UML applied

Yvonne Dittrich

IT-University in Copenhagen
Design and Use of IT

For the exercises today:

- select a part of your structure diagram you would like to discuss. (e.g. books and copies, reservation, lending of books)
- prepare to present the issue you want to discuss and the part of the diagram that relates to it.
- do the same for a part/one of your behavioral diagrams.
- each group has a half hour for presentation and discussion

Roadmap

- what is the difference between Design, analysis and design
- what to use the diagrams for
  - how to go about for the analysis
  - how to go from analysis to design
  - **if we have time** I give an example how to use a metaphor to relate analysis, architectural design, and the use of patterns. (The last 23 slides)

What is Design

'By design we mean a specific type of insight-building process that is geared to producing feasible and desirable results within a particular domain.'

- concerns we would like to fulfill
- limited resources
- different implementation options

© Christiane Floyd

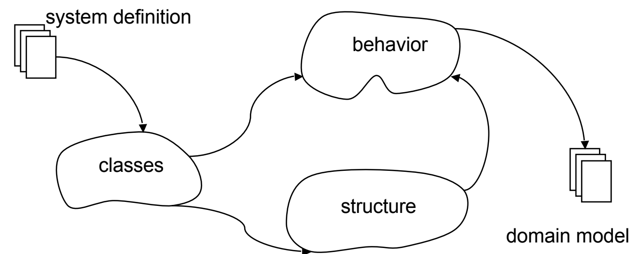
analysis versus design

- analyzing the 'what' - designing the 'how'
- 'how do things look today?' vs. 'how do we want things work tomorrow?'

there is no sharp border between a & d
  - you analyze with a purpose in mind
  - the language of the analysis mirrors the purpose
  - both define what is interesting to model
  - when designing and implementing you will find aspects you missed while analyzing

- questions to ask in the analysis:
  - does the model mirror the domain?
  - do the results of the application domain analysis mirror the users needs?
- questions to ask in design:
  - how does that work when it runs on a computer?
  - do the documents provide a good enough input for the implementation phase?

activities when modeling the problem domain



analysis of the problem domain

- finding classes and events
- developing and evaluating a class structure
- analysing behaviour of objects (of a class)

What do we use here?

evaluating the candidate classes and events systematically

- is the class or the event within the system definition?
- is it relevant for the problem domain model?

classes

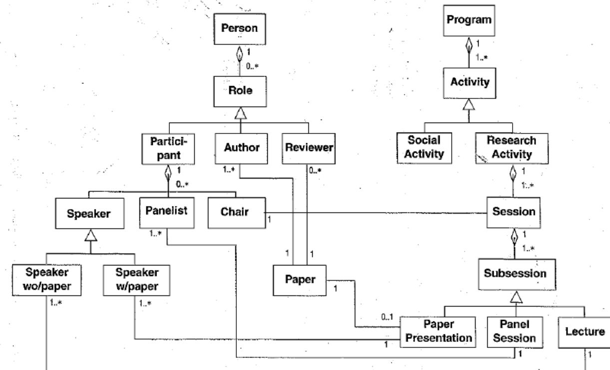
- can you identify objects from the class
- does the class contain unique information
- does the class contain more than one object
- has a class a suitable number of events

events

- is the event instantaneous
- is the event atomic
- can you decide whether the event took place and when it took place

evaluate your structures

- your structures must be used correctly
- your structures must be conceptually true
- your model must be simple

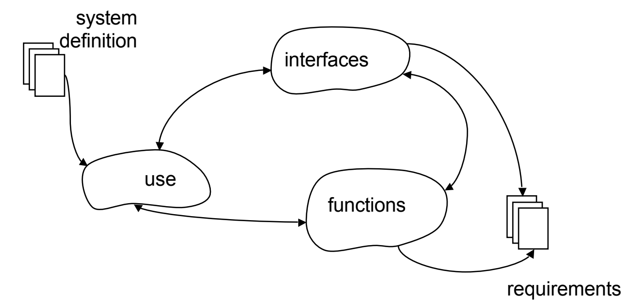


behavior

- event sequence:
  a sequence of events a specific object is involved in
- behavioral pattern:
  a description of possible event sequences for all objects of a specific class
- attribute:
  a descriptive attribute of a class or an event

