Software Engineering

Part 6 – The Unified Modeling Language

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

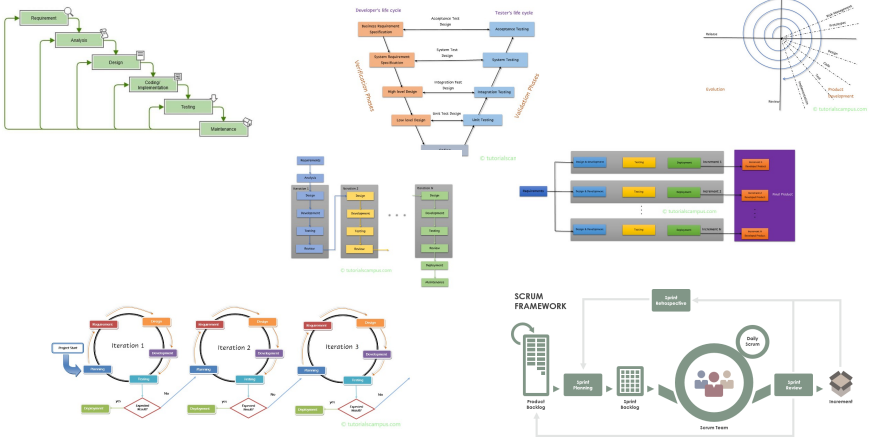
Software Engineering

Part 6 – The Unified Modeling Language

6.1 Introduction

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Software Development Methods



Software Development Methods

- *processes* that distinguish development stages in the software life cycle. Should:
  - be modular, reduce complexity, reuseable, at the right level of abstraction
- Using a *representation formalism* that facilitates communication, organization and verification
- Production of a set of *artifacts* that facilitate design feedback and application evolution
  - documents, models, prototypes

Existing software Development Methods

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

Object-oriented SD methods

- Statement:
  - at the beginning of the 90's, there are about 50 object oriented methods,
  - linked only by a consensus around common ideas (object, class, subsystems, ...)
  - BUT each with its own notation,
  - WITHOUT being able to fulfill all the needs and to correctly model the various fields of application.
- **Definition of a single common language**
  - usable by any object method,
  - in all phases of the life cycle,
  - compatible with current production techniques.

  → **UML**
- **Definition a common unified development process**

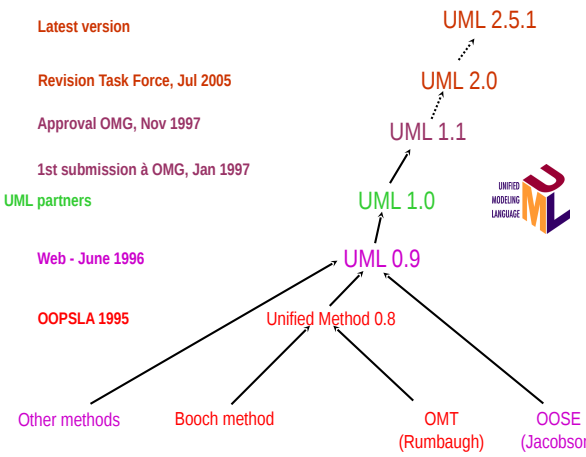
  → Unified Process (obsolete, use Scrum or other more recent processes)

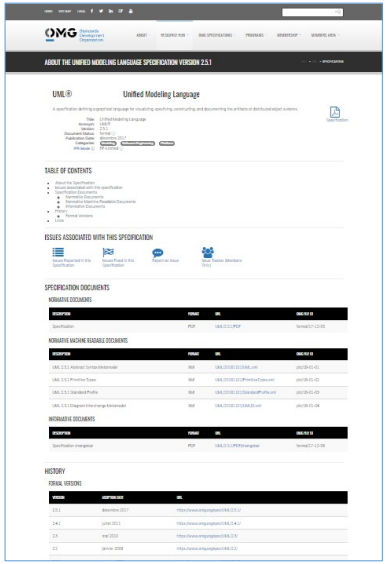
UML (Unified Modeling Language)

