

# DOMAIN-DRIVEN ARCHITECTURE FOR OBJECT-ORIENTED SOFTWARE SYSTEM

**Shafeeq Ahmad**

Azad Institute of Engineering & Technology, India  
ahmad\_shafeeq@rediffmail.com

**Dr. Vipin Saxena**

Babasaheb Bhimrao Ambedkar University, India  
vsax1@rediffmail.com

## ABSTRACT

The Unified Modeling Language (UML) is one of the important modeling languages used to design the software problems. The main aim of this paper is to develop a complete process of software architecture for the object-oriented software system. This software architecture will ensure non-functional requirements as well as the functional requirements of the software system. The software architecture will also consider the requirements for the domain of the problem. To describe the functional & non functional requirements, a case study of ATM machine is considered. The major finding of the paper is to trace the outline of architecture from problem domain.

**Keywords:** UML, software architecture, functional & non-functional requirements

## 1 INTRODUCTION

The term “Software Architecture” is very difficult to define. Some of the researchers define this term. In [3], [8], [9], [12] and [13] Software Architecture defines with the highest level of a system design and system architecture can be described as a set of elements (components) along with their externally visible properties and relationships among them. This term can also be defined in terms of pattern oriented software architectures & this is available in [4], [2] and [5]. The Unified approach of software development is designed & developed by Jacobson et al. [6], [7] and [10]. The role of software architecture is similar in nature to the role architect plays in building construction. Building architects look at the building from various viewpoints which is useful for civil engineers, electricians, plumbers, carpenters and so on. This allows the architects to see a complete picture before construction begins. Similarly, Software Architecture System is described as different viewpoints of the system being built. These viewpoints are captured in

different models & available in [9] and [11]. View Models play an eminent role in all scientific and engineering disciplines. For Physics, Mathematics, Biology, Chemistry and Economic works, lots of models are provided to solve the complex tasks. In the present work, a procedure for the different views of software architecture is explained with functional and non-functional requirements and these requirements are according to the domain of the software problem.

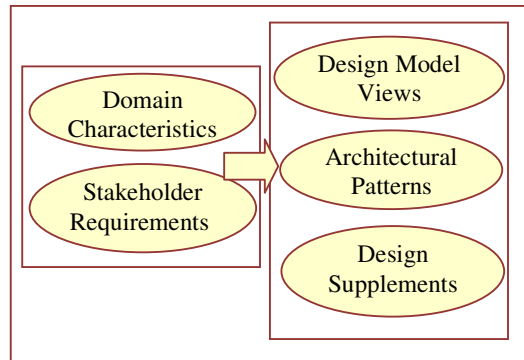
## 2 ELEMENTS OF SOFTWARE ARCHITECTURE

System architecture is defined as a set of design decisions. These decisions may be technical and commercial in nature. The present paper suggests and describes different elements of software architecture shown below in Fig. 1.

The below software architecture is influenced by some important factors [1] i.e. stakeholders requirements and domain characteristics for which software system is being developed. These factors provide help to derive the software architecture.

The derived-architecture avoids irrelevant things, which are not concerned with the problem domain and this makes development procedure very simple.

Influence factors; domain characteristics and stakeholders requirements work as input to object-oriented software architecture as shown below in Fig. 1. The influence factors help in describing architectural elements and taking different design decisions.



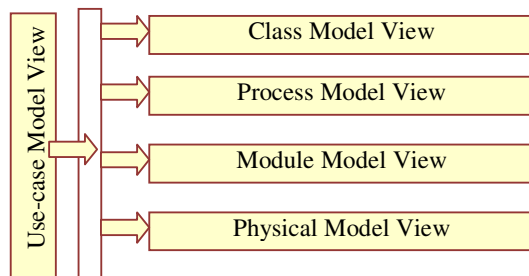
**Figure 1:** Elements of the software architecture

Design model views, Architectural patterns & design supplements are described below in brief:

### 3 DESIGN MODEL VIEWS

The Model view is an abstraction that excludes details that are not relevant for a particular model view of the system. Each model view can be considered as a software blueprint and each can use its own notation, can reflect its own choice of architectural patterns, and can define what is meant in its case by components, and relationships.

Model views are not fully independent. The components of one model can relate to components in another model. It is also necessary to find out the relations between them. There is no standard set of model views to consider. However, the model views, shown in the following Fig. 2, are taken on the basis of the works of [9], [3] and [7].



