Experiment No. : 7

Title: Modelling Structure diagram using UML

Batch:B3 **Roll No.: 1714103** **Experiment No.: 7**

Aim: To model Structure diagram using UML

Resources needed: IBM Rational Rose / Open Source UML Tool

Theory

Structure Diagrams

Structure diagram shows static structure of the system and its parts on different abstraction and implementation levels and how those parts are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts.

Structure diagrams are not utilizing time related concepts; do not show the details of dynamic behavior. However, they may show relationships to the behaviors of the classifiers exhibited in the structure diagrams.

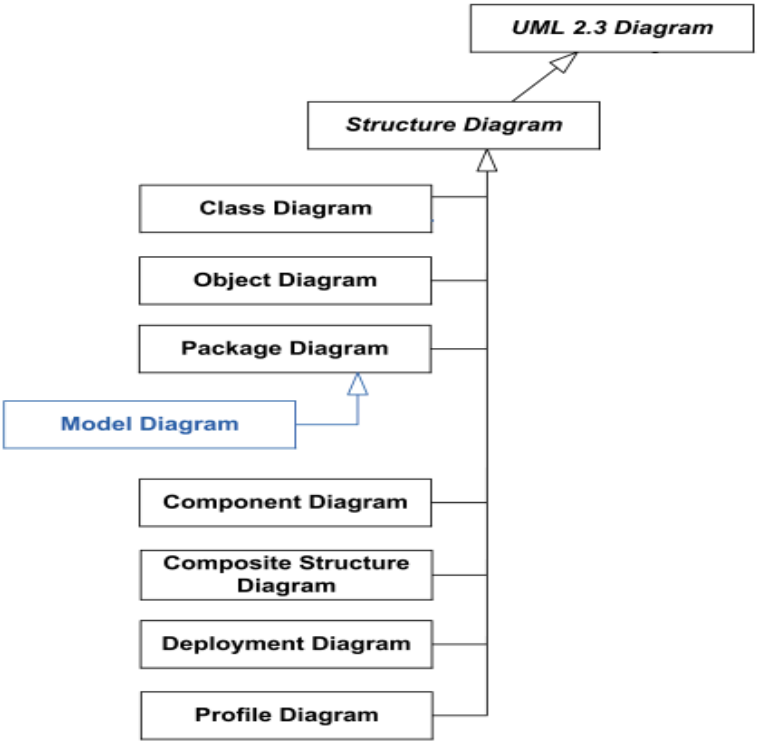


Figure 6.1 UML Structural Diagrams

Class diagram is static structure diagram describing structure of a system on the (lowest) level of classifiers (classes, interfaces, etc.). It shows system's classifiers, their attributes, and the relationships between classifiers.

Object diagram shows instances of classifiers and links (instances of associations) between them.

Package diagram shows packages and dependencies between the packages. Models allow to show different views of a system, for example, as multi-layered (aka multi-tiered) application

Component diagram shows components and dependencies between them. This type of diagrams is used for Component-Based Development (CBD), to describe systems with Service-Oriented Architecture (SOA).

Composite structure diagram could be used to show:

Internal structure of a classifier

Internal Structure diagrams show internal structure of a classifier - a decomposition of the classifier into its properties, parts and relationships.

Behaviour of collaboration

Collaboration use diagram shows objects in a system cooperating with each other to produce some behavior of the system.

Deployment diagram shows execution architecture of a system that represents the assignment (deployment) of software artifacts to deployment targets (usually nodes).

Profile diagram is auxiliary UML diagram which allows defining custom stereotypes, tagged values, and constraints. The Profile mechanism has been defined in UML for providing a lightweight extension mechanism to the UML standard. Profiles allow to adapt the UML meta model for different platforms (such as J2EE or .NET), or domains (such as real-time or business process modeling).

Profile diagrams were first introduced in UML 2.0.

Class diagram:

Class diagram is a schema, pattern or template for describing many possible instances of data. A class diagram describes object classes. A given class diagram describes infinite set of instance diagram. Class diagram is used to define a detailed design of the system.

