22. Entwurfsmuster (Design Patterns) - Eine Einführung

Prof. Dr. Uwe Aßmann
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
19-1.1, 5/11/19

1) Warum Entwurfsmuster

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

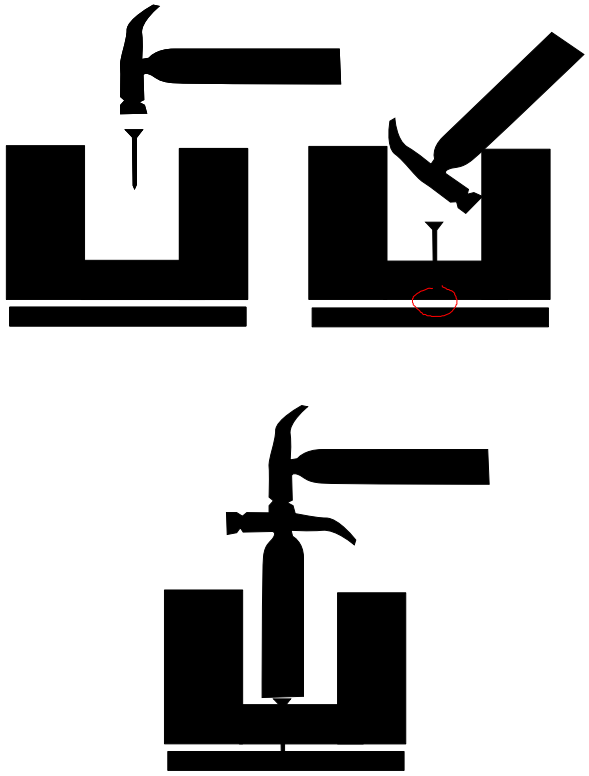
Achtung: Dieser Foliensatz ist teilweise in Englisch gefasst, weil das Thema in der Englisch-sprachigen Kurs “Design Patterns and Frameworks” wiederkehrt.
Mit der Bitte um Verständnis.

Obligatorische Literatur

- JDK Tutorial für J2SE oder J2EE, www.java.sun.com
- Dokumentation der Jgrapht library http://www.jgrapht.org/
  - Javadoc http://www.jgrapht.org/javadoc
  - http://sourceforge.net/apps/mediawiki/jgrapht/index.php?title=jgrapht:Docs
- Dokumentation der Library für verteilte Graphen GELLY (Teil von Apache Flink)
  - http://ci.apache.org/projects/flink/flink-docs-master/gelly_guide.html

Obligatory Literature

- ST für Einsteiger, Kap. Objektentwurf: Wiederverwendung von Mustern
- also: Chap. 8, Bernd Brügge, Allen H. Dutoit. Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java. Pearson.



Recommended Books

- [The GOF (Gang of Four) Book] E. Gamma, R. Johnson, R. Helm, J. Vlissides. Design Patterns. Addison-Wesley.
  - Auf Deutsch: Entwurfsmuster.
- Head First Design Patterns. Eric Freeman & Elisabeth Freeman, mit Kathy Sierra & Bert Bates.O'Reilly, 2004, ISBN 978-0-596-00712-6
  - German Translation: Entwurfsmuster von Kopf bis Fuß. Eric Freeman & Elisabeth Freeman, mit Kathy Sierra & Bert Bates. O'Reilly, 2005, ISBN 978-3-89721-421-7
- There is a lot of free material on the web.
  - http://en.wikipedia.org/wiki/Book:Design_Patterns is a free collection of patterns, available as pdf
  - James W. Cooper. Java™ Design Patterns: A Tutorial. Addison Wesley, 2000, ISBN: 0-201-48539-7
  - http://www.informit.com/store/java-design-patterns-a-tutorial-9780201485394 Section Download
  - Download books at http://www.freebookcentre.net/SpecialCat/Free-Design-Patterns-Books-Download.html

Introductory Papers, Recommended

- A. Tesanovic. What is a pattern? Paper in Design Pattern seminar, IDA, 2001. Available at ST
  http://www-st.inf.tu-dresden.de/Lehre/WS04-05/dpf/seminar/tesanovic-WhatIsAPattern.pdf
- Brad Appleton. Patterns and Software: Essential Concepts and terminology.
  - http://csis.pace.edu/~grossman/dcs/Patterns%20and%20Software-%20Essential%20Concepts%20and%20Terminology.pdf Compact introduction into patterns.
  -

Other References

- F. Buschmann. N. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-orientierte Software-Architektur. Addison-Wesley.
  - Design patterns and architectural styles. MVC, Pipes, u.v.m.
- M. Fowler. Refactoring. Addison-Wesley.
- W. Pree. Object-Oriented Software Construction. 1995. Springer.
- Papers:
  - D. Riehle, H. Zülinghoven, Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems 2 (1), 1996. Explains different kinds of patterns. http://citeseer.ist.psu.edu/riehle96understanding.html
  - W. Zimmer. Relationships Between Design Patterns. Pattern Languages of Programming (PLOP) 1995.

- MVC
  - http://exadel.com/tutorial/struts/5.2/guess/strutsintro.html
  - http://www.c2.com/cgi/wiki?ModelViewController
- “Quality without a name” (QWAN principle)
  - http://en.wikipedia.org/wiki/The_Timeless_Way_of_Building

Software Engineering is a Positive Activity

Thus it will be seen that *engineering* is a distinctive and important profession. To some even it is the topmost of all professions. However true that may or may not be to-day, certain it is that some day it will be true, for the reason that engineers serve humanity at every practical turn.

Engineers make life easier to live—easier in the living; **their work is strictly constructive, sharply exact; the results positive.**

Not a profession outside of the engineering profession but that has its moments of wabbling and indecision—of faltering on the part of practitioners between the true and the untrue. Engineering knows no such weakness. Two and two make four. Engineers know that. Knowing it, and knowing also the unnumbered possible approach a problem with a certainty of conviction and a confidence in the powers of their working-tools nowhere permitted men outside the profession.

Charles M. Horton. Opportunities of engineering. www.gutenberg.org, eBook #24681; Harper and Brothers, 1922.

“The Tiger” (William Blake)

http://www.gutenberg.org/files/1934

Tiger, tiger, burning bright
In the forests of the night,
What immortal hand or eye
Could frame thy fearful symmetry?

In what distant deeps or skies
Burnt the fire of thine eyes?
On what wings dare he aspire?
What the hand dare seize the fire?

And what shoulder and what art
Could twist the sinews of thy heart?
And, when thy heart began to beat,
What dread hand and what dread feet?

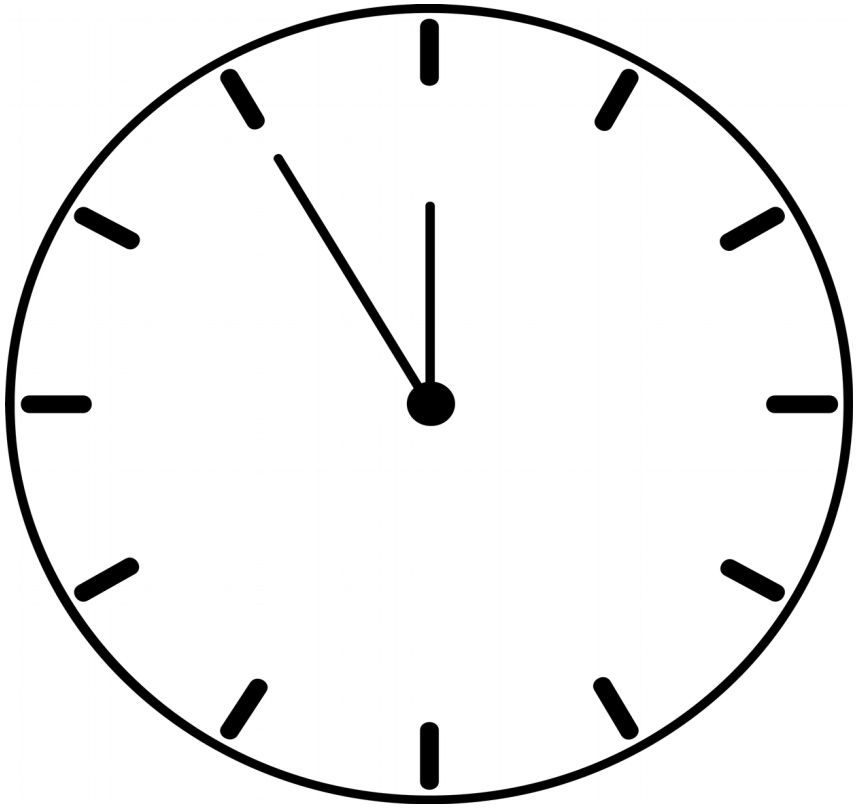
What the hammer? what the chain?
In what furnace was thy brain?
What the anvil? what dread grasp
Dare its deadly terrors clasp?