**Figure 2:** Design model views

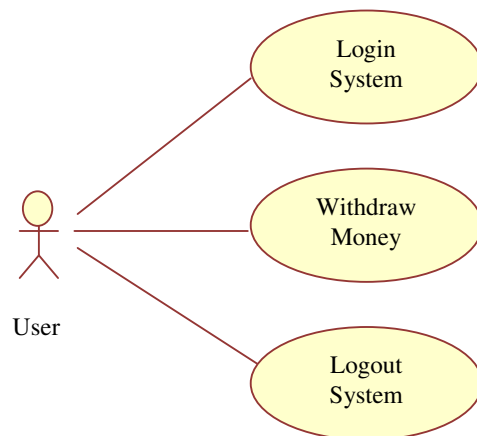
#### 3.1 Use-Case Model View

A use-case is a description of a set of sequences

of actions, including variants. A system yields an observable result of value to an actor. An actor is a coherent set of roles that users play when interacting with the system. An actor might be another system.

The components in this model view are the actors and the use-cases. The relationships are associations between actors and the use-cases, and dependency relationships between use-cases. Use-cases can be used to describe all types of stakeholders' requirements in a context, not only the functional requirements but also non-functional requirements.

Example 1: The following Fig. 3 shows a use-case diagram with an actor for the functioning of the ATM machine:



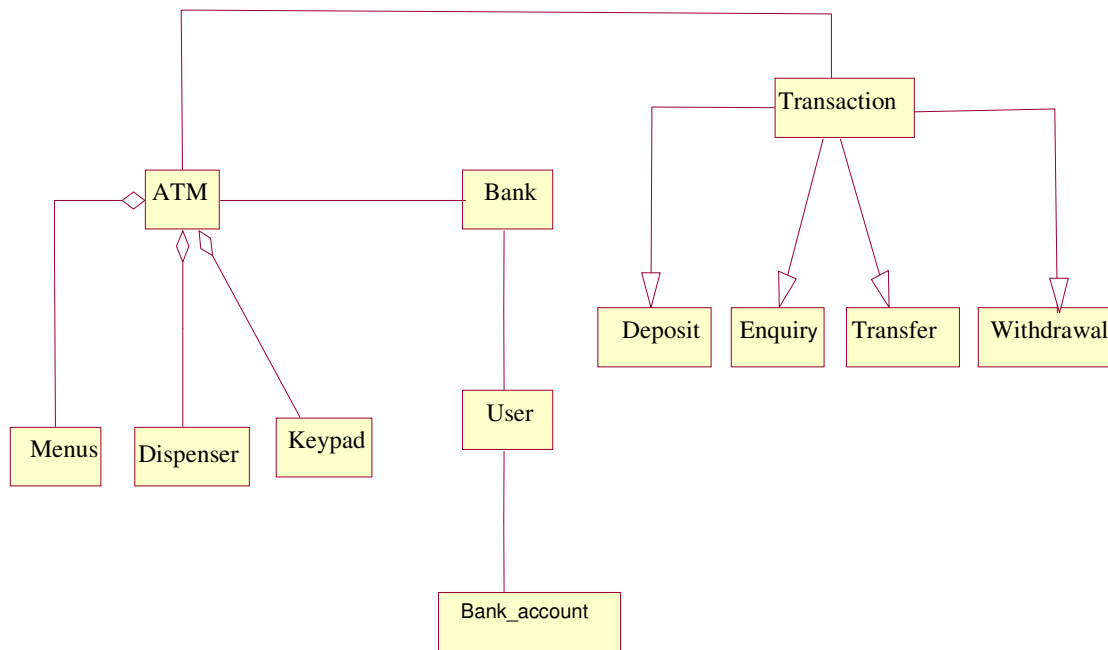
**Figure 3:** A Use-case model view of an ATM system

In the above figure User is an actor who can login the ATM machine & withdraw the desired amount and after the use of system, user log out the system. The above figure is only use case model view of the ATM system. When one wants to design the software architecture then one should focus on the use-cases that pose control on the software architecture. This means use-cases that capture the system's critical requirements, e.g. developer's requirements such as modifiability and client's requirements such as the functionality are most important and are used most frequently. Use-cases are used both to drive the process of defining the software architecture and to evaluate if the stakeholders requirements have been fulfilled. [3] describes a Scenario based Software Architecture Analysis Method (SAAM), that is close to the use-case approach. SAAM shows how all requirements can be described in scenarios that are same as use-cases, and how they can be used to define architecture and to assess the fulfillment of the stakeholders' requirements. An excellent software

development process is that in which testing is applied at all stages of development.

