## CONTENT

### A Conceptual Model of the UML

#### Building Blocks of the UML

1. Things
   - [1.1] Structural Things
     - (1.1.1) Class
     - (1.1.2) Interface
     - (1.1.3) Collaboration:
       - (1.1.4) Use case
     - (1.1.5) Components:
     - (1.1.6) Node:
   - [1.2] Behavioral things
     - (1.2.1) Interaction:
     - (1.2.2) State machine:
   - [1.3] Grouping things
     - (1.3.1) State machine:
   - [1.4] Annotational things
     - (1.4.1) Note:

2. Relationship
   - [2.1] Dependency
   - [2.2] Association
   - [2.3] Generalization
   - [2.4] Realization

3. Diagrams

#### Rules of the UML

#### Common Mechanisms in the UML

1. Specification
2. Adornments
3. Common divisions
   - [3.1] Classes and Objects
   - [3.2] Interfaces and Implementations
4. Extensibility mechanisms
   - [4.1] Stereotypes
   - [4.1] Tagged values
   - [4.1] Constraints
A Conceptual Model of the UML

To understand the UML, a person needs to form a conceptual model of the language, and this requires learning three major elements:

I. Building Blocks of the UML
II. Rules of the UML
III. Common Mechanisms in the UML

Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks:

1. Things
2. Relationships
3. Diagrams

1. Things

Things are the abstractions that are first-class citizens in a model.

There are four kinds of things in the UML:

[1.1] Structural things
[1.2] Behavioral things
[1.3] Grouping things
[1.4] Annotational things

These things are the basic object-oriented building blocks of the UML.

[1.1] Structural Things

Structural things are the nouns of UML models.

These are the mostly static parts of a model, representing elements that are either conceptual or physical.

In all, there are several kinds of structural things.

(1.1.1) Class

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

Graphical notation of class:

<table>
<thead>
<tr>
<th>Window</th>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>Attributes</td>
</tr>
<tr>
<td>size</td>
<td>Operations</td>
</tr>
<tr>
<td>open()</td>
<td></td>
</tr>
<tr>
<td>close()</td>
<td></td>
</tr>
<tr>
<td>move()</td>
<td></td>
</tr>
<tr>
<td>display()</td>
<td></td>
</tr>
</tbody>
</table>
(1.1.2) Interface

- An interface is a collection of operations that specify a service of a class.

(1.1.3) Collaboration:

- Collaboration defines interaction between elements.
- Graphical notation of Collaboration

(1.1.4) Use case:

- Use case represents a set of actions performed by a system for a specific goal.
- Graphical notation of Use case

(1.1.5) Components:

- Component describes physical element that exists at run time.
- Graphical notation of Component
(1.1.6) Node:
- A node can be defined as a physical element that exists at run time.
- Graphical notation for Node

[1.2] Behavioral things
- Behavioral things are the dynamic parts of UML models.
- These are the verbs of a model, representing behavior over time and space.
- There are several kinds of Behavioral things.

(1.2.1) Interaction:
- An interaction is a behavior that includes a set of messages exchanged among a set of objects.
- Graphical notation for interaction

(1.2.2) State machine:
- A state machine is a behavior that specifies the sequences of states an object.
- Graphical notation for State
[1.3] Grouping things

- Grouping things are the organizational parts of UML models.
- These are the boxes into which a model can be decomposed.
- There is only one primary kind of grouping thing, namely, packages.

(1.3.1) State machine:

- A package is a general-purpose mechanism for organizing elements into groups.
- Graphical notation for Package

![Diagram of a package]

[1.4] Annotational things

- Annotational things are the explanatory parts of UML models.
- It can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.
- There is only one Annotational thing available:

(1.4.1) Note:

- A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements...
- Graphical notation for note

![Diagram of a note]
2. Relationship

Relationship is another most important building block of UML.
Relationships tie things together.

There are four kinds of relationships in the UML:

[2.1] Dependency

Dependency is a relationship between two things in which change in one element also affects the other one.
Graphical notation for Dependency:

[2.2] Association

An association is a structural relationship among classes that describes a set of links.
Graphical notation for Association:

0..1
-------------
employer employee

[2.3] Generalization

A generalization is a specialization relationship in which objects of the child are substitutable for objects of the generalized element the parent.
In this way, the child shares the structure and the behavior of the parent.
Graphical notation for Generalization:

[2.4] Realization

A realization is a semantic relationship between elements, wherein one element (class) specifies a responsibility (method) and other one implements them.
This relationship exists in case of interfaces.
Graphical notation for Realization:
3. Diagrams

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships).

In general terms, Diagrams group interesting collections of things.

Draw diagrams to visualize a system from different perspectives, so a diagram is a projection into a system.

the UML includes thirteen kinds of diagrams:

1. Class diagram
2. Object diagram
3. Component diagram
4. Composite structure diagram
5. Use case diagram
6. Sequence diagram
7. Communication diagram
8. State diagram
9. Activity diagram
10. Deployment diagram
11. Package diagram
12. Timing diagram
13. Interaction overview diagram

Rules of the UML

The UML's building blocks can't simply be thrown together in a random fashion.

