Today’s Agenda

- Class (concept) diagrams (domain models)
  - Classes
  - Associations
  - Attributes
  - Operations
  - Generalization, aggregation, composition
  - Navigability
  - Multiplicity
  - Object diagrams
- Communication Diagrams
- Package Diagrams

Example: Library System (next class we will continue with the stopwatch)

Readings: Arlow and Neustadt Ch. 6,7,8,9 (skip 9.5), 10, 11,18.1-18.5
Objects

An object is "A discrete entity with a well-defined boundary that encapsulates state and behavior." [Rumbaugh]

Every object has:

- an identity
- state (attribute values and relationships with other objects at a particular point in time)
- behaviour (services offered) – invoking an operation may cause a change in state

An operation is the specification of a behaviour.
A method is the implementation of the behaviour.

Classes

A class is a collection of objects with a common set of features (attributes, operations, relationships). (think "type")

Every object belongs to one class.

A class diagram depicts the kind of objects involved in the problem and their static relationships. It is a form of ERD.

Elements of a class diagram:

- Classes
  - Name
  - Attributes
  - Operations
- Relationships:
  - Associations
  - Generalizations (is-a)
  - Aggregations, Compositions (has-a/whole-part)

Example

```
<table>
<thead>
<tr>
<th>Driver</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>price: Money</td>
</tr>
<tr>
<td>address: String</td>
<td>itemNum: integer</td>
</tr>
<tr>
<td>changeAddress</td>
<td>owns</td>
</tr>
</tbody>
</table>

| newAddress: String | String |
```

See p. 140 of UML text for UML primitive types. But feel free to invent meaningful primitive types for your specification, such as "money" or "weight" and include these in the glossary.
Class Diagrams: Perspectives

UML class diagrams can be used in three distinct ways depending on the phase of system development:

According to Fowler and Cook&Daniels:

1. **Conceptual (Domain Models)**
   - requirements phase (elaboration phase)
   - the diagram represents the entities in the problem/business domain
   - not necessarily a direct mapping to how these will be implemented (although they may end up being classes in the software)
   - also called analysis classes

2. **Design**
   - the diagram depicts only the interfaces of software classes, but still avoids implementation details
   - information hiding

3. **Implementation**
   - the diagram depicts interfaces and implementations of classes

Conceptual Diagrams

This course is about writing conceptual diagrams (domain models):

- Diagrams that describe the system’s behaviour in terms of how it affects its environment
- Describe real-world entities – the information the system will monitor, store, transform, analyze, display, etc; physical and conceptual
- Larman refers to the concept (class) diagram for RE as a “visual dictionary”

What is a class?

Classes are entities from the problem domain:

- “crisp abstraction of the problem domain” [Arlow and Neustadt]
- map to real-world concepts
- any information that the system stores, analyzes, transforms, displays, etc.
- transient objects (e.g., business transactions, phone conversations)
- actors that interact with system. (If so, show the boundary of the system on your class diagram.)

Classes are named, usually, by short singular nouns.

Speak the customer’s language!
Attributes

An attribute is simple data associated with a class.
The data that is too simple to have a class of its own (e.g., numbers, text).
Attributes can also be viewed as properties of a class, information that distinguishes one instance of the class from another instance. They are distinguishing characteristics of the objects.
Syntax: name : type = default value
(default value is optional)
Again, these are just the attributes relevant at the RE level of description.

Operations

Operations are the responsibilities/services of an object in the class.
- query value of an object's attributes
- transform values of object’s attributes
  - may result in state changes
Syntax: name (parameters) : return-type {properties}
Together, a class's attributes and operations document the purpose of the class — what information it maintains, and how can that information be manipulated.

Associations

An association is a relationship between classes.
Relationships between objects, i.e., instances of associations are called links.
A link allows messages to be sent from one object to another. These are operation invocations.
Associations are names, usually short verbs. Some people name every association. Others name associations only when such names will improve understanding. e.g., avoid names like “is related to”, and “has”.
It is possible to have two associations between the same two classes.
**Associations**

We want to avoid having too many associations, which creates “visual noise”. Larman distinguishes two types of associations:

- **needs-to-know** – a relation between classes that needs to be remembered
- **comprehension-only** – information that helps us to understand the domain

Mostly we want to have “needs-to-know” associations.

**Common Associations List**

Examples of common associations (Larman, p. 156):

- A is owned by B
- A is a description for B
- A is a physical part of B (special type of association)
- A is a logical part of B (special type of association)
- etc.

Avoid making derivable information an association.

**Method**

To build a conceptual model:

1. Identify candidate list of classes
2. Draw class diagram
3. Add associations
4. Add attributes
5. Add operations

**Finding Classes**

There are a variety of methods:

- Noun/Verb Analysis
- Extract from System Sequence and State Diagrams (next class)
- CRC (Class Responsibilities Collaborators) Analysis
  - brainstorm about all the entities that are relevant to the system and their relationships
  - use sticky notes and a whiteboard

You can use multiple approaches and then consolidate.

