MINES Saint-Étienne | Institut Mines-Télécom

Object Oriented Modeling

The Unified Modeling Language (UML)

Luis Gustavo Nardin
gnardin@emse.fr

ICM – Computer Science Major | CPS2 Engineering and Development

September 25, 2024

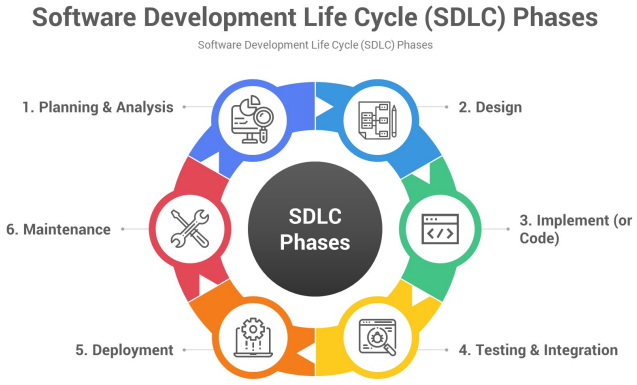
Based on the slides of Maxime Lefrançois – Software Engineering course

Outline

Software Development Life Cycle

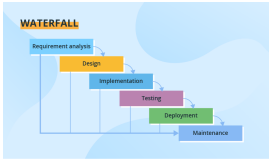
Software Development Life Cycle

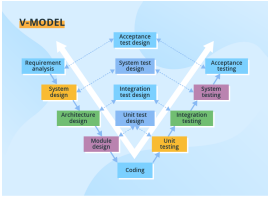
1. Planning and Analysis
2. Design
3. Implement
4. Testing and Integration
5. Deployment
6. Maintenance

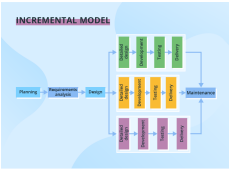


Source: slidesalad

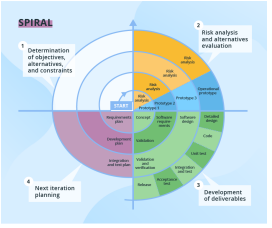
SDLC Models


WATERFALL
Requirement analysis
Design
Implementation
Testing
Deployment
Maintenance

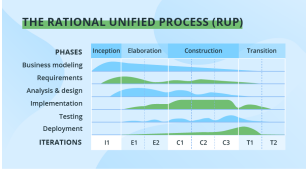

V-MODEL
Requirement analysis
System design
Architecture design
Module design
Coding
Unit test design
Integration test design
System test design
Acceptance test design
Unit testing
Integration testing
System testing
Acceptance testing

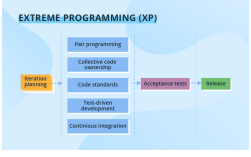

INCREMENTAL MODEL
Planning
Requirements analysis
Design
Maintenance

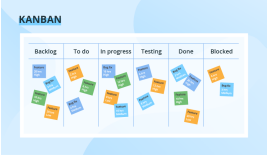

ITERATIVE MODEL
Planning
Requirements analysis
Design


SPIRAL
1 Determination of objectives, alternatives, and constraints
2 Risk analysis and alternatives evaluation
3 Development of deliverables
4 Next iteration planning


SCRUM
SPRINT 1
2-4 WEEKS
SPRINT 2
SPRINT 3
Plan
Review
Retrospect
Daily Scrum Meeting
2-4 WEEKS

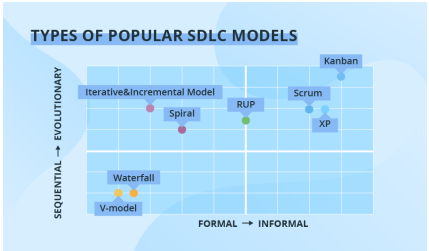

THE RATIONAL UNIFIED PROCESS (RUP)
PHASES
Inception
Elaboration
Construction
Transition
Business modeling
Requirements
Analysis & design
Implementation
Testing
Deployment
ITERATIONS
I1
E1
E2
C1
C2
C3
T1
T2


EXTREME PROGRAMMING (XP)
Iteration planning
Pair programming
Collective code ownership
Code standards
Test-driven development
Continious integration
Acceptance tests
Release


KANBAN
Backlog
To do
In progress
Testing
Done
Blocked

SDLC Models

- *Software development processes* distinguish the development stages in the software life cycle
- Use a *representation formalism* that facilitates communication, organization and verification
- Produce a *set of artifacts* that facilitate design feedback and application evolution



Source: ScienceSoft

Software Development Methods

- Hierarchical functional methods
  - Data-Flow/SADT/SA-SD, Structure-Chart, ...
- Data oriented methods
  - Entity-Relation, MERISE, ...
- Behavior oriented methods
  - SA-RT, Petri Net, ...
- Object oriented methods
  - OMT, OOA, Classe-Relation, OOD, ...

