Software Engineering 2
A practical course in software engineering

Ekkart Kindler

DTU Informatics
Department of Informatics and Mathematical Modeling

II. Modelling Software

DTU Informatics
Department of Informatics and Mathematical Modeling

Modelling Software

- Model based software engineering
(taking models a bit more seriously than we did traditionally)

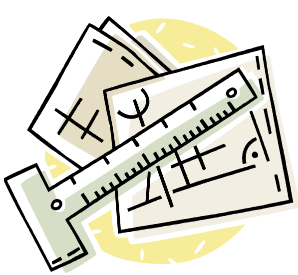
- Reverse engineering

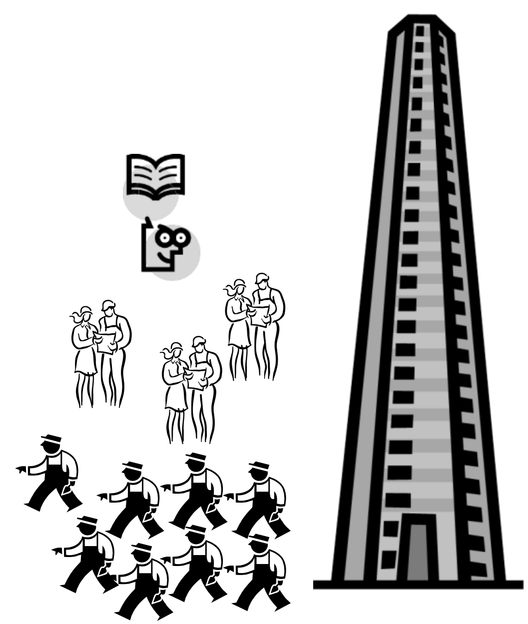
In the last lecture, the focus was on the future, and we jumped to conclusions (for motivation purposes). Now, we fill in some basics and "traditional" software engineering.

1. Motivation

- What are "software models"?
- What are they good for?
- Why do WE need them?

- What is software?
- What is a model?





DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

Modell [*lat.-vulgärlat.-it.*] *das; -s, -e*:

…

7. die vereinfachte Darstellung der Funktion eines Gegenstands od. des Ablaufs eines Sachverhalts, die eine Untersuchung od. Erforschung erleichtert od. erst möglich macht.

…

[nach Duden: Das Fremdwörterbuch, 1990].

Model

Modell [*lat.-vulgärlat.-it.*] *das; -s, -e*:

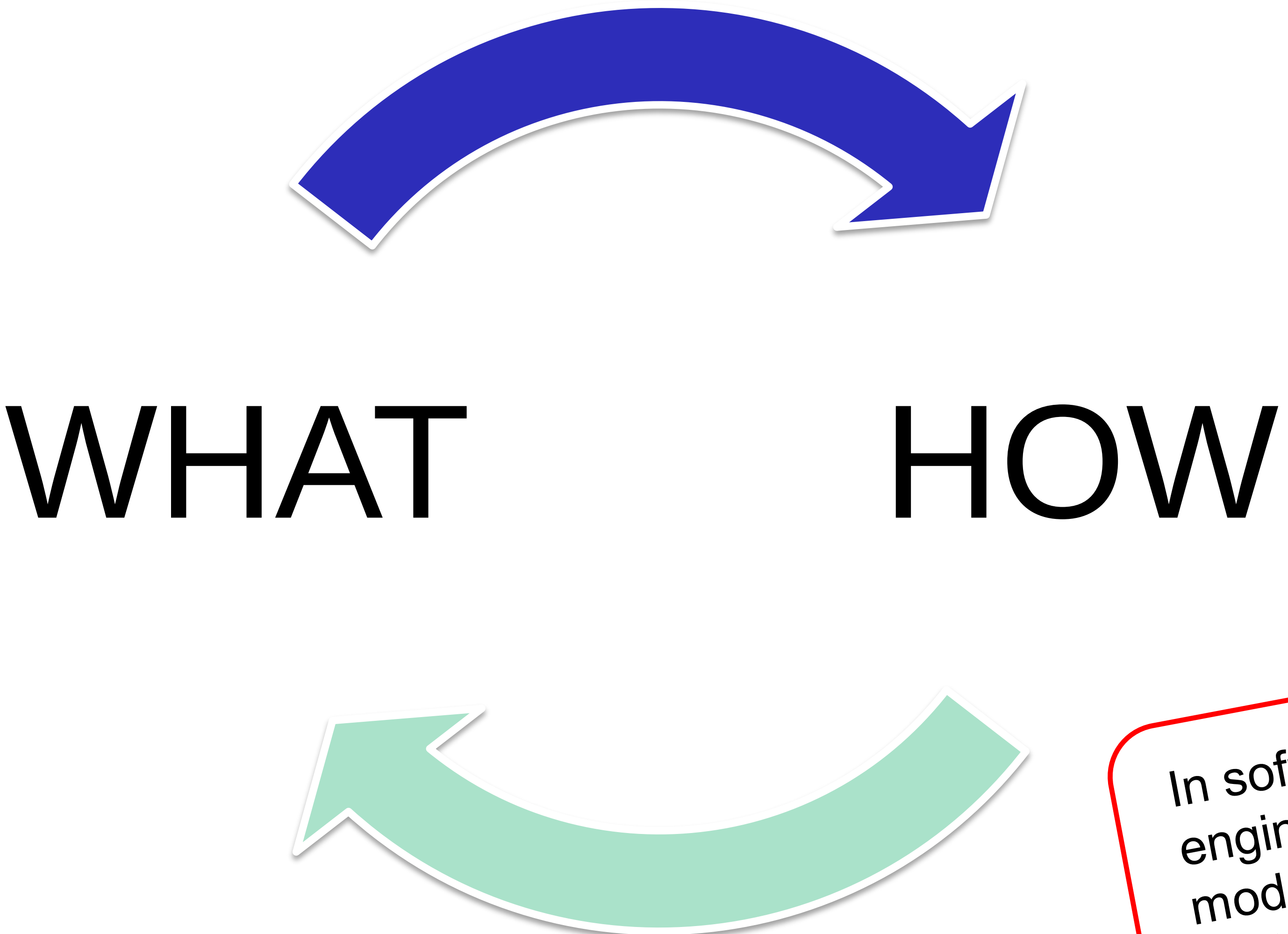
…

7. the simplified description of the function, purpose, or process of something; it enables us investigating and analysing this thing.

…

[nach Duden: Das Fremdwörterbuch, 1990].

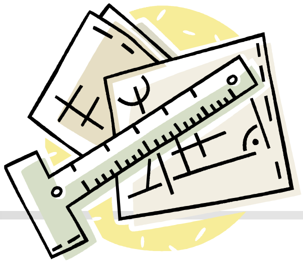
Reminder (cf. L01)



In software engineering, we do models for both, the WHAT and the HOW.

