Introduction to UML (Unified Modeling Language)

Minsoo Ryu

Hanyang University

msryu@hanyang.ac.kr
Contents

1. Overview of UML
2. Structural Diagrams
3. Behavioral Diagrams
UML Definition

- UML is a visual modeling language for distributed and concurrent systems
  - Visual modeling - a way of thinking about problems using models organized around real-world ideas
  - Model - an abstraction that portrays the essentials of a complex problem or structure by filtering out nonessential details, thus making the problem easier to understand

- UML is officially defined at the Object Management Group (OMG) by the UML metamodel, a Meta-Object Facility metamodel (MOF)
  - UML was designed to specify, visualize, construct, and document software-intensive systems
UML Definition

- Triangle for success
  - Notation: a language for communication (semantics and a form)
  - Process: a way of using the notation
  - Tool: support to document the artifacts of the work
UML History

“Method wars” during the nineties: confusion brought by different notations
- OMT (Object Modeling Technique, James Rumbaugh)
  - strong in analysis and weaker in design
- OOD (Object Oriented Design, Grady Booch)
  - weaker in analysis and strong in design
- OOSE (Object Oriented Software Engineering, Ivar Jacobson)
  - strong in behavior analysis and weaker in others

The end of the method as far as notation is concerned
- Version 0.8 (October, 1995): The first public draft
- Version 0.9 (July, 1996), version 0.91 (October, 1996)
- Version 1.0 (Jan, 1997)
  - Under the technical leadership of the Three Amigos, proposed by an international consortium called the UML Partners
- Version 1.1 (November, 1997)
  - Semantics were finalized
UML History

- UML 2.0
  - Several minor revisions: UML 1.3, 1.4, 1.5
  - The first part of UML 2.0, the Superstructure which describes
    the new diagrams and modeling elements available, was
    adopted by the OMG in October 2004
  - Other parts of UML 2, notably the infrastructure, the Object
    Constraint Language (OCL) and the diagram interchange
    were yet to be completed and ratified as of November 2005

- Now UML version 2.3 is the latest one
UML Diagram Classification

☐ Structure Diagrams
  ▪ Emphasize what things must be in the system being modeled:

☐ Behavior Diagrams
  ▪ Emphasize what must happen in the system being modeled

☐ Interaction Diagrams (a subset of behavior diagrams)
  ▪ Emphasize the flow of control and data among the things in the system being modeled
Contents

1. Introduction to UML
2. Structural Diagrams
   • Class Diagram
   • Component Diagram
3. Behavioral Diagrams
What is a class?

- A class defines what information an object can hold and what behavior it can have.
- A class provides a blueprint for an object, and simply provides a template for objects.

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes.
Class Structure and Representation

- Attributes can appear in an optional compartment
  - Each attribute is shown with at least its name, and optionally with its type, initial value, and other properties
    - "+" for public
    - "#" for protected
    - "-" for private
    - "~" for package

- Operations can appear in another compartment
  - Each operation is shown with at least its name, and optionally also with its parameters and return type
Class Diagram
Class Stereotypes

- A stereotype conveys a high level category, type, or meaning of an item
  - A stereotype is a mechanism you can use to categorize your classes
Class Stereotypes

- There are three primary class stereotypes in UML that are used for analysis
  - Boundary, entity, and control
A relationship is a semantic connection between classes

- It allows one class to know about the attributes, operations, and relationships of another class

Types of relationships

- Associations
- Aggregations
- Compositions
- Generalizations
- Realizations
- Dependencies
When an association connects two classes, each class can send messages to the other.
Associations

- Instead of roles, another way to state the association is to write its meaning in terms of verb phrases
  - By convention, the phrase is placed next to the direct object of the sentence
Associations can also be reflexive

- A reflexive association suggests that one instance of a class is related to other instances of the same class
Aggregations

- Aggregations are used to depict elements which are made up of smaller components.
  - An “has a” relationship between a whole and its parts

- Aggregation can occur when the contained classes do not have a strong life cycle dependency on the container--essentially, if the container is destroyed, its contents are not
Composition is a stronger variant of the "has a" or association relationship; composition is more specific than aggregation.

