UML LAB MANUAL

P.K Abhilash
Asst. Professor
Department of IT

Gokaraju Rangaraju Institute of Engineering and Technology
Bachupally, Hyderabad- 500072

Contents

Introduction

In late 1960's people were concentrating on Procedure Oriented Languages such as **COBOL, FORTRAN, PASCAL...etc.** Later on they preferred **Object Oriented Languages**. In the middle of 1970-80 three Scientists named as BOOCH, RUMBAUGH and JACOBSON found a new language named as **<u>Unified Modeling Language.</u>** It encompasses the Designing of the System/Program. It is a **Defacto** language.

What is UML?

* Is a ***<u>language.</u>*** It is not simply a notation for drawing diagrams, but a complete language for capturing knowledge (semantics) about a subject and expressing knowledge (syntax) regarding the subject for the purpose of communication.

* Applies to ***<u>modeling</u>*** and systems. Modeling involves a focus on understanding a subject (system) and capturing and being able to communicate in this knowledge.

* It is the result of ***<u>unifying</u>*** the information systems and technology industry's best engineering practices (principals, techniques, methods and tools).

* used for both database and software modeling

<u>Overview of the UML</u>

* The UML is a language for
    – visualizing
    – specifying
    – constructing
    – documenting

The artifacts of a software-intensive system

Visual modeling (visualizing)

- A picture is worth a thousand words!

  - Uses standard graphical notations
  - Semi-formal
  - Captures Business Process from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems

Specifying

- building models that are: Precise, Unambiguous, Complete
- UML symbols are based on well-defined syntax and semantics.
- UML addresses the specification of all important analysis, design, and implementation decisions.

Constructing

- Models are related to OO programming languages.
- Round-trip engineering requires tool and human intervention to avoid informationloss
  - Forward engineering — direct mapping of a UML model into code.
  - Reverse engineering — reconstruction of a UML model from an implementation.

Documenting

  - Architecture, Requirements, Tests, Activities (Project planning, Release management)

Conceptual Model of the UML

» To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements.

Elements:
1. Basic building blocks
2. Rules
3. Common Mechanisms

Basic Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks:

» Things
» Relationships
» Diagrams

Things in the UML

- There are four kinds of things in the UML:

1. Structural — nouns of UML models.

2. Behavioral — dynamic (verbal) parts of UML models.

3. Grouping — organizational parts of UML models.

4. Annotational — explanatory parts of UML models.

1. Structural Things
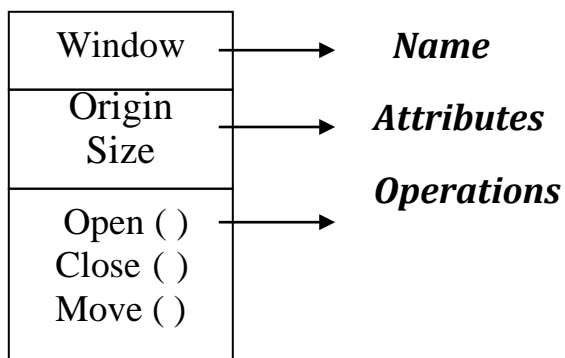
- These are the <u>Nouns</u> and <u>Static parts</u> of the model.
- These are representing <u>conceptual</u> or <u>physical</u> elements.

There are seven kinds of structural things:

1. Class
2. Interface
3. Collaboration
4. Use Case
5. Active Class
6. Component
7. Node

1. Class

Is a description of set of objects that share the same attributes, operations methods, relationships and semantics.


A Simple Class

2.Interface

A collection of operations that specify a service (for a resource or an action) of a class or component. It describes the externally visible behavior of that element


A Simple Interface
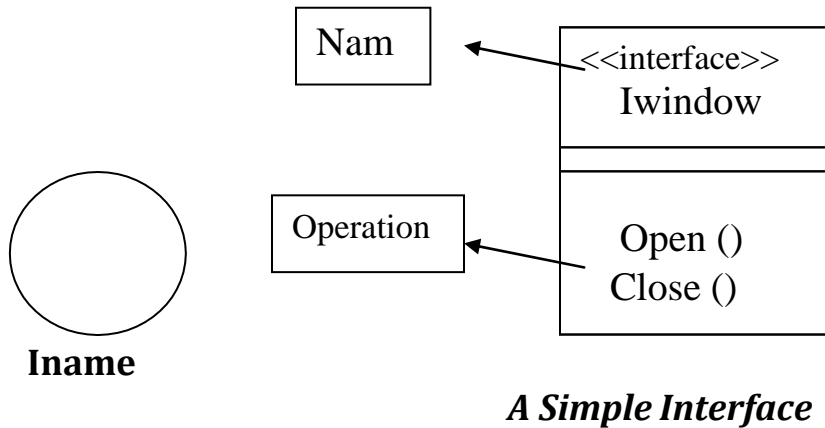
3.Collaboration

– Define an interaction among two or more classes.
– Define a society of roles and other elements.
– Provide cooperative behavior.
– Capture structural and behavioral dimensions.
– UML uses ‗pattern‖ as a synonym (careful)
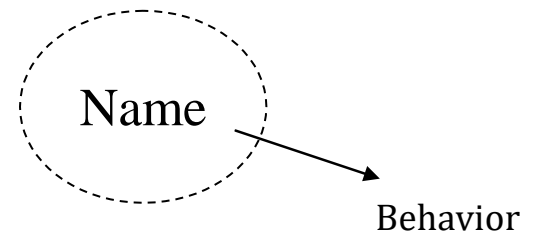
4.Use Case

– A sequence of actions that produce an observable result for a specific actor.
– A set of scenarios tied together by a common user goal.
– Provides a structure for behavioral things.
– Realized through a collaboration (usually realized by a set of actors and the system to be built).
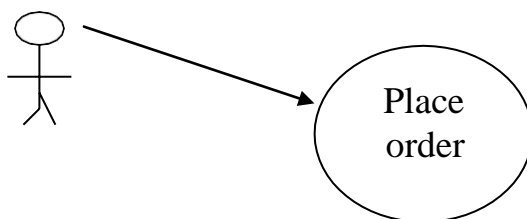
5. Active Class

– **Special class whose objects own one or more *processes* or *threads*.**
– **Can initiate control activity.**

Heavy border

Name

Attributes

Operations

Event Manager
Thread Time
Suspend () Flush ()
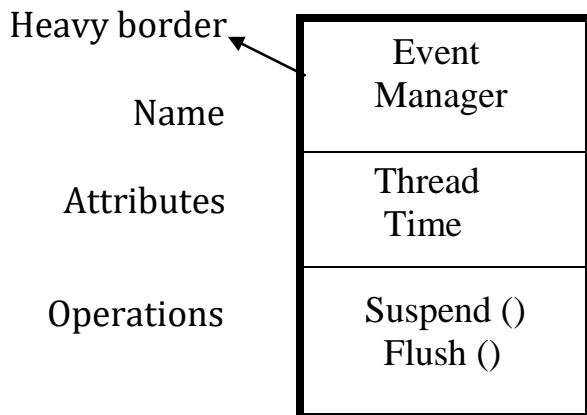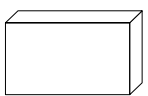
6. Component

- Replaceable part of a system.
- Components can be packaged logically.
- Conforms to a set of interfaces.
- Provides the realization of an interface.
- Represents a physical module of code

Orderform.java
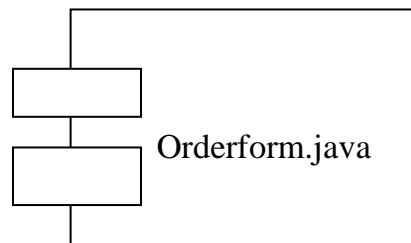
7. Node

- Element that exists at ***run time***.
- Represents a ***computational resource.***
- Generally has memory and processing power.

Web Server
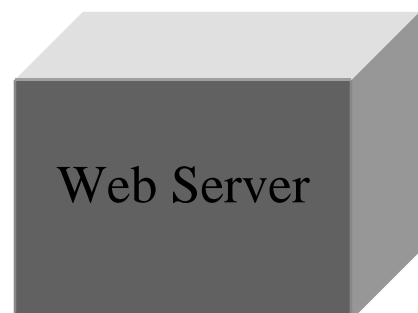
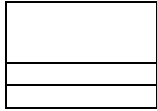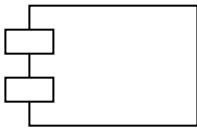| Construct	Description	Syntax
class	a description of a set of objects that share the same attributes, operations, methods, relationships and semantics.	
interface	a named set of operations that characterize the behavior of anelement.	«interface»
component	a physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces.	
node	a run-time physical object that represents a computational resource.	

Construct	Description	Syntax
constraint[1]	a semantic condition or restriction.	{constraint}

2.Behavioral Things

- These are <u>Verbs</u> of UML models.
- These are <u>Dynamic parts</u> of UML models: -behavior over time and space||
- Usually connected to structural things in UML.

  There are two kinds of Behavioral Things:

1.Interaction

- Is a behavior of a set of objects comprising of a set of messages exchanges within a particular context to accomplish a specific purpose.

Display
→

2.State Machine

- Is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.

Idle → Waiting
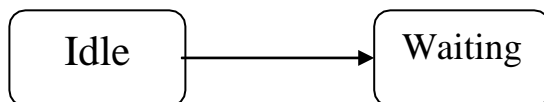
3. Grouping Things

- These are the organizational parts of the UML models.
- There is only one primary kind of group thing:

1.Packages

- General purpose mechanism for organizing elements into groups.
- Purely conceptual; only exists at development time.
- Contains behavioral and structural things.
- Can be nested.
- Variations of packages are: Frameworks, models, & subsystems.

```
┌──────┐
│      │
└──────┴──────────────┐
        │              │
        │   Business   │
        │    rules     │
        │              │
        └──────────────┘
```
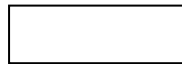
4.Annotational Things

- These are <u>Explanatory parts</u> of UML models
- These are the <u>Comments</u> regarding other UML elements (usually called adornments in UML)

        There is only one primary kind of annotational thing:

1. Note

A note is simply a symbol for rendering constraints and comments   attached to an element or collection of elements.
Is a best expressed in informal or formal text.

```
┌────────────────┐─┐
│                │ │
│                │─┘
│  Comments      │
│                │
│                │
└────────────────┘
```

Relationships

There are four kinds of relationships:

1. Dependency
2. Association
3. Generalization
4. Realization

» These relationships tie things together.
» It is a semantic connection among elements.
» These relationships are the basic relational building blocks of the UML.