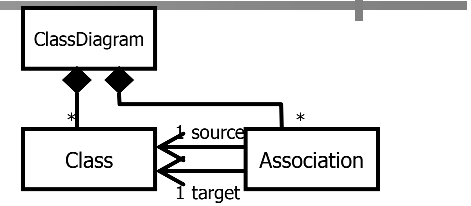
Purpose of Models

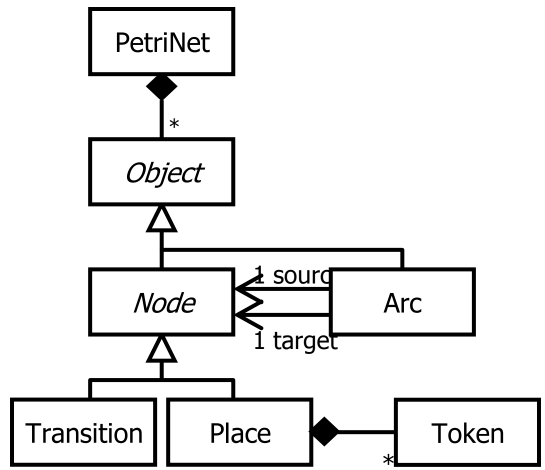
- better understanding the „thing“ under investigation (or development)
- communication
  - on the appropriate level of abstraction
  - with different kinds of people
  - from different angles
- abstraction / composition
- analysis and verification
  - consistency, completeness, correctness, performance, risks, effort, …
- code generation (cf. L01)

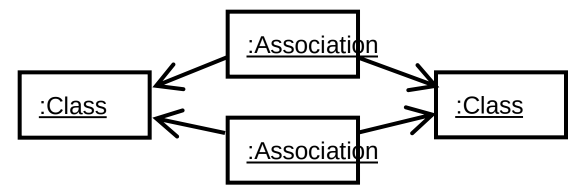


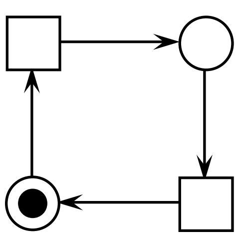
Reminder (cf. L01)

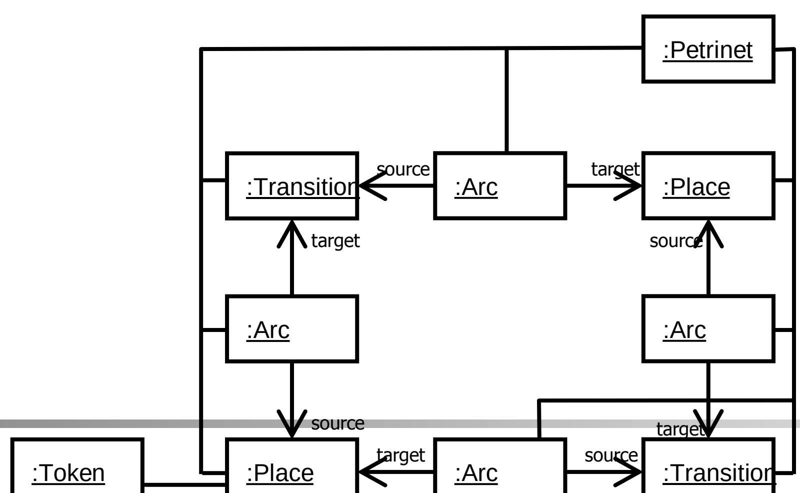
DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler











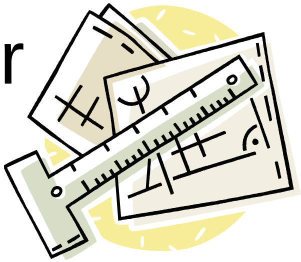
Roles of models in SE

- „traditional“: More or less automatic:
  - Forward engineering
  - Reverse engineering
  - Reengineering
- Model Driven Architecture (MDA)
  - Generating (at least part of) the software from models

→ Models ARE the software
(or a part of it)

2.1 „Traditional“

Initially: Informal sketches of software for discussion, for better understanding or for communicating an idea



Later: Standardized (graphical) notations (UML)

From these diagrams the program code was produced (mostly) manually!

Forward engineering

„Traditional“

- Since software is often not well-documented, it became necessary to retrieve or to extract the essential idea of the software from its code

Reverse engineering

- These models are used to better understand the existing software, and to change the software based on this understanding

Reengineering = Reverse + Forward engineering

Automation

- Some reverse and forward engineering tasks could be automated (mainly structural parts)
- Changes made in the models obtained by reverse engineering can (sometimes) be automatically transferred back into the original code

Roundtrip engineering

2.2. Reverse engineering

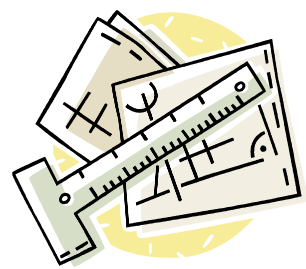
Starting point:

- Software cannot be used in isolation
- It interacts with other software
- In most cases, developers must extend existing software or integrate their software to existing one

- Existing software is often not documented (or at least not documented well)

Motivation

- Before you can (use,) change or extend software, we need to understand it



Definition

- **Reverse engineering** is the process that, for an existing software system, tracks down and retrieves ("mines") its underlying ideas and concepts and documents them in form of models

- The development process is run in the reverse direction (reverse engineering)

Result

- In the ideal case, the result of **reverse engineering** would be a specification of the software system

- Very important: abstraction and focus on the essentials

Is it possible to “mine” the ideas and to capture them in models at all?

Tools

- Tools can support reverse engineering

- But, they cannot fully relieve an engineer of the burden of abstraction and focus!

  This is the task of an engineer!

- Moreover, many of today's tools come up with wrong or incomplete results, which need to be corrected or amended by hand.

Example: Code

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

```
public interface Moveable {
   public void move();
}
public abstract class Element {
  ...
}
public class Track extends Element {
   private Track next;
   private Track prev;
   public Track getNext() {
      return this.next;
   }
   public void setNext(Track value) {
      if (this.next != value) {
         if (this.next != null) {
            Track oldValue = this.next;
            this.next = null;
            oldValue.setPrev (null);
         }
         this.next = value;
         if (value != null) {
            value.setPrev (this);
         }
      }
   }
   public Track getPrev() {
      return this.prev;
   }
   public void setPrev(Track value) {
      if (this.prev != value) {
         if (this.prev != null) {
            Track oldValue = this.prev;
            this.prev = null;
            oldValue.setNext (null);
         }
         this.prev = value;
         if (value != null) {
            value.setNext (this);
         }
      }
   }
}
```

```
public class Shuttle extends Element implements Moveable {
   private boolean driving;
   private Track at;
   private Simulation simulation;
   public Track getAt() {
      return this.at;
   }
   public void setAt(Track value) {
      if ((this.at == null && value != null) ||
           (this.at != null && !this.at.equals(value))) {
         this.at = value;
      }
   }
   public boolean isDriving() {
      return this.driving;
   }
   public void setDriving(boolean value) {
      this.driving = value;
   }
   public Simulation getSimulation() {
      return this.simulation;
   }
   public void setSimulation(Simulation value) {
      if (this.simulation != value) {
         if (this.simulation != null) {
            Simulation oldValue = this.simulation;
            this.simulation = null;
            oldValue.removeFromShuttles (this);
         }
         this.simulation = value;
         if (value != null) {
            value.addToShuttles (this);
         }
      }
   }
   public void move() {
     ...
   }
}
```

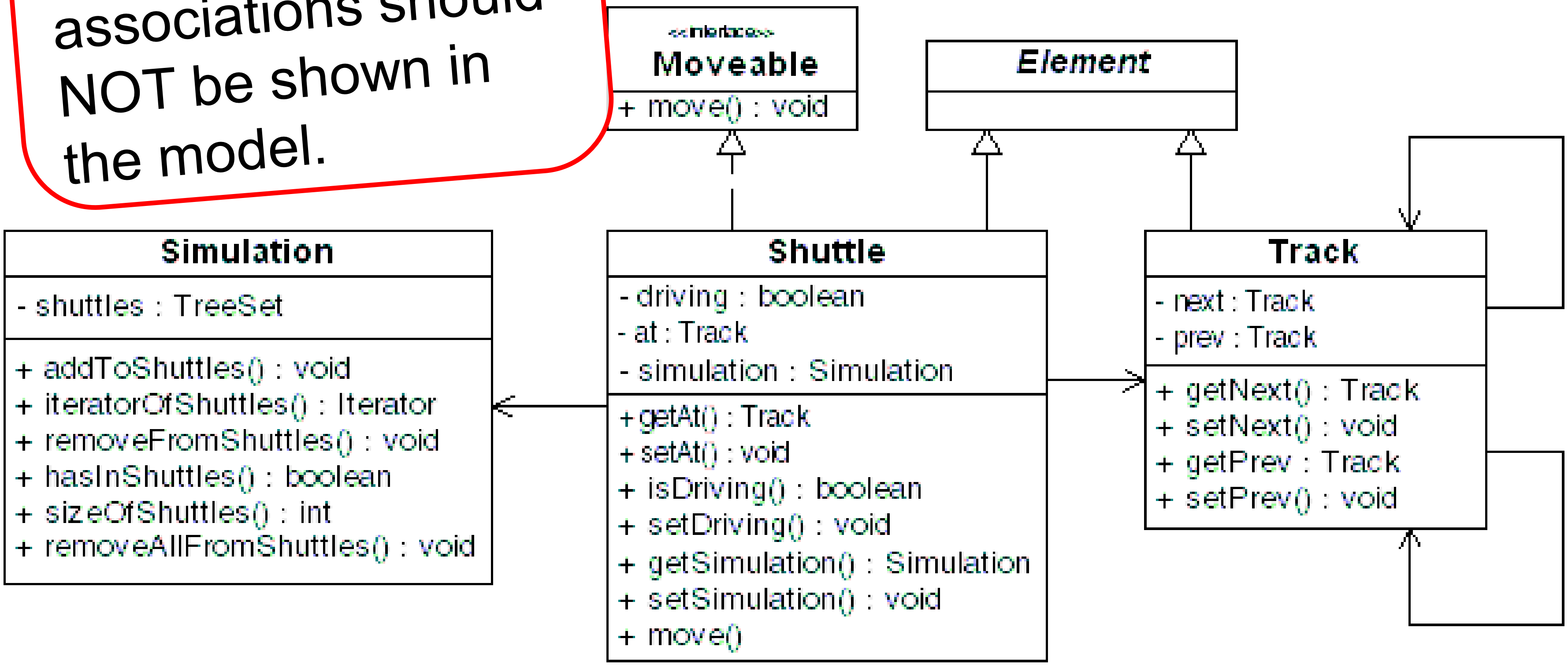
Example: Code

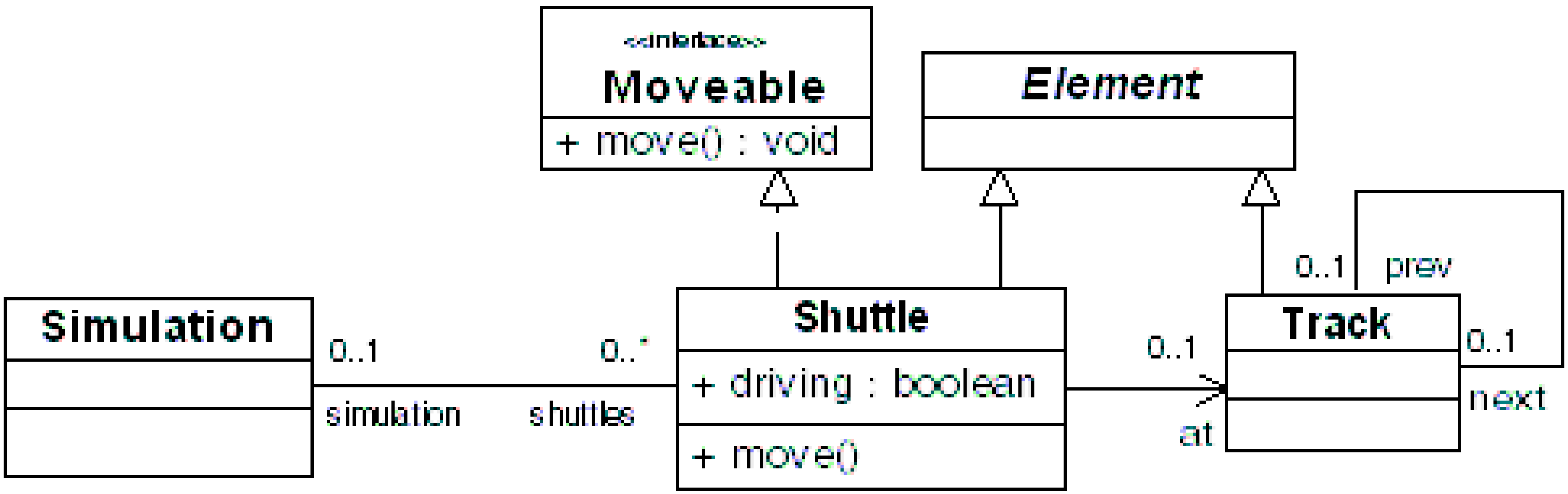

```
public class Simulation {

   private TreeSet shuttles = new TreeSet();
   public void addToShuttles(Shuttle value) {
      if (value != null) {
         boolean changed = this.shuttles.add (value);
         if (changed) {
            value.setSimulation (this);
         }
      }
   }
   public Iterator iteratorOfShuttles() {
      return this.shuttles.iterator ();
   }
   public void removeFromShuttles(Shuttle value) {
      if (value != null) {
         boolean changed = this.shuttles.remove
(value);
         if (changed) {
            value.setSimulation (null);
         }
      }
   }
   public boolean hasInShuttles(Shuttle value) {
...
   }
   public int sizeOfShuttles() {
      ...
   }
   public void removeAllFromShuttles() {
            ...
   }
}
```

Example: Result (tool)

NB: "Getters and setter methods" for class attributes and associations should NOT be shown in the model.





<<interface>>
Moveable
+ move() : void
Element
Simulation
0..1
simulation
0..*
shuttles
Shuttle
+ driving : boolean
+ move()
0..1
at
Track
0..1
prev
0..1
next

Tools (cntd.)

