Slide 11, 12, 13: Object Modeling Using UML

Made by Md. Mehedi Hasan Rafy

Model

- Captures aspects important for a specific application, omitting irrelevant details.
- Can be graphical, textual, mathematical, or code-based.
- Used for documentation of design/analysis results and to aid those processes.
- Graphical models (like UML) are popular due to ease of understanding.
- Often require accompanying text for explanations.

Need for a model

- Helps manage complexity.
- Uses during development: analysis, specification, code generation, design, visualization, testing.
- Models vary depending on purpose.

Unified Modeling Language (UML)

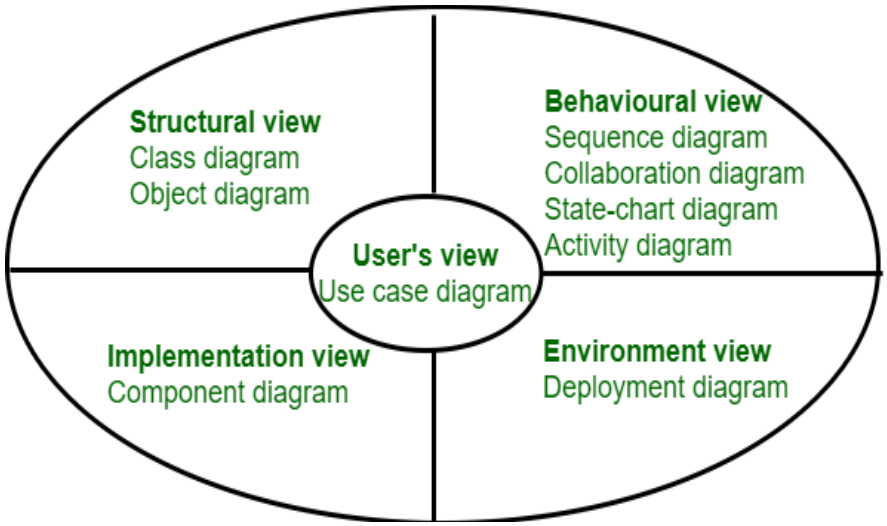
- A modeling language for visualizing, specifying, constructing, and documenting software artifacts of a software system.
- Provides a set of notations (rectangles, lines, ellipses, etc.) to create a visual model.
- Has syntax (rules) and semantics (meanings).
- Not a development methodology, but can document object-oriented analysis or design results from any methodology.

UML Diagrams and Views

UML can be used to construct nine different types of diagrams to capture five different views of a system. Here are the ***five views*** of a system:

1. ***User's View*** – Defines the system's functionalities from the external user's perspective. It is a black-box view that hides internal structure, dynamic behavior, and implementation details. Unlike other object-model views, it is a functional model and serves as the central view to which all other views must conform.
2. ***Structural View*** – Defines key objects or classes, its implementation, and relationships among them. It is also called the static model since system structure does not change over time.
3. ***Behavioral View*** – Shows how objects interact to realize system behavior, focusing on time-dependent (dynamic) aspects of the system.
4. ***Implementation View*** – Depicts important system components and their dependencies.
5. ***Environmental View*** – Maps system components to the hardware on which they are implemented.

UML supports ***nine diagram types*** to represent these views.



Use Case Model

The Use Case Model for any system consists of a set of 'use cases' which represent the different ways in which a system can be used by users.

For example, for the Library Information System (LIS) use cases should be, issue-book, query-book, return-book, create-member, add-book etc.

Use cases

Represent high-level functional requirements, dividing system behavior into transactions that provide useful actions from the user's perspective. A transaction may involve one or multiple message exchanges between user and system. They define coherent behavior without exposing internal structure, algorithms, or data representation, and typically describe a sequence of user–system interactions.

Mainline Sequence

Mainline Sequence is the normal, most occurring interaction flow between user and system. For example, in a bank ATM "withdraw cash" use case — draw, complete transaction, receive amount. Variations (alternative paths) occur under specific conditions, for example, invalid password or insufficient balance.

Another definition for Use Case: A use case is a set of related scenarios or instances (mainline + variations), each representing a single path of user events and system actions toward a common goal.

Representation of Use Cases

Use cases are represented by drawing use case diagram and writing an accompanying text elaborating the diagram.

Diagram*:*

- Each *use case = ellipse with name written inside*.
- *All use cases inside a rectangle (system boundary).*
- System name appears inside rectangle.
- Users = stick person icons (actors).
- ***Actor:*** An actor is a role played by a user with respect to the system use. It is possible that the same user may play the role of multiple actors. Each actor can participate in one or more use cases.
- Actors connected to use case via communication relationship line.
- External systems also shown as actors and is annotated with stereotype `<actor>`.

Text Description:

Each use case (ellipse) should have an accompanying text detailing user–system interactions, covering:

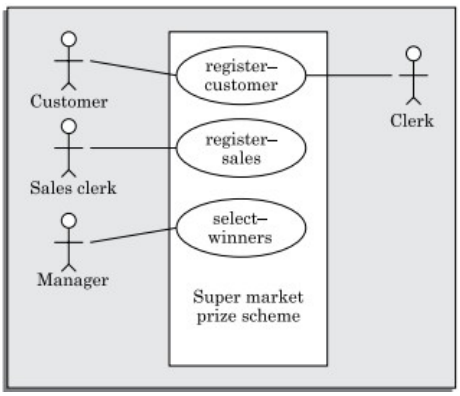
- ***Behavior*** – Mainline sequence, alternative scenarios, system responses, exceptional conditions.
- ***Style*** – Often conversational; may be informal but should have structure.