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

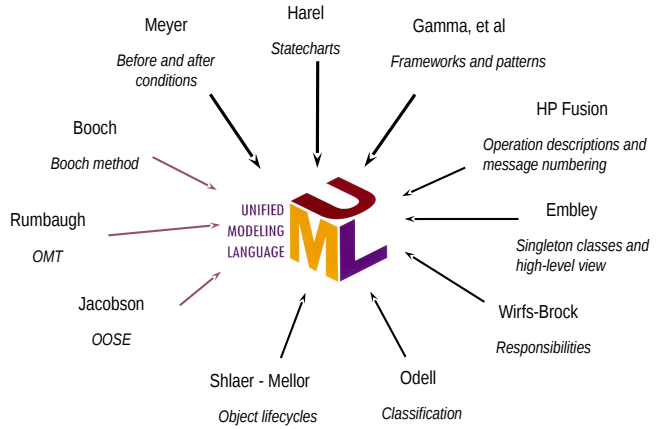
OMG Standards Development Organization

UML : Evolution

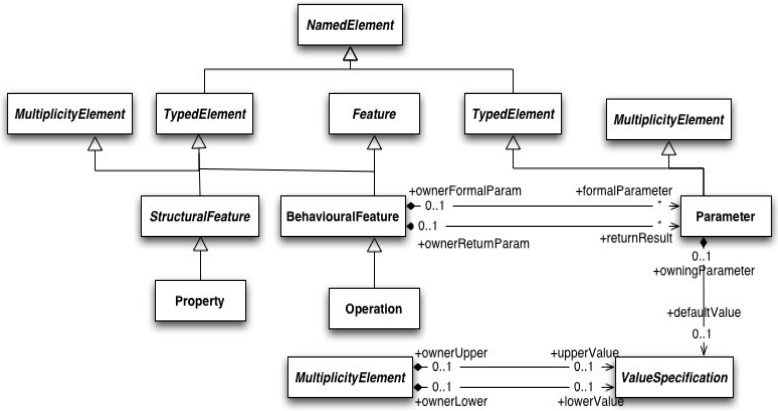




Contributions to UML 1.X

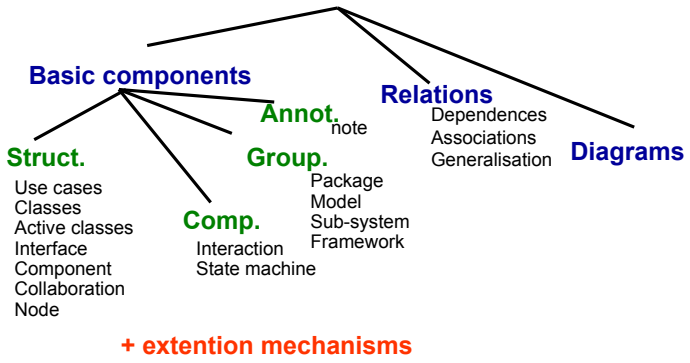


UML Meta-Model



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

UML Vocabulary



UML Diagrams

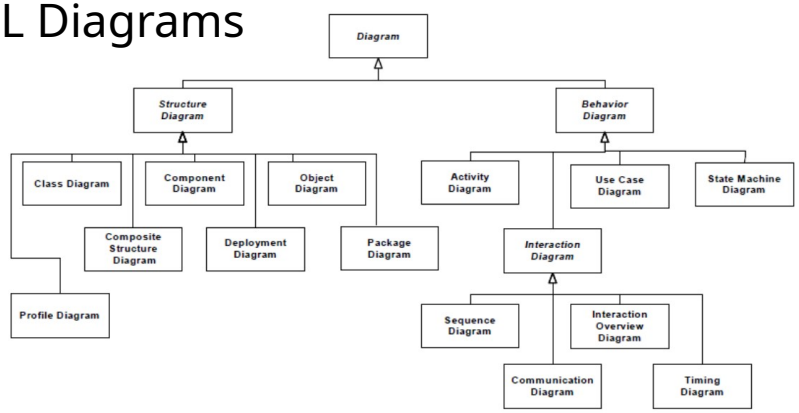
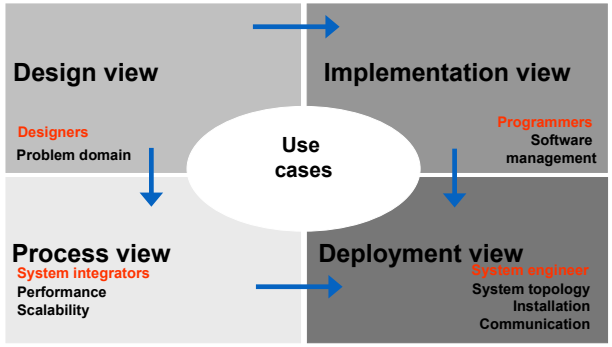


Figure A.5 The taxonomy of structure and behavior diagrams

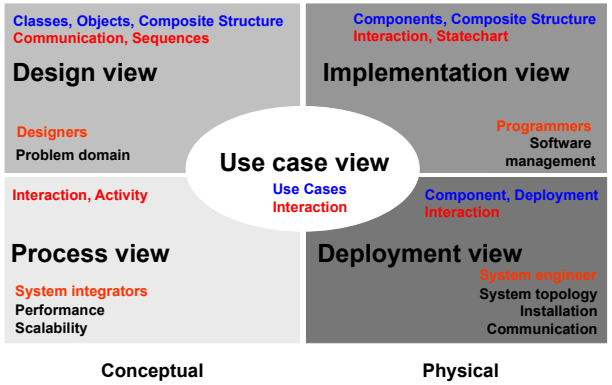
UML Specification, v2.5.1, p727 https://www.omg.org/spec/UML/2.5.1

Possibility of representing the same diagram at different levels of detail

Views on the Software



Diagrams within Views on the Software



Rules of thumb

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

Pointers

- The UML Specification https://www.omg.org/spec/UML/About-UML/
- https://www.uml-diagrams.org/

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.1 – Use case diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

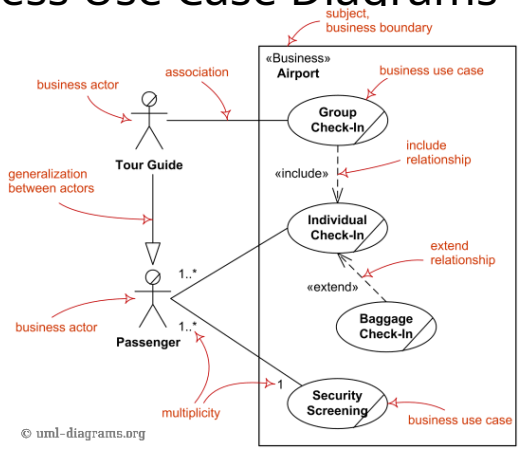
Use case diagrams

*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).*

terminology: **use case**, **actor**, **subject**, **extend**, include, **association**.

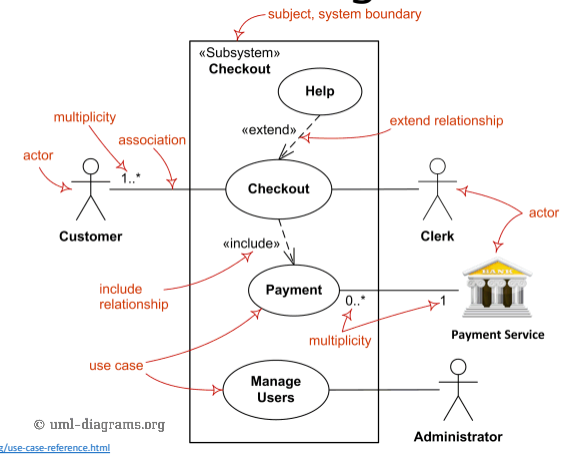
see also: https://www.uml-diagrams.org/use-case-reference.html

Business Use Case Diagrams



see also: https://www.uml-diagrams.org/use-case-reference.html

System Use Case Diagrams



see also: https://www.uml-diagrams.org/use-case-reference.html

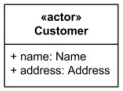
Actors and use cases

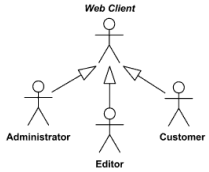
Actor

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.

Use case

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.





Generalization between actors

see also: https://www.uml-diagrams.org/use-case-reference.html

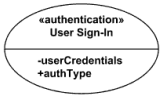
Actors and use cases

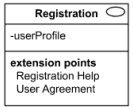
Actor

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.

Use case

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.



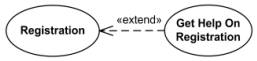


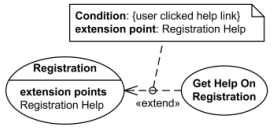
see also: https://www.uml-diagrams.org/use-case-reference.html

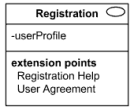
Includes and Extends

Extends

Extend is a **directed relationship** that specifies how and when the behavior defined in usually supplementary (optional) **extending use case** can be inserted into the **behavior** defined in the **extended use case**.





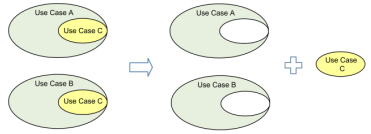


Includes

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.

see also: https://www.uml-diagrams.org/use-case-reference.html

Includes and Extends

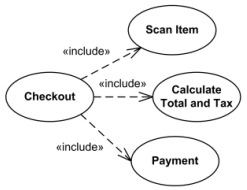


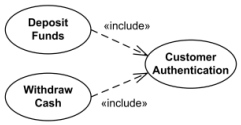
Includes

Use case include is a **directed relationship** between two **use cases** which is used to show that behavior of the **included** use case (the addition) is inserted into the **behavior** of the **including** (the base) use case.