- Much information missing (wrong)
- Redundant information

- Typically, the models cover the structure only; behaviour models missing
- The results that tools come up with are on a very low level of abstraction (class diagrams or very basic design patterns)

- → Still very helpful (and current research improves the situation)

In this course

- We start from existing project (ePNK)
- Models are part of the software;
  (it won‘t be necessary to retrieve them)
- We don‘t need to reverse engineer the main structure of the software (domain model)
  (but some ideas might be hidden in the manually written code).

2.3 Model based SE

Today: We can generate parts of the code form the UML class diagrams automatically (MDA, MDE, **EMF**, EMFT/GMF)

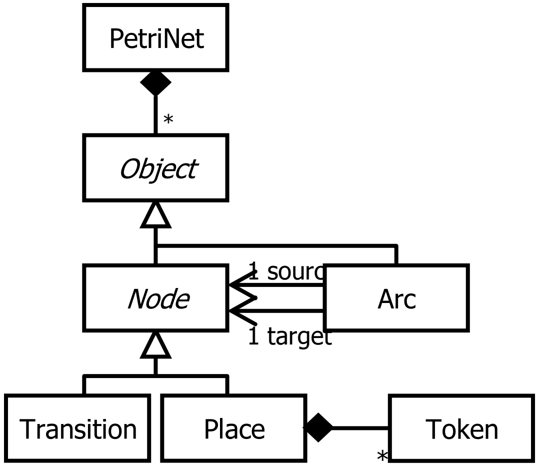
- Class diagrams → Java class stubs with standard access methods (see RE example)
- Implementation of standard behaviour:
  - Loading and saving models
  - Accessing and modifying the models
  - Editors and graphical user interfaces
- The actual functions is implemented by hand

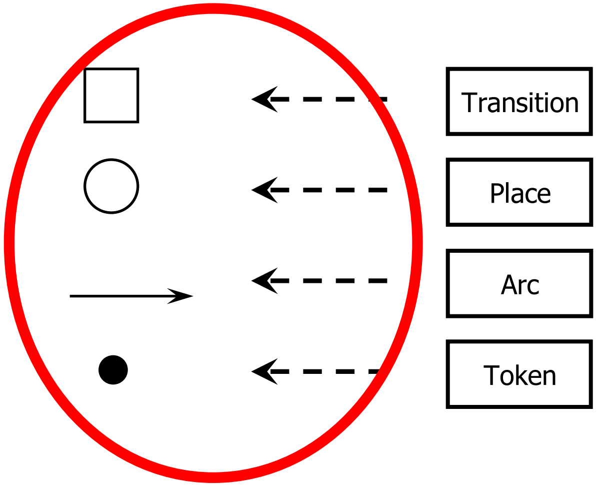
Future: Actual functions also „modelled“ and code generated

My favourite example

From this (EMF) model for Petri nets:
Generation of (Java) code for

- all classes
- methods for changing the Petri net
- loading and saving the Petri net as XML files (→XMI)





DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

The domain models are an (the) essential part of the software

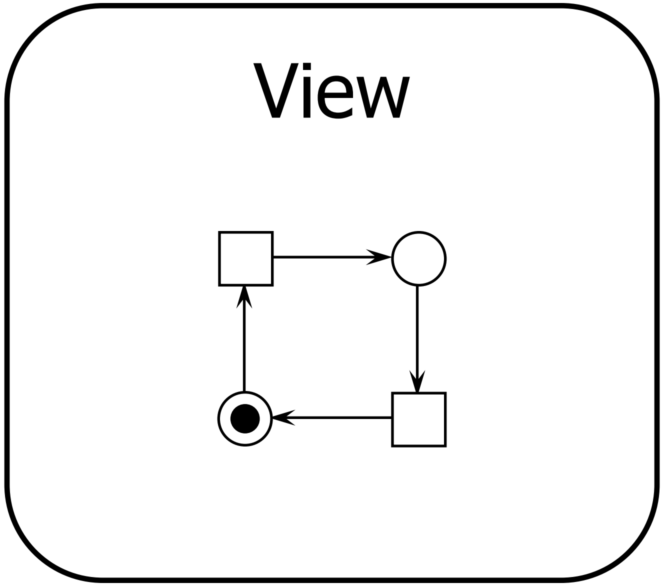
In addition to that we need

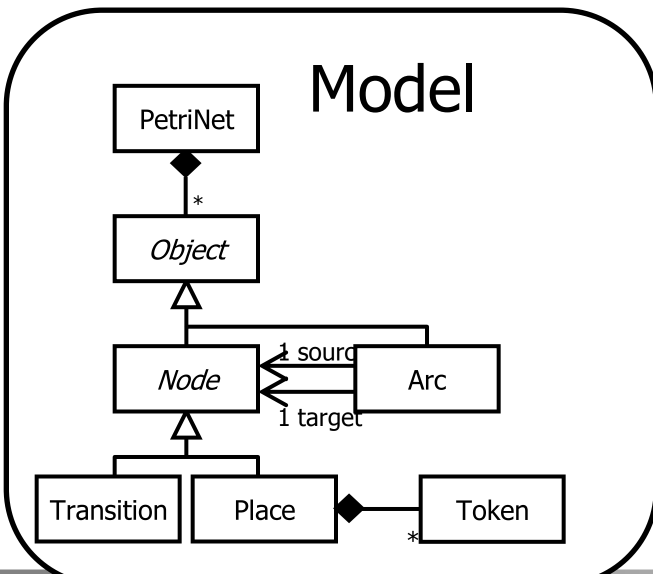
- Information about the presentation of the model to the user
- The coordination with the user

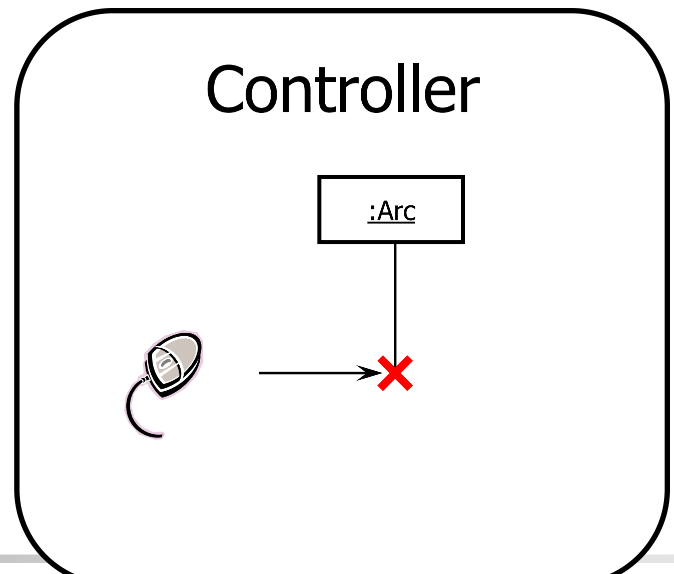
Note: These parts of the software can be modelled too (don't get confused: „models are everywhere"); domain model vs. software model

Modelle View Controller (MVC)

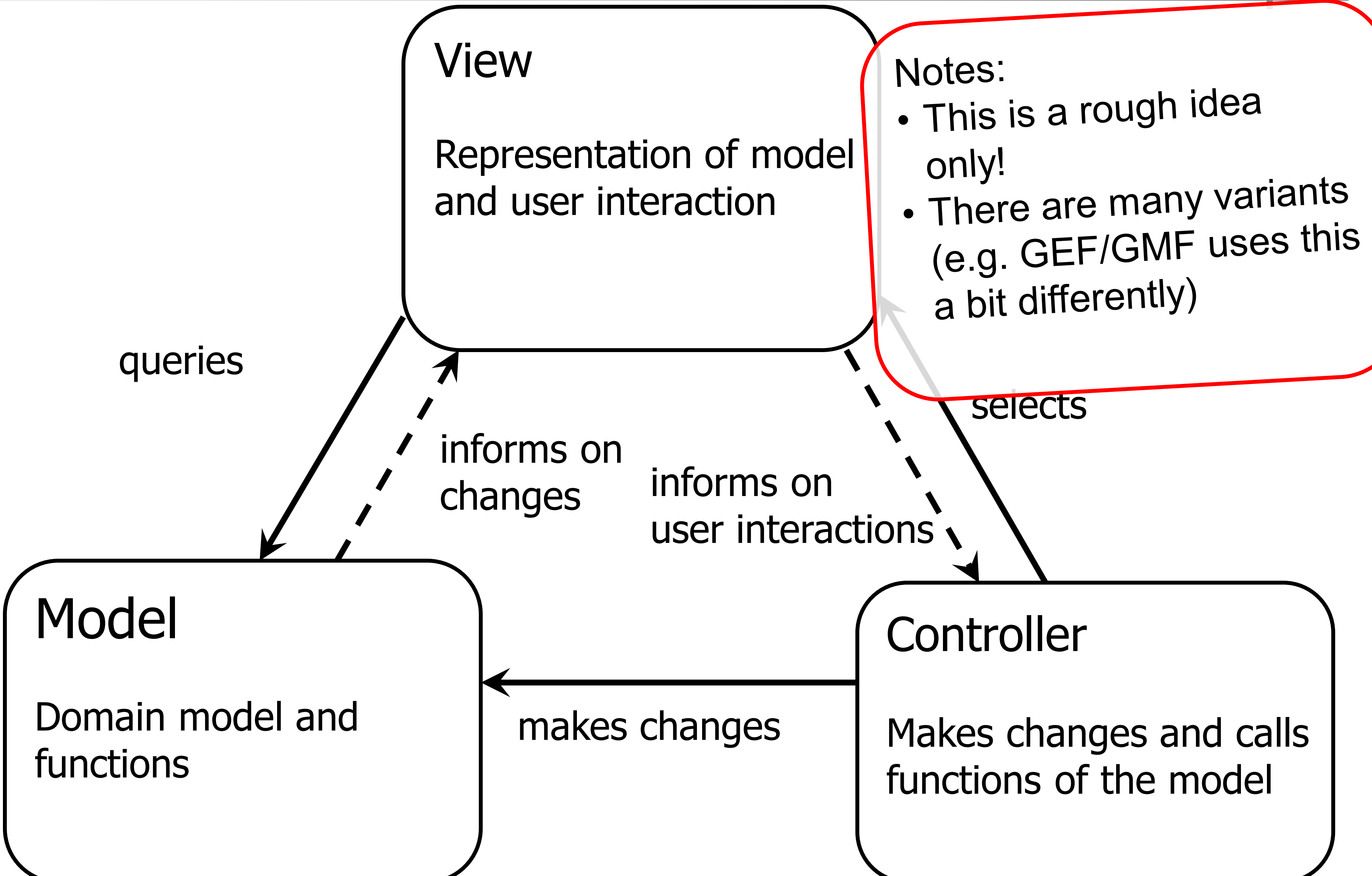
DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler







MVC

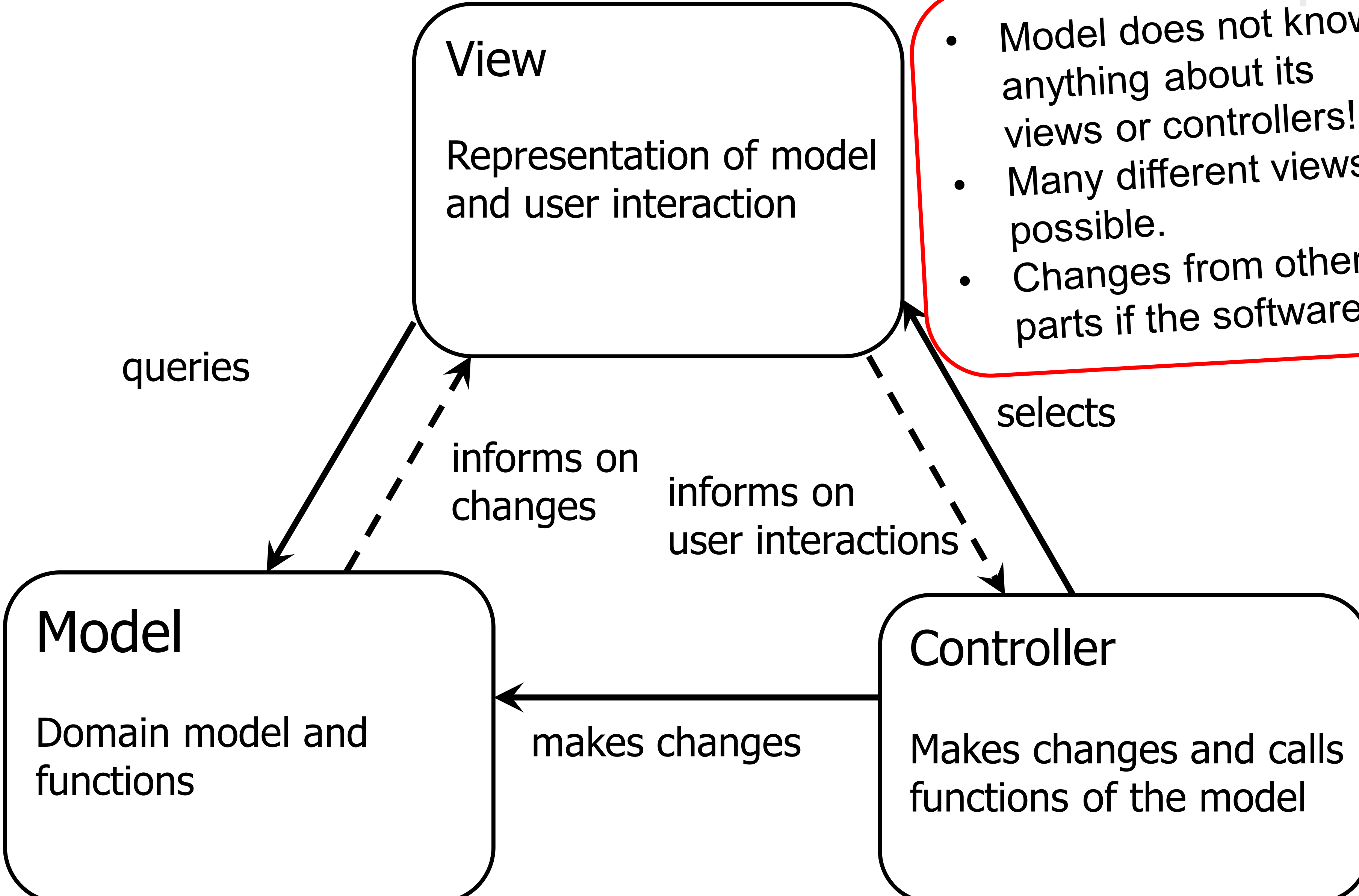


Notes:

- This is a rough idea only!
- There are many variants (e.g. GEF/GMF uses this a bit differently)

MVC

- Model does not know anything about its views or controllers!
- Many different views possible.
- Changes from other parts if the software.