Expect this process to be iterative!
Noun/Verb Analysis

- Go through the use cases:
  - nouns or noun phrases -> classes or attributes or associations
  - verbs or verb phrases -> operations
- Watch out for ambiguities and redundant concepts (synonyms and homonyms)
- Watch for hidden classes (useful abstractions that make the diagram simpler)

Attributes are things that seem to be part of another thing.

Common Types of Classes

- interface/boundary classes – those that communicate with external actors (user interfaces, system/device interfaces)
- controller/manager classes – coordinate system behaviour; cut across multiple use cases
- “entity” classes – simple behaviour – get and set values
- “specification or description class”
  - If a description of an item is needed independent of the existence of the item
  - Often found for items that can be deleted
  - Example: may require a description of a service separate from the actual sold services.

These common types of classes can be designated using stereotypes.

Library Example

A library system has patrons who borrow publications using a loan.
The publications can be books or periodicals.
There are due dates and fines associated with loans.
The library tracks the accumulated fines of all patrons.

Association Class

An association class allows you to add attributes, operations, and other features to an association.

Sometimes there is information that cannot be attributed to solely one class or another, but is really information about the association.

Syntax: Class connected to association by dashed line.

There must be only 1 instance of association class between any two associated objects – i.e., the instance must be uniquely determined.
**Generalization**

Generalization indicates a sub-class relation.

All instances of the subclass are instances of the superclass.

A subclass inherits all attributes, operations and associations of the parent.

The common attributes and operations are placed in the superclass; subclasses extend the attributes, operations, and associations as they need them.

**Syntax:** open triangle at the superclass end of the association

---

**Generalization**

Each level in the generalization hierarchy should be at the same level of abstraction.

In RE, generalization is not for the sake of code reuse! Explore inheritance if these abstractions help make the model simpler. Sometimes looking at the general case can result in simpler models.

---

**Generalization**

Is property permanent for object or does it change at run time?

If property is permanent for an object, then the property is the basis for a subclass. Objects of differing subclasses can have different components.

If property changes at run time for an object, then the property is an attribute to be set at run time.

The components of an object do not change at run time.

---

**Generalization**

Generalization is often called the “is-a” relationship.

Often, we can use the “is-a” test to see if you have a generalization relation between two classes:

Can you say that an instance of one class “is-a” an instance of the other class?

**Example:** A student in this class is a student at the University of Waterloo.
Warning!

The “is-a” test doesn’t always work!
From Fowler:
1. Shep is a Border Collie.
2. A Border Collie is a Dog.
3. Dogs are Animals
4. A Border Collie is a Breed.
5. Dog is a Species

Whole-part Associations

- aggregation
- composition
Also called “has-a” relationships.
These associations can have multiplicities.
These whole-part associations cannot contain cycles (i.e., an object can never directly or indirectly be part of itself).
They are transitive.

Aggregation

Aggregation is the “part of” relation, in which one object consists of the associated objects.
If a complex entity is decomposed to reveal its pieces, e.g., a project team is made up of its members, we want to depict this type of relationship between compound class and component classes.
Examples of aggregation:
- computer and its peripherals

Composition

Composition is a particular kind of aggregation in which one component is physically part of another (single) component. The subcomponent dies if the whole component dies. The subcomponent can only be part of one aggregate at a time.
We would not want to use this notation for a team being composed of its team members, because it would imply that the members die when the team dies.
Examples of composition:
- a car is composed of an engine, wheels, a gear box, etc.
- a periodical is composed of articles; the articles don’t have a life outside of the periodical.
- a tree and its leaves
**Aggregation and Composition**

- Composition (physically part of)
- Aggregation (logically part of)

**Navigability**

Navigability: which object can call the operations of the other object (direction of flow of messages)

- **Bidirectional**
  - Company —> Person

- **Unidirectional**
  - Company —> Person

**Multiplicities**

Multiplicities give lower and upper bound on the number of objects that can be related to another object as a particular moment in time.

Examples:
- * (zero or more)
- 1..* (one or more)
- 1..40 (one to 40)
- 5 (exactly 5)

If no multiplicity is specified, it is undecided.

**Object Diagrams**

"An object diagram is snapshot of the objects of the system at a point in time." (Fowler)

It shows objects (instances of classes) and links (instances of association).

Object diagrams use the same syntax as UML class diagrams, except that object names are underlined (instance_name: class).

Values for attributes can also be shown.

If there are multiple objects of the same class, use object names to distinguish them.
Role Names for Associations

Association ends can be labelled with role names to describe how one object is viewed by the associated object.

Class Diagram:

Object Diagram showing management hierarchy:

Abstract Classes

Abstract classes do not have instances.

They show operations that subclasses must implement. These operations are written in italics.

The name of the abstract class is also written in italics.

Class Scope Attributes and Operations

Class scope attributes and operations are attributes and operations that are applied to a class of objects, rather than to individual objects.

Class scope attributes are used to model data values shared by all objects of the class, as opposed to the usual situation of each object of the class having its own value for that attribute.

