UNIT 5 ARCHITECTURE MODEL

Structure **Page No.**

5.0 Introduction
5.1 Objectives
5.2 Model
    5.2.1 Modeling
5.3 Architecture Modeling
    5.3.1 Aspects of Architecture Modeling
    5.3.2 Logical Architecture
    5.3.3 Physical Architecture
5.4 Implementation Diagram
    5.4.1 Component Diagram
    5.4.2 Deployment Diagram
5.5 Collaboration
5.6 Summary
5.7 Solutions/Answers
5.8 References/Further Readings

5.0 INTRODUCTION

We know that object-oriented concepts play an important role in the development of the software system. So, the question is that how do we apply these concepts efficiently? The answer is simple; we need to model using object-oriented concepts. Also, while designing the system, we need to apply the object-oriented concepts appropriately and efficiently to develop error-free software. Such modeling provides us with a goal-oriented way of analysing real-world problems. System modeling is required to better understand the assigned problems and understand the better coverage of the developing stages of the software system. To build any system, model provides us with a better template. If we want to visualise the structure or behaviour of the system, the model provides a blueprint of such a system.

This unit will cover the concepts of the architecture model, its importance, component, and deployment model using relevant examples.

5.1 OBJECTIVES

After going through this unit, you should be able to:

- explain the concept of system modeling,
- explain different aspects of architecture modeling,
- describe the difference between logical and physical architectural model,
- describe the concept of the implementation diagram, and
- differentiate the component and deployment diagram.

5.2 MODEL

A model is an abstract representation of a system, which promotes a better understanding of its representation. It helps to give better insight into the development of the system among communication of project team members. The model also gives

us an overview of the system's planning, design, testing, and implementation. A model is a way to represent something in the same or in a different medium. It always captures the important aspect of a thing and omits the unwanted attributes or components associated with it. We need a convenient medium to express the model. If you want to design a software system model, you need a modelling language such as UML. The model includes pictures as well as text for a better understanding of the model. Models are the best way to handle large software development more economically. They also deal better with complex systems. Such complex systems may be difficult to handle directly. By observing the model, you can analyse the possible impact of any change before developing the model. Selection of the best-suited model is important. It helps to solve the problems in a better way. Also, it helps in handling the related issues during the software development.

The best thing about the model is that it represents the abstract view of the required structure of a system. The model helps to provide a partial or full description of the system. You can obtain the full description of a system by carefully observing the abstract model of that system. Models may have different levels as per stakeholders' requirements. These different levels and forms serve different purposes. Relationships between models of different levels of abstraction are important. Two main classifications of the model are Static and Dynamic models.

Static features are easier to model as they do not include changes over time. The static model specifies the structure of the system using the object's operations and association. If you observe and analyse carefully, such structures exist in the problem statement itself. The static model encapsulates only those characteristics of the system which are independent of time and presents the state of a system at a given moment. It includes class diagrams, use case diagrams and composite structure diagrams. The dynamic model represents the object interactions during runtime and expresses the system's behaviour over time with the help of the system's dynamic behaviour. It includes sequence, activity, collaboration, and state chart diagrams.

5.2.1 Modeling

Modeling is a critical part of any developing software project. It always leads to the final deployment of a good software system. If you want to understand the system in a better way, you need the modeling of the system. Modeling helps to visualise and control the system's architecture. System modelling is the best possible way to manage the risk and explore your system's desired structure and behavior. For the different goals, designers build different types of models before constructing and developing the system.

Semantic information and notations are involved in the modelling process. Semantic involves the classes, relationships, states, and communication. The details of the syntactic structure can be obtained by using it because elements in this model carry the meaning of the model. The notations in modelling also play a significant role. They provide a visual representation and a better understanding of the model. Modeling is part of the larger environment which involves various modelling tools, languages, implementing constraints etc. It includes numerous information about the software system environment like project management, code generation etc.

According to Grady Booch "Modeling is a central part of all the activities that lead up to the deployment of good software. We build models to communicate the desired structure and behavior of our system. We build models to visualise and control the system's architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and reuse. And we build models to manage risk."

So we can summarise the Modeling as:

- Provides the visualisation of the system as per our needs,
- Help us to reduce the complexity of the system,
- Facilitate a template for constructing a system,
- Supports in the testing of the system before its implementation, and
- Provides communication with the end-users.

As a further extension of modeling concepts, we will discuss architecture modeling in the next section.

5.3 ARCHITECTURE MODELING

To understand the complete picture of the UML, various components can be linked in distinct ways, which are considered as a diagram. The main purpose of showing a diagram is to view the visual representation of the system.
The architecture of software deals with the design and implementation of the high-level structure of the software. It involves the association of various architectural components to show the correct functional and non-functional requirements of the system. For a better understanding of the architecture of a software system, there are different diagrams and models. Among these, architectural model has its own importance. Various modeling elements and techniques are required to describe the architectural model. According to Taylor et al, "An architectural model is an artifact that captures some or all of the design decisions that comprise a system's architecture. Architectural modeling is the reification and documentation of those design decisions. An architectural modeling notation is a language or means of capturing design decisions."

The success of object-oriented systems mostly depends upon the architectural model of the system. Architectural model represents the complete framework and outline of the complete software system. The architectural model contains both structural and behavioural diagrams of the system and can be defined at both a logical and physical level. The logical level shows the architecture's common view, whereas the physical level shows more details about how the software and systems are designed. It also includes detail about different technologies that exist within the organisation. In the architectural models, dependencies exist; for example, one system component, or package, is dependent on another.

We use a model composed of multiple views or perspectives to describe a software architecture. Architectural modeling provides a view that is important to solve various problems in a software system. This modelling typically includes a logical view, a process view, a physical view, a deployment view, and a use case view. Architectural modelling is centred on the idea of reducing the complexity of software through abstraction, separation of concerns, and reuse. There are numerous concepts which need to be modelled like components, connectors, interfaces, rationale etc.

5.3.1 Aspects of The Architectural Modeling

There are various aspects of architectural modelling. It focuses on that what do you model? These aspects are static, dynamic, functional, non-functional etc.