MVC

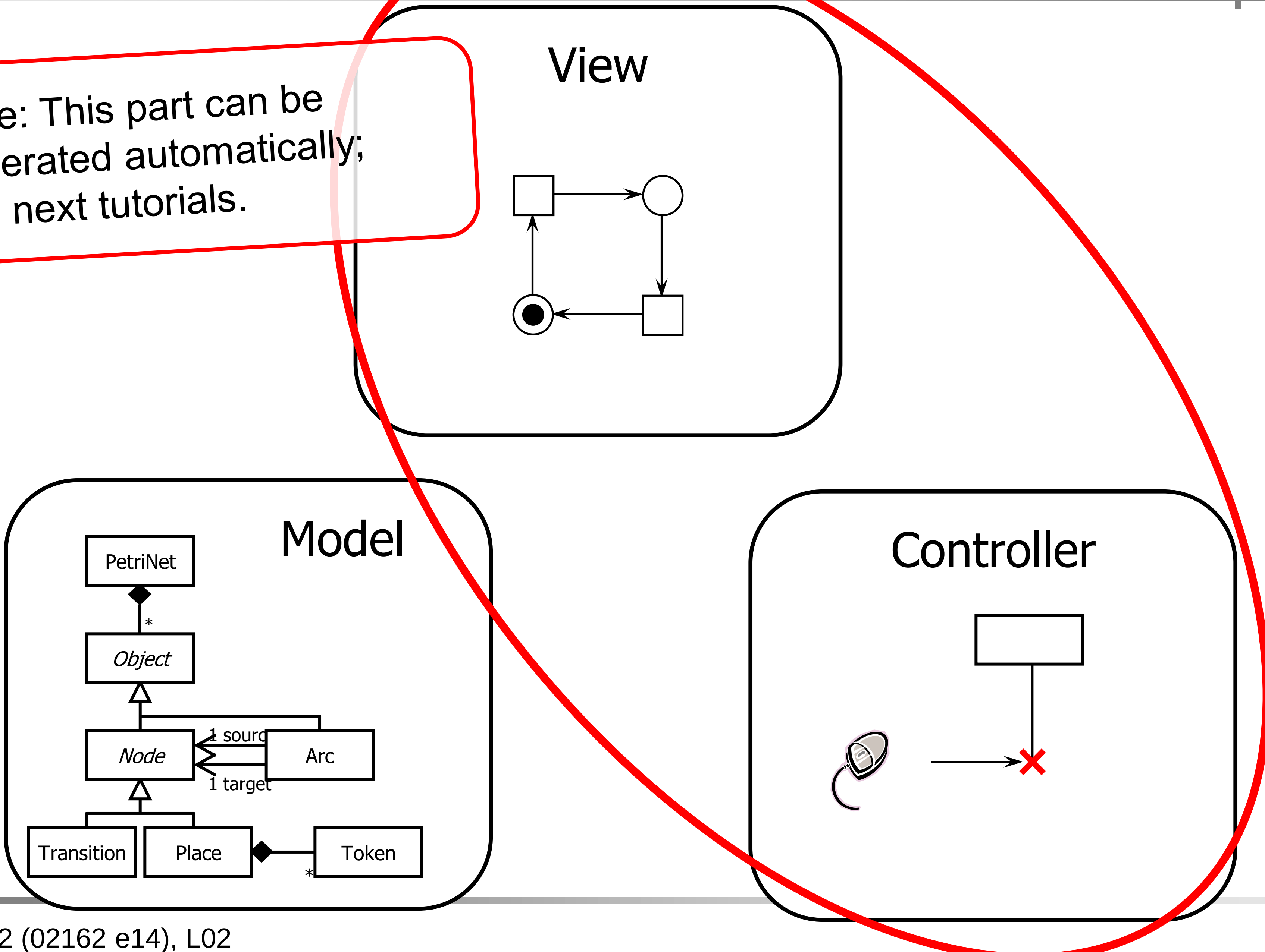
MVC is a principle (pattern / architecture) according to which software should be structured

Eclipse and GEF (as well as GMF) are based on this principle and guide (force) you in properly using it

If things do not work out with EMF for you, you might have messed with the MVC pattern.

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

Here: This part can be generated automatically; see next tutorials.



Originally, the term was used in architecture: Alexander et al. 1977.

Design patterns (in software engineering) are the distilled experience of software engineering experts on how to solve standard problems in software design.

Freeman & Freeman call this "experience reuse"!

From the MBSE point of view, this is only half the way!

Excursion Design Patterns

Often called the "Gang of Four" (GoF / Go4).

- Gamma, Helm, Johnson, Vlissides: Design Patterns. Addison-Wesley 1995.
- Eric Freeman, Elisabeth Freeman: Head First Design Patterns. O'Reilly 2004.
- …

