISSN 2229-7367, March 2012.
[6]. Adaptive framework of the internet banking services based on customer classification", A.Meiappane, K.Gideon and Dr.V.Prasanna Venkatesan ,International Conference on Advances in Engg and Tech, (ICAET-2011).
[7]. Fundamental Banking Patterns Lubor Sesera SOFTEC & FIIT STU Slovakia.
[8]. A Navigation Pattern for Scalable Internet Management, K.-S. Lim, R. Stadler, 2001 IEEE.
[9]. Patterns for E-commerce applications, Gustavo Rossi , Fernando Lyardet , Daniel Schwabe, LIFIA Facultad de Informática.
[10]. Architectural Design Patterns for Flight Software, Julie Street Fant, Hassan Gomaa, Robert G. Pettit, 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops.

Author's biography with Photo

M.Hemalatha

Assistant Professor,

Sri Manakula Vinayagar Engineering College,

Pondicherry,

India

A.Vikneshraj,

Department of Information Technology,

Sri Manakula Vinayagar Engineering College,

Pondicherry,

India

N. Adithya Prasanna

Department of Information Technology,

Sri Manakula Vinayagar Engineering College,

Pondicherry,

India