When the stars threw down their spears,
And watered heaven with their tears,
Did He smile His work to see?
Did He who made the lamb make thee?

Tiger, tiger, burning bright
In the forests of the night,
What immortal hand or eye
Dare frame thy fearful symmetry?

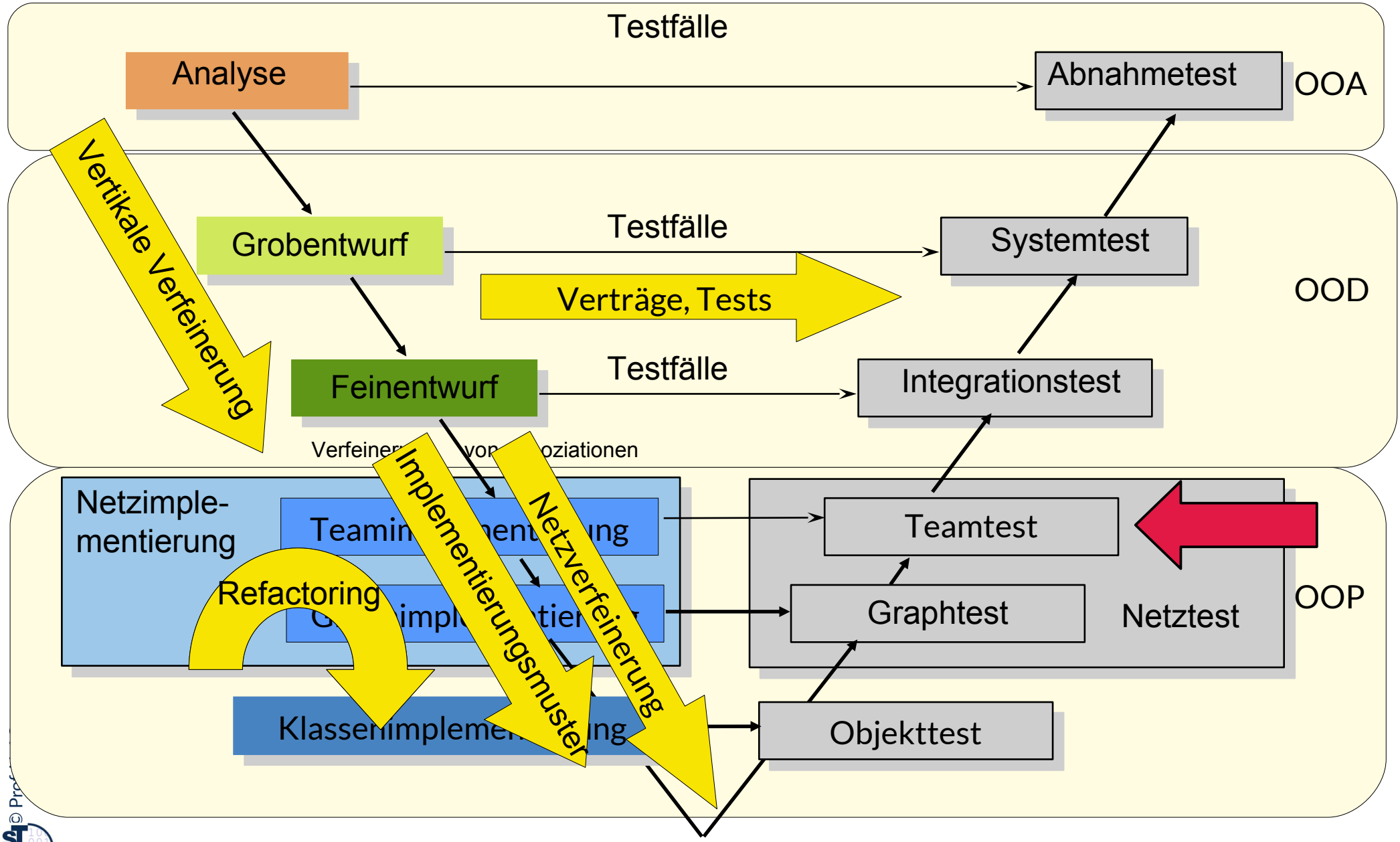
Vorsicht

- Wer hat schon ein Java Programm übersetzt?