The collection of components represents the static aspect of an architectural model. They may form a component hierarchy. In this hierarchy, components are represented as sub-components and interactions between them. The static aspects are stable as they represent those parts of a system that forms the system's main structure. Static parts of the system are represented by classes, interfaces, objects, components, and nodes. Static aspects of a system do not change when a system runs or is executed i.e.

it does not involve system's behaviour while execution for e.g. topologies, assignment of components/connectors to hosts etc. It includes static diagrams like class diagram, component diagram and deployment diagram.

The dynamic aspect of the architectural model is represented by a collection of component and connection configurations which are controlled by nodes. The dynamic aspect is suitable for handling the runtime modification of the structure of a system. This aspect is concerned with the time and sequencing of the operations. Dynamic aspects emphasise the system's dynamic behaviour by showing collaborations among objects and changes to the internal states of objects. Note that dynamic aspects of a system do change during a system's runs. It shows the system's run time behaviour, e.g., state of individual components or connectors, state of a data flow through a system, etc. It includes dynamic diagrams like use case, sequence, activity, and state chart diagrams.

The functional aspect of the architectural model includes the system functions and the data-flows i.e. interaction between them. It defines how the functions will operate together to perform the system goal. The functional aspect is necessary in architectural modelling because it helps to identify the different functionalities in software systems and explores how software components interact among them and with the outer components of the system. The functional aspect describes what transactions the software product must carry out to satisfy the specified requirements. The functional aspect is used to support functional and performance test development. The functional aspect usually drives the definition of the other architectural views like information, concurrency, deployment, operational etc.

The non-functional aspect of the architectural model describes the system's operational capabilities and constraints that enhance its functionality. To accomplish the non-functional aspect of any architectural modeling is a difficult task. Finding the non-functional aspect and matching them with architectural concerns are more complicated than dealing with functional requirements. Non-functional aspects are the needs of a system to carry out its operations under constraints and quality expectations. Non-functional aspects are often captured less rigorously, as they are qualitative and subjective. Non-functional aspects are related to the quality of service for a software system. These aspects are usually very abstract and stated only informally. They refer to both observable qualities and internal characteristics of a software system. To satisfy non-functional aspects, the software architect applies architectural strategies to the system design, such as design patterns, layering techniques, etc.

5.3.2 Logical Architecture

If we want to understand the structure and organisation of the system's design, we use an architectural view which is also known as logical architecture or logical view. The logical architecture starts with the end user's functional requirements and provides a top-down abstraction of the system's overall design. A logical model provides the static view of the objects and classes of the system. Object-oriented systems intend for layered architecture. It is concerned with the functionality that the system provides to end-users. UML diagrams used to represent the logical architecture for object-oriented systems include class diagrams, object diagrams, sequence diagrams, activity diagrams, and state diagrams. Figure 5.1 shows the representation of the logical architecture of a system.

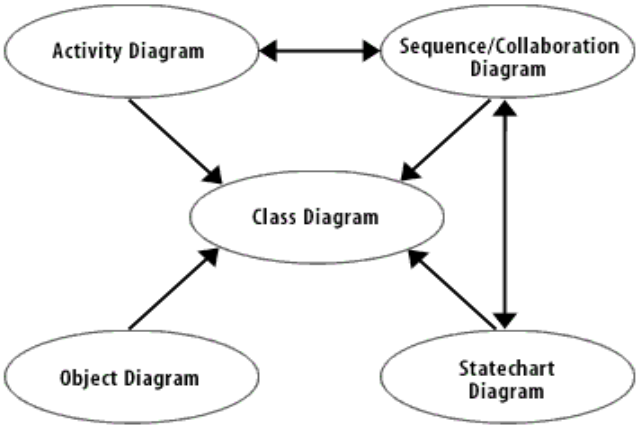

Figure 5.1: Logical Architecture of a system

The logical architecture consists of software classes and sub classes into different elements like packages, subsystems, and layers. It consists of the subset of the classes, subsystems and packages, and use-case realisations. The name, logical architecture is called due to there is no focus on how these elements are deployed across the software system. This architecture focuses only on the main components, their association, and data flow between them while structuring and grouping the elements into larger-scale modules.

