CHAPTER 3

The Techniques of UML

Unified Modeling Language (UML) was defined in October 1994 by Grady Booch and Jim Rumbaugh. In 1995 Ivar Jacobson merged his method, the OOSE, with UML [19]. UML serves as a basis for representing most methods using a common set of modeling constructs and a common notation. Basically, they have combined the concepts of the Booch method, OMT (Object Modeling Technique) and OOSE (Object-Oriented Software Engineering) to develop UML [20]. UML is a visual modeling language that is specially designed to specify, visualize, construct and document the artifacts of a software system. The UML specifications can be divided into the following parts:

- The UML semantics – a meta-model that specifies the abstract syntax and semantics of the UML object modeling concepts.

- The UML notation – a graphic notation for the visual representation of the UML semantics.

3.1 The Approach

The architecture of UML is based on four-layered meta-model structure, which consists of user objects, model, meta-model and meta-metamodel [2, 19]. The UML meta-model is a logical model. It emphasizes on declarative semantics and suppresses the implementation details. For the implementation details, the logical model needs to conform to its semantics. The disadvantage of the logical model is that it is lack of imperative semantics required for accurate and efficient implementation [19].
The meta-model of UML can be described in three views as follows:

- **Abstract syntax** – It is a model described in a subset of UML. It is represented in a diagram that helps to show the meta-classes and define the constructs and their relationships.

- **A well-formedness rules** – It is provided by using a formal language and natural language. A set of well-defined rules is used to describe the static semantics of each construct in UML.

- **Semantics** – the semantics is described primarily in natural language and gives a meaning to the constructs used in each diagram. The usage of the notation depends on the part of the model.

UML has presented three major architectural elements to represent concepts in its model [19].

- **Class** – It gives a description of objects that share the same attributes, operations, and relationship.

- **Interface** – It consists of a set of operations. An interface does not have attributes, operations or any association with other objects.

- **Data type** – A value that can be distinguished from classes.
3.2 Techniques in UML

UML provides eight techniques that can be used to model a scenario or problem domain in a proper way. The techniques are Class Diagram, Use Case Diagram, Sequence Diagram, Collaboration Diagram, State Diagram, Activity Diagram, Component Diagram and Deployment Diagram.

3.2.1 Class Diagram

The Class Diagram is used to show the static structure of the model. It is a graph of classifier elements connected by their various static relationships. A class may contain interfaces, packages, relationships, and instances such as objects and links.

3.2.1.1 Semantics

A Class Diagram is a graphic view of the static structural model. Individual class diagrams do not represent divisions in the underlying model. A Class Diagram describes the types of objects in the system and the static relationships which exist between them. There are three main kinds of relationships which are association, generalization and aggregation.

3.2.1.2 Notation

A Class Diagram is a collection of (static) declarative model elements, such as classes, interfaces, and their relationships, connected to form a graph. Class Diagrams may be organized into packages either with their underlying models or as separate packages that build upon the underlying model packages. The description of the model elements are given below:

- **Class** – A set of objects that share common attributes, operations, methods, relationships and semantics. It can also use a set of interfaces to specify collections of operations.
- **Metaclass** – It is a class whose instances are classes.

- **Interface** – It is a specifier for the declaration of a collection of operations offered by an instance. It has no attributes, states, or associations. An interface can be shown as a rectangle with three compartments and the keyword `<<interface>>`. It can also represent as a small circle attached to a line.

- **Package** – It is a group of model elements. A package can be nested in other packages. A package is shown as a large rectangle and a small rectangle attached at the top of the large rectangle.

- **Object** – An object represents an instance of a class. It has an identity and attribute values. The symbol for representing an object is a rectangle in the form of two compartments. The top compartment is the name of the object and second compartment shows the attributes for the object and their values.

- **Qualifier** – It is an attribute or a list of attributes. The value of one or more attributes for the set of objects is used to associate with another object. It is shown as a small rectangle attached to the end of an association. It is part of the association path.

- **Types of relationships:**
  - **Association** – It can be a binary or n-ary association. The binary association is shown by a solid line connecting two class symbols. The n-ary association is an association among three or more classes. A large diamond represents it. The multiplicity and label can be stated on the line.

  - **Aggregation** – A special form of association that specifies a whole-part relationship between the aggregate (whole) or class and its part or instances.
- **Generalization** – A taxonomic relationship between a more specific element and a general element. It is used for classes, packages, use cases and others. Generalization is shown as a solid line with a hollow triangle at the end of the line.

- **Composition** – It is a form of aggregation but with a strong ownership. The part of a composition can include classes and associations which implies that an object connected via some links must come from the same container object.

- **Dependencies** – This type of relationship indicates a semantic between two or more model elements such as classes. Any changes to be made to the target element depend on the source element. Dependency is shown as a dashed line with an arrow pointing to the target element.