Need of class diagram:

An object model captures a static structure of a system by showing the objects in the system, relationship between the objects, and the attributes and operation that characterize each class of objects. Object model provide an intuitive graphic representation of a system and are valuable for communicating with customers and documenting the structure of the system.

Elements of Class Diagram:

Class: Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.

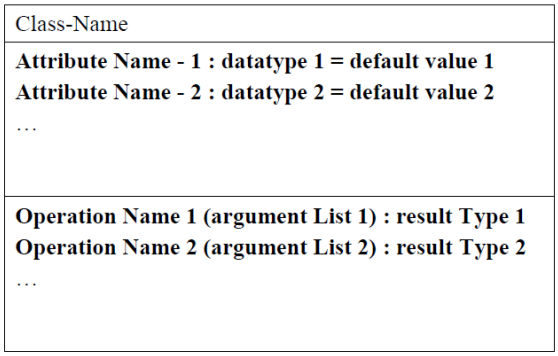


Figure 6.2 Class Diagram

Class

Attributes: An attribute is data value held by the objects in a class. An attribute should be a pure data value, not an object. Unlike objects, pure data values do not have identity.

Operations and Methods: An operation is a function or transformation that may be applied to or by objects in a class. Operations are listed in the lower third of the class box.

Links and Association: Links and association are the means for establishing relationships among objects and classes. A link is a physical or conceptual connection between object instances. An association describes a group of links with common structure and common semantics.

Association: An association describes a set of potential links in the same way that a class describes a set of potential objects. Associations are inherently bi-directional.

Multiplicity: Multiplicity specifies how many instances of one class may relate to a single instance of an associated class. Multiplicity constrains the number of related objects. Multiplicity is often described as being “one” or “many” but more generally it is subset of non-negative integers.

Object diagram indicate multiplicity with special symbols at the ends of association lines. Multiplicity can be specified with a number or set of intervals, such as “1”, “1+”(1 or more), “3-5”(3 to 5, inclusive), and “2,4,18” (2,4 or 18) .

Generalization and Inheritance: Generalization and Inheritance are powerful abstractions for sharing similarities among classes while preserving their differences.

Generalization is the relationship between a class and one or more refined versions of it. The class being refined is called the super class and each refined version is called subclass. Attributes and operations common to a group of subclasses are attached to the super class and shared by each subclass. Each subclass is said to inherit the features of its super class. Generalization is sometimes called the "is-a" relationship because each instance of a subclass is an instance of the super class as well.

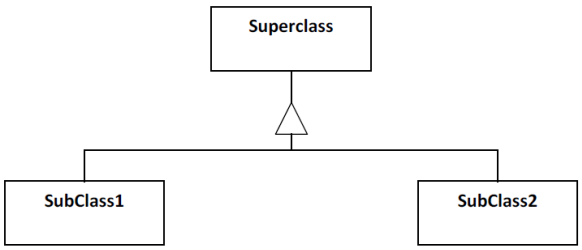


Figure 6.3 Generalization

Aggregation: Aggregation is the "part- whole" or "a-part-of" relationship is which objects representing the components of something are associated with an object representing the entire assembly. Aggregation is a tightly coupled form of association with some extra semantics. The most significant property of aggregation is transitivity that is if A is part of B, and B is part of C then A is part of C. Aggregation is also asymmetric, that is if A is part of B then B is not part of A. Finally, some properties of the assembly propagate to the components as well possibly with some local modifications.

Figure 6.4 Aggregation

Object diagram:

The object diagram is a special kind of a class diagram. An object is an instance of a class. This essentially means that an object represents the state of a class at a given point of time while the system is running. The object diagram captures the state of different classes in the system and their relationship or associations at a given point of time.

Component Diagram

A component represents a software module with a wee- defined interface. The interface of a component is represented by one or several interface elements that the component provides.

