Development of Component- and Service-Oriented Software Architectures with UML and UML-RT — Objects and Classes —

Ingolf H. Krueger
ikrueger@ucsd.edu

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0404, USA

California Institute for Telecommunications and Information Technologies
La Jolla, CA 92093-0405, USA
Overview

- Objects
- Classes
- Class and Object Diagrams
Advantages of Object-Orientation

• Seamless methodological approach covering analysis to implementation
  – OO modeling techniques (classes/objects)
  – OO implementation languages (C++, Java, Smalltalk, ...)

• Support for understanding the application domain
  – Mapping of “real” entities to objects of the model
  – Abstraction mechanisms available

• Starting points for reuse of structure and behavior:
  – Classes,
  – Interfaces,
  – Objects/components
Advantages of Object-Orientation

• Adequate even for modeling of large systems

• Support for developing “safe” components:
  – Interfaces
  – Encapsulation of data structures and behavior

• Support for distribution/parallelism
  – Objects as independent and active entities executing in parallel
Objects

- Objects model (individual) entities of the application domain

- Example (telecommunication):
  - Caller
  - Connection
  - Account
  - ...

- Objects have unique identity

- Objects encapsulate
  - Data/state
  - Behavior
Objects

- Access only via explicit message exchange

⇒ Integrity of data

- Operation/Method:
  enable specification of pre-/postconditions
Example: Stack-Object

- **Attribute(s):** Encapsulated data
- **Operations:**
  - void push(int)
  - int pop()

Diagram:
- **Object**
- **Attribute(s):**
- **Operations:**
- **int [10] stack**
Example: Stack-Object

- int [10] stack
- void push(int)
- int pop()
Example: Stack-Object

```c
int [10] stack
void push(int)
int pop()
```

Example:
push(3)

```
3
2
```

push(3)
Example: Stack-Object

- int [10] stack
- void push(int)
- int pop()
- pop()
- 2
- 3
Example: Stack-Object

int [10] stack

void push(int)

push(5)

int pop()

5
2
Example: Stack-Object

- `void push(int)`
- `int [10] stack`
- `int pop()`

Input: `2`

Output: `5`
Example: Stack-Object

Example: Stack-Object
Example: Stack-Operations

- **Operation “push(arg)”**:  
  - only allowed, when stack not full  
    (otherwise: exception handling)  
  - arg becomes new top element of stack,  
  - rest of stack remains unchanged  

- **Operation “pop()”**:  
  - only allowed, when stack not empty  
    (otherwise: exception handling)  
  - top stack element is returned,  
    and removed from stack,  
  - rest of stack remains unchanged  

Precondition

Postcondition
Example: Elevator Controller

- Cabin
- Position Display
- Door
- User Console
- Button
- Lock
- Elevator Control
- Scheduling strategy

Object

Relationship
Example: Elevator Controller

**State:**
- unlocked/locked/
- unlocking/locking

**Operations:**
- unlock
- lock

**Position Display**

**Door**

**User Console**

**Button**

**Lock**

**Elevator Control**

**State:**
- Running/Stopped

**Operations:**
- switch on
- switch off

**State:**
- unlocked/locked/
- unlocking/locking

**Operations:**
- unlock
- lock
Overview

- Objects
- Classes
- Class and Object Diagrams
Classes

• Objects model (individual) entities of the application domain

• Many objects follow the same “blue-print”
  Example (elevator):
  – Cabins
  – Locks
  – Buttons
  – Scheduling strategies

• “blue-print” consists of
  – data-/state-structures
  – Patterns of behavior
**Classes**

- **Class** in OO approach:
  - Description of the similarities (attributes and operations) of a set of objects
  - Template for instantiating individual objects

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Marital status</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Christina</td>
<td>24</td>
<td>m</td>
<td>Teacher</td>
</tr>
<tr>
<td>Herbert</td>
<td>56</td>
<td>m</td>
<td>Technician</td>
</tr>
<tr>
<td>Martin</td>
<td>31</td>
<td>u</td>
<td>HR consultant</td>
</tr>
</tbody>
</table>

```python
p1: Person
Christina, 24, m, Teacher

set_name(adr)is_adult()
```
Classes

• Classes are one of the key means for structuring and abstraction in OO

• A class models a **concept** of the application domain / implementation

• Each object is an **instance** of a class
  synonyms: object, instance, class instance, ...

• Classes define templates for individual objects

• Class structure, relations among classes: **class diagram**

• Object structure, relations among objects: **object diagram**
Overview

- Objects
- Classes
- Class and Object Diagrams
Class and Object Diagrams

• **Class diagrams:**
  – Representation of classes and their relationships
  – Examples for relationships among classes:
    • Existence of a communication path
    • Inclusion
    • Generalization/Specialization

• **Object Diagrams:**
  – Syntax as in class diagrams
  – But: representation of objects instead of classes
Class Diagrams: Modeling Elements

- Classes
- Attributes
- Operations

Description of "object template"

- Association
- Aggregation/Composition
- Generalization/Specialization

Description of relationships among multiple classes

- Conditions

Restrictions of the corresponding object model
Class Diagrams

<table>
<thead>
<tr>
<th>Name</th>
<th>Attributes</th>
<th>Operations/Methods</th>
<th>Association</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>name, address, birthday</td>
<td>change_adr(adr: String), adult() : Bool</td>
<td>is_owner_of</td>
<td>*</td>
</tr>
<tr>
<td>Account</td>
<td>account_number, balance, rate_of_interest</td>
<td>credit(x:Real), debit(x: Real), close</td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
Associations

Associations describe navigation paths through the class/object model.

