Unit IV

CHAPTER 4

User Application Analysis : System Design

University Prescribed Syllabus

Application Analysis: Application interaction model; Application class model; Application state model; Adding operations. Overview of system design; Estimating performance; Making a reuse plan; Breaking a system into sub-systems, Identifying concurrency; Allocation of sub-systems; Management of datastorage; Handling global resources;

Choosing a software control strategy, Handling boundary conditions; Setting the trade-off priorities; Common architectural styles; Architecture of the ATM system as the example.

Specific Instructional Objectives :

At the end of this lesson the student will be able to :

- Explain the software system performance estimation process.
- Define reuse and reuse plan.
- Make a reuse plan.
- Explain the organization of a system into subsystems.
- Explain what a concurrent inherency is.
- Identify concurrency inherent in the problem.
- Explain subsystems allocation to hardware platforms.
- Define data store management.
- Manage data stores.
- Handle global resources.
- Set trade-off priorities.
- Explain different architectural styles.
- Draw and explain a component diagram.
- Draw and explain a deployment diagram.

4.1 Estimating System Performance

GQ. Write a short note on: System performance estimation.

Tentative performance estimation should be done in system planning phase while developing a new software system.

☞ **This kind of performance estimation of a software system is known as a "back of the envelope" valuation.**

- The performance estimation process should be fast and progressive in nature. The basic need for the performance estimation is that, the software developers should try to simplify the assumptions made during the requirements analysis phase of the software development.
- The main agenda behind the performance estimation of a software system is to determine the feasibility of a software system and it is not at all dedicated for attaining the greater accuracy in the final outcome.
- Basically, the software system performance estimation process comprises of two activities : prediction, approximation and estimation.
- That is, software developers should first of all predict and approximate the performance of a proposed system which is then followed by the thorough system performance estimation process.
- For the simplification of a performance estimation process, let us consider an ATM system example. Assume that, a particular network service provider provides an ATM network service to a bank having around 500 branches all over the country.
- Suppose there are an equivalent number of workstations or terminals in superstores and other markets. Suppose on a busy days; half of the workstations or terminals are abundantly working at once.
- Assume that, each purchaser takes at least one minute to accomplish a single session and maximum transactions initiated by customers encompass a single withdrawal or deposit.
- System designers can estimate an ultimate prerequisite of about 500 transactions per minute or per second. This may not be accurate, but it indicates that a system do not necessitate extraordinarily fast, progressive and advanced computer hardware.
- The computer hardware would become a giant dispute in case of online transactions. Software system developers can accomplish same estimate for the purpose of information storage. In this case, system designers should first of all count the number of customers involved in the particular scenario which is then followed by estimation of the information required by each customer.
- The necessities of information storage for the bank activities are supplementary and simple in comparison to the storage required for ATM computing power.
- In case of satellite based systems, the circumstances would be different where information storage and access bandwidth would be the crucial architectural disputes.

4.2 Making a Reuse Plan

GQ. Define Reuse plan. Explain in brief the following terms with respect to Reuse plan :
(a) Library (b) Framework (c) Pattern

- Reuse plan is considered as one of the key advantage of the object oriented technology but the reuse of system components or system itself does not happen unexpectedly. Reuse should be well planned.
- Two possible facets of reuse are :
  - With the help of existing things
  - By means of producing reusable new things.
- The reutilization of existing things is easier than the proposition of new things. Only the constraint is somebody must have designed the reusable things in the past so that, designers can make use of them for their present purpose.
- Only a limited number of software system developers produce new things for their use and rest of the system developers recycle the existing things. Reusable things contain libraries, models, frameworks and patterns.
- Reuse of model is the most practical form of reuse. The logic in a model can be applied to several problems.

4.2.1 Libraries

- A library is a collection of classes that are convenient and valuable in various circumstances. The collection of classes should be systematized and arranged judiciously in such a way that the system users can find them.
- Thorough and profound work is essential for good organization of contents or things involved in a particular scenario and sometimes it becomes challenging to reside all the things in place.
- Moreover, the classes must have precise and exhaustive explanations in order to assist end users for determination of their significance.
- Characteristics of good class libraries given by *Korson* are :

1. Efficiency	2. Genericity	3. Consistency
4. Completeness	5. Coherence	6. Extensibility

  - ***Efficiency*** : A library should offer alternative implementations of algorithms that trade space and time.
  - ***Genericity*** : A library should use parameterized class definitions where appropriate.
  - ***Consistency*** : Polymorphic operations should have consistent names and signatures across classes.
  - ***Completeness*** : A class library should offer complete behavior for the chosen themes.
  - ***Coherence*** : A class library should be organized about a limited, well-focused themes.
  - ***Extensibility*** : The user should be able to define subclasses for library classes.

