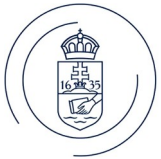
Software Technology 07

OOP Principles, Refactoring, Modeling

Why OOP?



– *Object Oriented Programming*

- What is it?
- Why are we using OOP?
- What are the tools for OOP?
- What is a good OOP design?
  - How can we evaluate?
  - How can we generate?

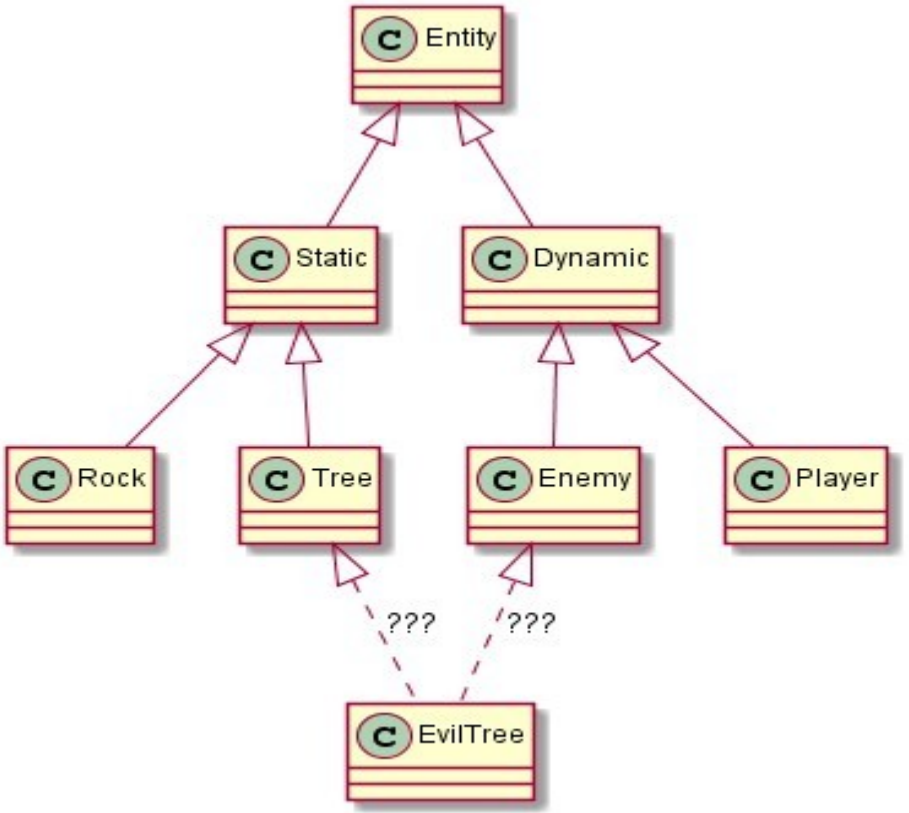
OOP Example


OpenTTD
OpenTTD 1.9.3
New Game
Load Game
Play Scenario
Play Heightmap
Scenario Editor
Multiplayer
Game Options
Advanced Settings
AI/Game Script Settings
NewGRF Settings
Check Online Content
Highscore Table
Exit

OOP Example

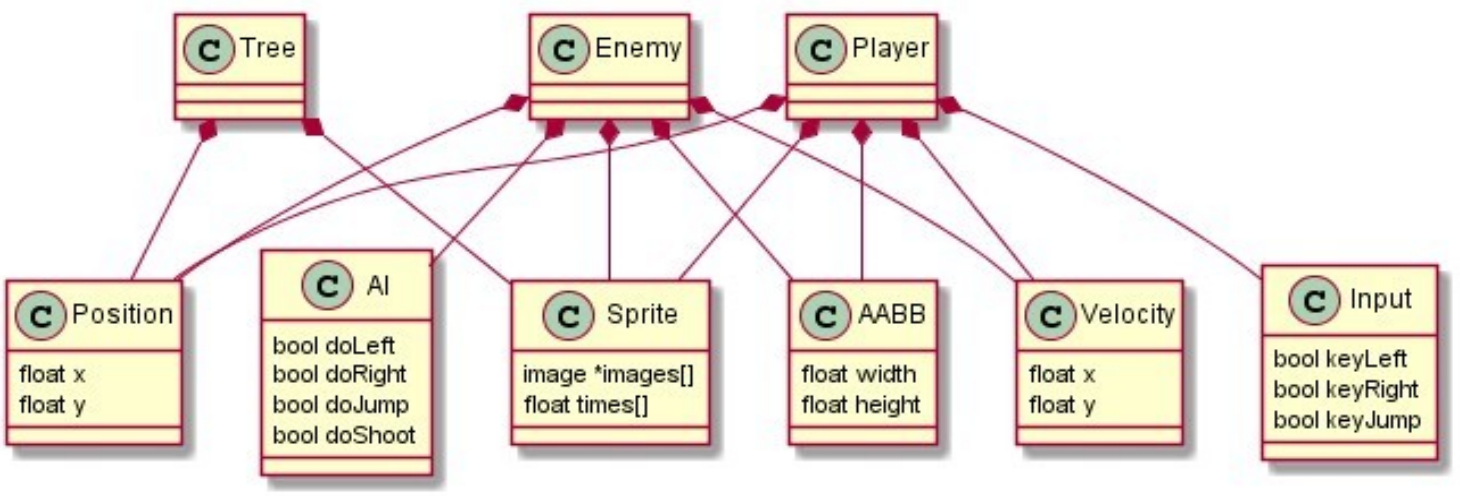

Wrote screenshots/shot0093.jpg
Wrote screenshots/shot0094.jpg
Neko was machinegunned by qubodup
You fragged Neko
1st place with 1
Neko
1
Neko
1 0 1 0
45
48

OOP Example



OOP Example

- Composition over inheritance →
  ECS (Entity-Component-System)

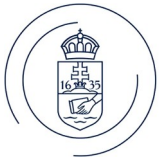


OOP Tools

- Classes, Objects…
- Composition or inheritance or delegation?
- Dynamic dispatch (late binding) or message passing
  - What about static (parametric) polymorphism? (templates)

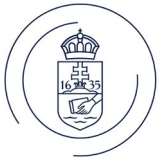
- Goals
  - Reusability
  - Maintainability
  - Support team work

OOP Tools



- Tools are not enough
  - Design Patterns
  - Control flow vs Data flow
  - Responsibility-driven Design
  - Data-driven Design

OOP Tools



- Unlimited ways of code + data grouping

  → How to do encapsulation?

  → Which one is the best?

- Maybe: Think about
  - SW processes
  - Feature introduction
  - Agile

SOLID

– 5 principles of Object Oriented Programming and Design

– Principles to remove Code Smells

S – Single Responsibility Principle

O – Open/Closed Principle

L – Liskov Substitution Principle

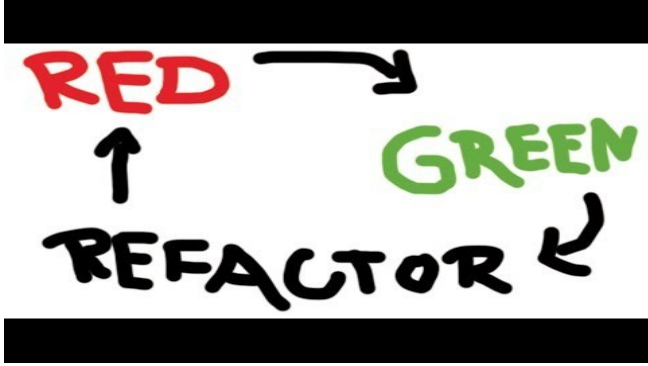
I – Interface Segregation Principle

D – Dependency Inversion Principle

Architectural Improvements

- Refactoring
- Class Normalization
- Design Patterns (and anti-patterns)

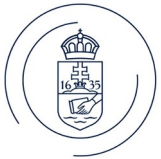
Refactoring



- No external behavior change, but
- Improve
  - Readability
  - Ease of understanding (lower complexity)
  - Extensibility
  - Maintainability

→ Improve all goals of OOP (teamwork)

Refactoring



- Program transformations
  - Rename (understanding – most important!!!)
  - Move
  - Break into components (new class or method)
    - Encapsulate
  - Generalize
  - Branching into Compound State or Polymorphic behavior
- Tools (...many IDEs)

Class Normalization

- Comes from DB normalization
- 1st object normal form (1ONF)

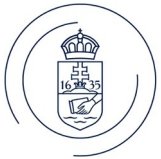
  Encapsulate behavior of multiplicity >1
- 2nd object normal form (2ONF)

  Encapsulate any shared behavior
- 3rd object normal form (3ONF)

  Encapsulate one set of cohesive behavior per class

Behavior = code, data or combination

Design Patterns



...coming soon!

- *Types*
  - *Creational*
  - *Structural*
  - *Behavioral*
  - *Concurrent*
  - *Architectural*
  - *...*

OOA

– *Object Oriented Analysis*

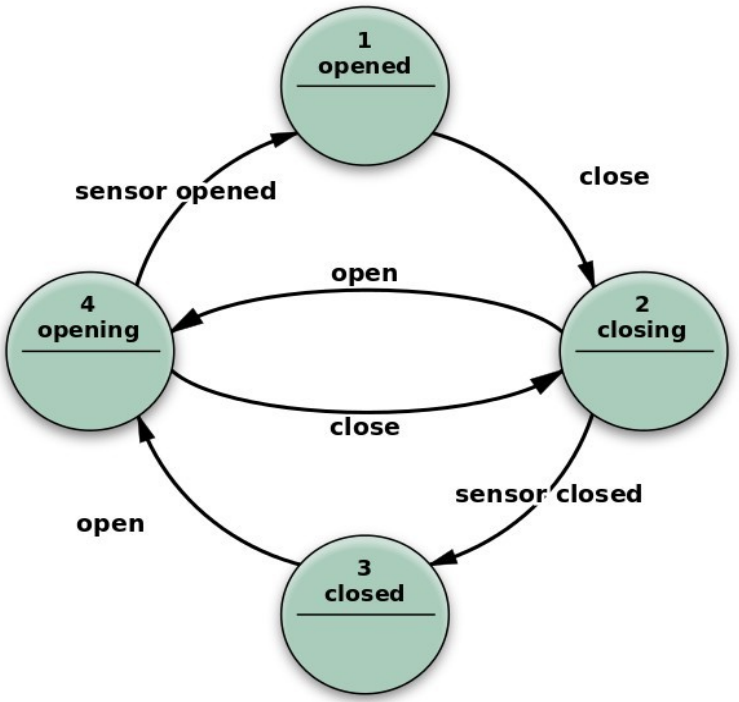
- Structured analysis and design was something like
  - Sketch up system (to some level of detail)
  - Implement
  - Improve
- Instead do deeply precise analysis →

OOA

→ OOA (Shlaer-Mellor Method)

- Translation instead of Elaboration
- Logic in Finite-State Machines
- Action Data Flow Diagram

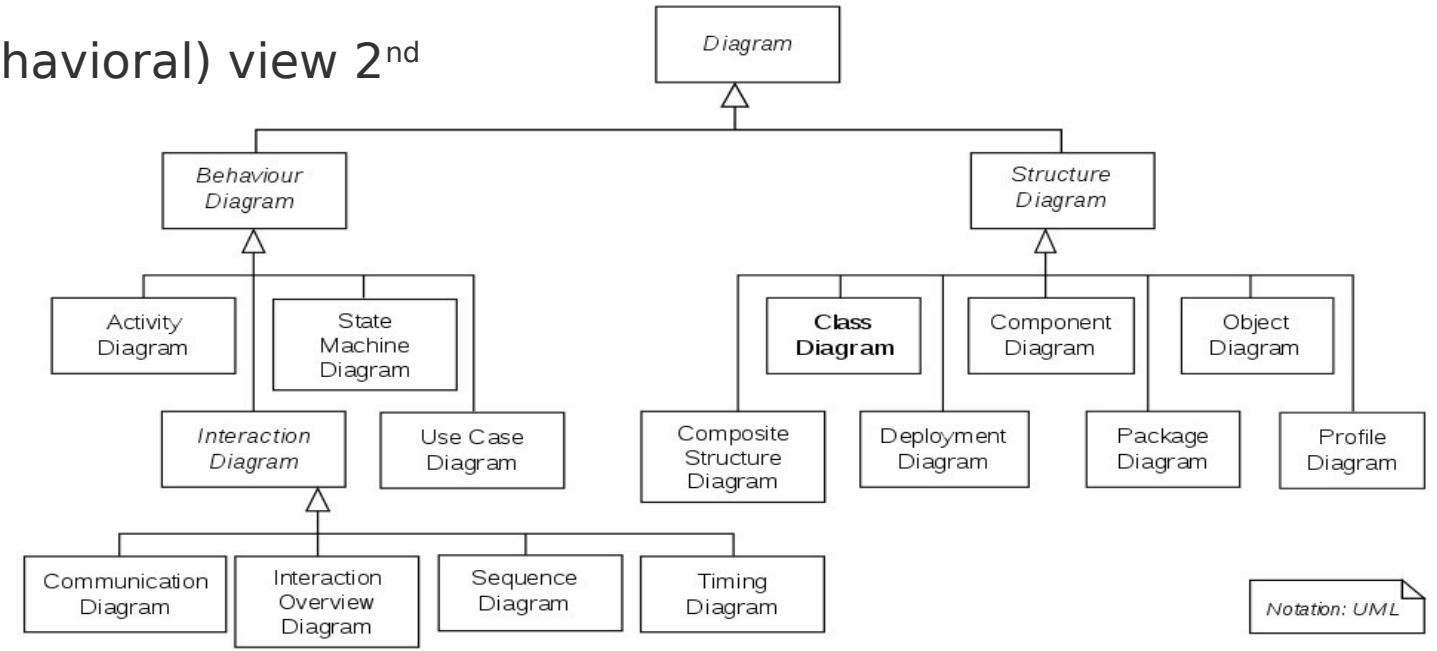
  or **Action Language**
- Virtual Machine
- Cross language, Cross platform compilable
- Simulation
- Test



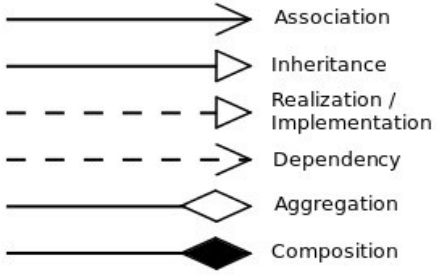
UML

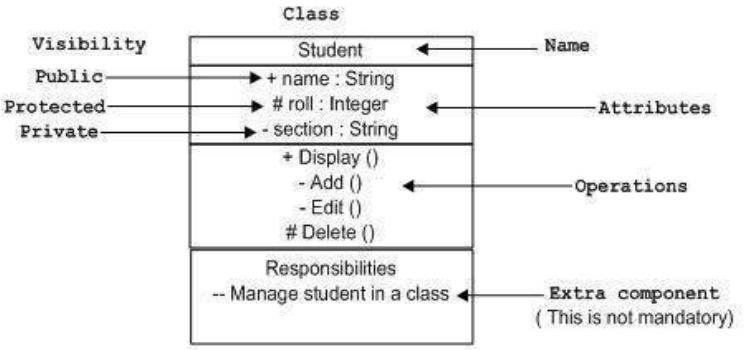
– *Unified Modeling Language*

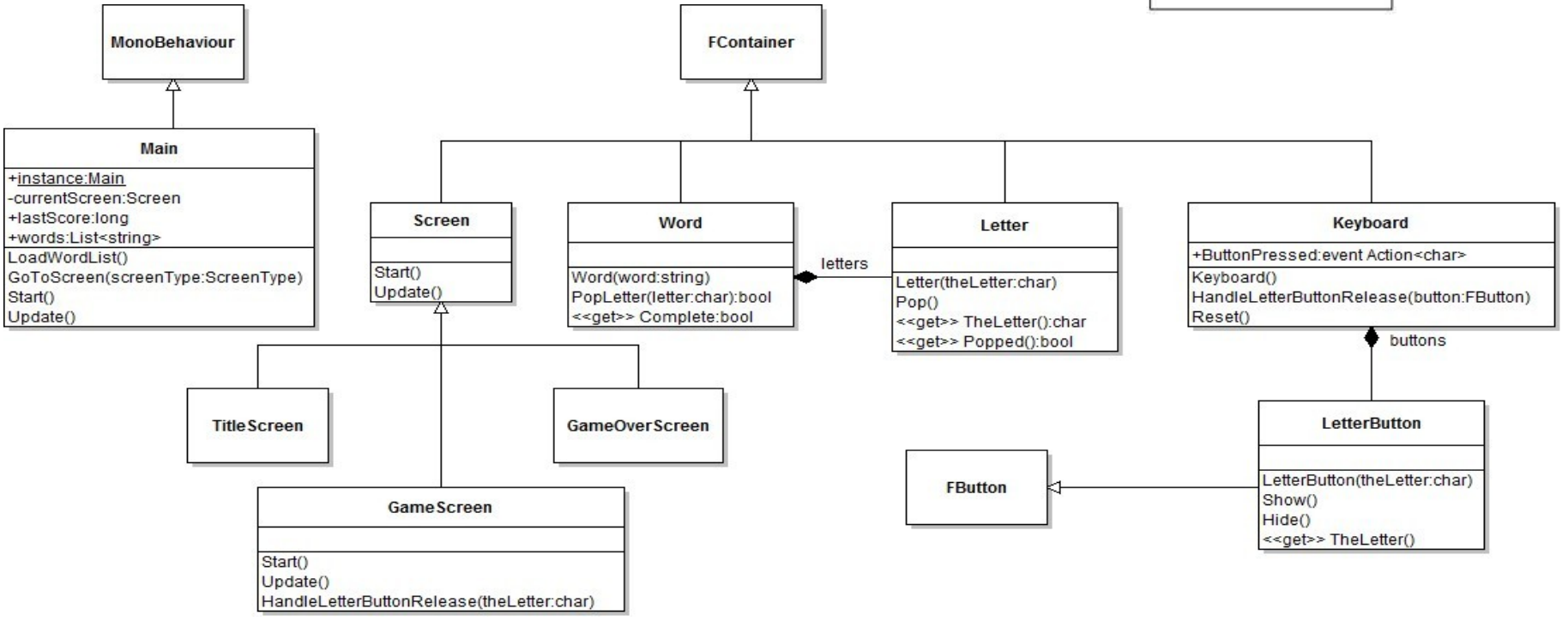
- Static (structural) view 1st
- Dynamic (behavioral) view 2nd