Meaning: There is a link (e.g. communication path) between instances of classes “Cabin” and “User Console”.

Here: navigation in both directions possible.
Associations: Implementation

class Cabin {
    User_Console bc = null;
    void setCo(User_Console _bc) {
        bc = _bc;
    }
}

class User_Console {
    Cabin c = null;
    void setCa(Cabin _c) {
        c = _c;
    }
}

User_Console console = new User_Console();
Cabin cabin = new Cabin();

console.setCa(cabin); cabin.setCo(console);
Other Association Types

“Cabin” instances can access “User Console” instances, but not the other way around!
Other Association Types

Name of Association

```
class Company {
    private Person[] employees;
    ...
}
```
Other Association Types

- Person
  - 1..* Employee
  - works_for
  - 0..1 Employer

“direction” of association
Other Association Types

A Person works for at most one (i.e. “0..1”) Company

A Company employs at least one (i.e. “1..*”) Person
Other Association Types

General Format for Multiplicities:

\[ a..b \quad (a, b: \text{natural numbers or "*"}) \]
\[ * = *..* = 0..* \]
\[ 1 = 1..1 \quad (\text{exactly one}) \]
\[ n = n..n \quad (n: \text{natural number}) \]

<table>
<thead>
<tr>
<th>Person</th>
<th>1..* works_for</th>
<th>0..1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>◀ employs</td>
<td>Employer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other Association Types: Aggregation

Every user console “has” 1-12 buttons and exactly one lock

Every lock/button belongs to exactly one user console
Other Association Types: Composition

Every user console contains 1-12 buttons and exactly one lock.

The user console is responsible for construction and destruction of locks and buttons; it controls the “life cycle” of its locks/buttons.

Every button/lock belongs to exactly one user console.
General Forms of Aggregation

- **Constant Aggregation**
  Every sub-object is part of at most one “container” over the system’s runtime

- **Dependent Aggregation**
  No sub-object can exist without a “container”; it must be part of at least one “container”

- **Encapsulated Aggregation**
  Sub-objects receive messages only from within their container; there is no communication beyond container boundaries

- **UML:**
  - Composition = Constant Aggregation + Dependent Aggregation
  - Weak Aggregation: No restrictions
Inheritance

Inheritance: Generalization/Specialization on class level

Diagram showing the inheritance relationship between Person (superclass) and Employee (subclass).
Inheritance

Inheritance: Generalization/Specialization on class level

Person

Employee

Attribute Inheritance

Method Inheritance

New Attributes and Methods
Inheritance

- Subclasses inherit attributes, associations and operations from their superclasses
- Subclasses can overwrite inherited operations
- Each instance of a subclass is also an instance of its superclass (Polymorphism)
Polymorphism

```java
void draw_content(Graphic_Element ge) {
    ge.draw();
}

draw_content( new Text() );
draw_content( new Shape() );
draw_content( new Group() );
```
Limited Inheritance

Public Member:
Accessible in every class definition

DemoClass
+publicInt:int
#protectedInt:int
-privateInt:int

+publicMethod():void
#protectedMethod():void
-privateMethod():void

Subclass
Limited Inheritance

DemoClass

+publicInt:int

#protectedInt:int

-privateInt:int

+publicMethod():void

#protectedMethod():void

-privateMethod():void

Protected Member:
Accessible in DemoClass and all of its subclasses

Subclass
Limited Inheritance

DemoClass

+publicInt:int
#protectedInt:int
-privateInt:int

+publicMethod():void
#protectedMethod():void
-privateMethod():void

Private Member:
Accessible only in DemoClass

Subclass
Multiple Inheritance

Attention: reserve for abstract classes/interfaces
Object Diagrams

- Syntax same as for class diagrams
- Typically no attributes and operations shown
- Object name and class name is underscored

```
Class name
:Cabin
:Position Display
:Door
:User Console
:Button
:Lock

Anonymous Object

Named Object

s:Scheduling Strategy

:Elevator Control
```

```
Summary Class and Object Diagrams

• Class and object diagrams describe classes and objects, respectively, and their relationships:
  – intuitively: objects of class A “know” objects of class B (have references to objects of class B)
  – Multiplicities, Association names, Role names
  – Aggregation/Composition
  – Generalization/Specialization
  – Polymorphism

• Object diagrams: “snapshot” of running system

• View on system structure

• Direct transformation into implementation