TB141Ic – ICT System Engineering and Rapid Prototyping

System modeling with UML - Class and Sequence diagrams

Jacopo De Stefani

Delft University of Technology, The Netherlands

27/03/2023

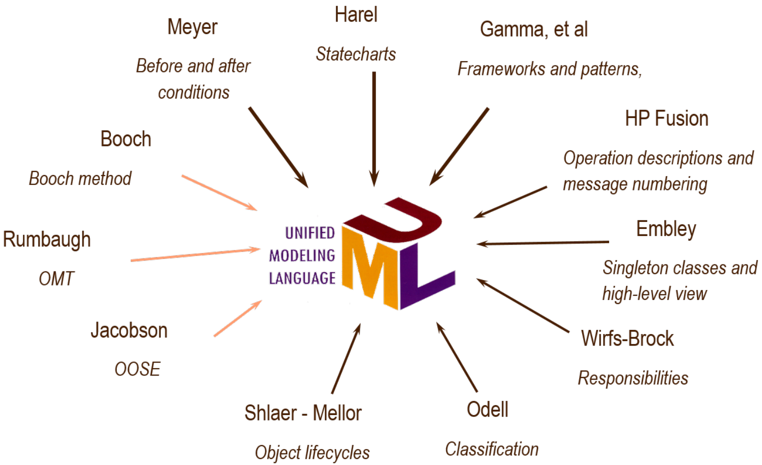
Licensing information

UML modeling

The tower of babel

- In 1994, more than 50 OO methods!!
- Fusion, Shlaer-Mellor, ROOM, Class-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS . . .
- ≪Meta-models≫ quite identical
- Graphical notations differ
- The process differs or remains vague
- But: Industry needs standards!

Unified Modeling Language



UML standard

- Designed by Booch, Rumbaugh, Jacobson (3 amigos)
  - Started in 1994; version 1.0 finished in 1997
  - Version 1.5 (1.4.2) since July 2004: current
  - Version 2.0 in beta since 2004 - final late 2005
- They put aside their own methods and notations to end the OO method wars
- Lack of standardization
- Has become the de facto standard

General goals of UML

- Model systems using OO concepts
- Establish an explicit coupling to conceptual as well as executable artifacts
- To create a modeling language usable by both humans and machines
- Models different types of systems (information systems, technical systems, embedded systems, real-time systems, distributed systems, system software, business systems, UML itself, ...)
- Two main concepts:
  - Views
  - Diagrams

UML Views

- Each view is a projection of the complete system.
- Each view highlights particular aspects of the system.
- Views are described by a number of diagrams.
- No strict separation, so a diagram can be part of more than one view.

UML Diagrams

- Use-Case diagram
- Class diagram
- Object diagram
- State diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Component diagram
- Deployment diagram

UML Diagrams

- **Use-Case diagram**
- **Class diagram**
- Object diagram
- State diagram
- **Sequence diagram**
- Collaboration diagram
- **Activity diagram**
- Component diagram
- Deployment diagram

UML Views - Diagrams mapping

Use case view

Shows the functionality of the system as perceived by external actors.

Diagrams	Used by
Use case	Customers
Activity	Designers
	Developers
	Testers

Component View

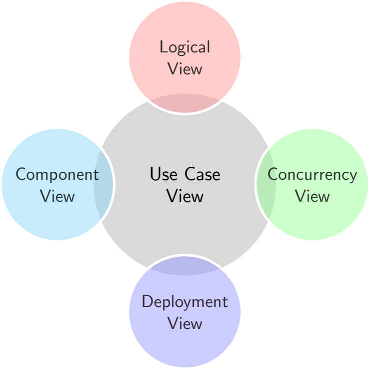
Shows the organization of the code components and their dependencies.

Diagrams	Used by
Component	Developers

Deployment View

Shows the deployment of the system into the physical architecture.

Diagrams	Used by
Deployment	Developers
	Testers
	System
	Integrators



Logical view

Shows how the functionality of the system is designed / provided.

Diagrams	Used by
Class	Developers
State	Designers
Sequence	
Collaboration	
Activity	

Concurrency view

Addresses the problems with communication and synchronization for a concurrent system.

