## Experiment No. - 1

**Name-** Bhavesh Kewalramani

**Roll No.-** A-25

**Section-** A

**Semester-** 6th

**Shift-** 1st

**Aim:**

To study Unified Modeling Language (UML) perspectives and notations in StarUML.

**Theory:**

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95 at Rational software, and its further development was carried out through 1996. In 1997, it got adopted as a standard by the Object Management Group.

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95 at Rational software, and its further development was carried out through 1996. In 1997, it got adopted as a standard by the Object Management Group.

What is UML

The UML stands for Unified modeling language, is a standardized general-purpose visual modeling language in the field of Software Engineering. It is used for specifying, visualizing, constructing, and documenting the primary artifacts of the software system. It helps in designing and characterizing, especially those software systems that incorporate the concept of Object orientation. It describes the working of both the software and hardware systems.

The UML was developed in 1994-95 by Grady Booch, Ivar Jacobson, and James Rumbaugh at the Rational Software. In 1997, it got adopted as a standard by the Object Management Group (OMG).

The Object Management Group (OMG) is an association of several companies that controls the open standard UML. The OMG was established to build an open standard that mainly supports the interoperability of object-oriented systems. It is not restricted within the

boundaries, but it can also be utilized for modeling the non-software systems. The OMG is best recognized for the Common Object Request Broker Architecture (CORBA) standards.

Goals of UML

- Since it is a general-purpose modeling language, it can be utilized by all the modelers.
- UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.
- The UML diagrams are made for business users, developers, ordinary people, or anyone who is looking forward to understand the system, such that the system can be software or non-software.
- Thus it can be concluded that the UML is a simple modeling approach that is used to model all the practical systems.

Characteristics of UML

The UML has the following features:

- It is a generalized modeling language.
- It is distinct from other programming languages like C++, Python, etc.
- It is interrelated to object-oriented analysis and design.
- It is used to visualize the workflow of the system.
- It is a pictorial language, used to generate powerful modeling artifacts.

Conceptual Modeling

Before moving ahead with the concept of UML, we should first understand the basics of the conceptual model.

A conceptual model is composed of several interrelated concepts. It makes it easy to understand the objects and how they interact with each other. This is the first step before drawing UML diagrams.

Following are some object-oriented concepts that are needed to begin with UML:

- **Object:** An object is a real world entity. There are many objects present within a single system. It is a fundamental building block of UML.
- **Class:** A class is a software blueprint for objects, which means that it defines the variables and methods common to all the objects of a particular type.

- **Abstraction:** Abstraction is the process of portraying the essential characteristics of an object to the users while hiding the irrelevant information. Basically, it is used to envision the functioning of an object.
- **Inheritance:** Inheritance is the process of deriving a new class from the existing ones.
- **Polymorphism:** It is a mechanism of representing objects having multiple forms used for different purposes.
- **Encapsulation:** It binds the data and the object together as a single unit, enabling tight coupling between them.

**Introduction to StarUML**

An open source modeling software that is used in Unified Modelling Language where system and software modeling is supported with the UML concept to generate code for different languages by providing different types of UML diagram to develop fast and flexible UML platform so that requirements and design is given before starting the project where it is called as a big upfront design approach for the users who require the software designs to improve their project and hence to support their UML design diagrams in the system.

How StarUML works?
Below are the points explain the works of starUML:

- StarUML is an open source software that can be installed directly from the homepage of the website and it is licensed under GNU Public License. On the home page, a new project by approach link is available from where an empty project can be selected. Set as the default approach should be unchecked so that users can create projects of customized versions. On the model explorer, by selecting the untitled model, go to Add or Design model, and Add or Design diagrams. This will help to create the diagrams of the user's taste and modify it based on the functionalities.

- The profile can be set which gives different symbols and conventions to be used. It is important to include Java profile in the project so that java code is generated. If multiple users are working on the same project, it is good to give a description for better understanding. Save the project in a suitable location to access it easily. The class can be created from the toolbox and proper name can be given to the same. Attributes can be added to the same. The data type should be selected and the design model can be expanded to see the full view of the project created. An interface is created by selecting the interface and suppressing the operations option. This will make the user to see the interface with the diagrams in view.
- UML class diagrams can be created easily with the help of StarUML. While creating the diagram, java stub code is easily generated in the diagrams and hence if any changes has to be made in the diagram, it can be done by modifying the diagram in the backend. While creating the diagram, code is generated but not specifically on the class structures. To modify the structures, the code can be edited and the functionalities on the structure can be added. Thus, the code describes what each structure does in the diagram.
- There are many options available to modify the diagram drawn. It is easy to draw it with the options given and once familiar, anyone can draw it easily.