Event	*Class* person	participant	author	paper	reviewer
participant registered	+	+			
paper registered	+		+	+	
paper submitted			+	+	
paper evaluated				*	+
review assigned	+			*	+
decision notified			+	+	

activities when modeling the application domain



evaluate

- each use case should be simple and constitute a coherent whole
- descriptions of actors and use cases should provide understanding and overview
- use cases should be described in enough detail to enable identification of functions and interface elements.

evaluate use cases in collaboration with the users

function:
a facility for making a model useful for actors

- for **update functions**: relate events and use cases
  - how is an event observed/registered? in which use case does it happen?
  - how can the use case be supported by update functions?
  - which objects, attributes and structures are effected?
- information needs that yield **reading functions**
  - what do the actors need to know about the model's state?
  - what part of the model will the actor need to know about?

functions II

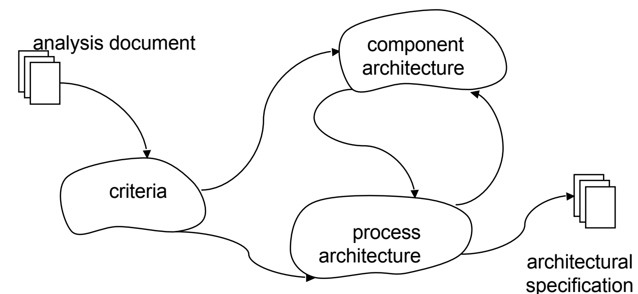
- **computations**
  - what computations (not necessarily based on the model) do the actor need to have carried out?
  - does the data the computation is based on come from the user, the model or both?
  - which computations form complete wholes in the use cases?
- **critical states**
  - what are critical states for the model?
  - what is the significance of these states, what are the consequences?
  - how does the function register that the model has entered a critical state?
  - what signals does each critical state give rise to? how reliable and strong do the signals have to be?

interfaces

- how should classes and objects be presented, updated, read
- how to support use cases and groups of use cases
- use sequence diagrams for the use cases
- list the resulting interface components (e.g. windows)
- 'describe' the different interface components their look, their functionality (e.g. commented mock-ups)

experiment and iterate

activities in the architectural design



Concepts

criteria:
preferred properties of an architecture

component architecture:
a system structure composed of interconnected components

process architecture:
a system-execution structure composed of interdependent processes

classical criteria

portable

usable

efficient

testable

secure

reusable

maintainable

inter-operation

reliable

correct

flexible

comprehensible

specific conditions

- technical
  - hardware, implementation technology, other systems
  - reuse of patterns and existing components
  - use of purchased standard components
- organisational
  - contractual arrangements
  - plans and continued development
  - division of work between developers
- human
  - design competences
  - experience with similar systems
  - experience with the technical platform

a good design …

… has no major weaknesses.

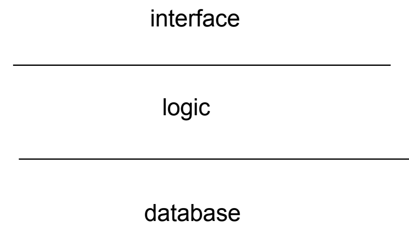
… represents a trade-off between different criteria.

… is (at least) usable, flexible and understandable.