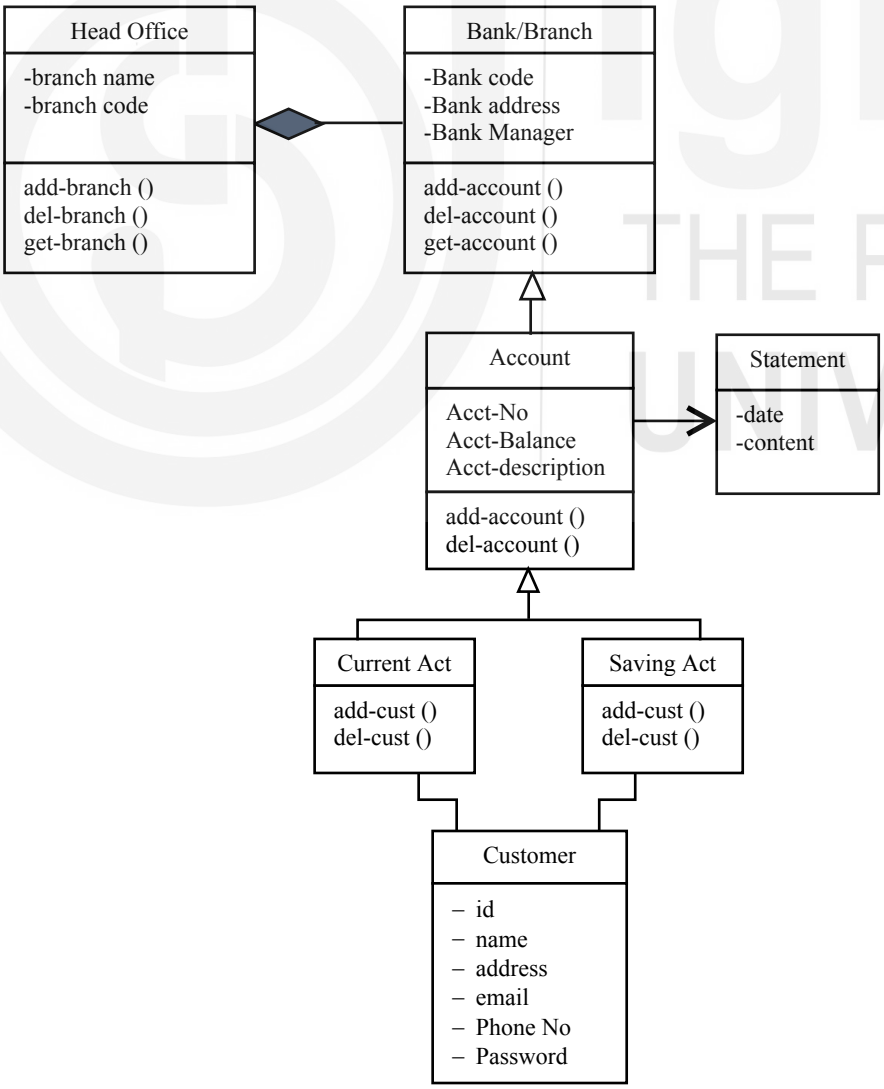

Figure 5.2: Logical architecture of banking management system as a class diagram

5.3.3 Physical Architecture

The physical architectural models are the systems' dynamic (run time) architecture. It shows the configuration of the topology of software components on the physical layer as well as the physical connections between these components. This model provides detailed information about how elements or components will be deployed across the system infrastructure. It maps the software elements and artifacts to its platforms, including the hardware elements (nodes), the network, and the supporting software platforms. This model focuses on detail about network resources, server configuration, hardware information and other relevant information which are required to develop and deploy the software system. The main objective of modeling physical architecture is to satisfy the proposed logical architecture elements and defined requirements of the system. This architecture supports the physical view of the system, and the deployment diagram of UML is used to represent it. Such a diagram is very beneficial because it clarifies which packages should be installed on which nodes. Basically, the physical architecture helps the system designers and administrators recognise the software's physical locations, linkage between nodes, deployment, and scalability.To implemention of physical architecture, you will study it in detail via implementation of the deployment diagram in subsequent sections by using real-life examples.

☛ Check Your Progress 1

1) What is a model? Why they are important for a software system.

……

……

……

……

2) What are the basic principles of modeling? Explain briefly.

……

……

……

……

3) Explain the use of modeling for the development of an efficient software system.

……

……

……

……

4) Describe how the nature of components, the significance of the links, and the layout play an important role in the architectural model?

……

……

……

……

5) How can you say that an architectural model contains both structural and behavioural diagrams of the system and can be defined logically and physically? Explain.

……

……

……

……

6) Differentiate the following with respect to the aspects of architectural modelling.

a. Static and dynamic aspects

b. Functional and non-functional aspects

……

……

……

……

Now, you are familiar with architectural modeling. In the next section, we will explore how to implement architectural modelling by using suitable UML diagrams. Now, let us discuss the basic concept of the implementation diagram.

5.4 IMPLEMENTATION DIAGRAM

The diagram, which is closely correlated with the Use Case, is the implementation diagram. The purpose behind this coupling is to highlight which types of design elements will implement the functionality of Use Case in a novel system. This diagram provides the physical environment of our system. The implementation view of a system incorporates the components that are used in the physical system. Using this view, one can address the various components of the system which are required for the configuration of the system. The implementation diagram is very much helpful in the physical implementation of the design. To illustrate the details of implementation, the most used diagrams are the component diagram and the deployment diagrm. Both diagrams are part of structural modeling. These diagrams are explained in the following sections.

5.4.1 Component Diagram

Component diagrams are used to show how the physical components in a system have been organised. It shows the set of components and their relationships in the system. It helps in understanding if the system's functional requirements are covered by planned development. The relationship among components are called dependencies. This diagram models the physical aspect of the system, including software components, source code, runtime code, & executables. To provide the details about a system's static design implementation view, such diagrams are very helpful. it represents the high-level parts that make up the system. They help us frame large systems by using smaller components and become necessary when designing and building complex systems. According to R. A. Maksimchuk and E. J. Naiburg, it is important to develop software based on multiple smaller parts (components), which you can use to assemble the overall system. This enables you to reuse software components instead of writing them all from scratch. They further explained and suggested that you can use a component diagram and components to model the architecture of different parts of your system. You can also use them to model the application internals, as we just discussed, as well as to view how different applications work together.The component diagram also represents the collaborations and internal structure of components. These components may be nested and are present as internal structure in components diagram along with class and its interfaces. Component diagrams commonly contain components, interfaces and dependency, generalisation, association, and realisation relationships, which also help specify the components' details for constructing an executable system.

Graphically, a component diagram is a collection of vertices and arcs. The essential components of component diagrams are :

- components,
- interfaces
- ports and
- connectors

Components-

In the component diagram, the component represents a reusable piece of software and is the physical and replaceable unit of the system. According to G. Booch, at the lowest level, a component is a cluster of classes that are themselves cohesive but are loosely coupled relative to other clusters. He suggested that each class in the system must live in a single component or at the top level of the system. Components are represented as rectangle with tab having the name of the object in it. A component may also contain other components. The components do not directly depend on other components but on interfaces. Dependencies may also exist among the components, which cause changes in components due to changes in one or more components. It represents by a dashed line in the component diagram. The component's name should be unique in the specified component diagram.

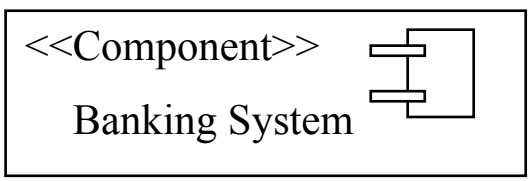

Figure 5.3: Component

Interfaces-

Components of the system use interfaces to communicate with each other. It defines the component's interaction details. It is a collection of operations that are used to specify a service of a class or a component. The interfaces are shown in the ball-and-socket notation or as a typical class with the stereotype of <<interface>>. In interfaces, the ball notation is used to specify the component's functionality. A component collaborates with other components through well-defined interfaces that specify the system's functionality to the environment. If a component provides service through interface to other component, then its called a provided interface, and it is shown as a straight line from the component box with an attached circle (ball). If a component receives the services from other components through the interface, it is called as a required interface, and it is shown as a straight line from the component box with an attached half-circle (socket).