Object-Oriented Software Development Method

At the beginning of the 90's

- There were about 50 object oriented methods
- Related only by their *consensus on common ideas* (object, class, . . .)
- Each with its own notation, yet *not able to fulfill the needs* and *correctly represent systems* in all application domains

Single Common Language

- Usable by any Object-Oriented method
- Cover all software development life cycle phases
- Compatible with contemporary production techniques

UML (Unified Modeling Language)

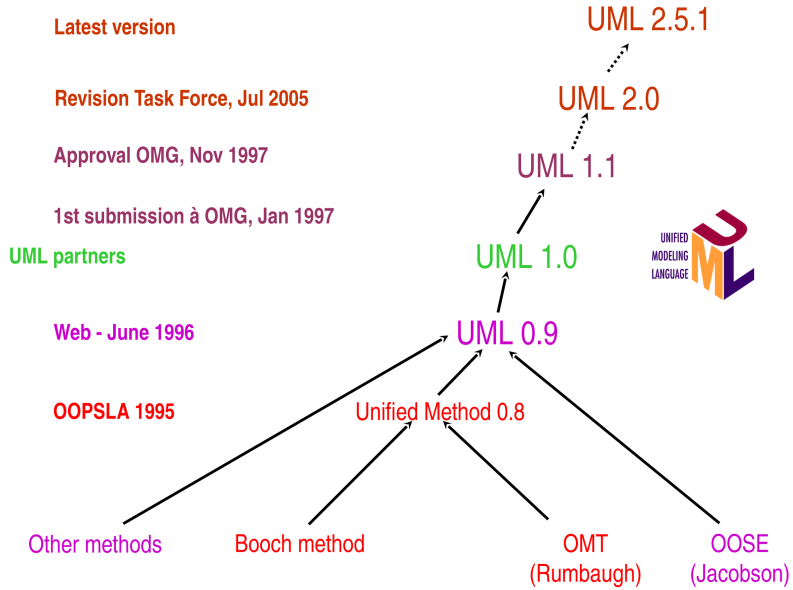
Overview

- The UML is a *general-purpose visual modeling language* intended to provide a standard way to *visualize*, *specify*, *construct* and *document* the artifacts of a software-intensive system
- The UML is intentionally *process independent* and could be applied in the context of different processes, but better suited to a software development process that is
  - Use case driven
  - Architecture centric
  - Iterative and incremental
- The UML is *semi-formal* (i.e., provide a reasonably well defined meaning to each construct)

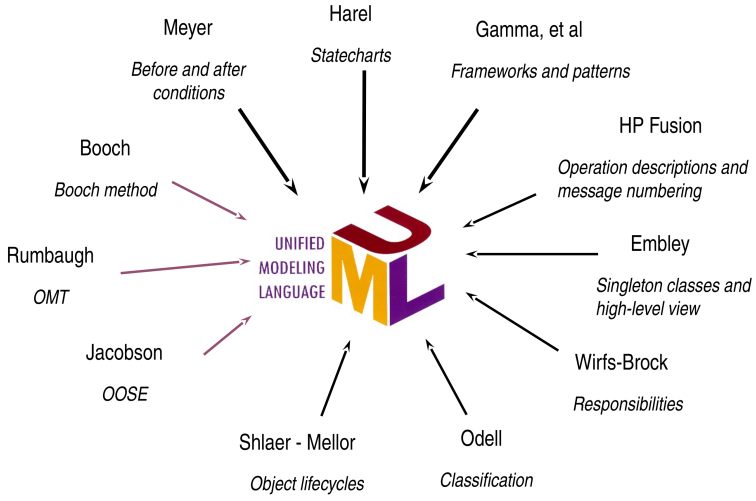
Origin

- Based on
  - *OMT notations* (J. Rumbaugh) for the analysis and design of data-based information systems
  - *Booch's method notations* (G. Booch) for the design and implementation phases
  - *OOSE notations* (I. Jacobson) for requirement analysis through "use cases"
- Proposed
  - Standardized development artifacts (models, notation, diagrams) without standardizing the development process
- Important role played by RATIONAL and OMG (http://www.omg.org)

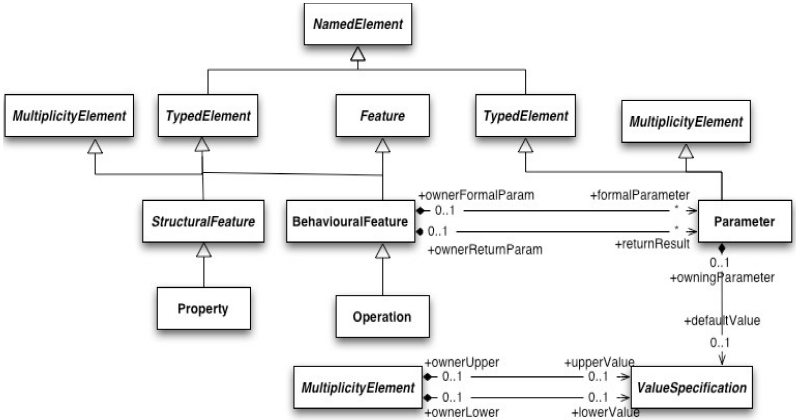
Evolution


Latest version
UML 2.5.1
Revision Task Force, Jul 2005
UML 2.0
Approval OMG, Nov 1997
UML 1.1
1st submission à OMG, Jan 1997
UML partners
UML 1.0
UNIFIED
MODELING
LANGUAGE
Web - June 1996
UML 0.9
OOPSLA 1995
Unified Method 0.8
Other methods
Booch method
OMT
(Rumbaugh)
OOSE
(Jacobson)