Diagrams	Used by
State	Developers
Sequence	System
Collaboration	Integrators
Activity	Testers
Deployment	
Components	

Class Diagrams

UML Class Diagram

- Static model type
  - A view of the system in terms of classes (entities) and relationships
- Two main variants:
  - **Domain modeling:** High level, focused on entities and relationships
  - **Implementation modeling:** Low level, focused on data types, visibilities, implementation choices (abstract classes vs interfaces)
- **Classes**
- **Objects**

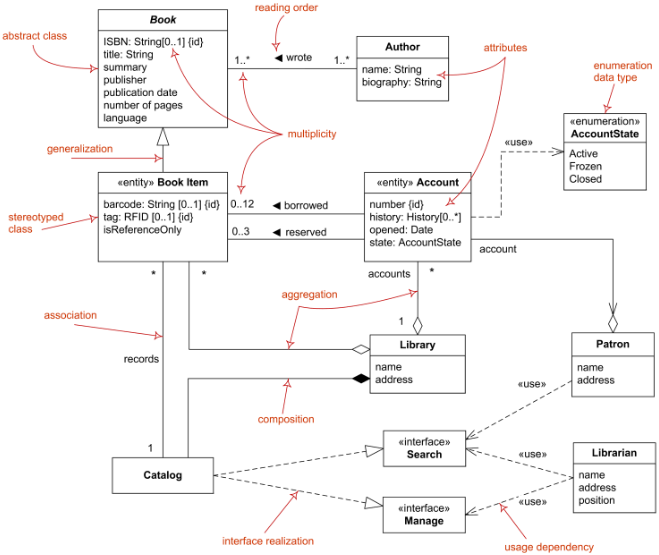
UML Class Diagram

- Static model type
  - A view of the system in terms of classes (entities) and relationships
- Two main variants:
  - **Domain modeling:** High level, focused on entities and relationships
  - **Implementation modeling:** Low level, focused on data types, visibilities, implementation choices (abstract classes vs interfaces)
- **Classes** → Blueprint/Recipe including common attributes **and** behavior
- **Objects**

UML Class Diagram

- Static model type
  - A view of the system in terms of classes (entities) and relationships
- Two main variants:
  - **Domain modeling:** High level, focused on entities and relationships
  - **Implementation modeling:** Low level, focused on data types, visibilities, implementation choices (abstract classes vs interfaces)
- **Classes** → Blueprint/Recipe including common attributes **and** behavior
- **Objects** → Concrete realization (instance) of a class including specific values for attributes.

Class diagram - Overview



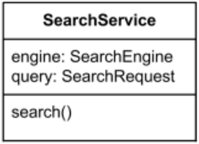
Class Diagram - Elements

Class - No compartments

A class is a classifier which describes a set of objects that share the same:

- features
- constraints
- semantics (meaning).

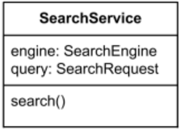
A class is shown as a solid-outline rectangle containing the class name, and optionally with compartments separated by horizontal lines containing features or other members of the classifier.



Class - With compartments

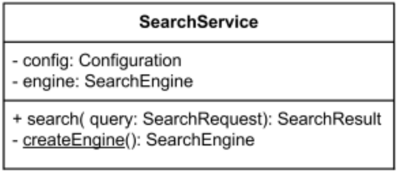
When class is shown with three compartments, the middle compartment holds a list of attributes (information that the class stores) and the bottom compartment holds a list of operations (or methods). Attributes and operations should be left justified in plain face, with the first letter of the names in lower case.

Class Diagram - Design vs Implementation



Class - Design level

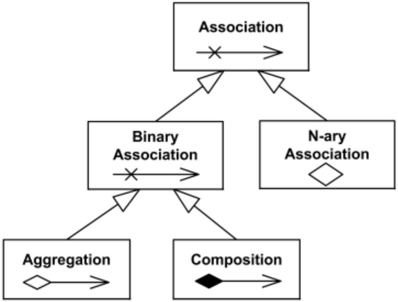
At the design level, the class contains general specifications of the attributes and operations of the class.

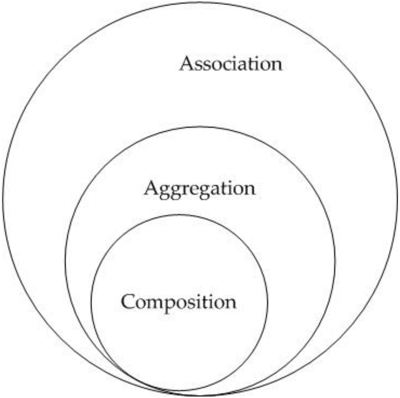


Class - Implementation level

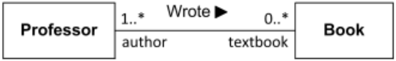
At the implementation level, the class contains precise specifications of the data type of the attributes, their visibility and the details of input operations of the class.

