Software Engineering in Practice
Software design

Diomidis Spinellis
Department of Management Science and Technology
Athens University of Economics and Business

dds@aueb.gr
http://www.dmst.aueb.gr/dds
@CoolSWEng

2024-03-11

Team presentations

- Select a (significant-popular) open source project and inspect its design elements:
  - look for good design practices,
  - identify factors that are significant for the selected project
  - select a software design strategy and the appropriate notations in order to provide a static representation of the software's design. You can use abstractions, a software design principle, in order to represent entities that you consider to be the most important.

Software design

- Design: one word, two meanings
  - The process
  - The outcome of the process
- Includes
  - An architectural overview of the system (high-level design)
  - Detailed designs for each component

Process

- Determine who are the stakeholders
- Determine their concerns
- Select viewpoints that can frame those concerns (e.g. 4+1, TOGAF, GERA)
  - Each viewpoint can have its own models and use its own language
- Populate the views according to the viewpoints ensuring consistency across viewpoints

Deliverables

- Models

Figure 1: College of Architecture and Planning

- High-level descriptions
- Documentation
  - Decisions
  - The reasoning during the decision making

Software design principles

- Abstraction of:
  - procedures
  - data
  - control structures
- Low coupling
- High cohesion
- Decomposition and modularization
- Encapsulation and information hiding
- Separation of interface and implementation
- Sufficiency, completeness, and primitiveness (simplicity)
- Separation of concerns

Increased cohesion between components

A good design should group highly cohesive units. We observe the following high-cohesive types:

- Coincidental cohesion
- Logical cohesion
- Temporal cohesion
- Procedural cohesion
- Communicational cohesion
- Sequential cohesion
- Functional cohesion

Coupling, from bad to worse

A good design should avoid coupling between components, as much as possible. We observe the following high-coupling types:

- Data coupling: Data exchange only.
- Stamp coupling: Unneeded structured data passed.
- Control coupling: Passing control flags.
- Common coupling: Sharing global data.
- External coupling: Shared knowledge about external systems.
- Content coupling: Access to implementation's internals.

Key issues

- Concurrency
- Control and handling of events
- Data persistence
- Distribution of components
- Error and exception handling
- Fault tolerance
- Interaction and presentation
- Security

4+1 design views

There are views from different perspectives:

- Logical (based on functional requirements)
- Process
- Physical (deployment)
- Development
- Scenarios

TOGAF Architectural Domains

According to The Open Group Architecture Framework enterprise architecture is divided into the following domains.

- Business (processes, guidelines, structures)
- Applications (to be developed)
- Data (logical and physical models)
- Technical (computing, network, storage resources)

Architectural styles

- General
  - Layers
  - Pipes and filters
  - Blackboard
- Distributed
  - Client-server
  - Three-tier
  - Broker
- Specialized
  - Model - View - Controller (MVC)
  - Reflection
  - Interpreter
  - Domain Specific Language (DSL)