Contributions to UML 1.X

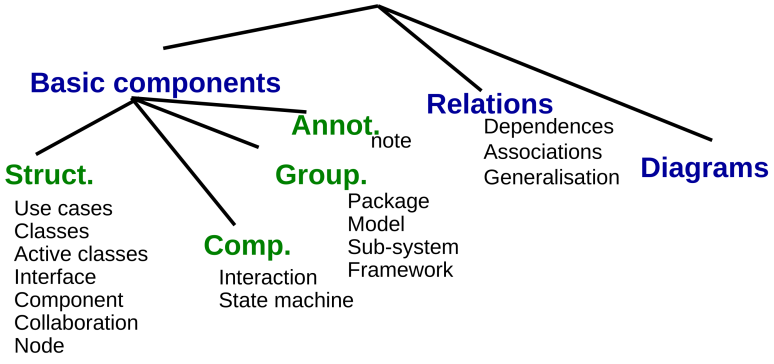


UML Meta-Model



Based on Martin Fowler UML Distilled and Viviane Jonckers OOSD-UML course

UML Vocabulary



UML Diagrams

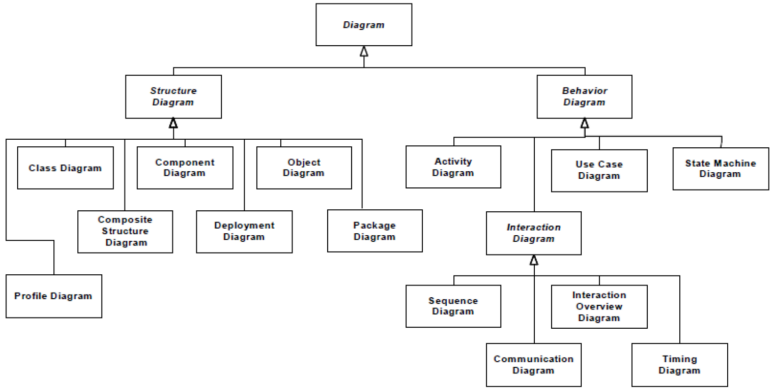
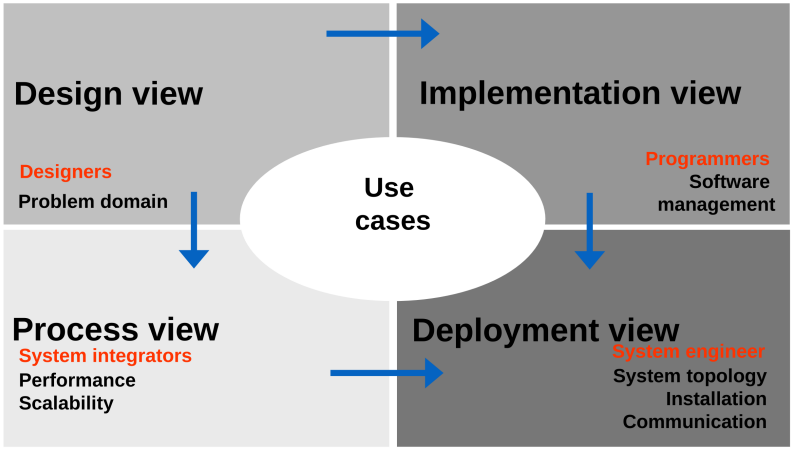


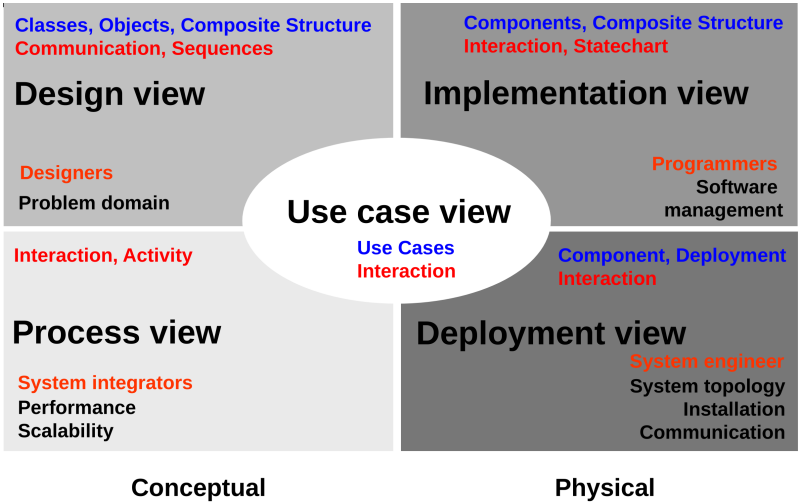
Figure A.5 The taxonomy of structure and behavior diagrams

Source: UML Specification v2.5.1, p. 727 (https://www.omg.org/spec/UML/2.5.1/PDF)