Benefits of StarUML

Below are the benefits of staruml in detail:

- An advantage that is most important is that the user can generate codes from the diagram drawn. If anyone who is not interested in front end diagram, they can use the backend coding to add the functionality and change the diagram to their need and modify it as per the usage by changing the Java code. This makes reverse engineering

possible i.e. generating a diagram from the code that is formed initially from the diagram drawn.

- The user interface is known to all as it uses visual studio along with other coding languages such as C and C#. Also, the diagrams drawn in UML software can be exported into JPG.XMI formats which helps the user to identify and check the patterns in different format and to explore more options.
- StarUML is faster, flexible and can be extended to accommodate other codes in the diagram. And the extensive features make the users to fall in love with the application. If any mistakes happen, we can undo and make necessary changes. This feature is not applicable in some other UML tools.
- The framework can be easily understood by anyone and hence the architecture can be modified for the extensive use of software. Performance and security can be tracked easily with the help of StarUML. Documentation is provided with proper guidelines as how to improve and communicate the business processes to others who use the software. We can say that StarUML is the visual language that communicates the information to the users in a diagrammatic manner.
- The tools in StarUML helps to know the requirements in the system and to apply the design patterns so that proper analysis can be done to understand and modify the diagrams. These tools are open source and for more highly requirements, tools can be purchased from the software vendors.

Applications of StarUML in various fields

- Unified modeling diagrams and class diagrams can be created easily in StarUML that can be used in design industry to know the flow of the product from one field to another and also to analyze the product's usage in different fields. Properties of the product can be incorporated within the diagram so that the user can easily understand

the product and modify the properties if needed. Data modeling helps to know the product and apply the properties to other streams.

- In the manufacturing industry, the number of products with its varieties can be known by the user. With the knowledge of inventory in the industry, product manufacturing can be controlled and directed efficiently. Diagrams direct the user about the steps to be done and manage the goods to shipment and do the transaction from the customers. These understandings can be done with the help of a single diagram rather than using an entire record.
- In the hospitality sector and hospitals, the UML diagram can be used to direct the visitors to proper places. The doctors in hospitals can be made to go through the diagram once to know the routine timetable they should follow on a daily basis. In hotels, available provisions can be drawn with the help of a UML diagram so that the visitors will get to know the privileges in a glance.

Diagrams help to save time and to understand the process flow better. StarUML helps to know the class diagrams and to know the process so that the users can change or get the knowledge of the same.

Elements of the **UML** (**Unified Modelling Language**) notation are provided as this is necessary for the comprehension of the diagrams presented. The UML notation is a notation conceived for modeling object of applications and continue and extend, in particular, the notations of OMT (Object Modeling Technique) and Booch methods. More precisely, here we describe the principles of the use-case diagrams, classes, objects and sequence diagrams.

A1.Use-case diagram

Use-case allows us to model and structure the interactions between the users of a system, called **actors**, and the system itself. The use-cases represent a means of analysis of the users' needs and enable us to relate the actions made by a user with the reactions considering of a system. More precisely, a use-case is an abstraction of a set of concrete scenarios carried out by a type of users (Figure 1).

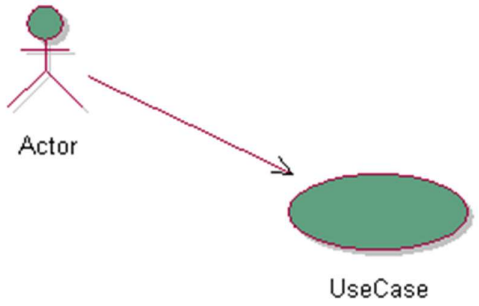


*Figure 1: use-case*

Use-cases can be structured and connected by two principal types of relations: relations of use (**uses**) and relations of extension (**extends**). A **uses** relation allows us to break up a use-case into use sub-cases. A relation of **extension** indicates a specialisation. In Figure 2, the interaction defined by the use-case A includes that of the use-case B, and the use-case C is a particular case of the use-case B.