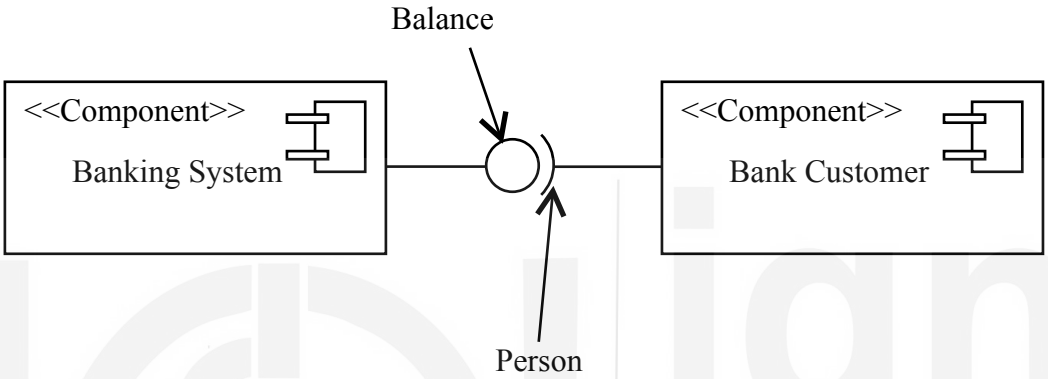

Figure 5.4: Interface

Ports-

Ports are used by the component for its interactions with its environment and also to control the implementation of all the operations in the component. Along the edge of the component, a port is shown as a small square. Ports provide encapsulation to the structured classifier. Ports have public visibility, but components may also have hidden ports. Interfaces may be linked to the port, but one-to-one relationship between ports and interfaces is not required. Generally, ports are unnamed, but you can provide the a name as port name: Port Type.

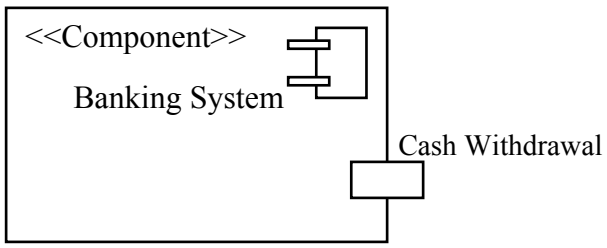

Figure 5.5: Port

Connectors-

A connector is used to show a link between the ports of components. It also specifies the communication between two or more classifiers. If one port provides the interface required by the other port, they can be linked together. There are mainly two types of connectors: the viz delegation connector and the assembly connector. Delegation connector links together the port of a part and port of a component, thus providing the

interface. Delegation Connectors are used when ports inside the composition need to be connected with ports outside the composition.

Assembly connector: It connects two or more components that define services provided by one component to another.