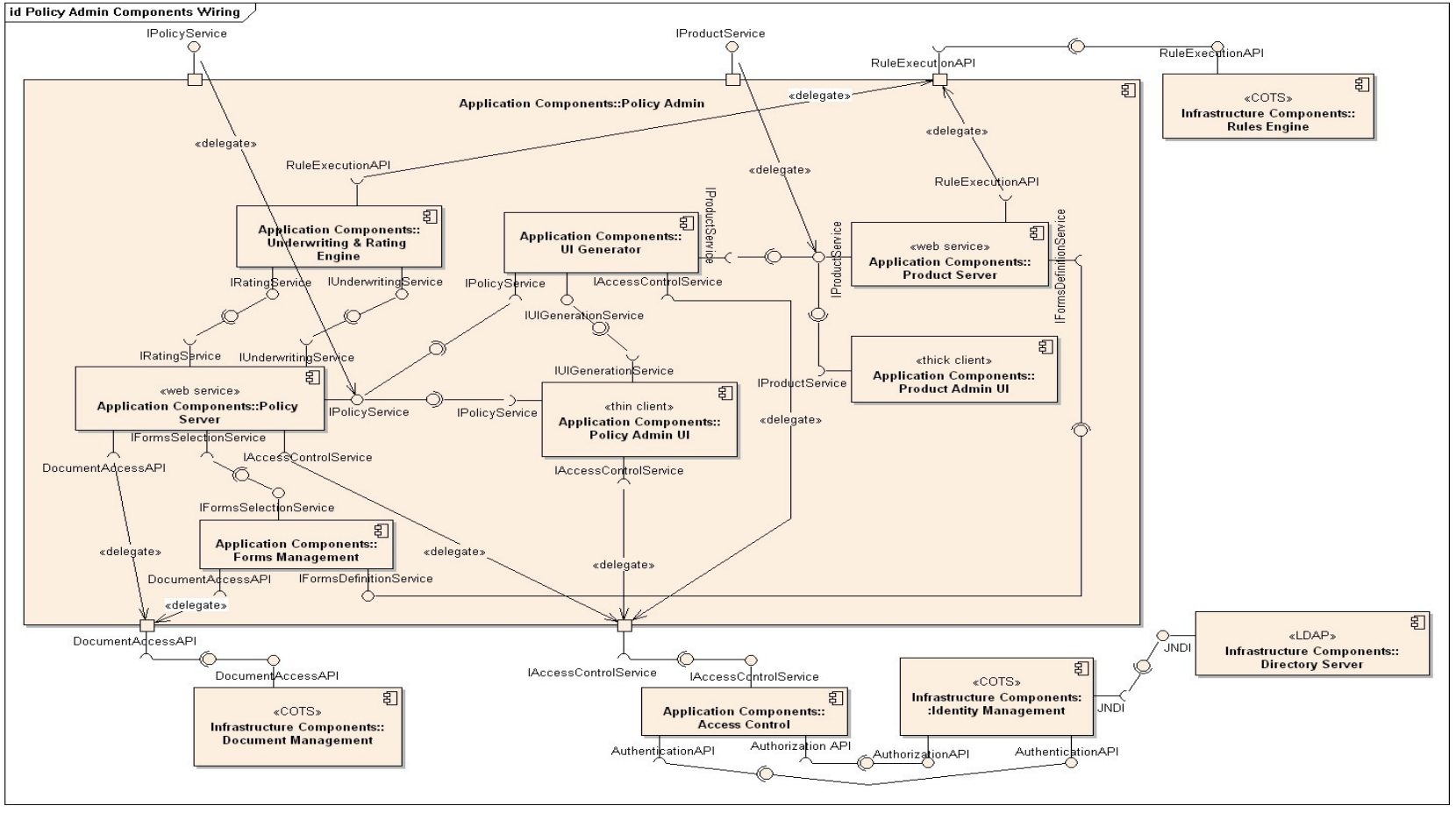
UML Class Diagram



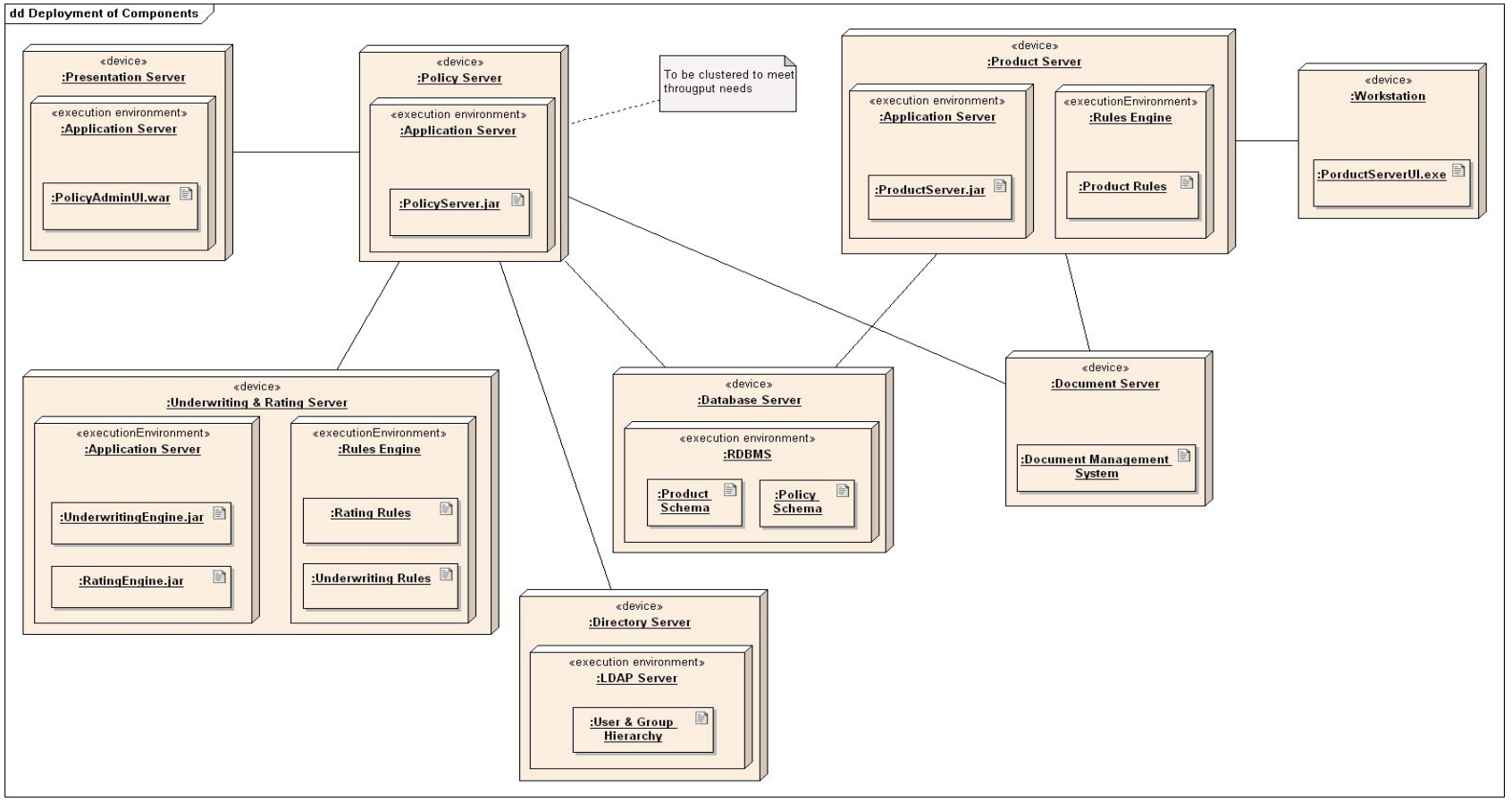




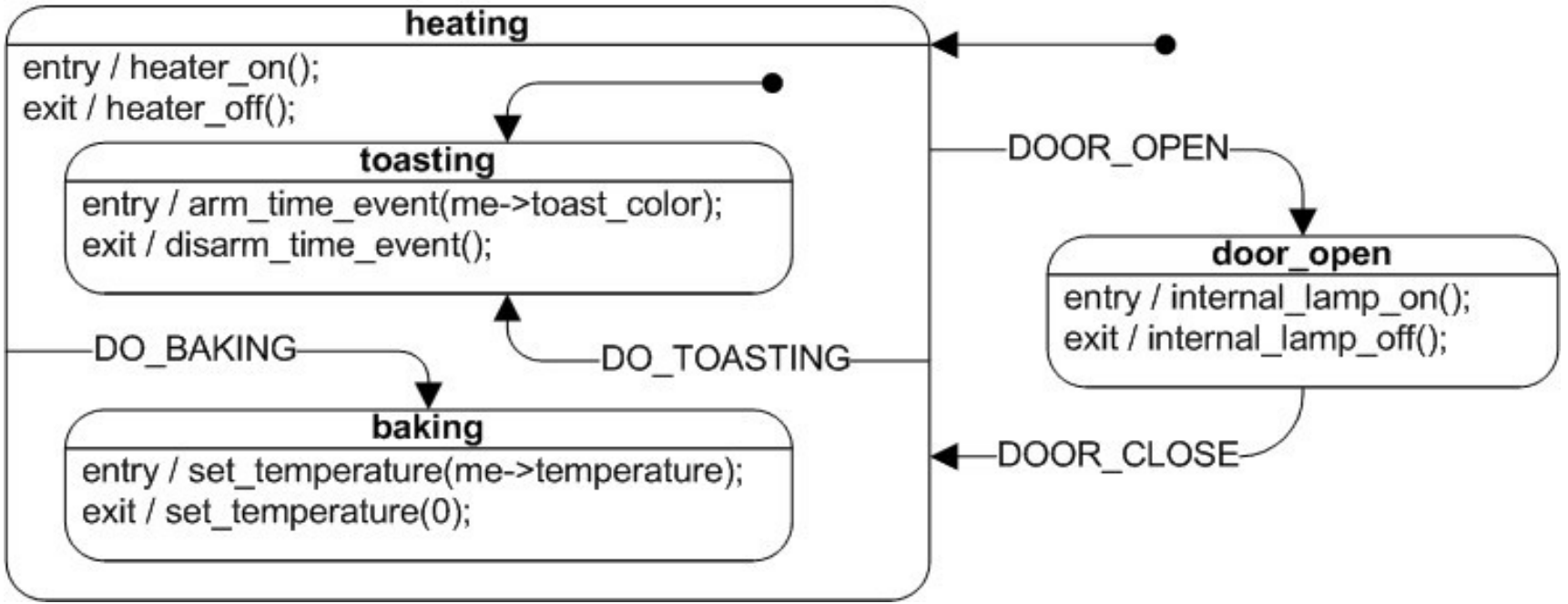
UML Component Diagram



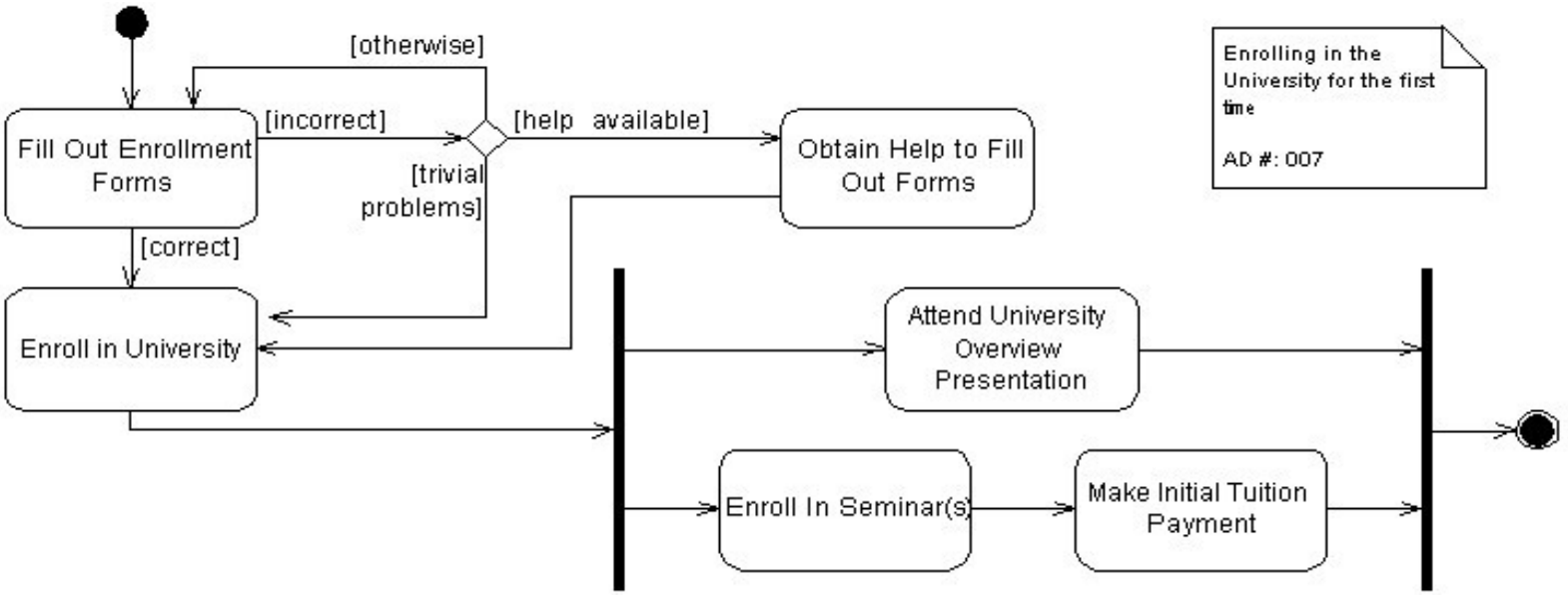
UML Deployment Diagram



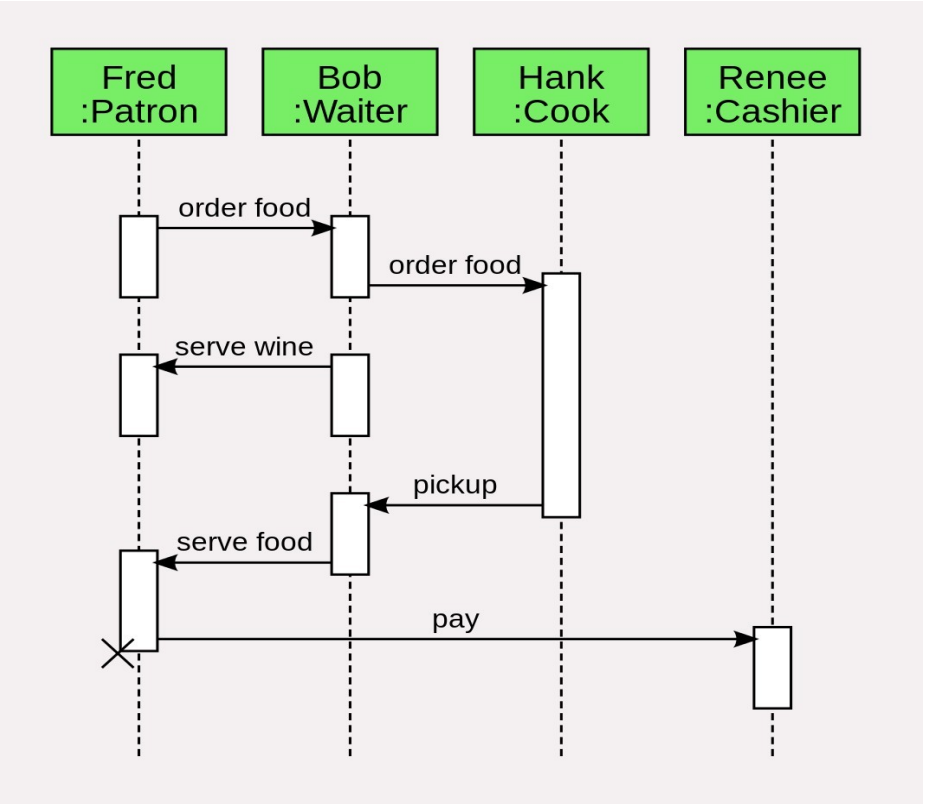
UML State Machine Diagram