- While integrating two or more class libraries from several sources, some problems mentioned below may arise.
  - Methods involved in one library can yield error codes to the calling routine.
  - Class libraries make use of several error handling mechanisms.
  - A class name collision is also one of the problem involved in class library.
  - Procedure-driven control and event-driven control can be assumed by different software system applications on the basis of their prerequisites. In this case, the key problem is we cannot combine both types of user interfaces in a particular software system application.

4.2.2 Frameworks

- Basically, a framework describes the broad approach to solve a particular problem. In short, a framework describes a complete architecture of a software system application.
- Hence, a software framework is used in order to design and develop a complete software system application and it can be expanded by software designers.
- As far as software modeling and design is concerned, frameworks are categorized into two broad categories : **Black Box frameworks** and **White Box frameworks**.
- In case of a Black Box framework, the internal structure of a software program is hidden from end user. End users can access only the general description of a software program.
- In White Box framework, software designers or end users can view all of the components and overall architecture of a framework.

4.2.3 Patterns

- A pattern is simply nothing but an established and confirmed solution to a general problem.
- Several patterns are meant for number of phases of the software development lifecycle. There are exclusive patterns for analysis, architecture, design and implementation phases of SDLC. Reuse can be attained by means of existing patterns.
- An individual pattern comes with rules and guidelines on when to use it, as well as trade-offs on its use. One of the major benefits of pattern is that a pattern has been judiciously considered by others and has already been applied to ancient problems.
- Subsequently, a pattern is more likely to be correct and robust than an unapproved and unverified custom solution. Correspondingly when a particular pattern is used, we tap into a language that is accustomed to many software system developers.
- A pattern is quite different from a framework. A pattern is usually a small number of classes and relationships. In contrast, a framework is much wider in scope and covers an entire subsystem or application.

4.3 Organizing a System into Subsystems

GQ. How can we organize a system into subsystems? Explain in detail.

- The very first step in software system design is to divide the system into number of pieces for better understanding of the software system scenario. Each key fragment of the system is known as a ***subsystem.***
- Each subsystem is based on some common theme, such as similar functionality, the same physical location, or execution on the equivalent kind of hardware.
- For instance, a spacecraft computer system might contain subsystems for life support, navigation, engine control, running scientific experiments and other activities.
- A subsystem is a group of classes, associations, operations, events and constraints that are unified, incorporated and have well-structured and defined small interface with other subsystems. A subsystem is typically acknowledged and approved by the numerous services provided by it.
- A service is a cluster of related functions that share some mutual purpose, such as I/O processing, drawing pictures or performing arithmetic.
- A subsystem articulates a rational technique of looking at a piece of the problem. For instance, the file system within an operating system is a subsystem that encompasses a set of associated abstractions that are generally independent of abstractions in other subsystems such as process management and memory management.
- Each subsystem has well-structured and defined interface to the rest of the system.
- Layers or Partitions can be used in order to divide the complete software system into subsystems in horizontal or vertical manner respectively. That is, software designers can organize the subsystems horizontally by means of layers and vertically by means of partitions.
- In both of these cases, all layers or partitions involved within a software system are interrelated with each other and interrelationships amongst these layers or partitions is of the type client server relationship.
- The key difference between partitions and layers is that, the subsystems defined with the help of partitions will have identical and same levels of abstraction whereas the subsystems defined by means of layers will have different levels of abstraction.
- One more difference is partitions are having peer to peer relationship amongst them and layers are having client server relationship in between them.
- Consequently, each partition can be considered as a self-governing peer and each layer is dependent on some other layer or group of layers.
- Furthermore, we can combine partitions and layers in order to get a software system as a whole. In conclusion, we can layer the partitions and partition the layers.

4.4 Identifying Inherent Concurrency in a Problem

- The state model is dedicated for detailed description of sequence of tasks and services involved within the system operations and do not deals with the contents like details of operations and services, how different services and operations are executed during implementation and execution of the proposed system, etc.
- The state model describes those aspects of objects concerned with time and the sequencing of operations. When two objects can get the events simultaneously deprived of interfacing, they can be said inherently concurrent.
- As far as implementation is concerned, all of the software objects are not concurrent, since one processor can support many objects. The state model is basically articulated by means of a state diagram.
- So, with the help of a state diagram, several objects involved in a scenario can be congregated into a solitary thread of control. A thread of a control is an end to end path through a set of state diagrams of discrete objects and only single object can be active at a particular instance of time.
- For instance, in case of an ATM system, the ATM machine is idle; when the bank is authenticating and confirming an account or processing bank transaction.
- If the central computer system proximately controls the ATM, the bank transaction object and an ATM object can be associated with each other as a solitary task.