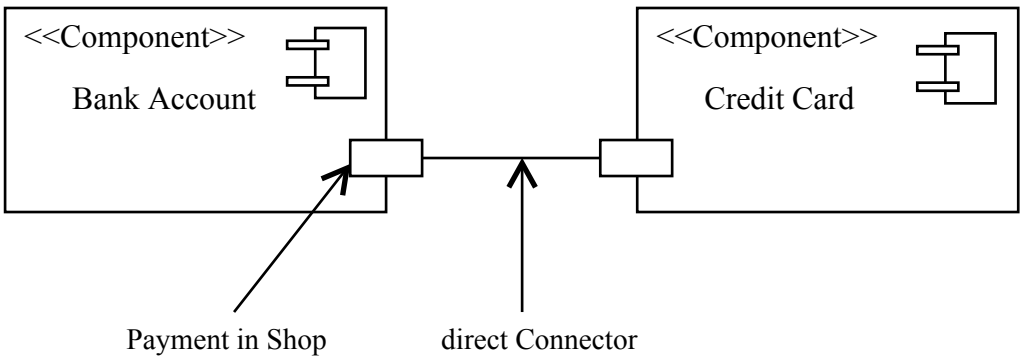

Figure 5.6: Connectors

Example: Component diagram of ATM System

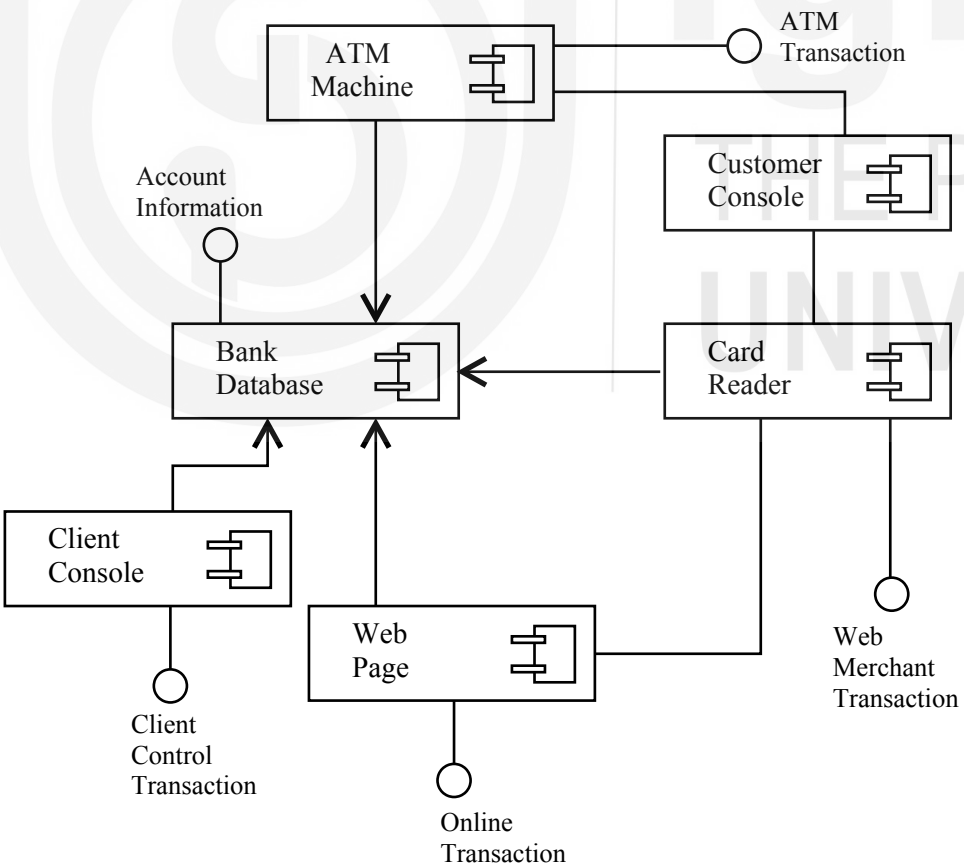

Figure 5.7: Component diagram of ATM System

5.4.2 Deployment Diagram

To address the static deployment view of architecture, we use the deployment diagrams. Deployment diagrams are related to component diagrams in which a node typically encloses one or more components. Deployment diagrams are used during development of the system. In this diagram we use to indicate the physical collection of nodes that serve as the platform for execution of our system. Before execution, developed components of a software system must be deployed on some set of hardware. Deployment diagram it represents the configuration of runtime processing nodes and the components that live on them. A deployment diagram consists of nodes and their relationships where a node may consists of one or more artifacts. This diagram helps us in the allocation of artifacts to nodes in the physical design of a system. It also helps us visualise, specify, and document embedded, client/server, and distributed systems. The three main elements of a deployment diagram are artifacts, nodes, and their connections.

Artifacts

An artifact represents physical entity that is used in software design or produced by software development process. As artifacts are physical entities deployed on nodes, devices, and executable environments, it is the source of a deployment to a node. Artifacts can be involved in associations with other artifacts, such as a dependency of one artifact on another artifact. Dependency between artifacts is notated as a general dashed line with an open arrowhead directed from client to supplier artifact. The notation for an artifact consists of a class rectangle containing the name of the artifact, the keyword label «artifact», and an optional icon that looks like a sheet of paper with the top right-hand corner folded over.

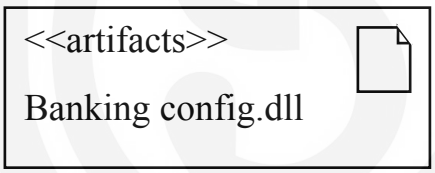

Figure 5.8: Artifact of Banking System

Node

Node, just like component, is an important building block in modeling the physical aspects of a system on which artifacts are deployed for execution. It exists at run time and represents a computational resource, generally having memory and processing capability. By using a node, we can easily model the topology of the hardware on which the system executes. Components may be deployed on nodes because nodes generally represent a device and execution environment. A device is a piece of hardware capable of executing programs and can have a list of processes on it, such as a computer, a modem, or a sensor. A device is a piece of hardware incapable of executing program. At the same time, the execution environment is software that provides for the deployment of specific types of executing artifacts. Execution environments are typically hosted by a device. In the deployment diagram, a node is represented as a cube with the name of the object in the deployment diagram. The node must have a name in the diagram, and it should be unique i.e., different from another node.

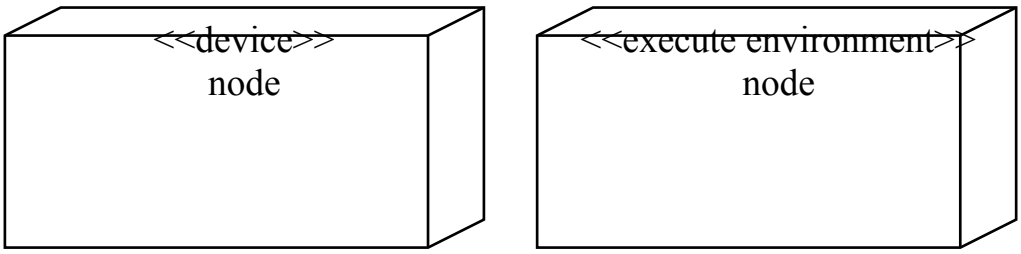

Figure 5. 9: Nodes

Connections

Communication is required among nodes in the deployment diagram. Nodes communicate with each othe, via messages and signals, through a communication path indicated by a solid line. This line is known as a connection. It represents the communication path used by the hardware to communicate with nodes. Communication paths may be unidirectional or bidirectional. Each communication path may include an optional keyword label that provides information about the connection. With the help of connections in nodes, we may represent the association, dependency, generalisation, and realisation. An association represents a physical connection or link among nodes using a solid line. We may use associations to model indirect connections, such as a satellite link between distant processors. Dependency indicates that a node dependent on another node using a dashed line. Generalisation is a relationship between a parent node and child node using a solid line with a triangle between nodes. The realisation is a relationship between interface and classes or components that realise them using a dashed line with the hollow triangle.

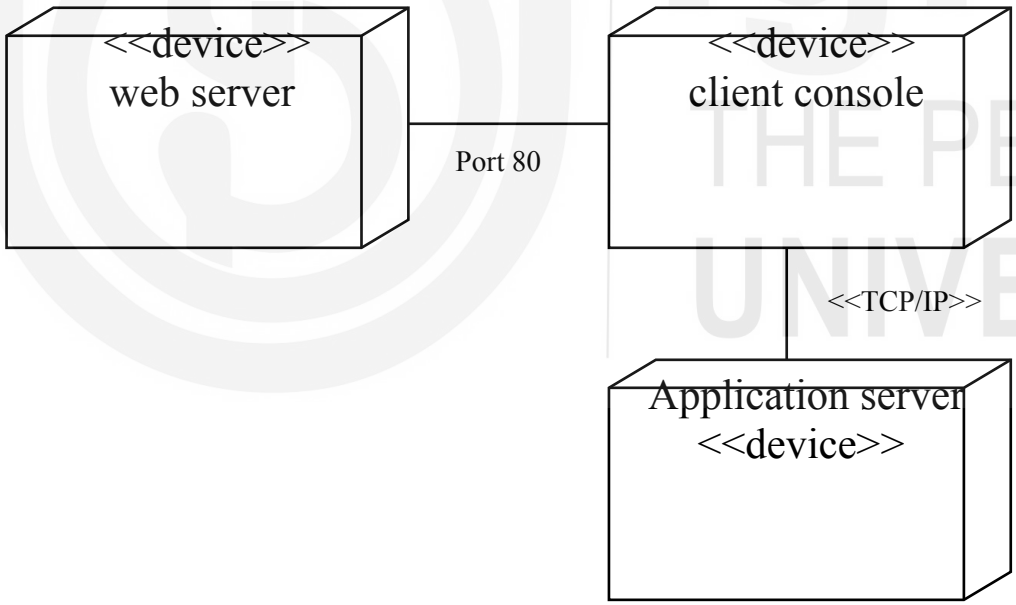

Figure 5.10: Connections

Example: Deployment Diagram of Banking Management System

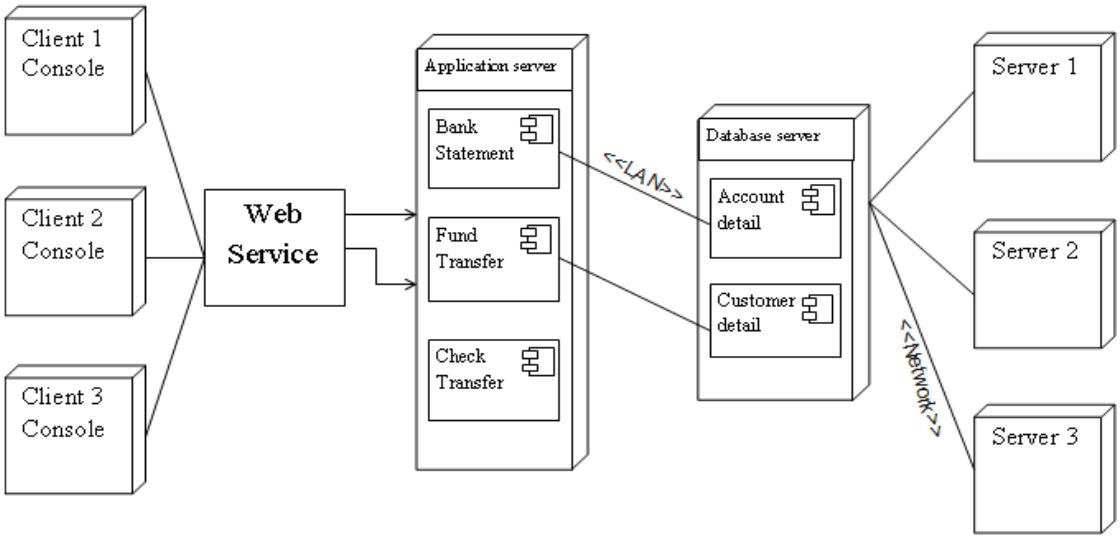

Figure 5.11: Deployment diagram of Banking System

5.5 COLLABORATION

Collaboration defines an interaction and describes how the elements work together to provide the system's behavioural requirements so that it must be larger than the combination of all elements. Here, the combination of elements may be classes, objects, interfaces, components, nodes, use cases or other elements. Even collaboration describes the dynamic interactions of classifier instances though it also has both types of dimensions, structural as well as behavioural. The structural dimension focuses on how elements work together, and the behavioural dimension focuses on the run time interaction of these elements. So, it covers both aspects, static as well as dynamic. Collaboration plays an important role in constituting a system by specifying the role of classifiers, structure of collaborating elements, operations, required functionality, etc. The main objective of the collaboration is to describe the working of the system, so it includes only important aspects of those elements which are helpful to define the system. Embedded collaborations may exist, and this nesting supports the abstraction concept.

We express the details about collaboration by using the collaboration diagram, which is part of the interaction diagram. Collaboration is represented in the form of the ellipse with dashed lines in its graphical form. It represents cooperating entities and connectors. Collaboration acts like a template in which roles are defined for classifiers, and during runtime, these classifiers are bound to their respective roles joined by the connectors. The entities show collaboration characteristics, and the connectors show the communication path between entities. Collaboration has a name, and it must be unique within its enclosing package. Roles are also marked with a name and their type. The role name is for a specific classifier instance, and the type is a constraint for this instance of classifier without considering its internal design structure or implementation detail.

With the help of a collaboration diagram, you can wire all components together, and complex systems can be represented in a simple illustration. In the collaboration diagram, we define the structure of collaborating elements which is known as role. This role is played by the instances. This diagram represents the association and

interaction among these instances also. The interaction or message passing is required to obtain the certain required result known as a task. In the collaboration diagram, we represent the objects by using the box. The messages are represented by using numbering on them. The following diagram represents the collaboration diagram for a banking system.

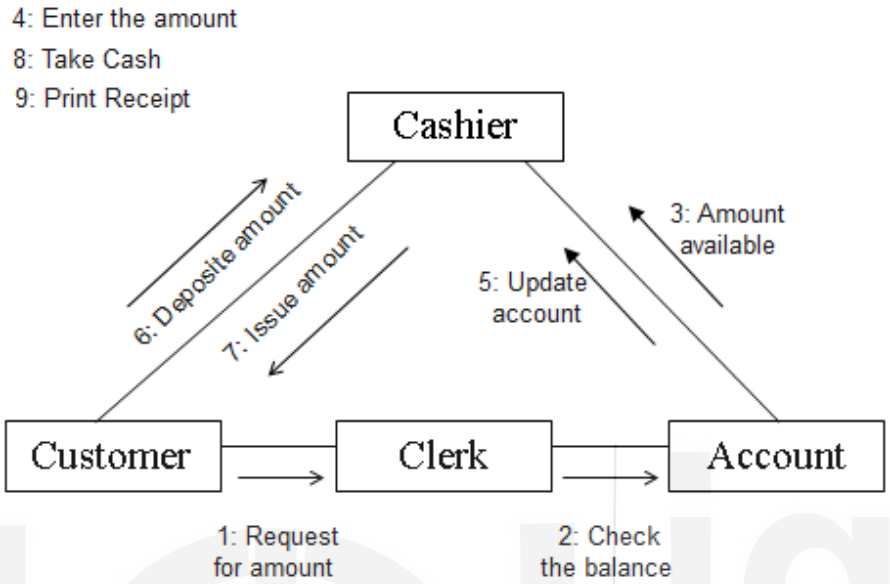

Figure 5.12: Collaboration Diagram for Banking System

☞ Check Your Progress 2

1) What do you mean by component? How does it help to implement the system?

……

……

……

……

2) Give the main advantages of the component diagram.

……

……

……

……

3) Draw the component diagram for the online banking system.

……

……

……

……

4) What role does physical architectural play in the design process of a software system?

……

……

……

……

5) Write the main advantages of the deployment diagram.