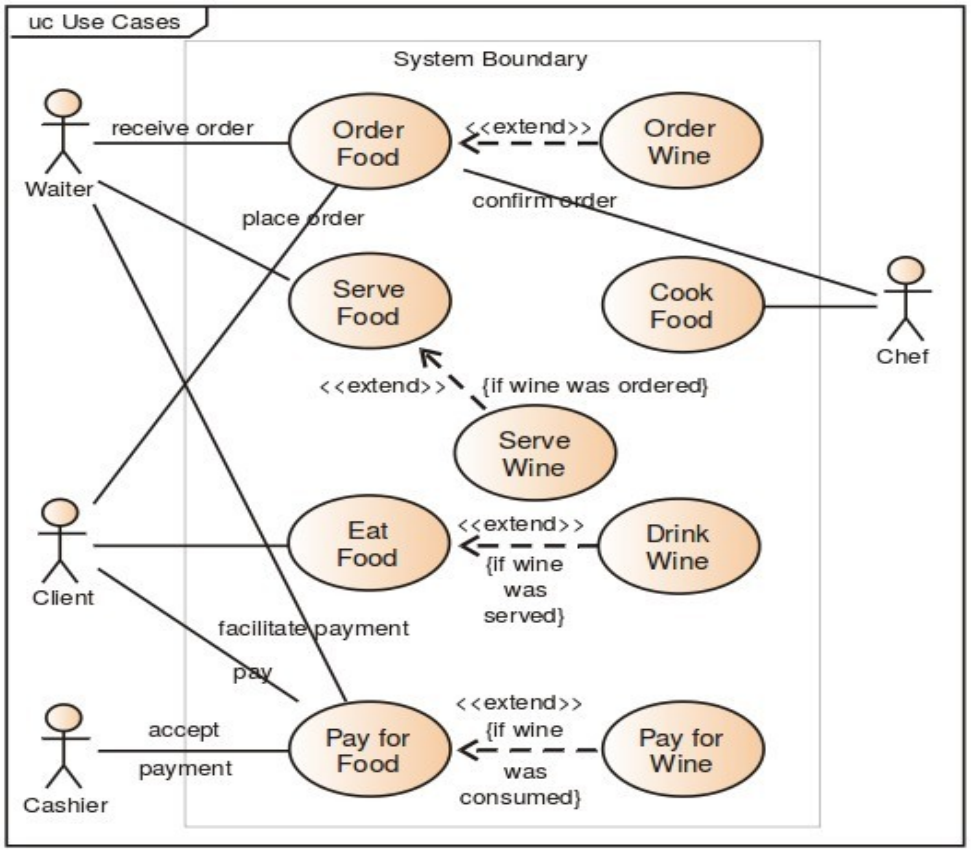
UML Activity Diagram



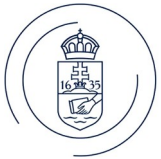
UML Sequence Diagram



UML Use Case Diagram

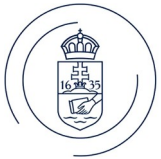


UML Criticism



- Good to visualize and present, but
- Nobody wants to program this way (creating diagrams)
- Complex diagrams cannot be overseen
- Simple diagrams are useless
- Only program stub is generated
- No round-trip editing

xtUML



– *eXecutable Unified Modeling Language*

- UML subsets (to make xtUML fully supported)
- Action Language
- Virtual Machine
- Testing, debugging (including state visualization), measurements are possible on original model without compilation
- Model Compilation
  - Into any **language**
  - On any **platform**
  - Possible optimization to target language / platform

MDD

– *Model-driven Development*

- Model-driven architecture design is useful for further development
- Model-driven Testing can be used independent of platform
- Model-driven Testing can give proof