Here are some of the variations addition to the mainline sequence:

1. ***Contact Persons*** – Client personnel with whom the use case was consulted, date/time of the meeting.
2. ***Actors*** – Identification and relevant details about actors which may aid implementation.
3. ***Precondition*** – Describe the system state before use case execution.
4. ***Post-condition*** – System state after successful completion of use case.
5. ***Non-functional Requirements*** – Constraints like platform, environment, qualitative goals, response time etc.
6. ***Exceptions / Error Situations*** – Only domain-related errors such as invalid input, lack of access rights.
7. ***Sample Dialog*** – Example interactions.
8. ***Specific UI Requirements*** – Contains forms, screenshots, interaction style.
9. ***Document References*** – Relevant domain documents for understanding system operation.

Example: Supermarket Prize Scheme

- Customers register with address, phone, driving license.
- System assigns unique Customer Number (CN).
- Purchases credited to CN.
- End of year: Top 10 customers get surprise gifts and purchases over Tk.100,000 get gold coin.
- Entries reset yearly.



- Three use cases: ***register-customer***, ***register-sales***, ***select-winners***.

Text description:

U1:

register-customer: Using this use case, the customer can register himself by providing the necessary details.

Scenario 1: Mainline sequence

1. Customer: select register customer option.
2. System: display prompt to enter name, address, and telephone number.
3. Customer: enter the necessary values.
4. System: display the generated id and the message that the customer has been successfully registered.

Scenario 2: at step 4 of mainline sequence

System: displays the message that the customer has already registered.

Scenario 3: at step 4 of mainline sequence

System: displays the message that some input information has not been entered. The system displays a prompt to enter the missing value.

The description for other use cases is written in a similar fashion.

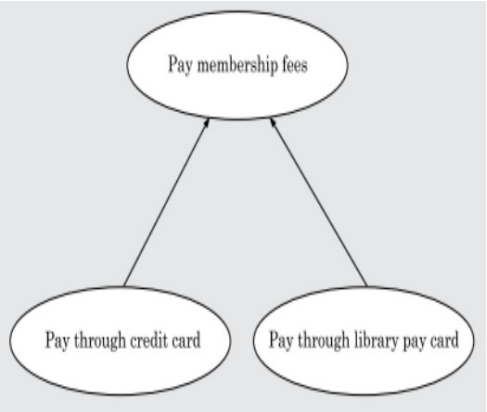
<u>Identifying Use Cases</u>

- ***SRS:*** Typically, the high-level requirements specified in the SRS document correspond to the use cases.
- ***Actor-based approach:*** In absence of a well informed SRS, this approach is applied. It involves identifying the different types of actors and their usage of the system. Also, for each actor the different
- Example (Library System):
    - Member – book issue/return/renew membership.
    - Librarian – manage members/books.
    - Accountant – manage fees/expenses.

Factoring Use Cases

Factoring of use cases are required under two situations:

1. ***Decompose Complex Use Cases*** – Break complex use cases into simpler use cases for comprehensive and manageable diagrams. Without decomposition, the interaction diagrams for complex use cases may become too large to be accommodated one a single sized paper.
2. ***Identify Commonality*** – Factor out based on common behavior such as error handling across a set of different use cases. This makes analysis of class design much simpler and elegant.
3. ***Generalization*** – Generalization is used when one use case is a specialized version of another (parent) use case. The child use case inherits the behavior of the parent but can add or modify it. Its notation is same as class inheritance (arrow pointing from child to parent). It is important to note that, both parent and child are separate use cases with their own descriptions.



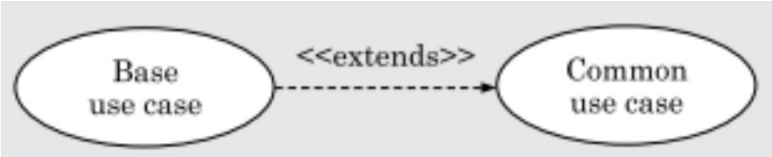
4. ***Includes*** – The include relationship implies one use case (base) *always* includes the behavior of another use case (common), as a reusable step. It is used to factor out common behavior shared by multiple use cases or to break complex use cases into manageable parts.

The notation for include is to use stereotype «`include`» pointing from the base use case to the included use case. Inclusion is mandatory and automatic whenever the base use case runs.

Examples:

- Use cases “Issue Book” and “Renew Book” both include “Check Reservation” because this check happens in both workflows.
- Use case “Create Account” includes “Validate Email” to ensure the email is valid during account creation

5. ***Extends*** – The extend relationship allows, one use case to show optional or conditional behavior to another use case (base). It is used to model optional or exceptional steps that happen only under certain conditions. It is similar to generalization but unlike generalization, the extending use case can add additional behavior only at an extension point only when certain conditions are satisfied.



Use stereotype «`extend`» pointing from the extending use case to the base use case.

Examples:

- Use case “Place Order” is extended by “Apply Discount” if a discount is applicable during ordering.
- Use case “Place Order” is extended by “Update Delivery Address” if the customer chooses to update the address during order placement.
- The “Update Delivery Address” can also be used independently by the customer outside of placing an order.