1. Dependency

Is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing).
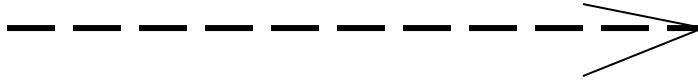
2. Association

Is a structural relationship that describes a set of links, a link being a connection among objects.

employer   employee

   0...1   *****
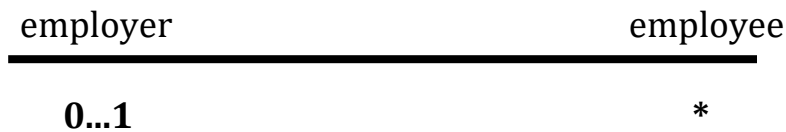
Aggregation

» Is a special kind of association. It represents a structural relationship between the whole and its parts.

» Represented by black diamond.
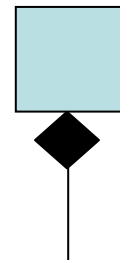
3. Generalization

Is a specialization/generalization relationship in which objects of the specialized element (the child) are more specific than the objects of the generalized element.
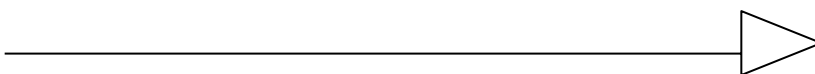
4. Realization

a semantic relationship between two elements, wherein one element guarantees tocarry out what is expected by the other element.
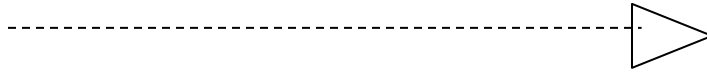
Where?

Between interfaces and classes that realize them...
Between use cases and the collaborations that realize them...

Structural Modeling: Core Relationships

Construct	Description	Syntax
association	a relationship between two or more classifiers that involves connections among their instances.	
aggregation	A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.	
generalization	a taxonomic relationship between a more general and a more specific element.	
dependency	a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).	

Construct	Description	Syntax
realization	a relationship between a specification and its implementation.	

<u>Diagrams</u>

- A diagram is the graphical presentation of a set of elements.
- Represented by a connected graph: Vertices are things; Arcs are behaviors.

  UML includes nine diagrams:

  - Class Diagram;
  - Object Diagram
  - Use case Diagram
  - Sequence Diagram;
  - Collaboration Diagram
  - State chart Diagram
  - Activity Diagram
  - Component Diagram
  - Deployment Diagram

Static Modeling	Dynamic Modeling
Class Diagram Object Diagram Component Diagram Deployment Diagram	Use case Diagram Sequence Diagram Collaboration Diagram State chart Diagram Activity Diagram

Both Sequence and Collaboration diagrams are called Interaction Diagrams.

1. Class Diagram

- Class Diagrams describe the **static structure** of a system, or how it is structured rather than how it behaves.
- A class diagram shows the existence of classes and their relationships in the logical view of a system

These diagrams contain the following elements.

- Classes and their structure and behavior
- Association, aggregation, dependency, and inheritance relationships
- Multiplicity and navigation indicators
- Role names

These diagrams are the most common diagrams found in O-O modeling systems.

Examples:

Registration

Student
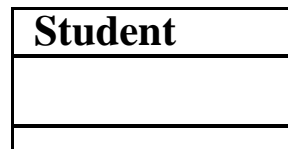
2. Object Diagrams

- Shows a set of objects and their relationships.
- A static snapshot of instances.
- Object Diagrams describe the **static structure** of a system at a particular time. Whereas a class model describes all possible situations, an object model describes a particular situation.

Object diagrams contain the following elements:

Objects   which represent particular entities. These are instances of classes.
Links   which represent particular relationships between objects. These are instances of associations.

```
┌─────────────────┐
│ Harry           │
│ (Studen         │
│ t)              │
├─────────────────┤
│ Name:"          │
│ Harry  Mat"     │
│ Major:  C.S     │
├─────────────────┤
│                 │
└─────────────────┘
```
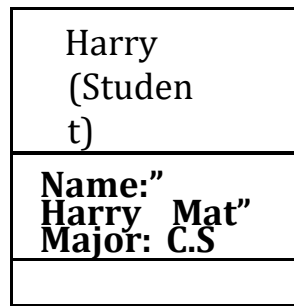
3. Use case diagrams

Use Case Diagrams describe the **functionality** of a system and **users** of the system.

These diagrams contain the following elements:

Actors: which represent users of a system, including human users and other systems.

Use Cases: which represent functionality or services provided by a system to users.

Registrar	Student	Maintain
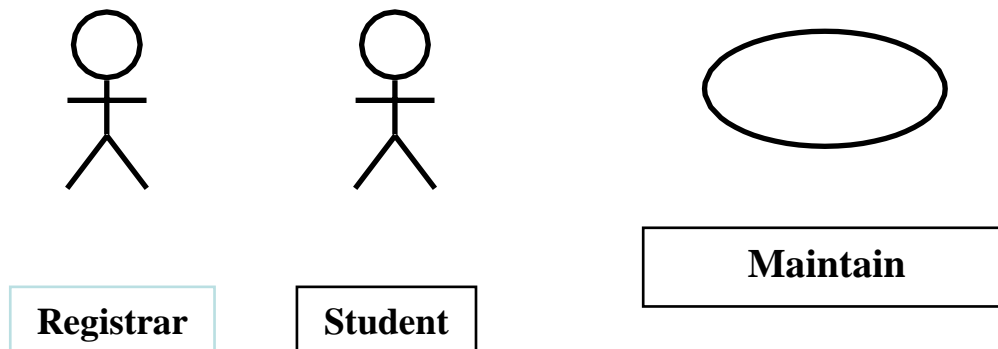
4. Sequence Diagrams

● Sequence Diagrams describe interactions among classes. These interactions aremodeled as exchanges of messages.
● These diagrams focus on classes and the messages they exchange to accomplishSome desired behavior.
● Sequence diagrams are a type of interaction

diagrams. Sequence diagrams contain the following

elements:

<u>Class roles:</u> which represent roles that objects may play within the interaction.
<u>Lifelines</u>:   which represent the existence of an object over a period of time.
<u>Activations</u>: which represent the time during which an object is performing an operation.

Messages: which represent communication between objects.

5. Collaboration Diagram

Collaboration Diagrams describe **interactions among classes and associations**. These interactions are modeled as exchanges of messages between classes Through their associations. Collaboration diagrams are a type of interactionDiagram.

Collaboration diagrams contain the following elements.

Cass roles: which represent roles that objects may play within the interaction.

Association roles: which represent roles that links may play within the interaction.

Message flows: which represent messages sent between objects via links. Links transport or implement the delivery of the message.

6. Statechart Diagrams

State chart (or state) diagrams describe the states and responses of a class. StatechartDiagrams describe the behavior of a class in response to external stimuli.

These diagrams contain the following elements:

States: which represent the situations during the life of an object in   which it satisfies somecondition, performs some activity, or waits for some occurrence.

Transitions: which represent relationships between the different states of an object.

7. Activity Diagrams

Activity diagrams describe the **activities of a class**. These diagrams are similar toStatechart diagrams and use similar conventions, but activity diagrams describe the behavior of a class in response to internal processing rather than external events as in Statechart diagram.

Swimlanes: which represent responsibilities of one or more objects for actions within an overallactivity; that is, they divide the activity states into groups and assign these groups to object that must perform the activities.

Action States: which represent atomic, or noninterruptible, actions of entities or steps in the execution of an algorithm.

Action flows: which represent relationships between the different action states of an entity.

Object flows: which represent the utilization of objects by action states and the influence of actionstates on objects.

8. Component Diagram

Component diagrams describe the organizations and dependencies among softwareImplementation components. These diagrams contain components, which represent Distributable physical units, including source code, object code, and executable code.
These are static implementation view of a system.

9. Deployment Diagrams

Deployment diagrams describe the configuration of run-time processing resource elements and the mapping of software implementation components onto them. These Diagrams contain components and nodes, which represent processing or computationalresources, including computers, printers, etc.

Rules of the UML:

- The UML building blocks can't simply put together in a random fashion.
- Like any language, the UML has a number of rules that specify what a <u>well-formed model</u>should look like.
- **<u>Well-formed models</u>**
  - Is a model semantically self-consistent and in harmony with all its related models
    - Semantic rules for:
      - **Names** — what you can call things.
      - **Scope** — context that gives meaning to a name.
      - **Visibility** — how names can be seen and used.
      - **Integrity** — how things properly and consistently relate to one another.
      - **Execution** — what it means to run or simulate a dynamic model.
- **<u>Avoid models</u>**

    Elided — certain elements are hidden for simplicity.
    Incomplete — certain elements may be missing.
    Inconsistent — no guarantee of integrity.

Common Mechanisms:

- The UML is made simpler by the presence of some features called common mechanisms.There are four kinds of mechanisms;

  1. Specifications: It provides a textual statement of the syntax and semantics of the buildingblocks.

  2. Adornments: It provides detail from an element's specification added to its basic graphicalnotation.

  3. Common divisions: There is a division of class and object and also the division of interfaceand implementation.

  4. Extensibility mechanisms: The UML provides a standard language for Software blueprints. The UML is opened, making it possible to extend the language in controlled waysto express all possible nuances of all models across all time.

The UML's extensibility mechanisms include

Stereotypes can be used to extend the UML notational elements(*extends vocabulary.)*
      » Name shown in guillemots *<<stereotype>>*

      » Stereotypes may be used to classify and extend associations,inheritance relationships, classes, and components

      » Examples:
            – Class stereotypes: boundary, control, entity,utility, exception
                  – Inheritance stereotypes: includes and extends
            – Component stereotypes:

subsystemEx: << include>>, <<extend>>

Definitions

♦ **System:** A system is a collection of subsystems organized to accomplish a purpose anddescribed by a set of models, possibly from different viewpoints.

♦ **Subsystem:** A subsystem is a grouping of elements, of which some constitute a specification ofthe behavior offered by the other contained elements.

♦ **Model:** A model is a semantically closed abstraction of a system, meaning that it represents a complete and self-consistent simplification of reality, created in order to better understand the system.

♦ **View:** A view is a projection into the organization and structure of a system's model, focused onone aspect of that system.

♦ **Diagram:** A diagram is the graphical presentation of a set of elements, most often represented asa connected graph of vertices (things) and arcs (relationships).

2. Class Diagrams

♦ Class diagram is the most common diagram found in modeling object oriented systems.
♦ Class diagram address the static design view of a system; a diagram that shows a collection ofdeclarative (static) elements.
♦ Graphically, a class diagram is a collection of vertices and arcs.

Philosophy:

A class diagram is a diagram that shows a set of classes, interfaces, collaborations and theirrelationships.

Contents:

Class diagrams commonly contain the following things:

- ● Classes
- ● Interfaces
- ● Collaborations
- ● Dependency, Generalization and association relationships.