### 3.2 Class Model View

The class model view deals both with the structural aspects of the system as well as the dynamic aspects of the system. The components in this model view are objects or classes. The relationships are generalizations, aggregations and associations.



**Figure 4:** A class model view of an ATM system

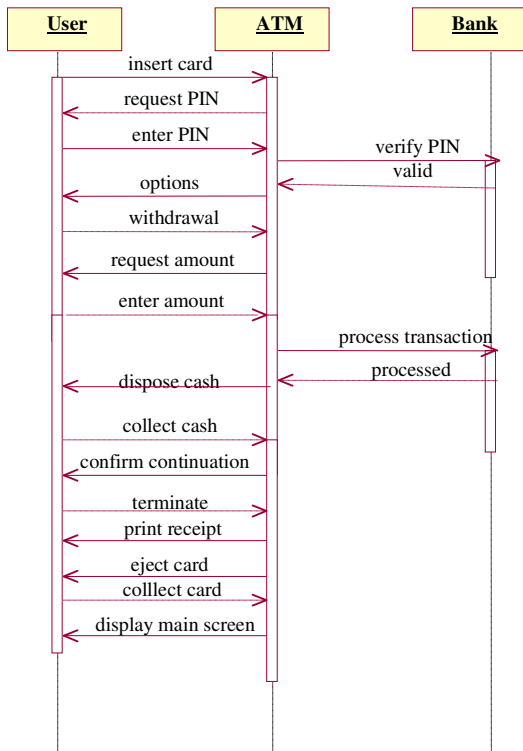
Classes are a way to organize the system, and the objects describe the dynamics of the running system by their behavior and interactions. The class model view encapsulates data and their related operations for reading, treating and updating of the data. The class concept can both be used to describe the conceptual entities of the system, its surroundings as well as the implementation of the system. The above Fig. 4 is an example of ATM system. In this diagram which is self explanatory, major classes are User, Bank, Bank\_account, ATM & Transaction and these classes are abstracted from the problem domain of the ATM. The Transaction class is further categorized as Deposit, Enquiry, Transfer & Withdrawal classes. ATM class consists of Menus, Dispenser & Keypad Class and User is associated with the ATM class & record of the User is available in the Bank. Bank\_account keeps the

record of the accounts of the User and it is associated with the Bank.

### 3.3 Process Model View

The process model view deals with the dynamic issues of communication and synchronization in a running system. The relationships between the processes deal with process communication, synchronization and concurrency. The components of process model view are processes and threads. A process is a sequence of instructions with its own control. Fig. 5 shows the sequence diagram under process model view. A process may have a number of threads.

A process can be started, shut down, recovered, reconfigured, and can communicate and synchronize as necessary with other processes. The diagram shows the behavior of the ATM system. There are three major objects selected from the class model view & these objects are the User, ATM & Bank. The vertical pipe shows the life line of each object. Initially user inserts a credit card & corresponding PIN number which is verified by ATM object through Bank. After this, options will appear on the screen & user will select option withdrawal and enter the withdrawal amount. After transaction and verification from Bank ATM machine will dispense the cash amount to the user. As per the user action, machine will print the receipt, and then eject the card and after this main screen will be displayed on the ATM machine.

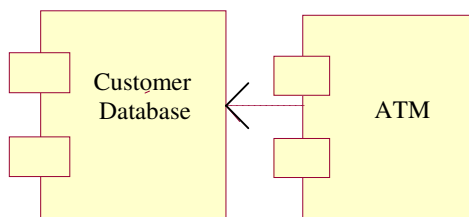


**Figure 5:** A sequence diagram of an ATM system

### 3.4 Module Model View

The module model view deals with the structural issues of the system. The components identified in this model view are modules. The relationships between the components are aggregation and dependency. The module model view is used to modularize and organize the system into comprehensible units, for example the organization of system into applications that are equivalent to subsystems. The module model view often reflects the conceptual elements of the system. The following Fig. 5 shows the module model view of an ATM system:

The above figure shows the user database attached with Cash dispenser, Display, Card Reader and Receipt Printer and these are associated with the computer systems.