- Data-driven

Example: layers

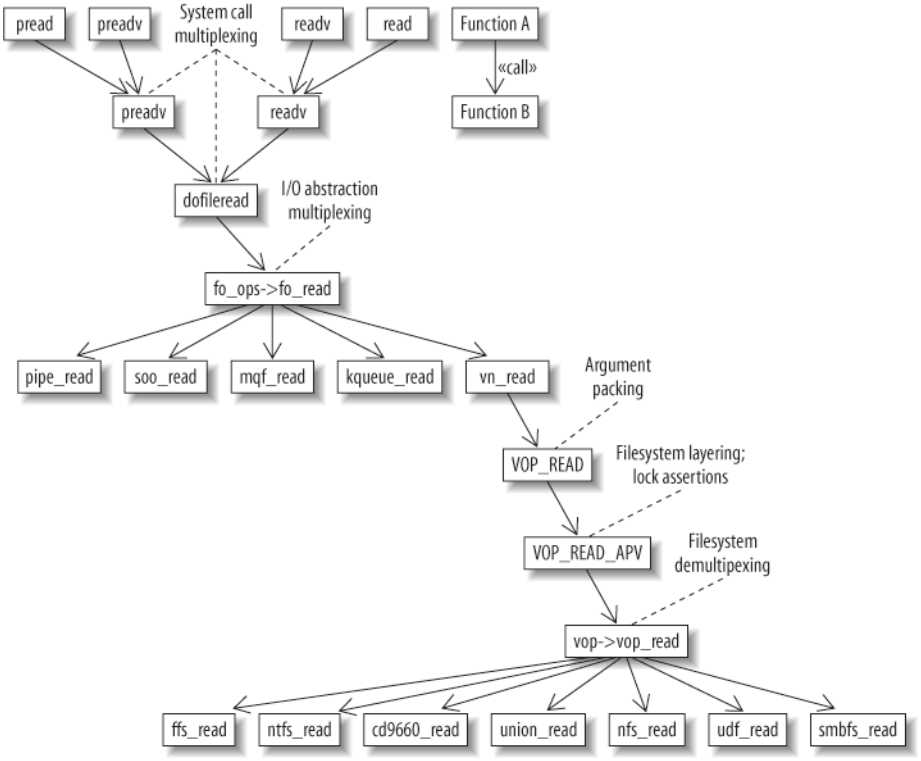

Figure 2: Layers of indirection in the FreeBSD implementation of the read system call

Example: pipes and filters

```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq |
comm -23 - /usr/dict/words
```

Example: DSL

```
{{Chess diagram small|=
| tright
|
```

```
|=
 8 |rd|  |  |qd|  |rd|kd|  |=
 7 |pd|pd|  |  |pd|pd|bd|pd|=
 6 |  |nd|pd|  |  |nd|pd|  |=
 5 |  |  |ql|  |  |  |bl|  |=
 4 |  |  |  |pl|pl|  |bd|  |=
 3 |  |  |nl|  |  |nl|  |  |=
 2 |pl|pl|  |  |  |pl|pl|pl|=
 1 |  |  |  |rl|kl|bl|  |rl|=
    a  b  c  d  e  f  g  h
| The position after 11. Bg5.
}}
```

Example: DSL output

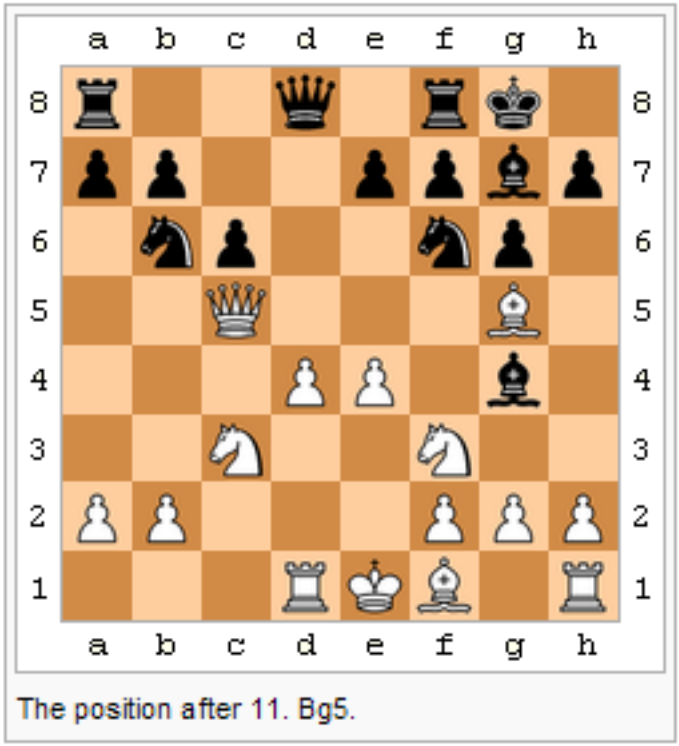

Figure 3: Chess board

Design patterns

- Creational: Builder, Factory, Prototype, Singleton

- Structural: Adapter, Bridge, Composite, Decorator, Facade, Flighweight Proxy
- Behavioral: Command, Intepreter, Iterator, Mediator

Example: Factory

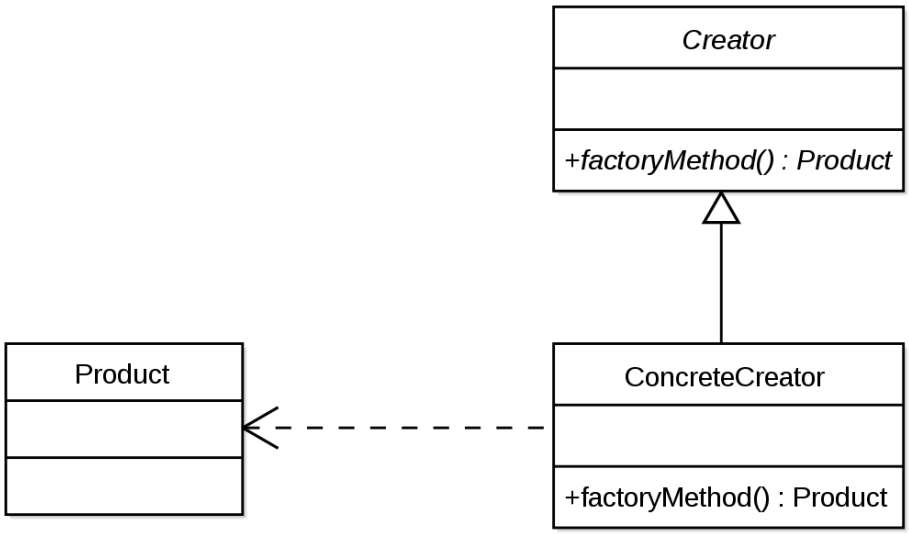

Figure 4: Factory

(Source: Wikipedia)

Example: Observer

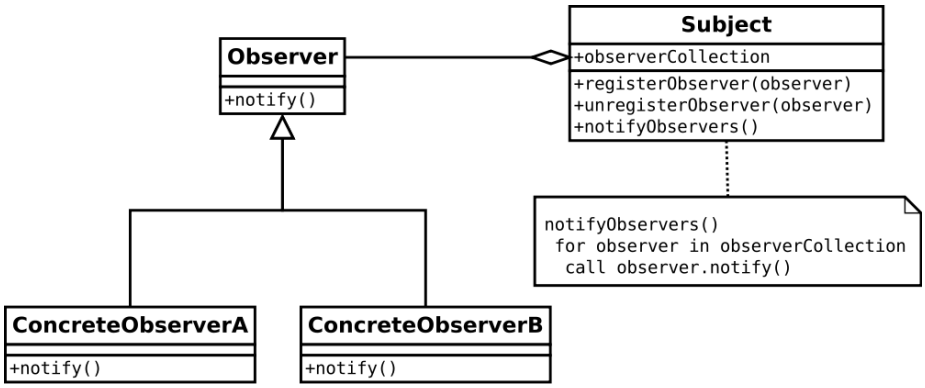

Figure 5: Observer

(Source: Wikipedia)

Example: Decorator

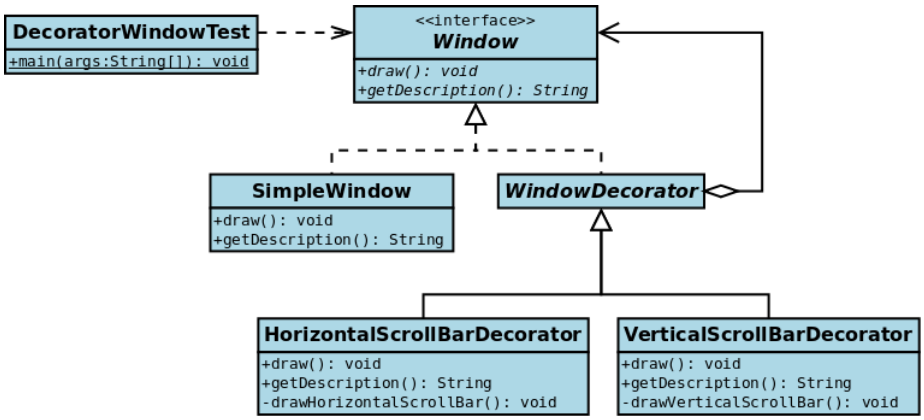

Figure 6: Decorator

(Source: Wikipedia)

Satisfying the developers' needs

- Families of programs
- Frameworks

User interface design principles

- Learnability
- User familiarity
- Consistency
- Minimal surprise (POLA)
- Recoverability
- User guidance
- User diversity

User interaction modalities

- Question-answer
- Direct manipulation
- Menu selection
- Form fill-in
- Command language
- Natural language

User interface design guidelines

- Quick response time
- Feedback and indication of progress
- Attention on the colour usage
- Internationalization and localization
- Use of metaphors

Software design notations

- UML class diagram
- UML object diagram
- UML component diagram
- UML deployment diagram
- Entity relationship diagram (ERD)
- «Class, responsibility, collaborator» (CRC) cards

UML diagrams

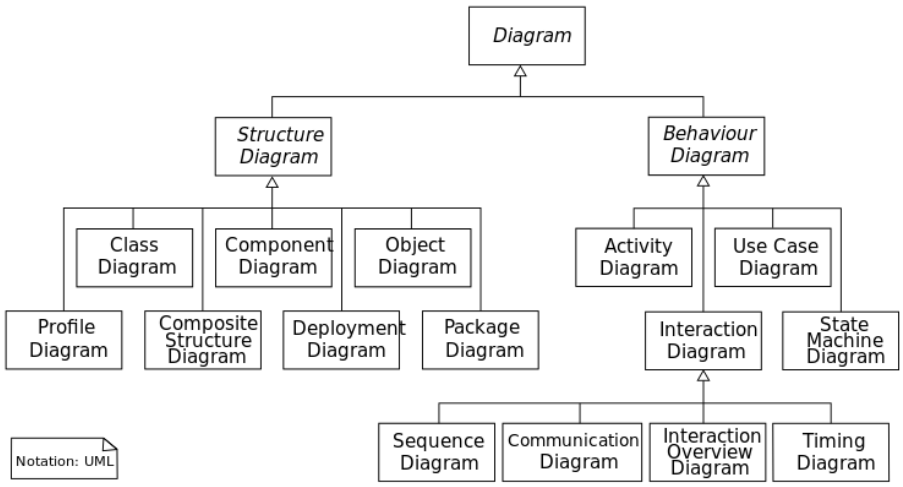

Figure 7: UML Diagrams

(Source: Wikipedia)

UML relationships

UML defines the following four basic relationships:

- Dependency
- Generalization

- Association
- Realization

Dependency

The dependency declares that a change of one entity will affect an other entity. It is represented with a dashed line between two entities, while the arrow shows towards the entity that it depends upon:

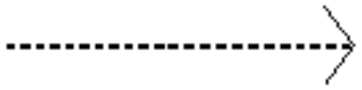

Figure 8: UML Dependency

Generalization

The generalization defines a relationship between a generalized entity (base class or parent) and a specialized entity (subclass or child class). It is represented with a continuous line with a closed arrow that points towards the base class:

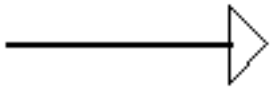

Figure 9: UML Dependency

Association

The association refers to related entities. When two classes are related someone can access objects of one class through the objects of the other class. The relationship is represented with a straight line between the related entities:

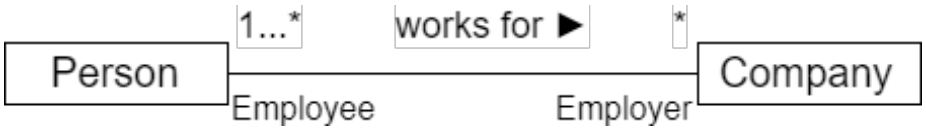

Figure 10: UML Association

Aggregation and composition

Relationships where an object part of a whole are defined as aggregations and are represented with a diamond at the side of the whole object:

Figure 11: UML Aggregation

When the objects that participate in the whole share the same lifecycle with the whole object, then this relationship is called composition and is represented with a filled diamond at the side of the whole object:

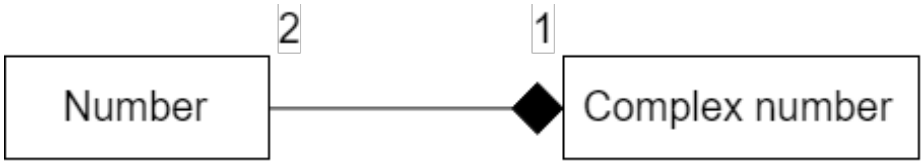

Figure 12: UML Composition

Relationship information

- Single-direction relationships are represented with an open arrow.
- The name of the relationship should be placed over the line while the direction of the relationship is defined by an arrow located next to the name.
- The roles of the participating entities should be stated at the two edges of the line.
- The number of the objects that participate in a relationship between two entities (multiplicity) should be defined as a number (i.e. 3), or a range (i.e. 1…* for one to many) over the appropriate edge of the relationship.
- An **X** on the edge of a line states that navigation towards that direction is not provided.

Realization

The realization defines a relationship where the entity at the tail of the arrow implements all the responsibilities defined by the interface at the head of the arrow:

UML: Class diagram

UML: Deployment diagram

(Source: Wikipedia)

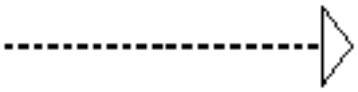

Figure 13: UML Realization

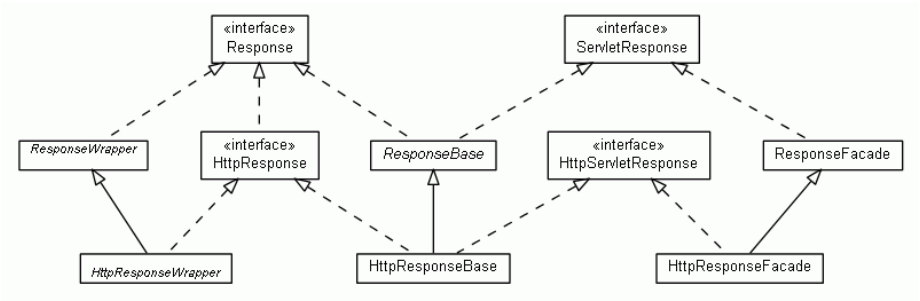

Figure 14: UML class diagram

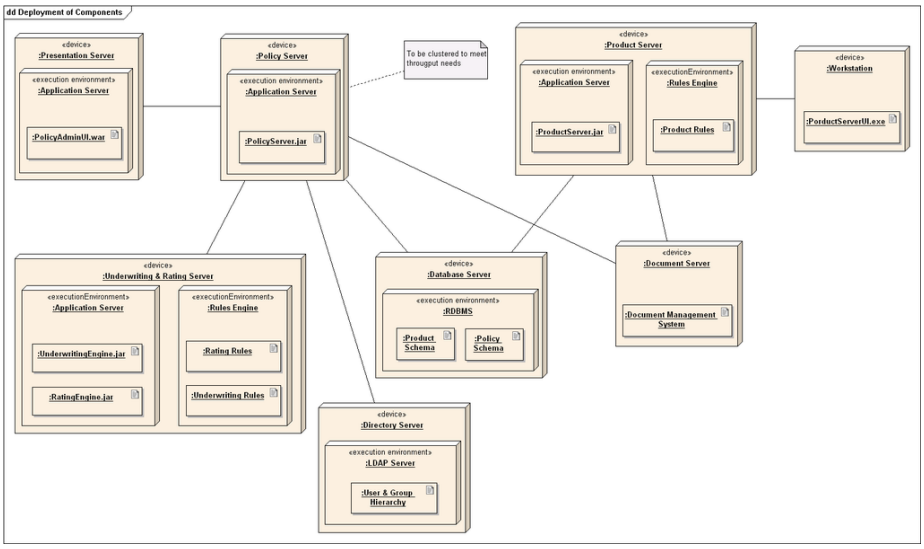

Figure 15: UML

Entity relationship diagram

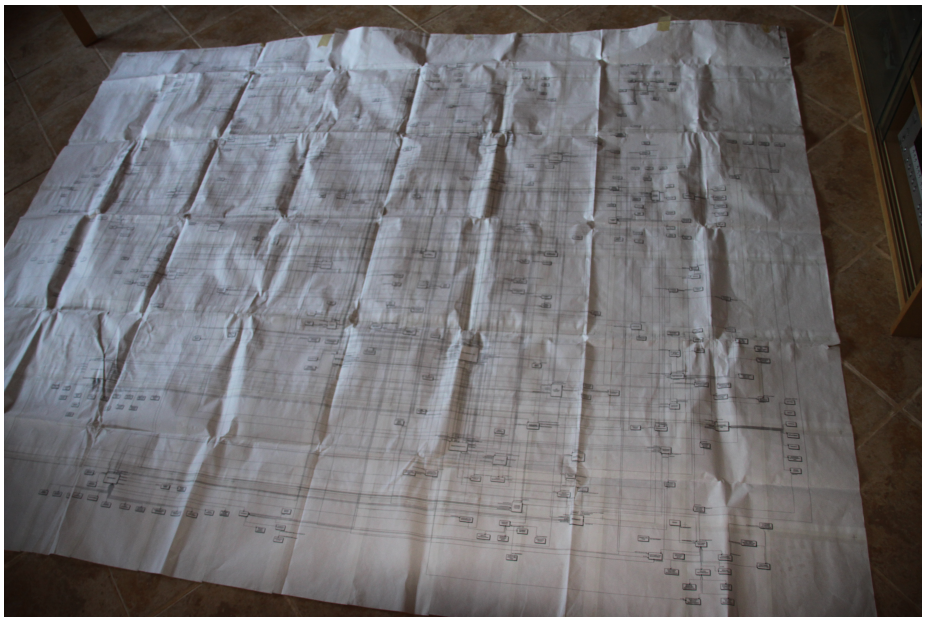

Figure 16: ERD

(Source: Christos Papadoulis)

UML: Component diagram

(Source: Wikipedia)

UML: Package diagram

(Source: Wikipedia)

Behavioural diagrams

- UML activity diagram
- UML statechart
- Data flow diagram
- Decision table
- Flow chart
- UML sequence diagram
- UML state diagram
- UML collaboration diagram