The **include** relationship could be used:

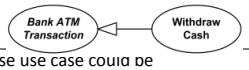
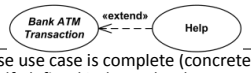
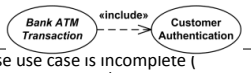
- to simplify large use case by splitting it into several use cases,
- to extract **common parts** of the behaviors of two or more use cases.





see also: https://www.uml-diagrams.org/use-case-reference.html

Use Case Relationships Compared

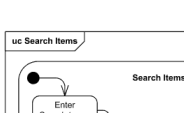
Generalization	Extend	Include
		
Base use case could be **abstract use case** (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (**abstract use case**).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

see also: https://www.uml-diagrams.org/use-case-reference.html

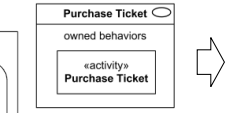
Describe Use Case Behavior

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

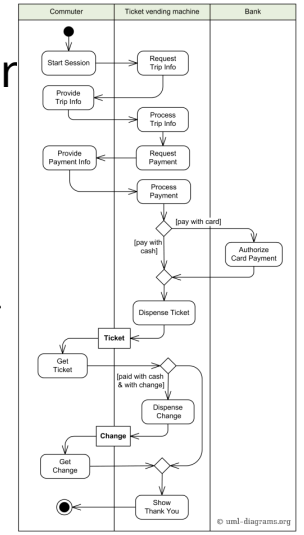
- **activity**,
- **state machine**,
- **interaction**.



description using a state machine diagram



link between a use case and an activity



Activity diagram: description of the Purchase Ticket activity

see also: https://www.uml-diagrams.org/use-case-reference.html

Use Case diagrams examples

See https://www.uml-diagrams.org/use-case-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.2 – Activity diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

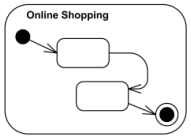
Activity diagrams

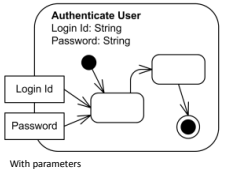
Activity diagram *is UML* ***behavior diagram*** *which shows* ***flow of control*** *or* ***object flow*** *with emphasis on the sequence and conditions of the flow. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available,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

terminology: **activity**, partition, action, **object**, control, activity edge.

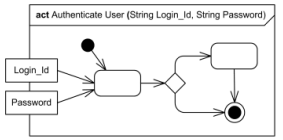
see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

Activity diagrams

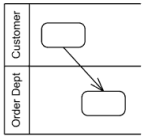




With parameters



Rendered with the frame notation for diagrams: keyword **act**

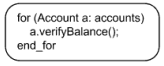


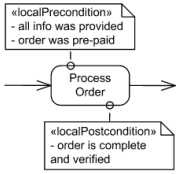
With swimlanes

terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

Actions

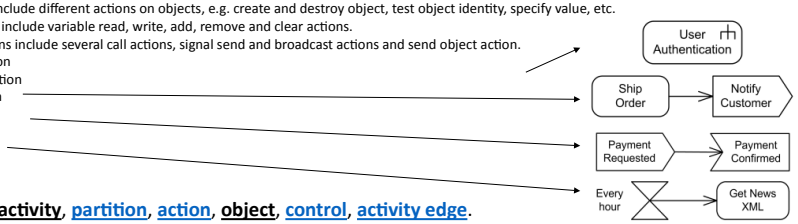




Types of actions

Action is a named element which represents a single atomic step within activity, i.e. that is not further decomposed within the activity. Activity represents a behavior that is composed of individual elements that are actions.

- Object actions include different actions on objects, e.g. create and destroy object, test object identity, specify value, etc.
- Variable actions include variable read, write, add, remove and clear actions.
- Invocation actions include several call actions, signal send and broadcast actions and send object action.
- Send signal action
- Accept signal action
- Wait time action
- ...



terminology: **activity**, partition, action, **object**, control, activity edge.

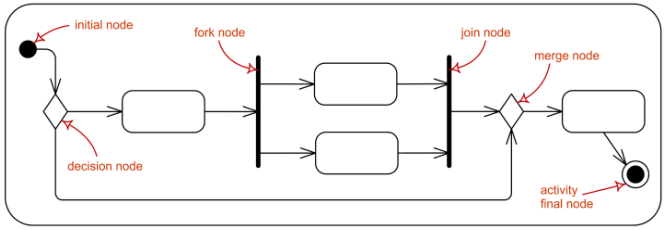
see also: https://www.uml-diagrams.org/activity-diagrams-actions.html

Controls

Types of controls

Control node is an activity node used to coordinate the flows between other nodes. It includes:

- initial node
- flow final node
- activity final node
- decision node
- merge node
- fork node
- join node



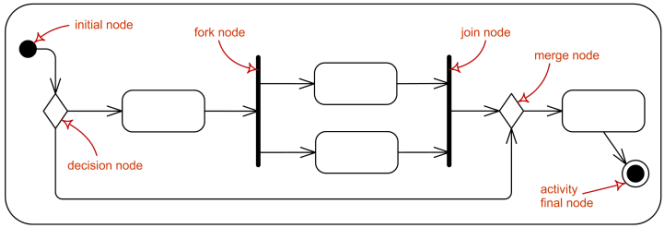
terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

Controls

Types of controls
Control node is an activity node used to coordinate the flows between other nodes. It includes:

- initial node
- flow final node
- activity final node
- decision node
- merge node
- fork node
- join node



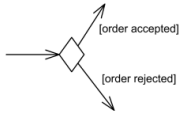
terminology: **activity**, partition, action, **object**, control, activity edge.

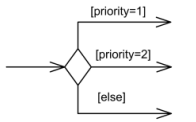
see also: https://www.uml-diagrams.org/activity-diagrams-objects.html

Controls

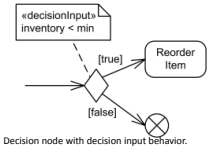
Types of controls
Control node is an activity node:

- initial node
- flow final node
- activity final node
- **decision node**
- merge node
- fork node
- join node

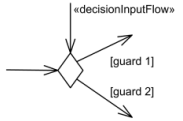




Decision node with outgoing edges with guards



Decision node with decision input behavior.



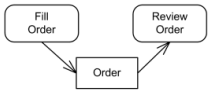
Decision node with decision input flow.

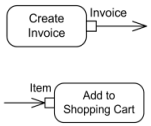
terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-controls.html

Objects

Objects flow in an activity





input and output pins

A data store is a central buffer node for non-transient information.

terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-controls.html

Activity diagrams examples

See https://www.uml-diagrams.org/activity-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.3 – State machine diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

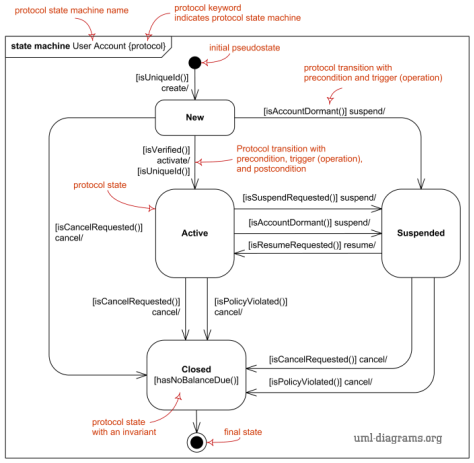
State machine diagrams

*Used for modeling discrete behavior through finite state transitions. In addition to expressing the **behavior** of a part of the system, state machines can also be used to express the **usage protocol** of part of a system. These two kinds of state machines are referred to as **behavioral state machines** and **protocol state machines**.*

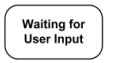
terminology: **behavioral state**, **behavioral transition**, **protocol state**, **protocol transition**, different **pseudostates**.