Composition is a strong life cycle dependency between instances of the container class and instances of the contained class(es).

- If the container is destroyed, any containees are also destroyed.
Generalizations

- The generalization relationship indicates that one of the two related classes (the supertype) is considered to be a more general form of the other (the subtype).
  - “is a” relationship

- Generalizations are used to show an inheritance relationship between two modeling elements (actors, use cases, classes, or packages)
Realizations

A realization relationship is a relationship between two model elements, in which one model element (the client) realizes the behavior that the other model element (the supplier) specifies.

- Realizes relationships are used to show the relationship between a class and its interface, between a package and its interface, between a component and its interface, or between a use case and a use case realization.
- The relationship connects a publicly visible interface (interface class or use case) to the detailed implementation of the interface (class, package, or use case realization).
- In other words, this relationship helps separate an interface from its implementation.
Realizations

ControlPanel

- LCDdisplay
- LEDindicators
- keypadCharacteristics
- speaker
- wirelessInterface

- readKeyStroke()
- decodeKey()
- displayStatus()
- lightLEDs()
- sendControlMsg()

WirelessPDA

MobilePhone

KeyPad

<<interface>>

- readKeystroke()
- decodeKey()
Dependencies

- A dependency is a semantic relationship where a change to the influent or independent modeling element may affect the semantics of the dependent modeling element.

- Dependency is a model-level relationship and not a run-time relationship -- describing the need to investigate the model definition of the client element for possible changes if the model definition of the supplier element is changed (‘using’ relationship, `#include` in C++)
Multiplicity

Multiplicity defines how many objects participate in a relationship

- It is the number of instances of one class related to one instance of the other class
- For each association and aggregation, there are two multiplicity decisions to make: one for each end of the relationship
- Multiplicity is represented as a number and a * is used to represent a multiplicity of many
Multiplicity

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Many</td>
</tr>
<tr>
<td>0</td>
<td>Zero</td>
</tr>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>0..*</td>
<td>Zero or more</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more</td>
</tr>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>1..1</td>
<td>Exactly one</td>
</tr>
</tbody>
</table>

There are books not sold to any customer

But to be a Customer, you must have purchased at least one book
Contents

1. Introduction to UML
2. Structural Diagrams
   • Class Diagram
   • Component Diagram
3. Behavioral Diagrams
In UML 1.1, a component represented implementation items, such as files and executables

- Unfortunately, this conflicted with the more common use of the term component," which refers to things such as COM components

Over time and across successive releases of UML, the original UML meaning of components was mostly lost.

In UML 2, components are considered autonomous, encapsulated units within a system or subsystem that provide one or more interfaces

- Although the UML 2 specification does not strictly state it, components are larger design units that represent things that will typically be implemented using replaceable" modules
Component Diagram

- A component diagram depicts how a software system is split up into physical components and shows the dependencies among these components.

  - Physical components could be files, headers, link libraries, modules, executables, or packages.
  - Component diagrams can be used to model and document any system’s architecture.
Component Representation

Components are represented as a rectangular classifier with the keyword «component»; optionally the component may be displayed as a rectangle with a component icon in the right-hand upper corner.
Assembly Connector

- The assembly connector bridges a component’s required interface (Component1) with the provided interface of another component (Component2); this allows one component to provide the services that another component requires.
Components with Ports

- Ports allow for a service or behavior to be specified to its environment as well as a service or behavior that a component requires
  - Ports may specify inputs and outputs as they can operate bi-directionally
  - The following diagram details a component with a port for online services along with two provided interfaces: order entry and tracking as well as a required interface payment
Contents

1. Introduction to UML
2. Structural Diagrams
3. Behavioral Diagrams
   - Use Case Diagram
   - State Machine Diagram
   - Activity Diagram
   - Communication Diagram
   - Sequence Diagram
The use case model captures the requirements of a system

- Use cases are a means of communicating with users and other stakeholders what the system is intended to do
A use case diagram shows the interaction between the system and entities external to the system. These external entities are referred to as actors. Actors represent roles which may include human users, external hardware or other systems.
Use Cases

- A use case is a single unit of meaningful work
- It provides a high-level view of behavior observable to someone or something outside the system
Uses Connector

- The notation for using a use case is a connecting line with an optional arrowhead showing the direction of control.

- The uses connector can optionally have multiplicity values at each end, as in the following diagram, which shows a customer may only have one withdrawal session at a time, but a bank may have any number of customers making withdrawals concurrently.
Generalization

- Actors can generalize other actors as detailed in the following diagram
Include Relationship

- Use cases may contain the functionality of another use case as part of their normal processing
  - In general it is assumed that any included use case will be called every time the basic path is run

![Diagram showing include relationship between Withdraw and Card Identification use cases]
One use case may be used to extend the behavior of another; this is typically used in exceptional circumstances.

For example, if before modifying a particular type of customer order, a user must get approval from some higher authority, then the <Get Approval> use case may optionally extend the regular <Modify Order> use case.
Contents

1. Introduction to UML
2. Structural Diagrams
3. Behavioral Diagrams
   - Use Case Diagram
   - State Machine Diagram
   - Activity Diagram
   - Communication Diagram
   - Sequence Diagram
State Machine Diagram

- A state machine diagram models the behaviour of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.

![State Machine Diagram Image]
A state is denoted by a round-cornered rectangle with the name of the state written inside it. The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.
Transitions

Transitions from one state to the next are denoted by lines with arrowheads.

- "Trigger" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time.
- "Guard" is a condition which must be true in order for the trigger to cause the transition.
- "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.
State Actions

- On transition
  - An effect can be associated with the transition

- On entry
  - If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions

- On exit
  - This can be done by defining an entry action for the state
Self Transitions

- A state can have a transition that returns to itself, as in the following diagram.
- This is most useful when an effect is associated with the transition.
A state machine diagram may include sub-machine diagrams, as in the example below.
Entry Point

- Entry point is used **when you won’t want to enter a sub-machine at the normal initial state**
  - For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn’t necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.
Exit Point

- In a similar manner to entry points, it is possible to have named alternative exit points
  - The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state
A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving.

- The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.
Junction Pseudo-State

- Junction pseudo-states are used to chain together multiple transitions
  - A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition
  - Junctions are semantic-free
  - A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch
Junction Pseudo-State

![Diagram showing the junction pseudo-state process involving receiving voice, SMS, and fax messages, with options for creating corresponding messages.](image-url)
History States

- A history state is used to remember the previous state of a state machine when it was interrupted
  - The following diagram illustrates the use of history states
  - The example is a state machine belonging to a washing machine
Concurrent Regions

- A state may be divided into regions containing sub-states that exist and execute concurrently
  - The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently
  - Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.
Contents

1. Introduction to UML
2. Structural Diagrams
3. Behavioral Diagrams
   - Use Case Diagram
   - State Machine Diagram
   - Activity Diagram
   - Communication Diagram
   - Sequence Diagram
An activity diagram is used to display the sequence of activities.

- Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity.
- They may be used to detail situations where parallel processing may occur in the execution of some activities.
- Activity diagrams are useful for business modelling where they are used for detailing the processes involved in business activities.
Activity Diagram

Requested Order

Receive Order

[order accepted]

A [order rejected]

Fill Order

Process Order

Ship Order

Close Order

Send Invoice

Make Payment

Accept Payment

Invoice
Activities and Actions

- An activity is the specification of a parameterized sequence of behaviour
  - An activity is shown as a round-cornered rectangle enclosing all the actions, control flows and other elements that make up the activity

- An action represents a single step within an activity
  - Actions are denoted by round-cornered rectangles
Action Constraints

- Constraints can be attached to an action
- The following diagram shows an action with local pre- and post-conditions

![Diagram showing an action with local pre- and post-conditions]
Initial Node and Final Node

- An initial or start node is depicted by a large black spot.
- There are two types of final node: activity and flow final nodes.
  - The activity final node indicates that the activity has been completed.