Class diagrams also contain notes and constraints for comments.
Class diagrams also contain packages or subsystems used to group the things
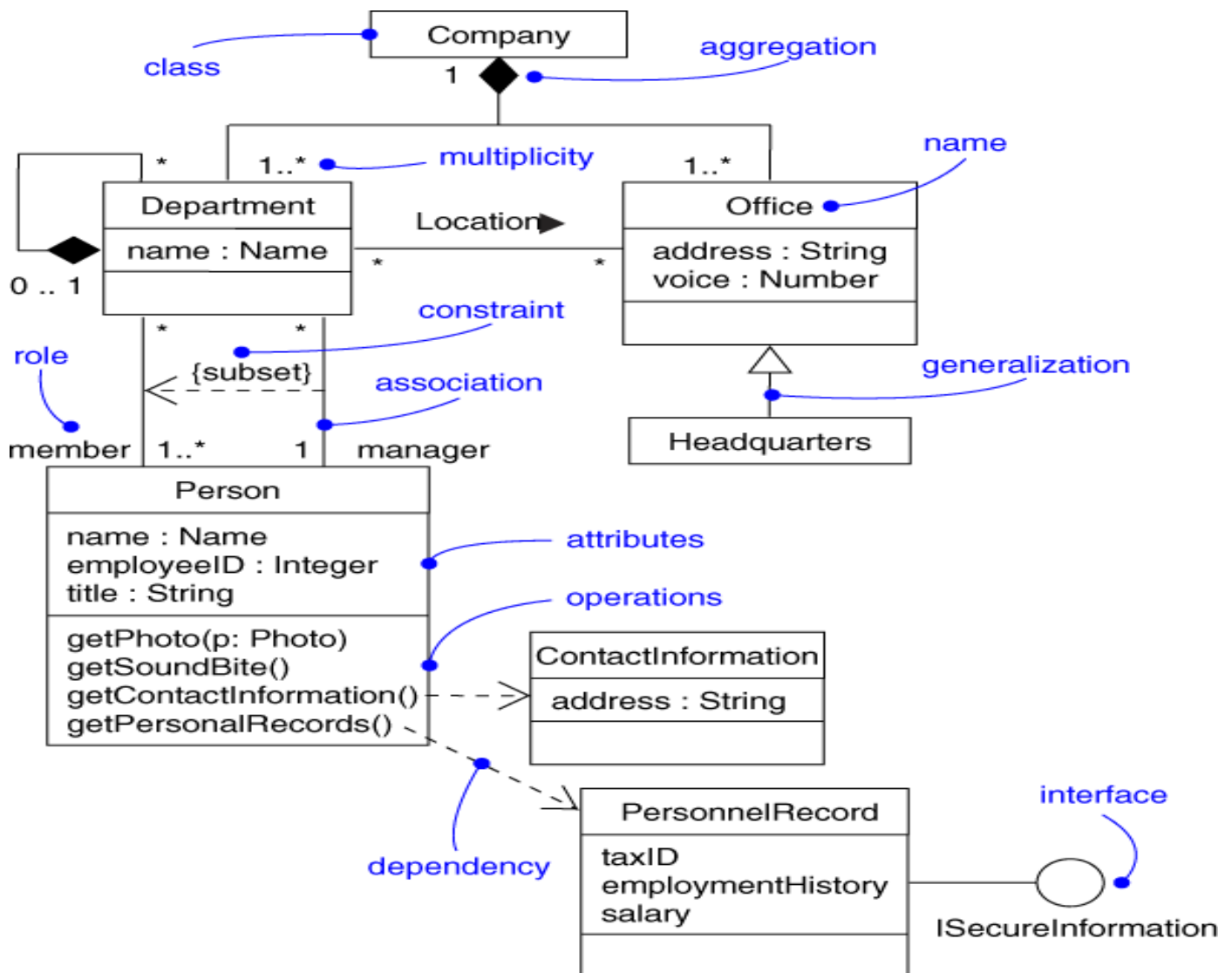
USES

You use class diagrams to model the static design view of a

system.Class diagrams are used
]
- ● To model the vocabulary of a system.
- ● To model the simple collaborations.
- ● To model a logical data base schema.

Example

A simple CLASS DIAGRAM example for a Company

Construction of a Class Diagram

1. Identify the classes that are modeled
    There are alternative approaches for identifying classes:
    - The Noun Phrase Approach
    - Use case Driven Approach

2. Relate the classes, giving them association names.

3. Consolidate similar classes.

4. Identify any appropriate role names.

5. Add classes for any independent functionality required being encapsulated in another class.

6. Add attributes and operations to provide the functionality required in the class diagram.

7. Detail all the operations and attributes giving them data types and parameters.

8. Identify the multiplicity used throughout.

Modeling Techniques

1. Modeling simple collaborations: collaboration is society of classes, interfaces and other elements that work together. Every class is in collaboration with other class. To visualize, specify,construct and document the collaborations, use class diagram as follows

- Identify the mechanisms required for modeling

- For each mechanism, identify the classes, interfaces and other collaborations and alsoidentify the relationships among these things.

- Use scenarios to walk through these things.

- Populate these elements with their contents.

2. Modeling a logical Database Schema: The UML is most suitable to model logical database and physical database schemas. In relational database models the E-R diagrams plays a vital role to model relational databases. Class diagrams are use to model the behavior of a system.

To model a logical database schema:

- Identify the class in the model.

- Design a class diagram and mark the classes as persistent.

- Expand the structured details of the classes.

- Watch for common patterns that complicate physical database design.

- Consider the behavior of classes by expanding operations.

- Use appropriate tools to transform logical design into a physical design.

Forward and Reverse Engineering

Forward Engineering is the process of transforming a model into code through mapping. Forward engineering results in a loss of information, because model written in the UML are semantically richer then any current object-oriented programming language. Structural features such as collaborations, and behavior features such as interactions can be visualized clearly in the UML.

To forward engineer a class diagram,

- Identify the rules for mapping the implementation language.

- Based on the semantics of the language, use certain UML features.

- Use tagged values to specify the target language.

- Use tools to forward engineer the models.

Reverse engineering is the process of transforming code into a model through a mapping from a specific implementation language. Reverse engineering provides good information but is incomplete.

To Reverse engineer a class diagram,

- Identify the rules from mapping from implementation language.

- Using a tool, point to the code we could like to reverse engineer.

- Using the tool, create a class diagram by querying the model.

Points to Remember for construction

A well structured class diagram

- Is focused on communicating one aspect of a system's static design view.
- Contains only elements that are essential to understanding that aspect.
- Provides detail consistent with its level of abstraction, with only those adornments that are essential to understanding.
- Is not so minimalist that it misinforms the reader about important semantics.

When you draw a class diagram,

- Give it a name that communicates its purpose.
- Expose its elements to minimize lines that cross.
- Organize its elements spatially so that things that are semantically close are exposed physically close.
- Use notes and colors as visual cues to draw attention to important features of your diagram
- Try not to show many kinds of relationships. In general, one kind of relationship will tend to dominate each class diagram.

3.   Object Diagrams

♦ Object diagrams model the instances of things contained in class diagrams.
♦ An object diagram is a group of instances.
♦ Graphically, an object diagram is a collection of vertices and arcs.

Philosophy

An object diagram is a diagram that shows a set of objects and their relationships at a pointin time.

Contents:

An object diagram has:

- Objects
- Links

Like all other diagrams, object diagrams may contain notes, constraints, packages andsubsystems.

Uses:

- Object diagrams are used to model object structures.

Modeling Techniques: (construction)

When use objects diagrams, we have to expose the important sets of concrete or prototypicalobjects.

To model an object structure,

- Identify the required mechanism you'd like to model.
- For each mechanism, identify the classes, interfaces and other elements.
- Consider one scenario that walks through this mechanisms.
- Expose the state and attribute values of each object.
- Expose the links among the objects, representing
  instances ofassociations among them.

Example:
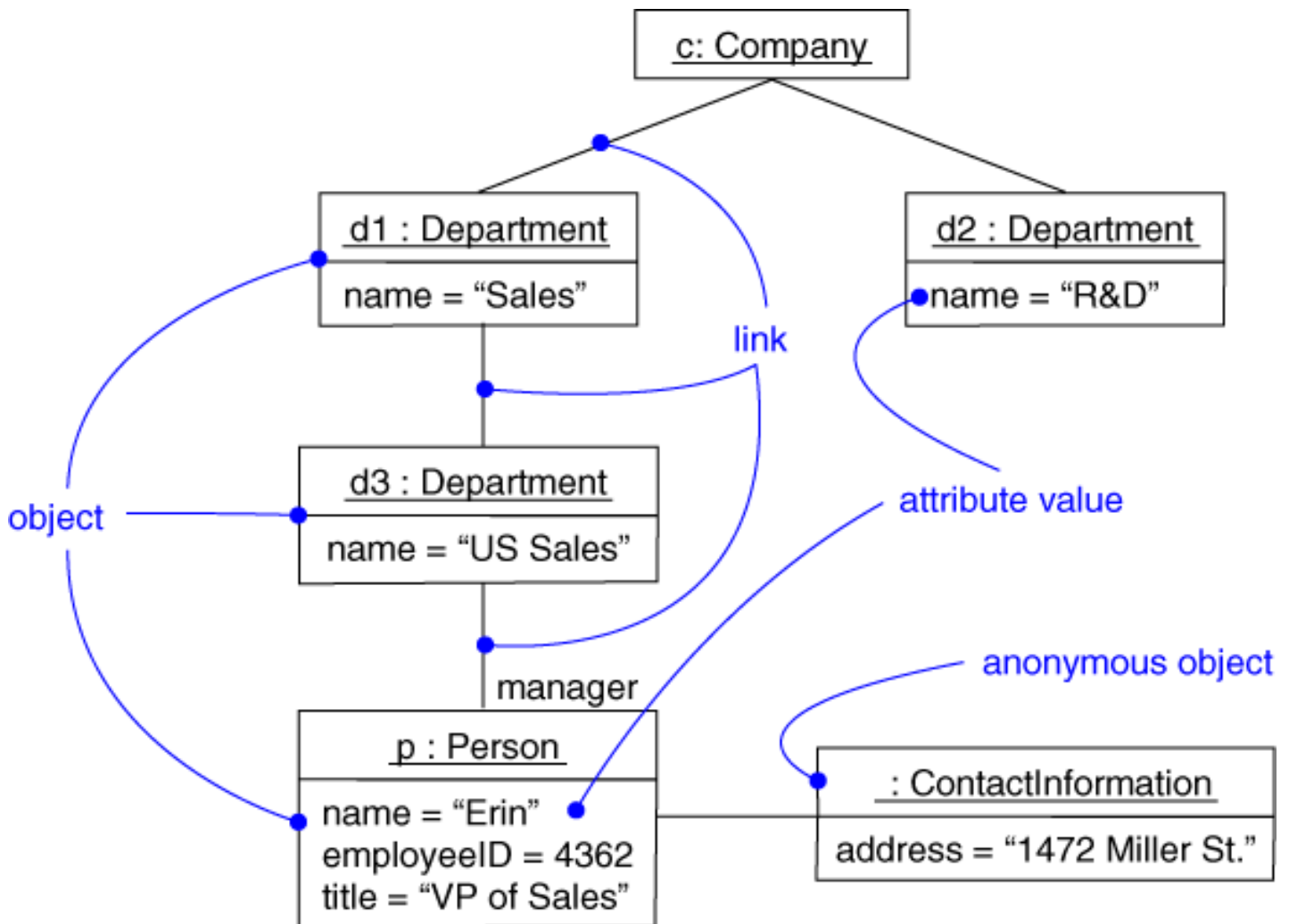
A simple OBJECT DIAGRAM for a Company

Forward and Reverse Engineering:

Forward engineering i.e. the creation of code from a model an object diagram is theoreticallypossible but pragmatically of limited value.

