Chapter 10 : Object-Oriented Analysis Modelling Using the UML

System Concepts for Object Modelling

* **Object, Attributes, Methods and Encapsulation**

The object-oriented approach to system development is based on the concept of objects that exist within a system’s environment. In object-oriented approaches to system development, the definition of an object is as presented in the margin. The types of objects may include a person, place, thing or event. An employee, customer, instructor and student are examples of person objects. A warehouse, regional office, building and room are examples of place objects. Examples of thing objects include a product, a vehicle or a computer. Examples of event objects include an order, payments, invoice, application and registration.

In object-oriented circles, this part of our definition refers to what are called attributes.

Object-oriented approaches to systems development are concerned with identifying attributes that of interest regarding objects. With advances in technology, attributes have evolved to include more than simple data characteristics. An important object-oriented principle is that an object is solely responsible for carrying out any functions or behaviours that act on its own data or attributes. This leads to an important concept om understanding objects: encapsulation. Applied to an object, both attributes and behaviour of the object are packaged together. They are considered part of that object. The only way to access or change an object’s attributes is through that object’s specific behaviours.

* **Classes, Generalization and Specialization**

An important concept of object modelling is the concept of categorizing objects into object classes. When levels of object classes are identified, the concept of inheritance is applied. The approach that seek to discover and exploit the commonalities between objects classes is referred to as generalization/specialization.

* **Object Class Relationships**

The object class relationship is inevitable, objects interact with other objects within a systems environment. The connecting line represents a relationship between the classes. UML refers to this line as an association. All relationships are bidirectional, meaning that they can be interpreted in both directions.

* **Messages and message sending**

Objects classes interact or “communicate” with one another by passing messages. Recall the concept of encapsulation, wherein an object is a package of attributes and behaviour. Only that object can perform its behaviour and act on its data.

* **Polymorphism**

An important concept that is closely related to messaging is polymorphism. Polymorphism is applied in object-oriented applications when a behaviour in the supertype needs to be overridden by a behaviour in the subtype. The subtype that requires the unique behaviour will contain in its behaviour list the same behaviour that is listed for its parent(supertype). Polymorphism is useful when making enhancements to an existing system, because adding new classes to an existing generalization/specialization relationship in order to satisfy new business rules or requirements may not be possible or practical.

**The UML Diagrams**

|  |  |
| --- | --- |
| Diagram | Description |
| Use case | Depicts the interactions between the system and external systems and users. |
| Activity | Depicts the sequential flow of activitites of a use-case or business process. It can be also be used to model logic with the system. |
| Class | Depicts the system’s object structure. It shows object classes that the system is composed of as well as the relationship between those objects classes. |
| Object | Similar to a class diagram, but instead of depicting object classes, it models actual object instances with current attribute values. |
| State machine | Models how events can change the state of an object over its lifetime, showing both the various states that an object can assume and the transitions between those states |
| Composite structure | Decomposes the internal structure of a class, component or use case. |
| Sequence | Graphically depicts how objects interact with each other via messages in the execution of a use case or operation. |

The Process of Object Modelling

There are three general activities in performing object-oriented analysis:

**1.Modelling the functions of the system**

During this activity the use cases were documented to contain only general information about the business event. The goal was to quickly document all of the business events in order to define and validate requirements.

**2. Constructing the analysis use-case model**

In object-oriented analysis we evolve the requirements use-case model into the analysis use-case model by performing the following steps:

1. Identify, define, and document new actors.
2. Identify, define, and document new use cases.
3. Identify any reuse possibilities.
4. Refine the use-case model diagram
5. Document system analysis use-case narratives.

**The Guideline for Constructing Activity Diagrams**

The following list presents an excellent process for constructing activity diagrams:

* Start with one initial node as a starting point
* Add partition if they are relevant to your analysis.
* Add an action for each major step of the use case
* Add flows from each action to another action, a decision point or an end point.
* Add decision where flows diverge with alternating routes. Be sure to bring them back together with a merge.
* Add forks and join where activities are performed in parallel.
* End with a single notation for activity final.

**Drawing System Sequence Diagrams**

A tool used by some OO methodologies in the logical design phase is the system sequence diagram. A full set of system sequences diagrams might have several diagrams for a single use-case.

1. Actor – the initiating actor of the use case is shown with the use case actor symbol.
2. System – the box indicates the system as a “black box” or as a whole.
3. Lifelines – the dashed vertical line extending downward from the actor and system symbols, which indicate the life of the sequence.
4. Activation bars – the bars that are set over the lifelines indicate the period of time when the participant is active in the interaction.
5. Input messaging – horizontal arrows from the actor to the system indicate the message inputs. The UML convention for messages is to begin the first word with a lowercase letter and append additional words with an initial uppercase letter and no space.
6. Output messaging - horizontal arrows from the system to the actor are shown as dashed lines.
7. Receiver Actor – other actors or external systems that receive messages from the system can be included.
8. Frame – a box can enclose one or more messages to divide off a fragment of the sequence.