Like any language, the UML has a number of rules that specify what a well-formed model should look like.

A well-formed model is one that is semantically self-consistent and in synchronizes with all its related models.

The UML has semantic rules for:

- **Names**: What designer can call things, relationships, and diagrams
- **Scope**: The context that gives specific meaning to a name
- **Visibility**: How those names can be seen and used by others
- **Integrity**: How things properly and consistently relate to one another
- **Execution**: What it means to run or simulate a dynamic model
Common Mechanisms in the UML  
(third topic of Unit: 6)

- UML is made simpler by the presence of four common mechanisms that apply consistently throughout the language.
  
  1. Specifications
  2. Adornments
  3. Common divisions
  4. Extensibility mechanisms

1. Specification

- The UML is more than just a graphical language.
- Rather, behind every part of its graphical notation there is a specification that provides a textual statement of the syntax and semantics of that building block.
- For example, behind a class icon is a specification that provides the full set of attributes, operations, and behaviors that the class represents; visually, that class icon might only show a small part of this specification.
- The UML's specifications provide a semantic backplane that contains all the parts of all the models of a system, each part related to one another in a consistent fashion.

2. Adornments (meaning: ornament)

- Most elements in the UML have a unique and direct graphical notation that provides a visual representation of the most important aspects of the element.
- For example, the notation for a class is intentionally designed to be easy to draw, because classes are the most common element found in modeling object-oriented systems. The class notation also exposes the most important aspects of a class, namely its name, attributes, and operations.
- A class's specification may include other details, such as whether it is abstract or the visibility of its attributes and operations.
- Many of these details can be rendered as graphical or textual adornments to the class's basic rectangular notation.
- Figure shows a class, adorned to indicate that it is an abstract class with two public, one protected, and one private operation.

<table>
<thead>
<tr>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ execute()</td>
</tr>
<tr>
<td>+ rollback()</td>
</tr>
<tr>
<td># priority()</td>
</tr>
<tr>
<td>- timestamp()</td>
</tr>
</tbody>
</table>
3. Common divisions

In modeling object-oriented systems, the world often gets divided in at least a couple of ways.

3.1 Classes and Objects

- A class is an abstraction.
- An object is one concrete manifestation of that abstraction.
- In the UML, one can model classes as well as objects, as shown in Figure.

\[
\begin{array}{c|c}
\text{Customer} & \text{Jan : Customer} \\
\text{name} & : \text{Customer} \\
\text{address} & Elyse \\
\text{phone} &
\end{array}
\]

- Graphically, the UML distinguishes an object by using the same symbol as its class and then simply underlying the object's name.
- In this figure, there is one class, named Customer, together with three objects: Jan (which is marked explicitly as being a Customer object), :Customer (an anonymous Customer object), and Elyse (which in its specification is marked as being a kind of Customer object, although it's not shown explicitly here).

3.2 Interfaces and Implementations

- An interface declares a contract.
- An implementation represents one concrete realization of that contract, responsible for faithfully carrying out the interface's complete semantics.
- In the UML, one can model both interfaces and their implementations, as shown in Figure.

\[
\begin{array}{c}
\text{IUnknown} \\
\text{ISpelling} \\
\text{SpellingWizard} \\
\text{IDictionary}
\end{array}
\]

- In this figure, there is one component named SpellingWizard.dll that provides (implements) two interfaces, IUnknown and ISpelling.
- It also requires an interface, IDictionary, that must be provided by another component.

4. Extensibility mechanisms

- The UML provides a standard language for writing software blueprints, but it is not possible for one closed language to ever be sufficient to express all possible things of all models across all domains across all time.
- For this reason, the UML is opened-ended, making it possible for designer to extend the language in controlled ways.
- The UML's extensibility mechanisms include
  - [4.1] Stereotypes
  - [4.2] Tagged values
  - [4.3] Constraints
[4.1] **Stereotypes**
- An extension of the vocabulary of the UML that allows to create new kinds of building blocks derived from existing ones but that are specific to problem.
- For example, if person is working in a programming language, such as Java or C++, he/she will often want to model exceptions.
- In these languages, exceptions are just classes, although they are treated in very special ways.
- Typically, person only want to allow them to be thrown and caught, nothing else.
- Person can make exceptions in their models meaning that they are treated like basic building blocks by marking them with an appropriate stereotype, as for the class *Overflow* in above Figure.

[4.1] **Tagged values**
- A tagged value extends the properties of a UML stereotype, allowing to create new information in the stereotype's specification.
- For example, if person is working on a shrink-wrapped product that undergoes many releases over time, he/she often want to track the version and author of certain critical abstractions.
- Version and author are not primitive UML concepts.
- They can be added to any building block, such as a class, by introducing new tagged values to that building block.
- For example in Figure, the class *EventQueue* is extended by marking its version and author explicitly.

[4.1] **Constraints**
- A constraint extends the semantics of a UML building block, allowing to add new rules or modify existing ones.
- For example, user might want to constrain the *EventQueue* class so that all additions are done in order.
- As Figure shows, one can add a constraint that explicitly marks these for the operation *add*. 