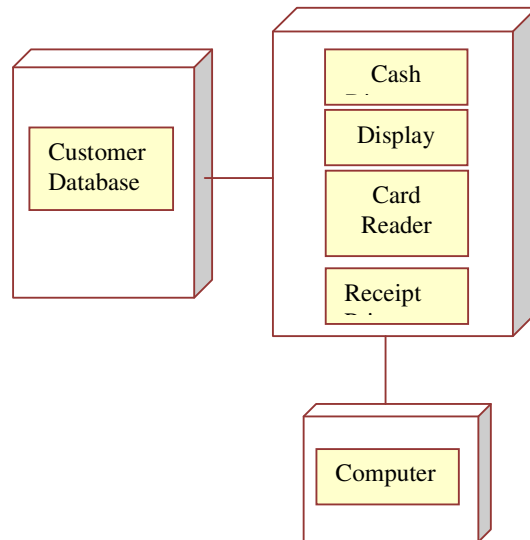


**Figure 6:** A module model view of an ATM

### 3.5 Physical Model View

The physical model view also deals with the dynamics of the running system, mainly how the processes are using processor capacity. The components in this model view are processors.

The relationships are “communicates-with” associations. The physical model view is the model view showing the deployment of software onto hardware. Fig.7 is the physical model view of an ATM system. If systems are running on only one machine then this model view is covered in the process model view.



**Figure 7:** A physical model view of an ATM system

## 4 ARCHITECTURAL PATTERNS

The goal of patterns within the software community is to create a body of literature to help software developers to resolve recurring problems encountered throughout the software development process. Architectural pattern describes a problem that occurs over and over again, and describes the core of the solution to that problem in such a way that the solution can be used again. Patterns can therefore be used when the problem is a recurring problem and that has been described and solved already. Christopher Alexander et al. [2] introduced the term pattern language means that one should go through patterns in a sequence, moving from the larger patterns to the smaller. Alexander thereby stressed the importance of recognizing the various levels of patterns in designing architecture. Both patterns and frameworks have to achieve large-scale reuse by capturing successful software strategies within a particular context. The primary difference is that frameworks focus on reuse at the level of detailed design, algorithms and

implementation. In contrast patterns focus more on reuse of recurring architectural design themes.

#### 4.1 Types of Patterns

The term "pattern" is often used to refer to any pattern that addresses issue of software architecture, design, or programming implementation. The following are the three types of patterns:

##### 4.1.1 Architecture Pattern

An Architecture Pattern expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

##### 4.1.2 Design Pattern

A Design Pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of communicating components that solves a general design problem within a particular context.

##### 4.1.3 Idiom

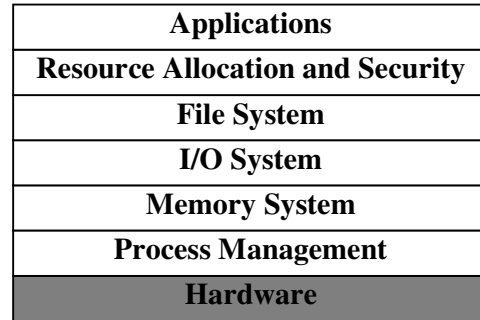
An Idiom (sometime called coding pattern) is a low-level pattern, specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of a given language.

Coding patterns, such as command, adapter, bridge etc. are now well established in the programming community. Patterns have also been extended into the architectural area; client-proxy server, pipe and filter, and so on. Adoption of these and other architectural patterns help to solve recurring architectural design problems in the same way that coding patterns have done for programmers. Some of patterns and their uses are given below in Table 1.

**Table 1:** Patterns and their uses

Patterns	Use
Client Proxy Server	Acts as a concentrator for many low-speed links to access a server
Adopter	Isolates code from technology-specific APIs
Reactor	Decouples event from its processing
Replicated Servers	Replicates servers to reduce burden on central server
Layered Architecture	A decomposition of services such that most interactions occur only between neighboring layers
Pipe and Filter	Transforms information in a series of incremental steps or processes
Subsystem Interface	Manages the dependencies between cohesive groups of functions

Example, Pipe-and-filter pattern is used where output from one component forms the input to the next. A typical example is the use of UNIX pipes. A layered pattern is used to focus on the different abstraction levels in a system, such as the software in a personal computer. A stack of boxes or a number of concentric circles is often used to represent a layered system as, shown in Fig. 8.



**Figure 8:** The layered pattern of a PC

## 5 DESIGN SUPPLEMENTS

Beside design model views and architectural patterns, there are some other issues that are considered during design of object-oriented software architecture. These are known as design supplements and are given below:

### 5.1 Skilled Levels

Software development process requires several types of skilled people who play important role in development of software system. The architect should suggest these skilled people and their skilled levels. Project manager initiates development process. System requirements are then collected from Subject Matter Experts (SME). SMEs are the people in the process who provide the information on what needs the system to be built. They serve in the most important role in the development process, despite not being the part of the permanent development team. In most cases they are often called client or user. Other important human beings are system analysts, architect, developers, testers, deployment managers and trainers.