4.5 Allocation of Subsystems to Hardware

- The necessity of advanced or superior performance basically forms the basis for the decision to use various processors or hardware functional units. The number of essential processors required totally depends on the speed of the machine and the volume of calculations.
- For instance, a parallel computing system produces too much data in a very short time span in order to handle in a solitary central processing unit. Many parallel machines should abstract the information or data handled within a system well before analysing a threat.
- The software system designer should assess and calculate the mandatory CPU processing power by means of computing the stable state load as the product of the number of transactions per second and the time required to process the particular transaction. The estimate will ordinarily be inaccurate or imprecise.
- The estimate should be increased in order to tolerate the transient effects due to random variations in load in addition to the synchronized bursts of activity.
- The sum of surplus capacity required depends on the adequate rate of failure due to inadequate resources.
- In case of an ATM system example, the ATM machine is mainly responsible for actual online transaction processing and providing a user interface to customer for communication and coordination amongst customer and ATM system.

- A solitary central processing unit is sufficient for an individual ATM machine. The main server of a bank can be considered as a routing machine fundamentally since it receives ATM requests and communicates them to proper bank computer situated at a particular branch of a bank.
- Sometimes, a large network may comprise of multiple processors in case of an ATM system scenario. The computer systems situated at the bank branch accomplish data processing and encompass database applications.
- Each device is an object that operates synchronously along with other objects these other objects may be hardware units or devices involved in a system.
- The software system developers should decide and agree upon the fact that, which subsystems will be implemented in software and which will be executed in hardware. Two reasons behind implementing subsystems in hardware are cost and performance.
- Much of the trouble of designing a software system comes from accomplishing externally enforced software and hardware constraints. Object oriented design affords no magic solution however the external packages can be demonstrated as objects.
- The software system designers should take into account the cost, compatibility and performance issues. Correspondingly, the system designers must think about flexibility for future modifications or alterations. The future modifications can be future software product versioning or improvements and software product design variations.
- In case of an ATM system example, no such performance issues are occurred.

4.6 Data Storage Management

GQ. Explain data storage management in software modeling.

- Number of substitutes are available for managing the data storage that we can use distinctly or in combination. For instance, databases, files, data structures, etc. Different types of data stores offer trade-offs between cost, access time, capacity, cost, reliability and access time.
- For instance, a personal computer system application can make use of files and data structures as per its requirement, an accounting can use a database to connect subsystems. Files are simple, modest, permanent and inexpensive.
- Implementations for sequential files are ordinarily standard however storage formats and commands for random-access files and indexed files fluctuate.
- Following kind of data is appropriate and can be suitable for files :
  - Sequentially accessed data.
  - Data that can be wholly read into memory.
  - Data with low data density.
  - Uncertain data with modest structure.
  - Data with high volume.
- Databases are the other type of data store which are typically managed, monitored and controlled by means of database management systems. Several types of database management systems like relational databases and object oriented databases are available.

- Following kind of data is appropriate and can be suitable for databases :
  - Larger amount of data that should be handled proficiently and skillfully.
  - Data that synchronized the updates through transactions.
  - Data that needs updates at satisfactory levels of detail by several users.
  - Data that must be accessed by numerous application programs.
  - Data that should be protected against malicious access.
  - Data that is long-lasting and highly valuable to an organization.
- The software system designers must consider object oriented database management systems merely for specialty domain applications that have a wide variety of data types or that must access low level data management primitives.
- These software system applications can be multimedia applications, embedded system applications, different engineering applications and many more.
- The software system designers should make use of relational database management systems for the software system applications that requires databases since the features of relational database management systems are adequate and satisfactory for utmost software system applications.
- Furthermore, if relational database management systems are used appropriately, they can offer a super execution of an object oriented archetype.

4.7 Global Resources Handling

GQ. Explain in brief : Global Resource Handling.

- The software system designer should recognize the global resources. The categories of global resources are mentioned in Table 4.7.1 below :

Table 4.7.1 : Types of global resources

Types of Global Resources	Examples
Physical Units	Processors, tape drives, communication satellites.
Space	A workstation screen, the buttons on a mouse.
Logical Names	Object IDs, filenames, class names.
Access to Shared Data	Databases.

- A physical unit like processor when considered as a resource, it can regulate and control overall activities involved in the scenario on its own.
- A global resource may also be segregated for self-governing control. The buttons on a mouse, a workstation screen can be the global resources of the type space.
- The entities dedicated for unique identification of a particular object in a given scenario can be deliberated as resources. For instance, class names in class model, object IDs or object names in object model, file names in file management system are the logical entities that are used as resources.
- In case of an ATM machine system, the account numbers of customers and the bank codes are global resources. Bank codes should be unique within the context of a bank.