Class scope operations are class-related operations not offered by instances of the class, e.g., create() and search() operations. (create is not usually shown on a conceptual model.)

Syntax: underlined attribute or operation

Class Scope Attributes and Operations

However, class-scope attributes and operations are rarely needed at the RE level of description.

Many such operations are implicit at the RE level (e.g., create/delete/find/count), so we don’t need to show them. They are given via associations.

An operation that is more involved than these simple operations, such as a “deleteUser” operation that has to check whether the user is involved in a call prior to deletion should not be a class-scope operation. Rather these should be operations of a “collection” (or controller) class that is in a 1:* relationship with the user class.

The behaviour this method would then be specified in the state diagram of this collection class, and during its behaviour it would call the “delete” method of the user account class.
Qualified Association

A qualified association is an association key that identifies the object at the other end of the association.

A qualifier is a key or index used to identify one or fewer objects from set of many objects.

Syntax: name in box at an end of an association

Often the qualifier is an attribute of the class at the other end of the association, an attribute that is recognized as uniquely identifying one or fewer objects of the class.

Summary: Concept Diagrams

- Classes
- Associations
- Attributes
- Operations
- Association classes
- Generalization, aggregation, composition
- Navigability
- Multiplicity

Good Analysis Classes

From Arlow and Neustadt:

- Name reflects intent
- Models problem domain (in vocabulary of problem domain)
- Small set of responsibilities
- High cohesion (responsibilities work towards same goal)
- Low coupling (few associations between classes)
- Keep it simple

Rules of Thumb

From Arlow and Neustadt:

- 3-5 responsibilities per class
- Every class has an association with at least one other class
- Beware of too many or too few classes
- Every class must have state! (Beware of “functoids”)
- Avoid deep inheritance.
- Avoid representing information that can be derived from other classes as a class
Project Concept Diagrams

The concept diagrams for your SRS should include:

- attributes and their types
- non-trivial operations and their types (set/get methods do not have to be listed)
- multiplicities on all associations (note: there are NO default multiplicities in UML 2 – a multiplicity of 1 must be explicitly stated)
- navigability on all associations – use the convention given in class or state otherwise
- names for all non-trivial associations

In a section following the concept diagram, list any preconditions and postconditions for the operations.

If the type information makes the diagram too large, this can be shown only in the section following the concept diagrams or in the glossary.

You should not show whether methods are public/private – all methods shown on a concept diagram should be public.

We recommend that you show actors on your concept diagrams. Actors can be shown as stick figures or as classes with the stereotype «actor» listed.

The value in showing actors is that it requires you to show multiplicities between actors and classes, which can be a valuable requirements detail.

Validation

- Follow the rules of thumb
- Evaluate cohesion and coupling of classes
- Include all responsibilities needed to implement system state diagram (more next class)

Communication Diagrams

- Form of event traces over an object diagram
- Depicts a scenario involving multiple objects
- Order of messages can be seen by explicit sequence numbers

A communication diagram is laid out on an object diagram and depicts specific objects and links between objects.

A communication is an annotation showing messages sent between objects, direction of messages, and sequence numbers of messages.

Messages are annotated with sequence numbers to show the order of events. The numbering can be simply 1, 2, 3, 4, ... However, to show decomposition of requests into subrequests and subresponses, messages can be annotated with Dewey-Decimal like sequence numbers.
**Communication Diagrams: Notation**

- boxes labelled by names of objects (underlined since these are instances)
- link lines showing communication paths
- messages label link lines; show sequence number, messages (possibly with parameters), arrow (for direction) of message
- messages have sequence numbers showing order; decomposition of requests into sub-requests can be shown with “1.1”, etc.
- conditional messages include labels 1 [cond] : msg
- iteration: 1 *[for i:=1..n]: msg(n)*
- may send messages to self

**Communication Diagram: Telephone**

**Sequence vs. Communication Diagrams**

<table>
<thead>
<tr>
<th>Sequence Diagram</th>
<th>Communication Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>depicts scenario</td>
<td>depicts scenario</td>
</tr>
<tr>
<td>emphasizes order of events</td>
<td>shows order of events but more difficult to see this order</td>
</tr>
<tr>
<td>can show real-time req</td>
<td>emphasizes links between objects</td>
</tr>
<tr>
<td>better to illustrate branching or iteration</td>
<td></td>
</tr>
</tbody>
</table>

---

U Waterloo SE1 (Winter 2006) – p.49/54

---

U Waterloo SE1 (Winter 2006) – p.50/54

---

U Waterloo SE1 (Winter 2006) – p.51/54

---

U Waterloo SE1 (Winter 2006) – p.52/54
Classes can be grouped into packages. The arrows show dependencies between packages.

Summary

- Conceptual class diagrams (domain models)
  - Classes
  - Associations
  - Attributes
  - Operations
  - Generalization, aggregation, composition
  - Navigability
  - Multiplicity
  - Object diagrams
- Communication Diagrams
- Package Diagrams

Warning! Don’t try and use all the features available to you in UML class diagrams.

Next Lecture: State diagrams for classes (stopwatch example)

Reading: none