Class diagram - Relationship Overview





Class diagram - Relationships - Association



Binary Association

Association is a relationship between classifiers which is used to show that instances of classifiers could be either linked to each other or combined logically or physically into some aggregation.
Binary association relates two typed instances. It is normally rendered as a solid line connecting two classifiers, or a solid line connecting a single classifier to itself (the two ends are distinct). The line may consist of one or more connected segments.

A small solid triangle could be placed next to or in place of the name of binary association (drawn as a solid line) to show the order of the ends of the association. The arrow points along the line in the direction of the last end in the order of the association ends. This notation also indicates that the association is to be read from the first end to the last end.

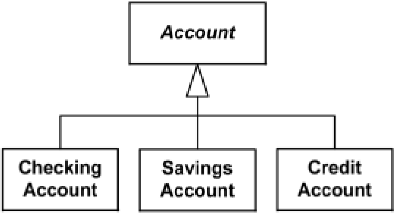
Multiplicity

Multiplicity is a definition of an inclusive interval of non-negative integers to specify the allowable number of instances of described element.

Some typical examples of multiplicity bounds:

0	Collection must be empty
1	Exactly one instance
5	Exactly 5 instances
$*$	Zero or more instances
0..1	No instances or one instance
1..1	Exactly one instance
$0..*$	Zero or more instances
$1..*$	At least one instance
$m..n$	At least m but no more than n instances

Class diagram - Relationships - Generalization



Generalization

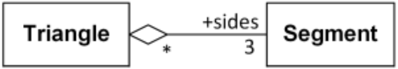
A generalization is a binary taxonomic (i.e. related to classification) directed relationship between a more general classifier (superclass) and a more specific classifier (subclass).

Each instance of the specific classifier is also an indirect instance of the general classifier, so that we can say "Patient is a Person", "Savings account is an Account", etc. Because of this, generalization relationship is also informally called "Is A" relationship.

Semantics

- Attributes and methods from the superclass are inherited in the subclass ⇒ They are included in the subclass implicitly.
- Generalization doesn't have multiplicity
- In implementation, notion of single versus multiple inheritance.

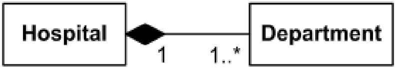
Class diagram - Relationships - Aggregation/Composition



Aggregation - Shared Aggregation

Shared aggregation (aggregation) is a binary association between a property and one or more composite objects which group together a set of instances. It is a "weak" form of aggregation when part instance is independent of the composite. Shared aggregation has the following characteristics:

- it is binary association,
- it is asymmetric - only one end of association can be an aggregation,
- it is transitive - aggregation links should form a directed, acyclic graph, so that no composite instance could be indirect part of itself,
- shared part could be included in several composites, and if some or all of the composites are deleted, shared part may still exist.



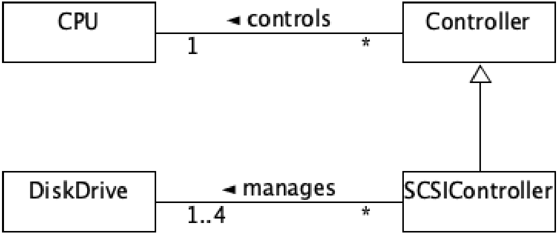
Composition - Composite aggregation

Composite aggregation (composition) is a "strong" form of aggregation with the following characteristics:

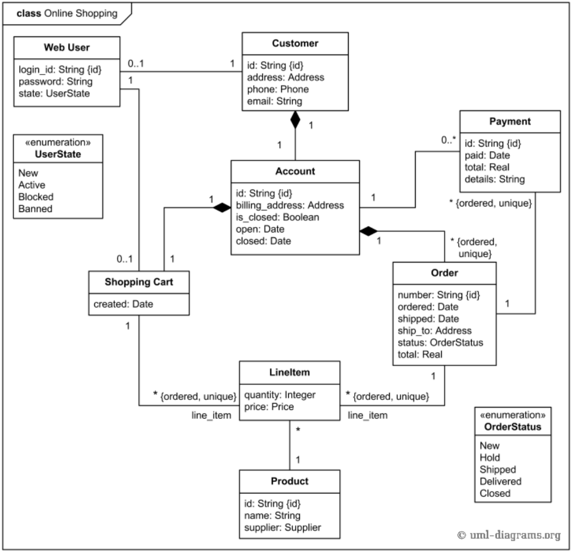
- it is binary association,
- it is a whole/part relationship,
- a part could be included in at most one composite (whole) at a time, and
- if a composite (whole) is deleted, all of its composite parts are "normally" deleted with it.

Note, that UML does not define how, when and specific order in which parts of the composite are created. Also, in some cases a part can be removed from a composite before the composite is deleted, and so is not necessarily deleted as part of the composite.

