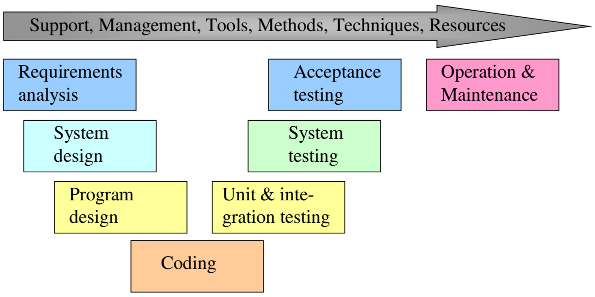
UML - Unified Modeling Language

Christoph Kessler, IDA, Linköpings universitet

Most slides by courtesy of Kristian Sandahl

Software engineering process



Modeling as a Design Technique

- Testing a physical entity before building it
- Communication with customers
- Visualization
- Reduction of complexity

- Models **supplement** natural language
- Models support understanding, design, documentation
- Creating a model forces you to take necessary design decisions
- **UML** is now the standard notation for modeling software.

Literature on UML

- Official standard documents by OMG: www.omg.org, www.uml.org
- Current version is UML 2.0 (2004/2005)
  - OMG documents: *UML Infrastructure*, *UML Superstructure*
- Books:
  - Pfleeger: *Software Engineering* 3rd ed., 2005 (mostly Chapter 6)
  - Rumbaugh, Jacobson, Booch: *The Unified Modeling Language Reference Manual*, Second Edition, Addison-Wesley 2005
  - Blaha, Rumbaugh: *Object-Oriented Modeling and Design with UML*, Second Edition, Prentice-Hall, 2005.
  - Stevens, Pooley: *Using UML: Software Engineering with Objects and Components*, 2nd edition. Addison-Wesley, 2006
  - And many others…

UML: Different diagram types for different views of software

Modeling (logical) structure of software:
- Static view: Class diagram
- Design view: Structure diagram, collaboration diagr., component d.
- Use case view: Use case diagram

Modeling behavior of software:
- Activity view: Activity diagram
- State machine view: State machine diagram
- Interaction view: Sequence diagram, communication diagram

Modeling physical structure of software
- Deployment view: Deployment diagram

Modeling the model, and extending UML itself
- Model management view: Package Diagram
- Profiles

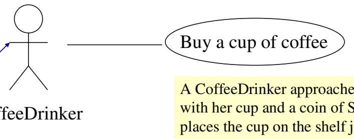
Use-case modelling

A use-case is:

"… a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction of sequence of interrelated events."

Jacobson, m fl 1992: Object-oriented software engineering. Addison-Wesley

Use-case diagram

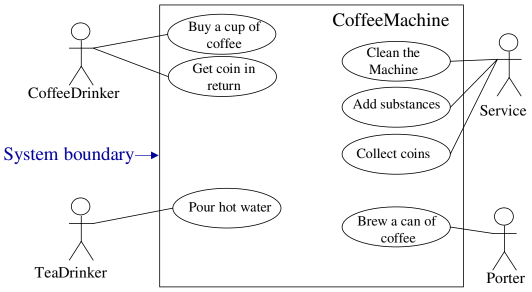


Actor: a user of the system in a particular role. Can be human or system.

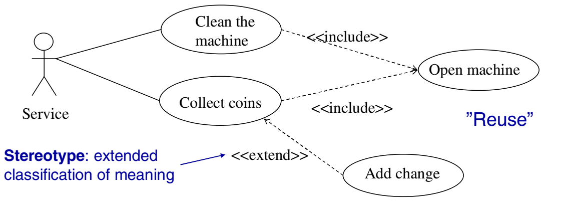
Detail of use-case

A CoffeeDrinker approaches the machine with her cup and a coin of SEK 5. She places the cup on the shelf just under the pipe. She then inserts the coin, and presses the button for coffee to get coffee according to default settings. Optionally she might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and rings a bell when it is ready. The CoffeeDrinker takes her cup from the shelf.

Use-case diagram for the coffee machine



Relations between use-cases



”Reuse”

Stereotype: extended classification of meaning

”Separating scenarious” (often conditional)

Please, keep as simple as possible.