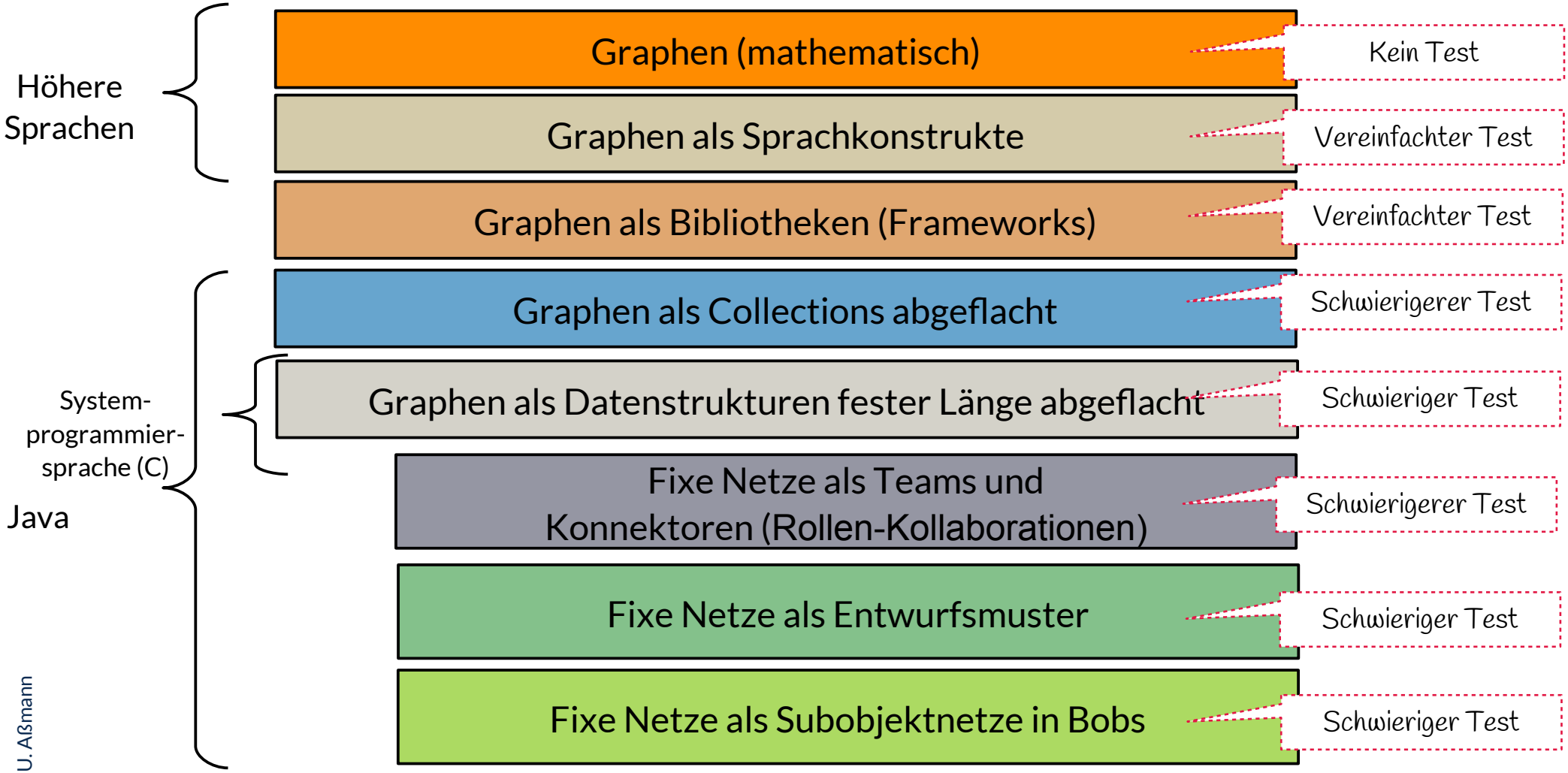


Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



Repräsentation von flexiblen und fixen Objektnetzen als Datenstrukturen (Netzverfeinerung)



22.1. Warum Entwurfsmuster?

Why is the Frauenkirche Beautiful?

History: How to Write *Beautiful* Software

- Beginning of the 70s: the window and desktop metaphors (conceptual patterns) are discovered by the Smalltalk group in Xerox Parc, Palo Alto
- 1978/79: Goldberg and Reenskaug develop the MVC pattern for user Smalltalk interfaces at Xerox Parc
  - During porting Smalltalk-78 for the Eureka Software Factory project
- 1979: Alexander's Timeless Way of Building
  - Introduces the notion of a *pattern* and a *pattern language*
- 1987: W. Cunningham, K. Beck OOPSLA paper "Using Pattern Languages for Object-Oriented Programs" discovered Alexander's work for software engineers by applying 5 patterns in Smalltalk
- 1991: Erich Gamma's PhD Thesis about Design Patterns
  - Working with ET++, one of the first window frameworks of C++
  - At the same time, Vlissides works on InterViews (part of Athena)
- 1991: Pattern workshop at OOPSLA 91, organized by B. Anderson
- 1993: E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. ECOOP 97, LNCS 707, Springer
- 1995: First PLOP conference (Pattern Languages Of Programming)
- 1995: GOF book

The Most Popular Definition

Def.: A **Design Pattern (Entwurfsmuster)** is a *solution pattern*,

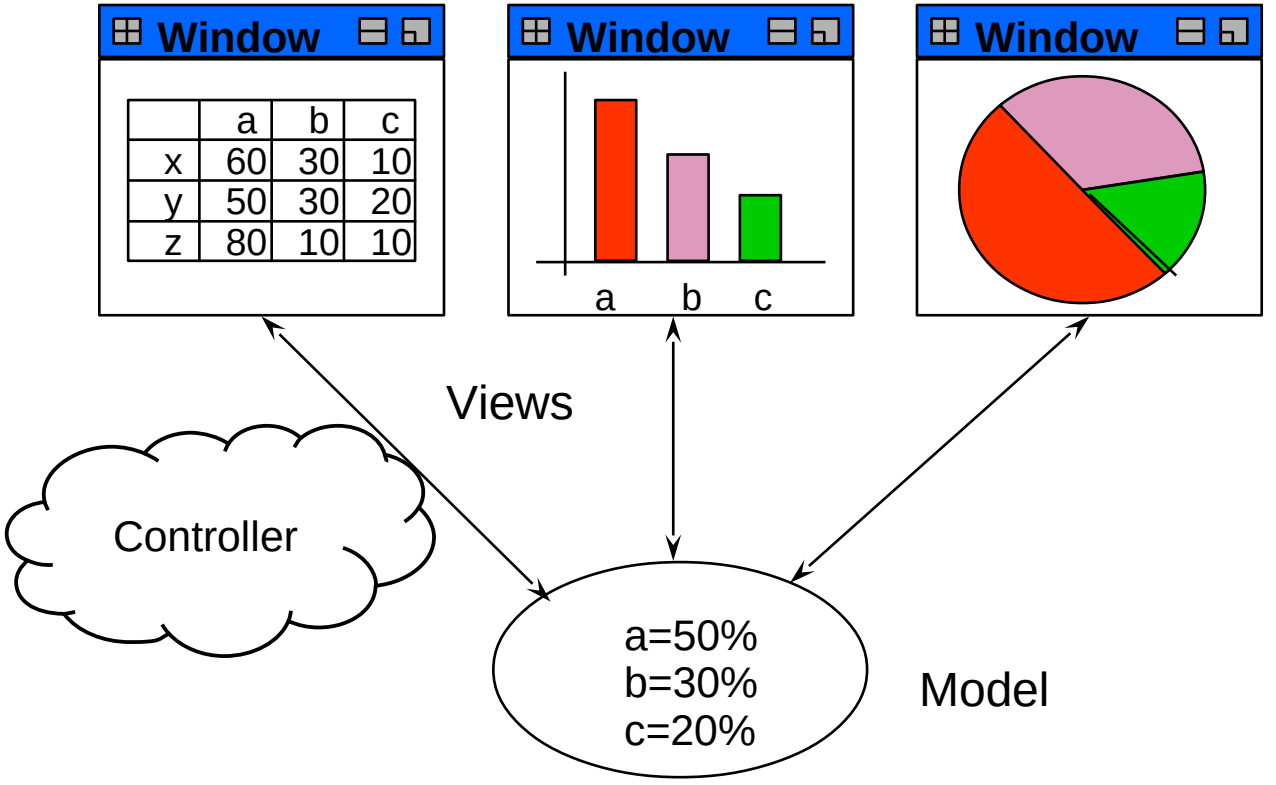
- a description of a standard solution for
- a frequent design problem
- in a certain context

- Goal of a Design Pattern: Reuse of design information
  - A pattern must not be “new”!
  - A pattern writer must have a “aggressive disregard for originality”

- Such *solution patterns* are well-known in every engineering discipline
  - Mechanical engineering
  - Electrical engineering
  - Civil engineering and architecture