Class diagram - Example - Reading - 1



Class diagram - Example - Reading - 2

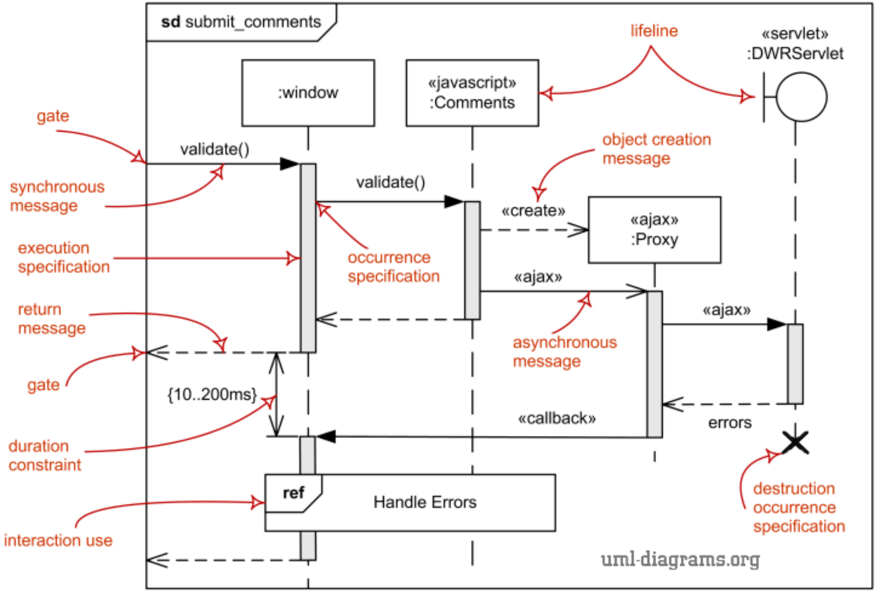


Sequence Diagrams

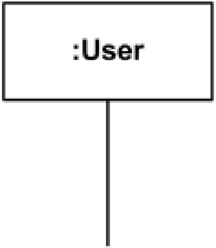
Sequence Diagrams

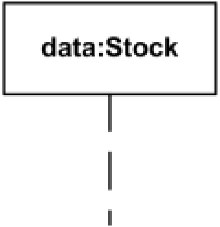
- Can be used to support interaction modeling (within the system)
- Focus on flow of events/objects/activities
- Allows to specify sequence/parallelism
- Support the visualization of information sharing across components and synchronous/asynchronous interactions

Sequence diagram - Overview



Sequence diagram - Elements





Lifeline - No name

Lifeline is a named element which represents an individual participant in the interaction. While parts and structural features may have multiplicity greater than 1, lifelines represent only one interacting entity.

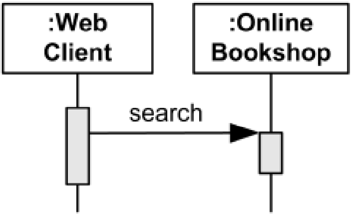
A lifeline is shown using a symbol that consists of a rectangle forming its "head" followed by a vertical line (which may be dashed) that represents the lifetime of the participant.

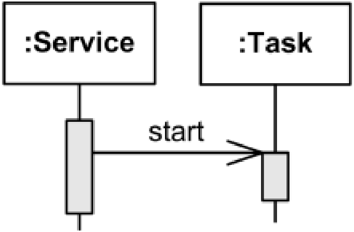
Lifeline - Named

While an unnamed lifeline refers to a generic class element, a named lifeline refers to a specific instance of the class indicated in the rectangle.

This notation is employed when the diagram needs to represent multiple instances of the same class interacting.

Sequence diagram - Messages





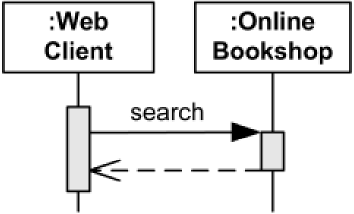
Synchronous message

Synchronous message typically represents operation call - send message and suspend execution while waiting for response. Synchronous call messages are shown with filled arrow head.

Asynchronous message

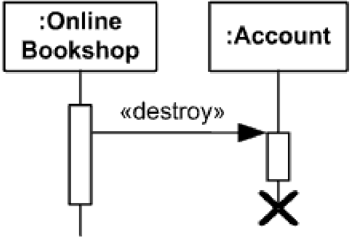
Asynchronous message - send message and proceed immediately without waiting for return value. Asynchronous messages have an open arrow head.

