Software Design

Static Modeling using the Unified Modeling Language (UML)

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

Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.
The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.
An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.
## Class Attributes (Cont’d)

Attributes are usually listed in the form:

```
attributeName : Type
```

A *derived* attribute is one that can be computed from other attributes, but doesn’t actually exist. For example, a Person’s age can be computed from his birth date. A derived attribute is designated by a preceding ‘/’ as in:

```
/ age : Date
```
### Class Attributes (Cont’d)

<table>
<thead>
<tr>
<th>Person</th>
<th>Attributes can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ name : String</td>
<td>+ public</td>
</tr>
<tr>
<td># address : Address</td>
<td># protected</td>
</tr>
<tr>
<td># birthdate : Date</td>
<td>- private</td>
</tr>
<tr>
<td>/ age : Date</td>
<td>/ derived</td>
</tr>
<tr>
<td>- ssn : Id</td>
<td></td>
</tr>
</tbody>
</table>

Attributes can be:
- public
- protected
- private
- derived
Class Operations

Person

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td>address</td>
<td>Address</td>
</tr>
<tr>
<td>birthdate</td>
<td>Date</td>
</tr>
<tr>
<td>ssn</td>
<td>Id</td>
</tr>
</tbody>
</table>

Operations describe the class behavior and appear in the third compartment.

eat
sleep
work
play
Class Operations (Cont’d)

NewEntry (n : Name, a : Address, p : PhoneNumber, d : Description)
getPhone ( n : Name, a : Address) : PhoneNumber

You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.
Depicting Classes

When drawing a class, you needn’t show attributes and operation in every diagram.
Class Responsibilities

A class may also include its responsibilities in a class diagram.

A responsibility is a contract or obligation of a class to perform a particular service.

<table>
<thead>
<tr>
<th>SmokeAlarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
</tr>
<tr>
<td>-- sound alert and notify guard station when smoke is detected.</td>
</tr>
<tr>
<td>-- indicate battery state</td>
</tr>
</tbody>
</table>
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

- dependencies
- generalizations
- associations
A *dependency* indicates a semantic relationship between two or more elements. The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*. 

![Diagram](attachment:dependency_diagram.png)
Generalization Relationships

A generalization connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.
Generalization Relationships (Cont’d)

UML permits a class to inherit from multiple superclasses, although some programming languages (e.g., Java) do not permit multiple inheritance.

![Diagram showing inheritance relationships between Student, TeachingAssistant, and Employee classes.](image-url)
Association Relationships

If two classes in a model need to communicate with each other, there must be a link between them.

An association denotes that link.
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:
The example indicates that every Instructor has one or more Students:
We can also indicate the behavior of an object in an association (i.e., the role of an object) using rolenames.
Association Relationships (Cont’d)

We can also name the association.

- Student
  - Team
    - membership
      - 1..*
      - 1..*
We can specify dual associations.

\[
\begin{array}{c}
\text{Student} \\
\quad \quad 1..* \\
\quad \quad 1 \\
\hline \\
\text{member of} \\
\quad \quad 1..* \\
\quad \quad 1..* \\
\text{Team} \\
\end{array}
\]
Association Relationships (Cont’d)

We can constrain the association relationship by defining the *navigability* of the association. Here, a *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of the *Router*.
Association Relationships (Cont’d)

Associations can also be objects themselves, called link classes or an association classes.

```
Registration
  modelNumber
  serialNumber
  warrantyCode

Product  Warranty
```
A class can have a *self association*. 
We can model objects that contain other objects by way of special associations called aggregations and compositions.

An aggregation specifies a whole-part relationship between an aggregate (a whole) and a constituent part, where the part can exist independently from the aggregate. Aggregations are denoted by a hollow-diamond adornment on the association.
A *composition* indicates a strong ownership and coincident lifetime of parts by the whole (*i.e.*, they live and die as a whole). Compositions are denoted by a filled-diamond adornment on the association.