Identifying classes: noun analysis

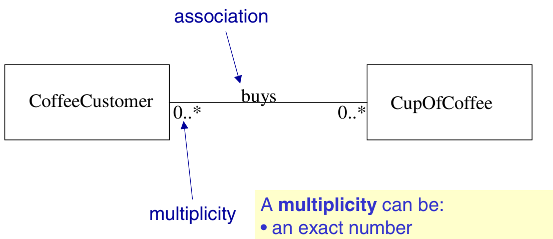
A CoffeeDrinker approaches the machine with her cup and a coin of SEK 5. She places the cup on the shelf just under the pipe. She then inserts the coin, and presses the button for coffee to get coffee according to default settings. Optionally, she might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and rings a bell when it is ready. The CoffeeDrinker takes her cup from the shelf.

- **•machine – real noun handled by the system**
- •cup – unit for beverage
- •coin – detail of user and machine
- •shelf – detail of machine
- •pipe – detail of machine
- **•button– handled by the system**
- •sugar – detail of coffee
- •whitener – detail of coffee
- **•cup of coffee – handled by the system**
- **•indicator – not discovered**

The single class model

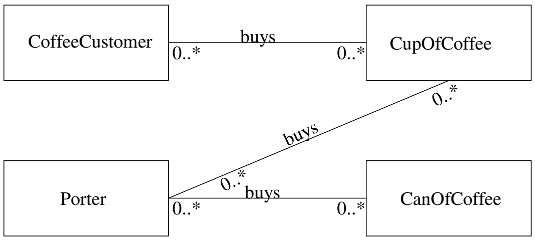
CoffeeCustomer	name
name: String	attribute
numberOfCoins() : Integer buy (c : CupOfCoffee)	operations

Associations between classes



A **multiplicity** can be:
- an exact number
- a range of numbers
- unspecified number denoted by *

Extended class model

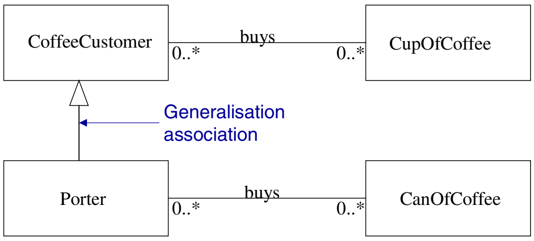


CoffeeCustomer 0..* buys 0..* CupOfCoffee

Porter 0..* buys 0..* CupOfCoffee

Porter 0..* buys 0..* CanOfCoffee

Revised class model

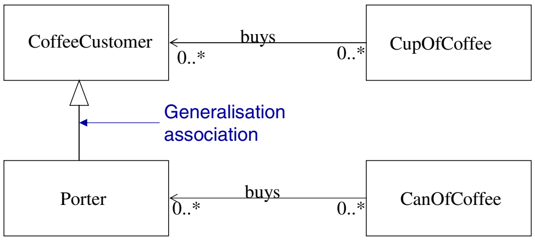


CoffeeCustomer 0..* buys 0..* CupOfCoffee

Generalisation association

Porter 0..* buys 0..* CanOfCoffee

Class model with navigability

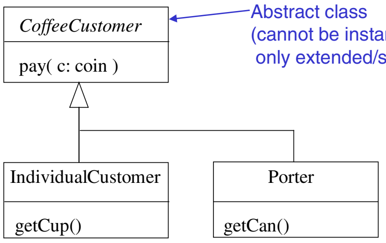


CoffeeCustomer 0..* buys 0..* CupOfCoffee

Generalisation association

Porter 0..* buys 0..* CanOfCoffee

Class model with inheritance and abstract classes



CoffeeCustomer

pay(c: coin)

IndividualCustomer

getCup()

Porter

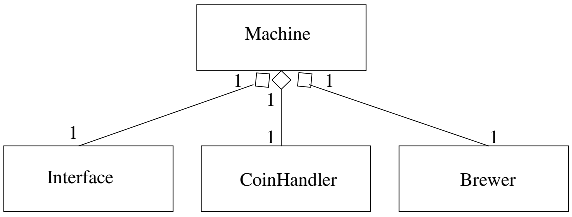
getCan()

Abstract class
(cannot be instantiated,
only extended/specialized)

pay() method is inherited from CoffeeCustomer

Class model with aggregation

Aggregation: part-of relationship



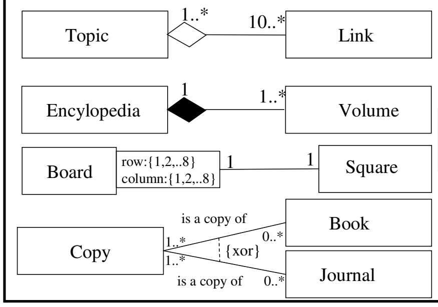
Machine

1 Interface 1

1 CoinHandler 1

1 Brewer 1

More relations between classes



Topic 1..* 10..* Link

aggregation

Encylopedia 1 1..* Volume

composition

Stronger form of aggregation: Composite has sole responsibility for managing its parts, e.g. allocation / deallocation

Board row:{1,2,..8} column:{1,2,..8} 1 1 Square

qualified association

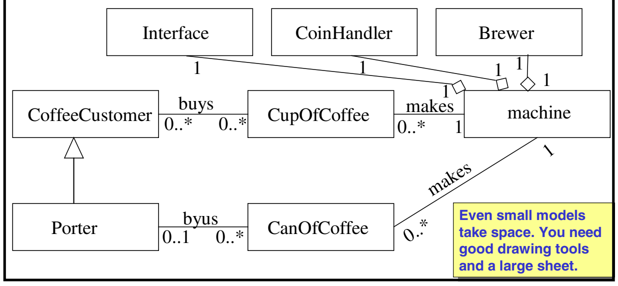
Copy 1..* is a copy of 0..* Book

Copy 1..* is a copy of 0..* Journal

{xor}

constraint

The coffee machine class model



Interface

CoinHandler

Brewer

CoffeeCustomer

buys

0..*

0..*

CupOfCoffee

makes

0..*

1

machine

1

1

1

1

1

1

1

Porter

byus

0..1

0..*

CanOfCoffee

makes

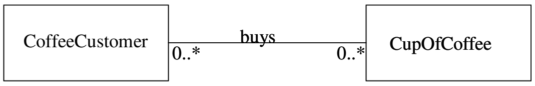
0..*

1

Even small models take space. You need good drawing tools and a large sheet.

Classes and objects

Classes:



CoffeeCustomer

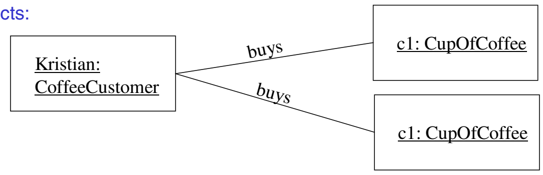
buys

0..*

0..*

CupOfCoffee

Objects:



Kristian: CoffeeCustomer

buys

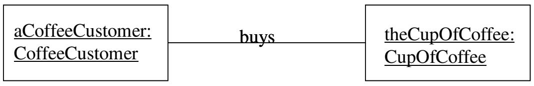
buys

c1: CupOfCoffee

c1: CupOfCoffee

Reasoning about an arbitrary object

Like this:

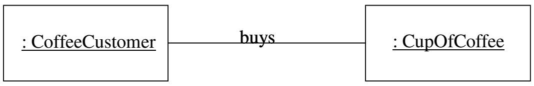


aCoffeeCustomer: CoffeeCustomer

buys

theCupOfCoffee: CupOfCoffee

...or simply like this:

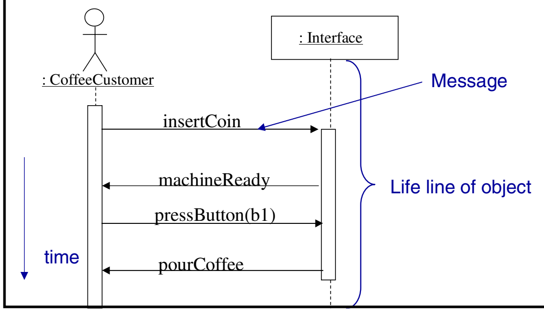


: CoffeeCustomer

buys

: CupOfCoffee

Sequence diagram



: CoffeeCustomer

: Interface

Message

insertCoin

machineReady

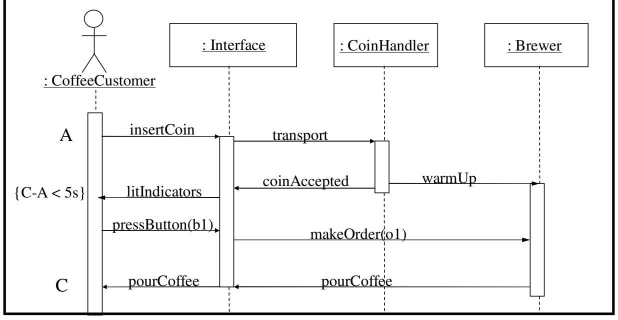
pressButton(b1)

pourCoffee

Life line of object

time

Sequence diagram with several objects



: CoffeeCustomer

: Interface

: CoinHandler

: Brewer

A

insertCoin

transport

{C-A < 5s}

litIndicators

coinAccepted

warmUp

pressButton(b1)

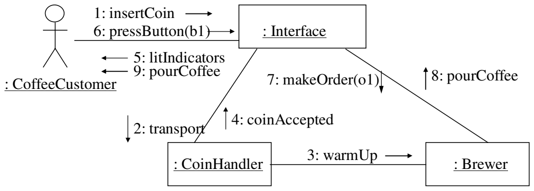
makeOrder(o1)

C

pourCoffee

pourCoffee

Communication diagram



1: insertCoin

6: pressButton(b1)

: Interface

5: litIndicators

9: pourCoffee

: CoffeeCustomer

7: makeOrder(o1)

8: pourCoffee

2: transport

4: coinAccepted

: CoinHandler

3: warmUp

: Brewer

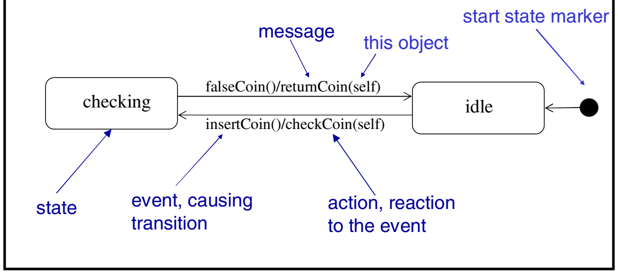
Shows message flows with sequence numbers

Similar information as sequence diagram

State machine diagram

Can formally describe protocols

For class CoinHandler:



message

this object

start state marker

checking

falseCoin()/returnCoin(self)

insertCoin()/checkCoin(self)

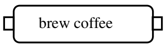
idle

state

event, causing transition

action, reaction to the event

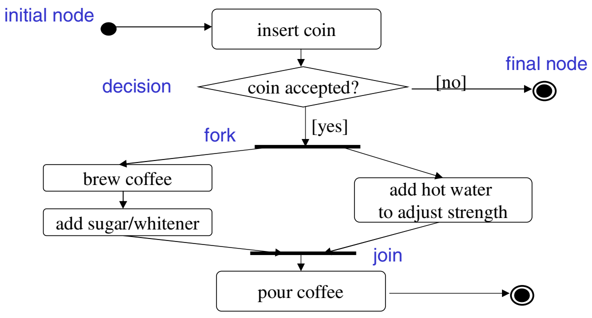
Activity Diagram



brew coffee

- Graph
  - Nodes are activities (actions)
    - Method invocations, operations, sending / receiving messages, handling events, creating / accessing / modifying / deleting objects, variables …
    - Data flow by input and output parameter pins
  - Edges are control flow transitions
  - To some degree dual to the state diagram
- Might be refined to a low-level specification; cf. control flow graph (~ compiler IR)
- A Petri Net
  - Interpretation by moving tokens along edges
  - Models concurrency by multiple tokens for ”current state”
  - Fork / join for synchronization
- Models real-world workflows

Activity diagram



initial node

insert coin

final node

decision

coin accepted?

[no]

[yes]

fork

brew coffee

add sugar/whitener

add hot water to adjust strength

join

pour coffee

Other features…

- Comments
- Constraints in OCL (Object Constraint Language)
- Profiles: Collections of stereotypes for specific domains, e.g. Realtime-profile for UML
  - Customize (specialize) UML elements, e.g. associations
  - Can introduce own symbols
- MOF (Meta-Object Facility):
  - UML is specified in UML
  - Powerful mechanism for extending UML by adding new language elements

UML Summary

- UML – the standard for modeling software
- Modeling before/during design, precedes coding
- Different diagrams for different views
- Model a software system only partially, focus on a certain aspect and/or part at a time
- Problem: Maintaining consistency across diagrams
- Tools
- Trend towards more detailed modeling
  - Stepwise refinement
  - ”executable UML”: UML 2 is almost a programming language…
  - UML is customizable and extendible: Profiles, MOF
- Trend towards automatized partial generation of models and code from models (MDA – model-driven architecture)

Homework Exercise

- Draw a class diagram for the following scenario:

A customer, characterized by his/her name and phone number, may purchase reservations of tickets for a performance of a show. A reservation of tickets, annotated with the reservation date, can be *either* a reservation by subscription, in which case it is characterized by a subscription series number, *or* an individual reservation. A subscription series comprehends at least 3 and at most 6 tickets; an individual reservation at most one ticket. Every ticket is part of a subscription series or an individual reservation, but not both. Customers may have many reservations, but each reservation is owned by exactly one customer. Tickets may be available or not, and one may sell or exchange them. A ticket is associated with one specific seat in a specific performance, given by date and time, of a show, which is characterized by its name. A show may have several performances.