4.8 Choosing a Software Control Strategy

GQ. Explain : (a) External control (b) Internal control.

- Basically, there are two types of control in a software system :

1. External control	2. Internal control

- The flow of the events between the objects involved in the software system scenario which are visible from outside are termed as external control flows.
- However, the control flow comprised by a process is known as an internal control flow.

4.8.1 External Control

GQ. . What are the different categories of external control? Explain in brief.

- Moreover, there are three types of control for external events :

1. Procedure-driven Control	2. Event-driven Control	3. Concurrent Control

- The choice of control flow totally depends on the existing resources involved in the software system application.

4.8.1.1 Procedure-driven Control

- In a procedure-driven sequential system, control exists within the software program coding.
- The procedure driven control is quite easy to implement by means of conventional languages.
- It is the honest responsibility of a software system designer to translate events into operations among objects.
- The drawback of procedure driven control is, concurrency inherency is significant amongst objects involved in a scenario.

4.8.1.2 Event-driven Control

- An event-driven control is associated with the circumstances where the measurement method is inherently event-based in nature.
- Event-driven control offers adaptable and compliant control.
- From implementation point of view, it is more challenging to implement as compared to the procedure-driven control.
- As far as modularity of a software system is concerned, event-driven control flow supports for additional modularity for breaking of a software system into subsystems.

4.8.1.3 Concurrent Control

- Concurrent control guarantees that, the respective transactions involved within a scenario are accomplished and executed concurrently devoid of violating the integrity of data and hence, data remains whole, complete and uninterrupted within a scenario.

- It is also known as yes-no control or screening.
- Ultimately, there are three basic classes of concurrent control mechanisms: pessimistic, semi-optimistic and optimistic.
- Pessimistic concurrent control refers to the blocking of a transaction if that particular transaction violates the rules. Semi-optimistic concurrent control blocks transactions in some circumstances that may cause violation and optimistic concurrent control do not blocks the transaction but it postpones the respective transaction.

4.8.2 Internal Control

- Throughout the software system design, the software system developer expands operations on objects into lower-level operations on the same or other objects.
- Software system designers can make use of identical system execution mechanisms for developing internal objects, external objects and communication and coordination amongst internal and external objects.

4.9 Handling Boundary Conditions

- Three issues should be deliberated while handling boundary conditions which are initialization, termination and failure.
- At the outset, the software system should initialize parameters, constants and variables.
- Termination is generally modest as compared to the initialization since many internal objects can merely be unrestricted. A particular executing transaction or task should release the resources.
- Failure is the unintended closure of a software system. Usually, software development team members recognize that, a proposed developed software system is not working as per the requirements mentioned in the software requirements specification.
- The failures are candidly observed by software testers during thorough and systematic software testing of a software system. A misbehavior in the execution of a software system can be considered as an indication of the failure.

4.10 Setting Trade-off Priorities

- The trade-off priorities should be established for the good software system design. It is the responsibility of software system designer to set trade-off priorities for a software system.
- The task of designing trade-off priorities encompasses several types of software development techniques. If customer needs a delivery of a particular software module earlier than the outstanding software modules then customer should sacrifice the overall functionality of the software system as a whole.
- It is the duty of the software system designer to define the comparative significance of the several criteria as a guide for creation of design trade-offs.
- The software system designer does not form all the adjustments but launches the priorities for constructing respective software system arrangements.

4.11 Selecting an Architectural Style

GQ. Write a short note on : Architectural styles.
GQ. What do you mean by batch transformation and continuous transformation?
GQ. Explain :
(a) Interactive interface (b) Dynamic simulation
(c) Real time system (d) Transaction manager

- Numbers of architectural styles are commonly used in existing software systems.
- Following are some types of architectural styles :

1.	Batch Transformation	2.	Continuous Transformation
3.	Interactive Interface	4.	Dynamic Simulation
5.	Real Time System	6.	Transaction Manager

4.11.1 Batch Transformation

- In batch transformation, the information transformation is executed once on a complete input dataset. This architectural style accomplishes sequential computations.
- The main objective is to calculate an answer and is achieved by the application which is meant for receiving the inputs. Stress analysis of a bridge, payroll processing are the classic applications of batch transformation.
- In batch transformation, software developers should first of all breakdown the complete transformation into stages with each stage accomplishing one part of the particular transformation which is then followed by Formulation of class models (class diagram) for the input, output and in between each pair of successive stages.
- Next step involved is, expansion of each phase or level until the operations are straightforward to implement and finally reorganize the ultimate pipeline for optimization.