see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

State machine diagrams

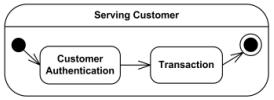


States

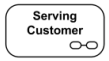


Simple state

List of internal activities

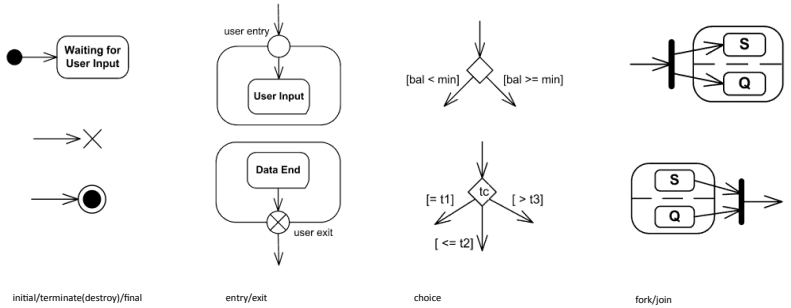


Composite state



Composite state with hidden decomposition

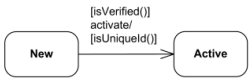
Pseudostates



initial/terminate(destroy)/final entry/exit choice fork/join

Protocol transition

A **protocol transition** is specialization of **(behavioral) transition** used for the protocol state machines which specifies a legal transition for an **operation**. Protocol transition has the following **features**: a pre-condition (guard), **trigger**, and a post-condition.



protocol-transition ::= [***pre-condition***] ***trigger*** '/' [***post-condition***]
pre-condition ::= '[' ***constraint*** ']'
post-condition ::= '[' ***constraint*** ']'

State Machine diagram examples

See https://www.uml-diagrams.org/state-machine-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.3 – Communication diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

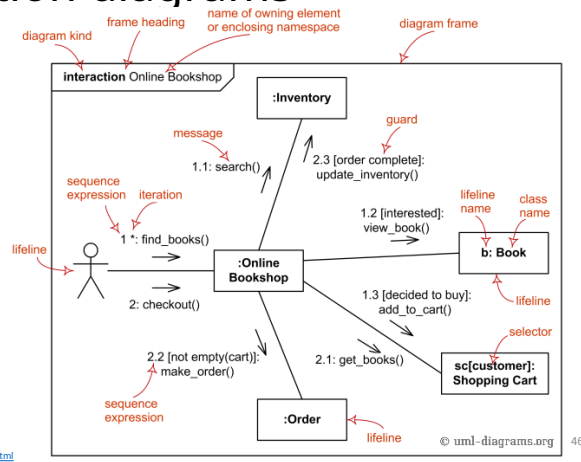
Communication diagrams

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.*

terminology: **frame**, **lifeline**, **message**

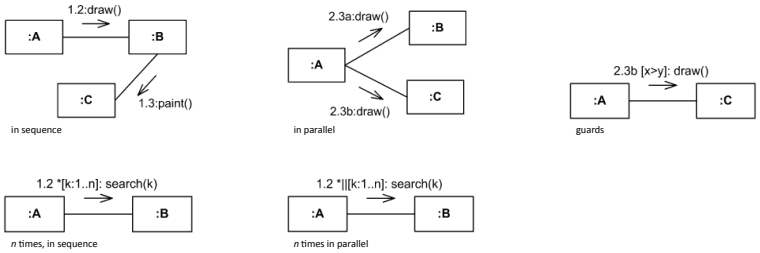
see also: https://www.uml-diagrams.org/communication-diagrams-reference.html

Communication diagrams



see also: https://www.uml-diagrams.org/communication-diagrams-reference.html

Communication diagrams



see also: https://www.uml-diagrams.org/communication-diagrams-reference.html

Communication diagram examples

See https://www.uml-diagrams.org/communication-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.1 – Class diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Class diagrams

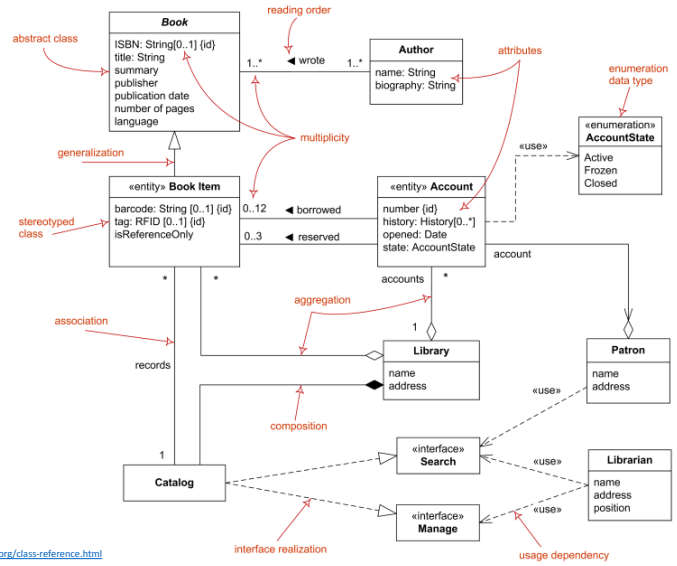
***Class diagram** is **UML** **structure diagram** which shows structure of the designed system at the level of **classes** and **interfaces**, shows their **features**, **constraints** and relationships - **associations**, **generalizations**, **dependencies**, etc.*

types of class diagrams are:

- **domain model diagram**,
- **diagram of implementation classes**.

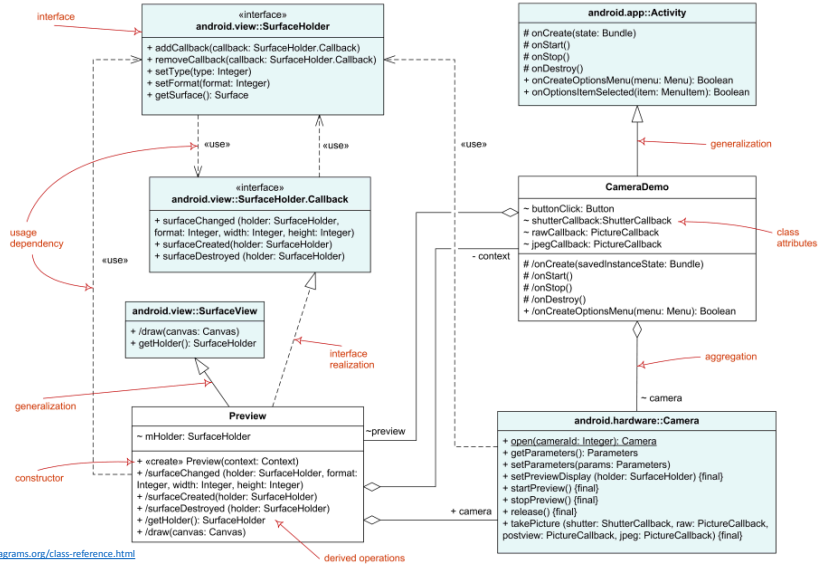
see also: https://www.uml-diagrams.org/class-reference.html

Domain model diagrams



see also: https://www.uml-diagrams.org/class-reference.html

Diagram of Implementation Classes



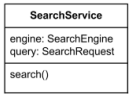
see also: https://www.uml-diagrams.org/class-reference.html

Class

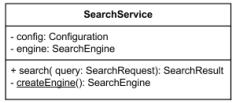
Class

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

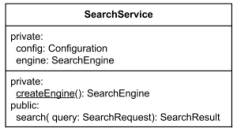
- **features**,
- **constraints**,
- **semantics (meaning)**.



