# Architectures of Distributed Systems 2011/2012

Introduction

Johan Lukkien



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

# Course responsible

- Section: Security and Embedded Networked Systems
  - Johan Lukkien
  - Rudolf Mak
  - Bojan Orlic
  - all three are members of SAN: System Architecture and Networking

- Similar as previous year:
  - no obligatory text, but one strongly recommended
  - emphasis on the systems

- Plan
  - 12 lectures + ~4 invited speakers

# Examination

- Exam in block 1, retry in block 2
- Practical, with two students
  - 4-5 exercises while going
    - must hand in, in time
- Research paper, with two students, in block 2
- Each counts for 1/3
- Minimal score per item is 4

# Goals of this lecture

- Students understand the notion of an architecture as a high-level description of a system from various viewpoints.
- Students understand the formalization of this for the domain of software intensive systems and the motivation behind this.
- Students know some common views and viewpoints.

# Some questions

- What *is*
  - (a) software architecture?
  - (a) system architecture?

  Why do we need them?

- What do I *make* when I say I make
  - a design?
  - an architecture?

  What is the *tangible* result of my work?

- What is the *quality* of an architecture?
  - discriminate between good and bad ones
  - is the tangible outcome according to accepted rules?

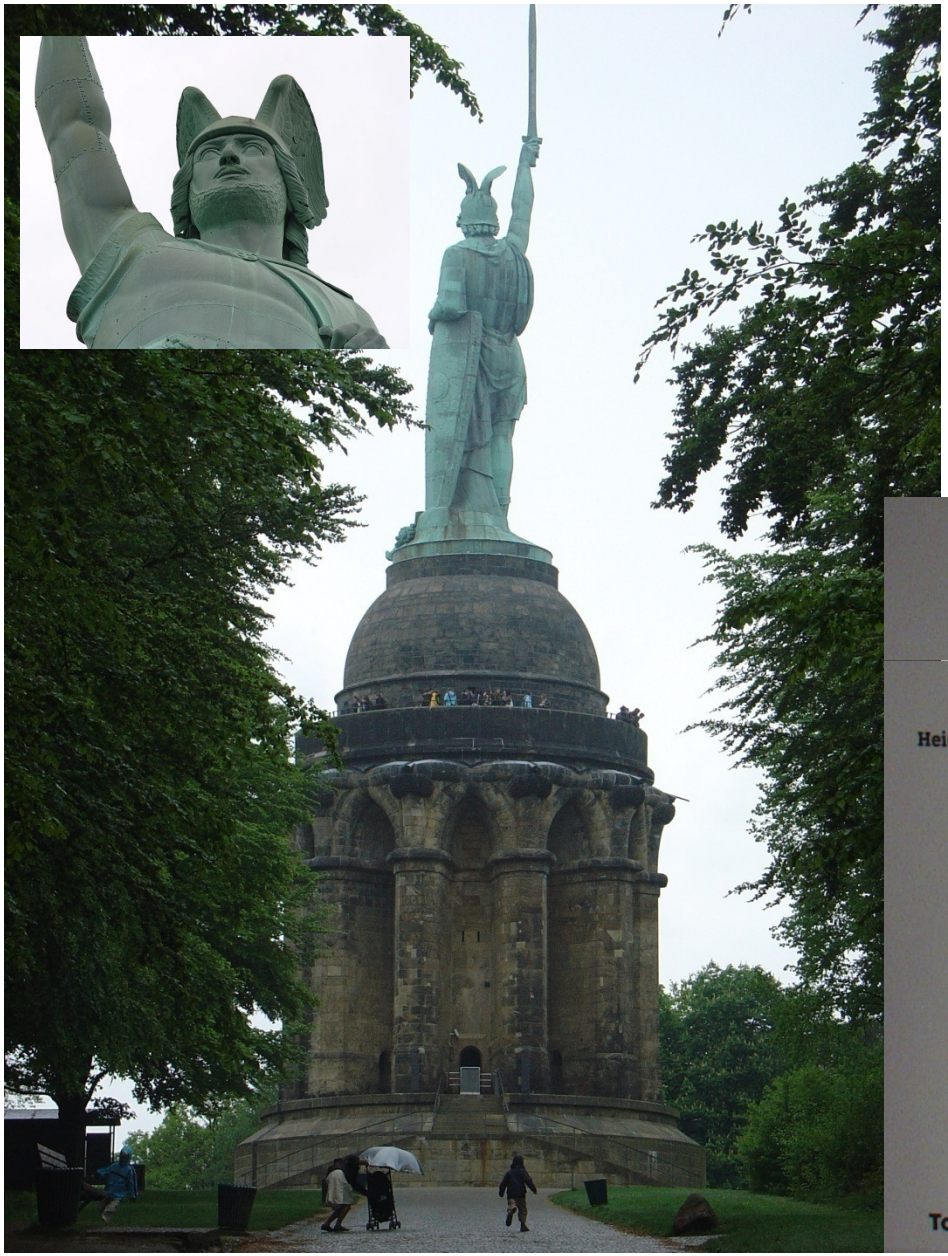
- Can I see that a system has been built according to a certain architecture?
  - does the architecture serve as a form of documentation or prescription for realization?

# Architecture: why bother?

- Do I need an architecture when I build
  - a fence?
  - a dog shed?
  - a program to compute the first 10000 primes?

- However, it is different for tasks that require
  - a global understanding of a complicated system
  - analysis of design alternatives, and of general system properties
    - *before such system is built*
    - *or without having access to it*
  - communication between team members
  - communication (documentation) for handing off parts of the work
  - documentation for later reference
  - decomposition and synthesis of parts

- Larger technical systems are simply too complex and long-lasting for one person



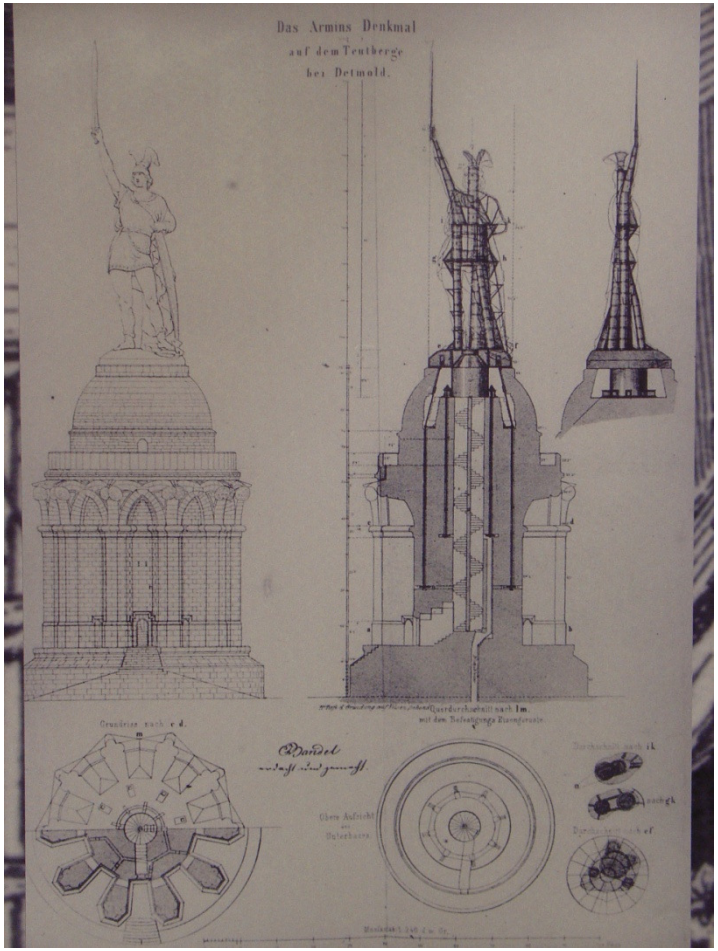
# Hermannsdenkmal

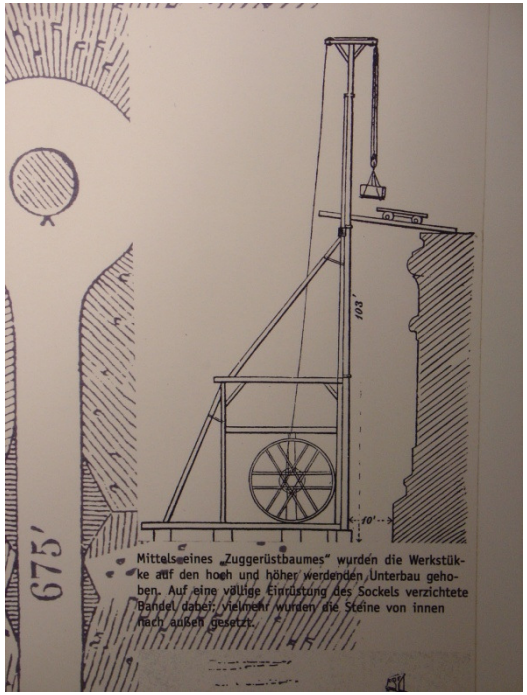
- Just south of Detmold, Lippe, Germany
- 1838-1875, statue of Arminius (Hermann) who defeated the Romans in A.D. 9 in the Teutoburger Wald

| | |
|---|---|
| | **The base** |
| Material | Square shaped sandstone blocks, cut on the hill |
| Quantity | 57 stone layers, 9 inches to 1 meter high |
| Height of the basement | 26.89 meters |
| Diameter | Pedestal of the base: 19 meters, Base: 21 meters |
| | Plateau for the figure: 7 meters |
| Number of steps | 13 in the pedestal, 69 in the tower, 28 in the dome |
| Visitors' gallery | 1.20 meters wide, space for 300 visitors |
| Height of the dome, incl. stone ring | 5.50 meters |
| | **The supporting frame scaffolding** |
| Material | sheet iron, thickness: 1.5 inches |
| Cylinder in the dome | height: 4 meters, diameter: 3 meters |
| Main cylinder | height: 7.50 meters, diameter: 1.50 meters, weight: 3,375 kg |
| Head cylinder | height: 4.50 meters, diameter: 1 meter, weight: 750 kg |
| Further cylinders | right leg: 1,650 kg, left leg: 1,200 kg |
| | Stomach cylinders: 1,000 kg, Ribs in the shield: 1,800 kg |
| | **The statue** |
| Material | copper sheet |
| Sword | height: 7 meters, weight: 550 kg |
| Shield | height: 10 meters, weight: 1,150 kg |
| Complete statue | height: 26.57 meters, weight: 42,800 kg |
| **Total height of the monument** | **53.46 m** |

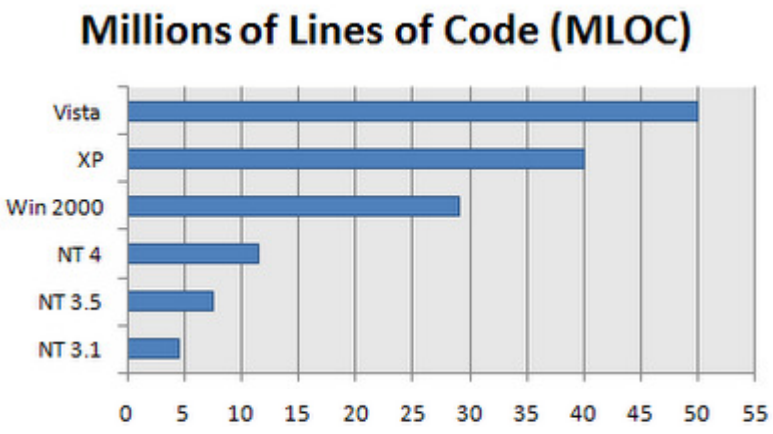
# Architectural descriptions

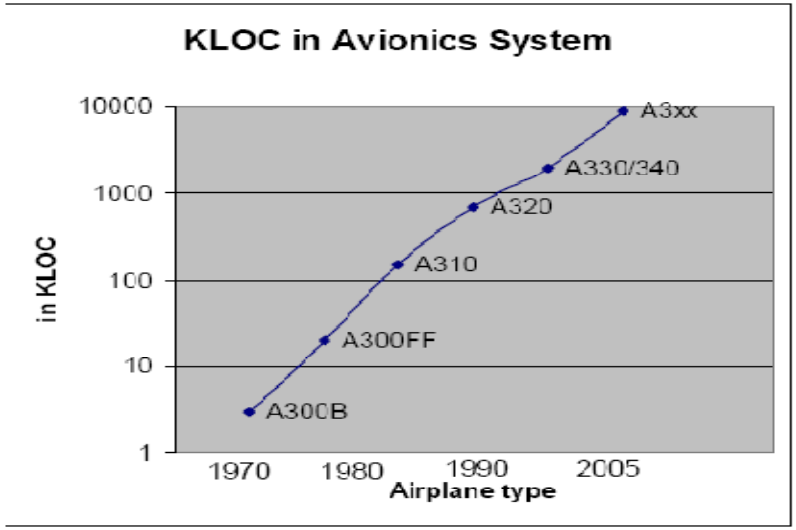
- Communication with builders
- Compute structure strength, relate to boundary conditions (e.g. wind)
- Design, ‘Decomposition’
- Structure
- Methods for putting together the parts



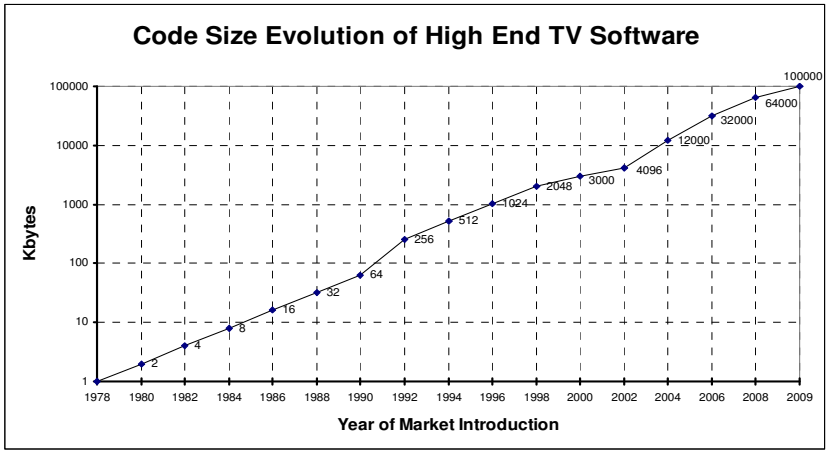



# Complexity of software, in numbers




- 1 KLOC = 1000 lines of code
- #faults is linearly related to LOC
  - ~3 / KLOC



# Modern cars

- What's Got 10M Lines of Code & An IP Address?
  - The Volt (http://gigaom.com/cleantech/whats-got-10m-lines-of-code-an-ip-address-the-volt)