4.11.2 Continuous Transformation

- It is a system in which the output of the system is aggressively dependent on varying inputs. An uninterrupted transformation updates system outputs frequently.
- Windowing systems, signal processing are the classic applications of continuous transformation.
- In continuous transformation, the first step involved is to breakdown the complete transformation into stages with each stage accomplishing one part of the transformation. Then, we should describe and summarize input, output and intermediate models amongst all of the pairs of successive stages, as for the batch transformation.
- After successful breakdown of the complete transformation and defining inputs and outputs, we should discriminate each operation in order to update the incremental modifications or alterations to each level. Finally, we have to add transitional objects for optimization purpose.

4.11.3 Interactive Interface

- It is a system that is conquered by interactions amongst the external agents and the system itself.
- The external agents can be devices or humans.
- These external agents are self-governing and are independent of the system, so the system cannot control the agents.
- While using an interactive interface as an architectural style, we should first of all Segregate interface classes from the application classes.
- Predefined classes should be used for communication and coordination amongst the external agents.
- Next, we should separate out the logical events from physical events. Logical events correspond to multiple physical events. At last, we should identify and state the application functions that are invoked by the interface.

4.11.4 Dynamic Simulation

- This architectural style is dedicated for designing and modeling real world objects. Video games can be the classic examples of this type.
- The internal objects in the dynamic simulation correspond to real world objects and hence the class model (class diagram) is ordinarily significant.
- While selecting a dynamic simulation as an architectural style, we should diagnose active real-world objects along with the discrete events that relates to discrete interactions with the object from the class model (class diagram). Constant dependencies should also be predicted.

4.11.5 Real Time System

- The real time system is an interactive system with close-fitting or tight time constraints on actions.
- Real time system design is multifaceted and encompasses issues like interrupt handling, coordination of multiple central processing units, etc.

4.11.6 Transaction Manager

- It is a system dedicated for retrieval and storage of data.
- The transaction manager deal with several users who write and read data at the same time that is, concurrently.
- Data should be protected from unauthorized access and accidental loss.
- Inventory control, airline reservations, order fulfillment are the classic examples of the transaction manager.
- Steps involved in the transaction manager are :
  - Map the class model to the database structures.
  - Determine the units of concurrency.
  - Determine the unit of transaction.
  - Design concurrency control for transactions.

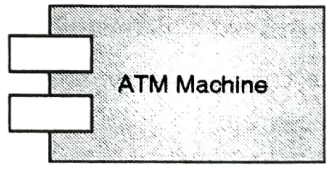
4.12 Component Diagram

GQ. Explain in detail the elements of a component diagram.

- A component diagram is basically meant for exhibiting the physical characteristics of the object oriented systems. It is also used for modeling the static implementation view of the software system.
- The physical belongings such as documents, tables, libraries, files, etc. that exist in a node are modeled with the help of a component diagram.
- A component diagram provides a framework for constructing the executable software systems and applications by means of forward and reverse engineering.
- A component diagram depicts a set of components and relationships among those components. Graphically, it is a collection of arcs and vertices.
- The main components of the component diagram includes :
  - Components
  - Interfaces
  - Relationships
    - Generalization
    - Realization
    - Dependency
    - Association
- Also, the component diagram may encompass constraints, packages, subsystems and notes likewise the other UML diagrams.
- The component diagram can be considered as a special type of a class diagram that mainly emphases on the components involved within a system.

Component

- A component is a physical and expendable part of a system that offers the realization of a set of interfaces.
- Graphically, it is represented as a rectangle with tabs.
- Each component involved within a system should be given a name (textual string) in order to identify a particular component uniquely from other components.



- The UML notation for component is depicted in Fig. 4.12.1. **Fig. 4.12.1 : UML notation of Component**
- Component shows the physical packaging of the logical elements such as collaborations and classes within a system.
- There are three types of components :
  1. **Work product components :** Examples are data files, source code files, etc.
  2. **Deployment components :** This type of components are essential for creation of execution components.
  3. **Execution Components :** Executable, Dynamic libraries, etc. are the typical examples.

Interface

- An interface is a collection of operations which are used for postulating a service of a particular component or a class.
- The relationship among interface and component is quite important.
- The interfaces are used as a glue for binding components altogether.
- Graphically, interface is depicted with the help of notation shown in the Fig. 4.12.2.