Class SearchService - analysis level details



Class SearchService - implementation level details.
The createEngine is static operation

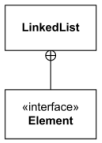


Class SearchService - attributes and operations grouped by visibility

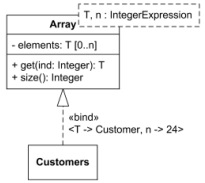
see also: https://www.uml-diagrams.org/class-reference.html

Abstract, Nested, Template, Interface

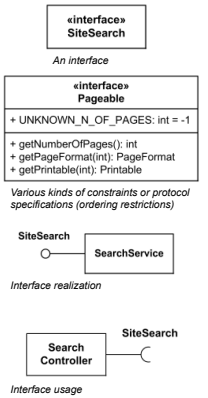
Abstract class (italics)



Class LinkedList is nesting the Element interface. The Element is in scope of the LinkedList namespace.



Template class Array and bound class Customers. The Customers class is an Array of 24 objects of Customer class.

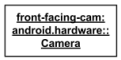


see also: https://www.uml-diagrams.org/class-reference.html

Objects

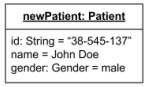
Anonymous instance of the Customer class

Instance newPatient of the unnamed or unknown class



Instance front-facing-cam of the Camera class from android.hardware package.

Instance orderPaid of the Date class has value July 31, 2011 3:00 pm.

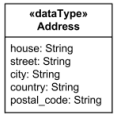


Instance newPatient of the Patient class has slots with values specified.

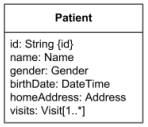
see also: https://www.uml-diagrams.org/class-reference.html

Data Type, primitive type, enumeration type

DateTime data type

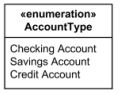


Structured data type



Attributes of the Patient class are of data types Name, Gender, DateTime, Address and Visit.

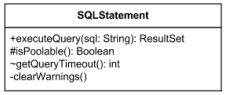
Primitive data type. Standard UML primitive types include:
- Boolean,
- Integer,
- UnlimitedNatural,
- String.



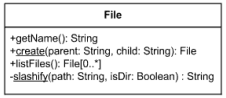
values are enumerated in the model as user-defined enumeration literals

see also: https://www.uml-diagrams.org/class-reference.html

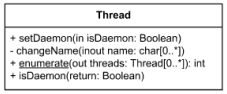
Operations



Operations with different visibilities
executeQuery is public, isPoolable is protected, getQueryTimeout has package visibility, clearWarnings is private.



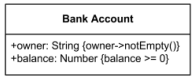
static operations are underlined
operations have a signature, with parameters, and a return type.
File has two static operations - create and slashify. Create has two parameters and returns File. Slashify is private operation. Operation listFiles returns array of files. Operations getName and listFiles either have no parameters or parameters were suppressed.



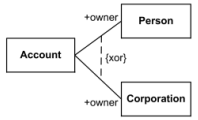
Operation setDaemon has one input parameter, while single parameter of changeName is both input and output parameter. Static enumerate returns integer result while also having output parameter - array of threads. Operation isDaemon is shown with return type parameter. It is presentation option equivalent to returning operation result as: +isDaemon(): Boolean.

see also: https://www.uml-diagrams.org/class-reference.html

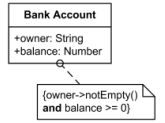
Write constraints



Bank account attribute constraints - non empty owner and positive balance.



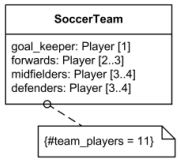
Account owner is either Person or Corporation, {xor} is predefined UML constraint.



Bank account constraints - non empty owner and positive balance

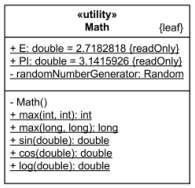
see also: https://www.uml-diagrams.org/class-reference.html

Members and multiplicity



Multiplicity of players for SoccerTeam class

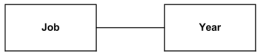
0	Collection must be empty
1	Exactly one instance
5	Exactly 5 instances
*	Zero or more instances
0..1	No instances or one instance
1..1	Exactly one instance
0..*	Zero or more instances
1..*	At least one instance
m..n	At least m but no more than n instances



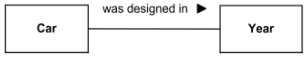
Utility: class that has only class scoped **static attributes and operations**.

see also: https://www.uml-diagrams.org/class-reference.html

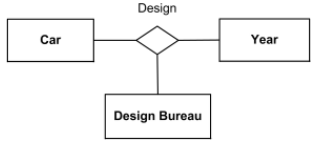
Associations



Association

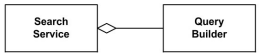


Order of the ends and reading: Car - was designed in - Year



Ternary association Design relating three classifiers.

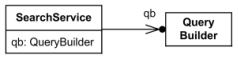
Aggregation/composition



Aggregation

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

Ownership



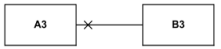
Association end qb is an attribute of SearchService class and is owned by the class.

Association qualifier

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

Navigability

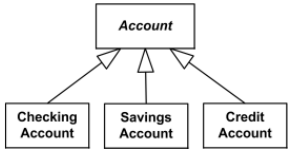
A2 has unspecified navigability while B2 is navigable from A2.



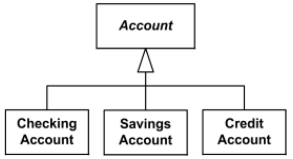
A3 is not navigable from B3 while B3 has unspecified navigability.

see also: https://www.uml-diagrams.org/class-reference.html

Generalization



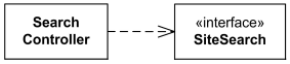
=

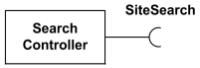


Checking, Savings, and Credit Accounts are generalized by Account.

see also: https://www.uml-diagrams.org/class-reference.html

Interfaces



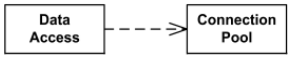


Interface SiteSearch is used (required) by Search Controller.

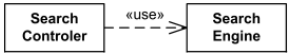
Interface SiteSearch is realized (implemented) by SearchService.

see also: https://www.uml-diagrams.org/class-reference.html

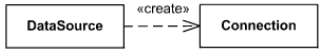
Dependency



Data Access depends on Connection Pool



Search Controller uses Search Engine.



Data Source creates Connection

see also: https://www.uml-diagrams.org/class-reference.html

Class diagram examples

See https://www.uml-diagrams.org/class-diagrams-examples.html

Pointers

- The UML Specification https://www.omg.org/spec/UML/About-UML/
- https://www.uml-diagrams.org/

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.2 – Package diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Package diagrams

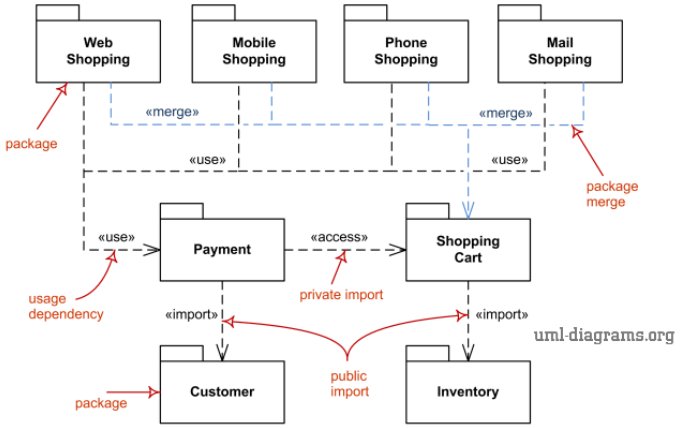
Package diagram *is* ***UML*** ***structure diagram*** *which shows structure of the designed system at the level of* ***packages****.*

Elements:

- **package**,
- **packageable element**,
- **dependency**,
- **element import**,
- **package import**,
- **package merge**.

see also: https://www.uml-diagrams.org/class-reference.html

Package diagrams

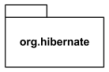


see also: https://www.uml-diagrams.org/package-diagrams-reference.html

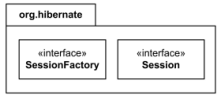
Package

Package

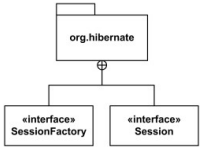
A **package** is a [namespace] used to group together elements that are semantically related and might change together. It is a general purpose mechanism to organize elements into groups to provide better structure for system model.