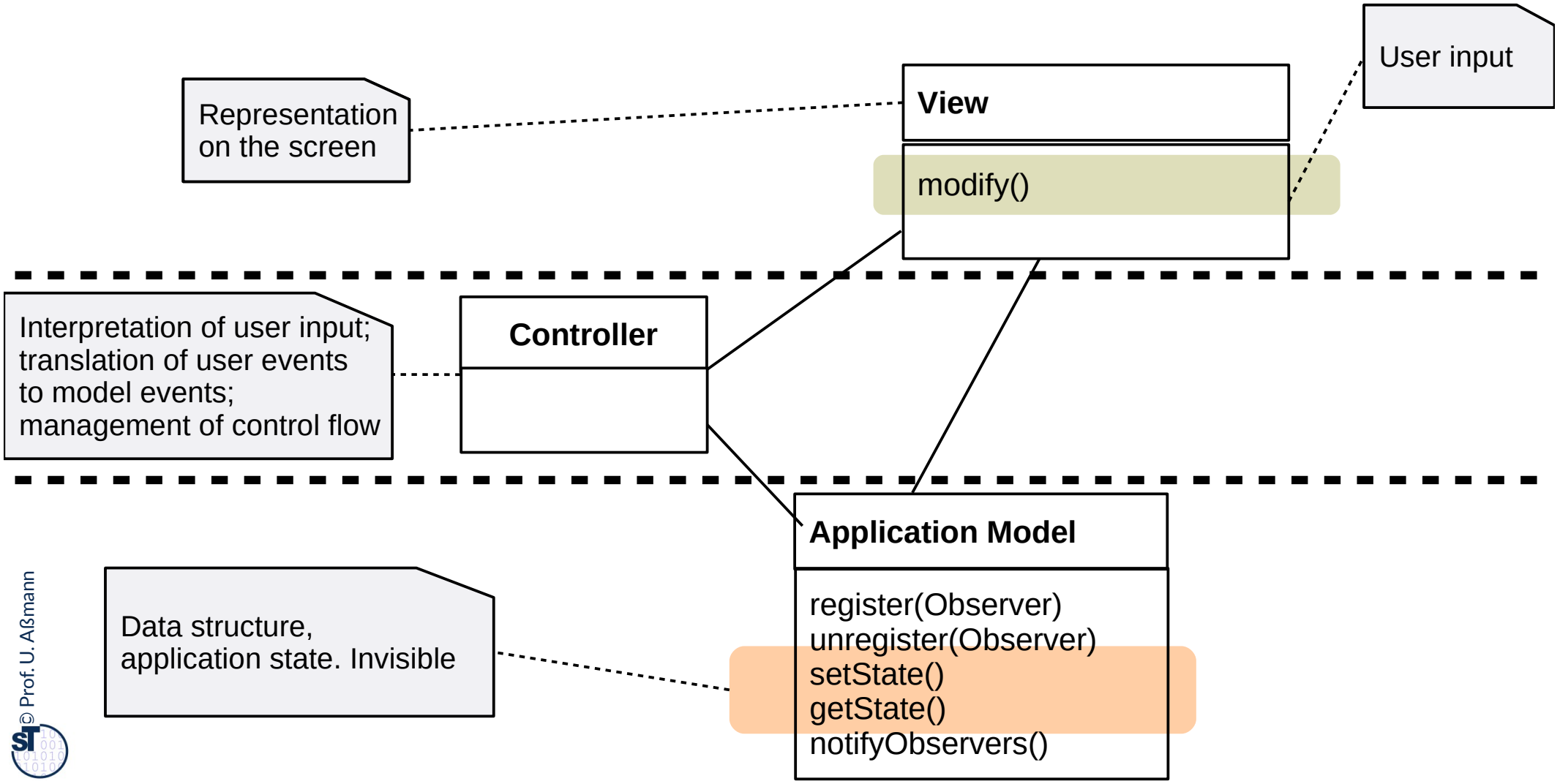
A Problem in Interactive Applications

- How do I display and edit a data structure on the screen?
  - Reaction on user inputs?
  - Maintaining several views
  - Adding and removing new views
- Solution: Model-View-Controller pattern (MVC), a set of classes to control a data structure behind a user interface
  - Developed by Goldberg/Reenskaug in Smalltalk 1978



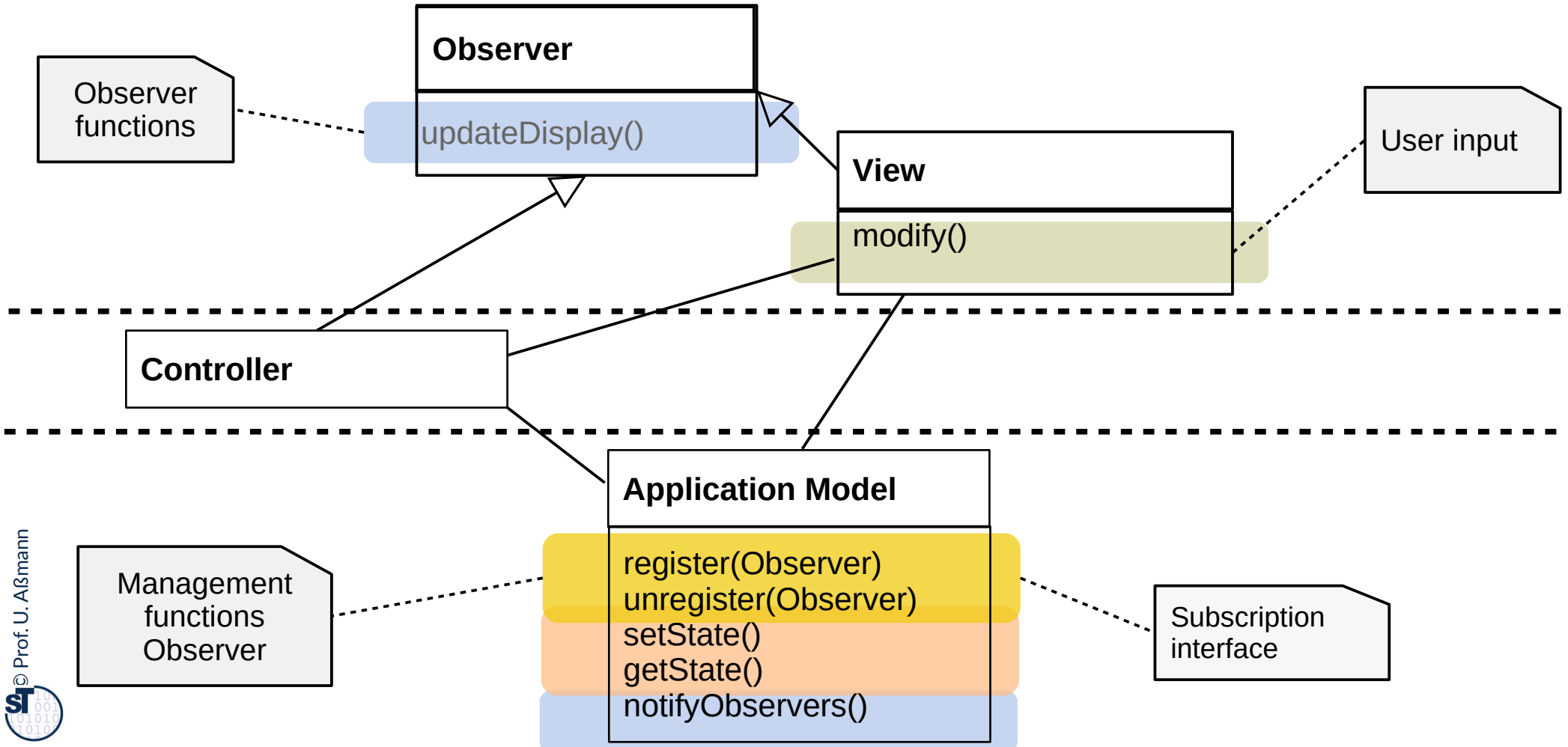
Design Pattern Model/View/Controller (MVC)

- MVC is a set of classes to control a data structure behind a user interface
- Layered structure of View, Controller and ApplicationModel