Sequence diagram - Message



Reply message

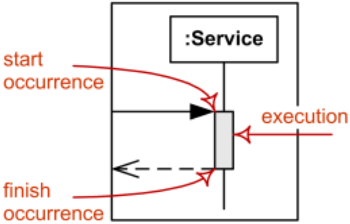
Reply message to an operation call is shown as a dashed line with open arrow head (looks similar to creation message).



Delete message

Delete message (called stop in previous versions of UML) is sent to terminate another lifeline. The lifeline usually ends with a cross in the form of an X at the bottom denoting destruction occurrence.

Sequence diagram - Execution



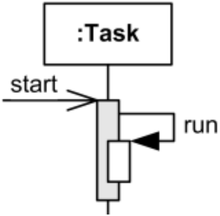
Execution

Execution (full name - execution specification, informally called activation) is interaction fragment which represents a period in the participant's lifetime when it is

- executing a unit of behavior or action within the lifeline,
- sending a signal to another participant,
- waiting for a reply message from another participant.

Note, that the execution specification includes the cases when behavior is not active, but just waiting for reply. The duration of an execution is represented by two execution occurrences - the start occurrence and the finish occurrence.

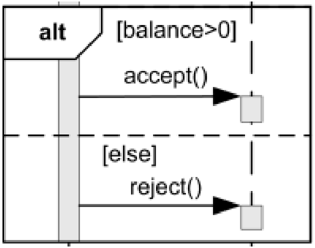
Execution is represented as a thin grey or white rectangle on the lifeline.



Overlapping execution

Overlapping execution specifications on the same lifeline are represented by overlapping rectangles.

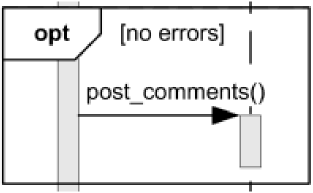
Sequence diagram - Operators



Alternative operator

The interaction operator alt means that the combined fragment represents a choice or alternatives of behavior. At most one of the operands will be chosen. The chosen operand must have an explicit or implicit guard expression that evaluates to true at this point in the interaction.
An implicit true guard is implied if the operand has no guard.

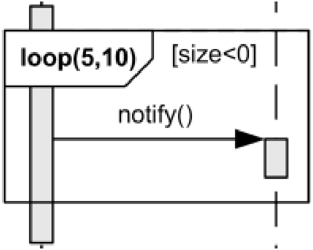
An operand guarded by else means a guard that is the negation of the disjunction of all other guards. If none of the operands has a guard that evaluates to true, none of the operands are executed and the remainder of



Option operator

The interaction operator opt means that the combined fragment represents a choice of behavior where either the (sole) operand happens or nothing happens. An option is semantically equivalent to an alternative combined fragment where there is one operand with non-empty content and the second operand is empty.

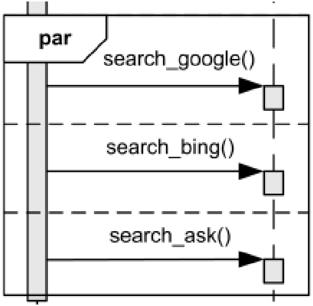
Sequence diagram - Operators



Loop operator

The interaction operator loop means that the combined fragment represents a loop. The loop operand will be repeated a number of times. The loop construct represents a recursive application of the seq operator where the loop operand is sequenced after the result of earlier iterations.
Loop could be controlled by either or both iteration bounds and a guard.

Loop operand could have iteration bounds which may include a lower and an upper number of iterations of the loop.

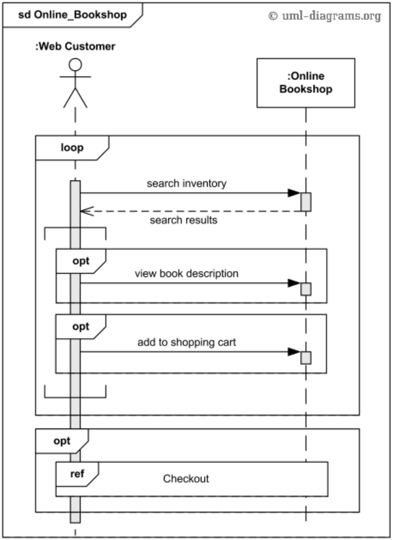


Parallel operator

The interaction operator par defines potentially parallel execution of behaviors of the operands of the combined fragment. Different operands can be interleaved in any way as long as the ordering imposed by each operand is preserved.

Set of traces of the parallel operator describes all the possible ways or combinations that occurrence specifications of the operands may be interleaved without changing the order within each operand.

Sequence diagram - Example - Reading - 1



Sequence diagram - Example - Reading - 2