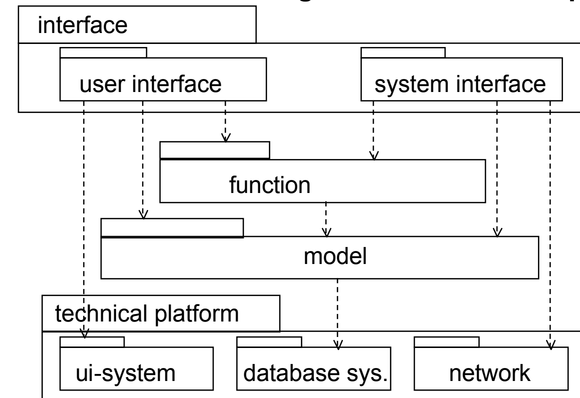
what is a component architecture

- you have a set of components that interact in a certain way
  - filters and pipes
  - clients and servers
- you can define different types of components and different ways they interact
- architectural styles that often relate to specific application domains provide concepts to think with.

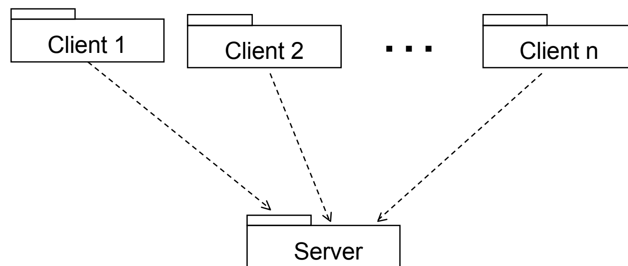
layers or (abstract machines)



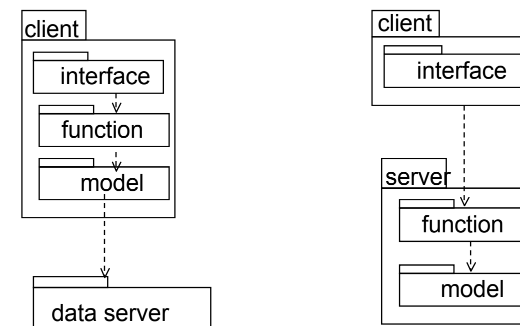
generic architecture pattern



client server architecture pattern



thick clients - thin clients



component design

- determining an implementation of requirements within an architectural framework
- designing components
- designing the connection between components

© Dittrich September 05

designing the model component

- private and unique events
  -> design fitting attributes
- private and repeating events
  -> consider designing a class that is holding the data of this event that is aggregated to the object of the class under discussion

-->

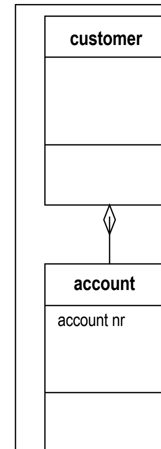
© Dittrich September 05

… continuation …

- common unique events
  -> design fitting attributes for the class the data fits best
- common repeating events for aggregations
  -> design a new class for the data of that event and aggregate it to the object/class that yields the simplest class diagram
- common repeating events for associations
  -> design a new class for the event that is aggregated to both partners of the association

© Dittrich September 05

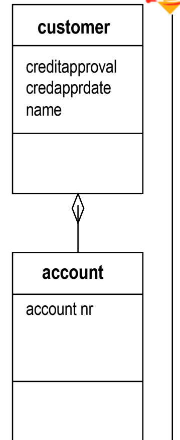
an example



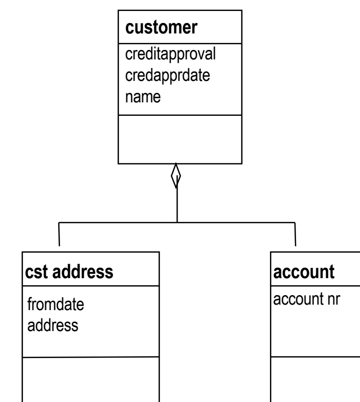
	classes	
events	*customer*	*account*
credit approval	+	
change address	*	+
account opened	*	+
account closed	*	*
deposit	*	*
withdraw	*	*