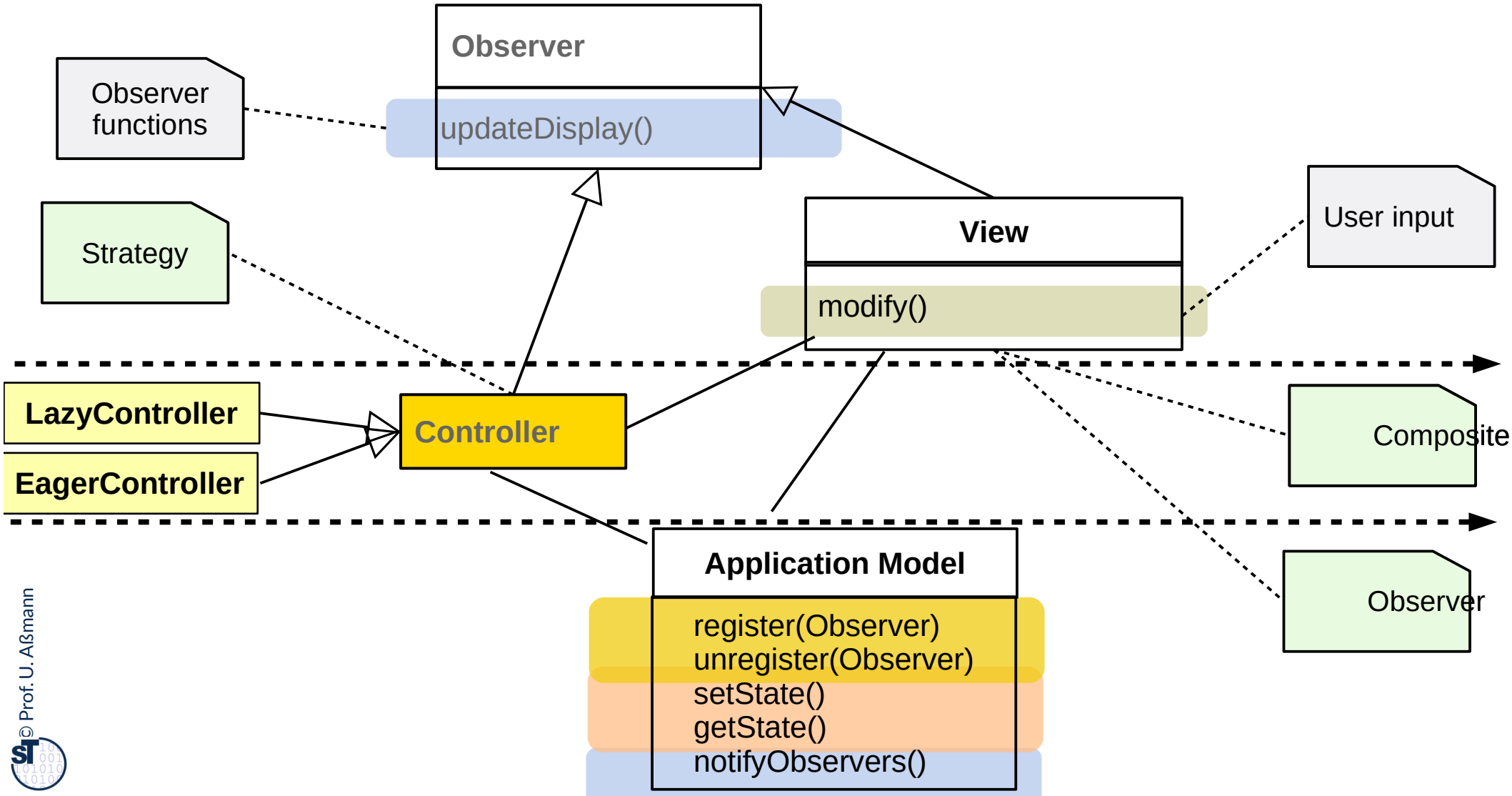
Design Pattern Model/View/Controller (MVC)

- The MVC is a complex design pattern. The layers are connected by the simpler patterns Observer, Composite, Strategy.
  - The Controller interpretes the input of the user and transmits them into actions on the model
  - Controller and View play Listener role from *Observer* (asynchronous communication)
  - Model plays Subject role



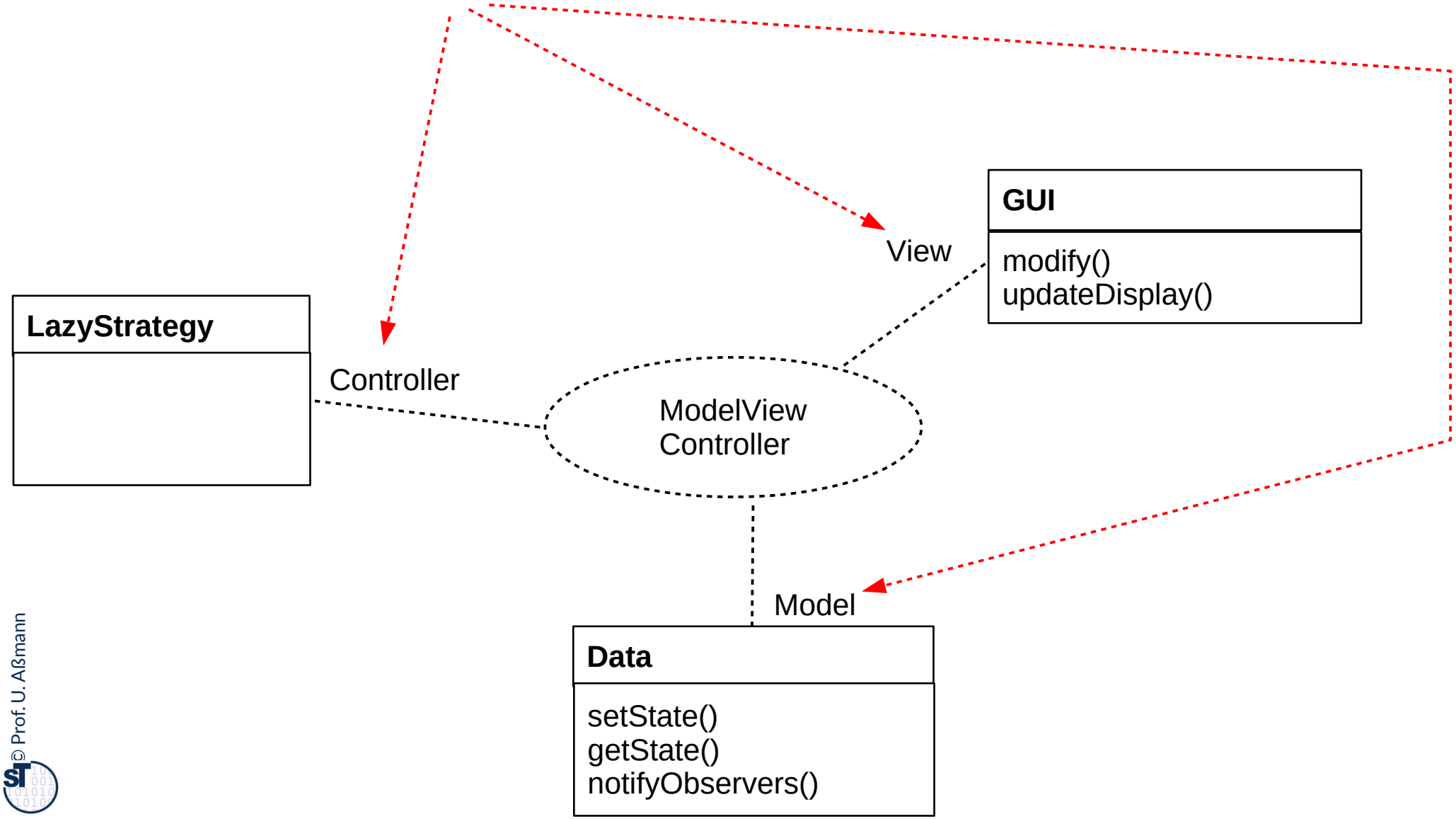
Design Pattern Model/View/Controller (MVC), Refined

- Controller follows Strategy pattern (variation of updating the screen)
- Relation within Views by Composite (tree-formed views)



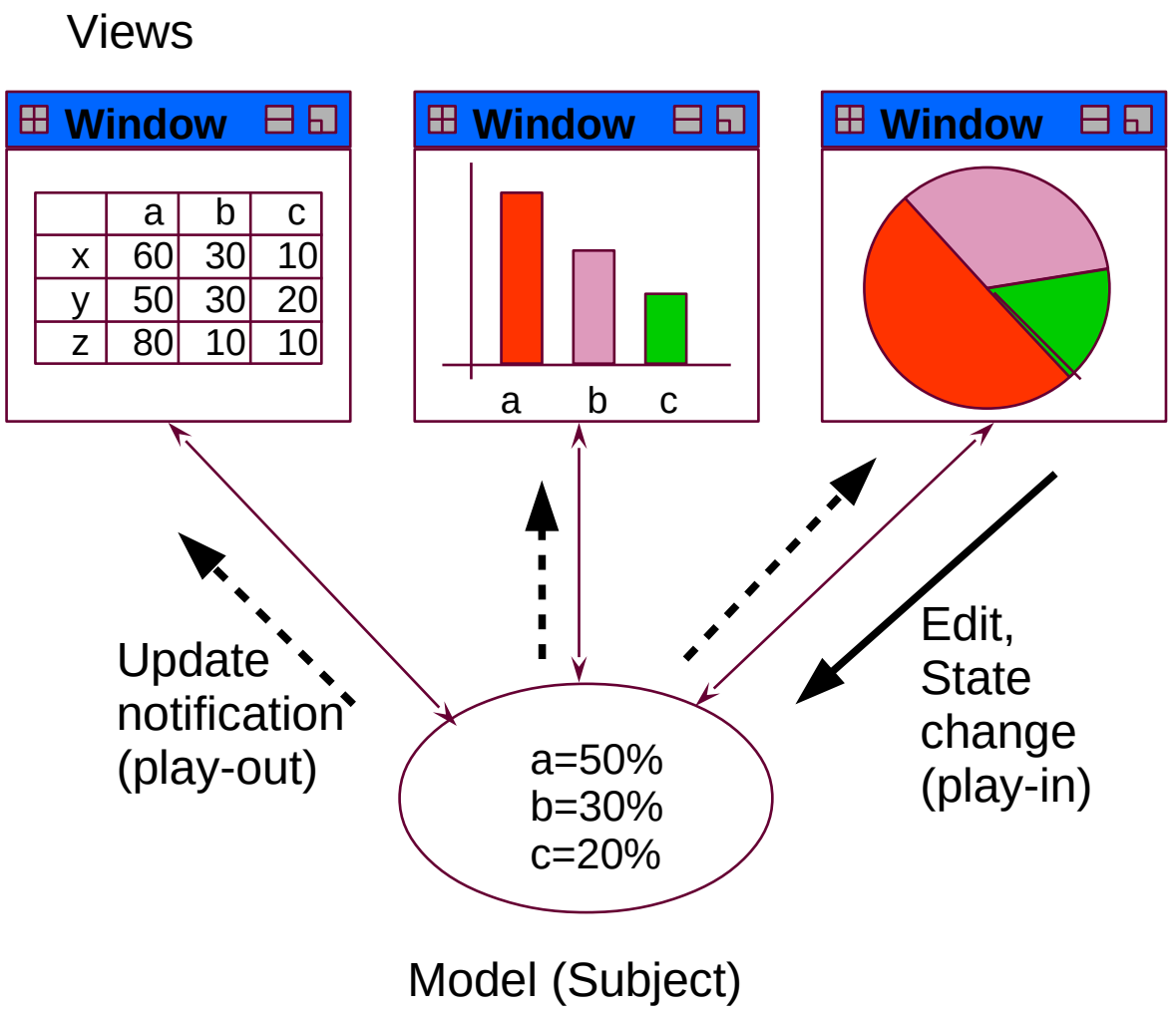
Design Pattern Model/View/Controller (MVC)

- UML has a specific notation for patterns (**collaboration classes**)
  - With role identifiers



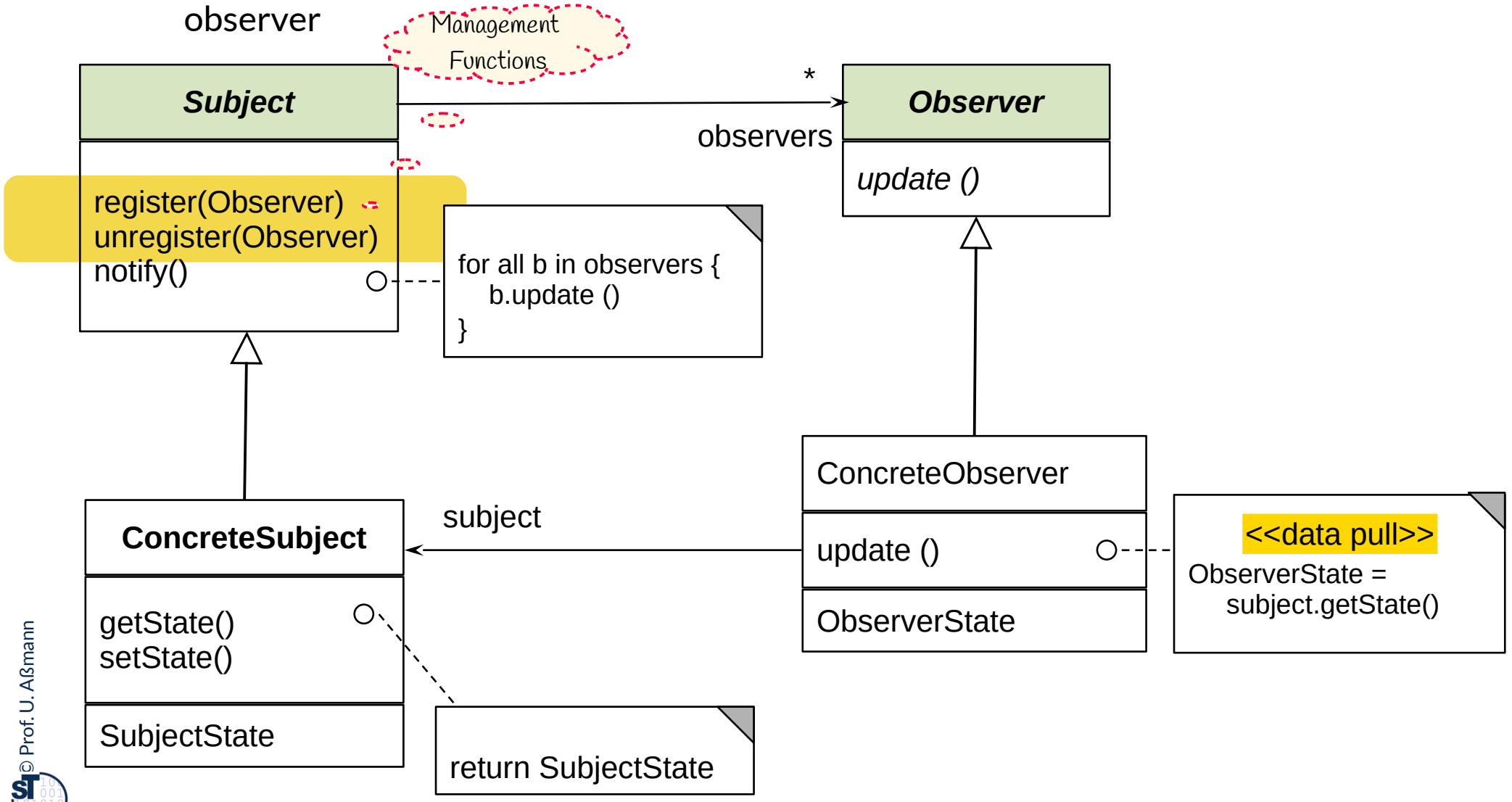
Pattern 1: Observer

- Views may register as Observer at the model (Subject)
  - They become *passive* observers of the model
  - They are notified if the model changes.
  - Then, every view updates itself by accessing the data of the model.
- Views are independent of each other
  - The model does not know how views visualize it
  - Observer decouples views strongly