Figure 3.1 shows a Class Diagram. The diagram describes the objects and classes about the Course Registration System. Each object is linked by a different type of relationship. The label on the line denotes the name of the relationship. The classes can include their attributes and operations that are appropriate to the system.

![Class Diagram](image)

**Figure 3.1 – The Class Diagram**
3.2.2 Use Case Diagram

A Use Case Diagram shows the relationships among actors and use cases within a system. It can also describe the scenario of a problem domain related to the actors, use cases or activities in the application system. Jacobson mentioned [12,22] that use cases are difficult to understand especially when a user performs “a behaviorally related sequence of transactions in a dialogue with the system”.

3.2.2.1 Semantics

The Use Case Diagrams provide a way to describe the external view of the system and how the elements interact with the real-world. The real-world is represented by an actor which can be a person or a computer system. The importance is related to the way the actors play their roles. A use case represents an interaction of the actor with the system.

3.2.2.2 Notation

A Use Case Diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and the use cases, and generalizations among the use cases. The description of the elements is given below:

- **Actor** – An actor is a role of object or objects outside of a system that interacts directly with the use case. It helps to characterize the role played by an outside object. Each actor defines a set of roles while interacting with the use cases. An actor can be shown as a rectangle with the stereotype `<<actor>>` or the standard stereotype icon of `"stick man"`.

- **Use cases** – A use case is a coherent unit of functionality provided by a system or class. It can be defined as sequences of messages exchanged among the system. The purpose of a use case is to describe the behavior of an entity without revealing its internal structure. A use case is shown as an ellipse containing the name of the use case.
Types of relationships:

- **Communication** – It shows the participation between an actor and a use case. A communication relationship is represented by a solid line.

- **Extends** – It defines that a use case may be extended with some additional behavior defined by other use cases. An extend relationship between use cases is shown by a generalization arrow from the use case providing the extension to the base use case. The arrow is labeled with the stereotype «extends».

- **Uses** – It is used to show the commonalities between use cases. A uses relationship between two use cases is shown by a generalization arrow from the use case (use case A) doing the use to the use case being used (use case B). The arrow is labeled with the stereotype «uses».

![Diagram](image)

**Figure 3.2 – The Use Case Diagram**

Figure 3.2 shows the Use Case Diagram which describes the scenario of the Course Registration System. The actors are Student, Registrar and Professor. All of them play their own roles in order to interact with the system. The three types of relationships are described in the diagram, which
are **communication** (between an actor and a use case), as well as **extends** and **uses** between use cases and other use cases.

### 3.2.3 Interaction Diagram

An Interaction Diagram is used to describe a pattern of interactions among objects. It has two forms where each emphasizes on a particular aspect. Interaction Diagrams are divided into Sequence Diagrams and Collaboration Diagrams.

#### 3.2.3.1 Sequence Diagram

This diagram shows an interaction between objects in time sequence. Each object has its own lifeline and the messages flow from one lifeline to another in time sequence. The Sequence Diagram is suitable for real-time specification and for a complex scenario as it shows explicit messages.

##### 3.2.3.1.1 Semantics

A Sequence Diagram shows an interaction between objects. This can be done via message-passings among them.

##### 3.2.3.1.2 Notation

A Sequence Diagram has two dimensions. The vertical dimension represents time and the horizontal dimension represents the different objects. The description of the elements used in the diagram is given below:

- **Object** – It represents the entity involved in the system. It is shown as a rectangle.
Object lifeline – It is a role for an object that describes the type of object that may play the role and describes its relationships to other roles. It also represents the existence of the object at a particular time. If the object is created at a particular time then a lifeline will start at the point. If the object is destroyed, a large ‘X’ will be attached at the end of the lifeline. It is shown as a vertical dashed line.

Activation – It shows the period which an object is performing an action either direct or through a subordinate. An activation represents the duration of the action and controls the relationship between the activation. It is shown by a tall thin rectangle.

Message – It is a communication between objects that occurs when an action is issued. A message is shown as a horizontal solid arrow. It is also labeled by the message name.

Types of relationships:

- Transition – A relationship between two states which indicates the first object will perform an action and invoke the second object when a specific event or condition is satisfied. It is shown as a line with an arrow attached at the end of it. Each transition has a label defined from the messages.

- Interaction – A relationship between the object and lifelines. It is shown by a dashed line.
Figure 3.3 – The Sequence Diagram

Figure 3.3 shows the interactions between objects. The transaction of messages shows the flow of the system. The rectangle in red is the lifeline of the object. The termination of a transaction is represented by a termination sign. It is marked with a cross symbol. The rectangle at the top of the lifeline is the object involved in the Course Registration System. Each lifeline is linked with transition relationship and it is labeled with the message that needs to pass to other object.