- IEEE Magazine, Januari 2009:
  - 30-100 ECUs (microprocessors) in modern cars
  - 10M-100M lines of code!

| | | |
|---|---|---|
| Air-bag system | Antilock brakes | Automatic transmission |
| Alarm system | Climate control | Collision-avoidance system |
| Cruise control | Communication system | Dashboard instrumentation |
| Electronic stability control | Engine ignition | Engine control |
| Electronic-seat control | Entertainment system | Navigation system |
| Power steering | Tire-pressure monitoring | Windshield-wiper control |

- Hence
  - .... 30K – 300K errors
  - integration problem: need clear structures, interfaces, standards

# Some observations on four aspects

- Understanding, documentation:
  - structure and meaning of a system can be understood without all details
  - a person completely new to a system must be able to learn how it works *after it has been created.*
  - feedback to users and customers must be given timely in order to see whether ***the right product is made***
  - a single person has only a partial understanding of a large system

- Analysis
  - need to examine design alternatives without detailed reference to implementation
  - system properties (e.g. performance) must be inferred at a high abstraction level
  - it must be possible to answer questions about a system

- Communication
  - larger software and systems are made in teams; people in the teams must communicate
  - teams work at different levels of detail
  - people in the teams have different background, knowledge and skills

- Construction
  - the development process cannot be 'single shot'
  - development evolves in stages, incremental design with different teams
  - systems are put together from existing, independent components

# Design *problem* – a perspective

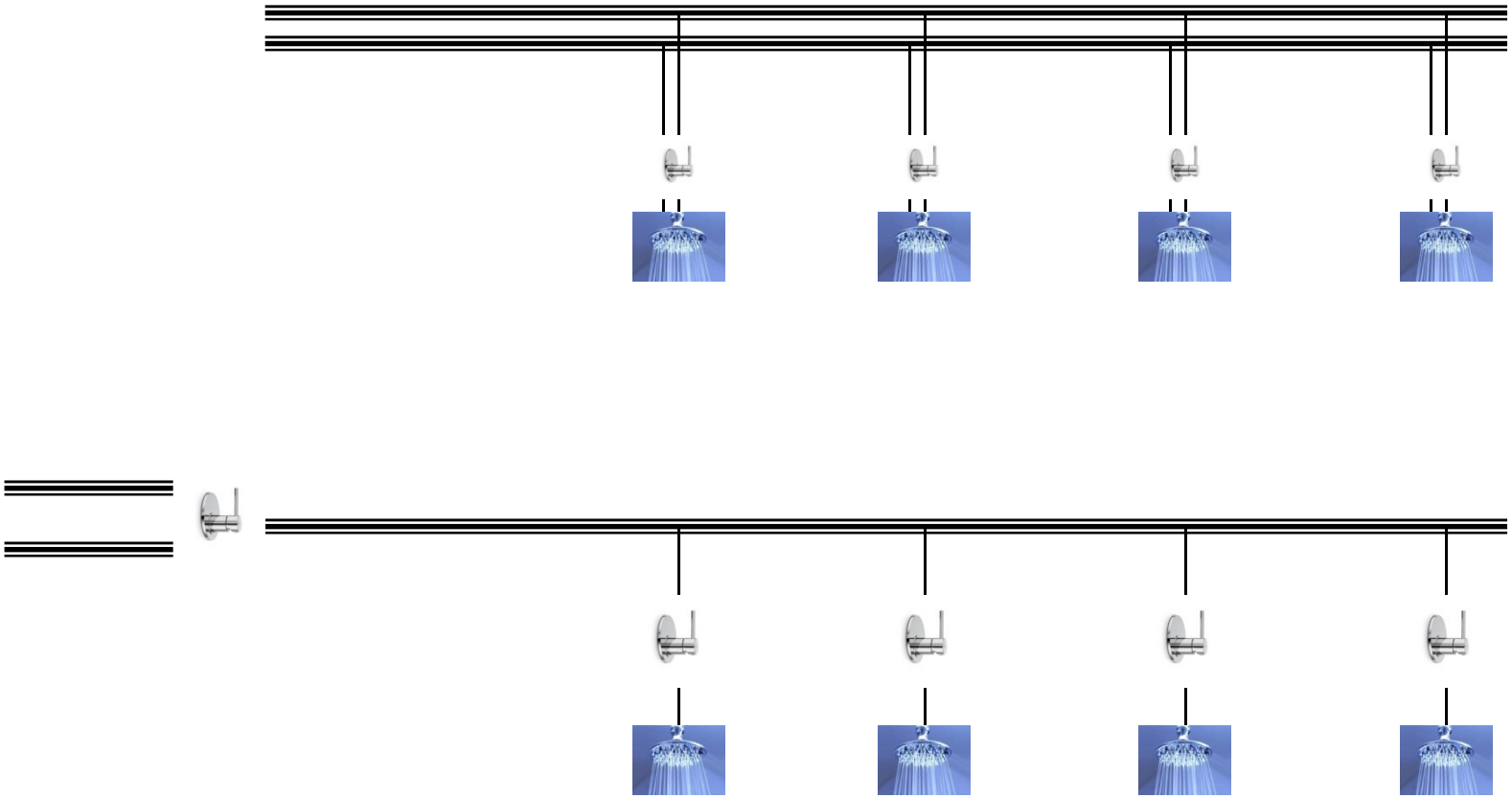
- Realize a *product design* from *building blocks* and *connectors (combinators)* satisfying *functional requirements...*
  - *functional requirements*: captured via *use cases*
  - *use cases:* interactions with the product
  - *product*: building blocks + connectors
  - *design*: drawings, blue prints

- ...subject to *boundary conditions* and *quality constraints...*
  - *rules (constraints)* for building blocks and connectors
  - *extra-functional* properties
    - '...ilities' [security, reliability, performance, .....]
  - *technical environment*
    - limitations: distribution, platform choices
    - tools, methods, languages

- ...within an *environment* or *context.*
  - assumptions on environment behavior
  - *stakeholders,* the involved parties and their roles
    - customer, manufacturer, maintenance team, design team (and their capabilities), user, ...

# Example: shower control system

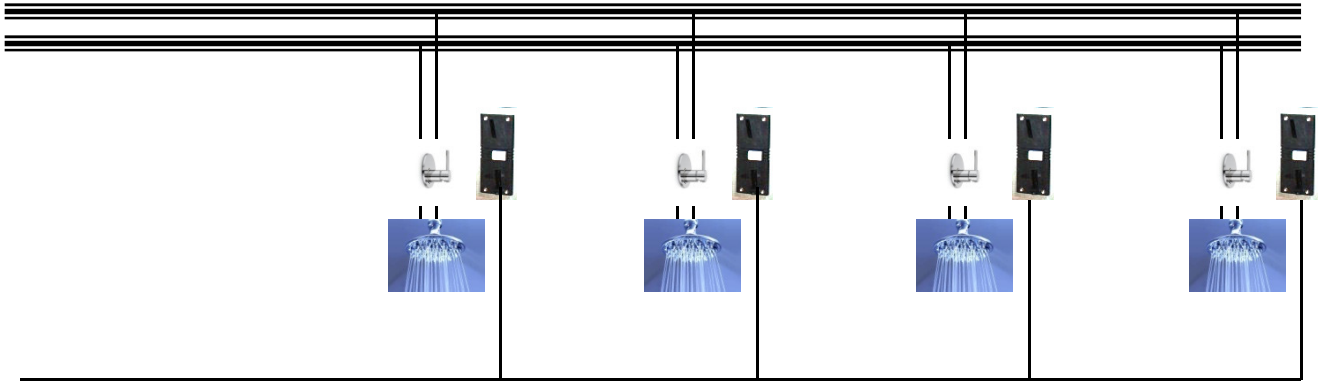
# Two alternatives





Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

# It's not for free

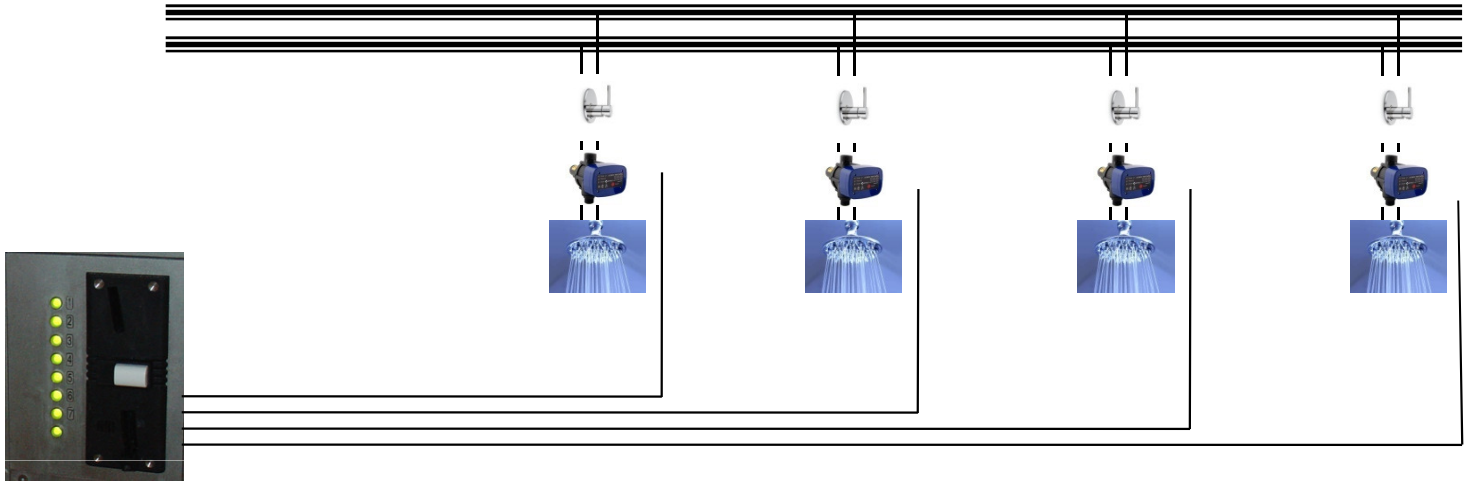


- Extra infastructure for locally controlling water outlet and payment
  - might be expensive
- Alternatives?


Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

# A centralized solution

# Centralized solution



- Central money collecting
- Some extra infrastructure, now just for switching on/off
  - might be cheaper
- Makes taking a prolonged shower difficult

- The real (built) system:
  - schedules shower assignment
    - makes a prolonged shower embarassing
  - uses a combined maximum on time and water used
    - changes this architecture
    - probably more expensive
  - is a COTS system (Commercial Off-The-Shelf)
    - cheaper, known by (installation) companies

# Design problem: design shower system

- Functional requirements:
  - shower functionality – use cases
  - cleaning functionality – use cases
  - ....
- *Building blocks*: money insert/controlling device, shower heads, switches
- *Connectors*: pipes and wires

- Constraints:
  - building block choices (perhaps), brands, regulations, ...
- Extra-functional:
  - performance (#liters/second, #liters/user), payment, user control, ....
- Technical environment:
  - the given building, the given infrastructure of pipes, heater perhaps, ...

- Environment assumptions
  - showering in the morning and evening
  - people tending to shower too long
- Stakeholders:
  - users, cleaner, repairmen, installer, owner, .....
  - **note**: *these stakeholders all have system perspectives and use cases*