Views on the Software



Diagrams within Views on the Software



Rules of Thumb

- Nearly everything in UML is optional
- The UML provides a language to capture information that varies greatly depending on the domain of the problem
- Parts of UML either do not apply to your particular problem or may not lend anything to the particular view you are trying to convey
- You do not need to use every part of UML in every model you create
- You do not need to use every allowable symbol for a diagram type in every diagram you create
- Use only what helps clarify the message you are trying to convey

Use Case Diagram

Use Case Diagram

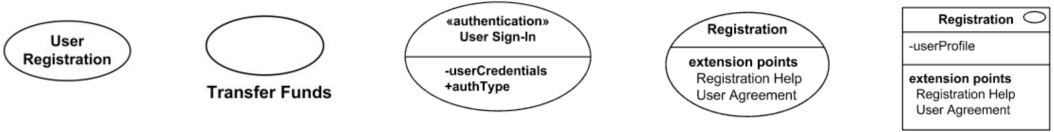
Definition

Describes a set of actions (*use cases*) that some system or systems (*subject*) should or can perform in collaboration with one or more external users of the system (*actors*) to provide some observable and valuable results to the actors or other stakeholders of the system(s).

Use Case

Definition

A *use case* is a kind of *behaviored classifier* that specifies a [complete] unit of [useful] functionality performed by [one or more] *subjects* to which the *use case applies* in collaboration with one or more *actors*, and which [for complete use cases] yields an observable result that is of some value to those actors [or other stakeholders] of each subject.



Actor

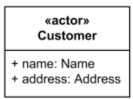
Definition

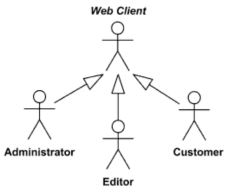
An *actor* is *behaviored classifier* which specifies a *role* played by an *external entity* that interacts with the *subject* (e.g., by exchanging signals and data), a human user of the designed system, some other system or hardware using services of the subject.

Student

Web Client

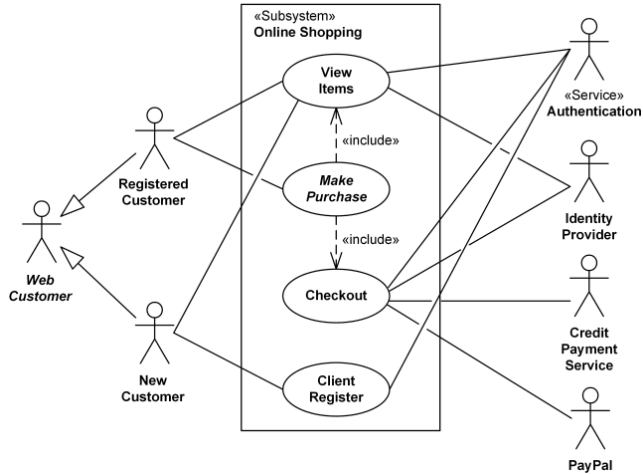
Bank





Example: Online Shopping

- Web Customer actor uses some web site to make purchases online. Top level use cases are View Items, Make Purchase and Client Register.



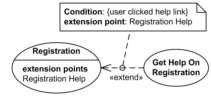
Decomposition

«include» stereotype

- The included use case is a mandatory part of the including one

«extend» stereotype

- The extending use case augments the functionality of the extended one

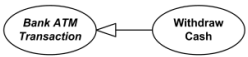


Generalization

- Generalization between use cases is similar to generalization between classes

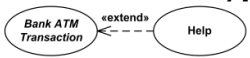
Decomposition

Generalization



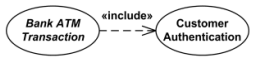
- Base use case could be abstract use case (incomplete) or concrete (complete)
- Specialized use case is required, not optional, if base use case is abstract
- No explicit location to use specialization
- No explicit condition to use specialization

«extend» stereotype



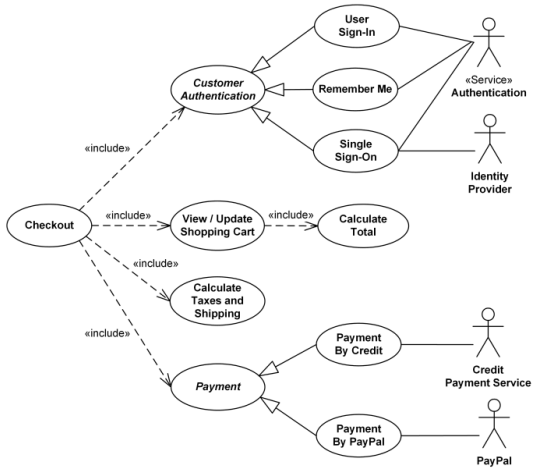
- Base use case is complete (concrete) by itself, defined independently
- Extending use case is optional, supplementary
- Has at least one explicit extension location
- Could have optional extension condition

«include» stereotype



- Base use case is incomplete (abstract use case)
- Included use case required, not optional
- No explicit inclusion location but is included at some location
- No explicit inclusion condition

Example: Online Shopping



Describe Use Case Behavior

Use case *behaviors* may be described in a natural language text (opaque behavior), which is current common practice, or by using UML *behavior diagrams* for specific behaviors such as

