UML Overview
By Mandar Chitnis, Pravin Tiwari, & Lakshmi Ananthamurthy

Introduction

Modeling is an activity that has been carried out over the years in software development. When writing applications by using the simplest languages to the most powerful and complex languages, you still need to model. Modeling can be as straightforward as drawing a flowchart listing the steps carried out by an application. Why do we use modeling? Defining a model makes it easier to break up a complex application or a huge system into simple, discrete pieces that can be individually studied. We can focus more easily on the smaller parts of a system and then understand the "big picture." Hence, the reasons behind modeling can be summed up in two words:

- Readability
- Reusability

Readability brings clarity—ease of understanding. Understanding a system is the first step in either building or enhancing a system. This involves knowing what a system is made up of, how it behaves, and so forth. Modeling a system ensures that it becomes readable and, most importantly, easy to document. Depicting a system to make it readable involves capturing the structure of a system and the behavior of the system.

Reusability is the byproduct of making a system readable. After a system has been modeled to make it easy to understand, we tend to identify similarities or redundancy, be they in terms of functionality, features, or structure.

Even though there are many techniques and tools for modeling, in this article series, we will be concerning ourselves with modeling object-oriented systems and applications using the Unified Modeling Language. The Unified Modeling Language, or UML, as it is popularly known by its TLA (three-letter acronym!), is the language that can be used to model systems and make them readable. This essentially means that UML provides the ability to capture the characteristics of a system by using notations. UML provides a wide array of simple, easy to understand notations for documenting systems based on the object-oriented design principles. These notations are called the nine diagrams of UML.

So the question arises, Why is UML the preferred option that should be used for modeling? Well, the answer lies in one word: "standardization!" Different languages have been used for depicting systems using object-oriented methodology. The prominent among these were the Rumbaugh methodology, the Booch methodology, and the Jacobson methodology. The problem was that, although each methodology had its advantages, they were essentially disparate. Hence, if you had to work on different projects that used any of these methodologies, you had to be well versed with each of these methodologies. A very tall order indeed! The Unified Modeling Language is just that. It "unifies" the design principles of each of these methodologies into a single, standard, language that can be easily applied across the board for all object-oriented systems. But, unlike the different methodologies that tended more to the design and detailed design of systems, UML spans the realm of
requirements, analysis, and design and, uniquely, implementation as well. The beauty of UML lies in the fact that any of the nine diagrams of UML can be used on an incremental basis as the need arises. For example, if you need to model requirements for a given system, you can use the use case diagrams only without using the other diagrams in UML. Considering all these reasons, it is no wonder that UML is considered "the" language of choice.

UML does not have any dependencies with respect to any technologies or languages. This implies that you can use UML to model applications and systems based on either of the current hot technologies; for example, J2EE and .NET. Every effort has been made to keep UML as a clear and concise modeling language without being tied down to any technologies.

This series aims to cover the basics of UML, including each of the nine diagrams of UML. In addition, you will get to learn about the tools available that support UML. At the end of each article, we will incrementally build each of the nine UML diagrams for a case study system in the coming weeks. We will wrap our study of UML by expanding into two different areas—Rational Unified Process and Design Patterns.

**UML Diagrams**

The underlying premise of UML is that no one diagram can capture the different elements of a system in its entirety. Hence, UML is made up of nine diagrams that can be used to model a system at different points of time in the software life cycle of a system. The nine UML diagrams are:

- **Use case diagram**: The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case.

- **Class diagram**: The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

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

- **State diagram**: A state diagram, as the name suggests, represents the different states that objects in the system undergo during their life cycle. Objects in the system change states in response to events. In addition to this, a state diagram also captures the transition of the object's state from an initial state to a final state in response to events affecting the system.

- **Activity diagram**: The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.
• **Sequence diagram**: A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

• **Collaboration diagram**: A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

• **Component diagram**: The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.

• **Deployment diagram**: The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

Now that we have an idea of the different UML diagrams, let us see if we can somehow group together these diagrams to enable us to further understand how to use them.

**UML Diagram Classification—Static, Dynamic, and Implementation**

A software system can be said to have two distinct characteristics: a structural, "static" part and a behavioral, "dynamic" part. In addition to these two characteristics, an additional characteristic that a software system possesses is related to implementation. Before we categorize UML diagrams into each of these three characteristics, let us take a quick look at exactly what these characteristics are.

• **Static**: The static characteristic of a system is essentially the structural aspect of the system. The static characteristics define what parts the system is made up of.

• **Dynamic**: The behavioral features of a system; for example, the ways a system behaves in response to certain events or actions are the dynamic characteristics of a system.

• **Implementation**: The implementation characteristic of a system is an entirely new feature that describes the different elements required for deploying a system.

The UML diagrams that fall under each of these categories are:

- **Static**
  - Use case diagram
  - Class diagram

- **Dynamic**
  - Object diagram
  - State diagram
Finally, let us take a look at the 4+1 view of UML diagrams.

**4+1 View of UML Diagrams**

Considering that the UML diagrams can be used in different stages in the life cycle of a system, let us take a look at the "4+1 view" of UML diagrams. The 4+1 view offers a different perspective to classify and apply UML diagrams. The 4+1 view is essentially how a system can be viewed from a software life cycle perspective. Each of these views represents how a system can be modeled. This will enable us to understand where exactly the UML diagrams fit in and their applicability.

These different views are:

- **Design View**: The *design view* of a system is the structural view of the system. This gives an idea of what a given system is made up of. Class diagrams and object diagrams form the design view of the system.
- **Process View**: The dynamic behavior of a system can be seen using the *process view*. The different diagrams such as the state diagram, activity diagram, sequence diagram, and collaboration diagram are used in this view.
- **Component View**: Next, you have the *component view* that shows the grouped modules of a given system modeled using the component diagram.
- **Deployment View**: The deployment diagram of UML is used to identify the deployment modules for a given system. This is the *deployment view* of the system.
- **Use case View**: Finally, we have the *use case view*. Use case diagrams of UML are used to view a system from this perspective as a set of discrete activities or transactions.

**Summary**

In the first article of this series, we took a quick background at what UML is and where it fits in the overall software life cycle. Each of the nine diagrams that make up UML will be covered step by step in the coming weeks. Before we start with our study of each of these nine diagrams, we will take a look at what UML tools are available in the market in the next article.