An Overview of Object-Oriented Modeling using UML Methodology

A. OO Modeling -- a Short Overview

Object-Oriented (OO) Modeling is a fairly new methodology used to develop software systems and applications. It started from the "bottom-up," i.e., as OO programming. Later on, it moved "up and down" on the systems/software development life-cycle. Namely, we now have OO analysis and design, and even OO testing.

Generally speaking, modeling is an activity of capturing a system components and behavior in an abstract form, using a given set of symbols, that when put together become a specific type of a diagram. A methodology comprises, besides a specific modeling technique, a process, a set of other methods and techniques, and of course, specific tools to implement it.

OO is an evolutionary step in the history of software and systems development, that also recently has appended the word "engineering" to indicate that such activities are precise, with considerable theoretical and practical, or proven support.

The OO approach is based on a natural process call *abstractization* or *generalization*. Abstract entities are represented by the concept of *classes*. Classes contain pertaining *attributes* or *properties* and *actions* or *behavior* that use the defined attributes, *encapsulated* like within a *package*. This encapsulation mechanism hides and protects data and behavior from the external world. Such combination can uniquely identify and completely represent any abstract entity. Within any domain context, many classes can exist. Therefore, *relationships* between such classes are being established. *Inheritance* is one of the most important such relationships that results into avoiding duplication -- since some entities can be derived from more general, existing ones, directly or even through cross combinations. When a class, an abstract concept, *instantiate* into something "real," it becomes an *object*. During their existence, objects interact with one each other, mostly based on the relationships between their respective classes, or extending those relationships to specific circumstances. That is that the interesting part about objects is that they can behave depending on the context, also known as *polymorphism*.

Since modeling needs to capture both the *structural* and *behavioral* (sometimes, time-based) aspects of a system, there are various OO types of diagrams to represent that aspect.

One of the highest expectations about OO approach to systems and software development is *reusability*, or avoidance of duplication of effort, time and cost. A new system can be fairly easy developed from an existing, more abstract, generic one. On the other hand, one of the main disadvantages of the OO approach is its traditionally stiff learning and development curve. However, this disadvantage is fading away, as OO becomes increasingly embraced and as better tools become more readily available and used.

B. UML Methodology -- Main Concepts and Symbols

The UML (Unified Modeling Language) has its own long evolutionary history. It is called "unified" because it is rather an attempt to unify many exiting OO methodologies, with their specific use of symbols and peculiarities, to implement the common but also the variations of OO modeling.

UML is still evolving, and lately it expands its capabilities beyond the software development, i.e., to the entire systems approach.

Unfortunately, UML is a *language* by itself that requires time and practice to get familiar with and later on, to be proficient in using it. Starting from the top, the UML uses the concept of *Actor*, or participant within any activity *domain*. Actors interact with systems components (processes or data) or with one each through those

components. This will result into what UML calls a *Use Case*. With other words, a Use Case is rather a view of a system or part of a system as being used by one of its users. Therefore, a Use Case is not a comprehensive view of the entire system. It rather has its own levels of granularity or perspectives. It is also common to group various related Use Cases into a larger component, called a *Package*. Usually, each Use Case can have one or more *Scenarios*, or uses. Like an ATM machine, as a Use Case of a Banking Application, that can be utilized in various ways (e.g., deposit, withdraw, check balances), or Scenarios.

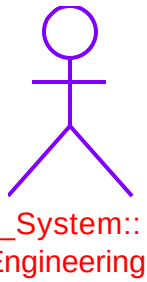
Therefore, the main purpose of a Use Case, as a UML modeling technique is to capture in an abstract, generic way, both a system's components and behavior. That is also known as the *requirements* or the *specifications* of that system's functionality.

C. UML and CASE tools; Basic Diagrams

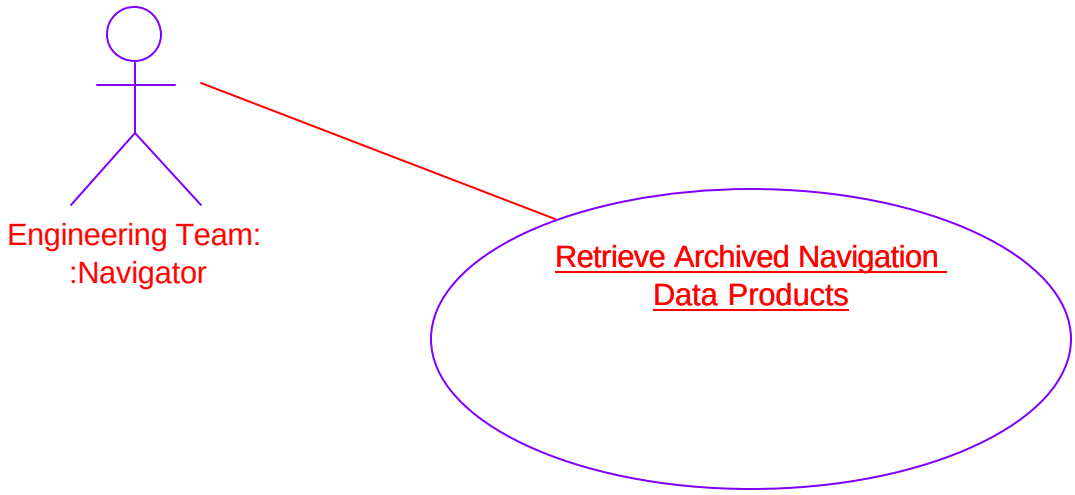
Modern OO development is using tools such as CASE (Computer Aided Software/Systems Engineering). The key word to emphasize here is "aided" as opposed to "automated." In a real world development environment, the above mentioned OO concepts and their implementation via a formal specification language such UML, is facilitated by such a package of (integrated, phase-based) tools, called CASE. There already are many CASE tools that supports UML.
The following URL http://newton.uor.edu/FacultyFolder/CKettemborough/SAD_PM_Resources.html contains links to many such tools. For instance, Iconix's Rhapsody is one of many available. Like any tool, Rhapsody has its own features, capabilities and limitations. The following are some examples of requirements modeling representations as captured with Rhapsody CASE tool.