- Activity
- State Machine
- Interaction

Activity Diagram

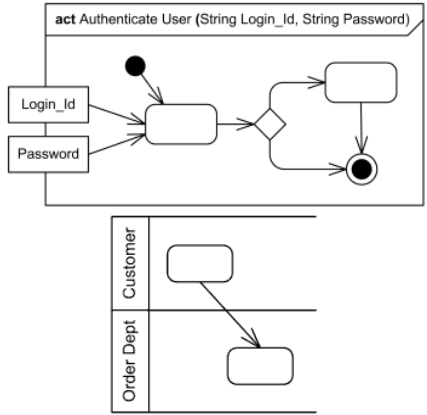
Activity Diagram

Description

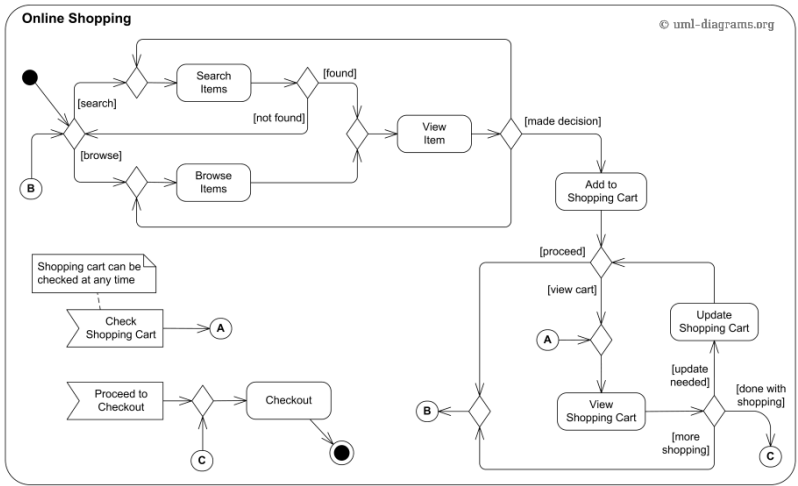
Activity diagram is UML behavior diagram which shows *flow of control* or *object flow* with emphasis on the sequence and conditions of the flow. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because some events external to the flow occur.

Activity

- Activity nodes
  - Action
  - Control
  - Object
- Actions of various kinds
  - Occurrences of primitive functions
  - Invocations of behavior
  - Communication actions
  - Manipulations of objects



Example: Online Shopping



State Machine Diagram

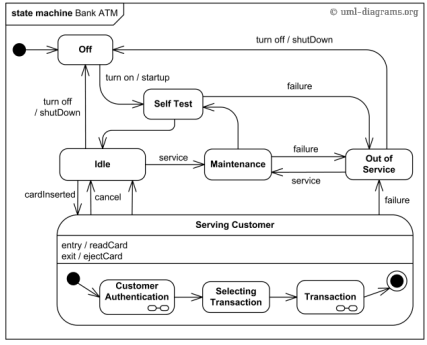
State Machine Diagram

Definition

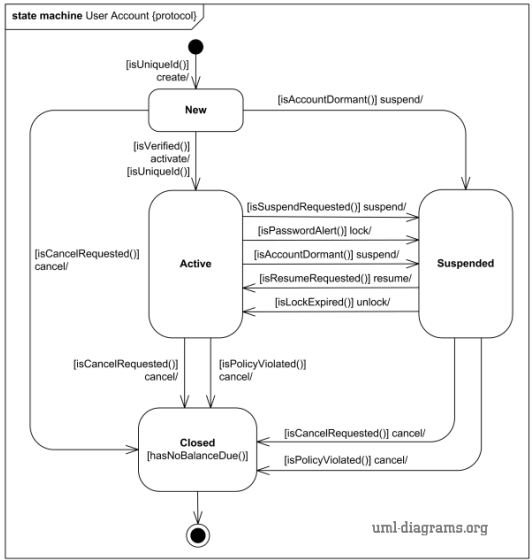
State machine diagram is a behavior diagram which shows *discrete behavior* of a part of designed system through *finite state transitions*. State machine diagrams can also be used to express the usage protocol of part of a system.

State Machine Diagram

- Behavior is modeled as a traversal of a graph of state nodes connected with transitions
- Transitions are triggered by the dispatching of series of events
- During the traversal, the state machine could also execute some activities