……

……

……

……

6) Draw the deployment diagram for the ATM system

……

……

……

……

5.6 SUMMARY

The model helps us to give a better understanding of the development of the system.It also helps in communication among project team members. The static model specifies the structure of the system using object operations and association. The dynamic model focuses on the system's dynamic behaviour to express the system's behaviour over time. Modeling includes the numerous information about the environment of software system like project management, code generation etc. The architectural model represents the complete framework and the outline of a complete software system that contains both structural and behavioural diagrams of the system. The logical model is a static view of the objects and classes that make up the design/analysis space and leans on patterns the most. The model provides details about how elements or components will be deployed across the system infrastructure. The implementation diagram is very much helpful in the physical implementation of the design. On the other hand, component diagram models the physical aspect of the system, including software components, source code, runtime code, & executables. Deployment diagrams are related to component diagrams in that a node typically encloses one or more components. In the collaboration diagram, collaboration acts like a template in which roles are defined for classifiers, and during runtime, these classifiers are bound to their respective roles (instances) joined by the connectors. This diagram represents the association and interaction among these instances also. This unit explains the architectural modelling and deployment diagram in detail.

5.7 SOLUTIONS / ANSWERS

Check Your Progress 1

1) Model is way to represent something in same/different medium. It provides abstract representation of a system, which promotes the better understanding by its representation. It helps to give better insight about the development of system among communication ofproject team members. Model also gives us the overview of planning, design, testing and implementation of the system. Each model within a design describes a specificaspect of the system under consideration. Model is important because it provides the information about organisation of the system and its be behavioural aspect. A model shows the blueprints of a system for better visualisation of the problem. With the help of model, you can easily understand the complexity level of any system in simplest way.

2) Model is a simplified, complete, and consistent abstraction of a system, created for better understanding of the system. selection of model is very important. It helps to analyse the model and guide to provide the respective solution. Model should be able to provide the complexity detail. It may be expressed at different levels of abstraction. Model should be able to fulfil the gap between analysis and design of the system. To provide the efficient solution of any problem, no single model is sufficient. We need to visualise the problem domain as per different model view.

3) Modeling helps to develop efficient software system. It provides the overall template of system to be design. It provides the abstract representation of the system and help development team to deal with the complexity of the problem. Modeling is helps us to understand the various aspects of software system, its processes, its organisation and association among different components and elements. Proper selection of model ensures the best development of software system even its complex in nature.

4) Components play a very important role in the architectural model. The main benefit is the reusability because we can use them wherever we require in the software system. Components also have the capability of encapsulation. Links are also significant because they involve the association of various architectural components to show the system's correct functional and non-functional requirements. For a better understanding of the software system's architecture, the layout of different diagrams and models is also very significant.

5) As you know, the architectural model represents the complete framework and outline of the complete software system because this model has both the static and dynamic aspects of the system. The static aspect is related to the structural model that forms the main structure of the model. It focuses on the system's static structure using objects, attributes, operations, and relationships. By this model, you can obtain the significant concepts which are related at the abstraction and implementational levels. The behavioural model is linked with the dynamic aspect and contains the changeable components of the model. So, by using an architectural model, we can easily represent the structural and behavioural models. The logical level shows the common view of the architecture, whereas the physical level shows more details about how the software and systems are designed.

6)

a. The static aspect of the architectural model is represented by the collection of components. In this aspect, components are represented as sub-components, including the interaction between them. Static aspects of a system do not change as a system runs. Static aspects represent those parts of a diagram which are stable. Such aspects are represented by classes, interfaces, objects, components, and nodes.

   The dynamic aspect of the architectural model is represented by the collection of component and connection configurations that are controlled by nodes. This aspect is concerned with the time and sequencing of the operations. The dynamic aspect is suitable for handling the runtime modification of a system's structure and focuses on the system's dynamic behaviour. Such aspects are represented by use case diagram, sequence diagram, activity diagram and statechart diagram.

b. The functional aspect defines how the functions will operate together to perform the system goal. This aspect describes the software product's transactions to satisfy the specified requirements. It is used to support functional and performance test development. The Functional aspect usually drives the definition of views like information, concurrency, Deployment, operational etc.

   The non-functional aspect of the architectural model describes the system's operational capabilities and constraints that enhance its functionality. Non-functional aspects are the needs of a system to carry out its operations under a set of constraints and quality expectations. These aspects are usually very abstract and stated only informally.

Check Your Progress 2

1) The component represents a reusable piece of software and is the physical and replaceable unit of the system. A component may also contain other components. The components do not directly depend on other components but on interfaces. Dependencies may also exist among the components, which cause changes in components due to changes in one or more components. The component's name should be unique in the specified component diagram. This diagram helps to model the implementation detail of the system and ensure that system is functioning as per requirement or not.

2) The main advantages of the component diagrams are as follow:

   a. It represents the arrangement of the various components of the system and how they are associated with each other.
   b. It helps to understand the functionality of the system caused by the interaction of components with each other.
   c. It helps to visualise the dynamic behaviour of components.
   d. It helps to develop the component-based software system by focusing on the association between components and their dependencies.
   e. It helps to visualise the physical structure of the system and the behaviour of its services to the interfaces.

3)