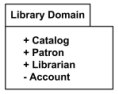
Package org.hibernate



Package org.hibernate contains SessionFactory and Session.



Package org.hibernate contains interfaces SessionFactory and Session.

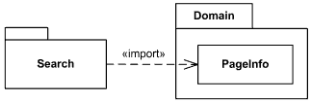


All elements of Library Domain package are public except for Account.

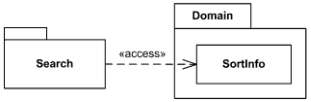
see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Import

Element Import

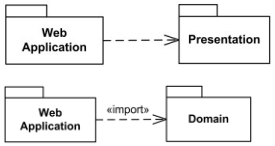


Public import of PageInfo element into Search namespace from Domain package.
Imported element are added to the namespace and made visible outside the namespace



Private import of SortInfo element into Search namespace from Domain package.
Imported element are added to the namespace but not visible outside the namespace

Package Import

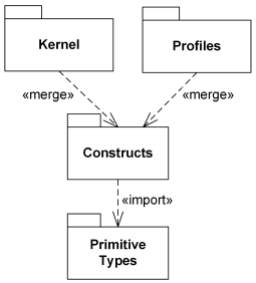


Public import:
All elements are added to the namespace and made visible outside the namespace

Private import:
All elements are added to the namespace but not visible outside the namespace

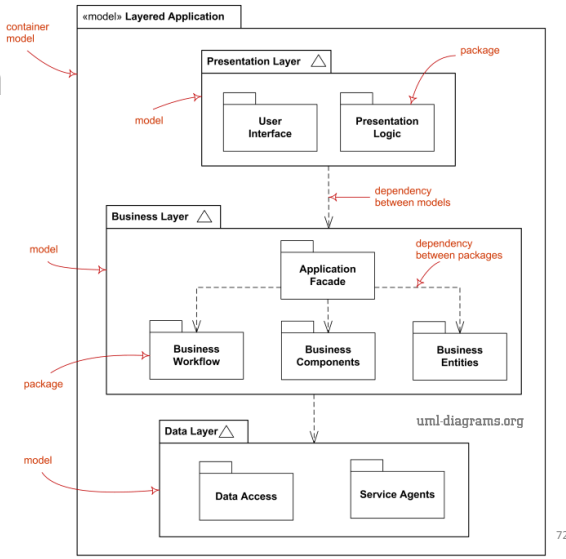
see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Package merge



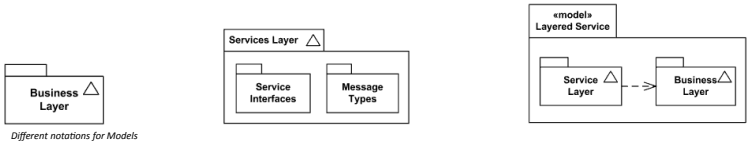
Kernel package merges Constructs package which imports Primitive Types.
The contents of Constructs is combined with the one of Kernel

Model diagra



see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Model



Different notations for Models

see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Package diagram examples

See https://www.uml-diagrams.org/package-diagrams-examples.html

Pointers

- The UML Specification https://www.omg.org/spec/UML/About-UML/
- https://www.uml-diagrams.org/

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.3 – Composite Structure diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Composite Structure diagrams

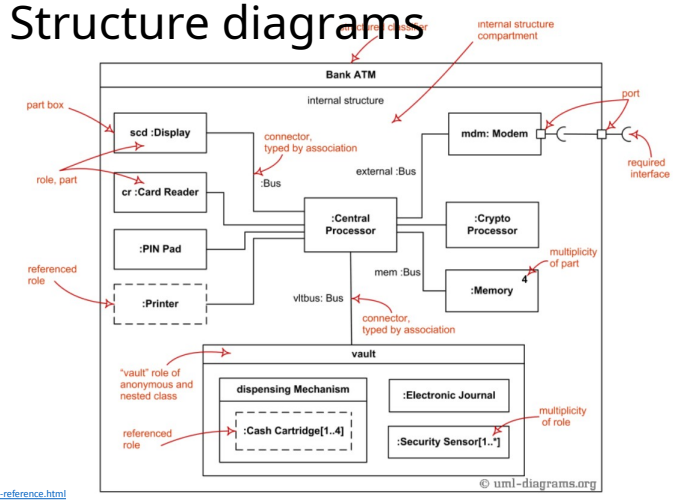
Composite Structure Diagram *could be used to show: internal structure of a classifier -* ***internal structure diagram****, classifier interactions with environment through* ***ports****, a behavior of a collaboration -* ***collaboration use diagram****.*

Elements:

- class,
- part,
- port,
- connector,
- usage

see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Composite Structure diagrams

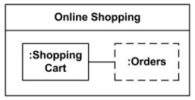


see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

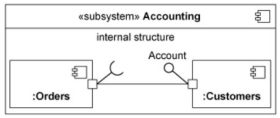
Structured classifier

Structured classifier

Structured classifier is classifier having **internal structure** and whose behavior can be fully or partially described by the collaboration of owned or referenced instances.



Different notations for structured classifiers

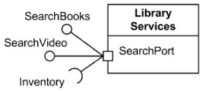


Simple ports joined directly by connector, mandatory UML notation. Customers component part provides Account interface to Orders part.

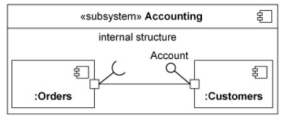
see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Encapsulated Classifier

Encapsulated classifier is structured classifier extended with the ability to own ports.



Library Services is classifier encapsulated through Search Port



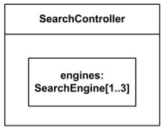
Simple ports joined directly by connector, mandatory UML notation. Customers component part provides Account interface to Orders part.

see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

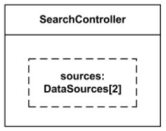
Part

represents a set of instances that are owned by a containing instance of a classifyer.

all parts are destroyed when the containing classifier instance is destroyed (composition)



Search Controller has 1 to 3 engines - Search Engine part

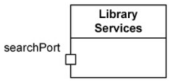


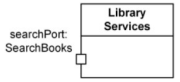
Two Data Sources is sources property - but not part - of Search Controller

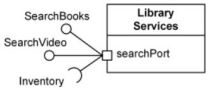
see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

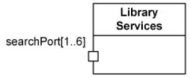
Port

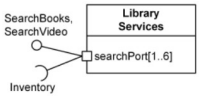
feature which specifies a link that enables communication between two or more instances playing some roles within a structured classifier.