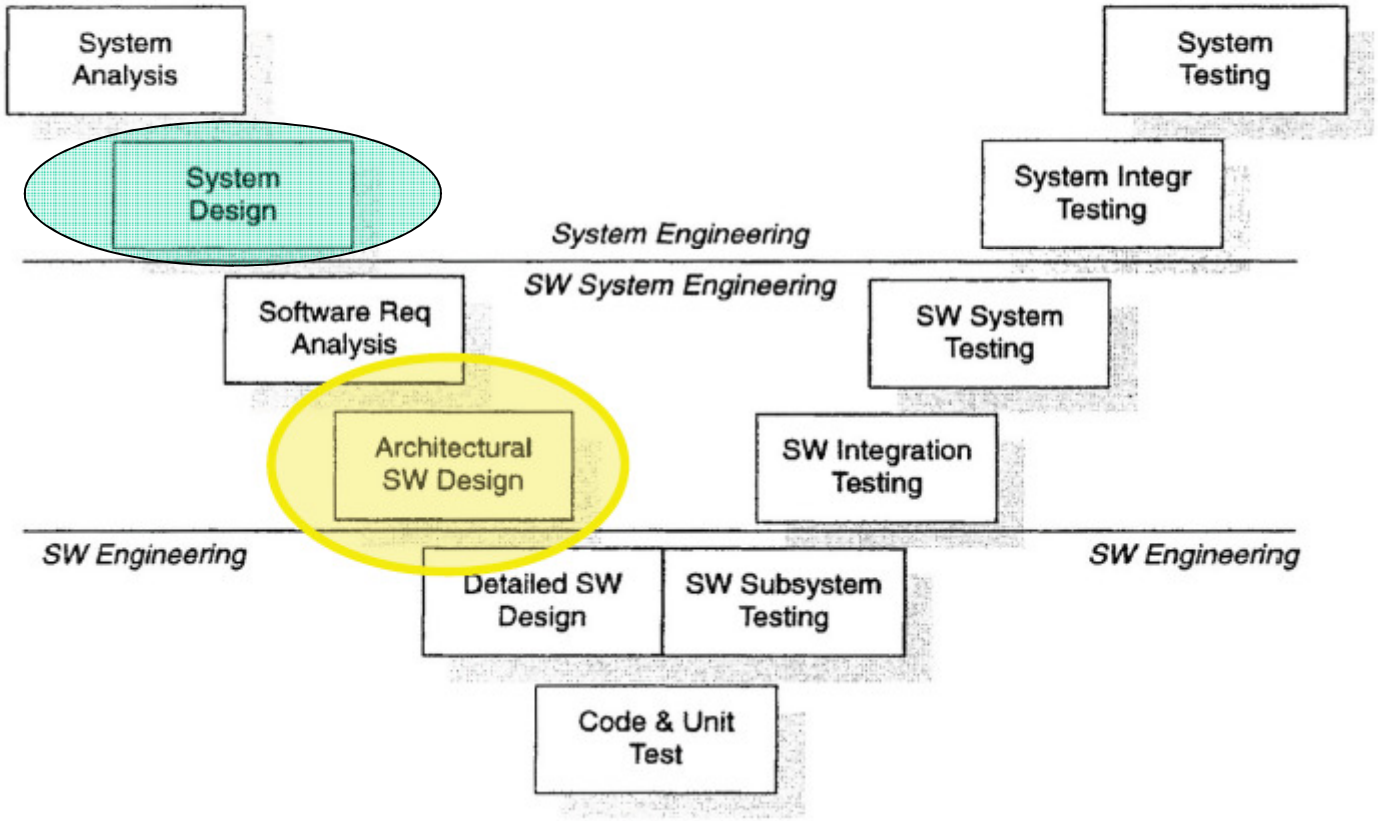
# Design *process* – a perspective

- Translate a *design problem* into (a *model/blueprint* of) a solution


Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

# Architecture: place in system development cycle

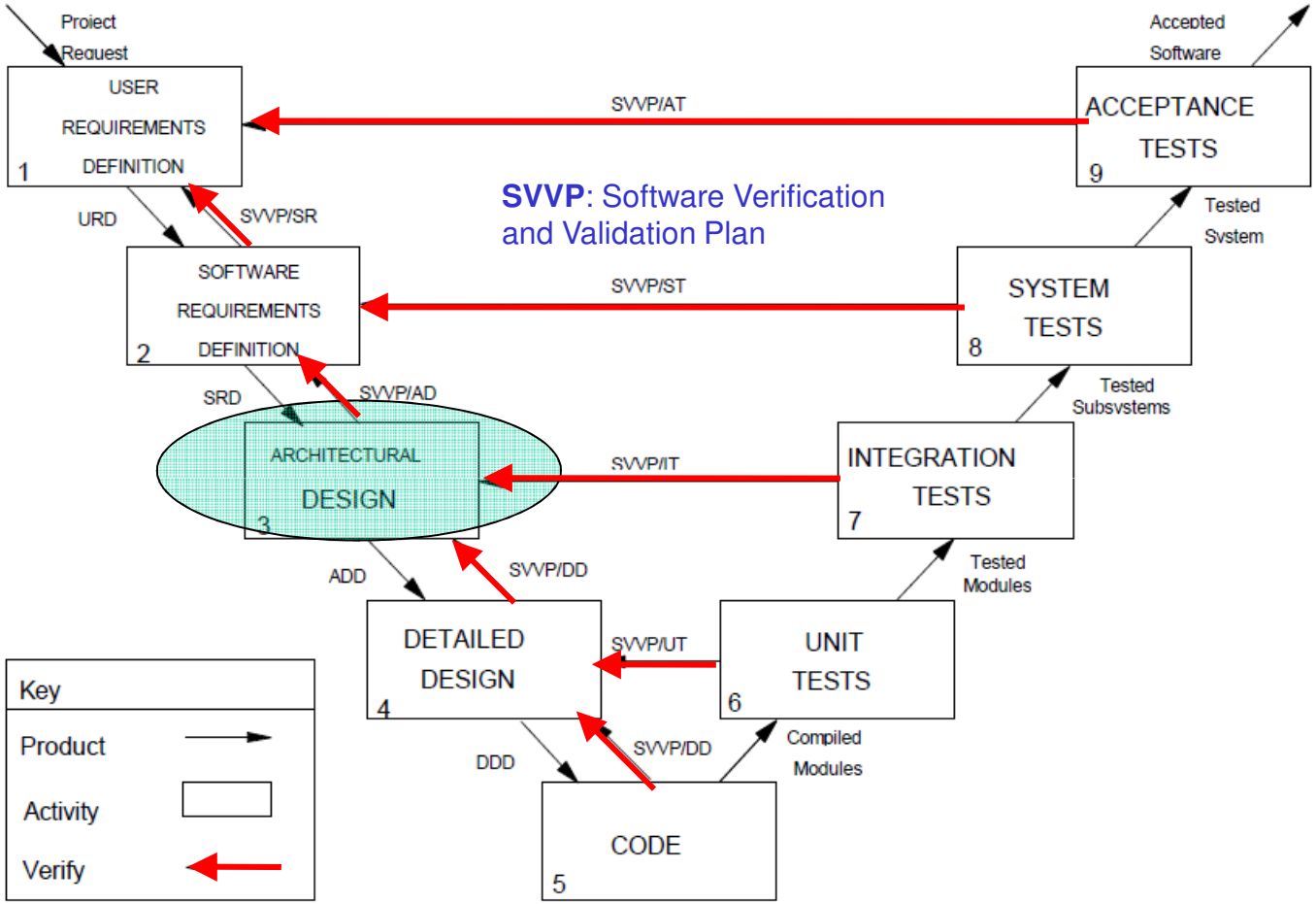
From: M.J. Christensen, R.H. Thayer. *The Project Manager's Guide to Software Engineering's Best Practices.* Wiley, 2002






Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

# Software Architecture: place in software development cycle




Esa standard: life cycle verification approach

# Design *process* – a perspective

- Translate a *design problem* into (a *model/blueprint* of) a solution

- The process is *iterative* (perhaps *recursive*)
  - level *n+1* solves a set of smaller design problems than level *n*
    - smaller problems are subject to the same approach
  - models are increasingly more detailed until the problems are no longer of a structural nature
    - hence, 'detailed design' is of a different nature than architecture design
  - during this process *design decisions* are taken
    - choosing among options
    - challenge: *document* these decisions and their rationales (whose concern??)

- Building blocks per level are different
  - needs a good understanding per level
    - and the mutual impact of lower levels on higher levels
  - may lead to *specialization* in the development process
    - ....a building architect usually does not design the plumbing details
    - ....a software architect is not concerned with implementation of simple data structures
  - note that building block and connector properties determine system functionality

# Design *process* – four elements

- (Domain) analysis
  - increase knowledge, make models
    - use cases, based on stakeholder viewpoints
  - feedback to stakeholders: validation of requirements ("Do we solve the right problem?")

- Apply strategies
  - hierarchical decomposition:
    - top-down (factorization): *specify* advanced building blocks (decompose functional specification, and derive extra-functional properties for the parts)
    - bottom up: *design* advanced building blocks
  - apply patterns, styles
    - pattern, style: *coherent set of design decisions*
  - generate alternatives

- Synthesis
  - evaluate and choose alternatives, combine partial solutions

- Verification
  - is the system according to specification? ("Did we solve the problem right?")

# Architecture

- The collection of models/blueprints comprises the architecture description
  - the models are organized into *views*
  - the architecture description can be examined at varying levels of abstraction and different *viewpoints*
  - the first (top-most) set of blueprints is special, and is also referred to as 'the architecture' (or better: 'the architecture description'):
    - it needs to make the transition to the real world (technical and operational environment, use cases)
    - it presents the system at a high level (the highest) of abstraction
  - as such it is used for *understanding*, *analysis*, *communication*, *construction*, *documentation* .... *answering questions*
    - e.g. evaluation of utility, cost and risk

- As a result, an architecture is
  - *"The fundamental organization of a system embodied by its components [jl: building blocks], their relationships to each other [jl: connectors and interfaces, dependencies] and to the environment and the principles guiding its design [jl: rationales for choices, rules & constraints for building blocks and connectors] and evolution"*

(IEEE Standard P1471 Recommended Practice for Architectural Description of Software-Intensive Systems)

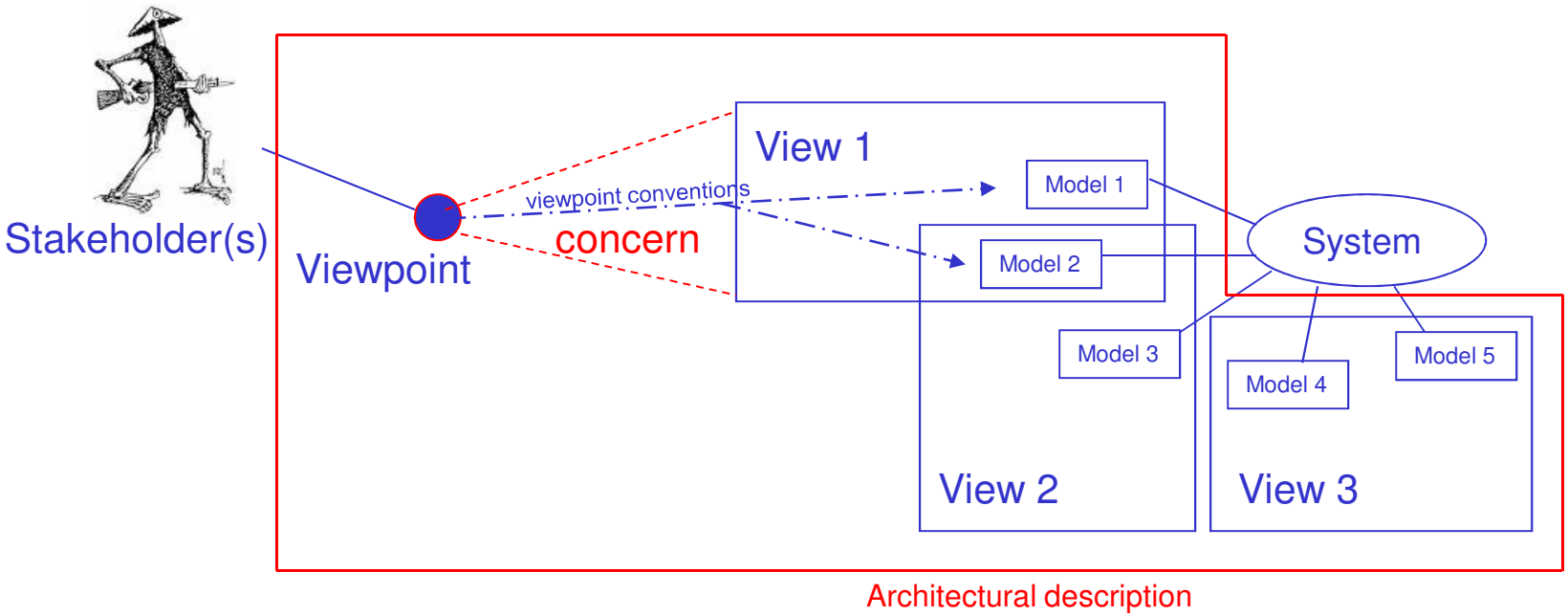
# Models, viewpoints and views

- **Model (of a system):** abstraction (of that system)
  - representation, leaving out details irrelevant to a given set of criteria ("concerns")
  - while preserving the properties of interest with respect to those concerns

- **Viewpoint:** a way of looking at a system from the perspective of a certain *stakeholder* with a particular *concern*
  - the viewpoint defines creation, depiction and analysis of a *view*
    - language, used models, notation, methods, analysis techniques, ...
  - the viewpoint defines the mentioned concerns *(criteria of abstraction)* to build a view from models
  - stakeholders: represent different areas of expertise / interest in the system and can have many concerns

- **View:** a collection of models
  - formal or informal, graphic, text
  - the view yields the means to address the concerns in the viewpoint, typically, via particular models
  - the view *conforms* to the viewpoint, i.e. it is described according to the conventions of the viewpoint and represents what you 'see' from that viewpoint

# Overall picture




from existing system to description: analysis, reverse engineering, documentation

from stakeholders and requirements to system: architecture design, detailed design, implementation

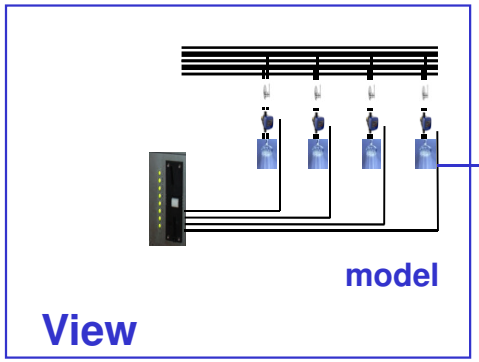
# Example

**Plumber, electrician**

Concern: how is the organization and operation of water and electrical systems

**Viewpoint:**

Detailed drawing of wires, pipes according to norm NEN ....




**System**

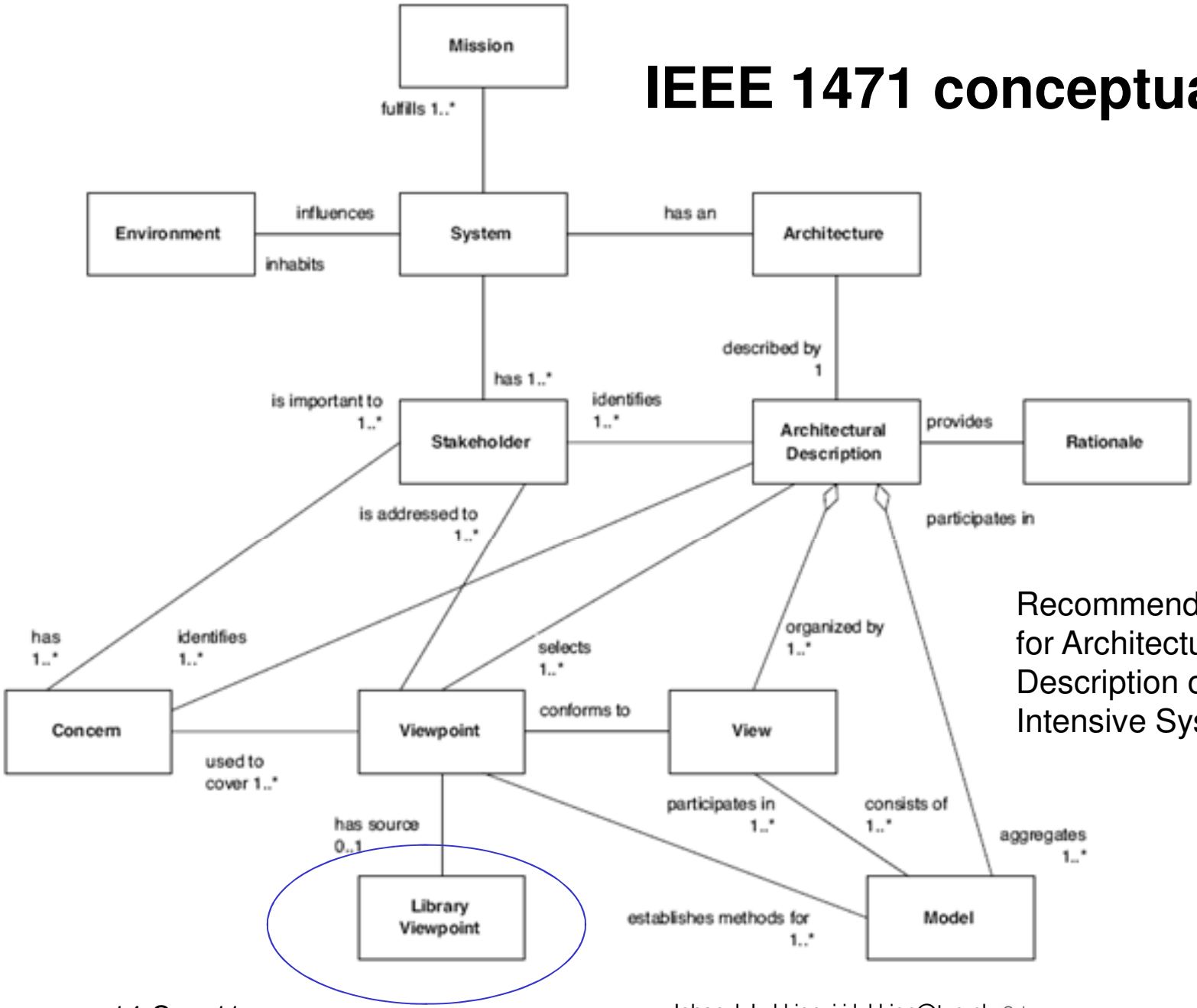
- This example view – as any architecture description - is incomplete:
  - e.g. sewer part is ignored
  - and *no dynamics is described!*
    - views have models describing static (structural, 'form') aspects as well as models describing dynamic (behavioral) aspects
    - often, the dynamic ones are ignored or assumed to be 'obvious'
    - they are *scenarios: system interactions ("structured use cases")*

# Example (cnt'd)

- Other viewpoints:
  - connection with environment
    - power and water source, heater interface
    - physical attachment, context
  - quality properties & evaluation (concerns of several stakeholders)
    - *system properties*: pipe sizes, pressure, heater capacity
    - *metrics*: water spending, cost, user experience
  - use cases, leading to scenarios in the views:
    - person taking shower, person collecting money, maintenance, installation

- Possible hierarchy
  - first level: decomposition as given
  - second level: architecture/design of building blocks, e.g. the shower control block, the switches

# IEEE 1471 conceptual model

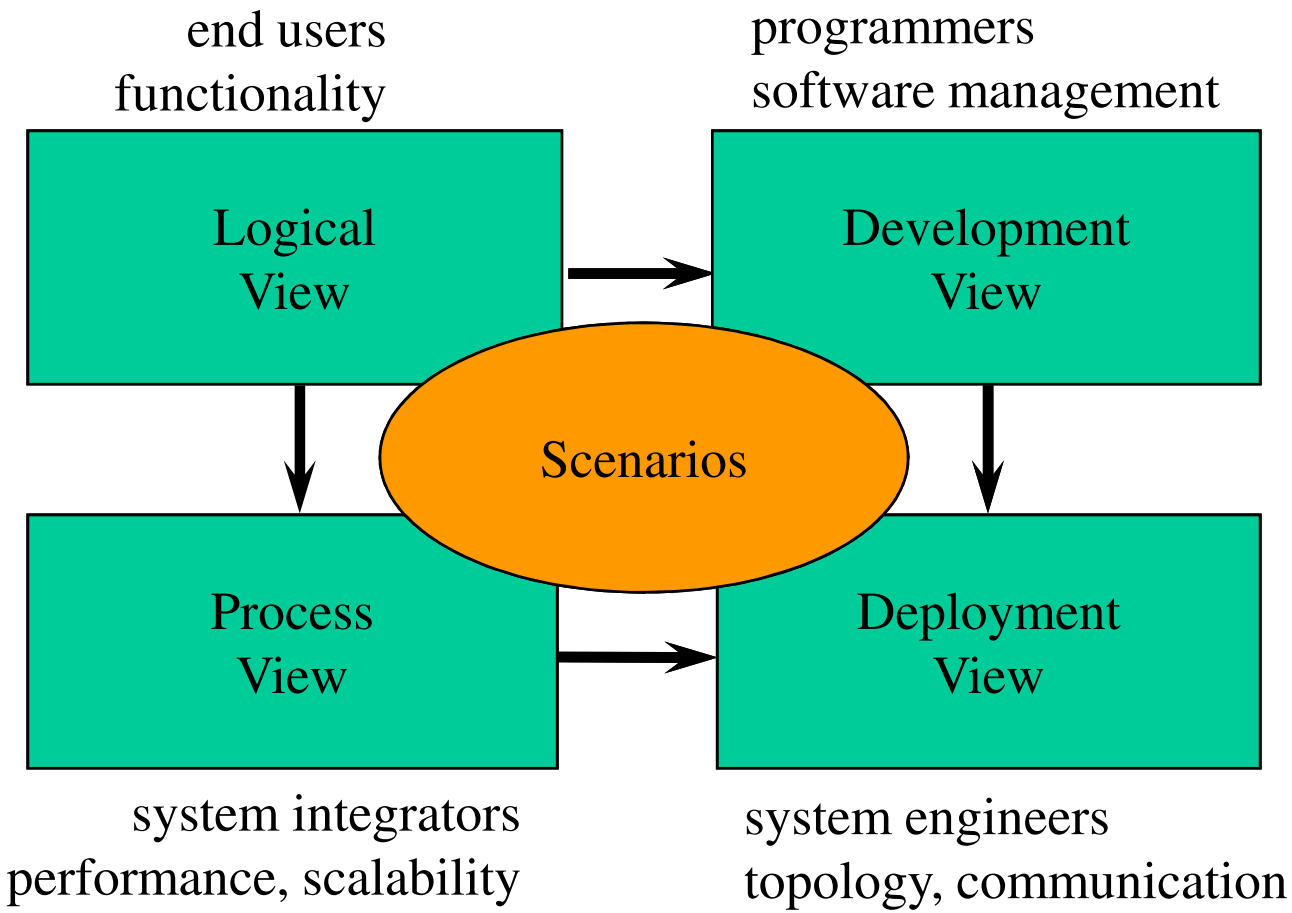



Recommended practice for Architecture Description of Software-Intensive Systems



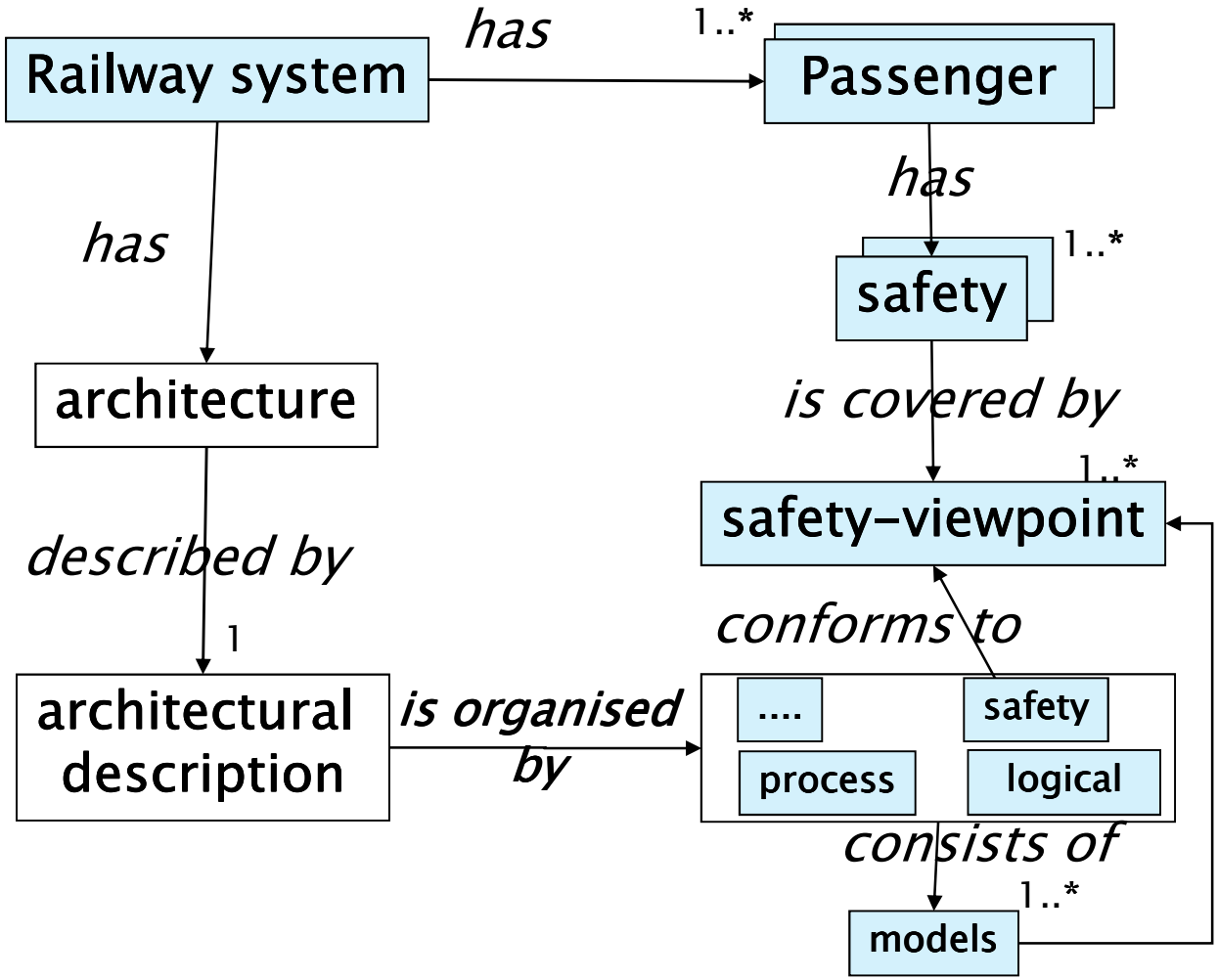
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

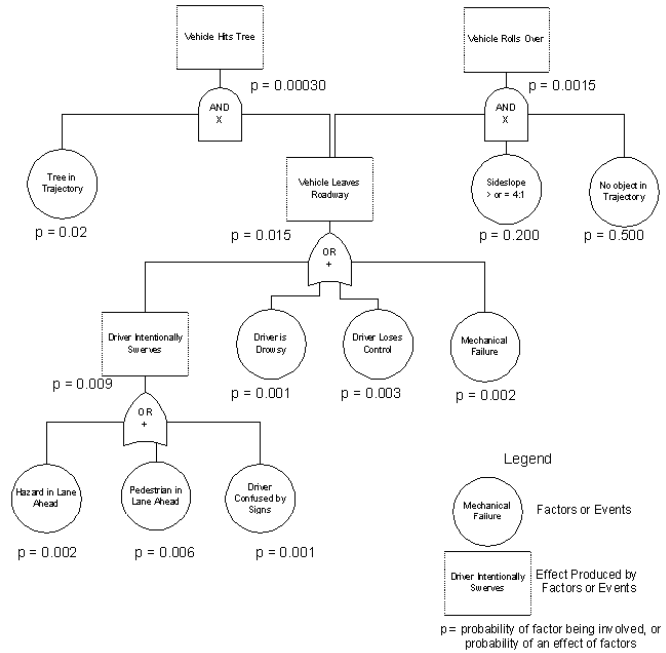
# A number of standard (library) views




4+1 View Model [Kruchten 95]

# Example (According to IEEE 1471)




*Example:*
*Fault tree model, using elements of several views*

based on slide from M.R.V. Chaudron

# Kruchten's views and stakeholders

- Article: Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12 (6), Nov. 1995, pp42-50
  - the paper does not describe it this explicit, but is interesting as an historic background

- Stakeholders: users, programmers, system engineers, system integrators
- Views:
  - User: *Logical view* – concerns: using the system, associated qualities
    - externally visible structure, typically modeled as an object diagram
  - Programmer: *Development view* – concerns: implementing and modifying the system
    - decomposition into subsystems, organization into files, components and modules
  - System integrator: *Process view* – concerns: performance aspects
    - units of deployment ('programs', 'components') and concurrency ('threads','processes')
  - System engineer: *Physical view, Deployment view* – concerns: putting sytems together
    - computers, networks, infrastructure, distribution, mapping of software to hardware (deployment), protocols
  - (+1) Scenario's (structured use cases) - sets of interactions with the system integrating the (models in the) views and providing behavioral models inside the views

# What's in an architecture? (or: what makes up the views?)

- Models, models, models, structural and behavioral
  - pictures, diagrams, tables, formal descriptons with enough detail...
  - ...to understand how the concerns are addressed
    - particularly, the relationship between 'ilities' and design decisions
  - ...to proceed with (to relate to) subsequent design stages
  - ...to examine and operate the system from the perspective of all *stakeholders,* admitting trade-offs
    - e.g. balancing stakeholder concerns, balancing cost functions

- Though the proposed views cover many common aspects, they are not complete
  - for example, economic aspects are ignored, as well as system ownership, security and many other 'ilities'

# Logical view (user view)

- Captures the user's view on the system, i.e. what a user encounters while using the system
  - classes and objects (class instances) documenting user-visible entities
    - including *interfaces,* that imply *responsibilities*
  - interaction diagrams: interactions describing usage scenario's
  - state diagrams: describing state changes as result of interactions

- Typical relations: is-a, has-a, consists-of, associates

- Don't misunderstand the name 'logical'; also other stakeholders use logical models
  - e.g. developers: functional blocks and patterns of their organization

# Example – Operating System logical view

- two types of users: ‘desktop user’, programmer
  - note: double role for programmer: designer + user of OS

- ‘desktop user’: interaction with the system through programs (shell, editors, GUI)
  - many use cases: installing programs, managing disks, issueing commands, ...
  - corresponding dynamics: the sequence of mouseclicks, commands

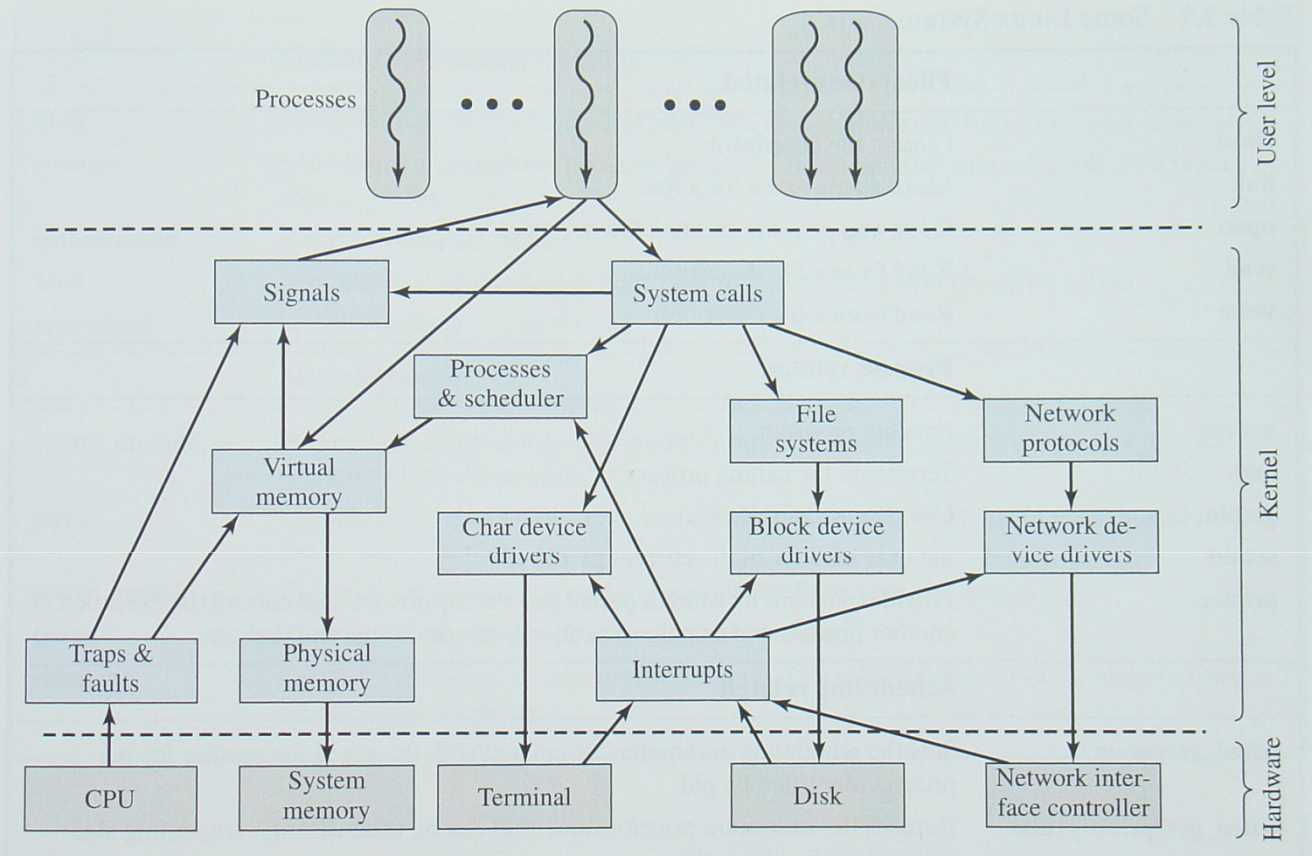
- programmer: API, interfacing with file system, process management, ... through *system calls*
  - dynamics: call sequences, IPC, signaling, ...

# Development view (implementation view)

- *Components, functions, subsystems* organized in modules and packages
- Component/module *interface descriptions*, *access protocols*
- Logical organization – layering of functionality, dependencies
- Organization into files and folders

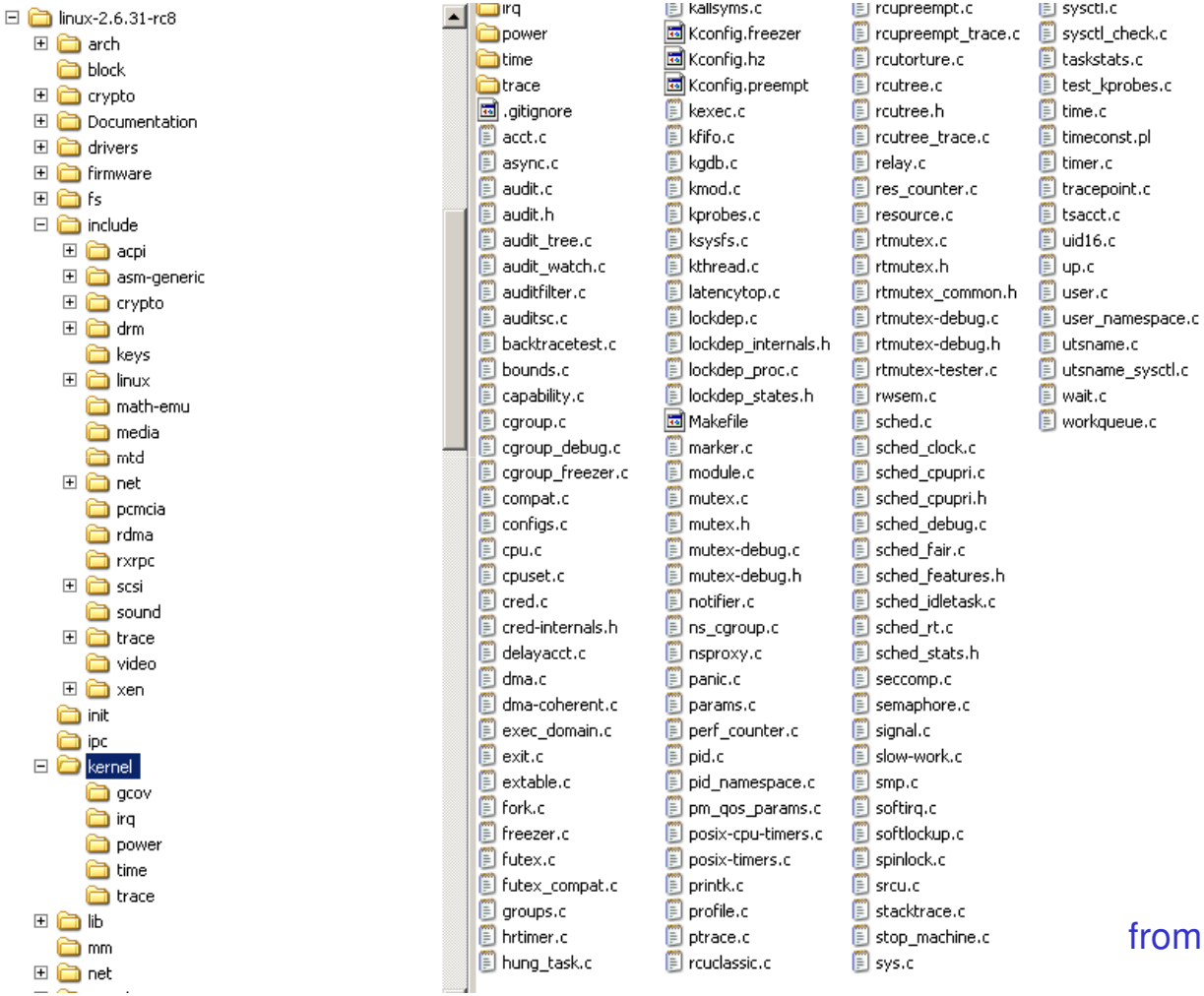
- Typical relations: uses, contains, shares, part-of, depends-on

# Example: Linux functional units and relations




- Note: semantics of this picture is unclear
  - boxes: functional units (with responsibilities indicated by their names)
  - arrows: ‘is used by’, ‘acts on’
- Can this be a model in a user (logical) view?

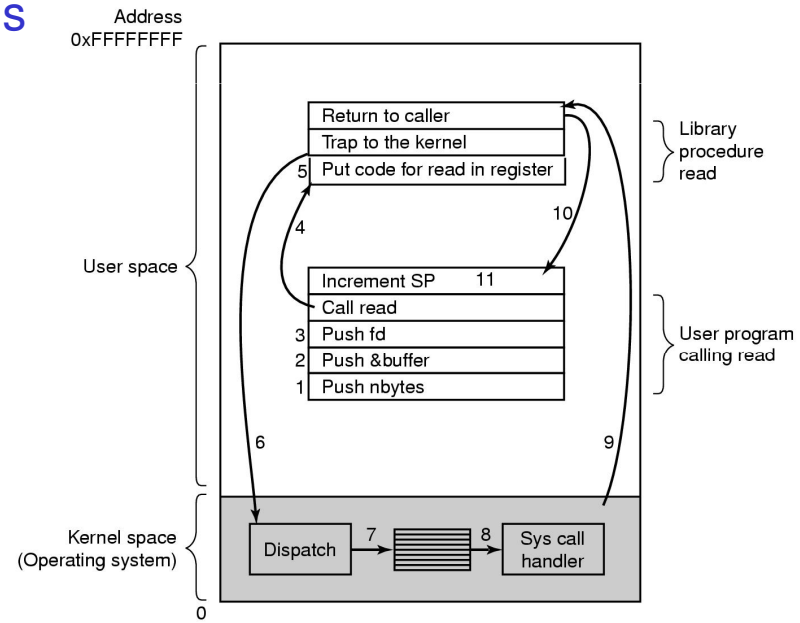
# Example: Linux source tree




from http://www.kernel.org/

# Further development descriptions for the OS?

- Relationships between source files and functional units
- Dependencies among source files
- Development view for system programs
  - i.e. how the components from the user view are developed
- User scenarios will show:
  - the relationship between kernel programs and kernel
  - the execution of system calls through the kernel
    - and also through the code
- Developer scenarios will show
  - how to add a new device driver
  - how to compile and debug a kernel
  - .....



# Process view

- Concurrent *activities* inside the system
  - activity: independent thread of (code) execution: *thread*
- Mapping of applications to distinct *memory spaces* and *units of execution*
  - unit of execution: process, thread
  - memory space: associated with a *process*
- (Choice of) inter-process communication (IPC), protocols
- *Scheduling* of activities such as to satisfy time and resource constraints

- Typical relations: derived from communication choices and from Logical and Development views
  - 'communicates-with' (messaging), 'calls' (RPC), 'call-back' (event), ...

- Addresses <u>performance concerns</u>, <u>mapping of functionality (components) to execution environment</u> and <u>information flow through the system</u>

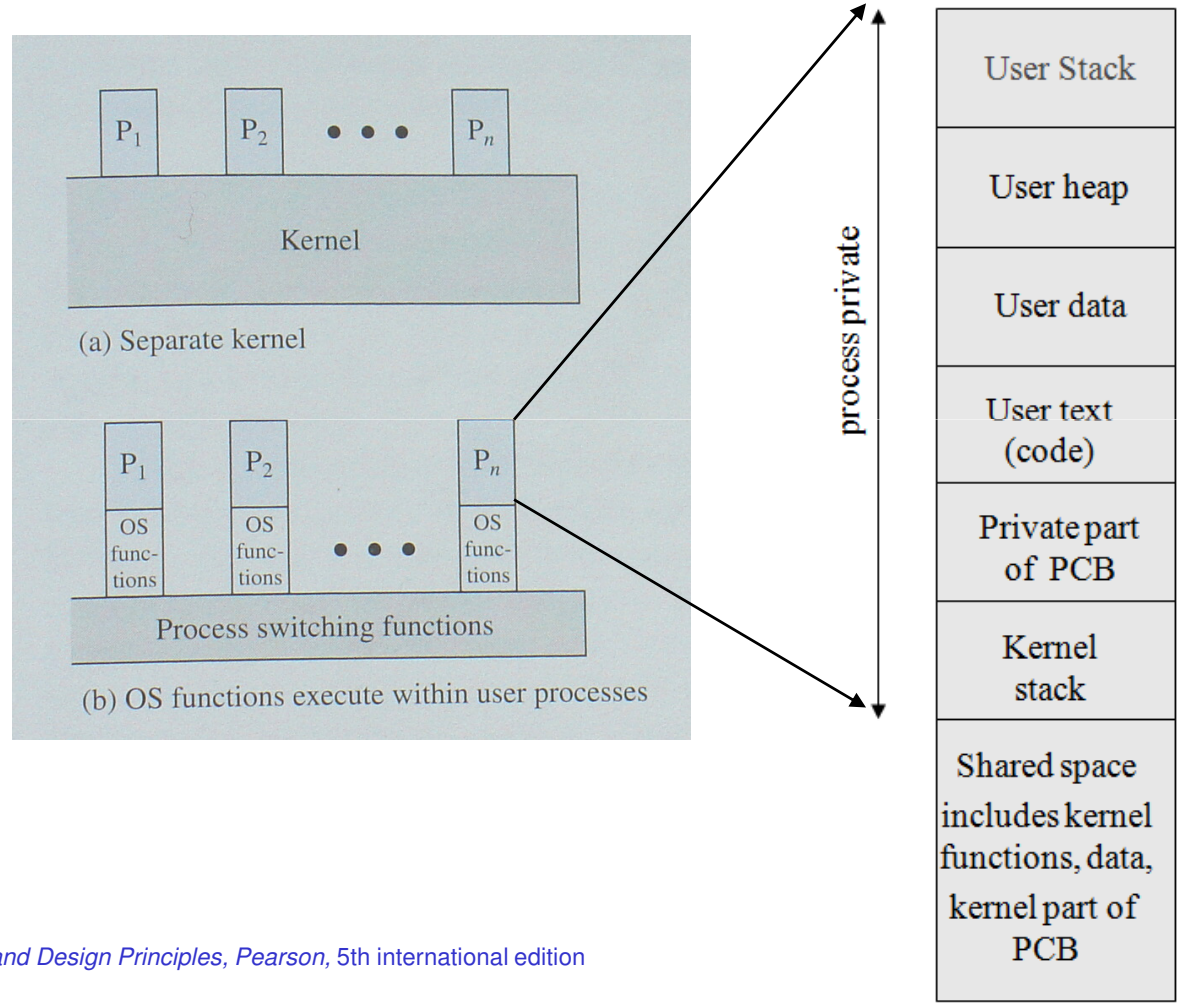
# Example: OS process view

- Specifies concurrency
  - within the kernel, e.g.,
    - interrupt service routines (ISR) concurrent with other activity
    - kernel execution within process scope
    - effect of blocking system calls on running process
  - during execution, e.g. kernel processes
    - scheduler – invoked upon yielding of the processor
    - kernel event handler – servicing *bottom halves* of ISR, and other pending kernel tasks (Linux: *keventd*)
- Inter-process communication

- Probably, since we're implementing processes:
  - logical structure of processes
    - (though this could be an element of the development view as well)

# Example: process organization

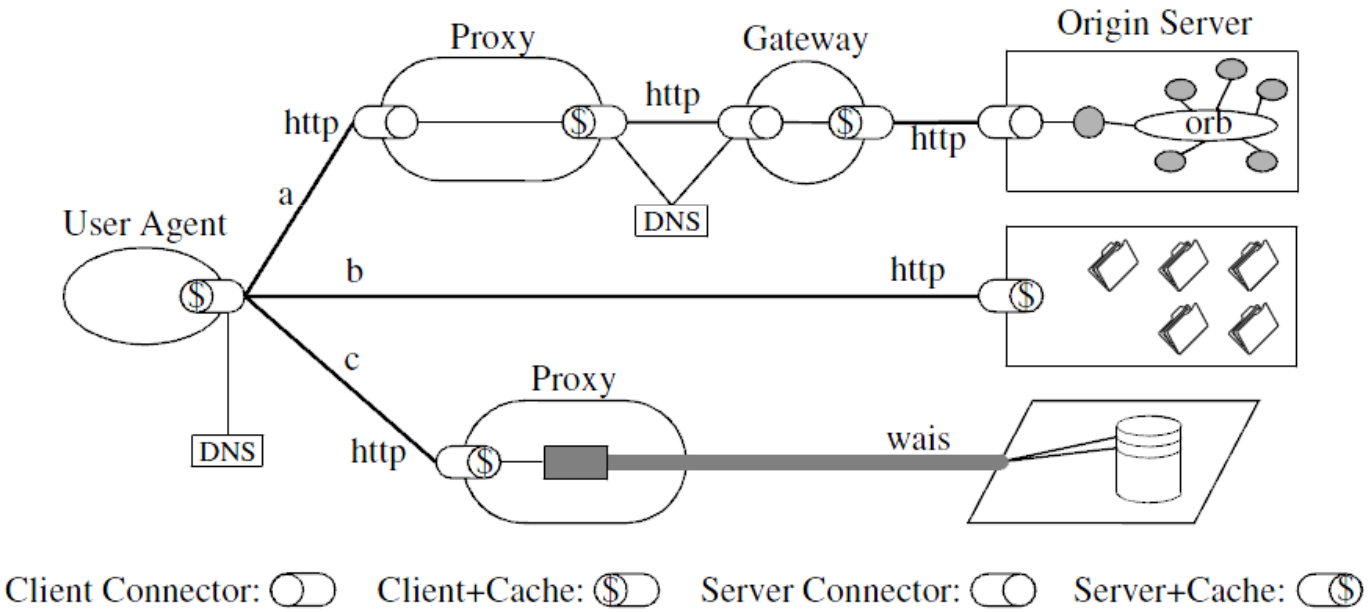
- (a) kernel as separate process
- (b) OS executing in process context




(from W.Stallings, *Operating Systems, Internals and Design Principles, Pearson,* 5th international edition

# Example: process view (model) in REST

- REST: Representational State Transfer




(model in the process view from R. Fielding, Architectural Styles and the Design Network-based Software Architectures)

# Physical and deployment view

- Machines (processors, memories), networks, organization of interconnect
  - including specifications, e.g. speeds, sizes
- Deployment view: mapping of elements of other views to machines

- Typical relations: connects-to, contains, maps-to
  - also: relations derived from purpose

- Addresses concerns of performance (throughput, latency), availability, reliability, etc., together with the process view

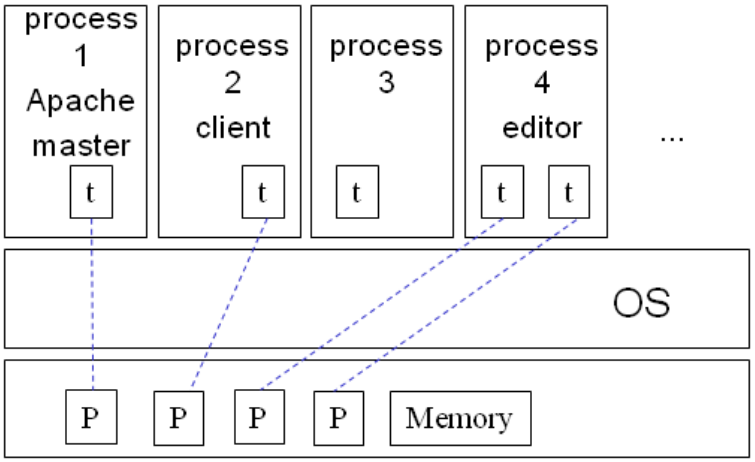
# Example: OS deployment view

- Procedures how to get an OS running on a specific machine
  - 'attachment' of OS to hardware
  - boot procedure

- In case of *distributed OS*: mapping of the OS parts from other views

- Mapping of threads and processes

- Processor type, memory



# Architecture, views and models

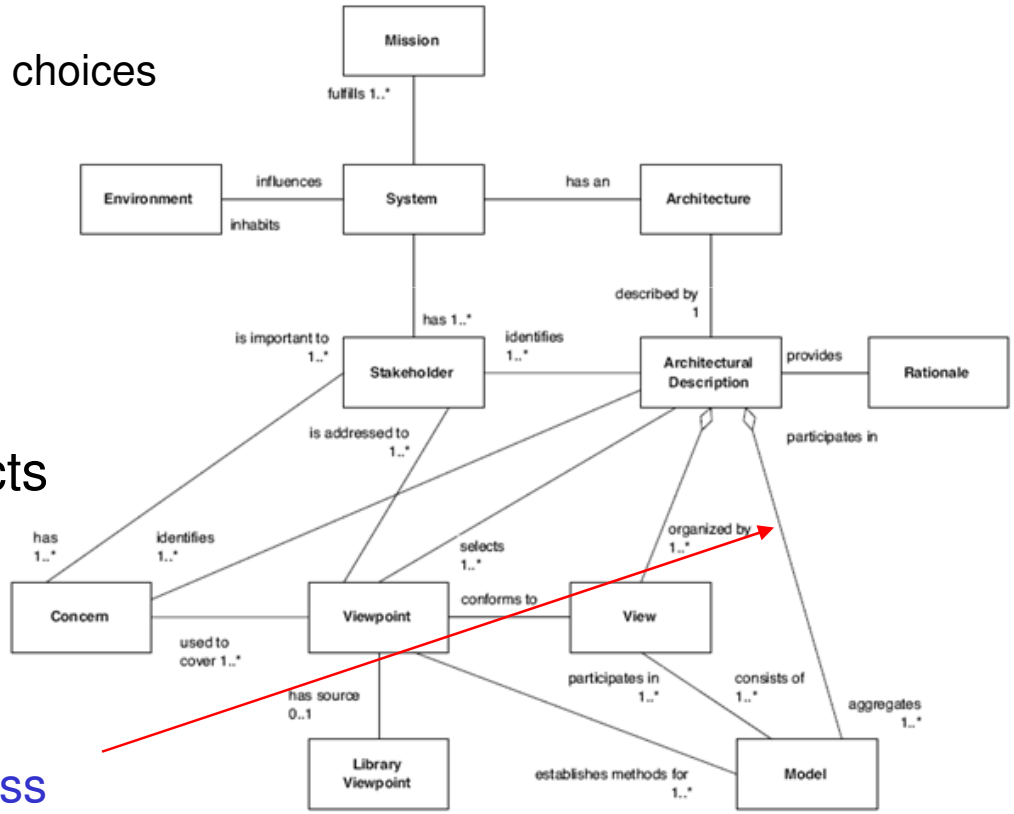
- Views need to be systematically related, typically by mappings
  - processes execute components, connectors are mapped to networks IPC or calls, objects use methods from their classes, etc.
  - scenario's are instrumental here
    - e.g. a behavorial model can be part of both logical view and process view

- Correctness concerns for the views
  - *consistent models:* inside as well as across views
  - *relative completeness*: do we have 'everything' that is important?

- Notations for the different views vary
  - are, in fact, defined by the viewpoint (UML for logical view is standard)
  - often ad-hoc for system architectures and physical view
  - both suffer from lack of completeness and lack of clear semantics

# Incompleteness

- An architectural description is always incomplete
  - it specifies a class of solutions
    - we want to leave freedom,
      e.g. for allowing implementation choices
  - it simply does not describe
    all system details
  - hence, *relative* completeness to
    a set of concerns




- The views do not cover all aspects
  - e.g. architecture design
    decisions may not be
    relevant in the logical view
  - hence, the architecture contains
    models that do not clearly address
    a stakeholder concern

# Architecture and extra-functional properties

- Extra-functional concerns are addressed through the architecture
  - Each stakeholder can come up with an extra-functional concern
    - e.g. security or performance
  - A very important general concern is *to limit dependencies*

- These concerns often specify *emergent system properties (cross-cutting concerns)*
  - they arise from the collaboration of system components
  - e.g. security, performance

- In the architecture, all these (conflicting) concerns are balanced

- It is highly desirable that the architecture can be used to see if these concerns are met
  - metrics, based on the architecture
  - transparency towards the final system realization(s)

# Some examples of ‘ilities’

Accessibility, Understandability, Usability, Generality, Operability, Simplicity, Mobility, Nomadicity, Portability, Accuracy, Efficiency, Footprint, Responsiveness, Scalability, Schedulability, Timeliness, CPU utilization, Latency, Throughput, Concurrency, Flexibility, Changeability, Evolvability, Extensibility, Modifiability, Tailorability, Upgradeability, Expandability, Consistency, Adaptability, Composability, Interoperability, Openness, Integrability, Accountability, Completeness, Conciseness, Correctness, Testability, Traceability, Coherence, Analyzability, Modularity, Reusability, Configurability, Distributeability, Availability, Confidentiality, Integrity, Maintainability, Reliability, Safety, Security, Affordability, Serviceablility, …

# Good questions to ask

- What are my building blocks (at this level, in this view), connectors, rules, constraints?
- How do diagrams relate? Are they consistent?
- Is the architecture complete?
- What is the relation between design decision and extra-functional properties?
  - can I reason or compute based on the diagrams?
- What is given from the environment?
- What is the importance of each 'ility'?
  - nobody wants a system that is not maintainable....
  - ....but perhaps you do not want to double the price for that reason
  - ....however, a tradeoff might be possible
- Can I realize (refine) the architecture towards an implementation?

# Alternative definitions

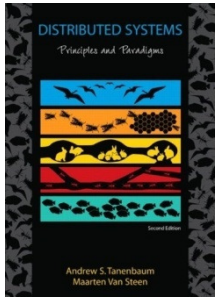
- *The software architecture of a program or system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships between them.*
  - *Software Architecture in Practice,* Bass, Clements and Kazman

- *[An architecture is] A representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.*
  - CMU, Software Engineering Institute

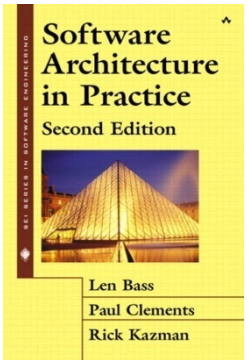
- Architecture = {Elements, Form, Rationale}
  - D.E. Perry and A.L. Wolf, "Foundations for the Study of Software Architecture", ACM Software Eng. Notes, vol. 17, no. 4, 1992, pp. 40–52.
- see also http://www.sei.cmu.edu/architecture/start/community.cfm

# Books

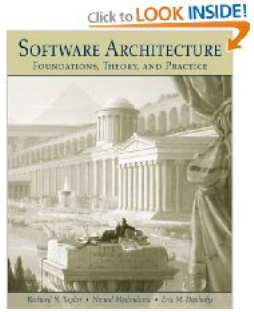
- Tanenbaum & van Steen, Distributed Systems, Principles and Paradigms 2nd ed., Pearson, 2007




- Software Architecture in Practice, Second Edition, L. Bass, P. Clements, R. Kazman, SEI Series in Software Engineering, Addison-Wesley, 2003




- Software Architecture, Foundations, Theory, and Practice, R.N. Taylor, N. Medvidovic, E.M. Dashofy, Wiley & Sons, 2009




- Recommended Practice for Architectural Description, IEEE STD 1471-2000, 23 pages

# For next two weeks

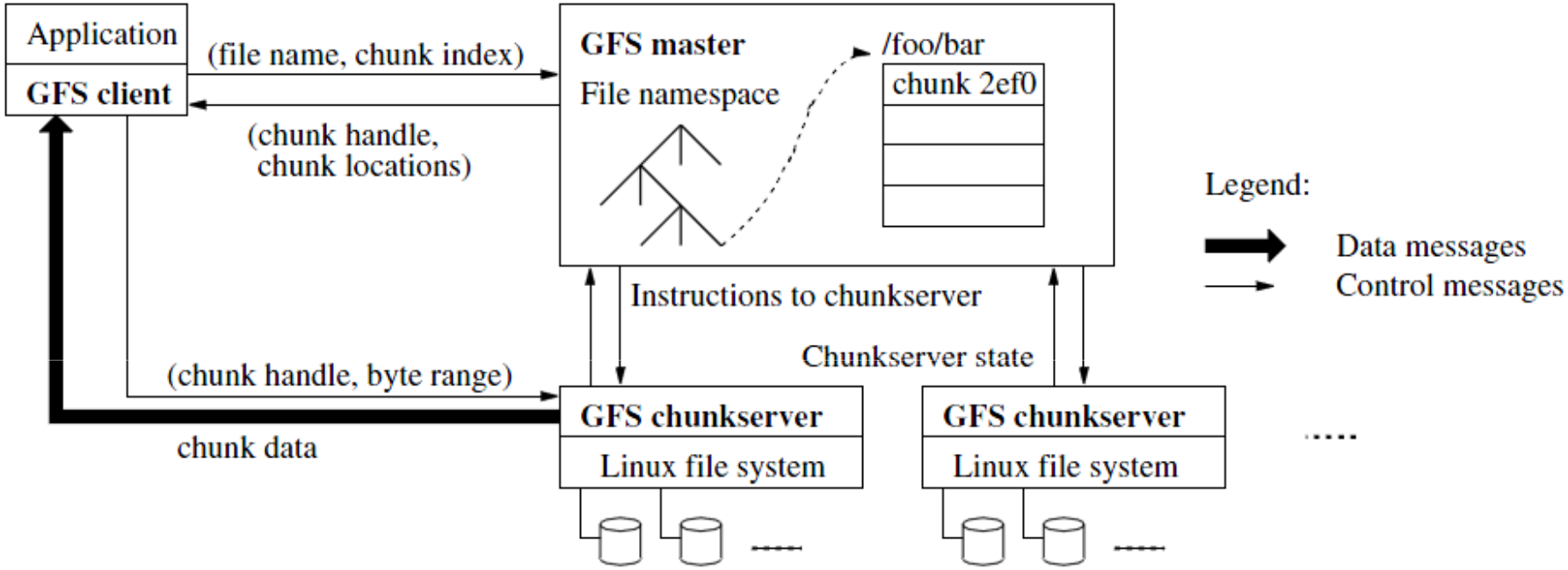
- Software Architecture: IEEE Standard 1471-2000
- (for historic reference) P. Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12 (6), Nov. 1995, pp42-50
- The Decision View's role in Software Architecting Practice, IEEE Software March/April 2009, pp1-8.

- Make the next exercise
  - (hand in through Peach before third lecture)

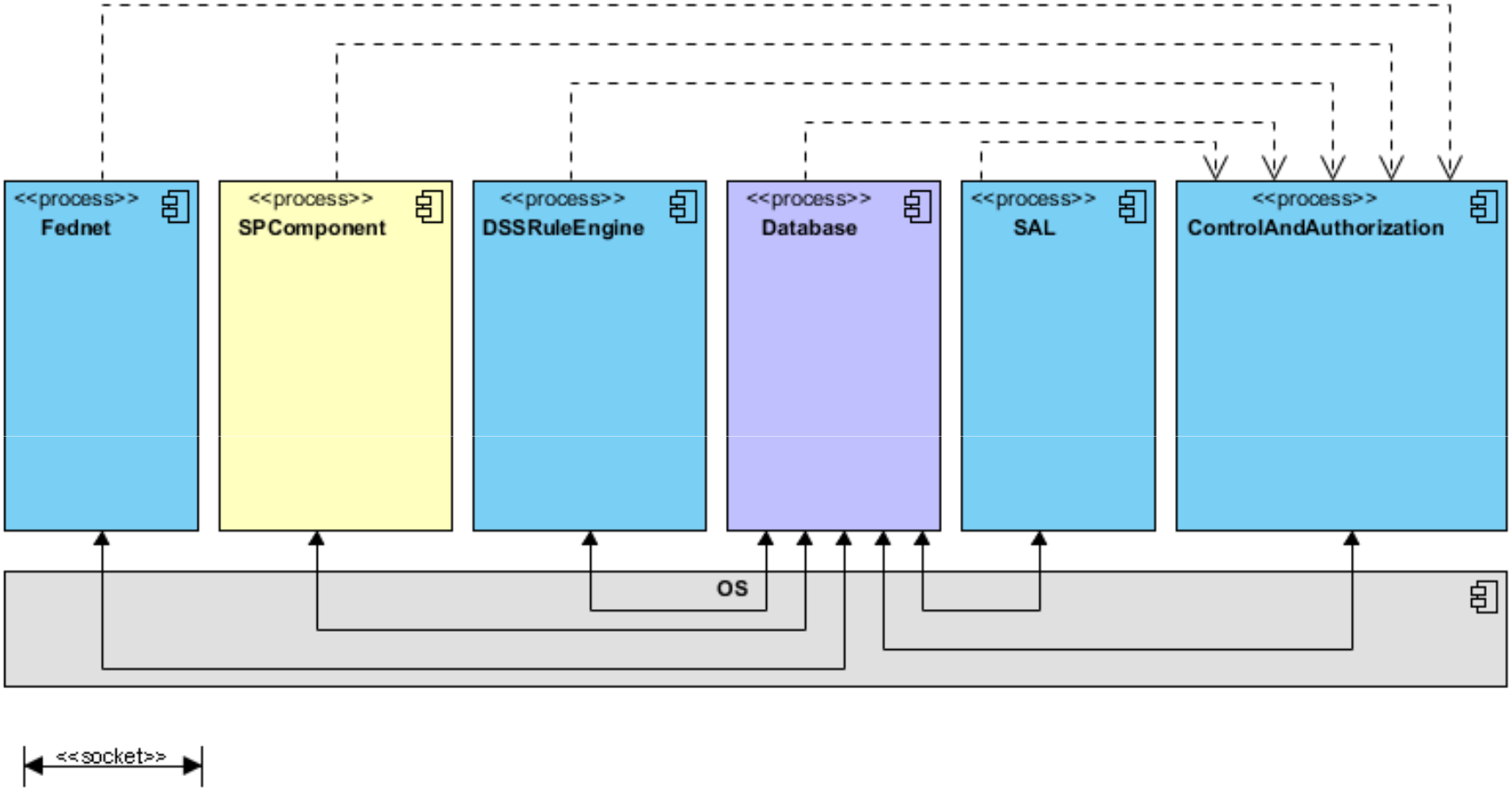
# Exercise

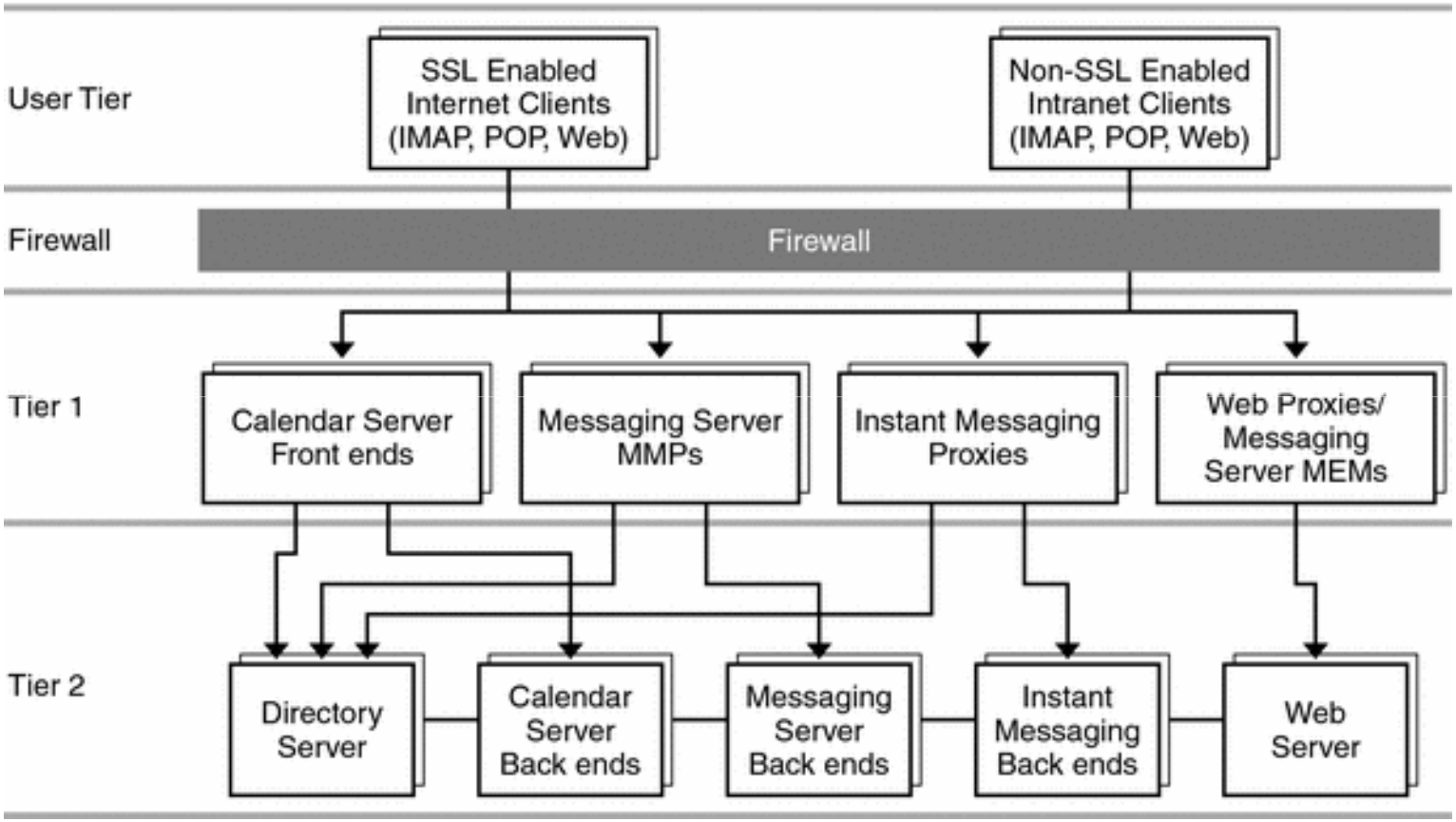
- Consider the models on the following slides:
  - what view(s), concern(s) does it address (motivate)?
  - for which stakeholders is it relevant?
  - can you use it to discuss extra-functional requirements?
    - performance, reliability, security, maintainability
  - is there a concept of distribution?
  - what building blocks and connectors do you see? what do they mean? are they conceptual or physical?
  - comment on the clarity/semantics of the diagram

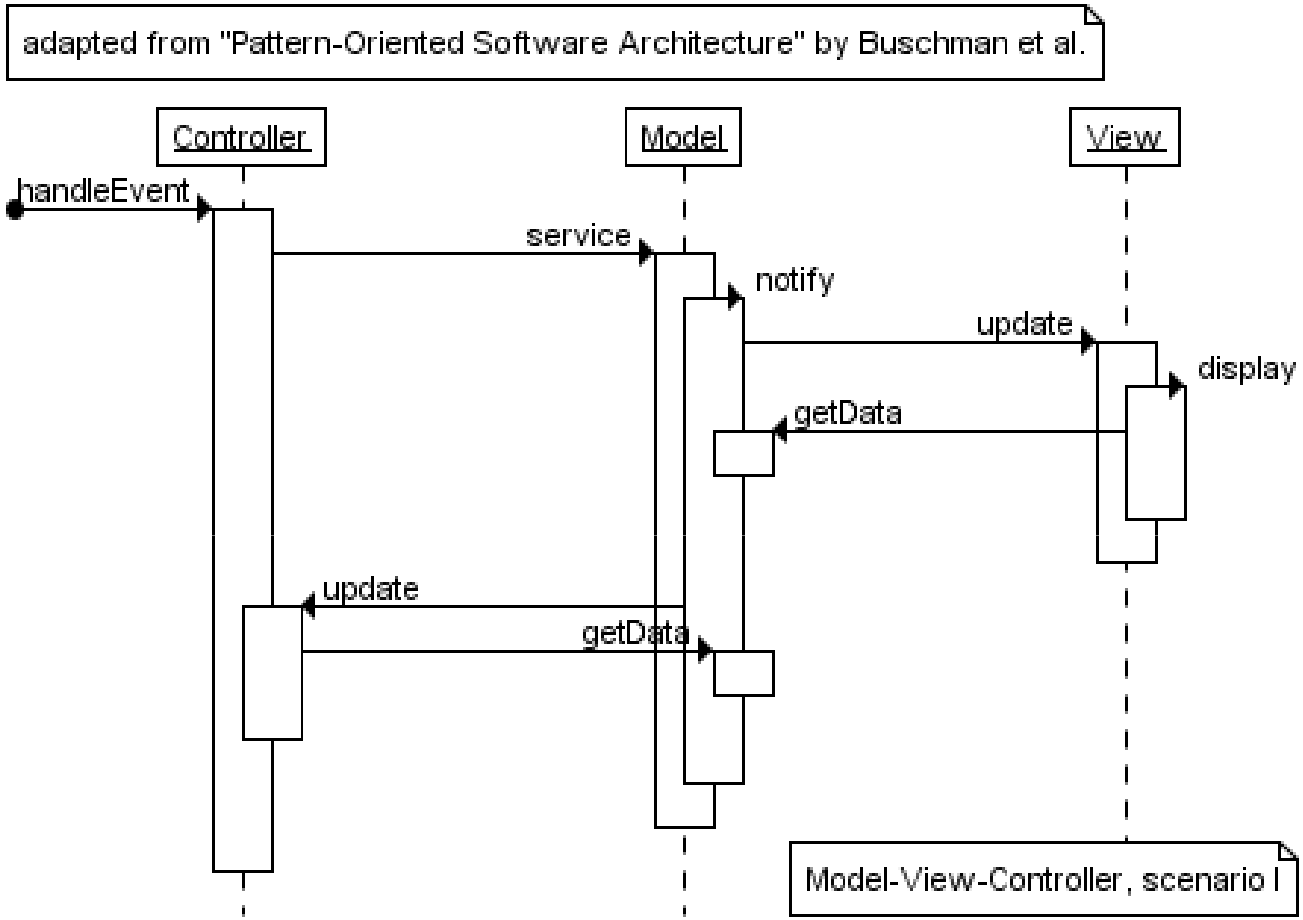
(pictures thanks to M.R.V. Chaudron, and WWW)