Reverse engineering: It is the creation of a model from code an object diagram is a very differentthing.

To reverse engineer an object diagram,

- Choose the target you want to reverse engineer.
- Using a tool or simply walking through a scenario, stop execution at a certain momentin time.
- Identify the set of interesting objects that collaborate in that context and render them inan object diagram.
- As necessary to understand their semantics, expose these object's states.
- As necessary to understand their semantics, identify the links that exist among theseobjects.

Points to remember for construction

A well-structured object diagram

- Is focused on communicating one aspect of a system's static design view or staticprocess view.
- Represents one frame in the dynamic story board represented by an interaction diagram.
- Contains only those elements that are essential to understanding that aspect.
- Expose only attribute values and other adornments that are essential to understanding.

When you draw an object diagram,

- Give it a name that communicates its purpose.
- Expose its elements to minimize lines that cross.
- Organize its elements spatially so that things that are semantically close are exposedphysically close.
- Use notes and colors as visual cues to draw attention to important features of yourdiagram.
- Include the values, state and role of each object as necessary to communicate your idea.

4. **Interaction Diagrams**

Introduction

Sequence diagram and collaboration diagrams- both of which are called interaction diagrams. These two diagrams are used in the UML for modeling the **dynamic aspects ofsystems.**

An interaction diagram shows an interaction, consisting of a set of objects and their relationships including messages that may be dispatched among them.

An interaction diagram that emphasizes the time ordering of messages called <u>Sequence</u> **diagram.**

An interaction diagram that emphasizes the structural organization of the objects that sendand receive messages called **collaboration diagram.**

Contents:

Interaction diagram commonly contain

- Objects
- Links
- Messages

Interaction diagram may also contain notes and constraints.

1. **Sequence Diagram:**

- A sequence diagram is an interaction diagram that emphasizes time ordering ofmessages.
- A sequence diagram shows the objects participating in the interaction by their lifelinesand the messages that they exchange/ arranged in time sequence.
- Sequence diagrams show the explicit sequence of messages and are better for real-timeSpecification and for complex scenarios.

- A sequence diagram has two dimensions.
  - Horizontal axis: represents different objects.
  - Vertical axis: represents time i.e. it shows the sequence of messages

► Sequence diagram has two features that differentiate them from collaboration diagram.

1. Object Lifeline: The lifeline represents the existence of the object at a particular time. An object lifeline is the vertical dashed line that represents the existence of an object over a period of time.

2. Focus of control: The focus of control is a tall, thin rectangle that shows the period oftime during which an object is performing an action.

Uses:

- To model the flows of control by time ordering.

Construction of a Sequence diagram:

- Identify the workflows that you model as separate sequence diagrams.
- Expose the objects for the individual sequence diagrams.
- Include messages and conditions in the order of control flow for your individualSequence diagrams.

Modeling Techniques:

To model a flow of control by time ordering

- Set the context for the interaction, whether it is a system, subsystem, operation, or class,or one scenario of a use case or collaboration.
- Set the stage for the interaction by identifying which objects play a role in the Interaction. Expose the objects on the sequence diagram from left to right, placing themore important objects to the left and their neighboring objects to the right.
- Set the lifeline for each object.
- Starting with the message that initiates this interaction, Expose each subsequent message from top to bottom between the lifelines. Showing each message's properties,as necessary to explain the semantics of the interaction.
- If you need to visualize the nesting of messages or the points in time when actual computation is taking place, represent each object's lifeline with its focus of control.
- If you need to specify time or space constraints, represent each message with timingmark and attach suitable time or space constraints.
- If you need to specify this flow of control more formally, attach pre and post conditionsto each message.

Example

:Library User

Lookup

Display

Issue

Issue licence

Accept licence

Compress

Deliver
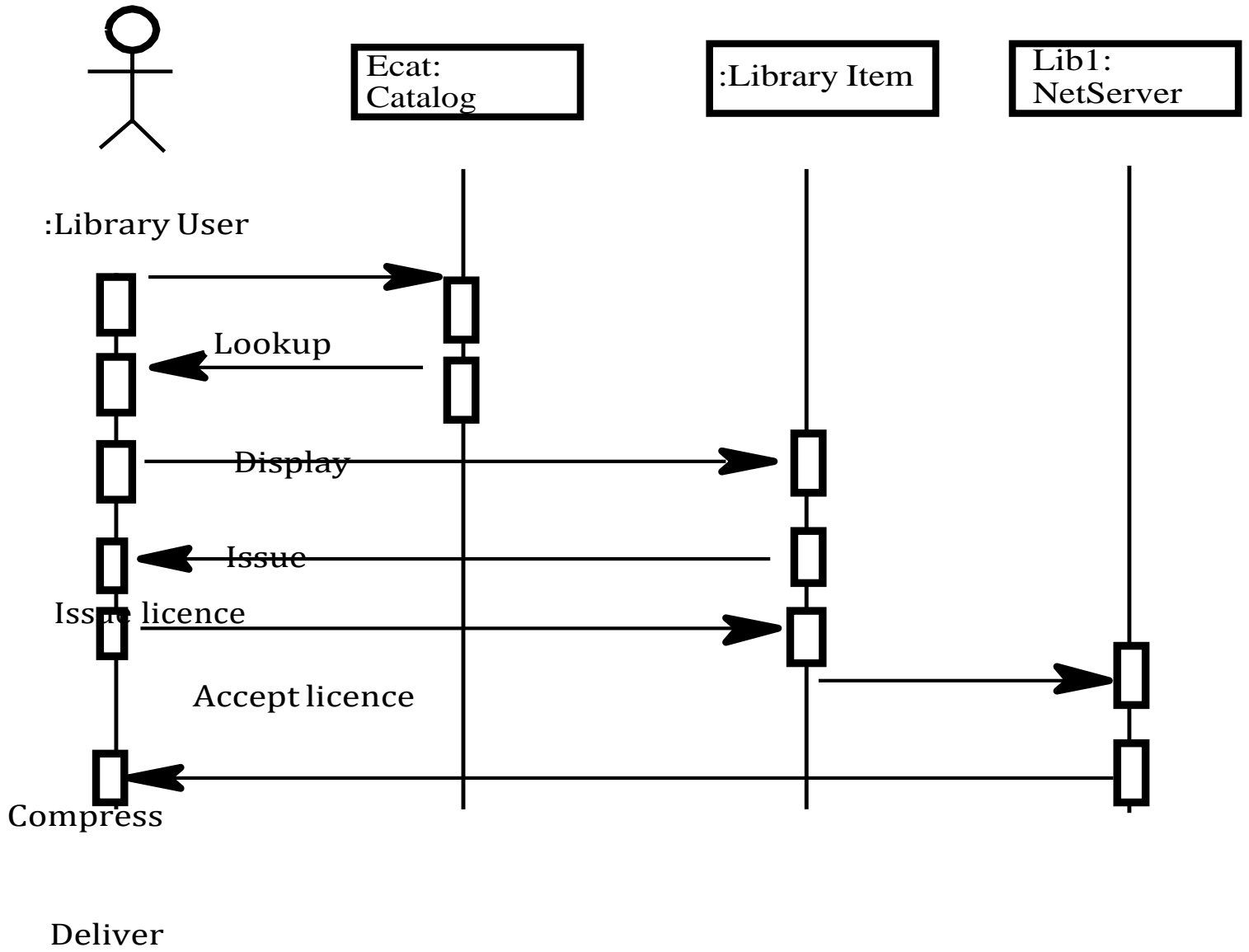
2. Collaboration Diagram:

A Collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages called **collaboration diagram.**

Construction

- First identify the objects that participate in interaction. And adorn them as the vertices ina graph.
- Secondly establish the connection between the objects using links.
- Finally, adorn these links with the messages that objects send and receive messages.

Collaboration diagram has two features that distinguish it from sequence diagram.

1. Path: How one object is linked to another object. We can attach a path stereo type to the far endof a link.

2. Sequence Number: It indicates the time order of messages.

Uses:

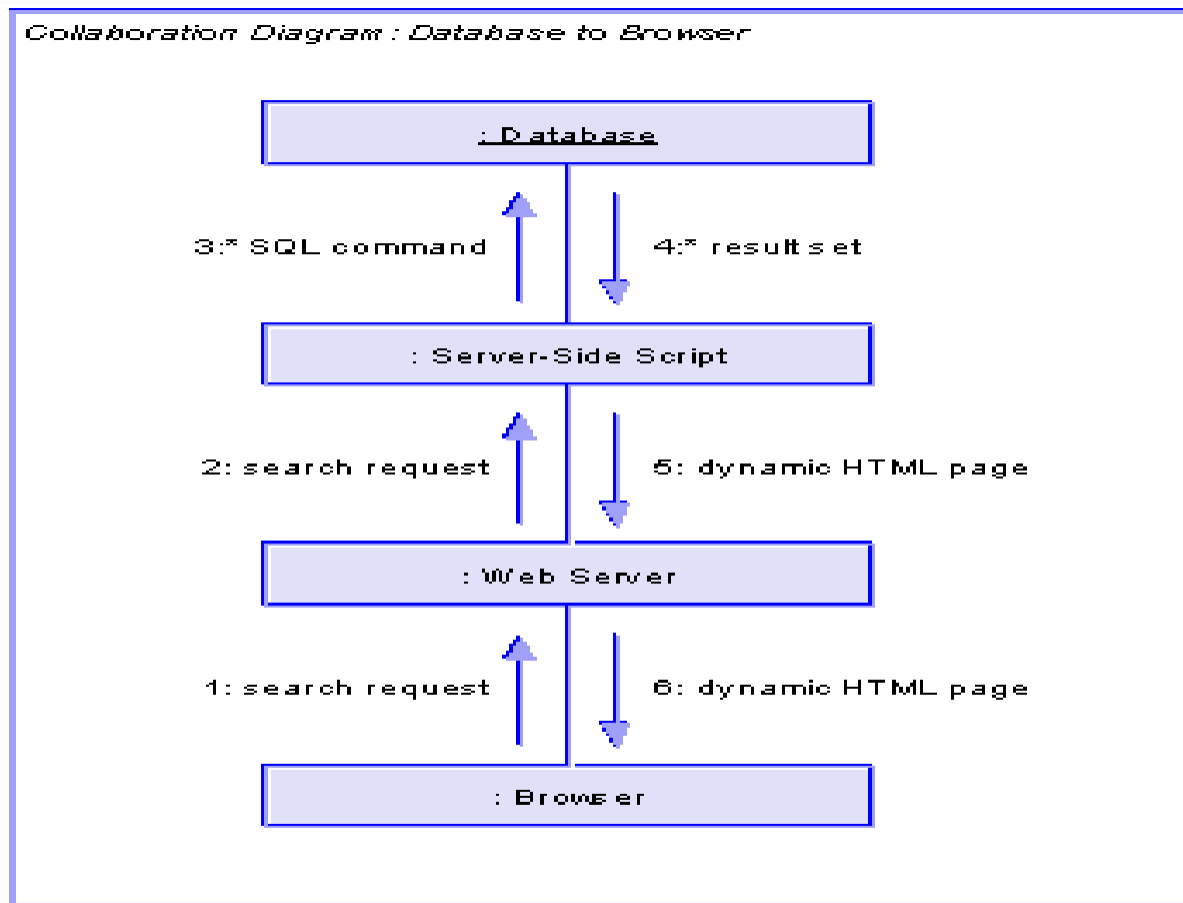- To model the flows of control by organization.