According to UML the central mechanism a system is viewed or modeled is the Use Case. As earlier mentioned, Use Cases are usually combined into Packages. A key participant in a Use Case is an Actor. The following will shortly describe these components as well as the symbols used to represent them.

Actor. An Actor is a role a user plays with respect to the system. For instance, the Mission Team is an actor. There will probably be many mission team members in the system, such as Navigator, Scientist, Project Manager. However, as far as the system is concerned, they all play the same role: Mission Team. A user may also play more than one role. For example, a Navigator is a Mission Team member, but it also can be a person that performs a very specific job that only a Navigator can do. The following symbol is used within this document to represent an Actor.



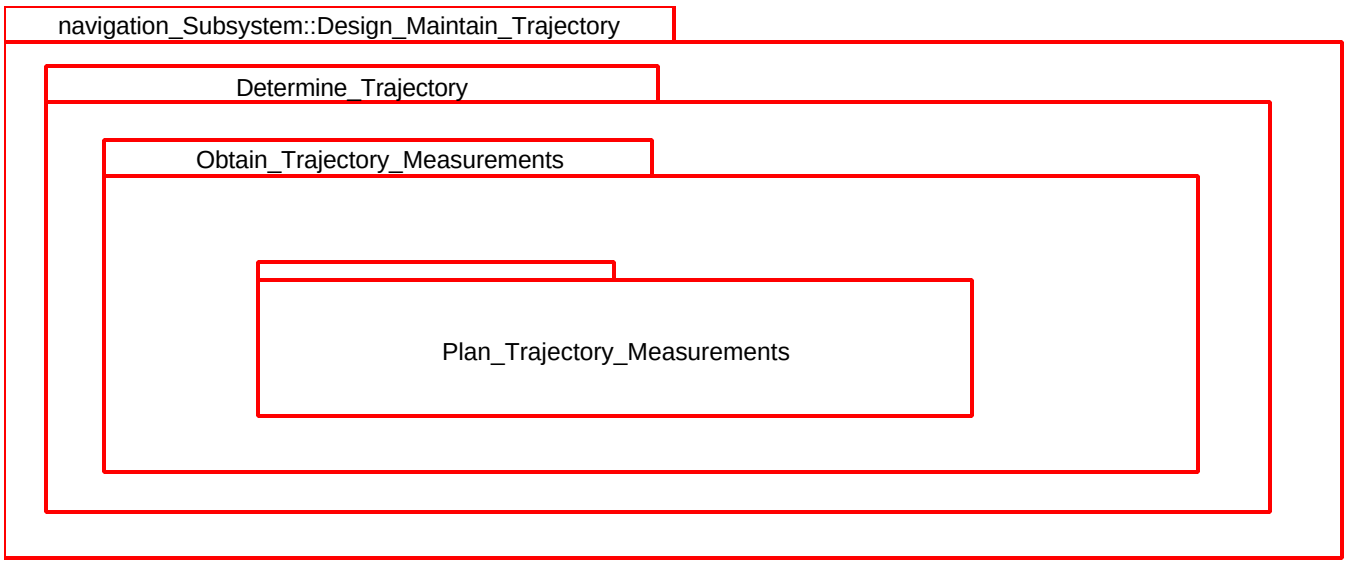
An Actor can carry out many use cases. A single actor may perform in many use cases; conversely, a use case may have several actors performing it. The following is an example of a simple Use Case involving Actors.



In practice, actors are most useful when trying to come up with the use cases. Faced with a big system, it can often be difficult to come up with a list of use cases. In addition, let us note that actors do not need to be human, even though actors are represented as stick figures within a use case diagram. An actor can also be an external system that needs some information from the current system.

Use Cases. Use cases are all about externally-required functionality. If the DSN system needs a Navigation related file, that is a requirement that needs to be satisfied. A Use Case, in addition to a diagram that represents it, can have associated textual specification, i.e., a description of what it does. This later feature it usually helps in understanding and/or capture the system requirement(s). Within a Use Case, as in the above example, besides the system itself, the Actor(s) is/are the other key component(s).

Packages. As above said, one or more use cases are combined into what is called a Package. The following is a simple example of such (package) diagram:



Scenarios. The term scenario is being used in conjunction with use cases. This word is used inconsistently. Sometimes, a scenario is used as a synonym for a use case. Within UML, a scenario refers to a single path through a use case, one that shows a particular combination of conditions within that use case. For example, if a mission team member wants to request a navigation trajectory, we would have a single use case with several

associated scenarios: one in which radio-metric data processing is involved; one in which optical data processing is involved; and so forth. The following is a simple example of a scenario diagram: