A short introduction to UML and UML definition

based on **Bernd Bruegge's** book, **UML Reference manual** and including some personal points of view/opinions and examples using **OCLE**

Dan CHIOREAN
Babeş-Bolyai University
chiorean@cs.ubbcluj.ro

Introduction to UML

What is modeling?

Modeling consists of building an abstraction of reality

Model – an abstract description of a problem/problem solution

Abstractions are simplifications because:

- They ignore irrelevant details and
- They only represent the relevant details

What is *relevant* or *irrelevant* depends on the purpose of the model

We use modeling to produce software because:

- Software is getting increasingly more complex
- Modeling is a means for dealing with complexity

Introduction to UML

Abstraction as key technique in modeling

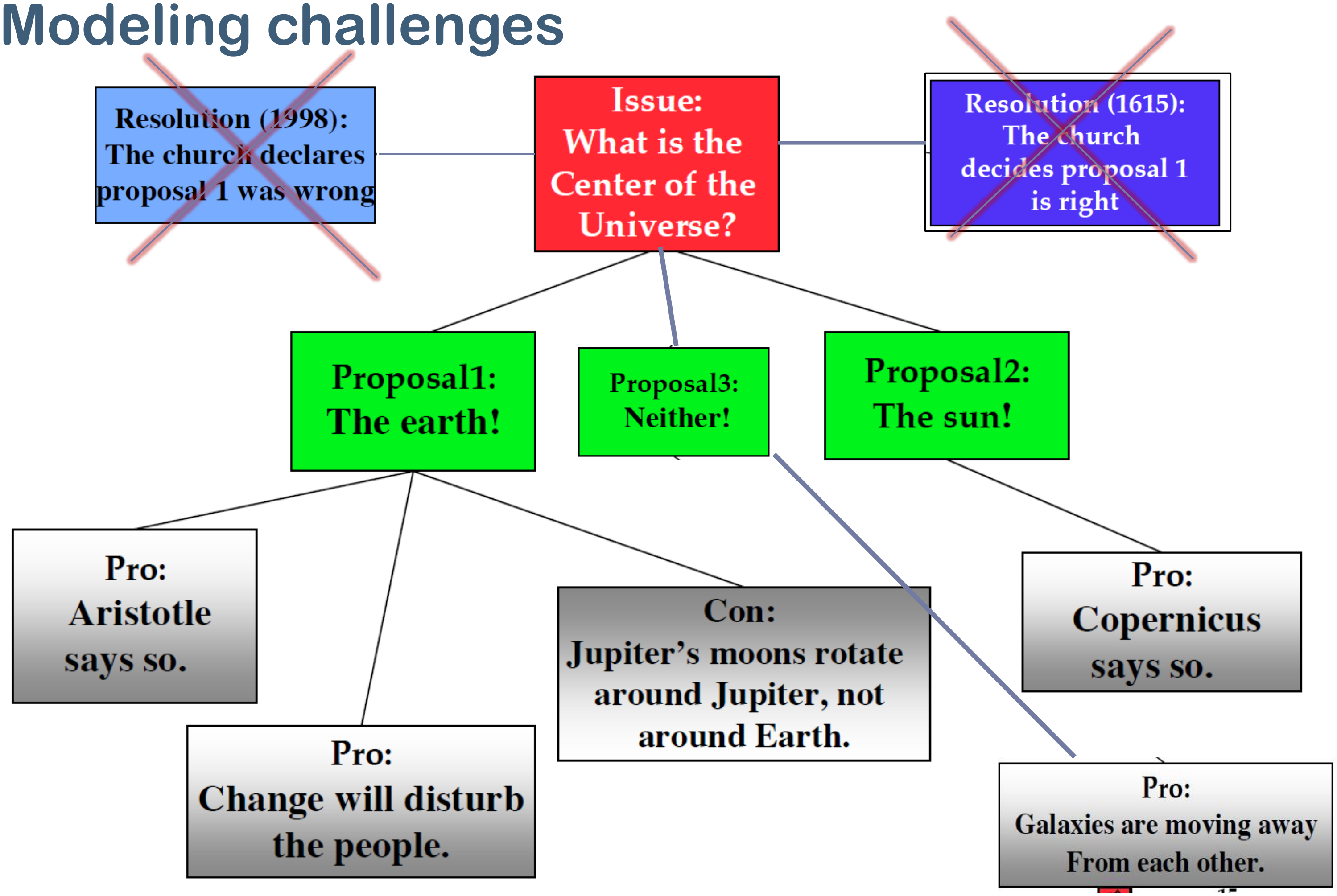
Two definitions for abstraction:

- Abstraction is a *thought process* where ideas are distanced from objects
- **Abstraction as activity**
- Abstraction is the *resulting idea* of a thought process where an idea has been distanced from an object
- **Abstraction as entity**
- Ideas can be expressed by models

Abstraction allows us to ignore unessential details

Introduction to UML

Modeling challenges



~~Resolution (1998): The church declares proposal 1 was wrong~~

Issue: What is the Center of the Universe?

~~Resolution (1615): The church decides proposal 1 is right~~

Proposal1: The earth!

Proposal3: Neither!

Proposal2: The sun!

Pro: Aristotle says so.

Pro: Change will disturb the people.

Con: Jupiter's moons rotate around Jupiter, not around Earth.

Pro: Copernicus says so.

Pro: Galaxies are moving away From each other.

Introduction to UML

Models must be falsifiable

- **Testing: The act of disproving a model.**

Introduction to UML

Models must be falsifiable

- **Karl Popper** ("Objective Knowledge):
- There is no absolute truth when trying to understand reality. One can only build theories, that are "true" until somebody finds a counter example
- **Falsification**: **The act of disproving a theory or hypothesis**
- **The truth of a theory is never certain**. We must use phrases like:
- "by our best judgement", "using state-of-the-art knowledge"
- **In software engineering any model is a theory**:
- We build models and try to find counter examples by:
    - Requirements validation, user interface testing, review of the design, source code testing, system testing, etc.
-

Introduction to UML

Three ways to deal with complexity

- Abstraction and Modeling
- Decomposition
- Hierarchy

UML supports all these three ways/techniques

Introduction to UML

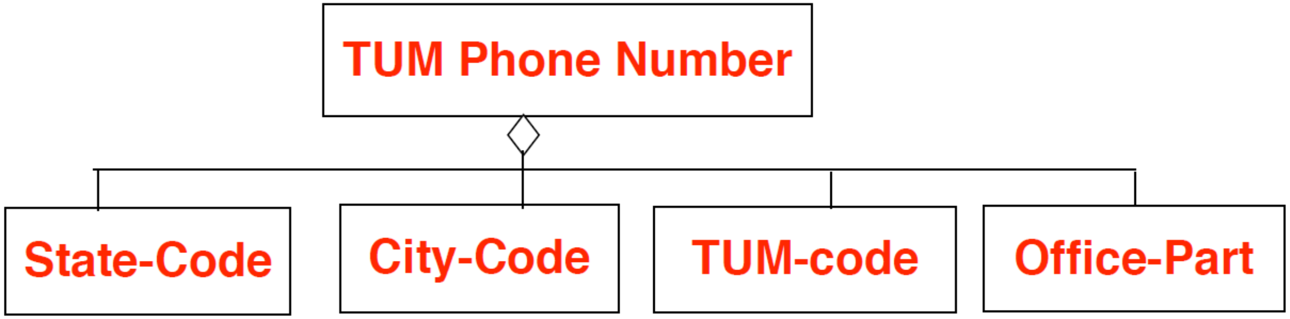
Decomposition

Complex systems are hard to understand

- The **7 +- 2 phenomena**
- Our short-term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
- TUM Phone Number: 498928918204

Chunking:

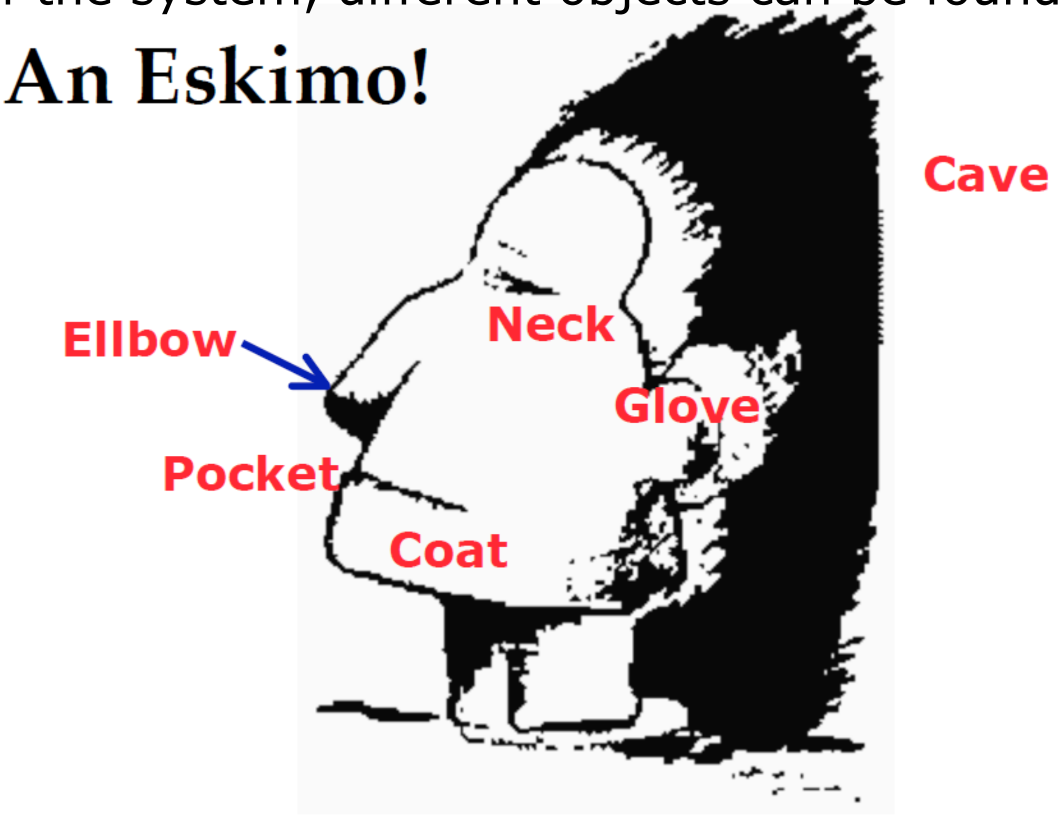
- Group collection of objects to reduce complexity
  - State-code, city-code, TUM-code, Office-Part



Introduction to UML

Object-oriented decomposition - What is this?

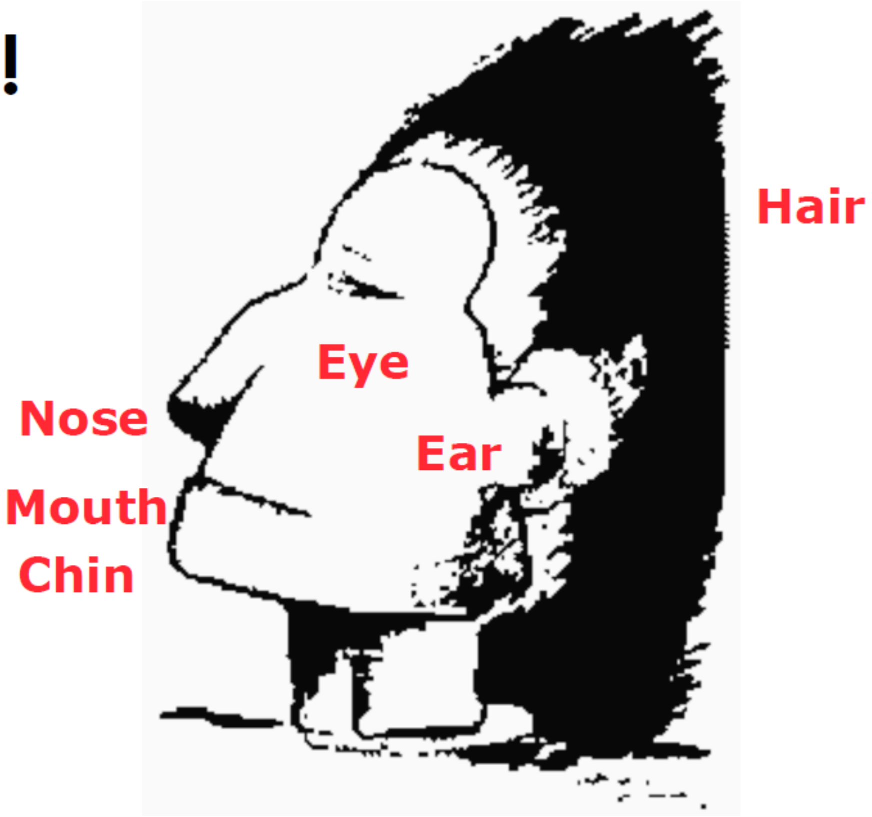
Object-oriented decomposition is good. Unfortunately, depending on the purpose of the system, different objects can be found



Introduction to UML

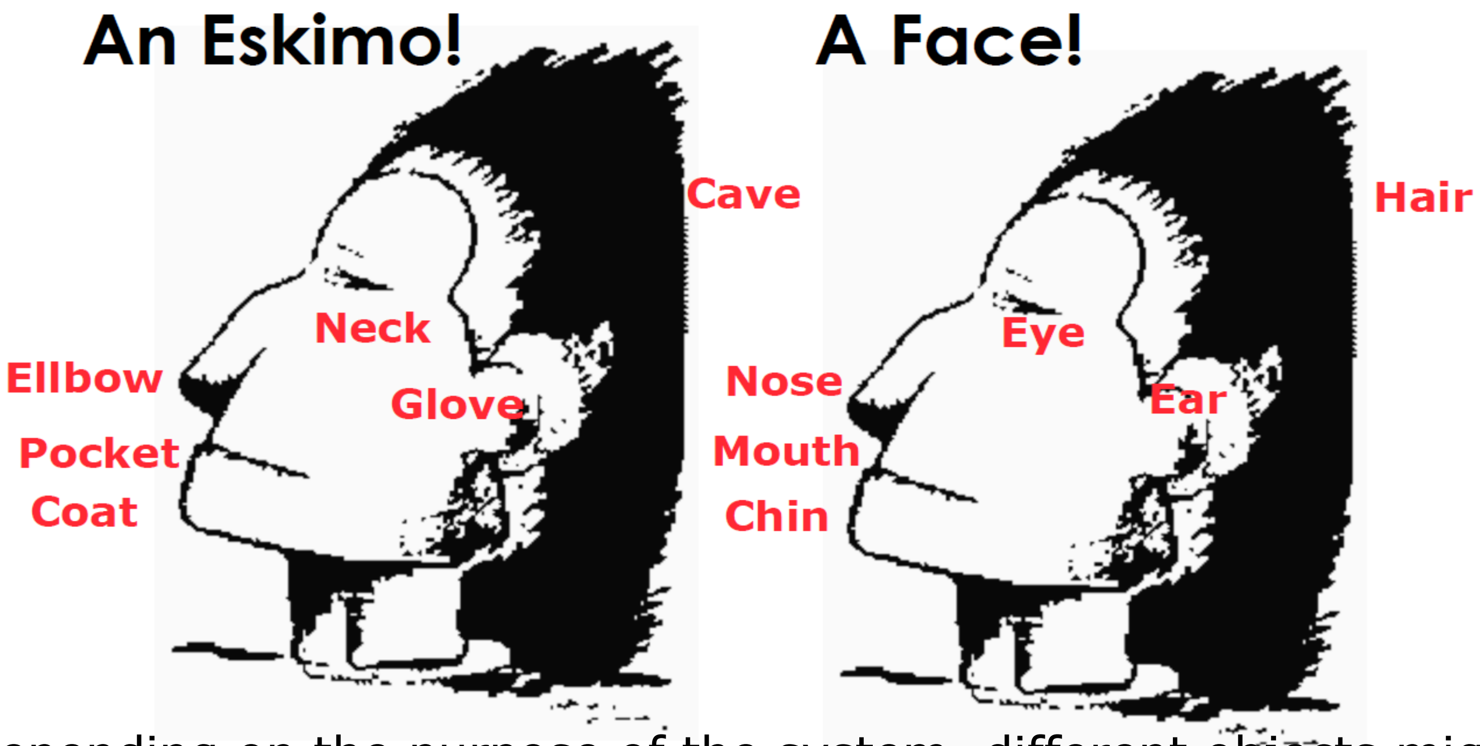
Object-oriented decomposition - What is this?

A Face!



Introduction to UML

Object-oriented decomposition - What is this?



Depending on the purpose of the system, different objects might be found

Identify the purpose of a system is crucial

Introduction to UML

Hierarchy

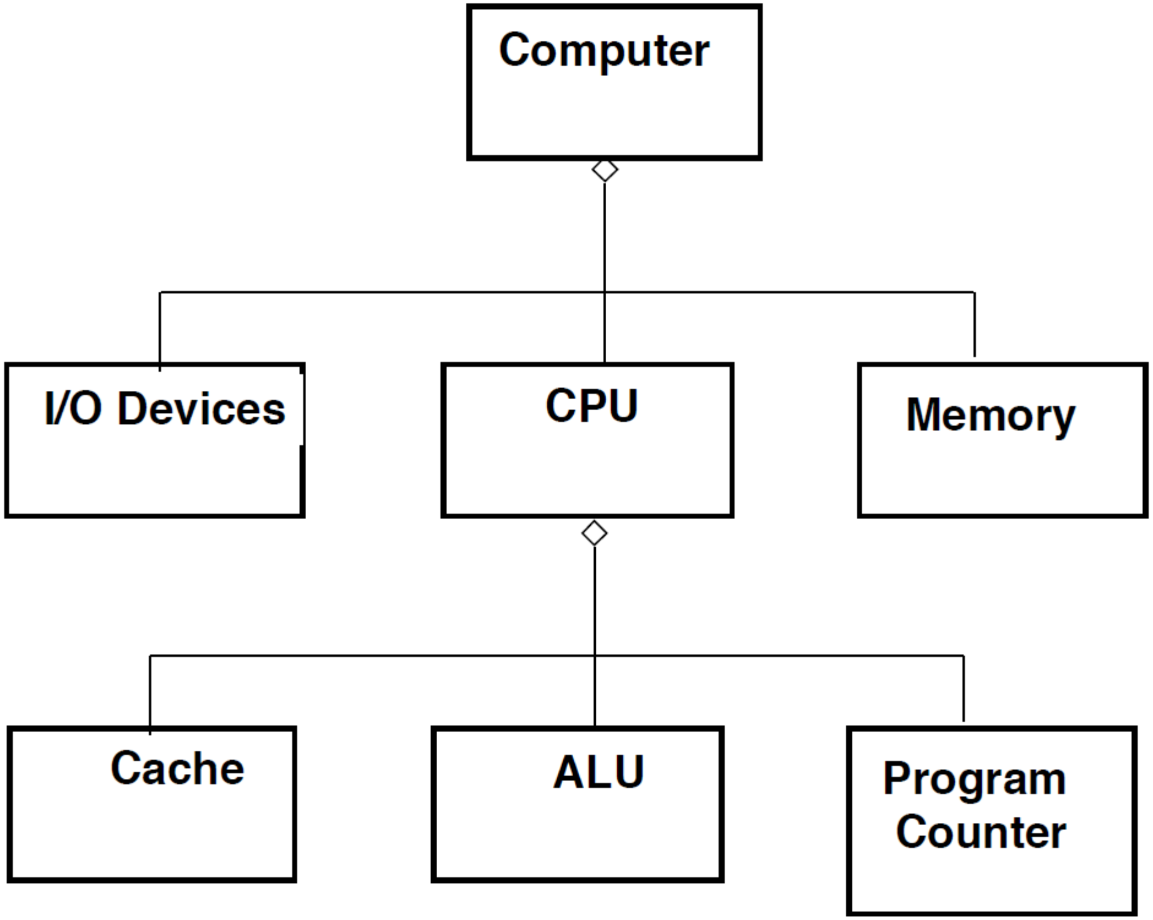
Another way to deal with complexity is to provide **relationships between these chunks**

One of the most important relationships is hierarchy. There are 2 useful/special hierarchies:

- "**Part-of**" hierarchy
- "**Is-kind-of**" hierarchy

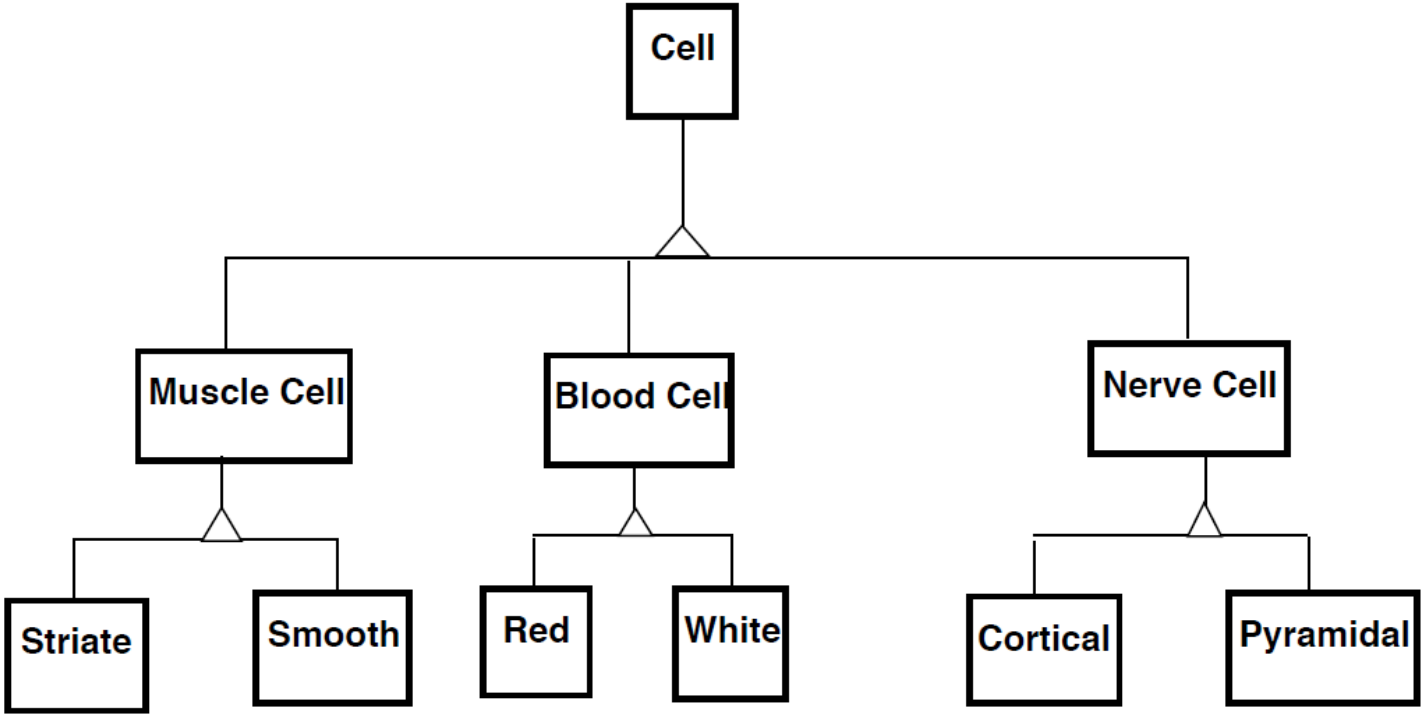
Introduction to UML

Part of Hierarchy - Aggregation



Introduction to UML

Is Kind of Hierarchy - Taxonomy



Introduction to UML

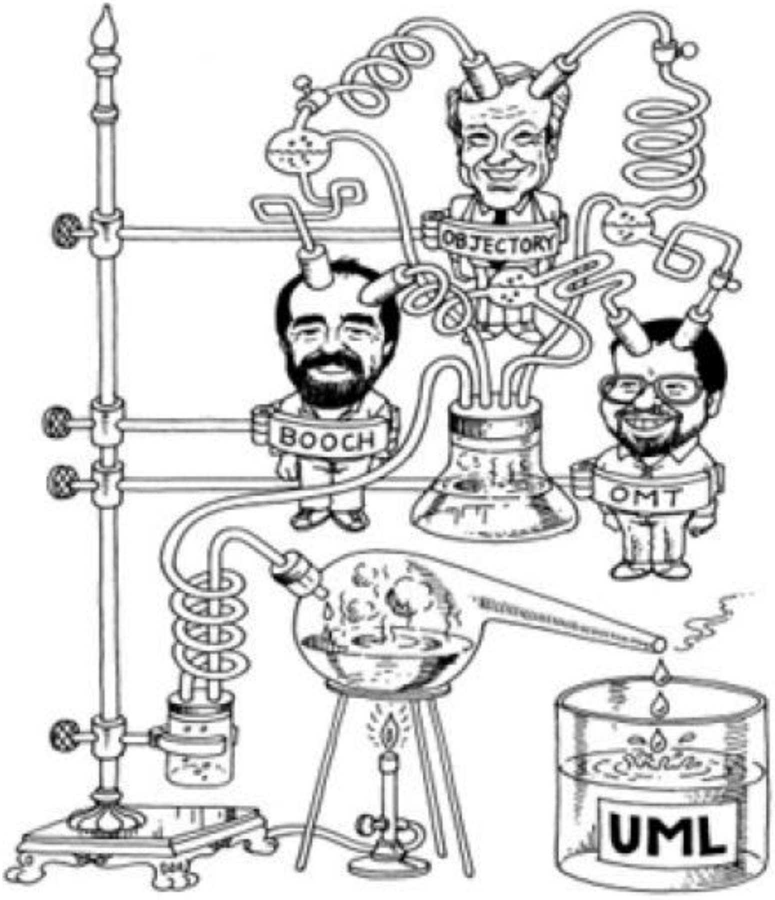
What is UML?

- **UML** (Unified Modeling Language) - 1997
- Nonproprietary standard for modeling software systems, **OMG**
- Convergence of notations used in object-oriented methods:
  - **OMT** (James Rumbaugh and colleagues)
  - **Booch** (Grady Booch)
  - **OOSE** (Ivar Jacobson)
- Information at **the OMG portal http://www.uml.org/**
- Commercial tools: Rational Software Modeler (IBM), Rational Rhapsody, Enterprise Architect, Visual Paradigm, MagicDraw

Introduction to UML

History of UML

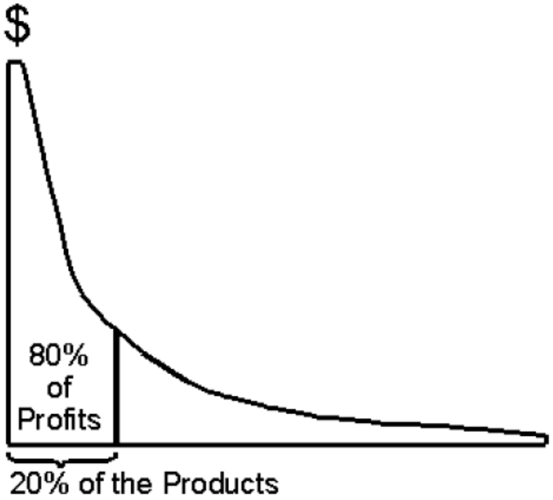
- the first version of UML- November 1997
- The version 1.5 implemented in OCLE was adopted in March 2003, formal-03-03-01.pdf, 736 pag. – including OCL 47 pag. chap. 6
- The current Version 2.5.1 - formal-17-12-05.pdf, 796 pag. – OCL specified in another doc. 262 pag. - formal-14-02-03.pdf, OCL 2.4, that is aligned with UML 2.4.1



Introduction to UML

Important to know

- You can solve 80% of the modeling problems by using 20 % UML
- We teach you those 20%
- 80-20 rule: Pareto principle



Vilfredo Pareto, 1848-1923

- Introduced the concept of Pareto Efficiency,
- Founder of the field of microeconomics.

Introduction to UML

Most important features of UML

- Apart from "languages‟ of OMT, OOD/Booch, OOSE methods contributing at UML definition, UML is specified in a more rigorous manner by means of concrete syntax, abstract syntax and semantics

- The abstract syntax is described by means of the UML metamodel and WFRs (invariants on metamodel classes). The language describing the UML metamodel, MOF is a subset of the UML language

- Beginning with the first version, UML included the OCL

Introduction to UML

What is a Model, View, Diagram?

- ***UML*** is a language supporting model specification

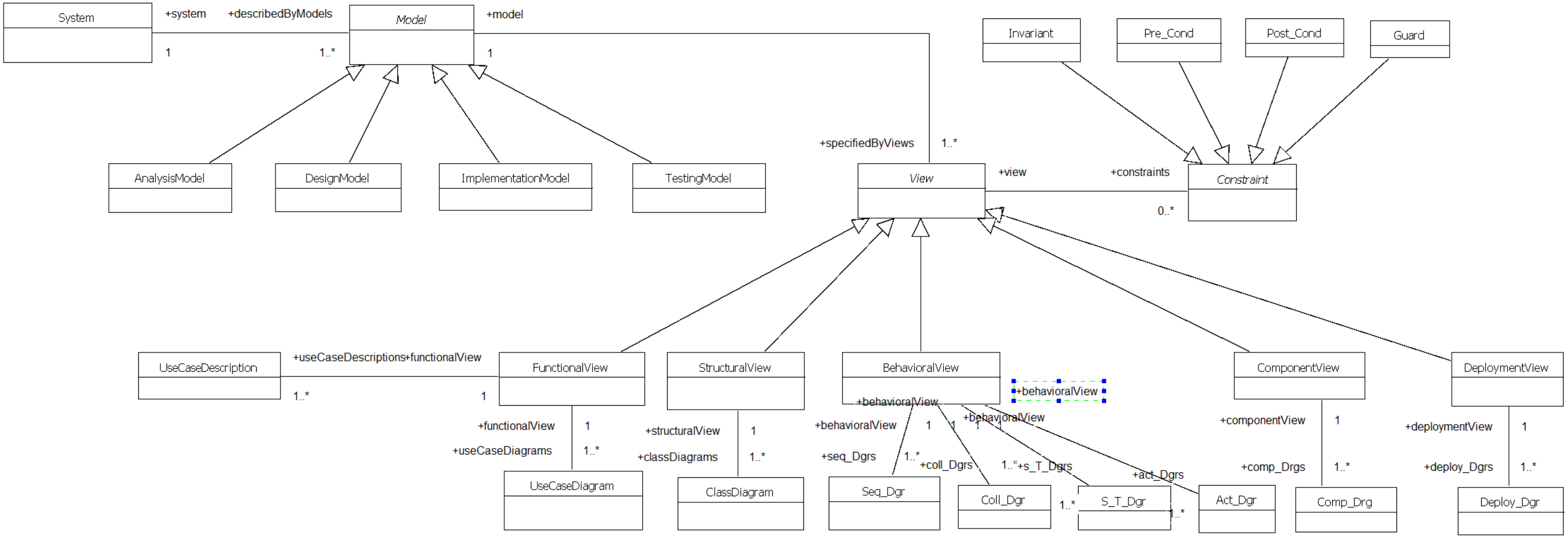
- A ***model*** is an **abstraction** describing a problem/problem domain, a problem solution/ system or a subsystem:
  - a system that no longer exists,
  - an existing system,
  - a future system to be built.

- A ***view*** depicts selected aspects of a model

- A ***diagram/notation*** is a set of graphical/textual rules for depicting views and formal specifications

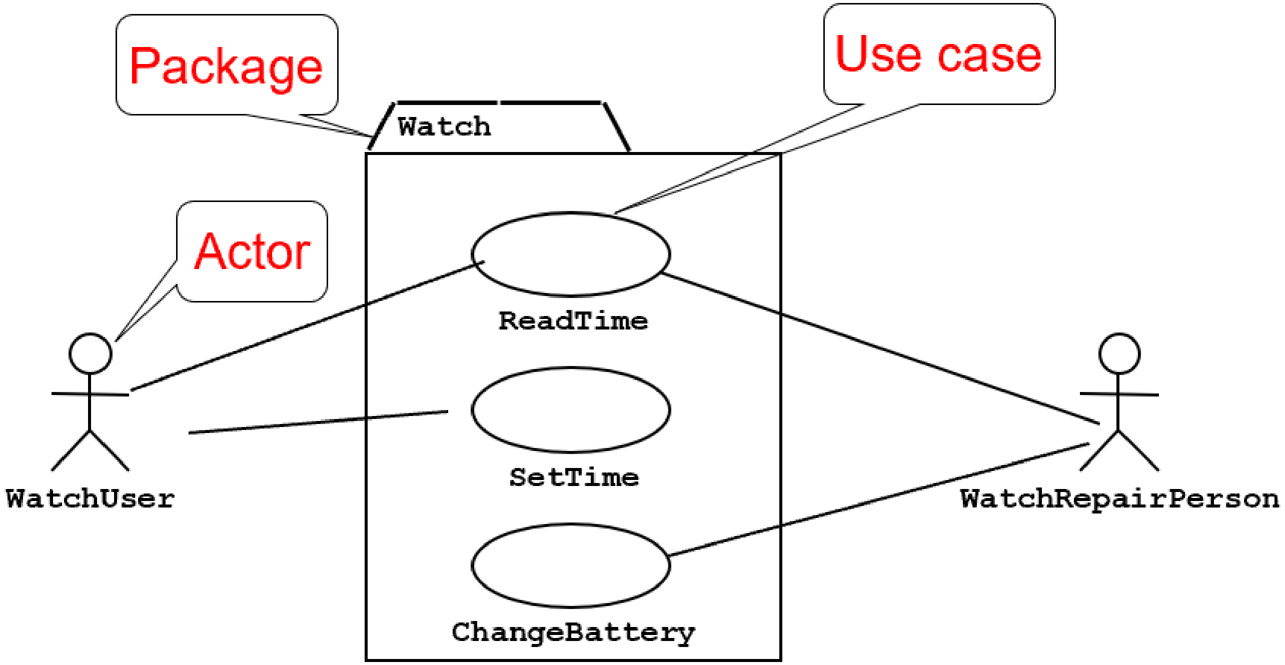
Introduction to UML

UML Models, Views, Diagrams



Introduction to UML

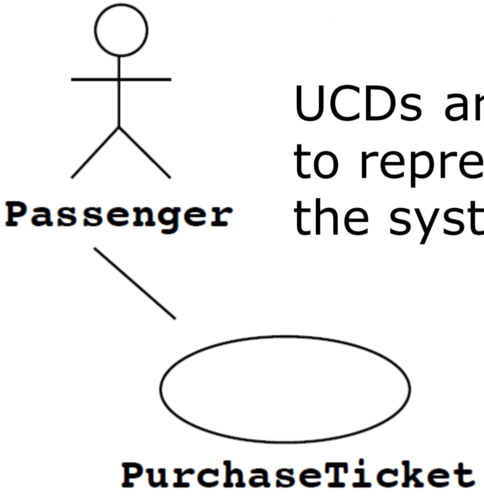
UML – A better understanding of the problem – the functionality



Use case diagrams represent the functionality of the system from user’s point of view

Introduction to UML

UML – the functional view



UCDs are used during requirements elicitation and analysis to represent external behavior ("visible from the outside of the system")

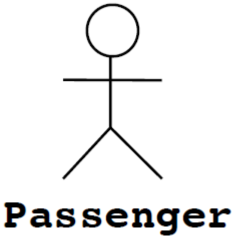
A **use case** represents a functionality provided by the system

An **Actor** represents a role, that is, a type of user of the system

The set of all use cases that completely describe the functionality of the system form the **Use Case View**/Model

Introduction to UML

UML – the functional view - Actors



An **actor** is a model of an external entity which interacts (communicates) with the system:

- User
- External system (Another system)
- Physical environment (e.g. Weather)

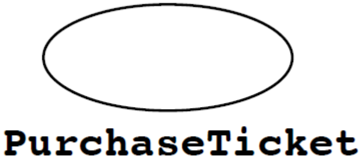
- An **actor** has a **unique name** and an **optional description**

Examples:

- **Passenger**: A person in the train
- **GPS satellite**: An external system that provides the system with GPS

Introduction to UML

UML – the functional view – Use Case



A **Use Case** represents a class of functionality provided by the system

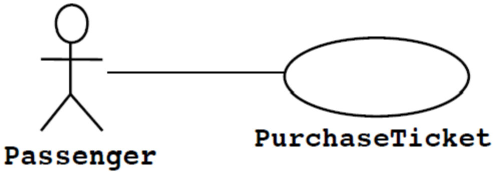
Use Cases can be described textually, with a focus on the event flow between actor and system

The textual **Use Case description** consists of 6 parts:

1. Unique name
2. Participating actors
3. Entry conditions
4. Exit conditions
5. Flow of events
6. Special requirements.

Introduction to UML

UML – the functional view – Use Case



1. Name: Purchase ticket

2. Participating actor: Passenger

3. Entry condition: Passenger stands in front of Ticket Distributor

- Passenger has sufficient money to purchase ticket

4. Exit condition: Passenger has ticket

5. Flow of events:

1. Passenger selects the number of zones to be traveled

2. Ticket Distributor displays the amount due

3. Passenger inserts money, at least the amount due

4. Ticket Distributor returns change

5. Ticket Distributor issues Ticket

6. **Special requirements:** None.

Introduction to UML

UML –Use Cases relationships

Inheritance Relationship

- To express specialization/generalization relations

Extends Relationship

- To represent seldom invoked use cases or exceptional functionality

Includes Relationship

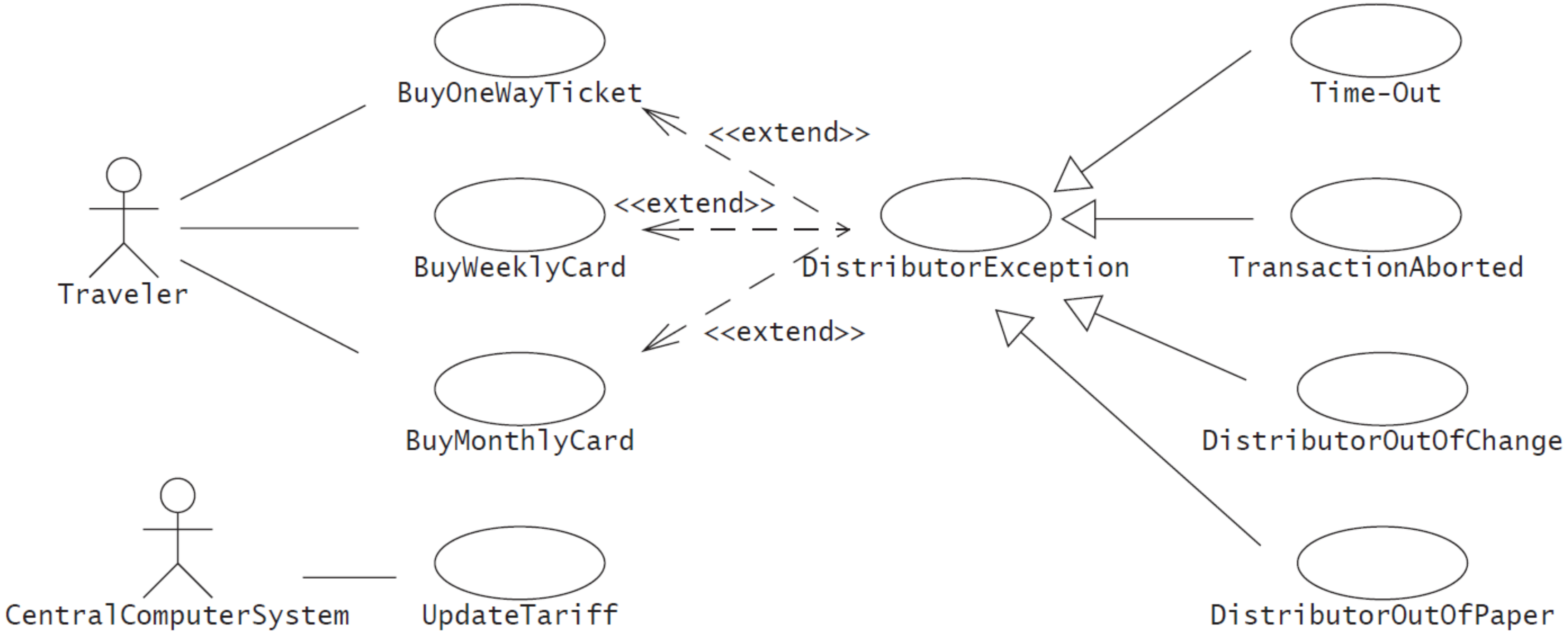
- To represent functional behavior common to more than one use case.

`<<extends>>` relationships model exceptional or seldom invoked cases

- The exceptional event flows are factored out of the main event flow for clarity
- The direction of an `<<extends>>` relationship is to the extended use case
- Use Cases representing exceptional flows can extend more than one use case.

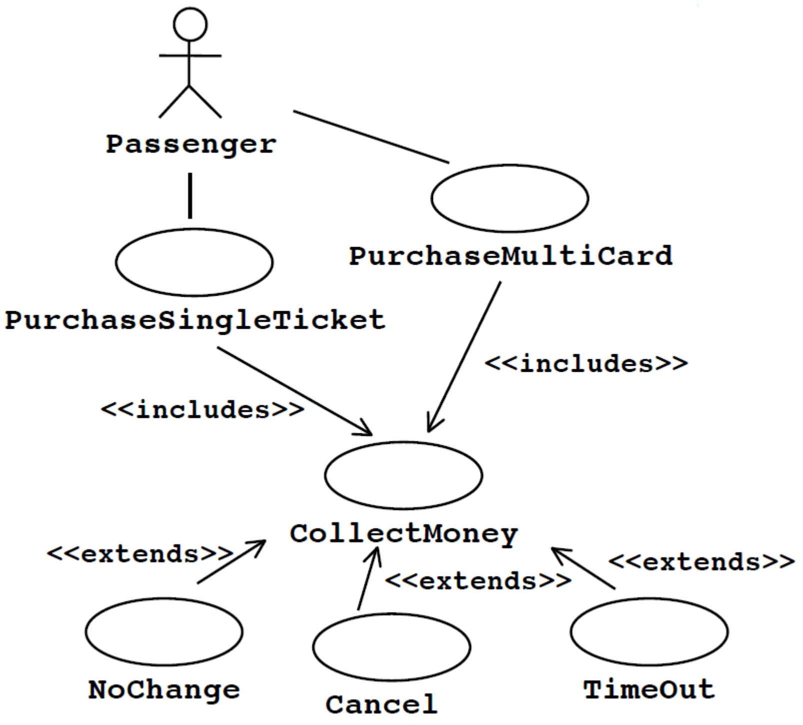
Introduction to UML

UML –Use Cases relationships



Introduction to UML

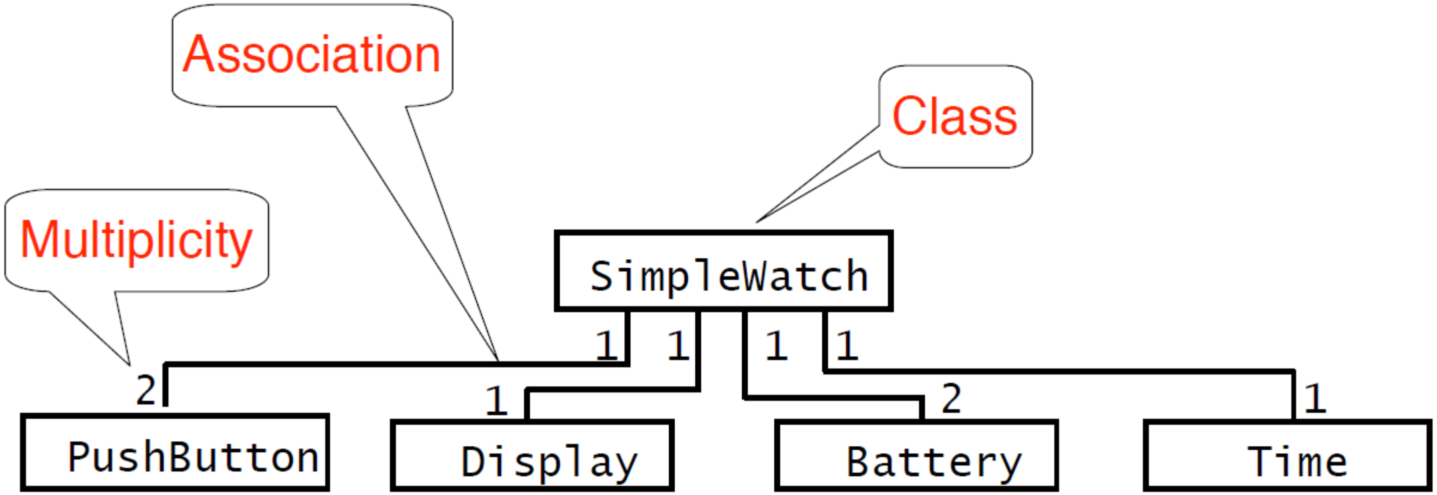
UML –Use Cases relationships



- `<<includes>>` relationship represents common functionality needed in more than one use case
- `<<includes>>` behavior is factored out for reuse, not because it is an exception
- The direction of an `<<includes>>` relationship is to the using use case (unlike the direction of the `<<extends>>` relationship).

Introduction to UML

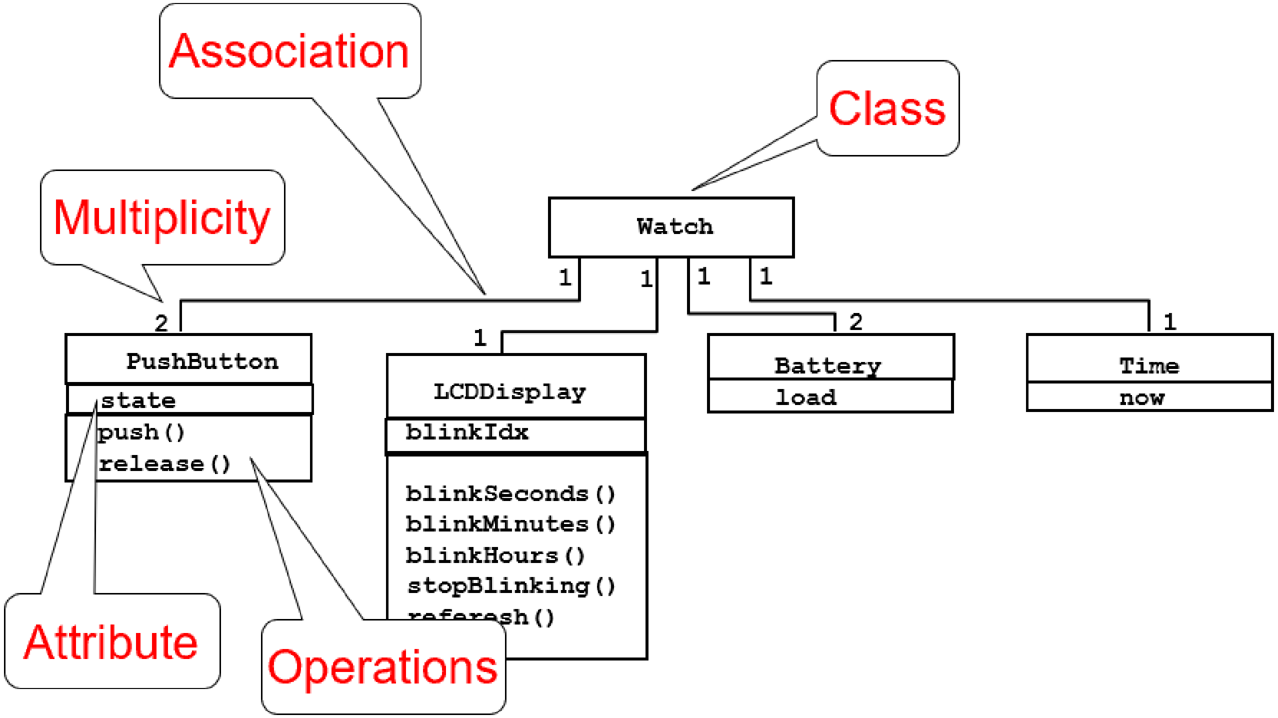
UML – A better understanding of the problem – the structure



Class diagram represents the structure of the system

Introduction to UML

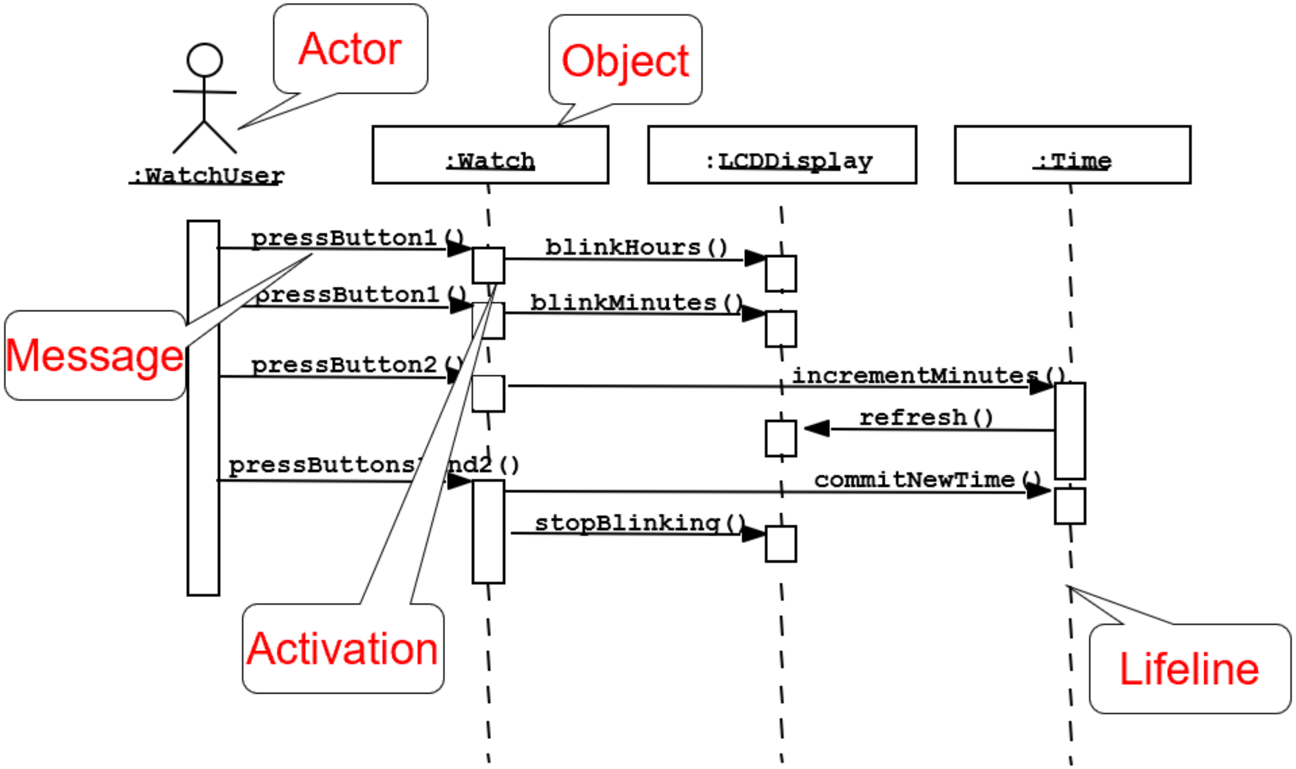
UML – A better understanding of the problem – the structure_2



Class diagram represents the structure of the system

Introduction to UML

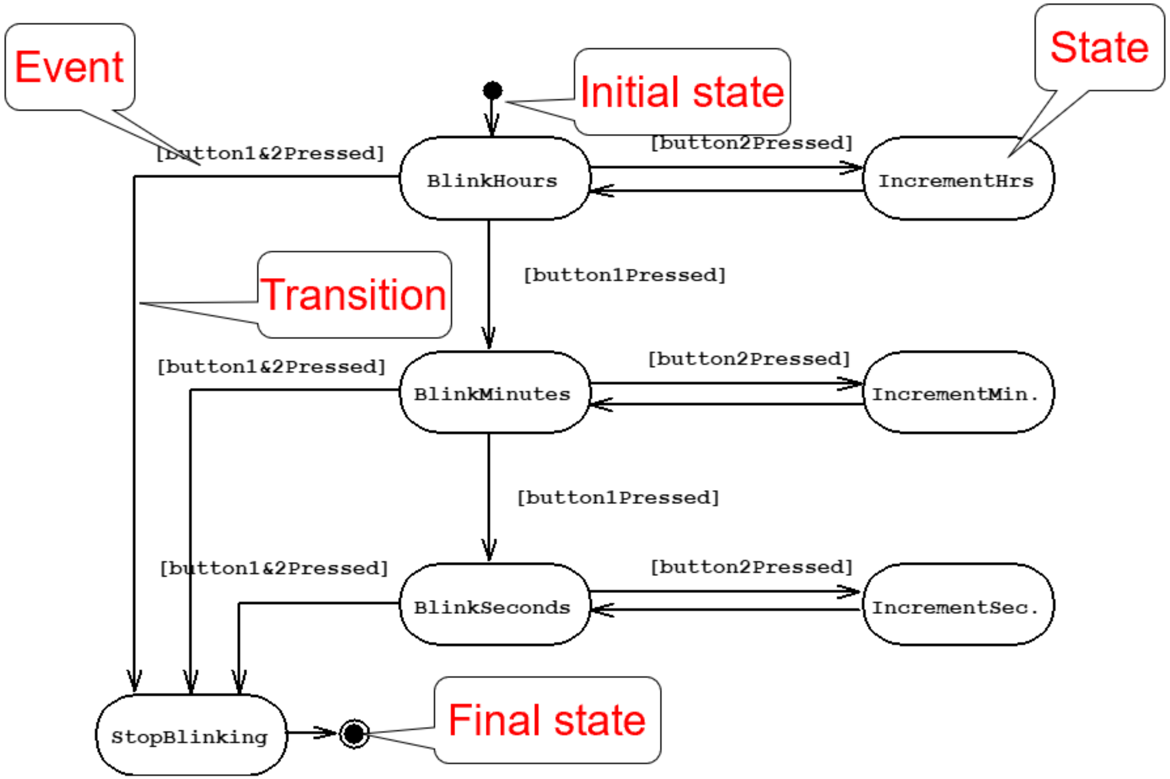
UML – A better understanding of the problem – the behavior



Sequence diagrams describe behavior as interactions between objects

Introduction to UML

UML – A better understanding of the problem – the behavior_2

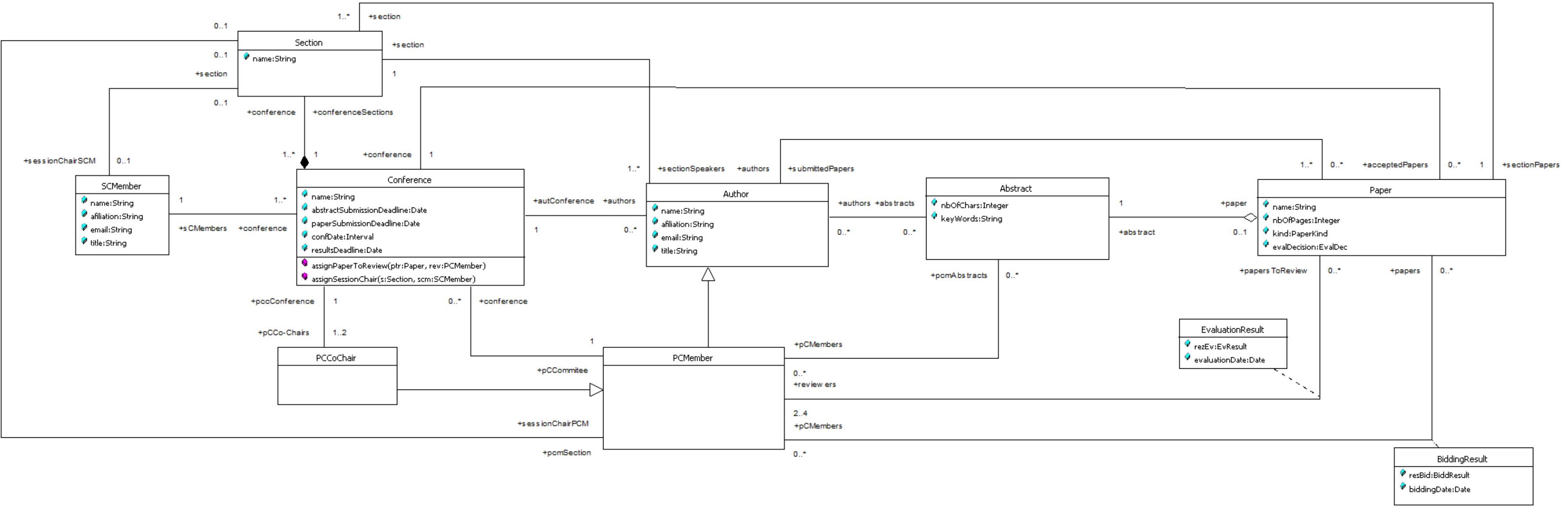


States transitions diagrams describe behavior by means of states and transitions

Introduction to UML

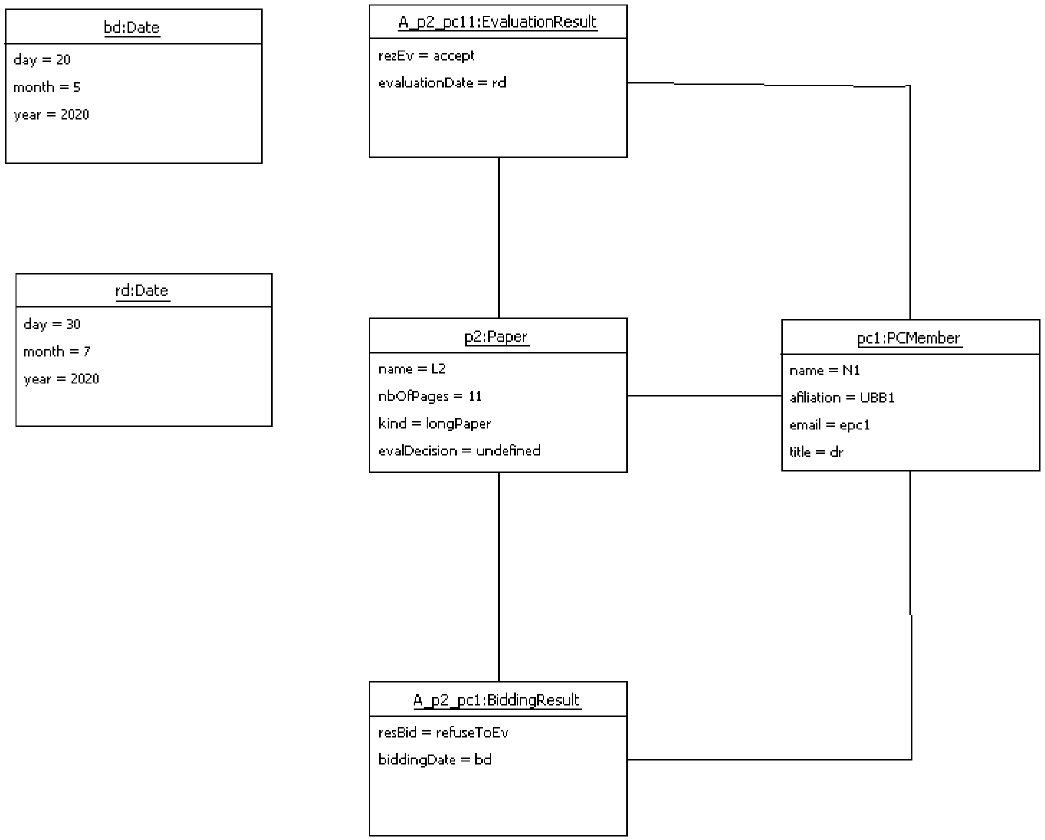
UML – Advantages of using Class Diagram

```
context PCMember
  inv approprPapToReview:
      self.papersToReview->select(p:Paper | Set{BiddResult::conflict,
BiddResult::refuseToEv}->includes(p.biddingResult->any(br|
br.pCMembers->includes(self)).resBid))->isEmpty and
        self.papersToReview.authors->excludes(self.oclAsType(Author))
```



Introduction to UML

UML – Object Diagram Snapshot



bd:Date

day = 20
month = 5
year = 2020

rd:Date

day = 30
month = 7
year = 2020

A_p2_pc11:EvaluationResult

rezEv = accept
evaluationDate = rd

p2:Paper

name = L2
nbOfPages = 11
kind = longPaper
evalDecision = undefined

pc1:PCMember

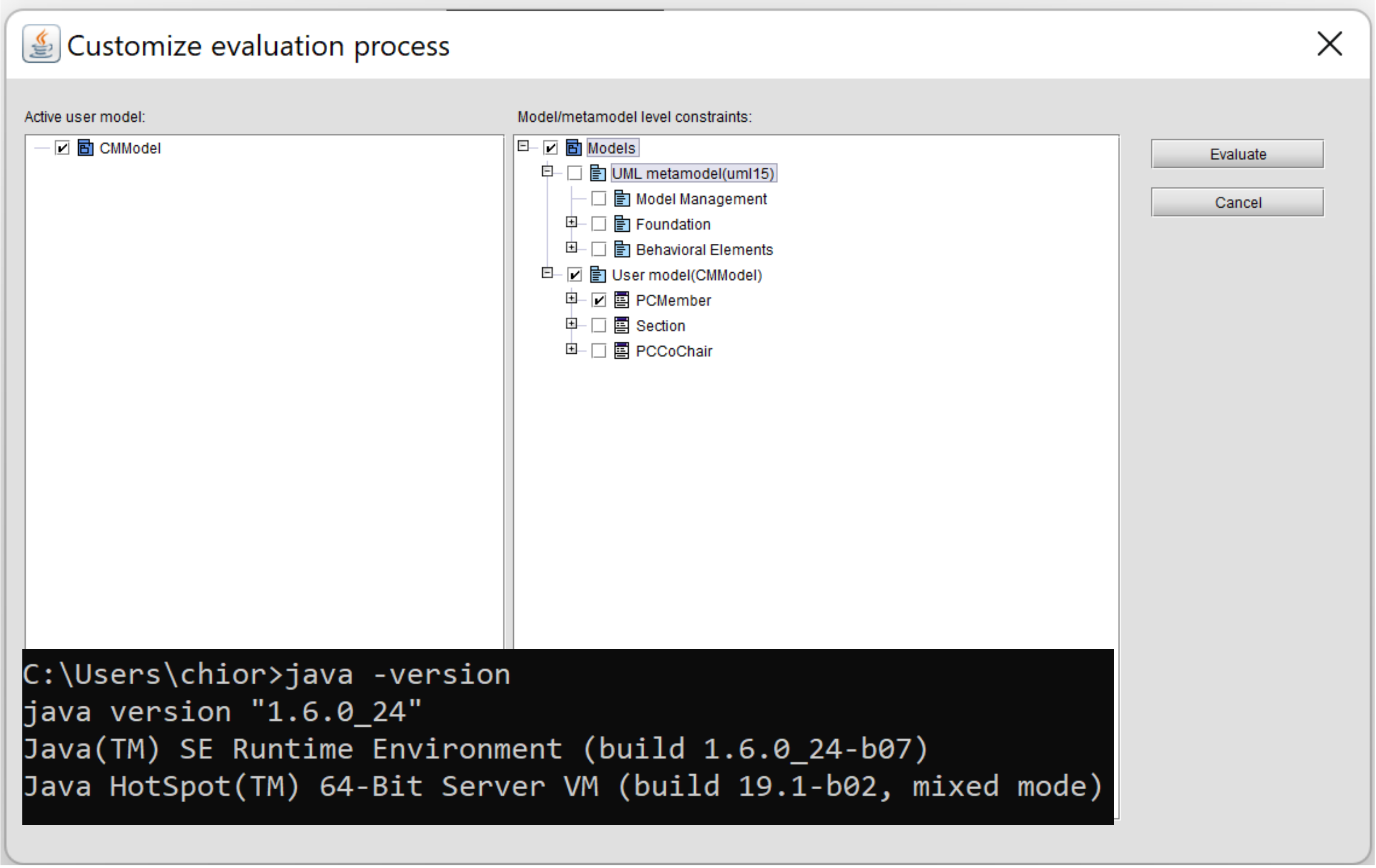
name = N1
afiliation = UBB1
email = epc1
title = dr

A_p2_pc1:BiddingResult

resBid = refuseToEv
biddingDate = bd

Introduction to UML

UML – OCLE Customize Evaluation



Customize evaluation process

Active user model:

- ☑ CMModel

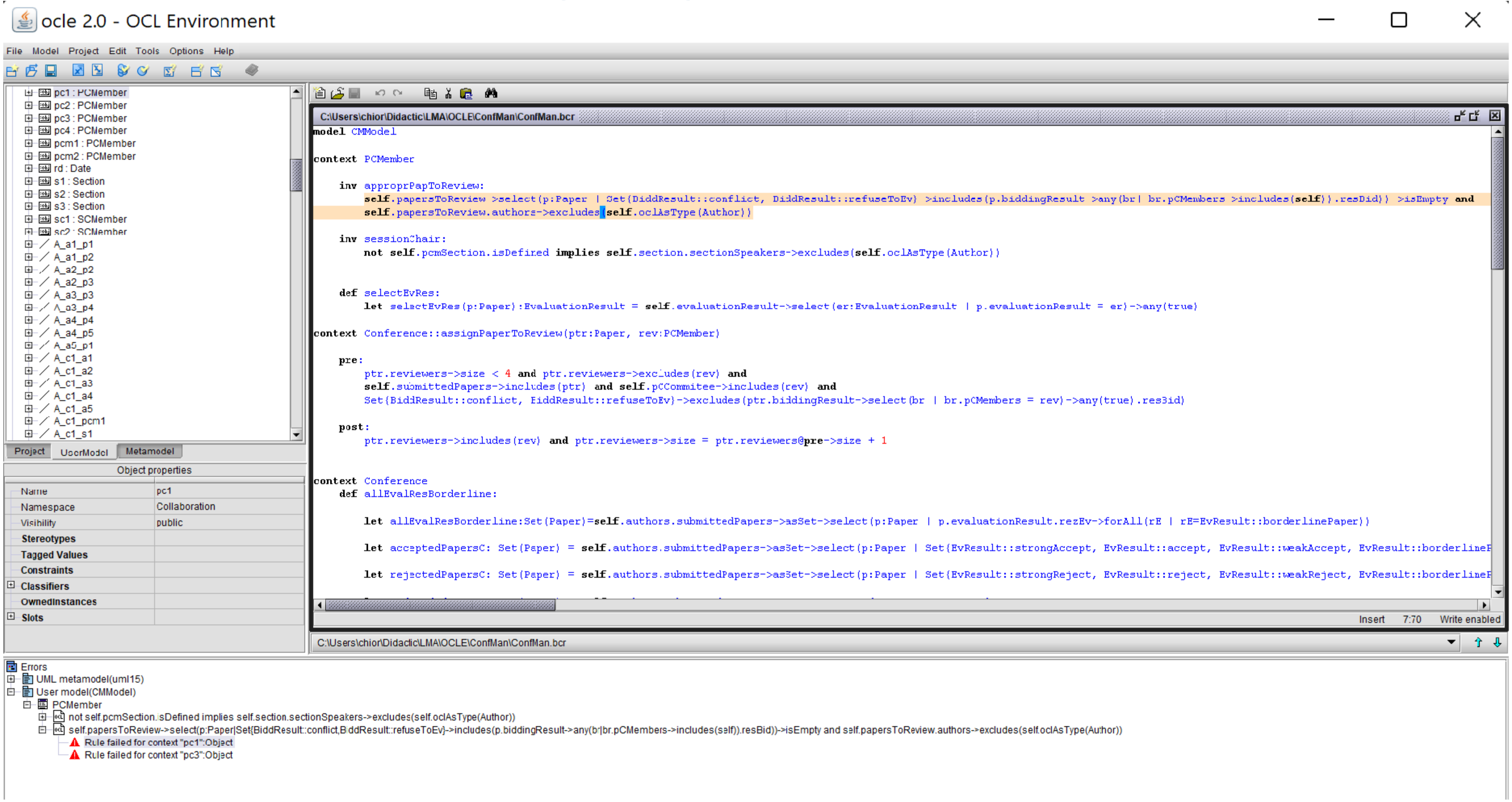
Model/metamodel level constraints:

- ☑ Models
  - □ UML metamodel(uml15)
    - □ Model Management
    - □ Foundation
    - □ Behavioral Elements
  - ☑ User model(CMModel)
    - ☑ PCMember
    - □ Section
    - □ PCCoChair

```
C:\Users\chior>java -version
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
```

Introduction to UML

UML – Model Checking using OCLE



Name	pc1
Namespace	Collaboration
Visibility	public
Stereotypes	
Tagged Values	
Constraints	
Classifiers	
OwnedInstances	
Slots	

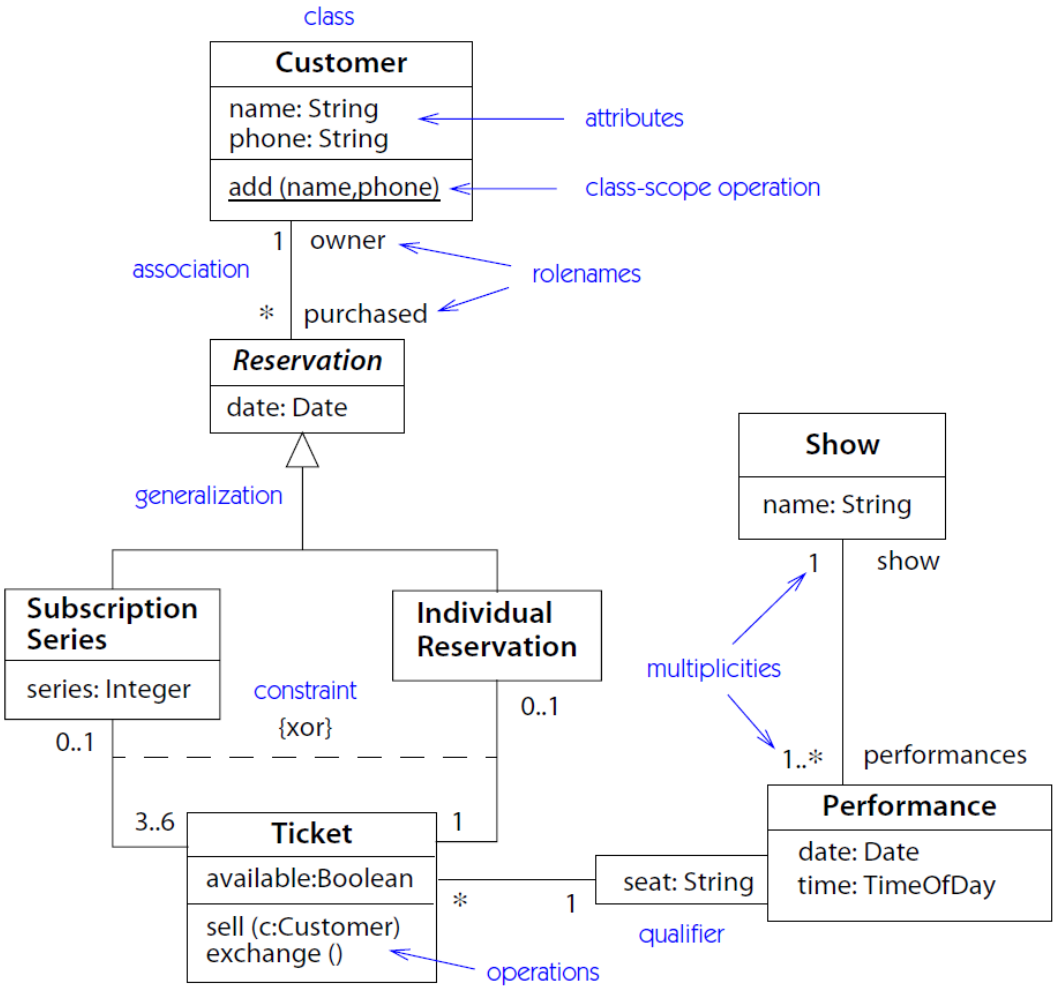
```
model CMModel

context PCMember

    inv approprPapToReview:
        self.papersToReview->select(p:Paper | Set{BiddResult::conflict, BiddResult::refuseToEv}->includes(p.biddingResult->any(br| br.pCMembers->includes(self)).resBid))->isEmpty and
        self.papersToReview.authors->excludes(self.oclAsType(Author))

    inv sessionChair:
        not self.pcmSection.isDefined implies self.section.sectionSpeakers->excludes(self.oclAsType(Author))


    def selectEvRes:
        let selectEvRes(p:Paper):EvaluationResult = self.evaluationResult->select(er:EvaluationResult | p.evaluationResult = er)->any(true)

context Conference::assignPaperToReview(ptr:Paper, rev:PCMember)

    pre:
        ptr.reviewers->size < 4 and ptr.reviewers->excludes(rev) and
        self.submittedPapers->includes(ptr) and self.pCCommitee->includes(rev) and
        Set{BiddResult::conflict, BiddResult::refuseToEv}->excludes(ptr.biddingResult->select(br | br.pCMembers = rev)->any(true).resBid)

    post:
        ptr.reviewers->includes(rev) and ptr.reviewers->size = ptr.reviewers@pre->size + 1


context Conference
    def allEvalResBorderline:

        let allEvalResBorderline:Set(Paper)=self.authors.submittedPapers->asSet->select(p:Paper | p.evaluationResult.rezEv->forAll(rE | rE=EvResult::borderlinePaper))

        let acceptedPapersC: Set(Paper) = self.authors.submittedPapers->asSet->select(p:Paper | Set{EvResult::strongAccept, EvResult::accept, EvResult::weakAccept, EvResult::borderlineP

        let rejectedPapersC: Set(Paper) = self.authors.submittedPapers->asSet->select(p:Paper | Set{EvResult::strongReject, EvResult::reject, EvResult::weakReject, EvResult::borderlineP
```

Insert 7:70 Write enabled
C:\Users\chior\Didactic\LMA\OCLE\ConfMan\ConfMan.bcr

```
Errors
  UML metamodel(uml15)
  User model(CMModel)
    PCMember
      not self.pcmSection.isDefined implies self.section.sectionSpeakers->excludes(self.oclAsType(Author))
      self.papersToReview->select(p:Paper|Set{BiddResult::conflict,BiddResult::refuseToEv}->includes(p.biddingResult->any(br|br.pCMembers->includes(self)).resBid))->isEmpty and self.papersToReview.authors->excludes(self.oclAsType(Author))
        Rule failed for context "pc1":Object
        Rule failed for context "pc3":Object
```

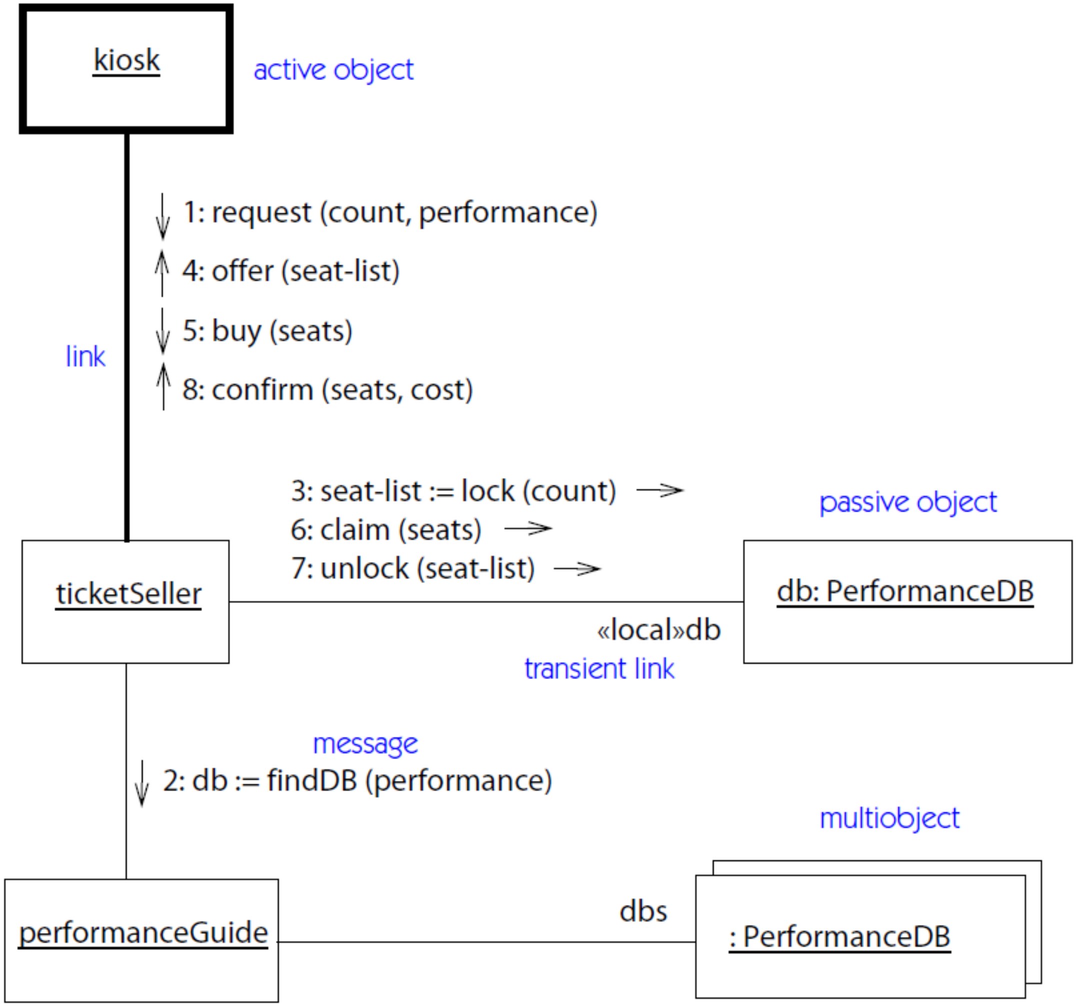
Introduction to UML

UML – CD qualified associations & constraints



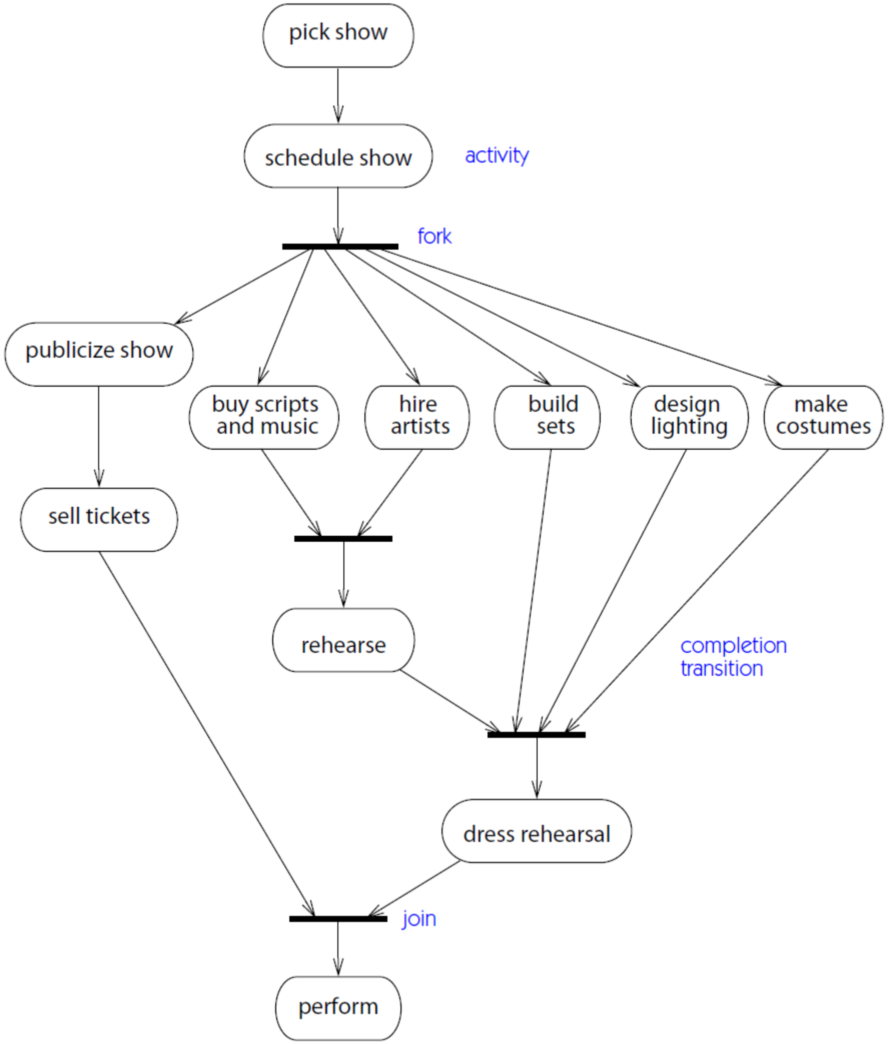
Introduction to UML

UML – Collaboration Diagram



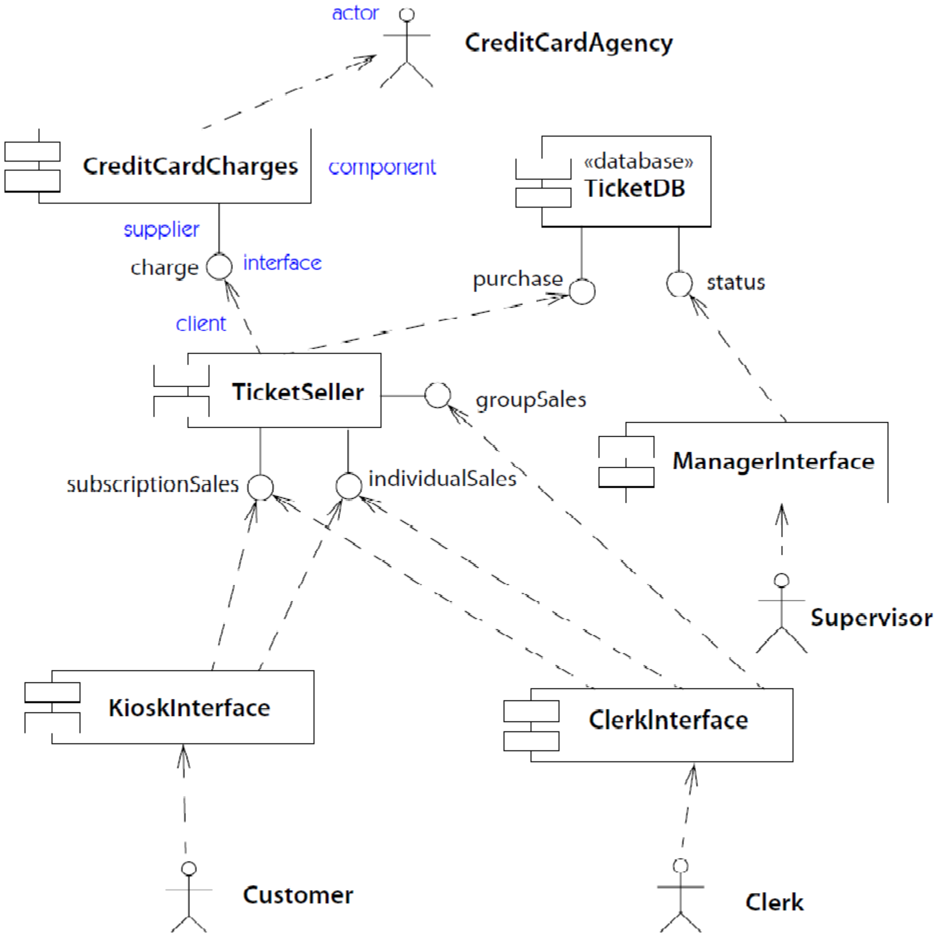
Introduction to UML

UML – Activity Diagram



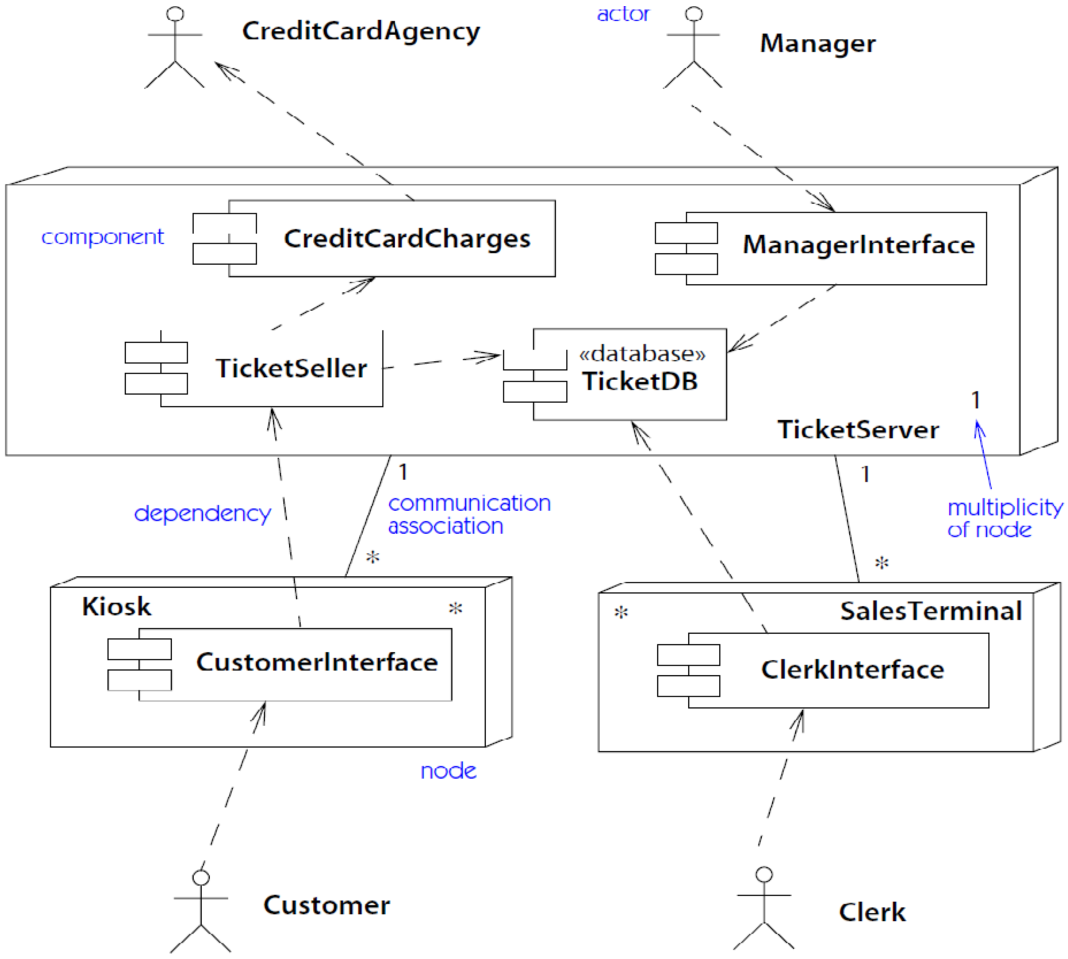
Introduction to UML

UML – Component Diagram



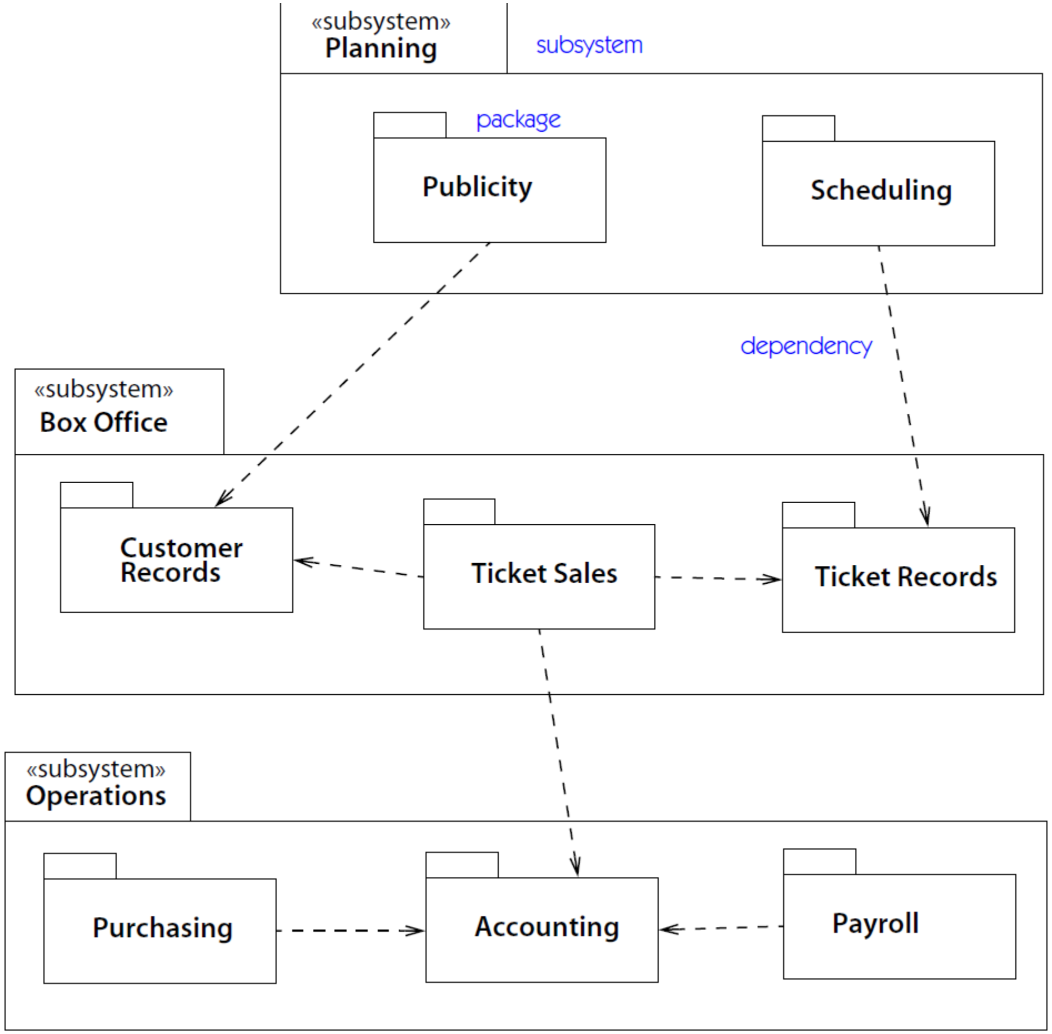
Introduction to UML

UML – Deployment Diagram



Introduction to UML

UML – System Architecture



Introduction to UML

UML – definition

Language definition1.5.1

- concrete syntax
- abstract syntax (metamodel + constraints)
- semantics

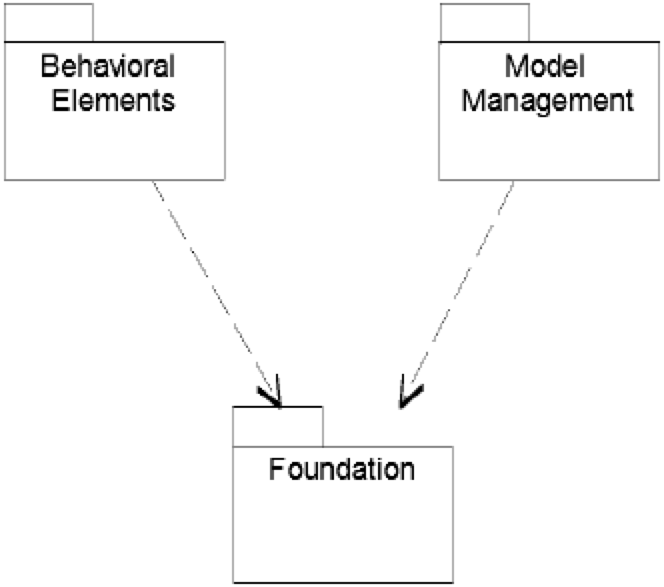


Figure 2-1 Top-Level Packages

Introduction to UML

UML – definition; Foundation Packages

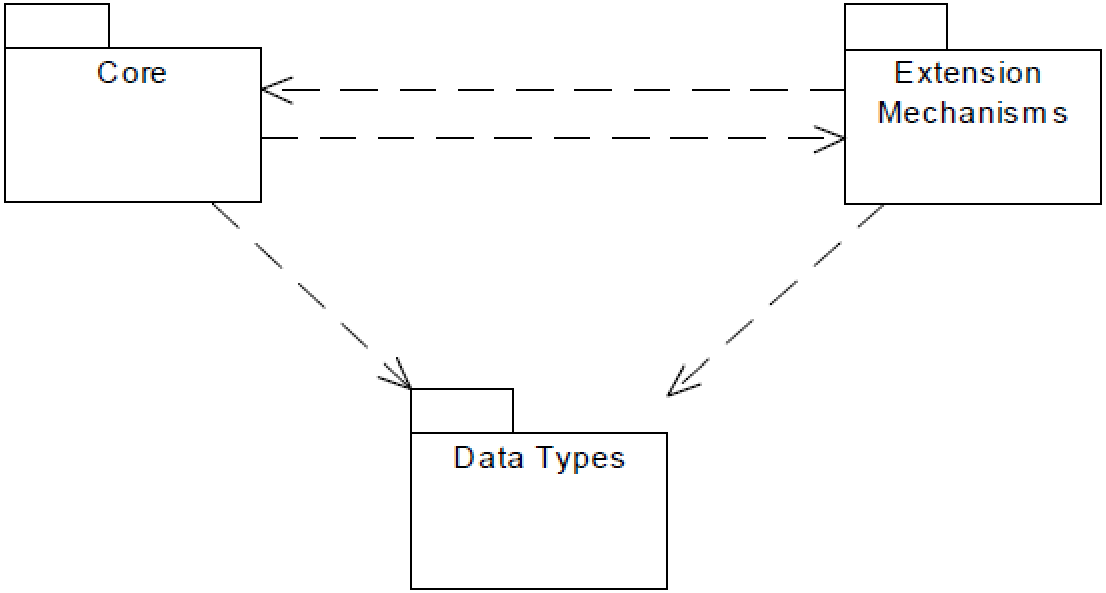


Figure 2-2 Foundation Packages

Introduction to UML

UML – definition; Behavioral Elements Packages

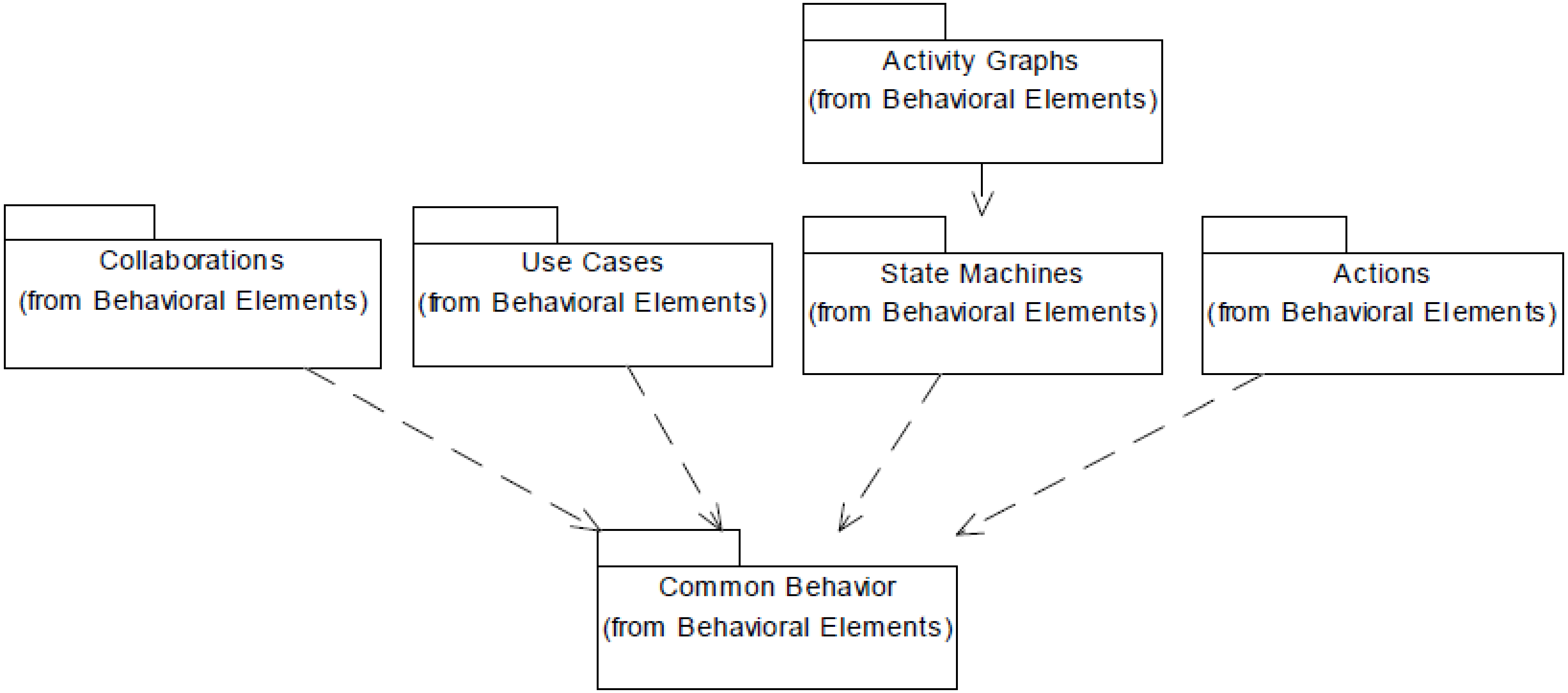

Activity Graphs
(from Behavioral Elements)
Collaborations
(from Behavioral Elements)
Use Cases
(from Behavioral Elements)
State Machines
(from Behavioral Elements)
Actions
(from Behavioral Elements)
Common Behavior
(from Behavioral Elements)

Figure 2-3 Behavioral Elements Packages

Introduction to UML

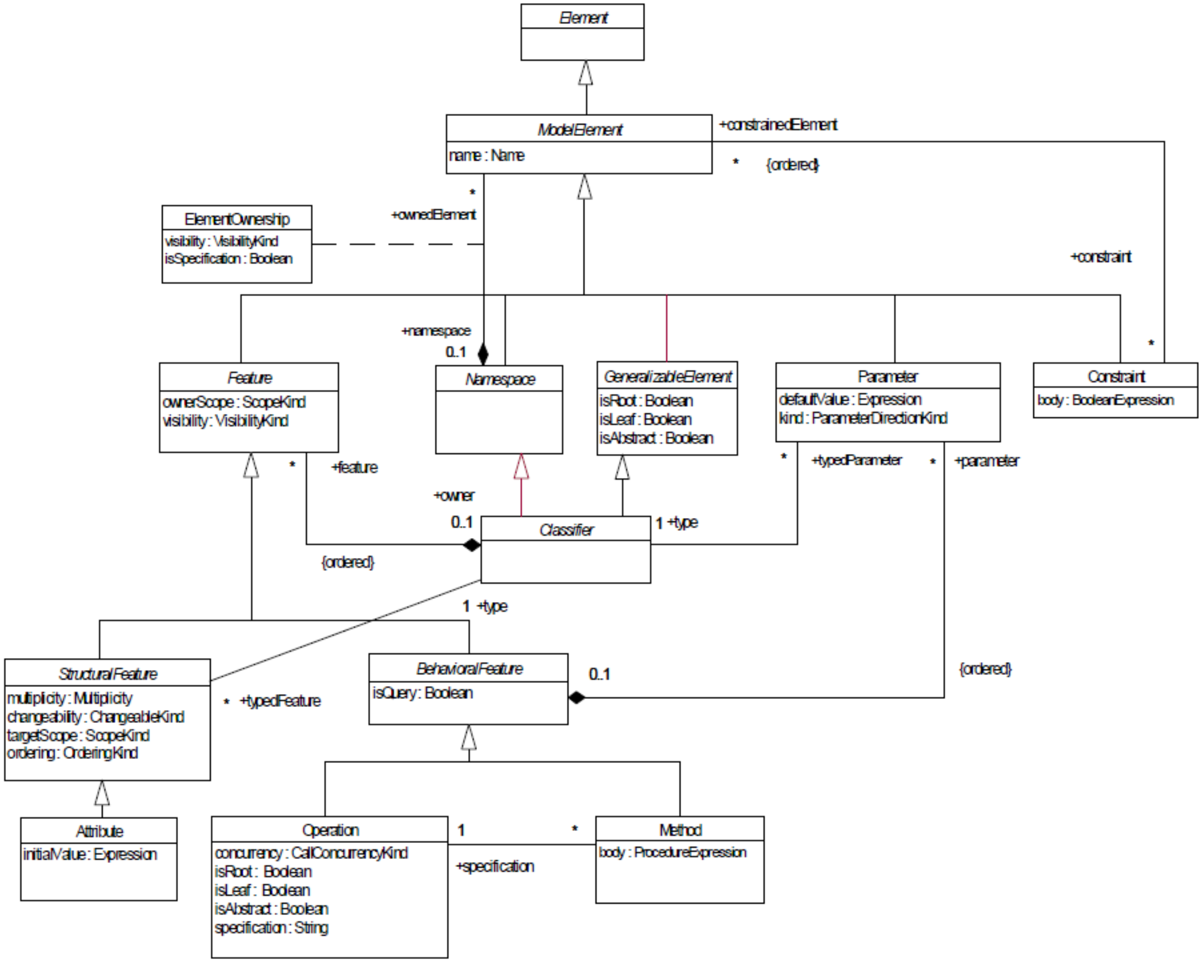
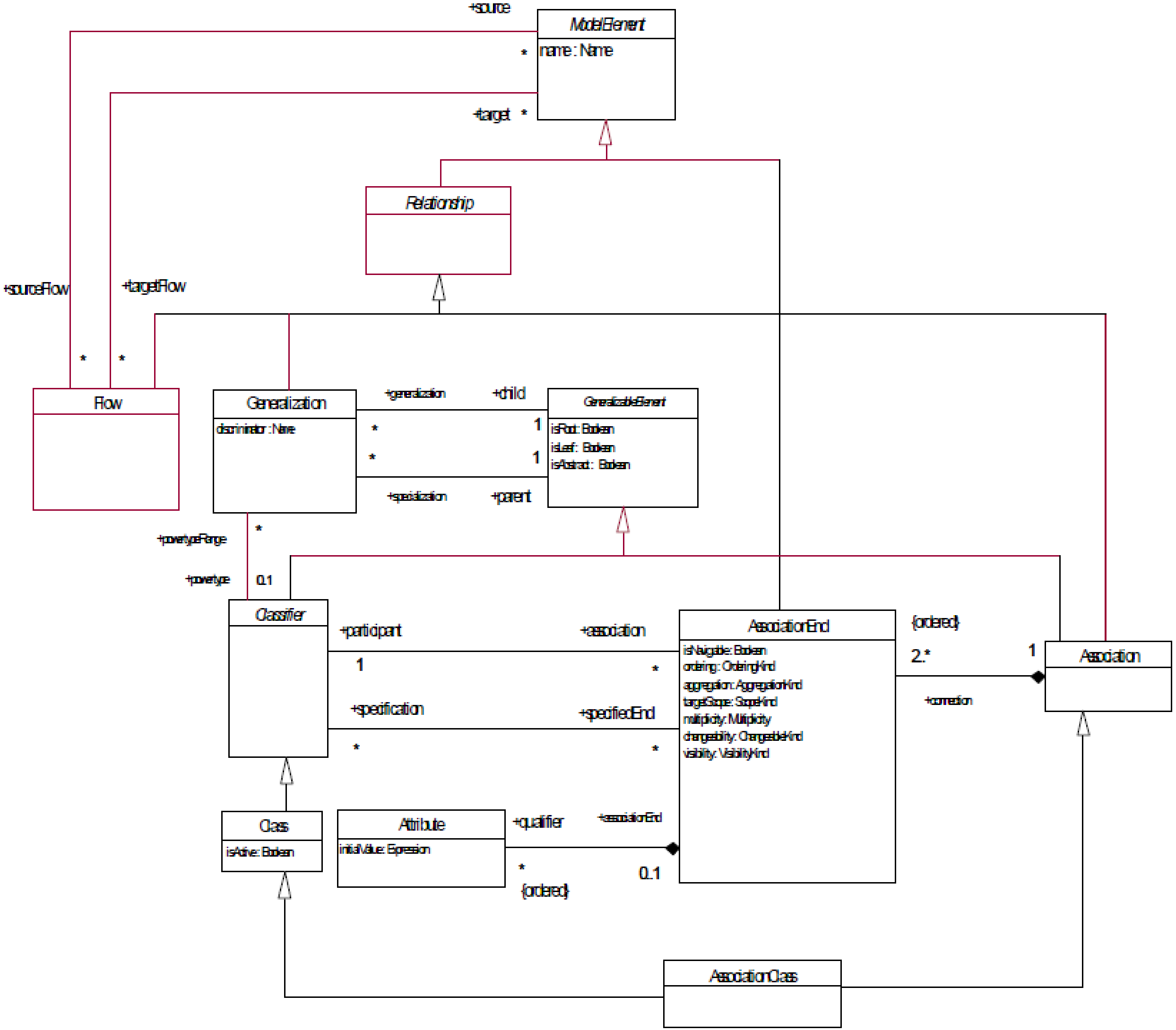


Figure 2-5 Core Package - Backbone

Introduction to UML



Introduction to UML

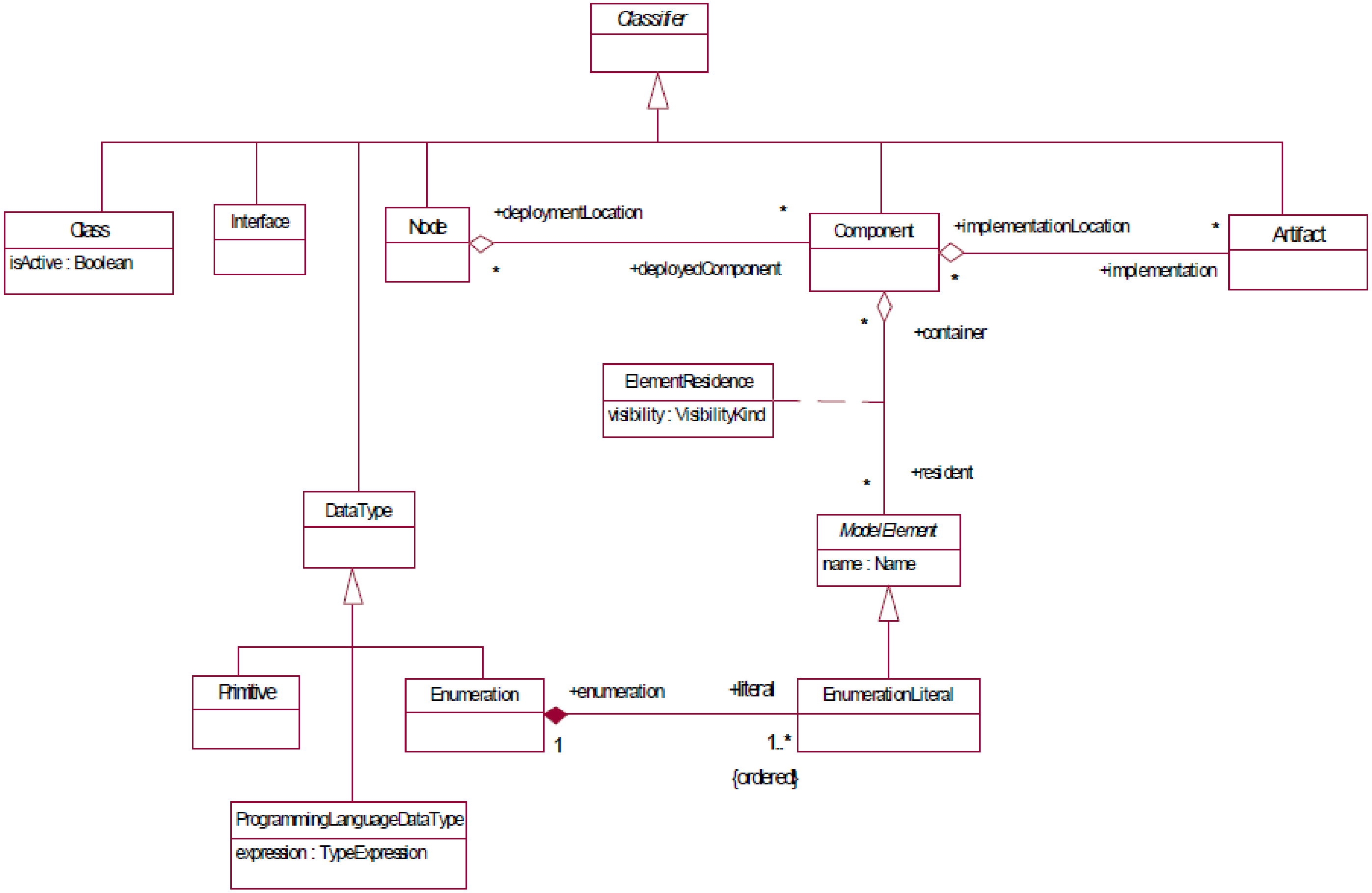
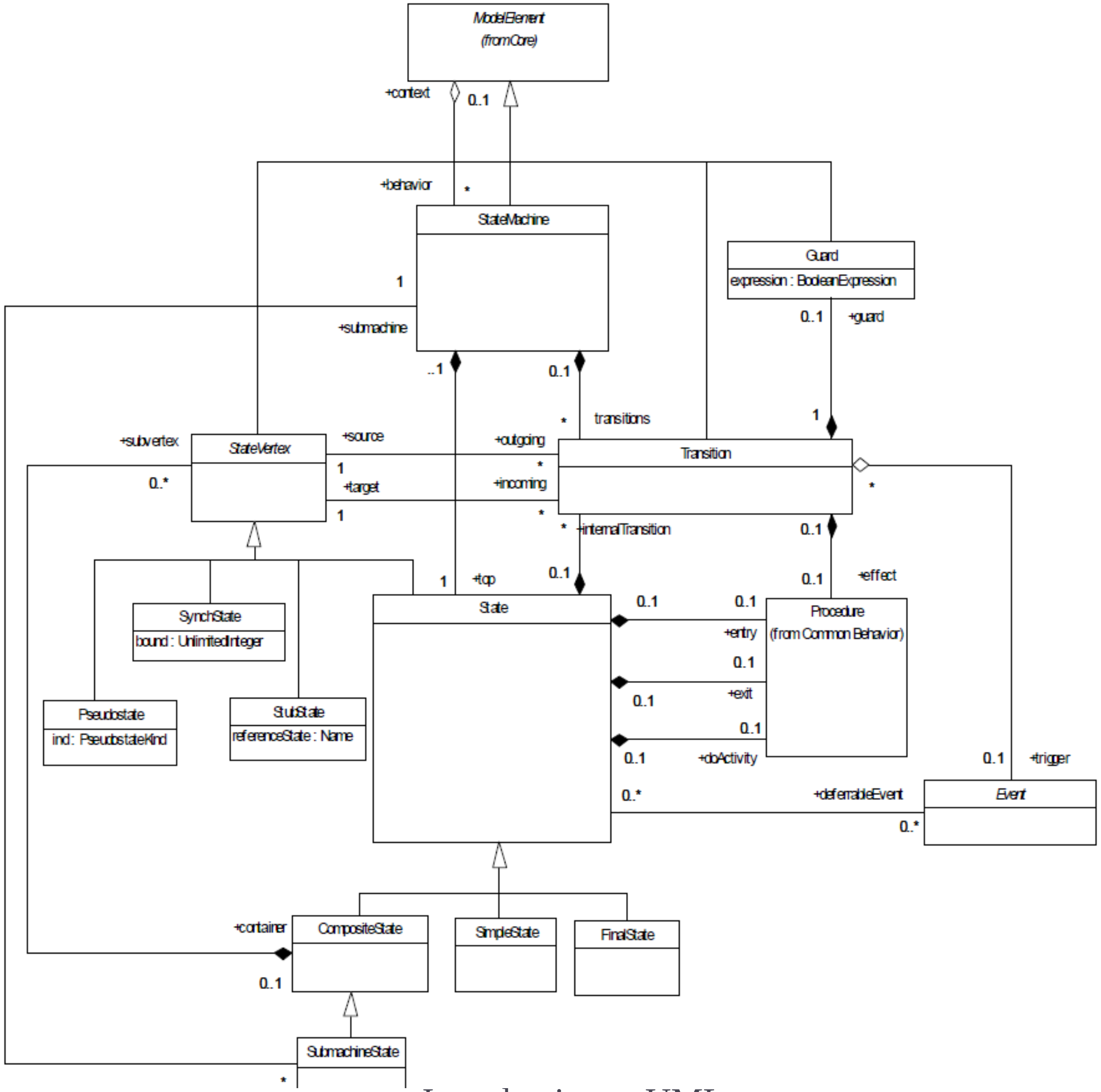
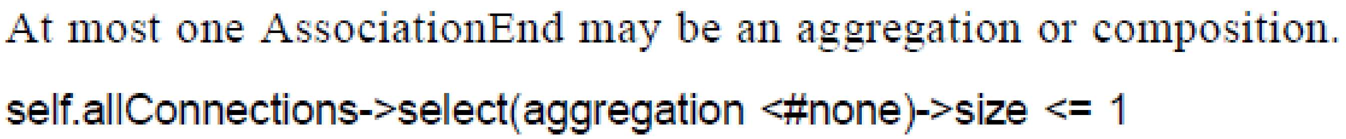


Figure 2-8 Core Package - Classifiers

Introduction to UML

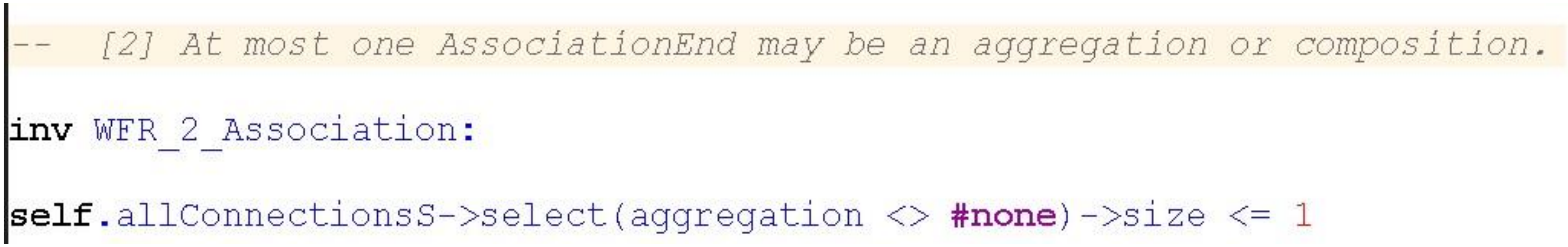


Introduction to UML

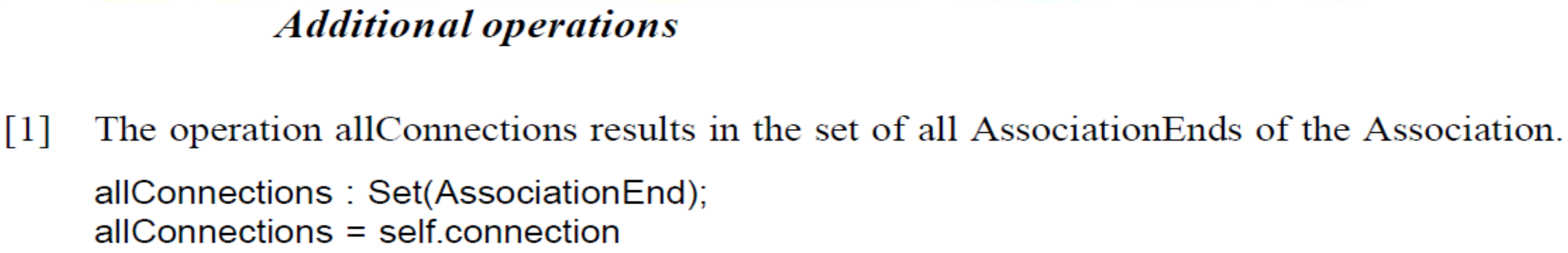


At most one AssociationEnd may be an aggregation or composition.

```
self.allConnections->select(aggregation <#none)->size <= 1
```



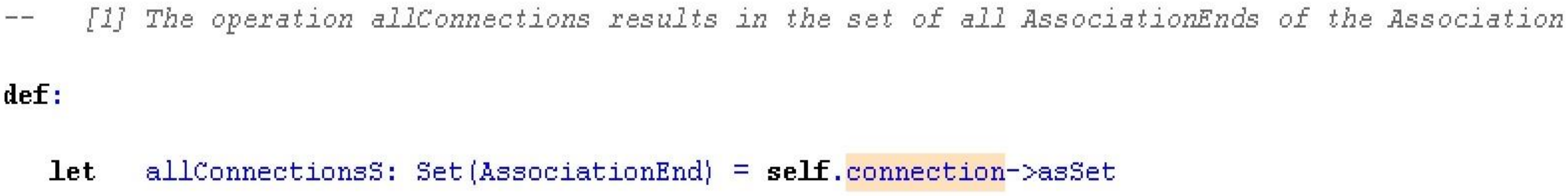
```
--   [2] At most one AssociationEnd may be an aggregation or composition.

inv WFR_2_Association:

self.allConnectionsS->select(aggregation <> #none)->size <= 1
```



Additional operations

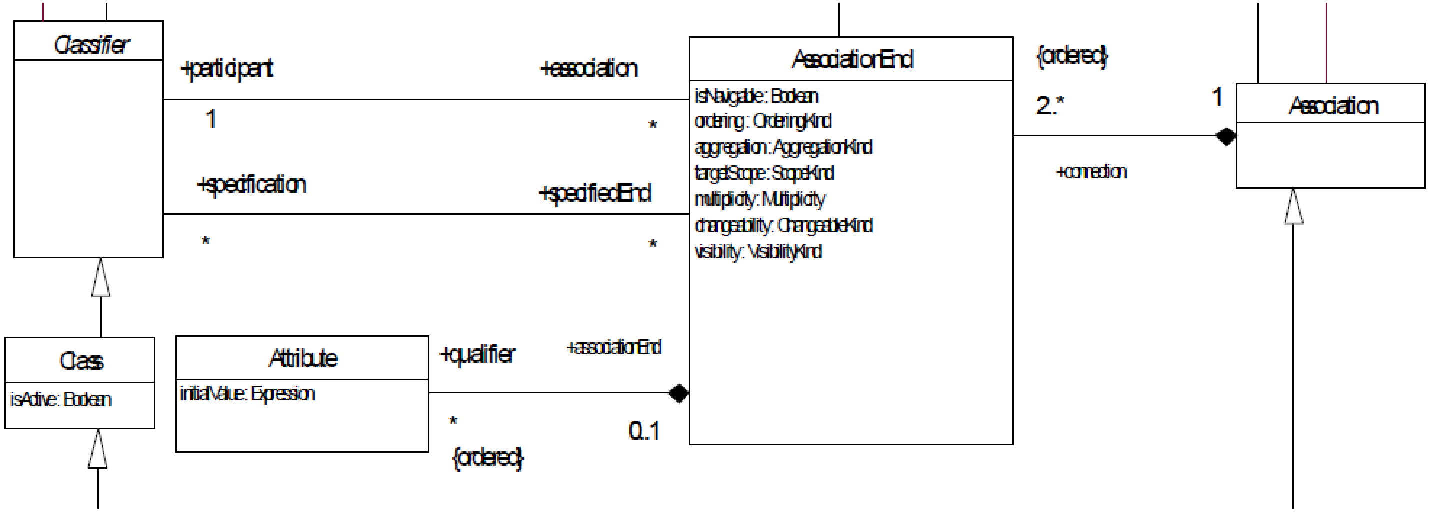
[1] The operation allConnections results in the set of all AssociationEnds of the Association.

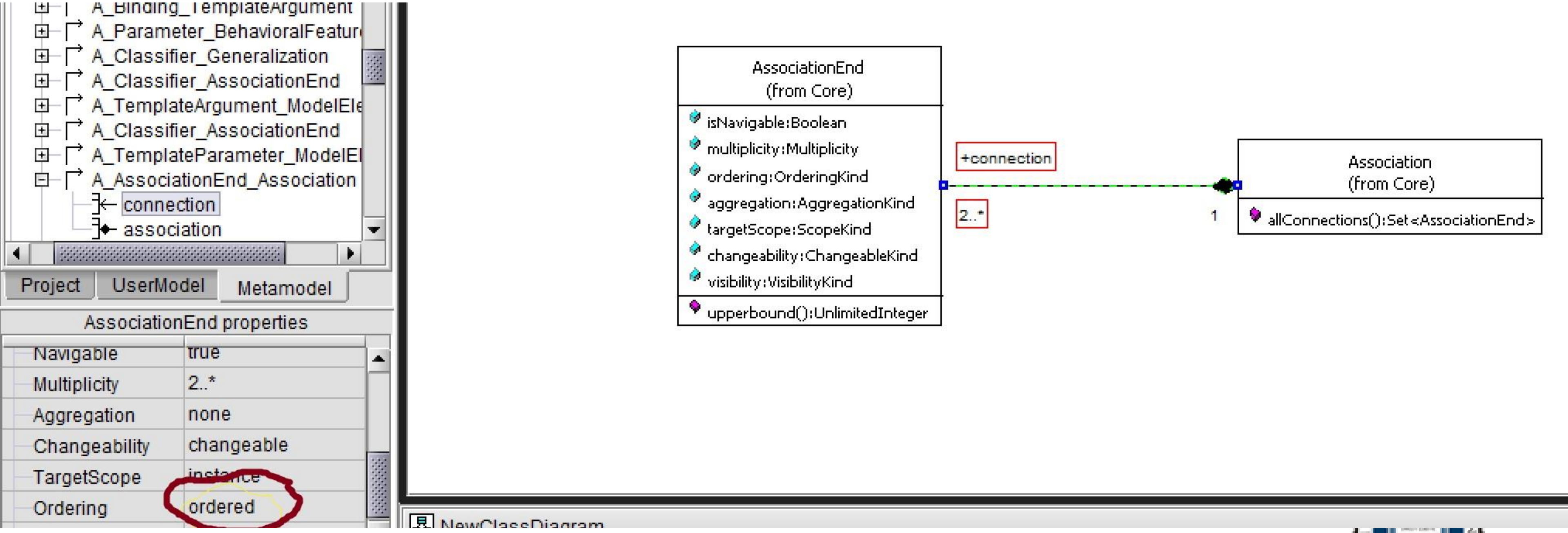
```
allConnections : Set(AssociationEnd);
allConnections = self.connection
```



```
--   [1] The operation allConnections results in the set of all AssociationEnds of the Association

def:

   let   allConnectionsS: Set(AssociationEnd) = self.connection->asSet
```

Introduction to UML





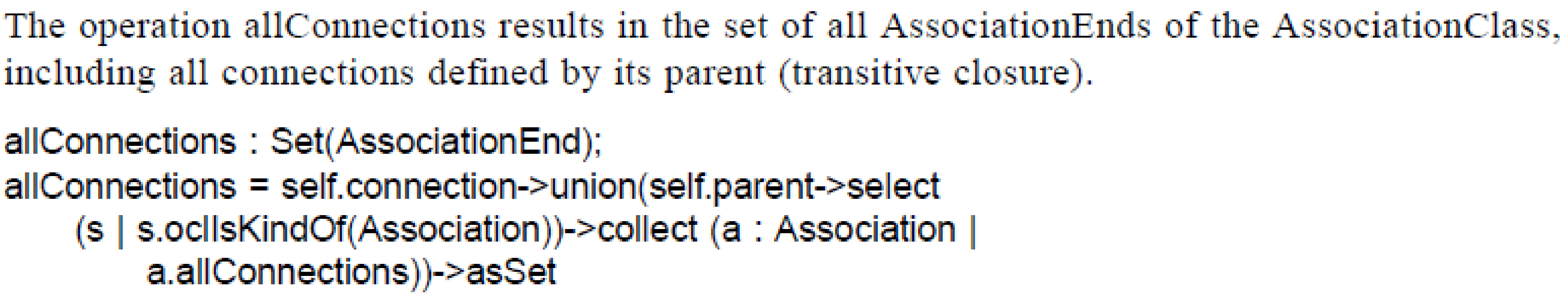
A_Binding_TemplateArgument
A_Parameter_BehavioralFeatur
A_Classifier_Generalization
A_Classifier_AssociationEnd
A_TemplateArgument_ModelEle
A_Classifier_AssociationEnd
A_TemplateParameter_ModelEl
A_AssociationEnd_Association
connection
association

AssociationEnd properties

Navigable	true
Multiplicity	2..*
Aggregation	none
Changeability	changeable
TargetScope	instance
Ordering	ordered

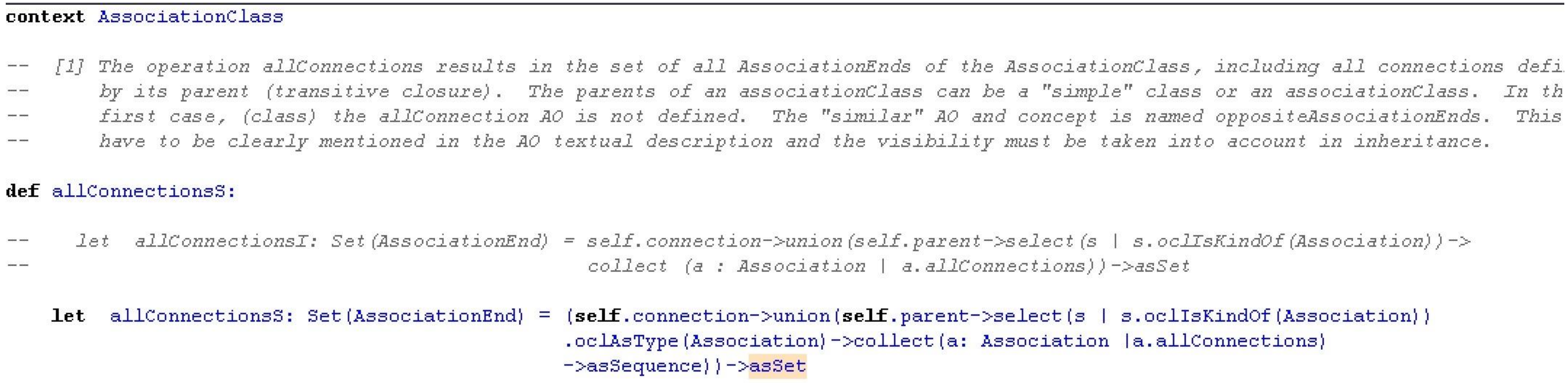
AssociationEnd
(from Core)
isNavigable:Boolean
multiplicity:Multiplicity
ordering:OrderingKind
aggregation:AggregationKind
targetScope:ScopeKind
changeability:ChangeableKind
visibility:VisibilityKind
upperbound():UnlimitedInteger
+connection
2..*
1
Association
(from Core)
allConnections():Set<AssociationEnd>
NewClassDiagram

Introduction to UML



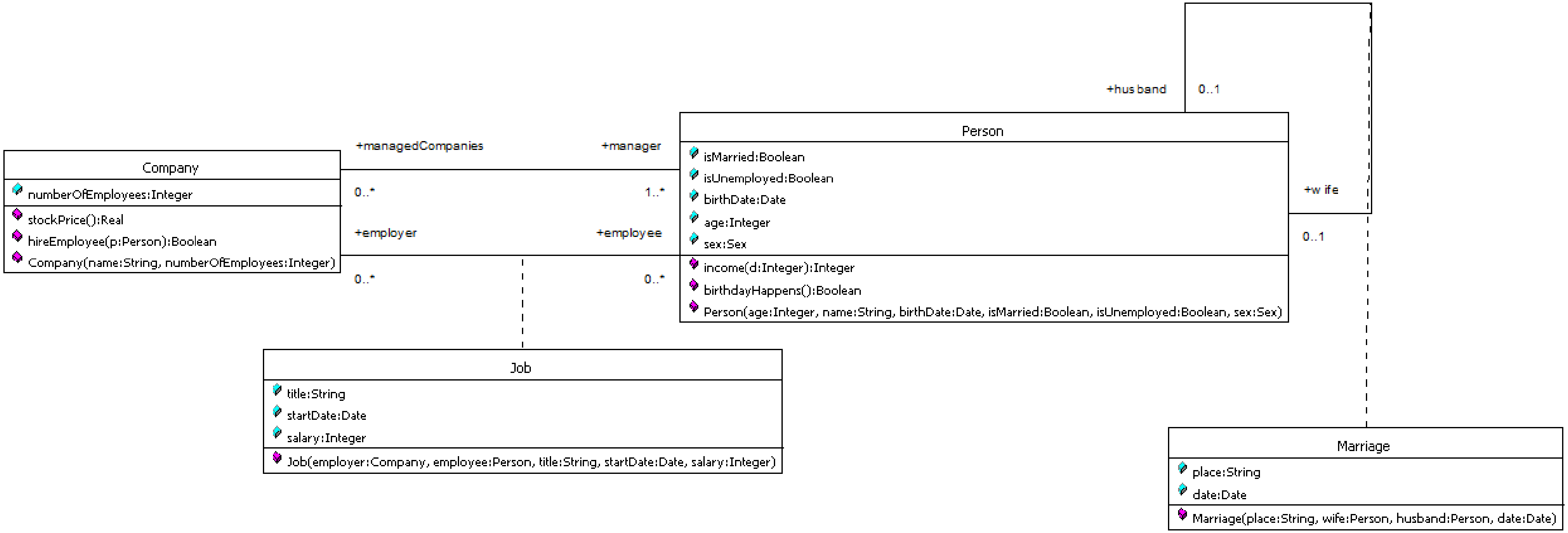
The operation allConnections results in the set of all AssociationEnds of the AssociationClass, including all connections defined by its parent (transitive closure).

```
allConnections : Set(AssociationEnd);
allConnections = self.connection->union(self.parent->select
     (s | s.oclIsKindOf(Association))->collect (a : Association |
          a.allConnections))->asSet
```



```
context AssociationClass

--  [1] The operation allConnections results in the set of all AssociationEnds of the AssociationClass, including all connections defi
--      by its parent (transitive closure).  The parents of an associationClass can be a "simple" class or an associationClass.  In th
--      first case, (class) the allConnection AO is not defined.  The "similar" AO and concept is named oppositeAssociationEnds.  This
--      have to be clearly mentioned in the AO textual description and the visibility must be taken into account in inheritance.

def allConnectionsS:

--    let  allConnectionsI: Set(AssociationEnd) = self.connection->union(self.parent->select(s | s.oclIsKindOf(Association))->
--                                                collect (a : Association | a.allConnections))->asSet

    let  allConnectionsS: Set(AssociationEnd) = (self.connection->union(self.parent->select(s | s.oclIsKindOf(Association))
                                                .oclAsType(Association)->collect(a: Association |a.allConnections)
                                                ->asSequence))->asSet
```

Introduction to UML



Company

numberOfEmployees:Integer

stockPrice():Real

hireEmployee(p:Person):Boolean

Company(name:String, numberOfEmployees:Integer)

+managedCompanies

+manager

0..*

1..*

+employer

+employee

0..*

0..*

Person

isMarried:Boolean

isUnemployed:Boolean

birthDate:Date

age:Integer

sex:Sex

income(d:Integer):Integer

birthdayHappens():Boolean

Person(age:Integer, name:String, birthDate:Date, isMarried:Boolean, isUnemployed:Boolean, sex:Sex)

+hus band

0..1

+w ife

0..1

Job

title:String

startDate:Date

salary:Integer

Job(employer:Company, employee:Person, title:String, startDate:Date, salary:Integer)

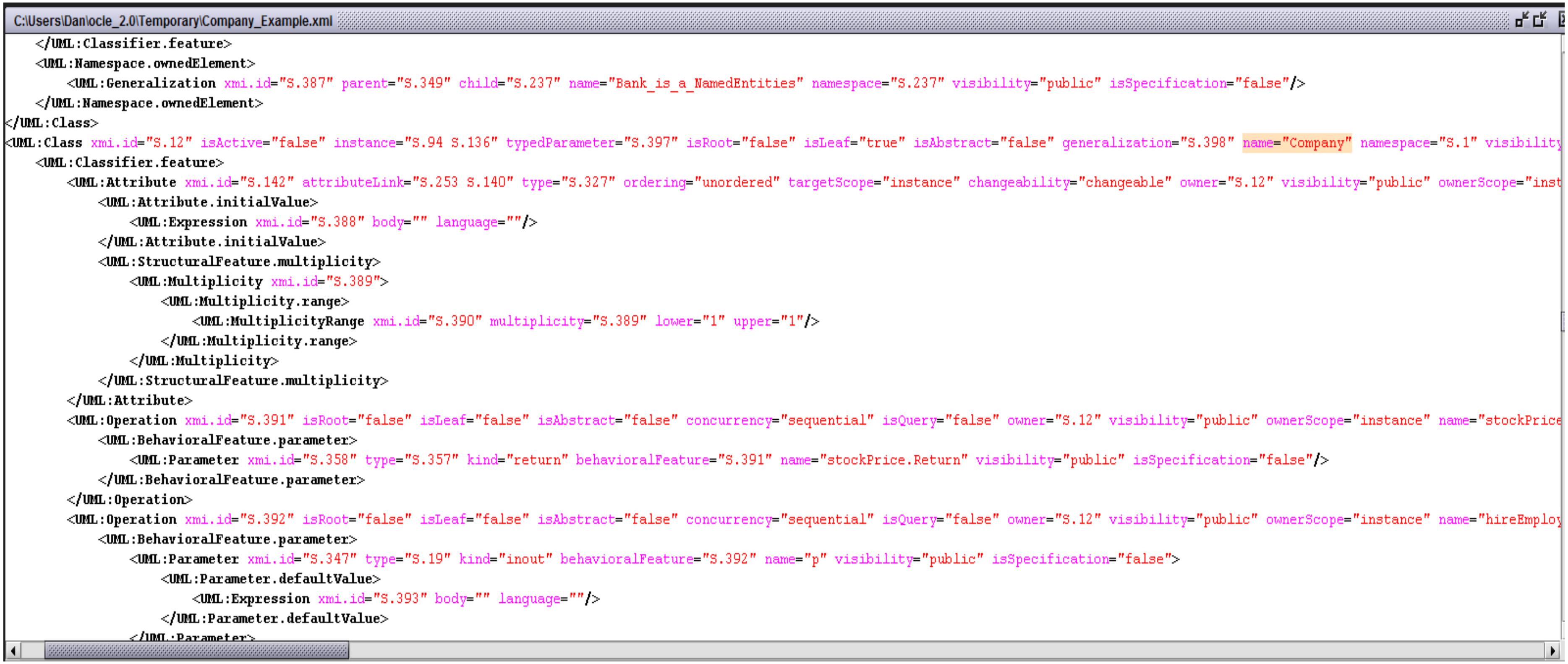
Marriage

place:String

date:Date

Marriage(place:String, wife:Person, husband:Person, date:Date)

Introduction to UML



```
    </UML:Classifier.feature>
    <UML:Namespace.ownedElement>
        <UML:Generalization xmi.id="S.387" parent="S.349" child="S.237" name="Bank_is_a_NamedEntities" namespace="S.237" visibility="public" isSpecification="false"/>
    </UML:Namespace.ownedElement>
</UML:Class>
<UML:Class xmi.id="S.12" isActive="false" instance="S.94 S.136" typedParameter="S.397" isRoot="false" isLeaf="true" isAbstract="false" generalization="S.398" name="Company" namespace="S.1" visibility
    <UML:Classifier.feature>
        <UML:Attribute xmi.id="S.142" attributeLink="S.253 S.140" type="S.327" ordering="unordered" targetScope="instance" changeability="changeable" owner="S.12" visibility="public" ownerScope="inst
            <UML:Attribute.initialValue>
                <UML:Expression xmi.id="S.388" body="" language=""/>
            </UML:Attribute.initialValue>
            <UML:StructuralFeature.multiplicity>
                <UML:Multiplicity xmi.id="S.389">
                    <UML:Multiplicity.range>
                        <UML:MultiplicityRange xmi.id="S.390" multiplicity="S.389" lower="1" upper="1"/>
                    </UML:Multiplicity.range>
                </UML:Multiplicity>
            </UML:StructuralFeature.multiplicity>
        </UML:Attribute>
        <UML:Operation xmi.id="S.391" isRoot="false" isLeaf="false" isAbstract="false" concurrency="sequential" isQuery="false" owner="S.12" visibility="public" ownerScope="instance" name="stockPrice
            <UML:BehavioralFeature.parameter>
                <UML:Parameter xmi.id="S.358" type="S.357" kind="return" behavioralFeature="S.391" name="stockPrice.Return" visibility="public" isSpecification="false"/>
            </UML:BehavioralFeature.parameter>
        </UML:Operation>
        <UML:Operation xmi.id="S.392" isRoot="false" isLeaf="false" isAbstract="false" concurrency="sequential" isQuery="false" owner="S.12" visibility="public" ownerScope="instance" name="hireEmploy
            <UML:BehavioralFeature.parameter>
                <UML:Parameter xmi.id="S.347" type="S.19" kind="inout" behavioralFeature="S.392" name="p" visibility="public" isSpecification="false">
                    <UML:Parameter.defaultValue>
                        <UML:Expression xmi.id="S.393" body="" language=""/>
                    </UML:Parameter.defaultValue>
                </UML:Parameter>
```

Introduction to UML

Thanks for your patience!