3.2.3.2 Collaboration Diagram

A Collaboration Diagram is used to show an interaction organized around the objects and how they link to each other. Unlike a Sequence Diagram, a Collaboration Diagram shows the relationships among the object. A Collaboration Diagram does not show time as a separate dimension, so the sequence of messages and the concurrent threads must be determined using sequence numbers.
3.2.3.2.1 Semantics

A Collaboration Diagram is a set of participants and relationships that are meaningful for a given set of purposes. The purposes can be accomplished by a set of objects that exchange messages within an overall interaction.

3.2.3.2 Notation

The behavior in a Collaboration Diagram involves two aspects: the structural description of its participants and the behavioral description of its execution. The structure of objects playing some roles in a behavior and their relationships is called a collaboration. A collaboration shows the context in which an interaction occurs. The interaction is shown by the message sequences exchanged among objects to accomplish a specific purpose. For instance, in the Course Registration System, to register the courses, the student needs to log into the system and then view the course schedule. Below is a description of model elements used in the diagram.

- **Class role or Collaboration role** - It is a slot for an object to describe the type of object playing the role and its relationships with other objects. Collaboration role is shown by a rectangle. The name compartment is specified with: `classRoleName: Classifiiername`

- **Actor** - An actor is a role of object or objects outside of a system that interacts directly with the use case. It helps to characterize the role played by an outside object. Each actor defines a set of roles while interacting with the use cases. An actor can be shown as a rectangle with the stereotype `<<actor>>` or the standard stereotype icon of "stick man".

- **Multiobject** – It represents a set of objects on the *many* end of an association, indicating the operation involving the entire set of objects. A multiobject is shown as two rectangles with the top one is slightly shifted.

- **Message flow** – It represents a message sending from one object to another. There are a few types of message flows described as follows:
- **Procedure call** – It is a nested sequence that sends the message which must be completed before the outer sequence responses to the message. It is shown as a full arrow (→).

- **Control flow** – It shows the progression of the next step of sequence. The messages are usually in asynchronous. It is shown as an arrow (→).

- **Asynchronous flow** – It is used to show that an asynchronous message flows between two objects. It is shown as a half stick arrow (←).

- **Association** – The binary association is shown by a solid line (----) connecting two class symbols. It can have a label stated on the line.

- **Composition** – It is a form of aggregation but with a strong ownership. It can include classes and associations which implies each object connected via some links and must come from the same container object. It is shown as a line ending with a solid diamond (→

![Diagram](image)

**Figure 3.4 – The Collaboration Diagram**

50
Figure 3.4 describes the sequence of messages that flow in the Sequence Diagram. The rectangle represents the *classrole* and the *actor* represents the external elements. In this case, the actor is a student. This diagram shows a *procedure call relationship*, which is one of the transition types.

### 3.2.4 State Diagram

A State Diagram shows the sequences of states that an object or an interaction goes through to respond and to receive stimuli. The behavior can be specified as a traversal graph of state interconnected by one or more transition arcs. The transitions will trigger to create an event.

#### 3.2.4.1 Semantics

A state machine is a graph of states and transitions that describes the response of an object of a given class to the receipt of outside stimuli. A state machine is attached to a class or a method.

#### 3.2.4.2 Notation

In a State Diagram, the transition will start from an initial state which is shown as a small solid black circle. This transition will be labeled with the event that creates the object. A state shows the condition of the object and how it responds to the stimuli. The completion of an activity will be enclosed with the final state which is shown as a circle surrounding a small black circle. Below is the description of the model elements used in the State Diagram.

- *State* – It is a condition during the life of an object or an interaction that will satisfy some condition or perform an action. A state may response to an incoming activity. It is shown as a rectangle with rounded corners.

- *Initial state* – It is shown as a small filled circle. An initial state can have a transition which starts the process.
- **Final state** – It is shown as a circle surrounding a small solid filled circle. A final state represents the end of transition.

- **Synchronization bar** – It allows for multiple transition to enter it. A bar can have multiple incoming arrows or a multiple outgoing arrows. The transition is mapped into a Join connected to a Fork pseudo-state by a single transition with no attachment. This is suitable for a complex transition.

- **Type of Relationship:**
  - **Transition** – A relationship between two states that indicates an object A will enter the second state and perform the action when a specific event occurs and when a specific condition is satisfied.

![State Diagram](image)

**Figure 3.5 – The State Diagram**

The State Diagram in Figure 3.5 is used to describe the behavior of a single object. Each state provides an action or performs an action when a certain event happens. A state is represented by a rounded rectangle with two compartments describing the state name and the action involved.
The black circle is the *initial state* and the circle with a black circle inside (Bull eye) is a *final state*. The blue bar is a *synchronization bar* that is used to describe a concurrent transition.