Components are used to show compiler and run-time dependencies, as well as interface and calling dependencies among software modules. They also show which component implement a specific class.

A system may be composed of several software modules of different kinds. Each software module is represented by a component in the model. To distinguish different kinds of components from each other, Stereotypes are used.

Need of Component diagram:

We use component diagram to visualize the static aspect of the physical components and their relationships and to specify their details for construction. This involves modeling the physical things that reside on a node, such as executables, libraries, tables, files and documents.

When we model the static implementation view of a system, we will typically use component diagram in one of four ways.

1. To model source code: With most contemporary object-oriented programming languages, we will cut code using integrated development environments that store our source code in files. We can use component diagrams to model the configuration management of these files, which represents work-product components.

2. To model executable releases: A release is a relatively complete and consistent set of artifacts delivered to an internal or external user. In the context of components a release focuses on the part necessary to deliver a running system. When we model a release using component diagram, are visualizing, specifying, and documenting the decisions about the physical parts that constitute our software –that is , its deployment components.

3. To model physical databases: Think of a physical database as the concrete realization of a schema, living in the world of bits. Schemas, in effect, offer an API to persistent information; the model of a physical database represents the storage of that information in the tables of the relational database or the pages of an object-oriented database .We use component diagram to represent these and other kinds of physical databases.

4. To model adaptable system: Some systems are quite static; their components enter the scene, participate in an execution, and then depart. Other systems are more dynamic , involving mobile agent or components that migrate for purpose of load balancing and failure recovery. We use component diagrams in conjunction with some of the UML's diagrams for modeling behavior to represent these kinds of systems.

Elements of Component Diagram:

Component: A component is a physical and replaceable part of a system that confirms to and provides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs, with the name of the object in it, preceded by a colon and underlined.

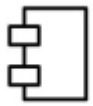


Figure 6.5 Component

Class/ Interface/ Object : Similar to the notation used in class and object diagrams.

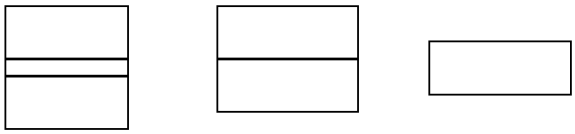


Figure 6.6 Class, Interface, Object

Relation / Association : Similar to the relation/ association used in class diagram.

Relation / Association

Deployment Diagram

A deployment diagram is a diagram that shows the configuration of run time processing nodes & the components that live on them. This diagram is by far more useful when a system is built and ready to be deployed. But, this does not mean that we should start on our deployment diagram after our system is built. On the contrary, our deployment diagram should start from the time our static design is being formalized using, say, class diagram. This deployment diagram then evolves and is revised until the system is build. It is always a best practice to have visibility of what our deployment environment is going to be before the system is build so that any deployment-related issues are identified to be resolved & not crop up at the last minute.

Need of Deployment Diagram:

We use deployment diagrams to model the static deployment view of a system. For the most part, this involves modeling the technology of the hardware on which our system executes. Deployment diagrams are essentially class diagrams that focus on a system's nodes. When we model the static deployment view of a system, we will typically use deployment diagrams in one of three ways.

- **To model embedded systems**

An embedded system is a software-intensive collection of hardware that inter. faces with the physical world. Embedded systems involve software that controls devices such as motors, actuators, and displays and that, in turn, is controlled by external stimuli such as sensor input, movement, and temperature changes. We can use deployment diagrams to model the devices and processors that comprise an embedded system.

- **To model client/server systems**

A client/server system is a common architecture focused on making a clear separation of concerns between the system's user interface (which lives on the client) and the system's persistent data (which lives on the server). Client server systems are one end of the continuum of distributed systems and require you to make decisions about the network connectivity of client to servers and about the physical distribution of your system's software components across the nodes. We can model the topology of such systems by using deployment diagrams.