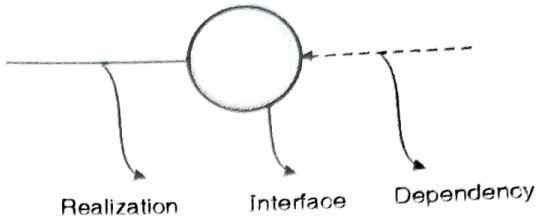


Fig. 4.12.2 : UML notation of Interface

Relationships

- Please refer Section 1.9.2 of this book.
- Case Study : ATM System
  - Fig. 4.12.3 depicts a component diagram for ATM system scenario.

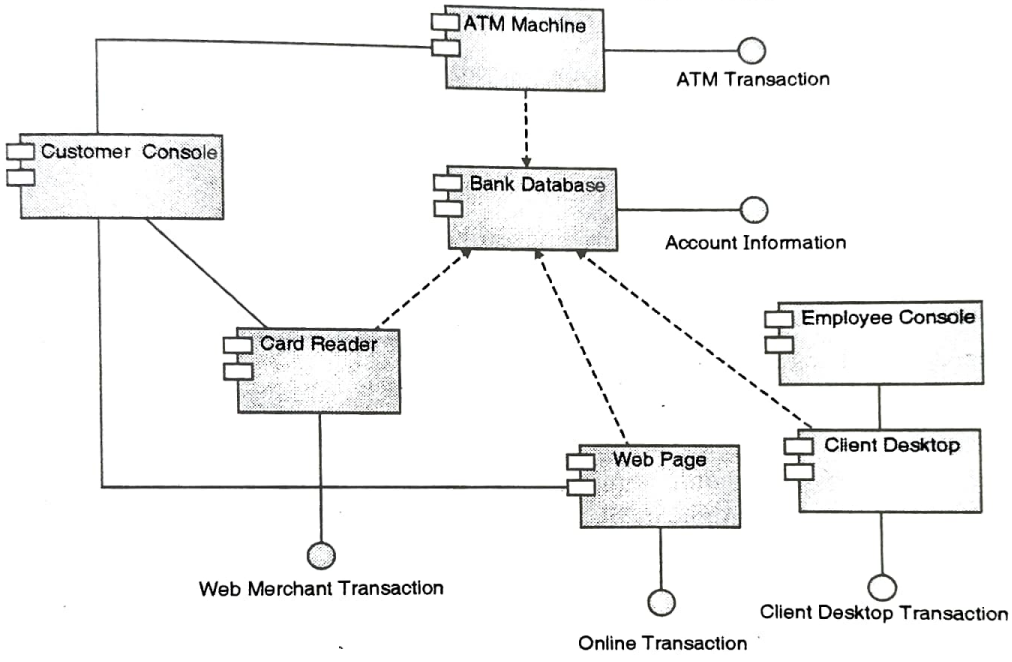


Fig. 4.12.3 : Component Diagram for ATM System

- In conclusion,
  - A component diagram should cover only those elements which are crucial for better understanding of the system.
  - It is focused on software systems static implementation view.

Guidelines for designing a Component Diagram

- A unique name should be given to each component involved in the scenario.
- In order to diminish the crossing lines, all the components and interfaces should be properly arranged.
- Notes and constraints should be used wherever necessary in order to draw attention to significant features of a software system.

4.13 Deployment Diagram

GQ. Explain the following with respect to Deployment diagram :
(a) Node (b) Association (c) Dependency

- A deployment diagram is the subsequent diagram meant for depicting the physical aspect of an object oriented system. It shows the configuration of run time processing nodes and the components involved in the software system.
- A deployment is basically used for modeling the static deployment view of a system. A deployment diagram provides a framework for constructing the executable software systems and applications by means of forward and reverse engineering.
- Following are the basic elements of a deployment diagram :

1. Nodes 2. Relationships

- A deployment diagram may contain components, subsystems and packages as per the necessity of a system.

4.13.1 Node

- Node in a deployment diagram is an important building block in forming the physical facets of a software system.
- Nodes are used to design the topology of the hardware on which a proposed software system executes.
- A solitary node basically represents a device or a processor on which components might be deployed.
- Fig. 4.13.1 depicts a UML notation of a node in deployment diagram.

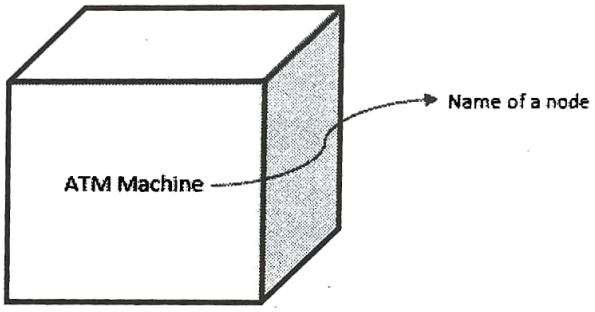


Fig. 4.13.1 : UML notation of a node

4.13.2 Relationship

1. Association 2. Dependency

4.13.2.1 Association

- Association is the most common type of relationship used amongst nodes involved in a deployment diagram.
- An association relationship shows a physical connection between nodes. For instance, shared bus, Ethernet line, etc.
- Association relationship can be furthermore used to define indirect connections.