Modeling Techniques:

To model flow of control by organization:

- Set the context for the interaction, whether it is a system, subsystem, operation, or class,or one scenario of a use case or collaboration.
- Set the stage for the interaction by identifying which objects play a role in the interaction. Expose them on the collaboration diagram as vertices in a graph, placing the more important objects in the center of the diagram and their neighboring objects to the outside
- Set the initial properties of each of these objects.
- Specify the links among these objects, along which messages may pass.

   1. Expose association links first; these are most important ones, because theyrepresent structural connections.

   2. Expose the other links if necessary and adorn them withsuitable stereotypes.

▪ Starting with the message that initiates this interaction, attach each subsequent message to the appropriate link. Setting its sequence number, as appropriate. Show the nesting sequenceusing decimal numbering.

▪ If you need to specify time or space constraints, represent each message with a timingmark and attach suitable time or space constraints.

▪ If you need to specify this flow of control more formally, attach pre and post conditions toeach message.

Example:

Collaboration Diagram : Database to Browser

: Database

3:* SQL command   4:* result set

: Server-Side Script

2: search request   5: dynamic HTML page

: Web Server

1: search request   6: dynamic HTML page

: Browser

Points to remember when you draw an interaction diagram:

A well structured interaction diagram

- ▪ Is focused on communicating one aspect of a system's dynamics.
- ▪ Contains only those elements that are essential to understanding that aspect.
- ▪ Provides detail consistent with its level of abstraction and should expose only thoseadornments those are essential to understanding.
- ▪ Is not so minimalist that it misinforms the reader about semantics that are important.

When you draw an interaction diagram

- ▪ Give it a name that communicates its purpose.
- ▪ Expose its elements to minimize lines that cross.
- ▪ Organize its elements spatially so that things that are semantically close are exposedphysically close.
- ▪ Use notes and colors as visual cues to draw attention to important features of yourdiagram.
- ▪ Include the values, state and role of each object as necessary to communicate your idea.

5. Behavioral Modeling

Introduction

USE CASES

- A *use case* is a description of a set of sequences of actions, including variants, and various *observable result* of value to an *actor*.
- Each sequence indicates the relation with outside things.
- A use case involves the interaction of actors and the system.

Names
- Every use case must have a name that distinguishes it from other use cases.
- A name alone is known as a simple name.

Use cases and Actors:

- An actor is a user playing a role with respect to the system.
- An actor models an external entity which communicates with the system by sending and receiving messages.
- An actor has a unique name and an optional description to represent roles or devices play as the system functions.
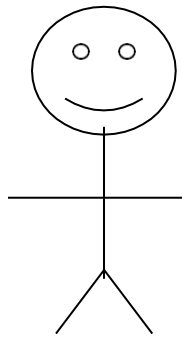
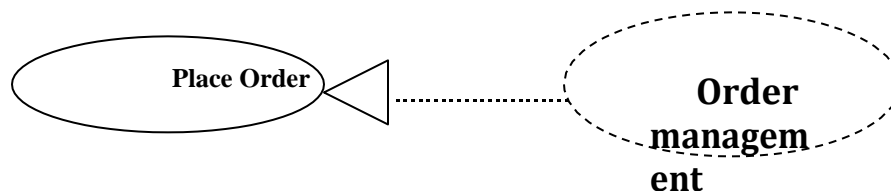

Actor

Use cases and flow of events

- A use case describes what a system does but it does not specify how it does it.
- We can specify the behavior of a use case by describing a flow of events.
- When you write the flow of events, you should include how and when the use case starts and ends.

Use cases and Scenarios:

- Generally a use case describes a set of sequences. It is not possible to describe all the details of a use case with one sequence.
- A scenario is a specific sequence of actions that describes behavior. Scenarios are use cases as instances are to classes.
- For each use case, there is a primary scenario and secondary scenarios.

Use cases and collaborations

- A use case considers required collaborations.
- Use cases are implemented by creating a society of classes and other elements that work together to implement the behavior of used case.
- The society of elements, including both its static and dynamic structure is modeled.

Modeling a Use Case

- Identify the actors that interact with the element.
- Organize actors by identifying general and more specialized roles.
- For each actor, consider the primary ways in which that actor interacts with the element.
- Consider also the exceptional ways in which each actor interacts with the element.
- Organize these behaviors as use cases, applying include and extend relationships to factor common behavior and distinguish exceptional behavior.

1.Use case Diagrams

Introduction

Use case diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems. Use case diagrams are central to modeling the behavior of a system, or a class. Each one shows a set of use cases and actors and their relationships.

Use case diagram is a graph of actors, a set of use cases enclosed by a system boundary. Use case diagrams are important for visualizing, specifying and documenting the behavior of an element.

Contents:

- Use cases
- Actors
- Dependency, Generalization, and association relationships.
- System boundary

Use case diagrams may contain notes and constraints to include comments. Also it contains packages, which are used to group elements of your model into larger chunks.

Uses:

- To model the context of a system.
- To model the requirements of a system.

Modeling Techniques:

To model the context of a system:
- Identify the actors that surround the system by considering which groups require help from the system to perform their tasks.
- Organize actors those are similar to one another in a generalization hierarchy.
- Where it aids understandability, provide a stereotype for each such actor.

● Populate a use case diagram with these actors and specify the paths of communication from each actor to the systems use cases.

To model the requirements of a system:
- Establish the context of the system by identifying the actors that surround it.
- For each actor, consider the behavior that each expects or requires the system to provide.
- Name these common behaviors as use cases.
- Model these use cases, actors, and their relationships in a use case diagram.
- Adorn these use cases with notes that assert nonfunctional requirements.

Construction of a Use case diagram:

- What are the main tasks or functions that are performed by the actor?
- What system information will the actor acquire, produce or change?
- Will the actor has to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Identifying the Actors:

- Who is utilizing the system?
- Which groups need help from the system to carry out a task?
- Which user groups are needed by the system to discharge the functions?
- Which external hardware or other systems use the system to execute tasks?
- What problems does the application solve?
- How the system is being used by the users? What are they doing with the system?

Identifying use cases:
- One of the methods used to identify use cases is **actor-based.**
  — The actors related to a system or organization is identified.
  — The processes, tasks, functions that are performed by each actor are identified.
- A second method to identify use cases is **event- based.**
  — **The** external events that a system must respond to are identified.
  — **Relate** the events to actors and use cases.

Forward Engineering:

— For each use case in the diagram, identify its flow of events and its exceptional flow of events.

— Depending on how deeply you choose to test, generate a test script for each flow, using the flow's preconditions as test's initial state and its postconditions as its success criteria

— Use tools to run these tests each time you release the element to which the use case diagram applies.

Reverse Engineering:

— Identify each actor that interacts with the system.

— For each actor, consider the manner in which that actor interacts with the system, changes the state of the system or its environment , or responds to some event.

— Trace the flow of events in the executable system relative to each actor. Start with primary flows and only later consider alternative paths.

— Represent these actors and use cases in a use case diagram, and establish their relationships.

## Points to Remember for constructing a use case diagram:

A well structured Use case diagram

— Is focused on communicating one aspect of a system's static use caseview.

— Contains only those use cases and actors that are essential to understandingthat aspect.

— Provides detail consistent with its level of abstraction.

— Is not so minimalist as to misinform the reader about semantics that are important.

When you draw a Use case diagram,

- Give it a name that communicates its purpose.
- Expose its elements to minimize lines that cross.
- Organize its elements spatially so that things that are semanticallyclose are exposed physically close.
- Use notes and colors as visual cues to draw attention to important features of your diagram.
- Try not to show many kinds of relationships. In general, one kind of relationship will tend to dominate each class diagram