- **To model fully distributed systems**

At the other end of the continuum of distributed systems are those that are widely, if not globally, distributed: typically encompassing multiple levels of servers. Such systems are often hosts to multiple versions of software components, some of which may even migrate from node to node .Crafting such systems requires we to make decisions that enable the continuous change in the systems topology. We can use deployment diagrams to visualize the systems current topology and distribution of components to reason about the impact of change on that topology.

Elements of Deployment Diagram

Processor: A piece of hardware capable of executing programs. The processor is represented as shaded cube with a name of project. A processor can have list of processes on it.

Device: A piece of hardware incapable of executing program is called as device.

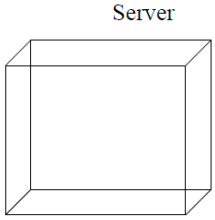


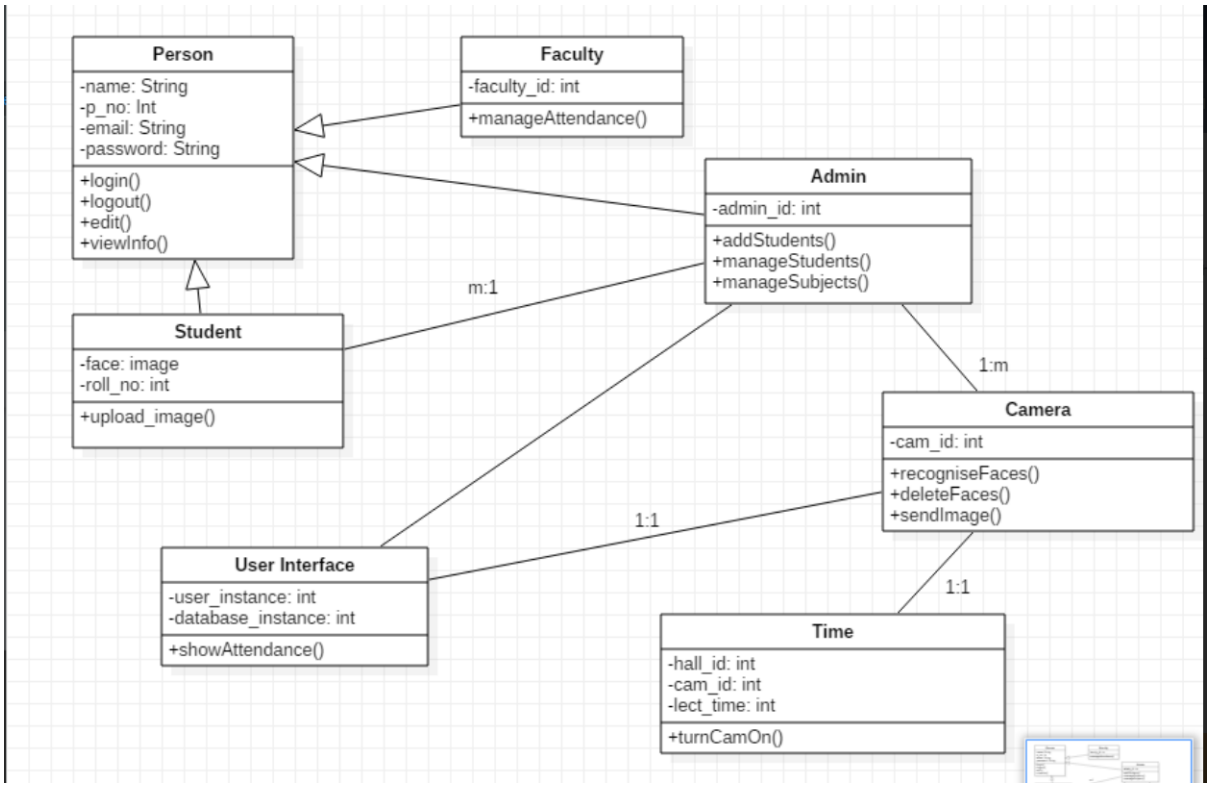
Figure 6.7 Processor, Device

Procedure:

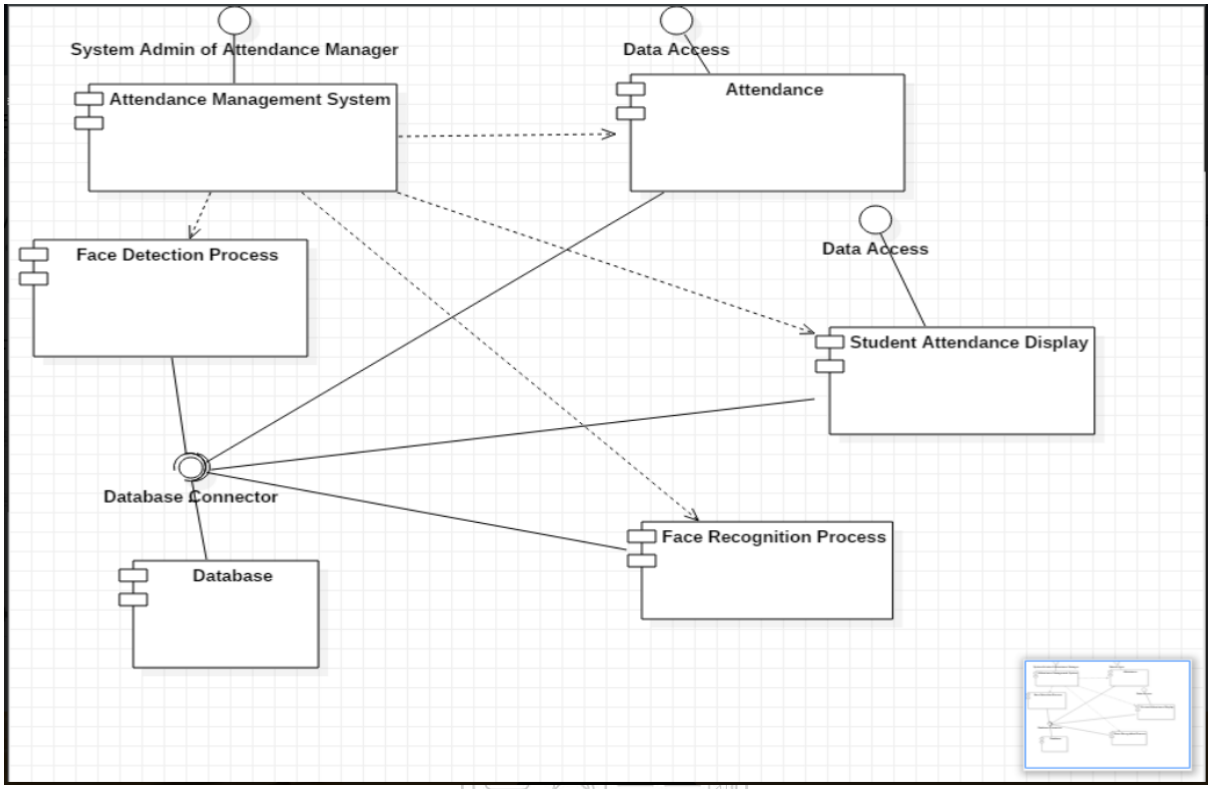
1. Prepare structural diagrams for chosen problem using Rational Rose/ any other Open Source UML tool.