© Dittrich September 05

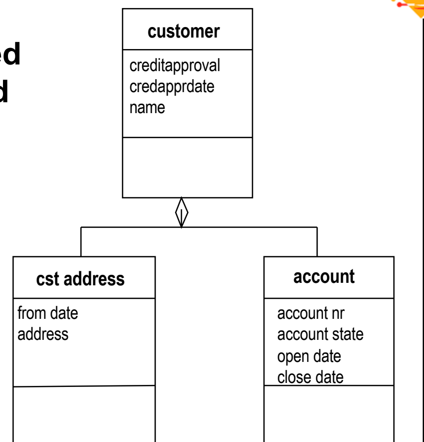
event:
credit approval



event:
change address

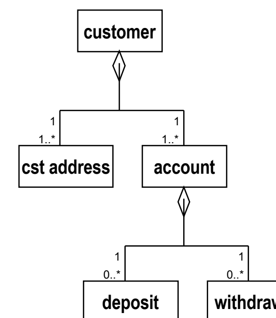


events:
account opened
account closed



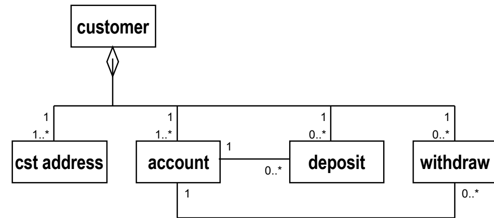
events:
deposit
withdraw

one solution



events:
deposit
withdraw

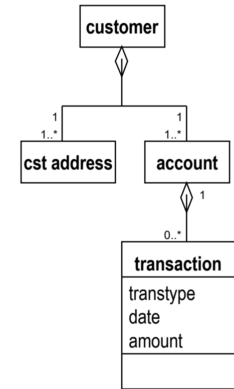
another solution



© Dittrich September 05

events:
deposit
withdraw

a third solution



© Dittrich September 05

	Class				
Event	person	participant	author	paper	reviewer
participant registered	+	+			
paper registered	+		+	+	
paper submitted			+	+	
paper evaluated				*	+
review assigned	+			*	+
decision notified			+	+	

© Dittrich September 05

design of function components

now it is time to do the sequence diagrams …

© Dittrich September 05

design of function components

now it is time to do the sequence diagrams …

the Tools & Materials Approach

© Züllighoven et al, http://www.jwam.de

combines

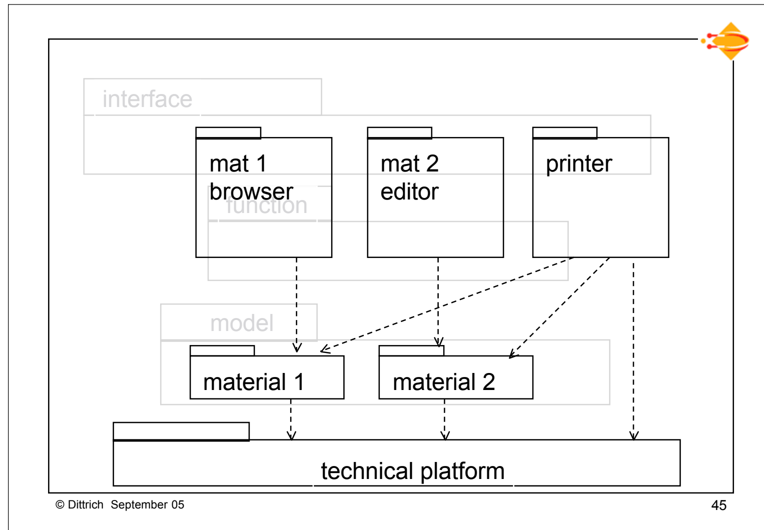
- use oriented development
- iterative process model
- a set of related analysis and design metaphors that anchor the design to the work practice of the future users
- an architectural style
- design patterns

the metaphor

- materials ‘lie around’ and are used for specific tasks: paper, folder, schedule, mails, kayaks, courses
- materials are only accessible through the tools manipulating them
- materials are often organised in specific containers
- tools are used on materials to perform tasks: pens, punch, editor, browser, viewer
- tools are passive: the hammer does not jump on you
- tools might be specialised, but can also be generic
- one might need automatons as well: printer, compiler

How does that relate to the pattern architecture

- materials can be seen as partitions of the model component, that is meaningful from a use pov
- tools are partitions in the view and function component. A tool component consists of a functional part and an interaction part and so connects a partition I the view and a partition in the function part
- a tool is a meaningfully related set of functions and related interface elements
- the overall program is designed as an environment where tools, materials and automatons ‘lie around’



what do we gain?

- tools & materials allow for a modeling that is close to the use context
- it introduces a way of partitioning the model component and the interface/function component
- the partitions on the both layers are independent of each other.
- and architecture that provides a frame for highly incremental development
- as an architectural style, one can use design and programming patterns for the connections of the different elements.

patterns

'A pattern describes a problem which occurs over and over again in our environment, and then descries the core of the solution to that problem, in such a way that you can use this solution a million times over without ever doing the same thing twice.'

© Christopher Alexander

a pattern has 4 elements

- **its name:** describing the problem and the solution in 1-2 words
- **the problem:** describing when to apply the pattern
- **the solution:** the elements of the design, their relationships, responsibilities, and collaborations
- **the consequences:** the result and trade-offs of applying the pattern

© E. Gamma et al.

patterns in the Tools&Materials design

- I only can present part of the design patterns
- the ones I selected should show
  - why to use patterns
  - how to reason about patterns
- we take the ones
  - coupling tools and materials
  - division of tools in interaction and functional parts and how they interact
  - composite tools
  - the event handler of the environment

Coupling tools and materials - the problem

- tools and materials have to fit together
- tools might work on different materials - e.g. a lister would work on different containers
- a material might be accessed and updated by different tools

coupling tools and materials - the first solution

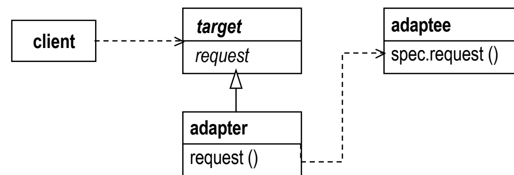
- design an abstract class, an aspect, that represents the interface a tool needs and a material has to implement.
- the materials inherit the aspect class and implement the abstract methods
- the tool only has to know about the aspect, not about the material

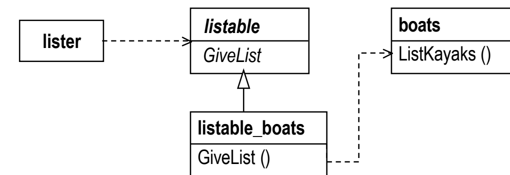
coupling of tools and materials - consequences of solution 1

+ the compiler can check that the interface is correctly implemented
+ information hiding: the tool only knows aout the aspect
+ the aspect classes can be used to specify the interface for parallel development
- uncommon usage of inheritance
- that might make the design difficult to understand
- what if you do not have multiple inheritance?

coupling of tools and materials - solution 2

- implement the adapter pattern (Gamma et al.)