Disclaimer

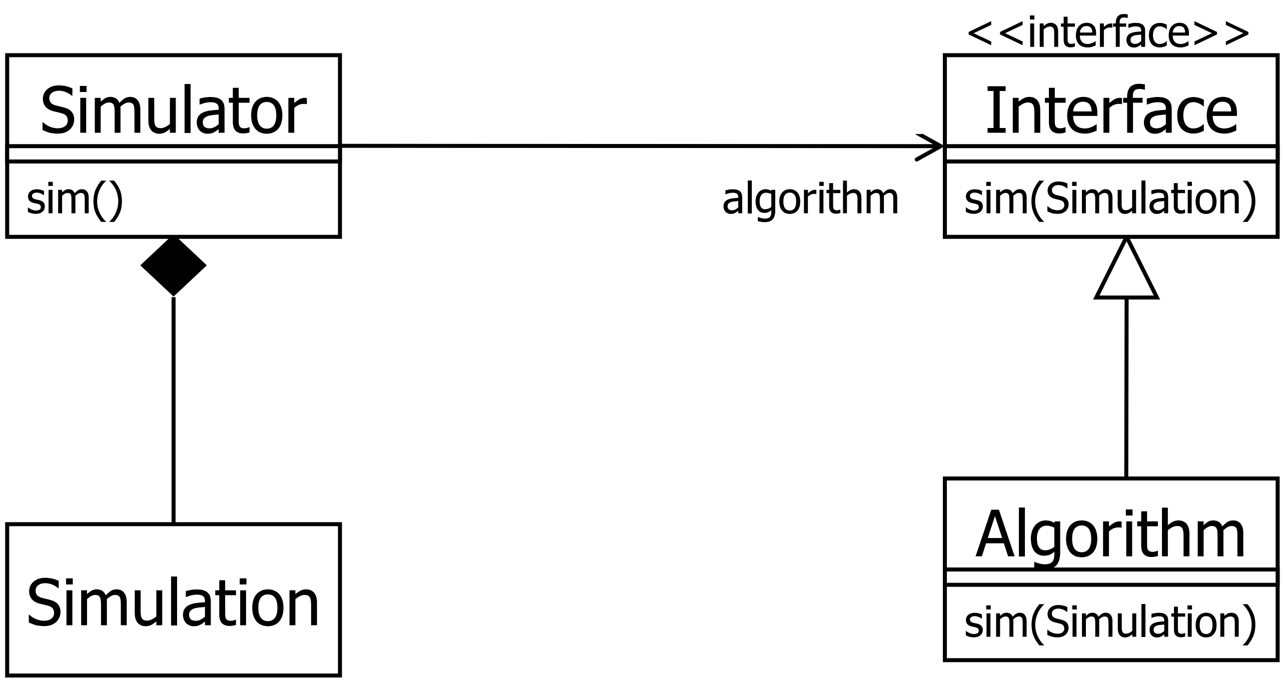
- Design patterns is a topic of its own
- Worth being taught as a separate course (e.g. seminar)
- This excursion gives just a glimpse of the idea and some recurring patterns

Scheme (GoF)

- Name and classification
- Intent
- Also known as
- Motivation
- Application
- Structure
- Participants
- Collaboration
- Consequences
- Implementation
- Sample code
- Known uses
- Related patterns

Sometimes there is more:
Variants, counter indications,
…

Example: Simulation Algorithm



Pattern: Strategy (GoF)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

Name and classification

Strategy, object-based, behavioural

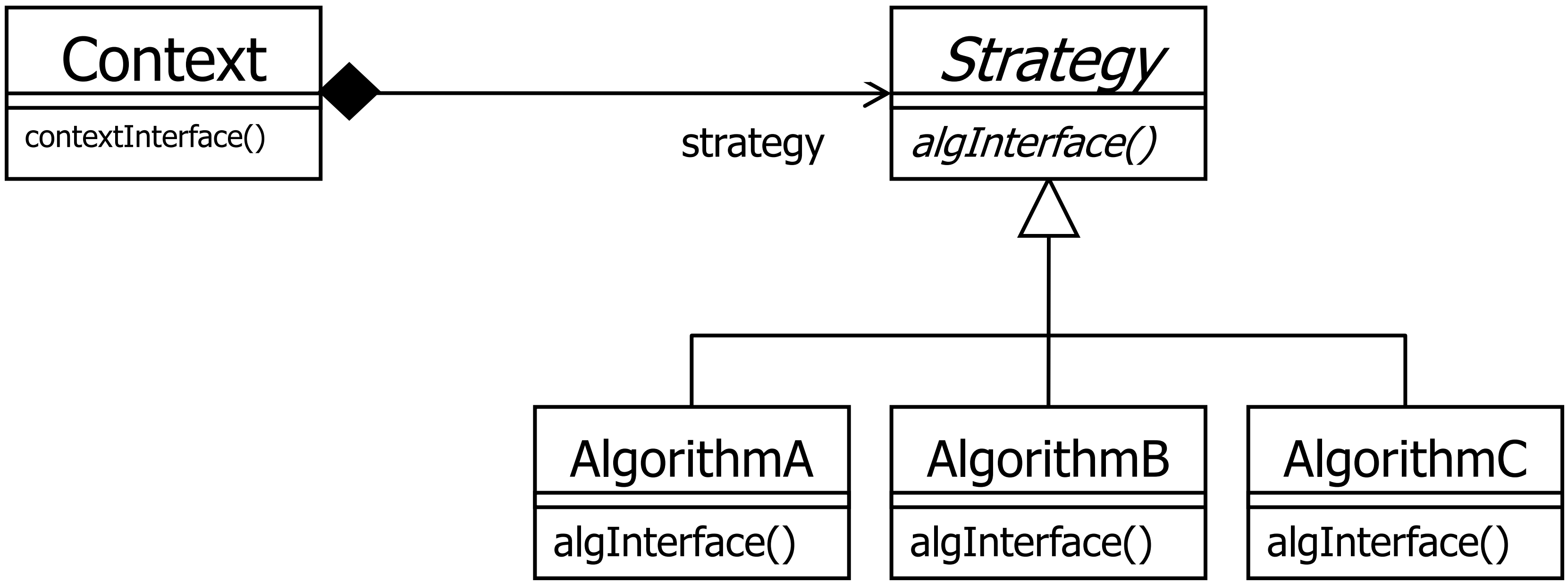
Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it [GoF]

Motivation

Avoid hard-wiring of algorithms for making it easier to change the algorithm …

Structure


Context
contextInterface()
strategy
Strategy
algInterface()
AlgorithmA
algInterface()
AlgorithmB
algInterface()
AlgorithmC
algInterface()

Pattern: Strategy (cntd)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

We skip the rest of the GoF scheme here.

Questions

- Is the "simulation algorithm" a strategy?
- Is the plugIn of simulation algorithms to the simulation manager a strategy in the CASE Tool?

Patterns should not be applied too mechanically!
But sometimes details make a difference (e.g. State Pattern vs. Strategy)