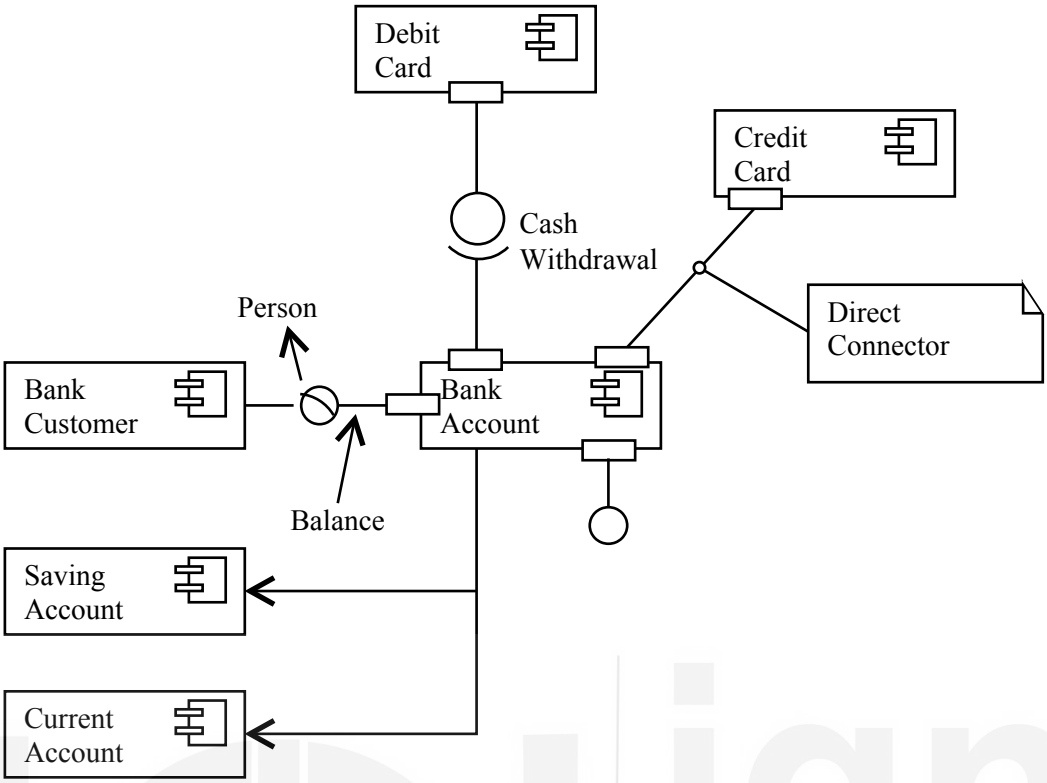

Figure 5.13: Component diagram for the online banking system.

4) A physical architecture model provides the details about how components will be arranged in the system. It shows the arrangement of physical elements that provide the solution for a product, service, or enterprise. As you know, the software design process describes a system that will be able to accomplish the established tasks and provides a base for developing software structure. The physical architecture model accomplishes this structure. Using this model, you can easily identify the major components like modules, architecture, data etc., based on the provided requirements along with their responsibilities and the association among them.

5) The main advantages of the deployment diagrams are as follow
   a. It helps to visualise the hardware components and elements of the system that are required to be physically deployed on the hardware.
   b. Helps to understand the execution of artifacts used by participating nodes i.e. physical hardware during the run time of the system.
   c. It helps to model the details of the hardware topology of the system.
   d. It is used to describe the distribution and relation of components in the physical architecture.

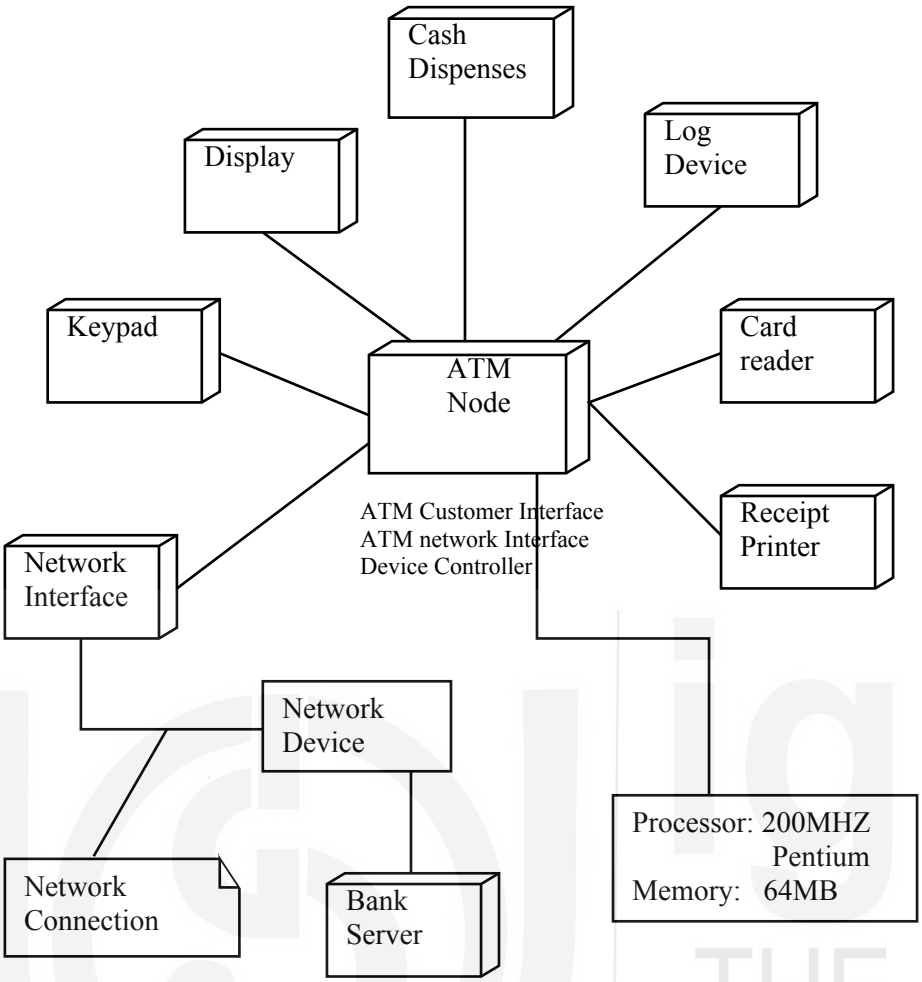

Figure 5.14: Deployment diagram for ATM system

5.8 REFERENCES/FURTHER READINGS

- Grady Booch, James Rumbaugh and Ivar Jacobson, " The Unified Modeling Language User Guide", 2nd Edition, Addison-Wesley Object Technology Series, 2005.
- Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston, "Object-Oriented Analysis and Design with Applications," 3rd Edition, Addison-Wesley, 2007.
- James Rumbaugh, Ivar Jacobson, and Grady Booch, "Unified Modeling Language Reference Manual," 2nd Edition, Addison-Wesley Professional, 2004.
- John W. Satzinger, Robert B. Jackson, and Stephen D. Burd, "Object-oriented analysis and design with the Unified process," 1st Edition, Cengage Learning India, 2007.
- Brett McLaughlin, Gary Pollice,and Dave West, "Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D," Shroff Publisher,First edition,2006.