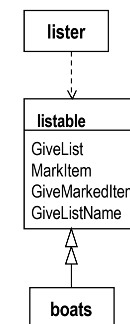


coupling of tools and materials - consequences of solution 2

+ aspects are clearly visible in the design
+ you can use that pattern also when the interfaces are not exactly matching
+ the designer can distinguish between professionally motivated operations on the material and the interface for specific tools
- an additional class for each tool-material pair
- the identity problem
- who is creating the adapter object and when

coupling of tools and materials - solution 3

if you have the possibility to define an interface in a programming language this can be used to implement aspects (interfaces in Java and C#)



coupling of tools and materials
- consequences of solution 3

+ aspects are clearly visible in the design
+ the compiler can check that the interface is correctly implemented
+ information hiding: the tool only knows about the aspect
+ the aspect interfaces can be used to specify the interface for parallel development
- you need a special language construct

© Dittrich September 05

interaction between interaction part and function part of a tool
- the problem

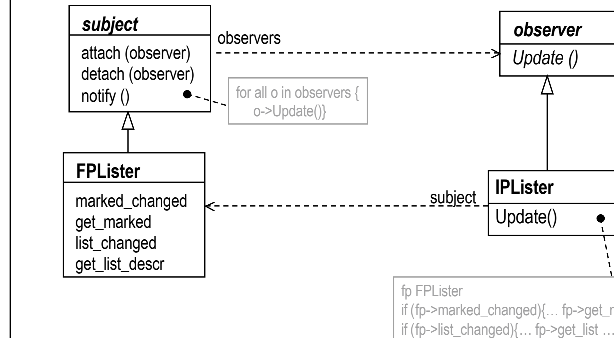
- you want to split the interaction part and the function part of a tool, to be able to implement and change them independently
- the interaction part knows the function part but should not make assumptions about the result of operations
- the function part should definitely not make assumptions about what is displayed or how

© Dittrich September 05

interaction between interaction part and function part of a tool
- solution 1

implement he observer pattern between interaction and function part of a tool

© Dittrich September 05



© Dittrich September 05

interaction between interaction part and function part of a tool - consequences of solution 1

+ the interaction part knows the function part, but the function part does not know about the interaction part
+ the interaction part does not make any assumptions about the result of the method of a function part
- if the request for information changes the state of a material you might get into a loop.
  you can handle that by implementing reading methods without side effects
- for complex tools the observer pattern might lead to runtime difficulties.
  you can expand the update-method with a parameter, indicating the type of change that occurred

interaction between interaction part and function part of a tool - solution 2

Implement an event handler:

- design a class ‘event’
- the function part creates a specific event-object for each relevant change
- the interaction part registers at the event-object that are relevant for it and hands over a method to be called for updating
- the event object keeps a list of all objects registered
- the function part announces a change to the respective event-object
- the event-object calls the methods that they handed over.

interaction between interaction part and function part of a tool - consequences of solution 2

+ differentiated notification of the objects that have to be informed about an event
+ the possible changes a function part can announce are visible in the event-object it creates
- there is the danger that the function parts get designed for specific interaction parts. this is actually the problem, we wanted to avoid…

what more about tools and materials

- combination of tools
  - how to manage the communication between different tools
- different tools might change the same material
- administration of materials, especially when accessing a database server that is used by more than one of such tools & materials clients
- containers is an own chapter
- framework support

- complex dependencies between materials? where to handle them?
- what when the world outside the workshop gets involved?
- sometimes media metaphors might be better suited…

- C. Floyd Software development as reality construction. In: Floyd et al. *Software development and reality construction.* Springer 1992.
- L. Mathiassen et al. O*bject Oriented Analysis & Design*. Marko Publishing ApS, Aalborg, Denmark 2000.
- D. Garlan, M. Shaw 'An Introduction to Software Architecture' in: V. Ambriola, D. Tortoga: *Advances in Software Engineering and Knowledge Engineering, Volume1*, New Jersey, 1993.
- H. Züllighonen et al. *Object-Oriented Construction Handbook*. D-Punkt Verlag, October 2004
- D. Riehle, H. Züllighoven 'A Pattern Language for Tool Construction and Integration Based on the Tools&Materials Metaphor.' *Proceedings of the PLOP '94, Monticello, Illinois, August 4-6, 1994*.
- C. Alexander et al. *A Pattern Language*. New York 1977.
- E. Gamma et al. *Design Patterns*. Addison-Wesley 1995.