![Diagram of initial and final nodes](image-url)
Flow Final Node

- The flow final node indicates that a single flow within an activity is complete
Objects and Object Flows

- An object flow is a path along which objects or data can pass.
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.
Objects and Object Flows

- An object flow must have an object on at least one of its ends
- A data store is shown as an object with the «datastore» keyword

![Diagram showing object flow and data store]
**Decision and Merge Nodes**

- **Decision nodes and merge nodes** have the same notation: a diamond shape
  - They can both be named
  - The control flows coming away from a decision node will have guard conditions which will allow control to flow if the guard condition is met
  - The following diagram shows use of a decision node and a merge node
Fork and Join Nodes

- Forks and joins have the same notation: either a horizontal or vertical bar
  - They indicate the start and end of concurrent threads of control
  - A join is different from a merge in that the join synchronizes two inflows and produces a single outflow
  - The outflow from a join cannot execute until all inflows have been received

![Diagram of fork and join nodes](image_url)
Signals

- When an activity sends or receives a message, that activity is called a signal.

  - **Input signal**
    - message receiving activity
  
  - **Output signal**
    - message sending activity
Time Signal

- The time signal is a special kind of a signal and denoted by a hour glass
  - A time signal is generated when a certain amount of time has passed
“Off-page” connectors are used to allow to span multiple pages

- There are two instances, a target of a flow and a source of a flow
- Both instances have the same name
- The connector indicates that a flow moves from one activity to another
Contents

1. Introduction to UML
2. Structural Diagrams
3. Behavioral Diagrams
   - Use Case Diagram
   - State Machine Diagram
   - Activity Diagram
   - Communication Diagram
   - Sequence Diagram
Communication Diagram

- A communication diagram, formerly called a collaboration diagram, is an interaction diagram that shows similar information to sequence diagrams but its primary focus is on object relationships.

- On communication diagrams, objects are shown with association connectors between them:
  - Messages are added to the associations and show as short arrows pointing in the direction of the message flow.
  - The sequence of messages is shown through a numbering scheme.
Communication Diagram

Staff Member
1: createContact
View Contact List
1.1: createContact
2: save
Edit Contact
2.1: saveContact
2.2: makePersistent
Contact Manager
1.2: openContact
Persistence Manager
2.3: insertContact
Contacts
1. Introduction to UML
2. Structural Diagrams
3. Behavioral Diagrams
   - Use Case Diagram
   - State Machine Diagram
   - Activity Diagram
   - Communication Diagram
   - Sequence Diagram
**Sequence Diagram**

- A sequence diagram is a form of interaction diagram which **shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline**
  - Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications
  - Sequence diagrams are not intended for showing complex procedural logic
Lifelines

- A lifeline represents an individual participant in a sequence diagram
  - A lifeline will usually have a rectangle containing its object name
  - If its name is "self", that indicates that the lifeline represents the classifier which owns the sequence diagram
Sequence Diagrams for Use Cases

Sometimes a sequence diagram will have a lifeline with an actor element symbol at its head

- This will usually be the case if the sequence diagram is owned by a use case
- Boundary, control and entity elements from robustness diagrams can also own lifelines
Messages

- Messages are displayed as arrows
  - A synchronous message (denoted by the solid arrowhead) operation does not allow the source object to continue its next step until the target object completes the flow of control.
  - An asynchronous message (denoted by line arrowhead) operation allows the source object to immediately continue with the next step.

![Diagram of Messages]
Execution Occurrence

- A thin rectangle running down the lifeline denotes the execution occurrence, or activation of a focus of control
  - In the previous diagram, there are three execution occurrences
  - The first is the source object sending two messages and receiving two replies; the second is the target object receiving a synchronous message and returning a reply; and the third is the target object receiving an asynchronous message and returning a reply
Self Message

- A self message can represent a recursive call of an operation, or one method calling another method belonging to the same object
A lifeline may be created or destroyed during the timescale represented by a sequence diagram

- In the latter case, the lifeline is terminated by a stop symbol, represented as a cross
- In the former case, the symbol at the head of the lifeline is shown at a lower level down the page than the symbol of the object that caused the creation
By default, a message is shown as a horizontal line. Since the lifeline represents the passage of time down the screen, when modelling a real-time system, or even a time-bound business process, it can be important to consider the length of time it takes to perform actions.

- By setting a duration constraint for a message, the message will be shown as a sloping line.