Example:
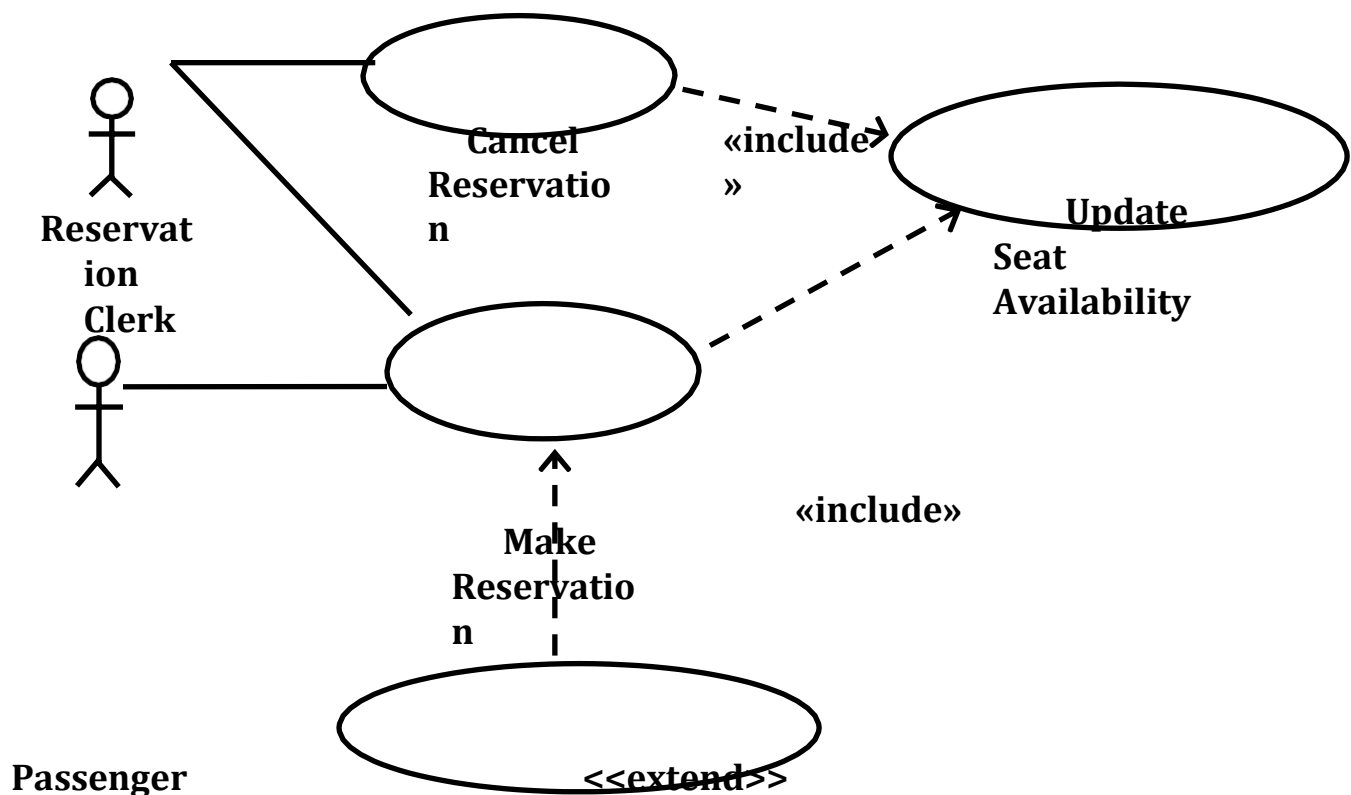
Generate Payment Failure Notice

6. Activity Diagrams

Introduction:

- An Activity Diagram is one of the five diagrams in the UML for modeling the dynamic aspects of systems.

- An activity diagram is a special kind of <u>state diagram.</u>

- An activity diagram is like a flow chart, showing flow of control from activity to activity. An activity is an ongoing nonatomic execution within a state machine.

- Activity diagrams may stand alone to visualize, specify, construct, and document the dynamic aspects of a society of objects, or they may be used to model the flow of control of an operation.

- Activity diagrams are used in situations where all or most of the events represent the completion of internally generated actions.

- Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression.

- Activity diagrams best for modeling business- level functions.

Contents:

An activity diagram contains:
- Activity states and action states
- Transitions
- Objects

Action state: In the flow of control modeled by an activity diagram, things happen. You might evaluate some expressions that sets the value of an attribute or that returns some value.

Ex: call an operation, create or destroy objects… …………… etc.

It is represented as a symbol with horizontal top and bottom and convex sides.

Activity state: An action state is an activity state, that can't be further decomposed. And activity state is a composite state, whose flow of control is made up of other activity states and action states.

Transitions:

- When the action or activity of a state completes, flow of control passes immediately to the next action of activity state. You can specify this flow by using transition to show the path from one action state to another.

- The transition is represented as a simple directed line.

Branching:

- A branch specifies alternate paths taken based on some Boolean expression.
- It is represented as diamond shaped symbol.
- A branch may have one incoming transition and two or more outgoing transitions.

Forking and Joining:

- To represent concurrent work flows; in UML we can use the synchronization bar.
- A synchronization bar is represented as a thick horizontal line.
- A Fork represents the splitting of a single flow of control into two or more concurrent flows of control. i.e. a fork may have one incoming transition and two or more outgoing transitions.

- A <u>Join</u> represents the synchronization of two or more concurrent flows of Control. i.e. a join may have two or more incoming transitions and one outgoing
    Transition.
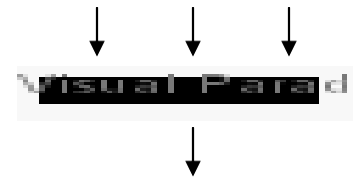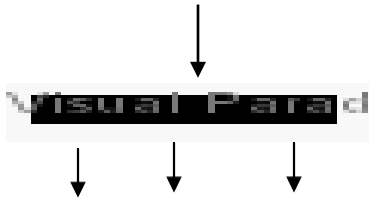
<u>Swim lanes:</u>
- Swim lane is a kind of package for organizing responsibility for activitiesWithin a class.

<u>Object Flow:</u>

- Objects may be involved in the flow of control associated with an activity diagram. It is also possible to show object's role, state and attribute value changes.

<u>Common uses:</u>

- To model a work flow.
- To model an operation.

<u>Modeling Techniques:</u>

To model a work flow:
- Establish a focus for the workflow. For nontrivial systems, it's impossible toshow all interesting workflows in one diagram.

- Identify the preconditions of the workflow's initial state and the post conditions of the workflow's final state.

- Beginning at the workflow's initial state, specify the activities and actions thattake place over time and render them in the activity diagram as either activity or action states.

▪ Render the transitions that connect the activity and action states. Start with the sequential flow in the workflow first, next consider branching, and only then consider forking and joining.

▪ For the complicated actions, or for sets of actions that appear multiple times, collapse these into activity states, and provide a separate activity diagram that expands on each.

▪ If there are important objects that are involved in the workflow, render them an activity diagram, as well. Show their changing values and state as necessary to communicate the intent of the object flow.

Modeling an operation:

▪ Collect the abstractions that are involved in this operation. This includes the operation's parameters, the attributes of eh enclosing class, and certain neighboring classes.

▪ Identify the preconditions at the operation's initial state and the post conditions at the operation's final state.

▪ Beginning at the operation's initial state, specify the activities and actions that take place over time and render them in the activity diagram as their activity states and action states.

▪ Use branching as necessary to specify conditional paths and iteration.

▪ Only if this operation is owned by an active class, use forking and joining as necessary to specify parallel flows of control.

Example:
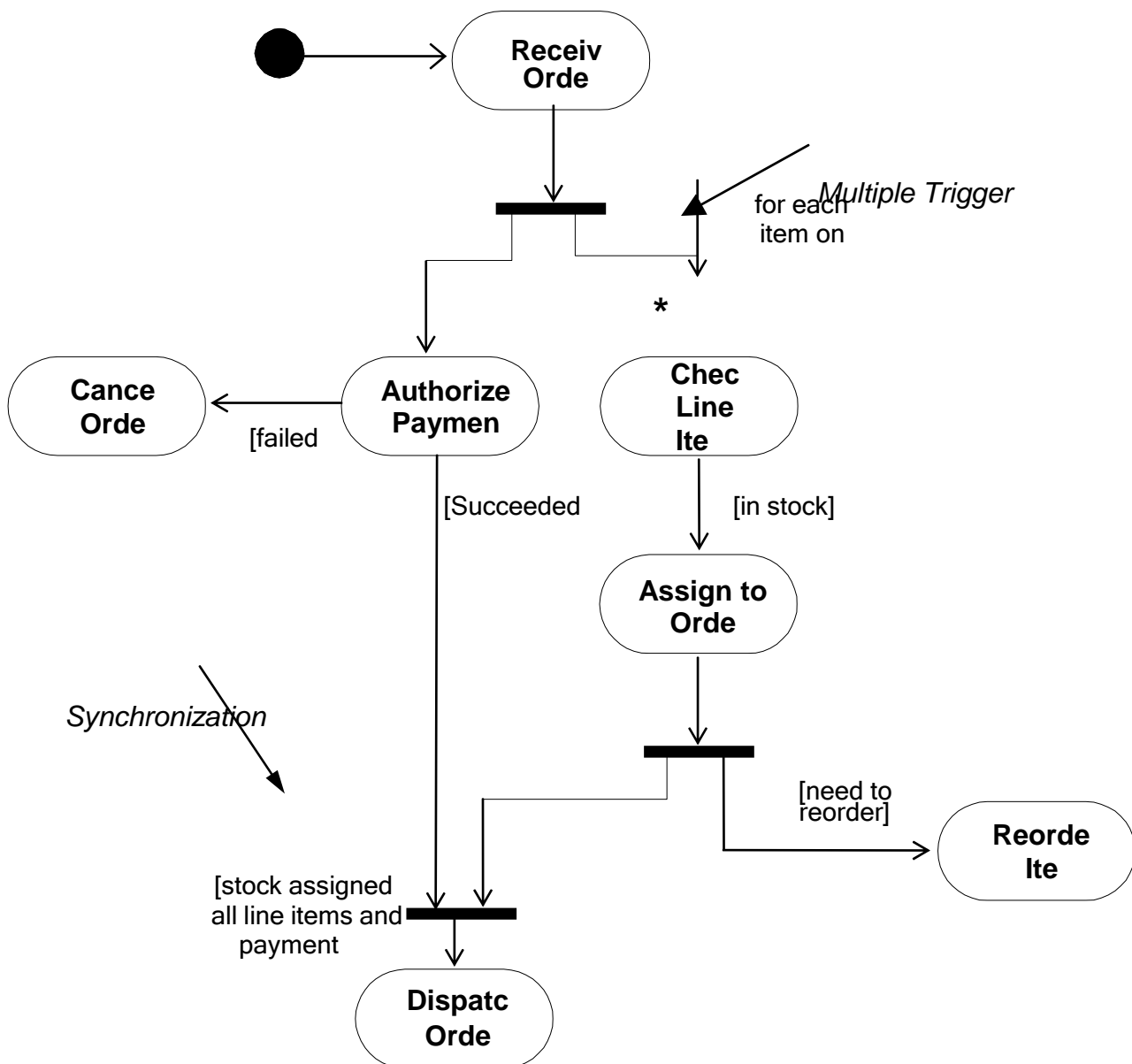
7. Advanced Behavioral Modeling

Introduction

Events:

- An <u>event</u> is the specification of a significant occurrence that has a location in time and space.
- Event is a designated condition becoming true.
- In the context of state machines, we can use events to model the occurrence of a stimulus trigger a state machine.
- In the UML, we can model four kinds of events: signals, calls, the passing of time, and a change in state.
- Events may be either external or internal.
    — External events are those that pass between the system and its actors.
    — Internal events are those that pass among the objects that live inside the system.

Signals:

- A signal represents a named object that is dispatched asynchronously by one object and then received by another.
- Signal may have instances.
- Signals may also be involved in generalization relationships.
- A signal may be sent as the action of a state transition in a state machine or the sending of a message in an interaction.

State Machines

- A state machine is one of the five diagrams in the UML for modeling the dynamic aspects of systems.
- A state machine is used to model the behavior of an object.
- State machines are also used to emphasize the flow of control between objects.
- In the total life time of an object, it may be exposed to many events such as creations or an object, destruction of the object, state changes, message

Passing...etc.