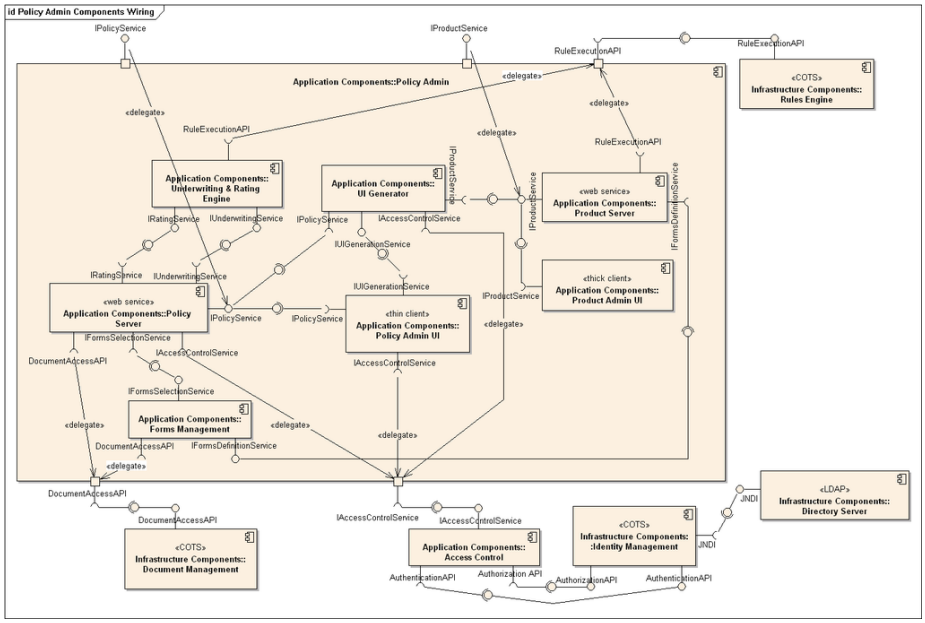

Figure 17: UML

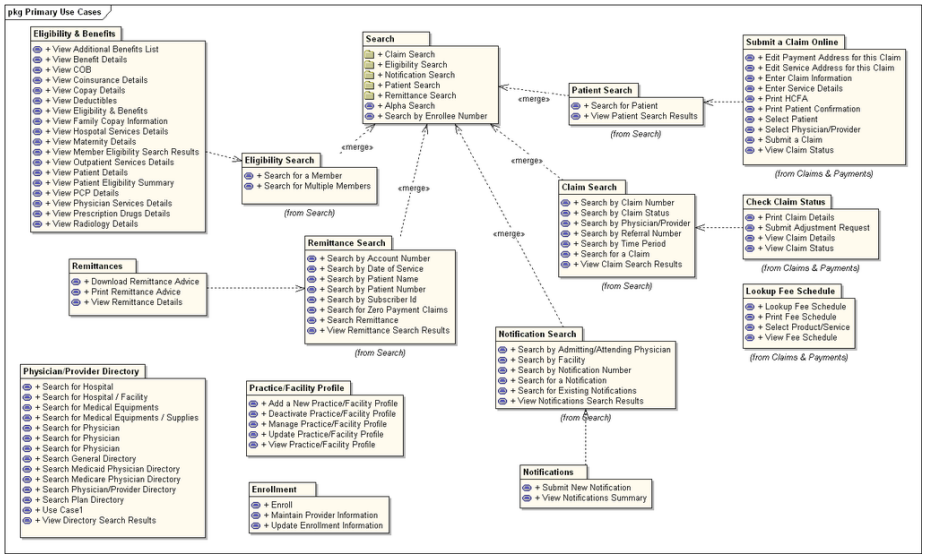

Figure 18: UML

UML: State diagram

UML: Sequence diagram

Tools and other material

- How to draw beautiful software architecture diagrams
- Community list of comparisons between Text to Diagram tools

Preparation for the next lecture (1)

- Study Chapter 3: “Software Construction” from SWEBOK v 3.0
- Assignment (Software Construction)
  - Evaluate the characteristics of a popular open source project with respect to the construction:
  - Look for construction guidelines and standards. How do they enable the validation of the software’s correctness?
  - Identify reusable and reused software components.
  - Look for code quality analysis techniques.
  - Try to improve the system’s construction quality. Contribute to a change that you propose.

Preparation for the next lecture (2)

- Video (Designing an Application Programming Interface (API)) https://www.youtube.com/watch?v=aAb7hSCtvGw

Advice for the assignment:

- Identify and focus on components that provide the main functionality of the system.
- Identify software construction standards and good practices.
- Comprehend the term drive-by commit.
- Make use of the Github service. Check the “How to”.