Example: Online Shopping User Account



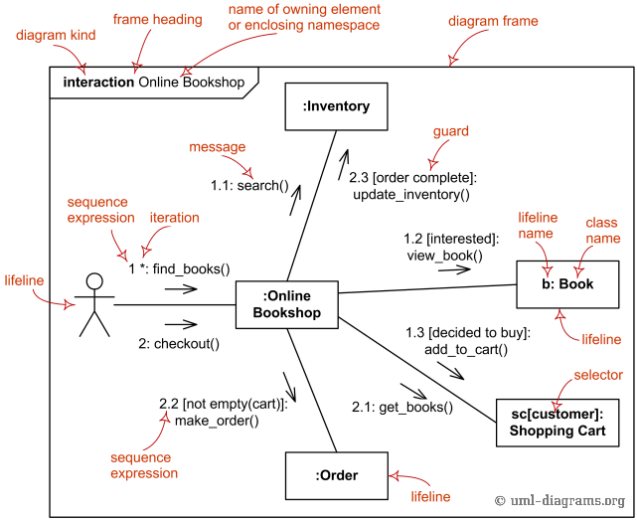
Communication Diagram

Communication Diagram

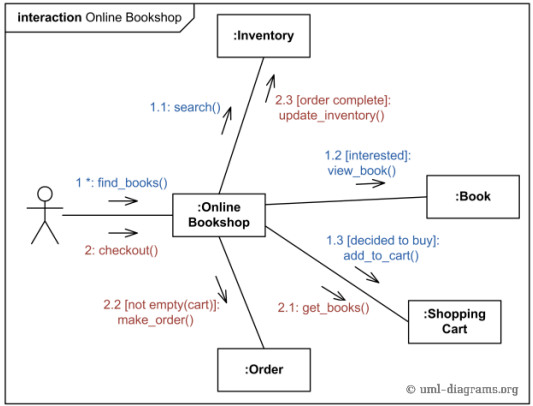
Definition

Communication diagram (called collaboration diagram in UML 1.x) is a kind of UML interaction diagram which shows interactions between objects and/or parts (represented as lifelines) using sequenced messages in a free-form arrangement.

Communication Diagram



Example: Online Shopping



Class Diagram

Class Diagram

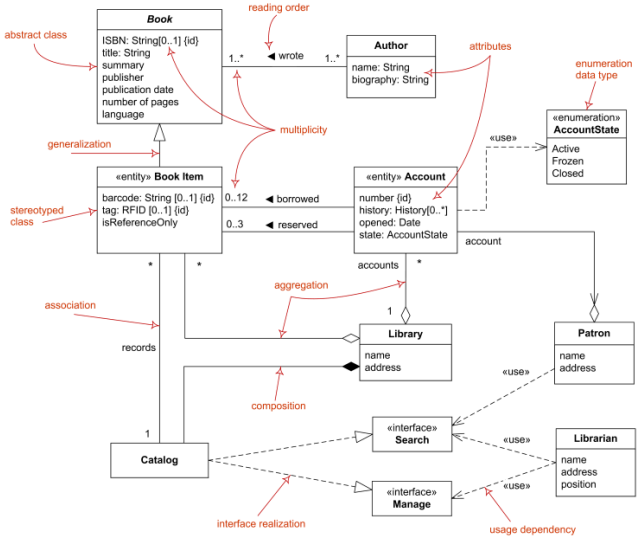
Definition

Class diagram is UML structure diagram which shows structure of the designed system at the level of classes and interfaces, shows their features, constraints and relationships - associations, generalizations, dependencies, etc.

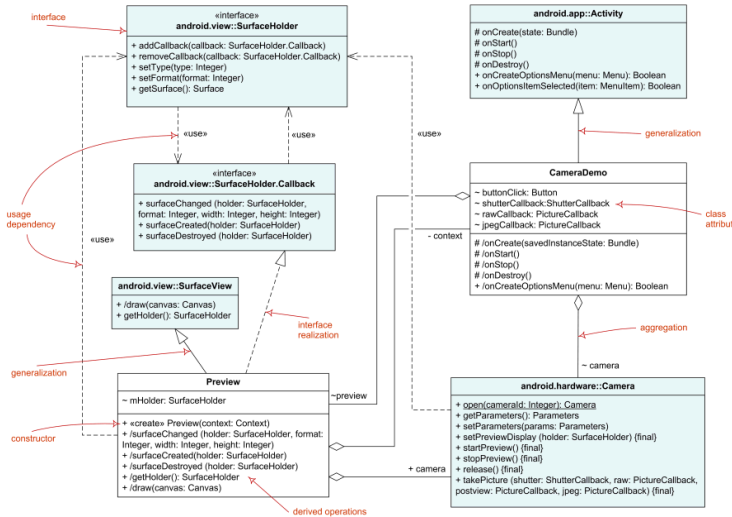
Types of Class Diagrams

- Domain model diagram
- Implementation classes diagram

Domain Model Diagram

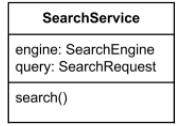


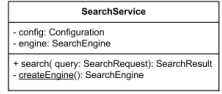
Implementation Classes Diagram



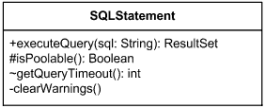
Classes

- Classes are specifications for objects
- Consist of (in the main)
  - A name
  - A set of attributes (aka fields)
  - A set of operations
    - ✓ Constructors: initialize the object state
    - ✓ Accessors: report on the object state
    - ✓ Mutators: alter the object state
    - ✓ Destructors: clean up

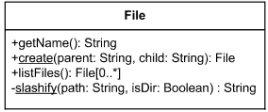




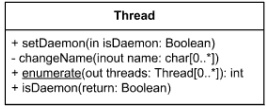
Operations



`executeQuery` is `public`, `isPoolable` – `protected`, `getQueryTimeout` – with package visibility, and `clearWarnings` is private

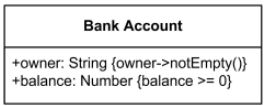


File has two static operations - `create` and `slashify`. `create` has two parameters and returns `File`. `Slashify` is private operation. Operation `listFiles` returns array of files. Operations `getName` and `listFiles` either have no parameters or parameters were suppressed.