Pattern: Abstract Factory

Name and classification

Abstract factory, object-based, creational

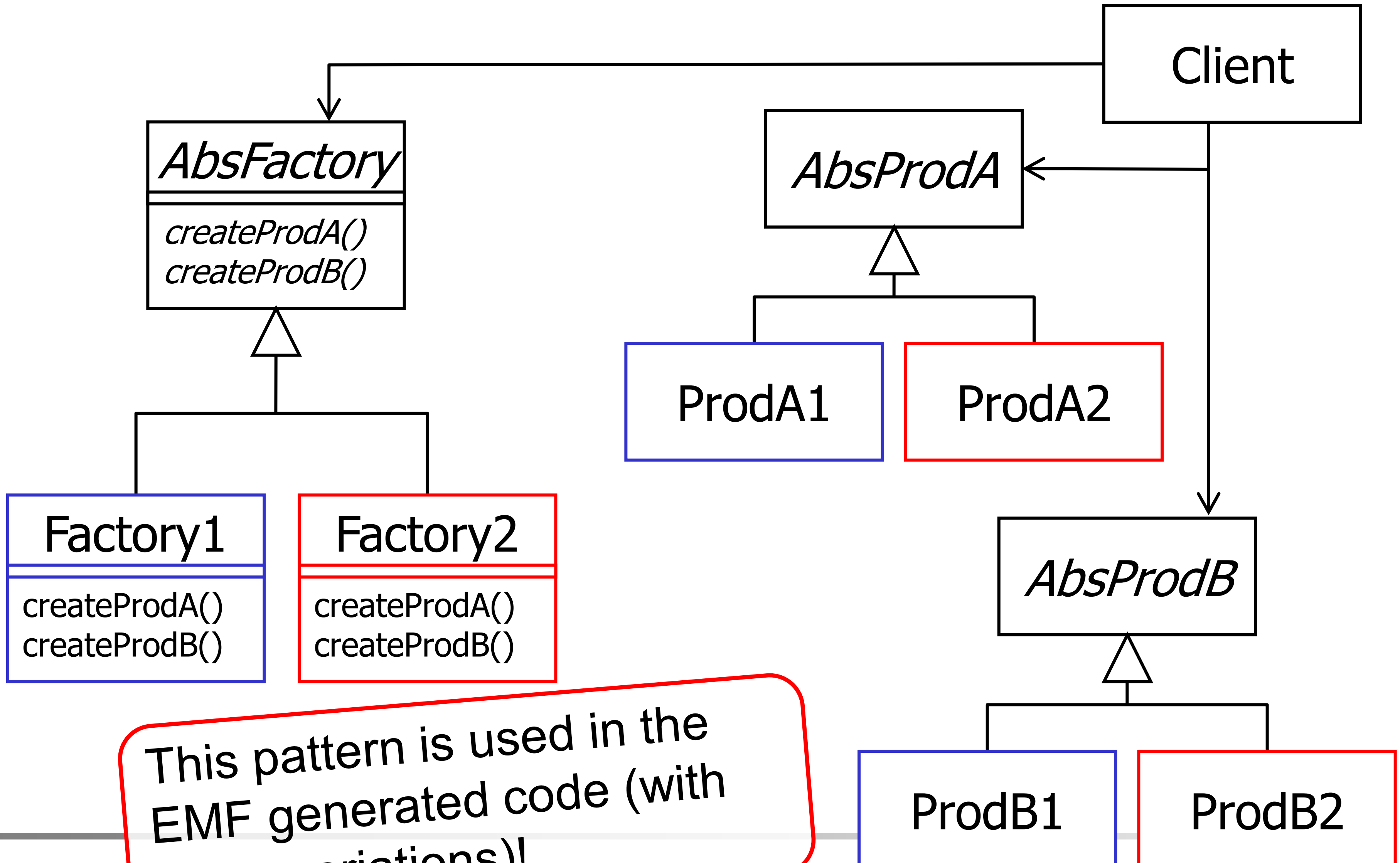
Intent

Provide an interface for creating families of related or dependent objects without specifying their concrete classes [GoF]

Motivation

Use of different implementations in different contexts with easy portability …

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler



This pattern is used in the EMF generated code (with some variations)!

Name and classification

Singleton, object-based, creational

Intent

Ensure that a class has only one instance, and provide a global point of access to it [GoF]

Motivation

…

See [GoF] or [FF] for details.

Pattern: Observer (GoF)

Name and classification

Observer, object-based, creational

- **Intent**

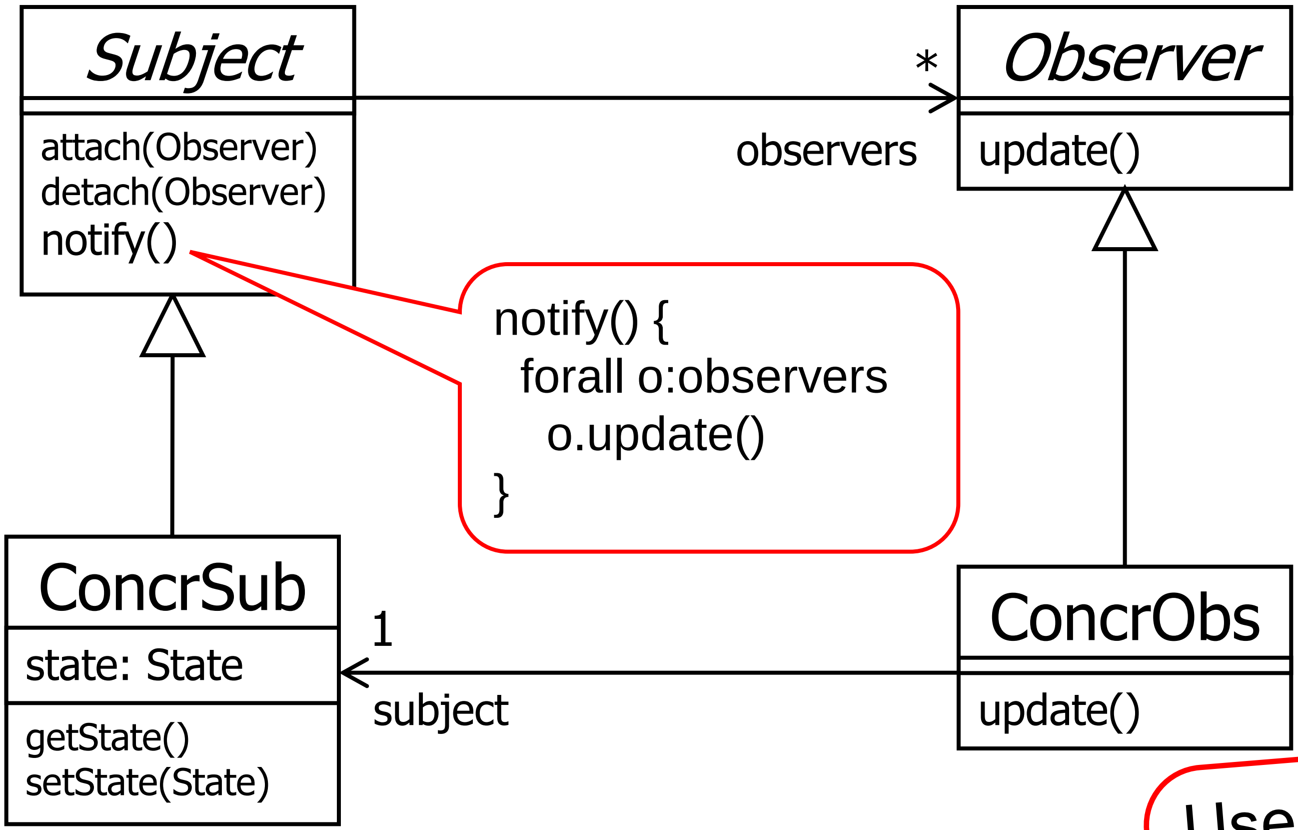
  Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically [GoF]

Motivation

Update a view when the model (subject) changes …

Observer (structure)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler



Used in MVC and in EMF/GMF editors (observers are called “adaptors” there).

Summary

- GoF present 23 patterns
- There are many more (and more complex combinations of patterns, e.g. MVC)

- "Pattern terminology" can be used to communicate design!
- Patterns should not be used to schematically (when used manually)
- Generated code, typically, makes use of many patterns. Automatic code generation "saves us making some design decisions" (observer, singleton, factory are part of the EMF-generated code)

Example

- Discussion of a simple model in the project session of today's course!