4.13.2.2 Dependency

- Dependency is the other kind of relationship that can be used for showing the interdependency amongst nodes involved within a system scenario.
- **Case Study : ATM System**

GQ. Draw Deployment diagram for ATM System.

- Fig. 4.13.2 shows a deployment diagram for ATM system.

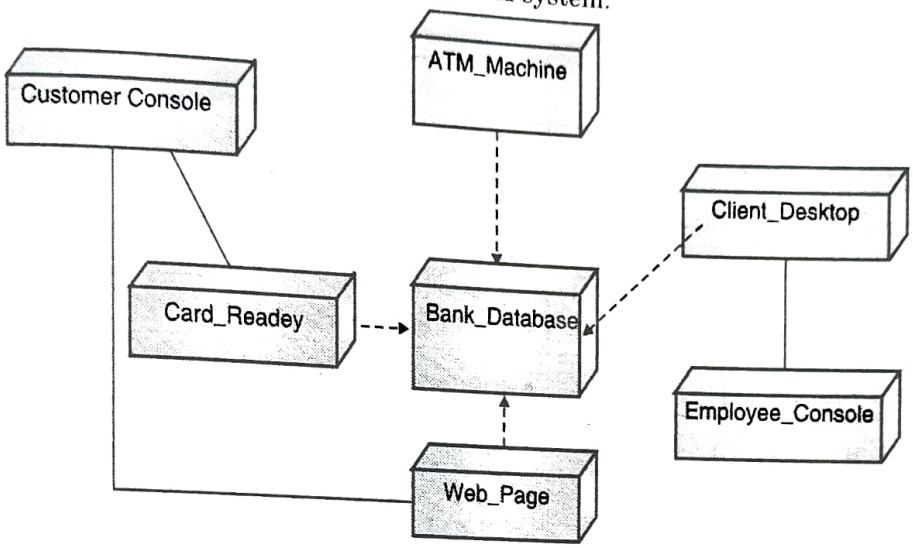


Fig. 4.13.2 : Deployment Diagram for ATM System

- In conclusion,
  - A deployment diagram should offer an abstraction of the hardware components and systems involved in the scenario.
  - Nodes in a deployment diagram should expose only those operations and attributes which are pertinent to the software system.
  - A deployment diagram should straightway deploy a set of components that exist in a solitary node.
  - A deployment diagram should focus on static deployment view of a system.
  - A deployment diagram should cover only those elements which are crucial for better understanding of the system.

Guidelines for designing a Deployment Diagram

- A unique name should be given to each node involved in the system scenario.
- In order to diminish the crossing lines, all the nodes and components should be properly arranged.
- Notes and constraints should be used wherever necessary in order to draw attention to significant features of a software system.
- Stereotyped elements should be used judiciously.
- A deployment diagram should offer a static deployment view of a system.

Key Concepts

• System Performance Estimation	• Reuse
• Reuse Plan	• Library
• Framework	• Pattern
• Layer	• Partition
• Concurrency Identification	• Inherent Concurrency
• Concurrent task	• Hardware Resource Requirements Estimation
• Hardware-Software Trade-offs	• Task Allocation
• Physical Connectivity	• Data Storage Management
• Global Resources	• Software Control
• Procedure-driven control	• Event-driven control
• Concurrent control	• Internal control
• Boundary condition	• Trade-off Priority
• Architectural Style	• Batch Transformation
• Continuous Transformation	• Interactive Interface
• Dynamic Simulation	• Real Time System
• Transaction Manager	• Component
• Interface	• Component Diagram
• Node	• Deployment Diagram

Summary

- The ***performance estimation*** process should be fast and progressive in nature. The basic need for the performance estimation is that, the software developers should try to simplify the assumptions made during the requirements analysis phase of the software development.
- The main agenda behind the performance estimation of a software system is to determine the feasibility of a software system and it is not at all dedicated for attaining the greater accuracy in the final outcome.
- Basically, the software system performance estimation process comprises of two activities: prediction, approximation and estimation.
- ***Reuse plan*** is considered as one of the key advantage of the object oriented technology; but, the reuse of system components or system itself does not happen unexpectedly. Reuse should be well planned.
- Two possible facets of reuse are :
  - With the help of existing things

- By means of producing reusable new things.