- Using a state machine, we can model the behavior of an individual object.
- A state machine can be visualized in two ways:

    Activity diagrams: In activity diagrams we can emphasize theflow of control from activity to activity.

    State chart diagrams: In state chart diagrams we can emphasizethe flow of control from state to state.

Sub states:

- A substrate is one which is nested in another state. Some state may have Substrates which inherit or derive the transitions and attributes of their Super states.

- All transition or actions applied to superstates are also eligible to apply on its substrates.
- It is also called as *Composite state.*

Sub states can also be classified as:

- **Sequential *Sub states*:** These states are used to decompose the state space ofthe composite state into disjoint states. It has atmost one ‗begin' and one and ‗end' state.

- *History States:* It remembers the history of the immediate nested state machine.

    It is categorized as shallow history and deep history which are symbolically represented as **H** and **H*** respectively.

- *Concurrent Sub states:* It allows us to specify two or more state machines Which are executed in parallel.

USES:

    To model the life time of an object**.**

State chart Diagrams

A State chart diagram shows the state machine focusing on the flow of controlfrom state to state.

In the UML, State chart diagrams are used to model the behavior aspects of a system.

A State chart diagram comprises states and events. A State is defined as the situation in the life of an object. An event can trigger a state transition. The relationship between the states can be represented by a transition.

Contents:

- Simple states and composite states
- Transitions, including events and actions

Uses:

- To model reactive objects.
- A state chart diagram is useful to model the dynamic aspects of the system.
- State chart diagrams are also used to describe the behavior of a single class of objects.

Modeling Techniques:

To model reactive objects:

A reactive object is one, which responds only when an event occurs.

- Identify whether the context of the system is a class or use case.
- Find the initial and final states.
- Decide the ordering of stable states.
- Find the events and their transitions.
- Attach actions to the identified transitions
- Use the required tools, such as forks, joins, sub states….etc to the state machine.

Forward and Reverse Engineering

Forward engineering is possible for state chart diagrams, especially in the context of the diagram is a class.

Reverse engineering is theoretically possible, but practically not very useful.
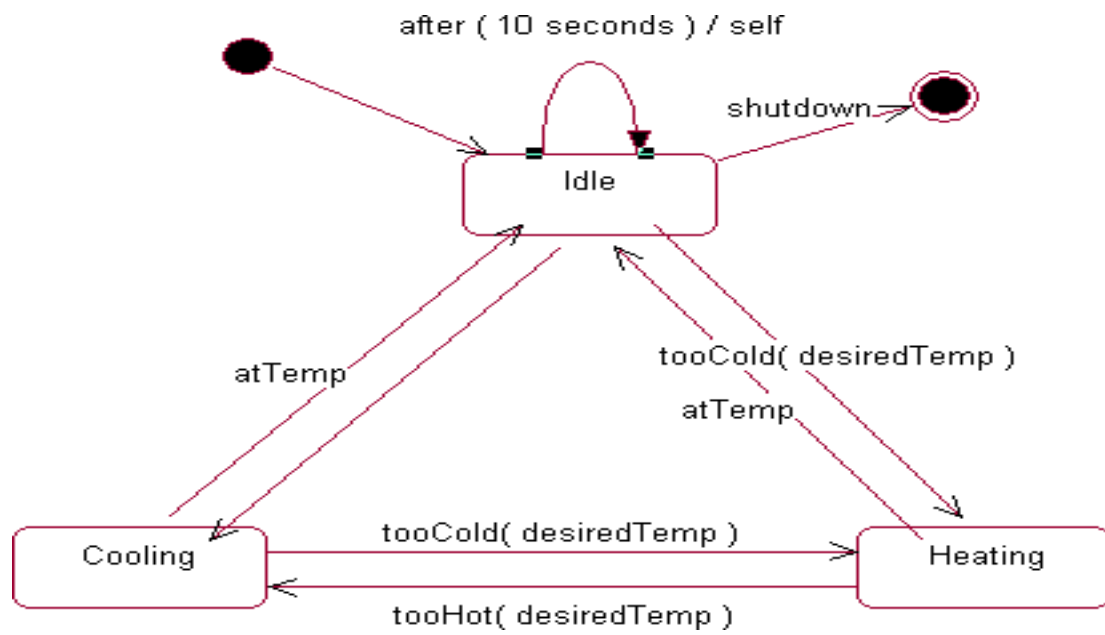
Points to remember:

A well structured state chart diagram

- Is focused on communicating one aspect of a system's dynamics.
- Contains only those elements that are essential to understanding that aspect.
- Provides detail consistent with its level of abstraction.
- Uses a balance between the styles of Melay and Moore machines.

When you draw a state chart diagram

same as like Activity diagram.

Example:

after (10 seconds) / self

shutdown

Idle

tooCold(desiredTemp)

atTemp

atTemp

Cooling

tooCold(desiredTemp)

tooHot(desiredTemp)

Heating

8. Architectural Modeling

1. Component Diagrams

Introduction

- Component diagrams are one of the two kinds of diagrams used in modeling the physical aspects of OOsystems.
- A component diagram shows the organization and dependencies among a set of components.
- A ***Component*** is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
- A ***Component*** is rendered as a rectangle with tabs.
- It is also useful for constructing executable systems through forward and reverse engineering.

Philosophy:

A component diagram shows a set of components and their relationships. Graphically, a component diagram is a collection of vertices and arcs.

Common Properties:

A component diagram is just a special kind of diagram and shares the same common properties as do all other diagrams --- a name and a graphical contents that are a projection into a model.

Contents:

Component diagram contains
- Components
- Interfaces
- Dependency, generalization, association, and realization relationships. Like all other diagrams component diagrams contains notes and constraints.

Uses:

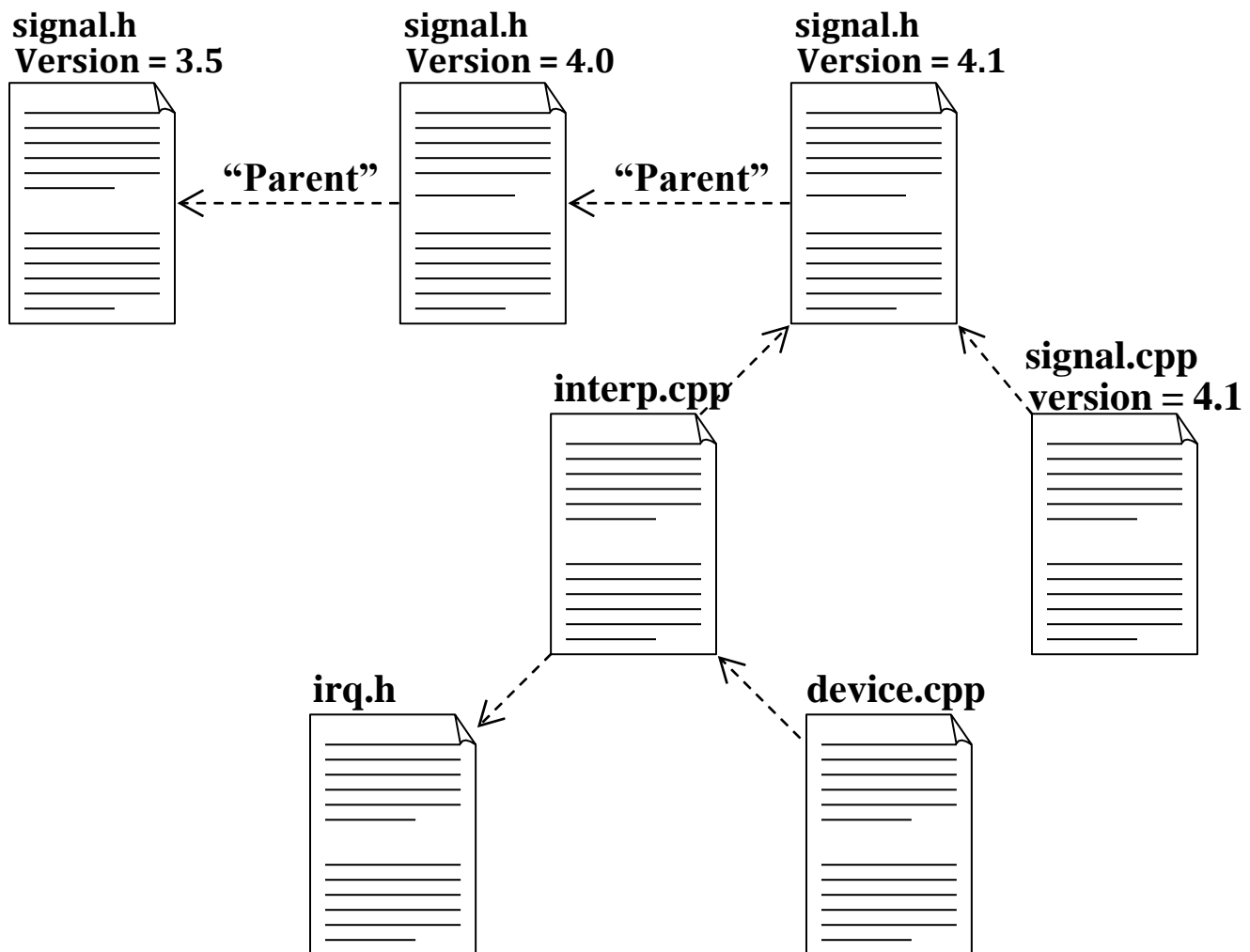We can use component diagrams to model the static implementation view of asystem.

- To model source code
- To model executable releases
- To model physical databases
- To model adaptable systems

Modeling Techniques:

To model a system's source code

- Identify the different files used in the source code.
- Under the configuration management, use the tools such as tagged values, author, version number...etc, for each file.
- Using UML, visualize and document the relationships noted in the following scenario.
- For larger systems, use packages to show groups of source code files.