### 5.2 Design Tools

Architect also confirms different design tools such as platform, programming languages, database, testing tools etc. to be used during software development.

### 5.3 Design Decomposition

Design decomposition is design abstraction, such as design-in-the-large, design-in-the-small and coding. Design-in-the-large considers mainly structuring, where design-in-the-small addresses functionality such as algorithms and data structures.

One can expand these design abstractions, because the terms design-in-the-large and design-in-the-small do not cover the increasing size and complexity of the systems with increasing demand on architecture focus. Design decomposition can be done as follows:

#### 5.3.1 Global System

Global system comprises several enterprises. The key issues addressed involve the impact of software that crosses enterprise boundaries. The global system can provide a set of standards and protocols that benefit organizations by allowing a general means of integrability and communication across different enterprises.

#### 5.3.2 Enterprise System

The enterprise system is the highest level within an organization. The enterprise system comprises multiple systems, where each system comprises several applications. The goal of the enterprise system is to provide software access through a consistent set of policies and services usable throughout the organization.

#### 5.3.3 A System

A system comprises several integrated applications. The applications provide the functionality where the system provides an infrastructure for the applications. The issues to address at this level are integrability, communication and coordination between applications. Access to data stores and management of inter-process resources occur at the system level. At a system, the applications use common functionality and data.

#### 5.3.4 Application

Applications typically involve numerous object classes, and one or more frameworks. At the application, the primary goal is to implement the end-user functionality defined by the requirements.

#### 5.3.5 Frameworks

In the framework, the goal is to allow the reuse of both software code and the design used in writing the code. An object-oriented framework is a semi-complete application. Programmers form complete applications by inheriting and instantiating parameterized framework components. A framework thereby provides an integrated set of domain-specific functionality.

#### 5.3.6 Objects

This is the lowest level, the code level, where in object-oriented programming; the classes and objects are defined and managed.

## 6 CONCLUSIONS & FUTURE SCOPE OF WORK

From the above work, it is concluded that architecture of object-oriented software system must contain domain characteristics, stakeholders' requirements, architectural model views,

architectural patterns and design supplements. In the above architecture, outline of architecture must be traced from problem domain. Based on the domain characteristics and stakeholders' requirements, the context of the architecture is called as domain-driven architecture. The above approach can be implemented by taking a case on domain-driven architecture of object-oriented software system.

## 7 REFERENCES

- [1] S. Ahmad and V. Saxena: Influence Factors for Software Architecture of Object-Oriented System, Journal of System Management (ICFAI Press, India) Vol. 5(2), pp. 35-43(2007)
- [2] C. Alexander, S. Ishikawa, M. Silverstein, M.: A Pattern Language, Oxford University Press (1977)
- [3] L. Bass, P. Clements and R. Kazman: Software Architecture in Practice- Second Edition, Pearson Education. (2003)
- [4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal: Pattern-Oriented Software Architecture - A System of Patterns, John Wiley (1977)
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns, Addison-Wesley(1994)
- [6] I. Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process, Pearson Education(1999)
- [7] I. Jacobson , M. Christerson, P. Jonsson and G. Overgaard : Object-Oriented Software Engineering, A Use Case Driven Approach, Addison-Wesley(1992)
- [8] P. Kruchten, H. Obbink and J. Stafford: The Past, Present, and Future for Software Architecture, IEEE Transactions of Software Engineering, 23(2), PP. 22-30. (2006)
- [9] P. Kruchten: The 4+1 View Model of Software Architecture, IEEE Transactions of Software Engineering, 12(6), PP. 42-50(1995)
- [10] P. Kruchten: The Rational Unified Process: An Introduction 3/e; Reading, MA, Addison-Wesley(2004)
- [11] B. Manfred: Architecture Driven Modeling in Software Development, Proceedings of the Ninth IEEE International Conference on

Engineering Complex Computer Systems  
Navigating Complexity in the e-Engineering  
Age( 2004)

[12] M. Shaw and G. David: Software Architecture,  
Prentice Hall(1996)

[13] M. Shaw and P. Clements: The Golden Age of  
Software Architecture, IEEE Transactions of  
Software Engineering, 23(2), PP. 31-39, (2006)