Results: Printout of structural diagrams

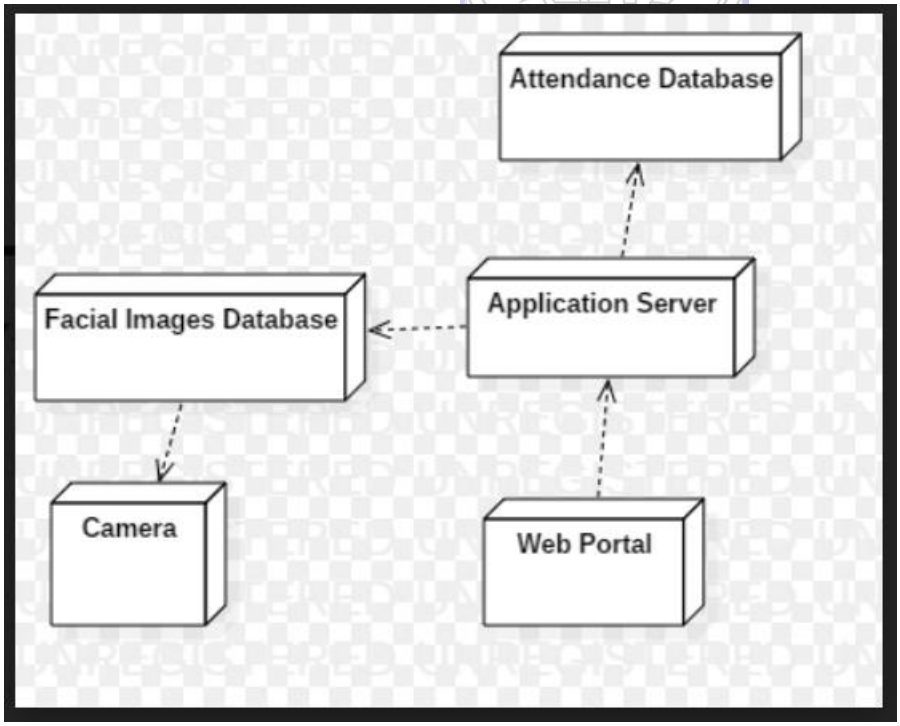
Class Diagram



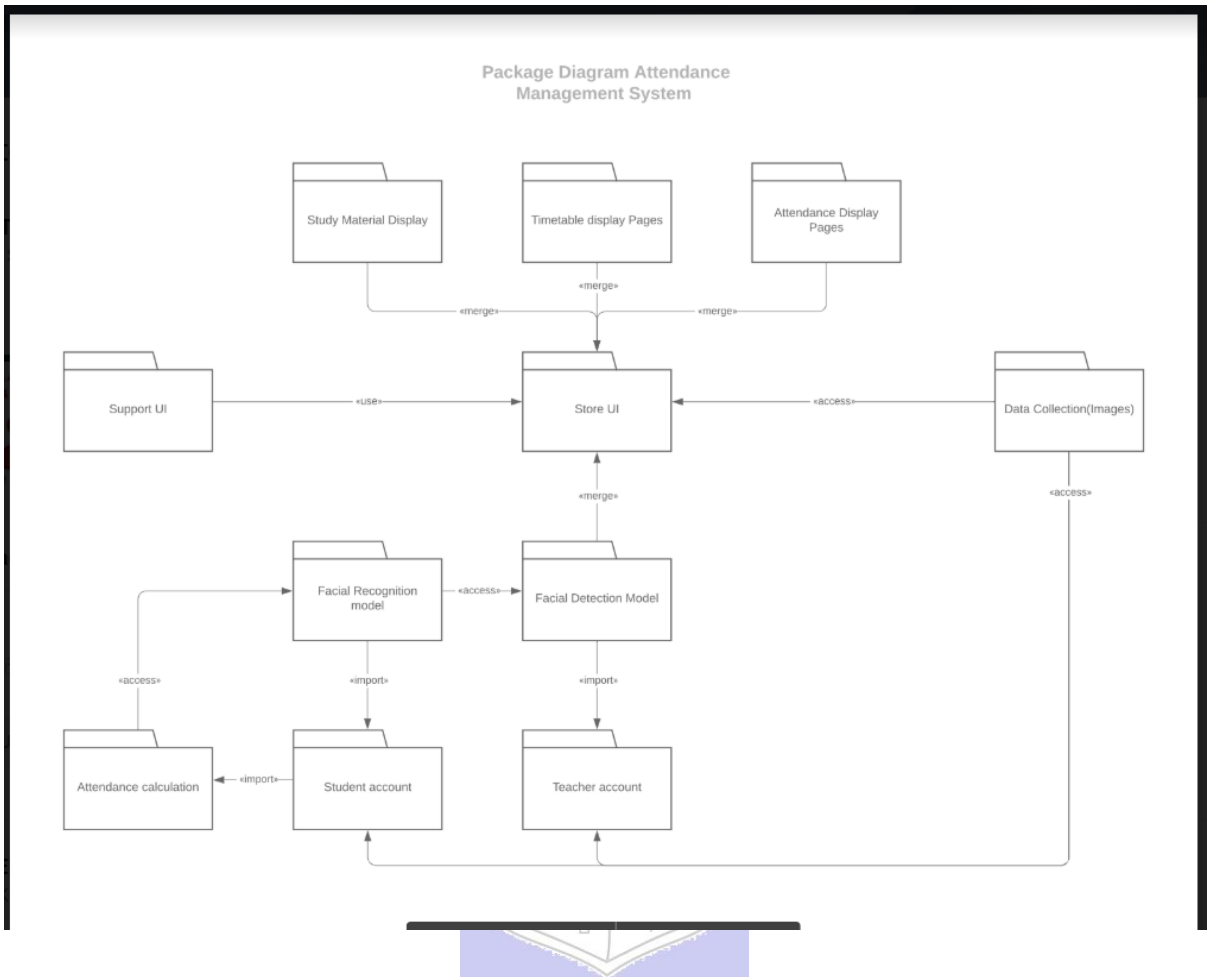
Component Diagram



Deployment Diagram



Package Diagram



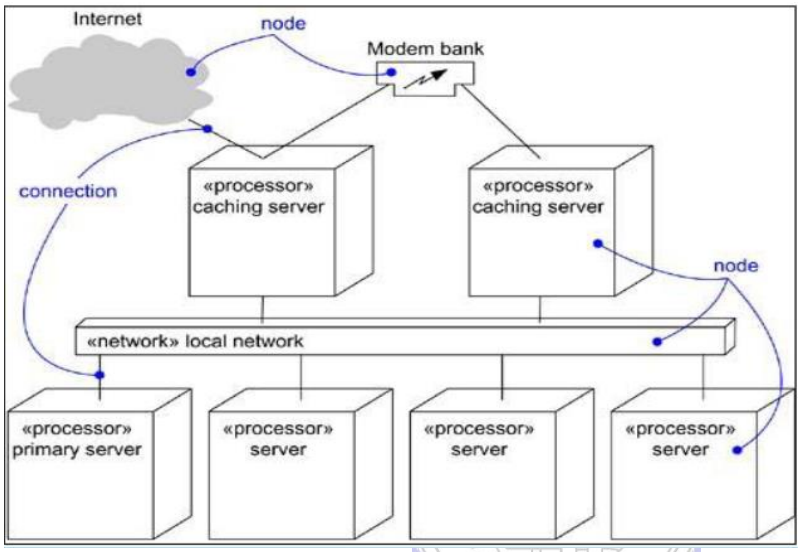
Questions:

1. **Give example of deployment diagram for modelling a fully distributed system.**

Answer:-

A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modeling the physical aspects of an object-oriented system. They are often be used to model the static deployment view of a system (topology of the hardware).

Following is an example:-



Outcomes:

Model the requirements using UML.

Conclusion:

We were successfully able to understand the importance of class,deployment,package and component diagrams and also created them using Star UML.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books / Websites:

1. Michael Blaha, James Rumbaugh, “Object-Oriented Modeling and Design with UML”, Prentice-Hall of India, 2nd Edition
2. Mahesh P. Matha, “Object-Oriented Analysis and Design using UML”, Prentice-Hall of India
3. Timothy C Lethbridge, Robert Laganiere, “Object-Oriented Software Engineering – A practical software development using UML and Java”, Tata McGraw-Hill, New Delhi.
4. http://www.uml-diagrams.org/uml-23-diagrams.html
5. http://vlabs.iitkgp.ernet.in/se/