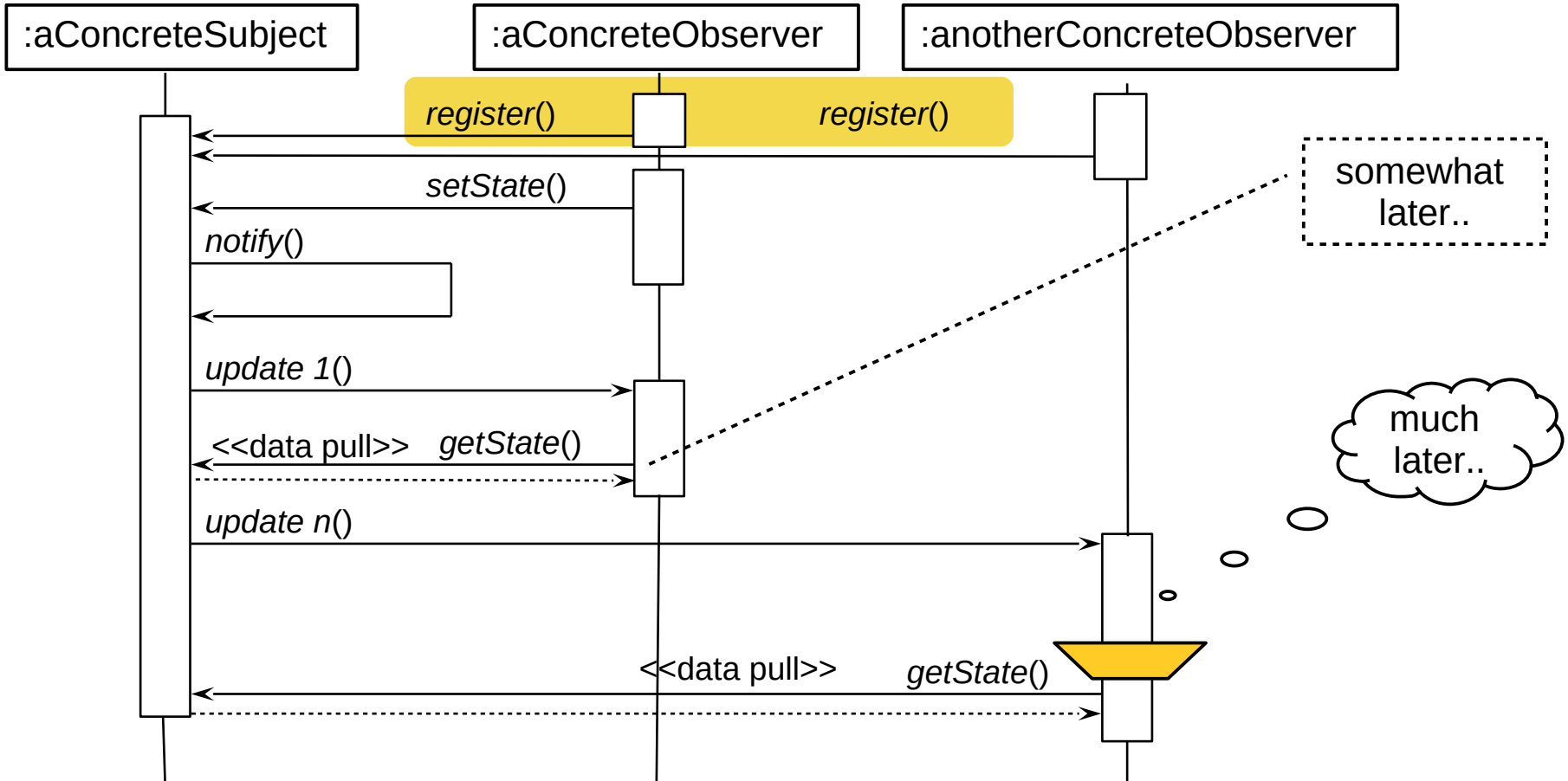
Structure Observer (pull-Variant)

- Aka Publisher/Subscriber
- Subject does not care nor know, which observers are involved: subject independent of observer



Sequence Diagram pull-Observer

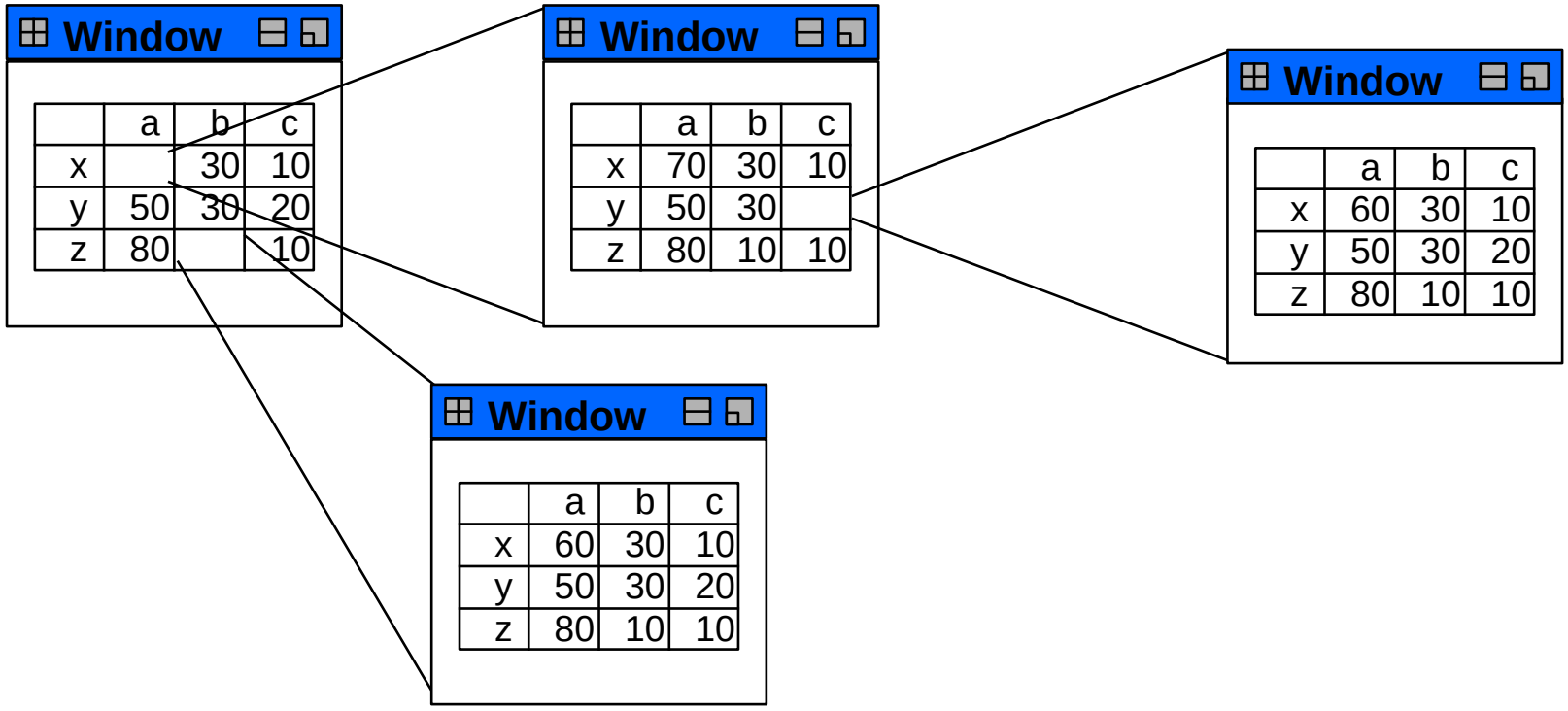
- Observer.update() does not transfer data, only announces an event
  - Anonymous communication possible
- Observer *pulls* data out itself
  - In the context of MVC, Controller or View pull data out of the application model themselves



Pattern 2: Composite (Rpt.)