Example:

signal.h
Version = 3.5

signal.h
Version = 4.0

signal.h
Version = 4.1

"Parent"

"Parent"

interp.cpp

signal.cpp
**version = 4.1**

irq.h

device.cpp

To model executable releases:

- With the help of reuse and configuration management issues, decompose the system.
- With the help of stereotypes model executables and libraries as

components.
- model the interfaces which are used by some components and are realized by other.
- Model the relationships among the executables , libraries and interfaces.

path.dll

collision.dll

driver.dll version = 8.1.3.2

IDrive

ISelfTest
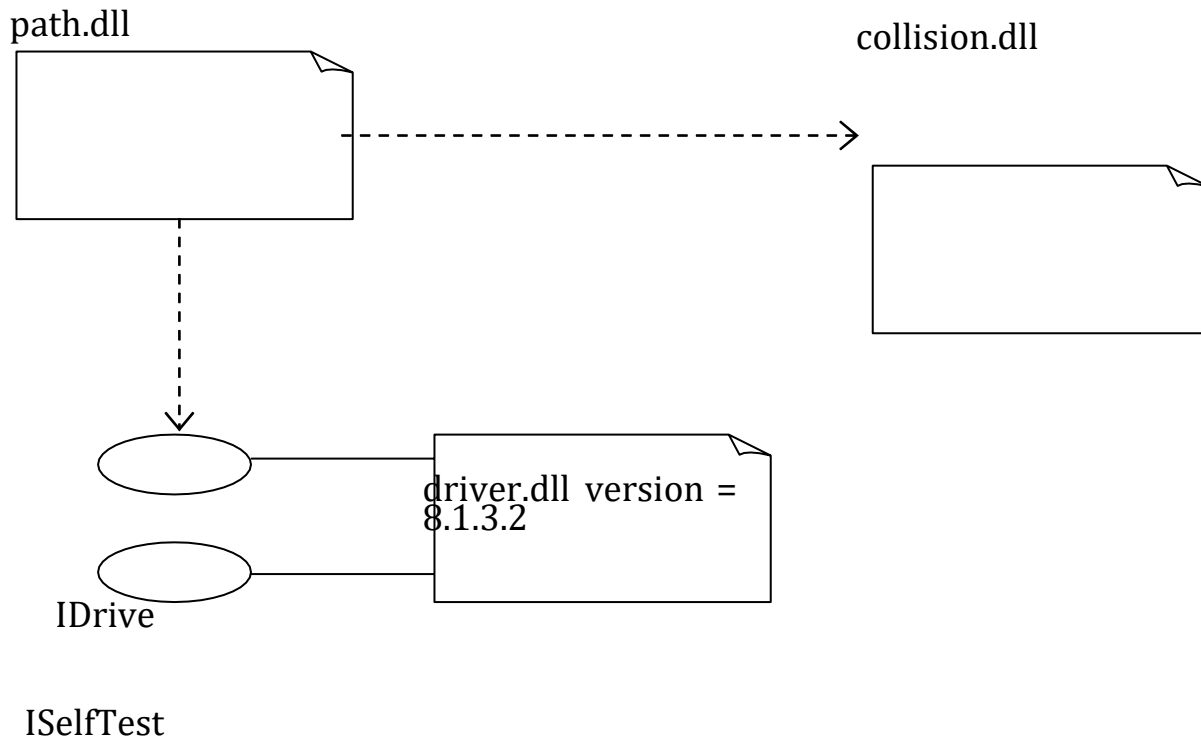
Modeling a Physical Database:

Generally, databases can be represented by tables. The logical database scheme is identified by the vocabulary of the system. It is cumbersome to map logical databases on to physical databases.

Typically, you can apply one or a combination of three strategies:

1. Define a separate table for each class.
2. Collapse your inheritance lattices so that all instances of any class in a hierarchy has the same state.
3. Separate parent and child states into different tables.

Also use Structural Query Languages (SQL) for some simple creation, deletion, reading, update, reading operations. Using Triggers and procedures for complex operations.

- Identify the logical and physical database schemas.
- Use methods to map the classes to tables.
- Using component diagrams, visualize the mapping.
- With the required tools map the logical scheme into physical database.

To model adaptable systems:

A distributed system may transfer the data from one server to the other database server. To model these dynamic views, we need to use object, component and interaction diagram.

- ▪ Identify the components which will be transferred from node to node.
- ▪ Consider the actions which forces the component to transfer and also create an interaction diagram.

2. Deployment Diagrams:

Introduction

- Deployment diagrams are one of the two kinds of diagrams used in modeling the <u>physical aspects of OOsystems</u>.
- A deployment diagram shows the configuration of <u>run time processing nodes</u> and the components that live on them.
- Deployment diagrams are essentially class diagrams that focus on a system's nodes.
- It is useful to model the static deployment view of a system.
- It is also useful for managing executable systems through forward and Reverse engineering.
- A deployment is represented as a <u>node</u>. A node is a physical element thatexists at run time and represents a computational resource.

Philosophy:

A deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Graphically, a deployment diagram is a collection of vertices and arcs.

Common properties:

A deployment diagram is just a special kind of diagram and shares thesame common properties as all other diagrams.

Contents:

A deployment diagram commonly contains:
- Nodes
- Dependency and association relationships

Like all other diagrams , deployment diagrams may contain notes and constraints.

Deployment diagrams may also contain components, each of which must live on somenode.

Uses:

1. To model embedded systems.
2. To model client/ server systems.
3. To model fully distributed systems.

Modeling embedded systems:

Now a days embedded systems are playing very important role in all areas like wrist watches, motors, temperature controllers…etc. An embedded system is a collection of software and hardware systems in the physical world. Deployment diagrams are used to model embedded systems.

Ex: Deployment diagrams are useful to establish communication between s/w develops and h/w engineers.

- Identify the nodes and the devices required for the system.
- Specify the UMLs mechanism and stereotypes with appropriate icons.
- Establish the relationships among the processors and devices.
- With the help of a well structured deployment diagram, expand the existing devices.

Modeling a Client / Server system:

In many data communication applications, clients and server interact to communicate the data passage. In some systems, a system is partitioned into clients and server. A server is always a master and the clients are slaves. That is clients will request a server and the server will always respond. Clients are classified as thin and thick clients.
A thin client has limited computational power. A thick client has good computational power.

- Identify the nodes which represent the client and server processors.
- Identify the devices that exploits the behavior of the system.
- Provide necessary mechanisms for the above mentioned processors and devices
- Design a deployment diagrams of these nodes.

Modeling a Distributed System:

Distributed systems may spread across many geographical areas. With distributed systems parallel communication paths will be established.

▪ Identify the system's devices and processors for client / server systems.
▪ If there is any change to the network or the performance of the n/w then model these communication devices accordingly.
▪ Identify the nodes and their corresponding packages.
▪ Model the devices and processors using deployment diagrams.
▪ Identify the use case diagrams to specify the behavior.

Example:
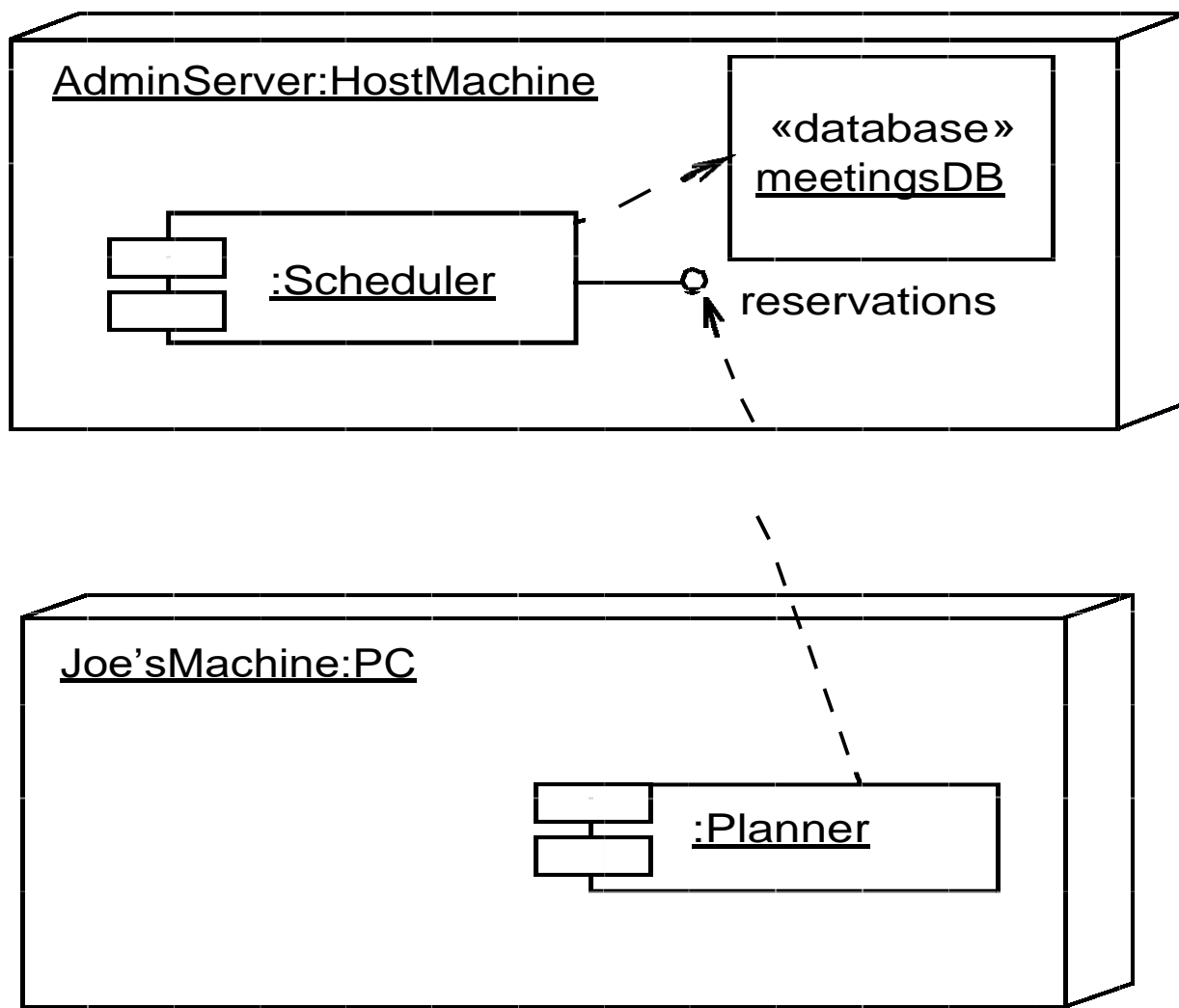
Forward and Reverse Engineering:

Forward engineering is the process of producing the executable code from a logical or physical model.
- Identify the components and simulate them to the real world.

Reverse engineering is a process of producing a logical or physical model from the executable code.

- Identify the target element to reverse engineer.
- Walk across the system to discover the h/w structure.
- Walk across the system to discover the components.
- With the help of modeling techniques, design, a create a deployment diagram.

Points to Remember for construction

A well structured class diagram

- Is focused on communicating one aspect of a system's static design view.
- Contains only elements that are essential to understanding that aspect.
- Provides detail consistent with its level of abstraction, with only those adornments that are essential to understanding.
- Is not so minimalist that it misinforms the reader about important semantics.

When you draw a class diagram,

- Give it a name that communicates its purpose.
- Expose its elements to minimize lines that cross.
- Organize its elements spatially so that things that are semantically close are exposed physically close.
- Use notes and colors as visual cues to draw attention to important features of your diagram
- Try not to show many kinds of relationships. In general, one kind of relationship will tend to dominate each class diagram.