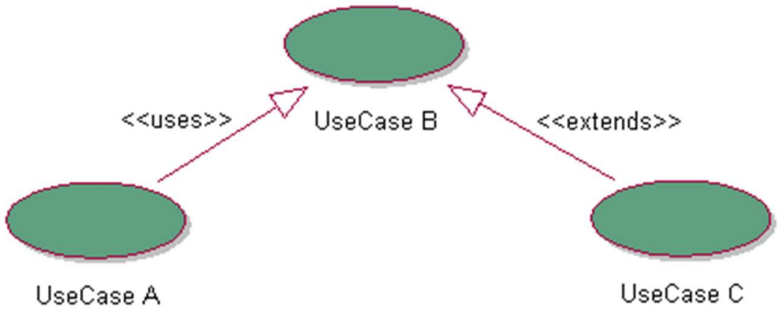


*Figure 2: types of relations between use-cases*

For example (Figure 3), if one wishes to model the operations of credit transfers, they could identify two types of actors: distant customers and local customers. The two types of customers carry out a transfer, but the distant customers use a Minitel terminal, which generates a specific interaction (relation of **extension**). In all the cases, the operation of transfer includes a phase of identification (relation of **use**).

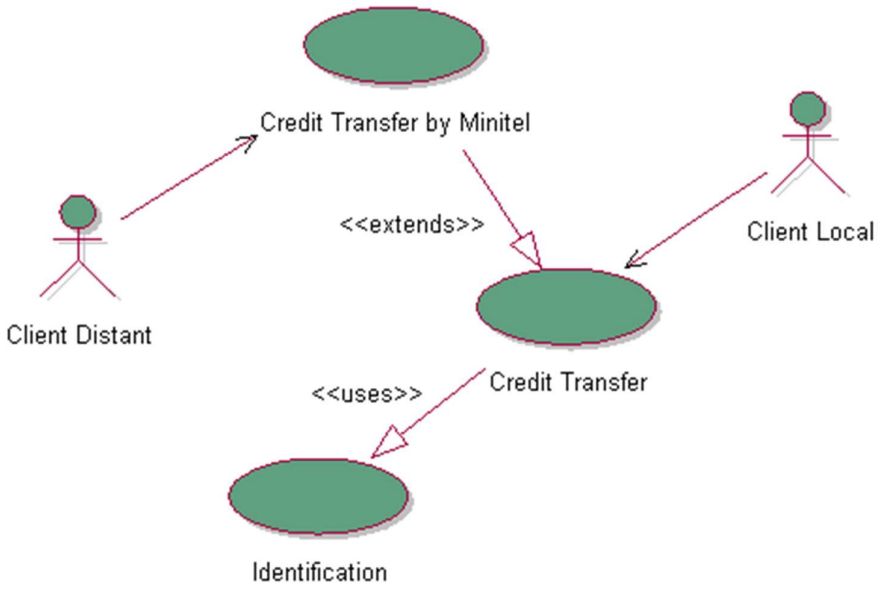


*Figure 3: example of use-case (Muller, 1997, page 127)*

Within the **Object-Oriented Framework** of CBR*Tools, we used the use-cases to express the interactions between a user and the system formed by the framework itself completed of its environment of handling. A use-case can then be connected to hotspots, which take part in the realisation of the use-case (Figure 4).

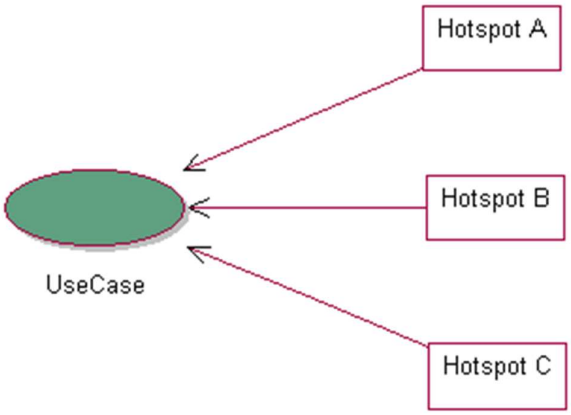


*Figure 4: hotspots and use-cases*

The use-cases also enable us to specify the type of the configuration of a hotspot in the realisation of an application made with CBR*cTools, while keeping the structure of the initial analysis (Figure 5). Three types of configuration are identified: specialisation (hotspot transparent box), instantiation (hotspot black box), or use of the behaviour by defect. If the same hotspot is used several times with various types of configuration, only one retains the most specific according to this command. One can then use the stereotypes of UML to indicate the type of the configuration on the relations between the hotspots and the use-cases.