- Reusable things contain libraries, models, frameworks and patterns.
- Reuse of model is the most practical form of reuse. The logic in a model can be applied to several problems.
- A ***library*** is a collection of classes that are convenient and valuable in various circumstances.
- A ***framework*** is a structure of a program that must be expanded in order to design and develop a complete software application.
- A ***pattern*** is simply nothing but an established and confirmed solution to a general problem.
- A software system should be divided into a small number of subsystems for better understanding of the software system scenario.
- A ***layered system*** is a well-ordered set of tiers where each tier is built in terms of the tier below it and providing the implementation foundation for the tier situated above it.
- ***Partitions*** are meant for vertical division of a main system into a number of self-governing and inadequately coupled subsystems where each subsystem delivers some type of facility or service.
- We can decompose a system into subsystem by merging partitions and layers.
- Partitions can be layered and layers can be partitioned.
- All objects are concurrent or synchronized in the system analysis model, as in the real world and in hardware.
- As far as implementation is concerned, all of the software objects are not concurrent, since one processor can support many objects.
- Two objects are inherently concurrent if they can receive the event at the same time deprived of interacting.
- A ***thread of control*** is a path through a set of state diagrams on which only a single object is active at a time.
- A thread remains within a state diagram unless and until an object sends an event to another object and waits for another event.
- The thread passes to the receiver of the event until it ultimately returns to the original object.
- The thread splits if the object sends an event and continues executing.
- On each thread of control, only single object is active at a time.
- We can implement thread of control as a particular task in computer system.
- The number of essential processors required totally depends on the speed of the machine and the volume of calculations.
- Numbers of substitutes are available for managing the data storage that we can use distinctly or in combination. For instance, databases, files, data structures, etc.
- Different types of data stores offer trade-offs between cost, access time, capacity, cost, reliability and access time.
- ***Hardware control*** closely matches the analysis model however there are quite a lot of ways for implementing and executing control in a software system.

- ***Internal control*** concerns the flow of control contained by a process.
- In a ***procedure-driven sequential system***, control exists within the program code.
- Procedure request external input and then wait for it; when input arrives, control resumes within the procedure that made the call.
- The location of the program counter and the stack of procedure calls and local variables define the system state.
- The main benefit of procedure-driven control is that, it is quite easy to implement with conventional languages while the drawback is that it necessitates the concurrency inherent in objects to be mapped into a sequential flow control.
- In an ***event-driven sequential system***, control exists within a dispatcher or monitor that the language, subsystem, or operating system offers.
- In a ***concurrent system***, control exist simultaneously in several independent objects, each a separate task.
- Internal object interactions are similar to external object interactions since we can make use of the same implementation mechanisms.
- The software system designer should set priorities that will be used to guide trade-offs for the rest of the software system design.
- Number of architectural styles are commonly used in existing software systems:
  - Batch Transformation
  - Continuous Transformation
  - Interactive Interface
  - Dynamic Simulation
  - Real Time System
  - Transaction Manager
- In ***batch transformation***, the information transformation is executed once on a complete input set.
- A ***continuous transformation*** is a system in which the output of the system is aggressively dependent on varying inputs. A continuous transformation updates outputs frequently.
- An ***interactive interface*** is a system that is conquered by interactions amongst the external agents and the system itself. The external agents can be devices or humans.
- ***Dynamic simulation*** tracks or models real world objects.
- The ***real time system*** is an interactive system with close-fitting or tight time constraints on actions.
- A ***transaction manager*** is a system dedicated for retrieval and storage of data.
- A ***component diagram*** is basically meant for exhibiting the physical characteristics of the object oriented systems.
- A component diagram provides a framework for constructing the executable software systems and applications by means of forward and reverse engineering.
- A ***component*** is a physical and expendable part of a system that offers the realization of a set of interfaces.

- Three kinds of components are :
  - *Work product components* : Examples are data files, source code files, etc.
  - *Deployment components* : These types of components are essential for creation of execution components.
  - *Execution Components* : Executable, Dynamic libraries, etc. are the typical examples.
- An ***interface*** is a collection of operations which are used for postulating a service of a particular component or a class.
- A ***deployment diagram*** is the subsequent diagram meant for depicting the physical aspect of an object oriented system.
- A deployment diagram may contain components, subsystems and packages as per the necessity of a system.
- ***Node*** in a deployment diagram is an important building block in forming the physical facets of a software system.
- Nodes are used to design the topology of the hardware on which a proposed software system executes.
- ***Association*** is the most common type of relationship used amongst nodes involved in a deployment diagram.
- An association relationship shows a physical connection between nodes. For instance, shared bus, Ethernet line, etc.

Chapter Ends...

❑❑❑