![Diagram of association relationships]
Interfaces

An interface is a named set of operations that specifies the behavior of objects without showing their inner structure. It can be rendered in the model by a one- or two-compartment rectangle, with the stereotype <<interface>> above the interface name.
Interfaces do not get instantiated. They have no attributes or state. Rather, they specify the services offered by a related class.
A realization relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.
Interfaces

A class’ interface can also be rendered by a circle connected to a class by a solid line.

FileWriter

FileReader

{file must not be locked}

{file must exist}
Software Design

Dynamic Modeling using the Unified Modeling Language (UML)
Use Case

“A use case specifies the behavior of a system or a part of a system, and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.”

- The UML User Guide, [Booch,99]

“An actor is an idealization of an external person, process, or thing interacting with a system, subsystem, or class. An actor characterizes the interactions that outside users may have with the system.”

- The UML Reference Manual, [Rumbaugh,99]
Use Case (Cont’d)

A use case is rendered as an ellipse in a use case diagram. A use case is always labeled with its name.

Register for Courses
Use Case (Cont’d)

An actor is rendered as a stick figure in a use case diagram. Each actor participates in one or more use cases.

Student
Use Case (Cont’d)

Actors can participate in a generalization relation with other actors.

Student ➔ Person
Use Case (Cont’d)

Actors may be connected to use cases only by associations.

Student → Register for Courses
Here we have a *Student* interacting with the *Registrar* and the *Billing System* via a “Register for Courses” use case.
State Machine

“The state machine view describes the dynamic behavior of objects over time by modeling the lifecycles of objects of each class. Each object is treated as an isolated entity that communicates with the rest of the world by detecting events and responding to them. Events represent the kinds of changes that objects can detect... Anything that can affect an object can be characterized as an event.”

- The UML Reference Manual, [Rumbaugh,99]
State Machine

An object must be in some specific state at any given time during its lifecycle. An object transitions from one state to another as the result of some event that affects it. You may create a state diagram for any class, collaboration, operation, or use case in a UML model.

There can be only one start state in a state diagram, but there may be many intermediate and final states.
State Machine

- start state
- simple state
- concurrent composite state
- sequential composite state
- final state
Software Design (UML)
Sequence Diagram

*A sequence diagram* is an interaction diagram that emphasizes the time ordering of messages. It shows a set of objects and the messages sent and received by those objects.

Graphically, a sequence diagram is a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis.

Sequence Diagram

An object in a sequence diagram is rendered as a box with a dashed line descending from it. The line is called the *object lifeline*, and it represents the existence of an object over a period of time.
Sequence Diagram

Messages are rendered as horizontal arrows being passed from object to object as time advances down the object lifelines. Conditions (such as \[\text{check = “true”}\]) indicate when a message gets passed.
Notice that the bottom arrow is different. The arrow head is not solid, and there is no accompanying message.

This arrow indicates a return from a previous message, not a new message.
An iteration marker, such as * (as shown), or *\[i = 1..n]\ , indicates that a message will be repeated as indicated.
an Order Entry window

prepare()

* prepare()

check()

[check = "true"]

remove()

needsToReorder()

[needsToReorder = "true"]

new A Reorder Item

[check = "true"]

new A Delivery Item

[Fowler,97]
Activity Diagram

An activity diagram is essentially a flowchart, showing the flow of control from activity to activity.

Use activity diagrams to specify, construct, and document the dynamics of a society of objects, or to model the flow of control of an operation. Whereas interaction diagrams emphasize the flow of control from object to object, activity diagrams emphasize the flow of control from activity to activity. An activity is an ongoing non-atomic execution within a state machine.

- The UML User Guide, [Booch,99]
Receive Order

Multiple Trigger

for each line item on order

Authorize Payment

[Succeeded]

Check Line Item

[in stock]

Assign to Order

[need to reorder]

Reorder Item

Dispatch Order

Synchronization Condition

[stock assigned to all line items and payment authorized]

Cancel Order

[failed]
References


[Brown99] First draft of these slides were created by James Brown.