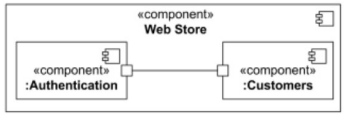




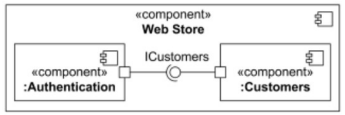
see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Connectors

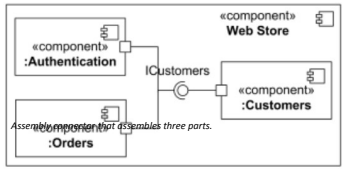
feature which specifies a link that enables communication between two or more instances playing some roles within a structured classifier.



Assembly connector between ports of Authentication and Customers components.



Assembly connector between simple ports of Authentication and Customers components.



Assembly connector that assembles three parts.

see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Composite structure diagram examples

See https://www.uml-diagrams.org/composite-structure-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.4 – Component diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Component diagrams

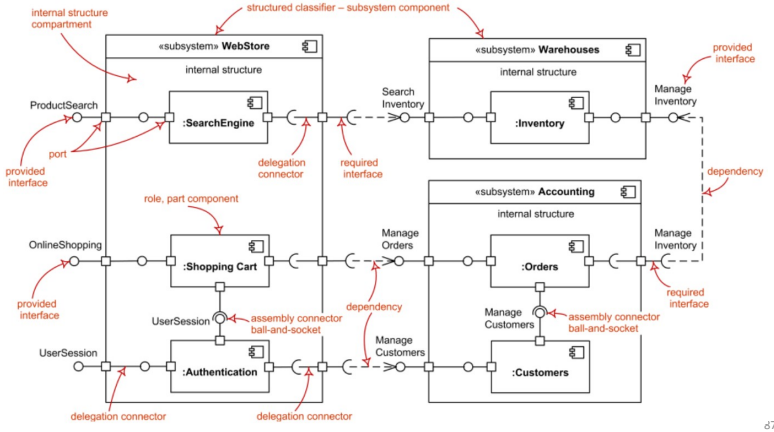
Component diagram *shows components, provided and required interfaces, ports, and relationships between them. This type of diagrams is used in* ***Component-Based Development*** *(**CBD**) to describe systems with* ***Service-Oriented Architecture*** *(**SOA**).*

Elements:

- component,
- provided interface,
- required interface,
- port,
- connectors.

see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Component diagrams



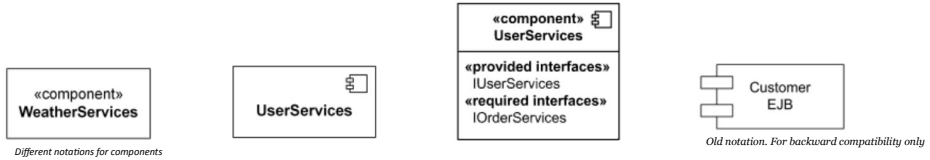
see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Components

Component

A *component* is a **class** representing a modular part of a system with encapsulated content and whose manifestation is replaceable within its environment.

A component has its behavior defined in terms of **provided interfaces** and required interfaces (potentially exposed via **ports**)

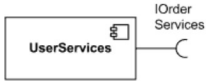


Different notations for components

Old notation. For backward compatibility only

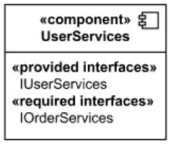
see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Interfaces

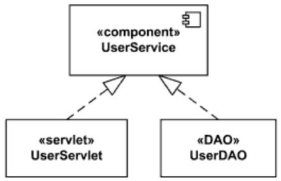


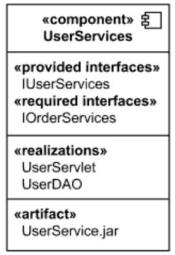
User Services component requires IOrderServices interface.

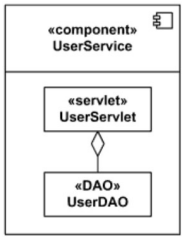
Weather Services component provides (implements) Weather Forecast interface.



Realization

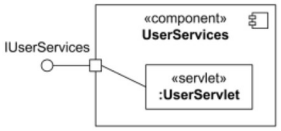




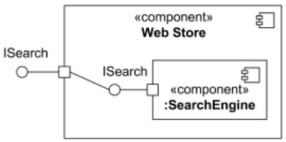


Different notations for:
Component UserService realized by UserServlet and UserDAO..

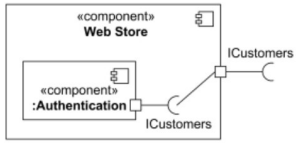
Delegation



Delegation connector from the delegating port to the UserServlet part.

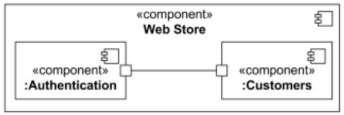


Delegation handled by a single port

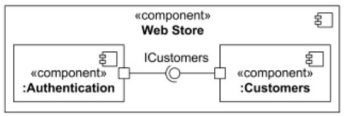


Delegation connector from the simple port of Authentication component to the delegating port.

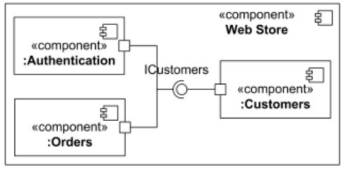
Assembly



Assembly connector between ports of Authentication and Customers components.



Assembly connector between simple ports of Authentication and Customers components.



Assembly connector that assembles three parts.

Component diagram examples

See https://www.uml-diagrams.org/component-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.5 – Deployment diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Deployment diagrams

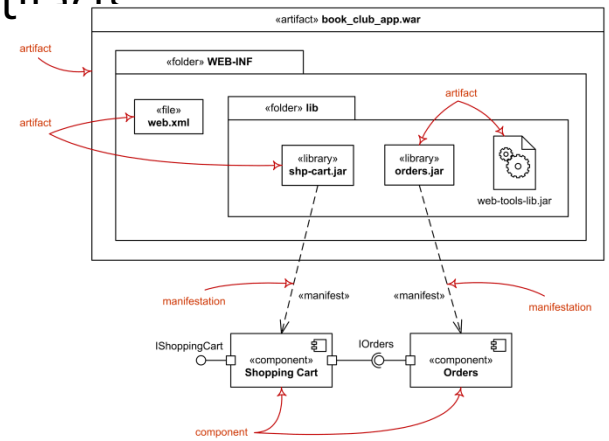
Deployment Deployment diagram *is a* ***structure diagram*** *which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets.*

Some common types of deployment diagrams are:

- **Implementation (manifestation) of components by artifacts**,
- **Specification level deployment diagram**,
- **Instance level deployment diagram**,
- **Network architecture of the system**.