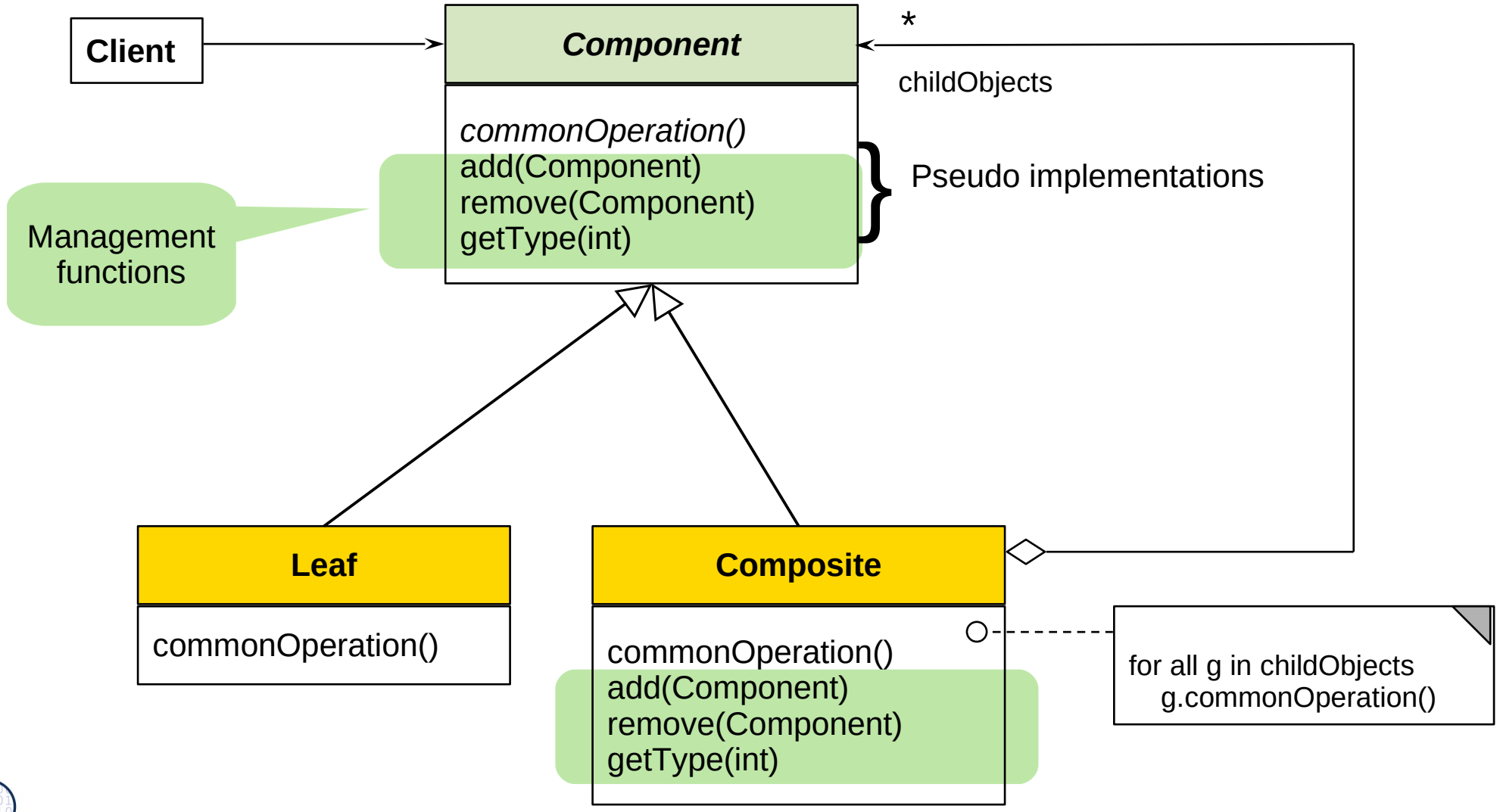
Views may be *nested* (*Composite*)

- Composite represents trees
- For a client class, Compositum unifies the access to root, inner nodes, and leaves
- In MVC, views can be organized as Composite



Structure Composite (Rpt.)

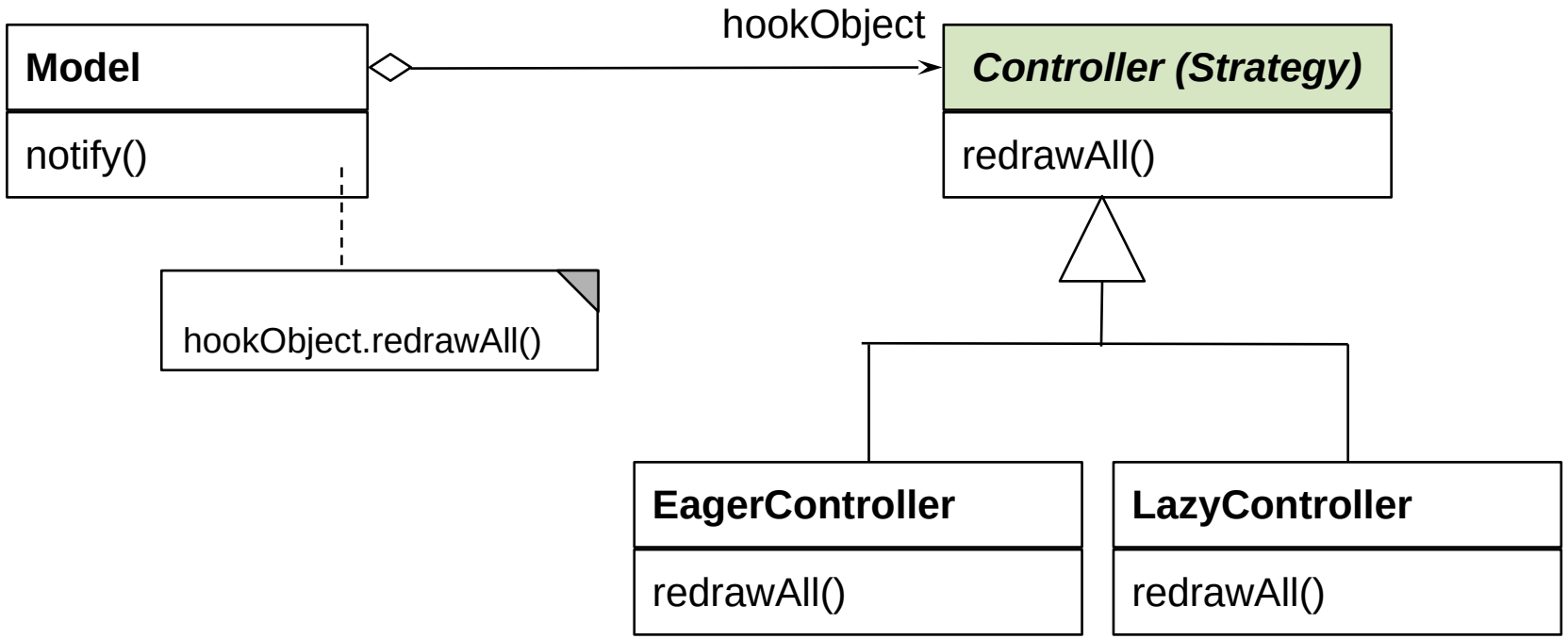
- Composite has an recursive n-aggregation to the superclass



Pattern 3: Strategy

The relation between *application model* and *controller* is a *Strategy* pattern.

- There may be different control strategies
  - Lazy or eager update of views
  - Menu or keyboard input
- A view may select subclasses of *Controller*, even dynamically; no other class changes
- Strategy is similar to Command pattern



Purposes of Design Patterns

- Design patterns improve **communication** in teams
  - Between clients and programmers
  - Between designers, implementers and testers
  - For designers, to understand good design concepts
- Design patterns create a **glossary** for software engineering (an "ontology of software design")
  - A "software engineer" without the knowledge of patterns is a programmer
- Design patterns **document** abstract design concepts
  - Patterns are "mini-frameworks"
  - Documentation: in particular frameworks are documented by design patterns
  - Prevent re-invention of well-known solutions
  - Design patterns capture information in reverse engineering
  - Improve code structure and hence, code quality

What Have We Learned?

- Design patterns grasp good, well-known solutions for standard problems
  - good for communication
- Design patterns serve for *beautiful* software