### 1.2.5 Activity Diagram

An Activity Diagram is used to document the logic of a single operation or method or the flow of the operation.

#### 1.2.5.1 Semantics

An Activity Diagram is a variation of a statechart diagram. The states represent the activities, which will perform the operations. A transition is triggered by the completion of the operation in the source state.

#### 1.2.5.1 Notation

The Activity Diagram focuses on activities, and in this sense, it is like the data flow diagram. It can describe parallel activities and their synchronization by using the *synchronization bar*. Below is the description of the model elements used in the Activity Diagram.

- **Activity** – An activity is shown as a rectangle with convex arcs on the sides. The action expression is stated in the symbol.

- **Initial state** – It is shown as a small filled circle. An initial state can have a transition which starts the process.

- **Final state** – It is shown as a circle surrounding a small solid filled circle. A final state represents the end of transition.

- **Synchronization bar** – It allows for multiple transition to enter it. A bar can have multiple incoming arrows or multiple outgoing arrows. The transition is mapped into a Join
connected to a Fork pseudo-state by a single transition with no attachment. This is suitable for a complex transition.

*Decision* – It is used to indicate different possible transitions. A decision is shown as a diamond shape. It allows one or more incoming transitions or one or more outgoing transitions.

*Type of Relationship:*

- *Transition* – A relationship between two states that indicates an object A will enter the second state and perform the action when a specific event occurs and when a specific condition is satisfied.

![Activity Diagram](image-url)

*Figure 3.6 – The Activity Diagram*
Figure 3.6 shows the flow of activities in an Activity Diagram. It is quite similar with a flow chart. The activity is represented by a rounded rectangle. The black circle is the initial state and the bull eye is the final state. The diamond shape is a decision indicating different transition and it depends on a Boolean condition.

3.2.6 Implementation Diagram

The Implementation Diagram is used to describe the implementation of the system. It includes the source code structure and the structure of the run-time system. The Implementation Diagram comes in two forms, a Component Diagram and a Deployment Diagram.

3.2.6.1 Component Diagram

A Component Diagram is used to describe the software components, the interfaces and relationships that build up the system.

3.2.6.1.1 Semantics

This diagram is used to show the dependencies among the software components, such as source code, binary code and executable code.

3.2.6.1.2 Notation

A Component Diagram shows the dependency relationships between the components or the interfaces. Below is a description of the model elements used in the diagram.

- Component – A component can be a large item such as a subsystem, executable file, etc. It is shown as a large rectangle with two small rectangles at the left side.
Interface – A declaration of operations that may be used to define a service to be offered by an instance.

Type of Relationship:

- Dependency – It is shown as a dashed line with an arrow. It is used to link between components and between interfaces and components.

![Figure 3.7 – The Component Diagram](image)

Figure 3.7 shows the components of the system in a Component Diagram such as the source code, the module and the executable component. A *component* is represented by a rectangle with two small rectangles on the left vertical edge of the rectangle. The *interface* is represented by a lollipop. The relationship used in this diagram is the *dependency relationship*, which links between the interface and a component or between components.

3.2.6.2 Deployment Diagram

A Deployment Diagram shows the dependency of components related to the configuration of a run-time processing component and the software that runs on them.
3.2.6.2.1 Semantics

A Deployment Diagram is related to the Component Diagram but it describes the run-time processing components, processes and the objects related to the system in a more detailed way.

3.2.6.2.2 Notation

It is a graph of nodes that can include other elements such as objects, components or interfaces. Below is a description of the model elements that are used in the diagram.

- **Node** – It is shown as a cube with 3-dimension. A node represents a processing resource. A node type has a type name: `name:node-type`.

- **Component** - A component can be a large item such as a subsystem, executable file, etc. It is shown as a large rectangle with two small rectangles on the left vertical edge of the rectangle.

- **Interface** - A declaration of operations that may be used to define a service to be offered by an instance.

- **Types of Relationships:**
  - **Dependency** – It is shown as a dashed line with an arrow. It is used to link between components and between an interface and a component.
  - **Instance** – An invisible relationship used to show the instance of the node such as component, object and interface.
Figure 3.8 – The Deployment Diagram

Figure 3.8 shows a run-time physical object in a Deployment Diagram. The Course Registration system has an architecture of a client-server system. The server and the client are represented by nodes. The nodes can have instances such as objects, components and interfaces. The relationship used in this diagram is the dependency relationship, which is shown as a dashed line with a stick arrow. An instance relationship is an invisible relationship. It is used to insert other elements in the node.

3.3 Summary

All the notation, syntax and semantics of UML must be followed in order to develop a CASE tool. This is important to ensure the consistency and the correctness of the techniques in the MetaEdit environment. All the diagrams described in this chapter are UML techniques that have been developed using the OPRR concepts.