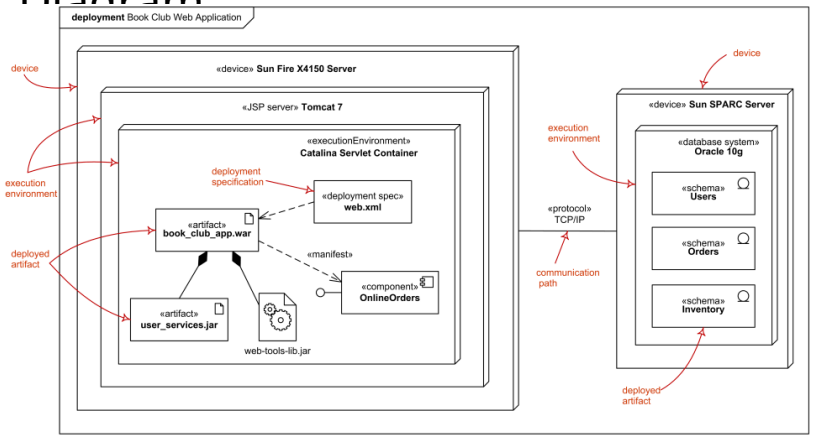
see also: https://www.uml-diagrams.org/deployment-diagrams-reference.html

Manifestation of Components by Artifacts



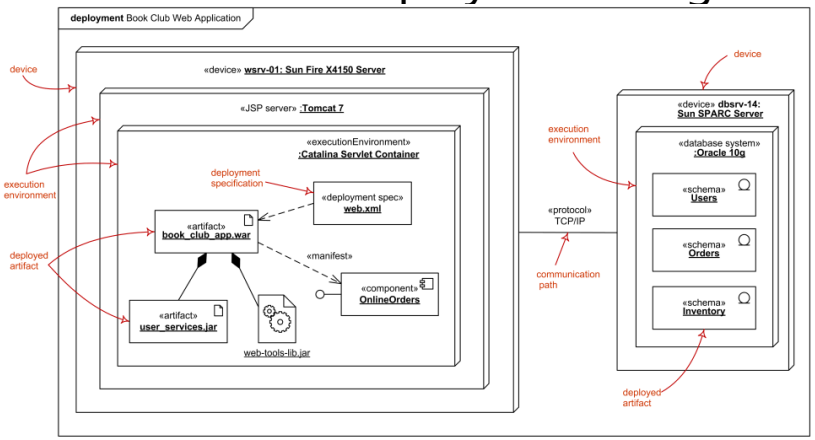
see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Specification Level Deployment Diagram



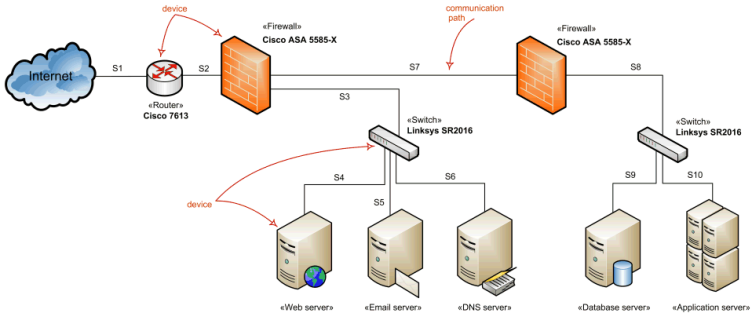
see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Instance Level Deployment Diagram



see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Network Architecture Diagrams



see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Deployment diagram examples

See https://www.uml-diagrams.org/deployment-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.6 – Sequence diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Sequence diagrams

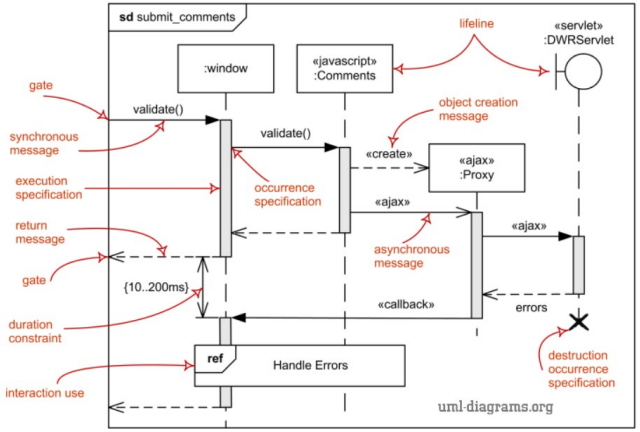
Sequence diagram *is the most common kind of* ***interaction diagram****, which focuses on the* ***message*** *interchange between a number of* ***lifelines****.*

Types of nodes

- **lifeline**,
- **execution specification**,
- **message**,
- **combined fragment**,
- **interaction use**,
- **state invariant**,
- **continuation**,
- **destruction occurrence**.

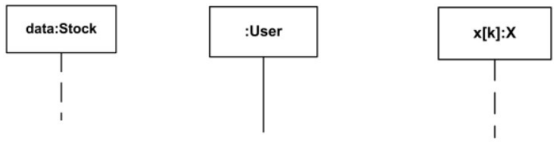
see also: https://www.uml-diagrams.org/sequence-diagrams-reference.html

Sequence diagrams



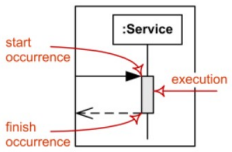
see also: https://www.uml-diagrams.org/sequence-diagrams-reference.html

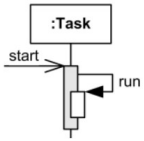
Lifeline

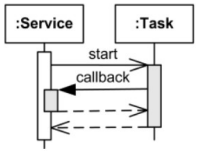


see also: https://www.uml-diagrams.org/sequence-diagrams-reference.html

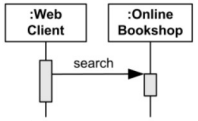
Execution



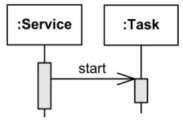




Calls

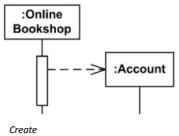


Synchronous
Web Client searches Online Bookshop and waits for results.

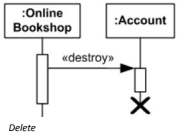


Asynchronous
Service starts Task and proceeds in parallel without waiting.

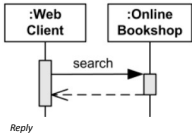
Messages



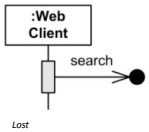
Create



Delete



Reply



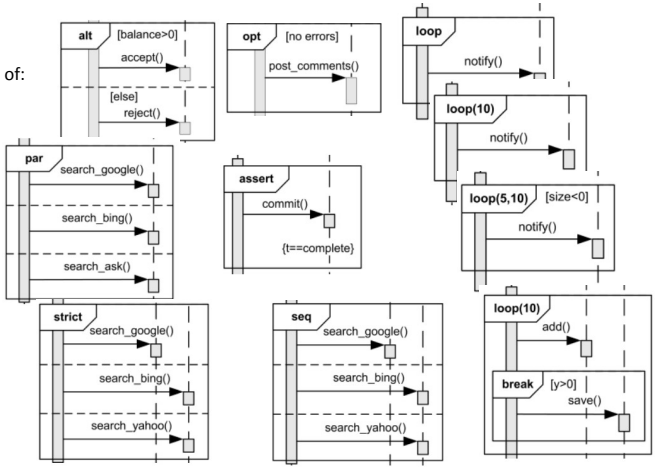
Lost

Found

Combined fragment with interaction operator

Interaction operator could be one of:

- **alt** - alternatives
- **opt** - option
- **loop** - iteration
- **break** - break
- **par** - parallel
- **strict** - strict sequencing
- **seq** - weak sequencing
- **critical** - critical region
- **ignore** - ignore
- **consider** - consider
- **assert** - assertion
- **neg** - negative



Sequence diagram examples

See https://www.uml-diagrams.org/sequence-diagrams-examples.html