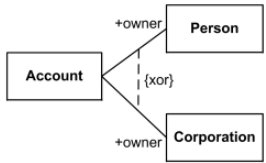


Operation `setDaemon` has one input parameter, while single parameter of `changeName` is both input and output parameter. Static `enumerate` returns integer result while also having output parameter – array of threads. Operation `isDaemon` is shown with return type parameter.

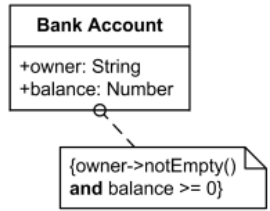
Constraints



Non empty owner and positive balance

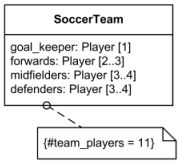


Account owner is either Person or Corporation



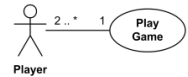
Non empty owner and positive balance

Multiplicity



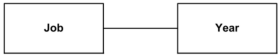
Multiplicity of Players for Soccer Team class

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

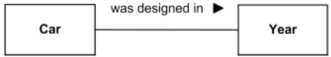


Two or more Player actors are required to initiate Play Game use case

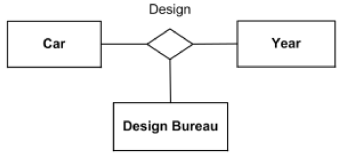
Associations



Association

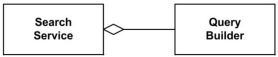


Order of the ends and reading: Car - was designed in - Year



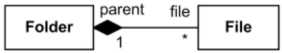
Ternary association Design relating three classifiers.

Aggregation/composition Ownership



Aggregation

Association end qb is an attribute of SearchService class and is owned by the class.



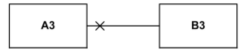
Composite Aggregation (= composition)
If folder is deleted, all files are deleted as well

Association qualifier

Given a company and a social security number (SSN) at most one employee could be found.

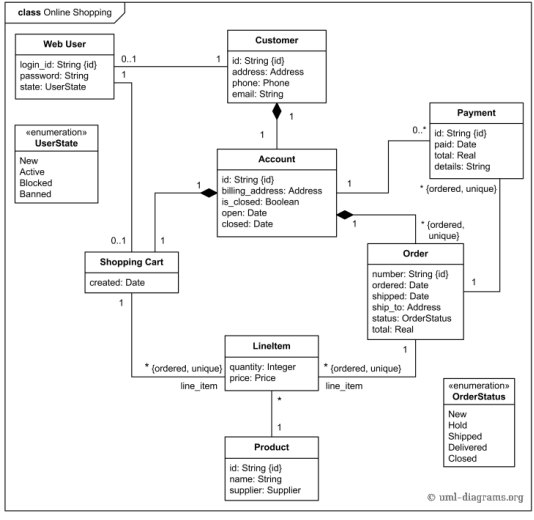
Navigability

A2 has unspecified navigability while B2 is navigable from A2.



A3 is not navigable from B3 while B3 has unspecified navigability.

Example: Online Shopping



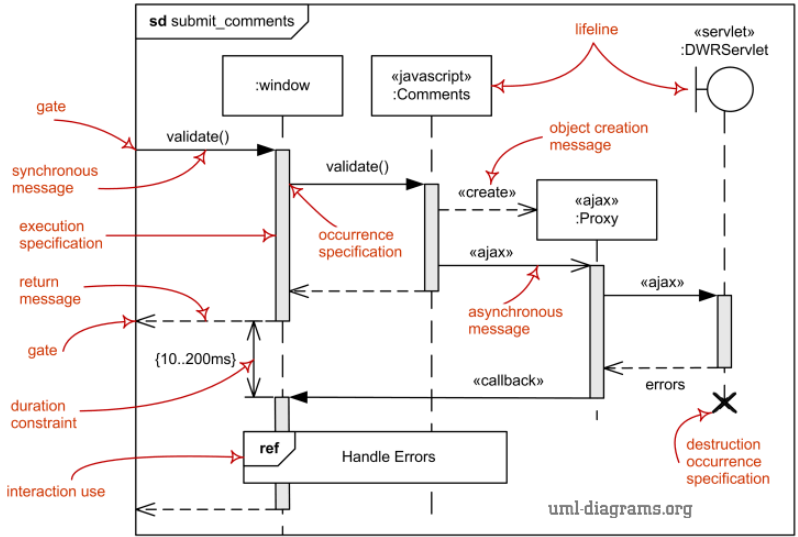
Sequence Diagram

Sequence Diagram

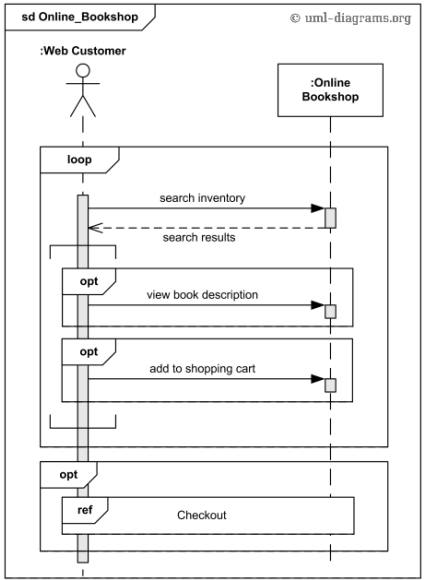
Definition

- Sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between a number of lifelines
- Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines

Sequence Diagram



Example: Online Shopping



Deployment Diagram

Deployment Diagram

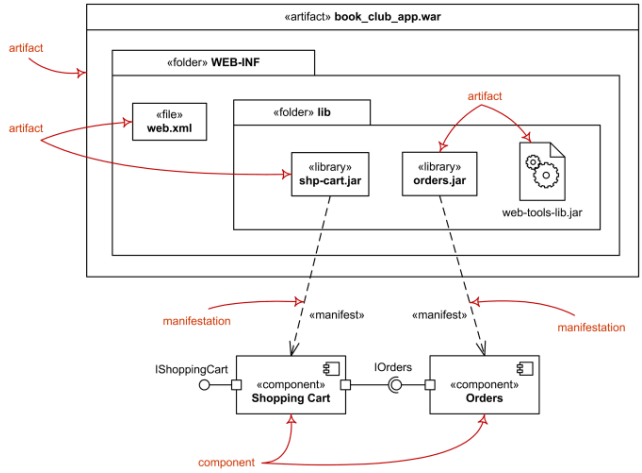
Definition

Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets.

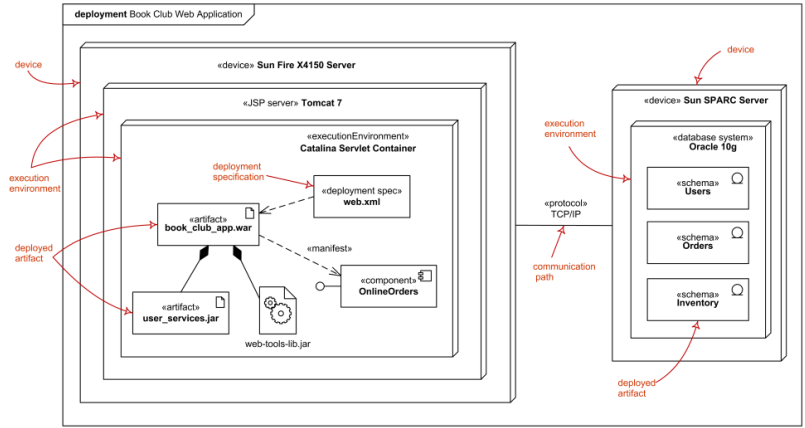
Types of Deployment Diagrams

- Manifestation of components by artifacts
- Specification level deployment diagram
- Instance level deployment diagram
- Specification level network architecture

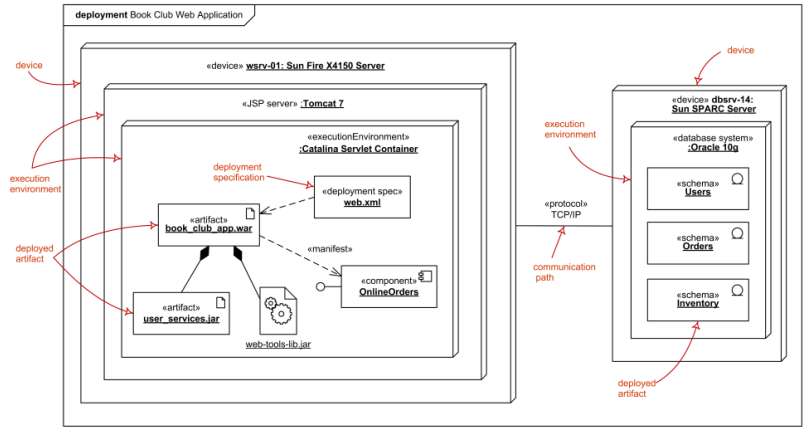
Manifestation of Components by Artifacts



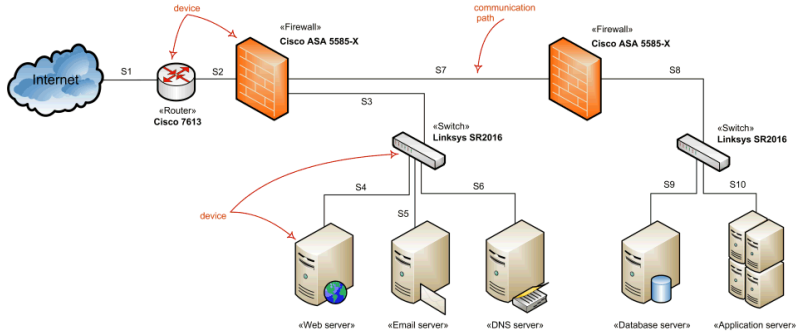
Specification Level Deployment Diagram



Instance Level Deployment Diagram



Specification Level Network Architecture



References

- Fakhroutdinov, K. (2024). The Unified Modeling Language. https://www.uml-diagrams.org
- OMG UML. (2017). OMG™ Unified Modeling Language™. https://www.omg.org/spec/UML/2.5.1