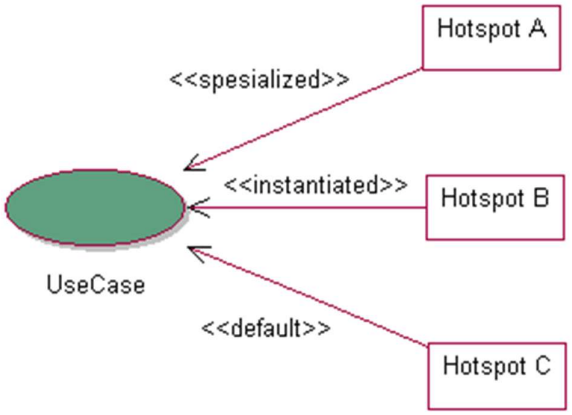


*Figure 5: use of hotspots*

A2.Class diagram

A class of objects is represented by a rectangle, which consists of three parts (Figure 6):

1. **name** of the class,
2. **attributes** and
3. **operations (or methods)**.

The lists of the attributes and the operations are however optional according to the degree of detail that we want to include in a diagram: these parts can be missed out empty or even to be totally absent. The attributes and the operations have a visibility (in particular public or protected), which is indicated by a symbol preceding their name: if the shape of a key is drawn, the access is called to be **protected**, because it is restricted to the current class and its subclasses.

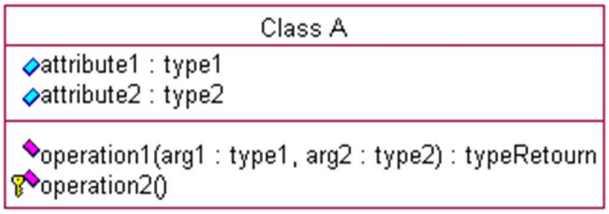


*Figure 6: attributes and operations of a class*

Three types of classes are used:

1. **concrete classes**,
2. **abstract classes** and
3. **interfaces**.

Contrary to a concrete class, an abstract class, whose definition is in italic (Figure 7), cannot be instantiated. An interface is a class defining only operations and does not contain any code. An interface is identified by the indication of the stereotype **Interface** followed by its name.

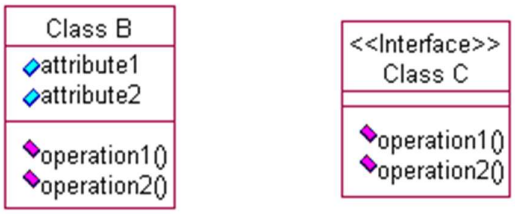


*Figure 7: abstract class and interface*

An association represents a structural relation between various classes. An association is generally bidirectional, but we specify each time the direction of navigation. Association is then known as one-way (Figure 8) and also indicates the direction for reading the association. Each class plays a role in the association, which carries an indication of multiplicity (1 for one and only one object, and a star for zero to many objects). For example, Figure 8 indicates that an object of class C1 is associated with a set of objects of class C2, knowing that a same object of class C2 is associated with a single object of class C1.

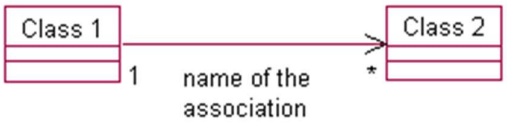


*Figure 8: one-way binary association*

Various types of associations are also defined (Figure 9):

1. **qualified associations**,
2. **derived associations**,
3. **associations with class** and
4. **aggregations**.

A qualified association is an association which allows us to restrict the objects referred in an association thanks to a key. A derived association is an association made starting from another association. An association with class is an association itself managed by a class, which can thus provide attributes or operations specific for association. Lastly, a aggregation is a strongly asymmetrical association.

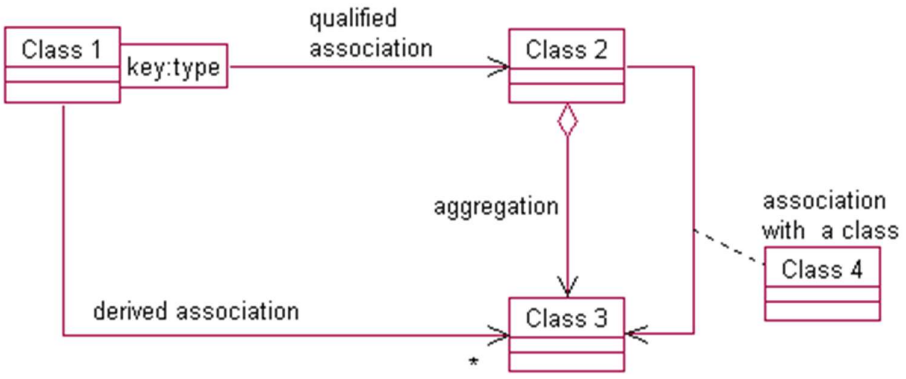

*Figure 9: type of associations*

Constraints can be posed on associations (Figure 10): constraint on the role of an association (for example, the objects are ordered), or between associations (for example, an association is a subset of another).