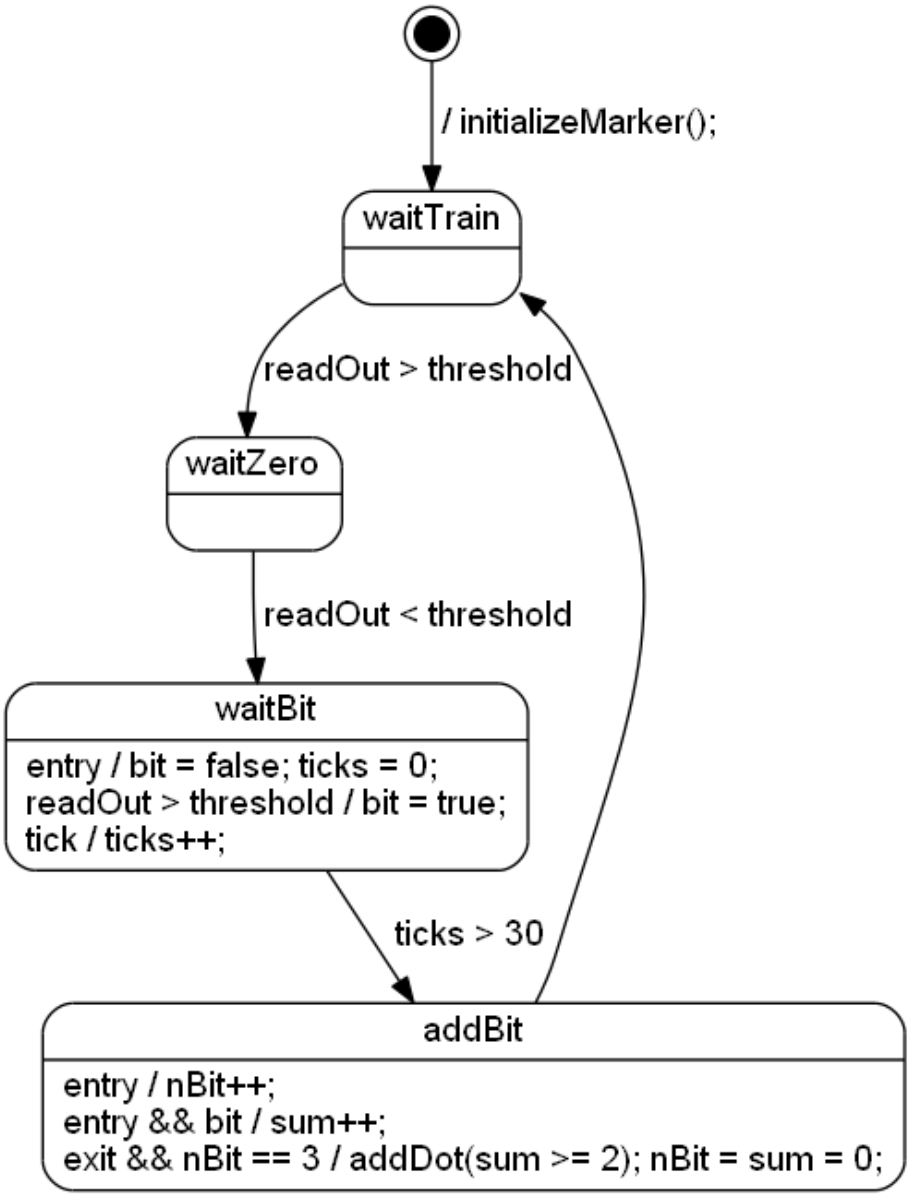

Figure 19: UML

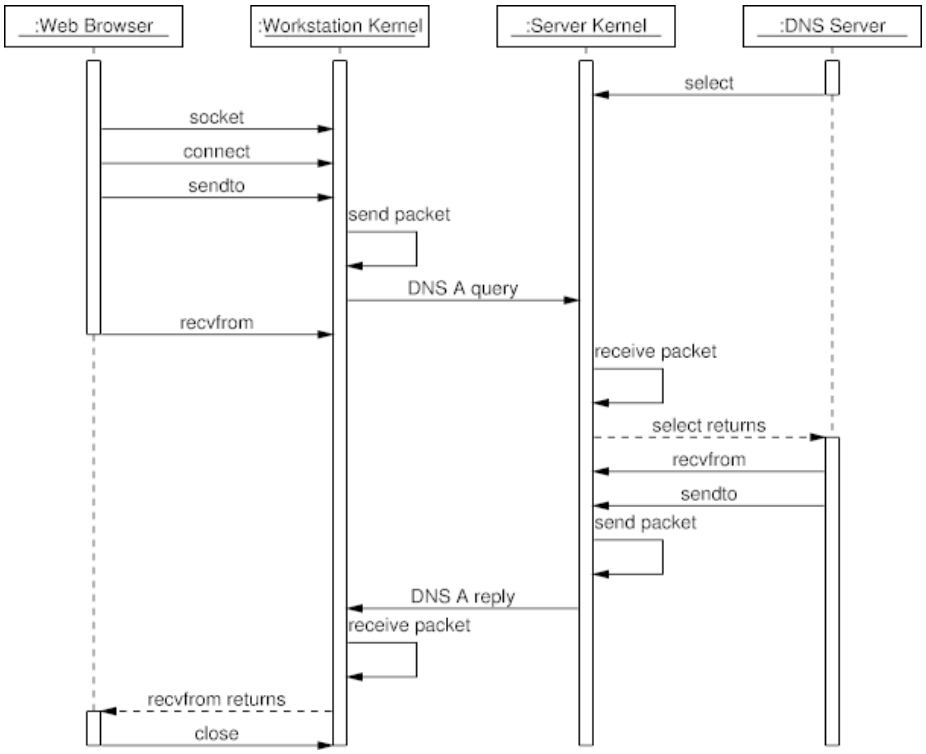

Figure 20: UML