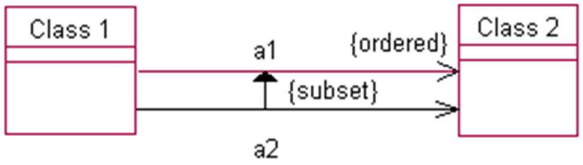


*Figure 10: constraints and relations*

Lastly, the classes can be connected by relation of simple or multiple inheritance (Figure 11).

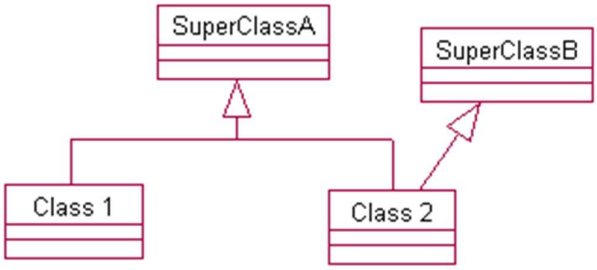


*Figure 11: relation of inheritance*

A3.Object diagram

An object diagram allows us to concretely show a connection between objects. An object (or authority) is represented by a rectangle which generally comprises the name of the object and its class (Figure 12). However, the class or the name of the object can be omitted.


*Figure 12: representations of an object*

Figure 13 gives an example of a diagram of objects showing the relations of aggregation between a car, its wheels and its engine.

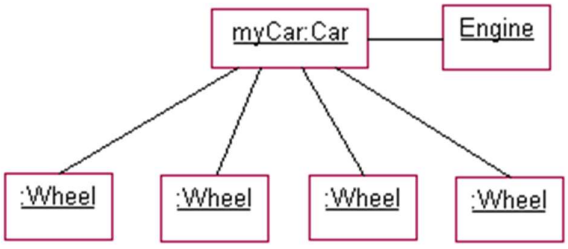

*Figure 13: diagram of objects (P.A. Muller. Modeling object with UML. Eyrolles, 421 pages, 1997, page 136)*

A4.Sequence/Interaction Diagram

A **sequence diagram** chronologically shows (from top to bottom) the interactions between a set of objects (Figure 14). Each object has a life line (vertical line). On these life lines, the periods of activity are indicated by fine rectangles which are superposed in the event of recursive call.

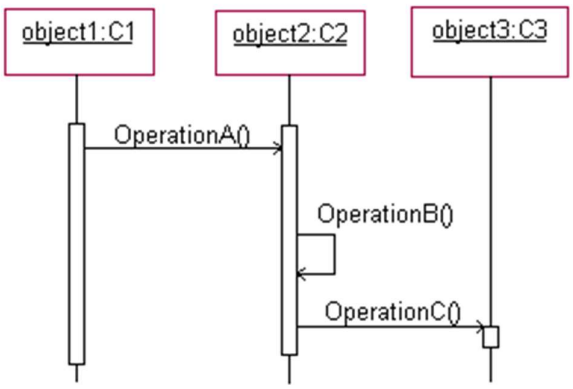


*Figure 14: sequence diagram*

**Conclusion:**

- In this Experiment, we learnt about UML diagrams, how they are created and which tool is used to create them. We also learnt about different UML diagrams available and when they are used specifically. he UML is non- proprietary and open to all. It addresses the needs of the user and scientific communities, as established by experience with the underlying methods on which it is based.
- Many methodologists, organizations, and tool vendors have committed to using it. Since the UML builds upon similar semantics and notation from Booch, OMT, OOSE, and other leading methods and has incorporated input from the UML partners and feedback from the general public, widespread adoption of the UML should be straightforward.

There are two aspects of “unified” that the UML achieves:

- First, it effectively ends many of the differences, often inconsequential, between the modeling languages of previous methods.
- Secondly, and perhaps more importantly, it unifies the perspectives among many different kinds of systems (business versus software), development phases (requirements analysis, design, and implementation), and internal concepts.