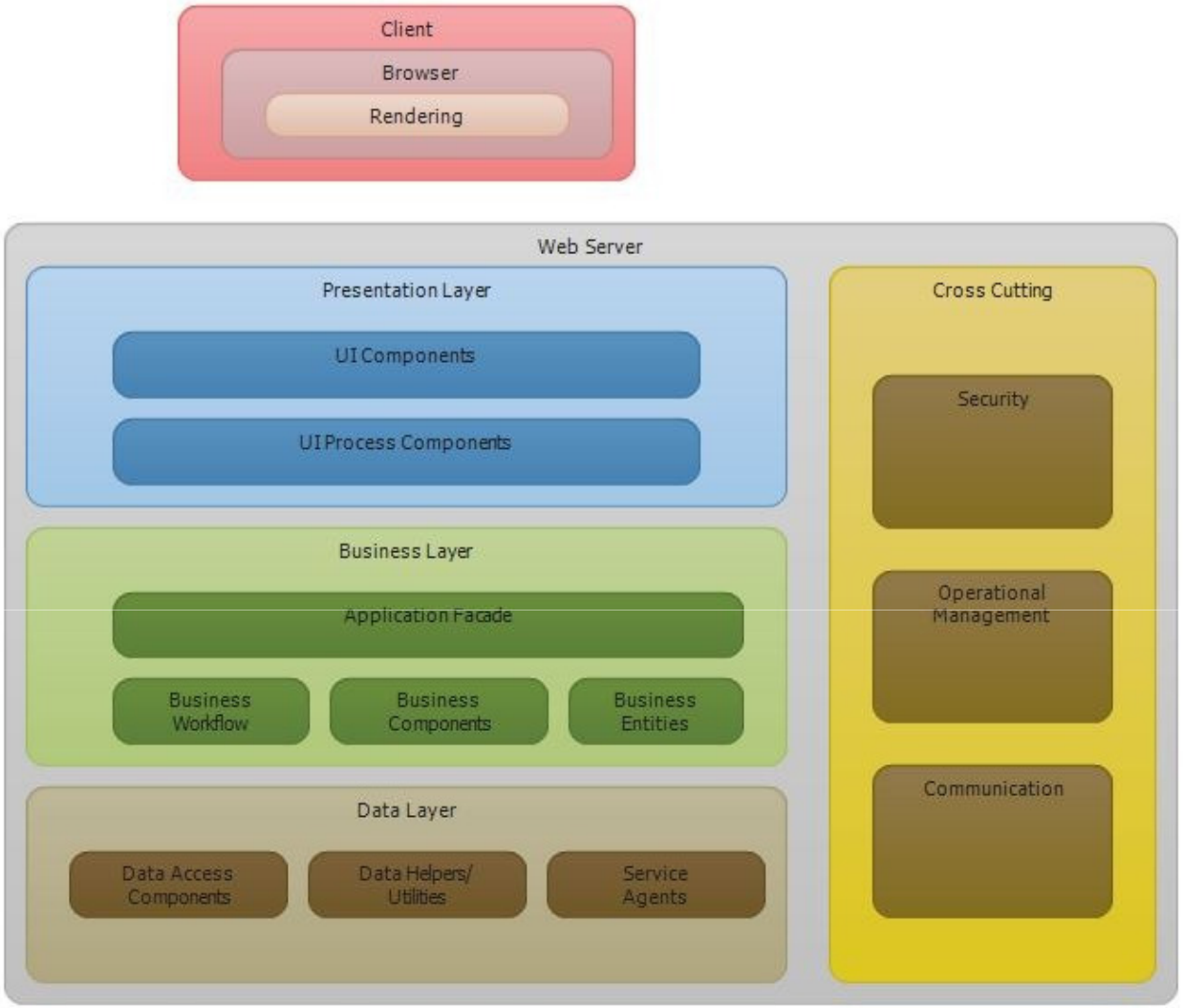



- see http://labs.google.com/papers/gfs-sosp2003.pdf

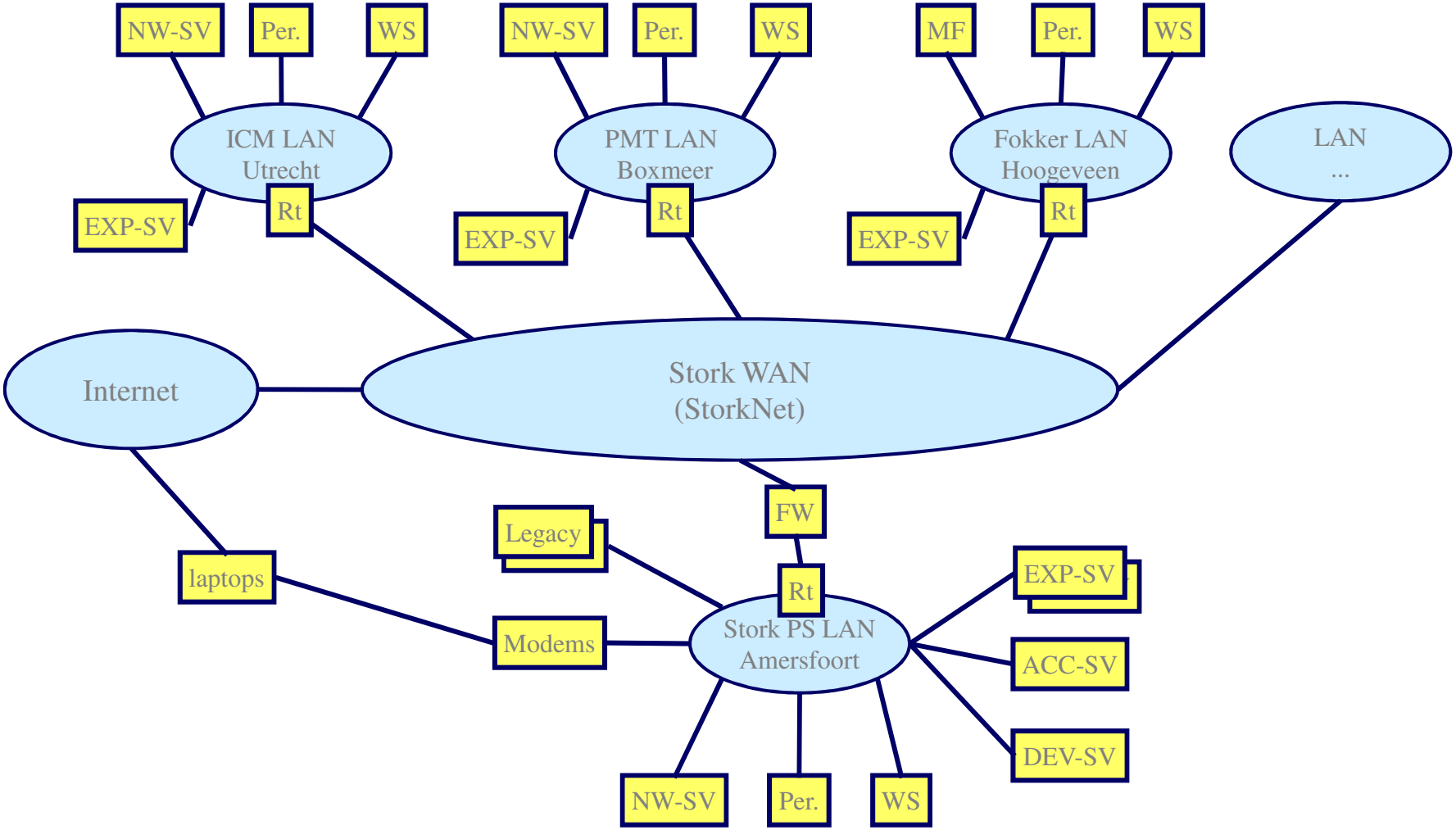

<<process>>
Fednet
<<process>>
SPComponent
<<process>>
DSSRuleEngine
<<process>>
Database
<<process>>
SAL
<<process>>
ControlAndAuthorization
OS
<<socket>>

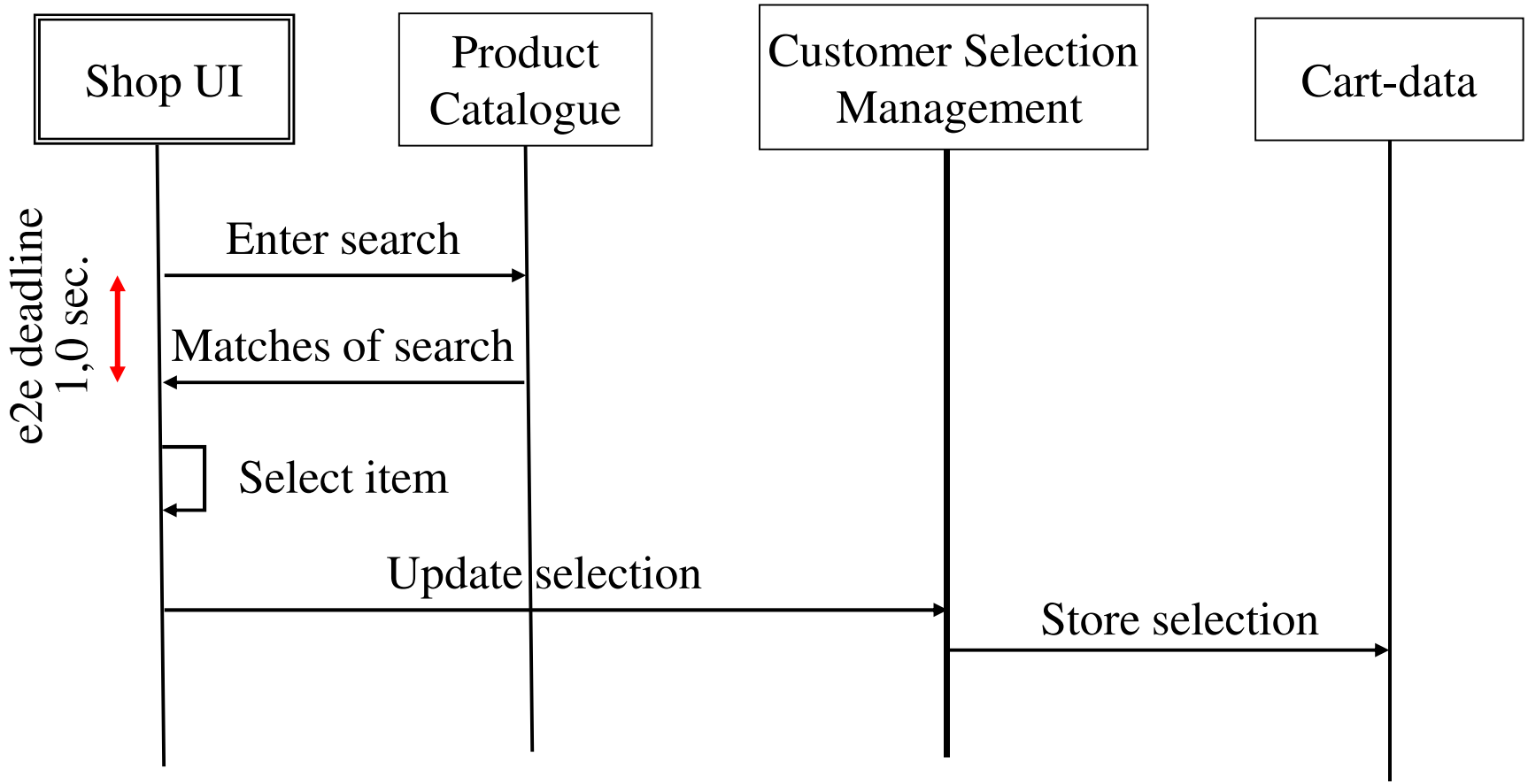

User Tier
SSL Enabled Internet Clients (IMAP, POP, Web)
Non-SSL Enabled Intranet Clients (IMAP, POP, Web)
Firewall
Firewall
Tier 1
Calendar Server Front ends
Messaging Server MMPs
Instant Messaging Proxies
Web Proxies/ Messaging Server MEMs
Tier 2
Directory Server
Calendar Server Back ends
Messaging Server Back ends
Instant Messaging Back ends
Web Server


adapted from "Pattern-Oriented Software Architecture" by Buschman et al.
Controller
Model
View
handleEvent
service
notify
update
display
getData
update
getData
Model-View-Controller, scenario I




Source: Microsoft Application Architecture Guide, 2nd Edition.
Refer to "Chapter 21: Designing Web Applications" for more information.
URL: http://msdn.microsoft.com/en-us/library/ee658099.aspx


NW-SV
Per.
WS
ICM LAN
Utrecht
EXP-SV
Rt
NW-SV
Per.
WS
PMT LAN
Boxmeer
EXP-SV
Rt
MF
Per.
WS
Fokker LAN
Hoogeveen
EXP-SV
Rt
LAN
...
Internet
Stork WAN
(StorkNet)
FW
Legacy
laptops
Rt
Modems
Stork PS LAN
Amersfoort
EXP-SV
ACC-SV
DEV-SV
NW-SV
Per.
WS


Shop UI
Product Catalogue
Customer Selection Management
Cart-data
e2e deadline 1,0 sec.
Enter search
Matches of search
Select item
Update selection
Store selection

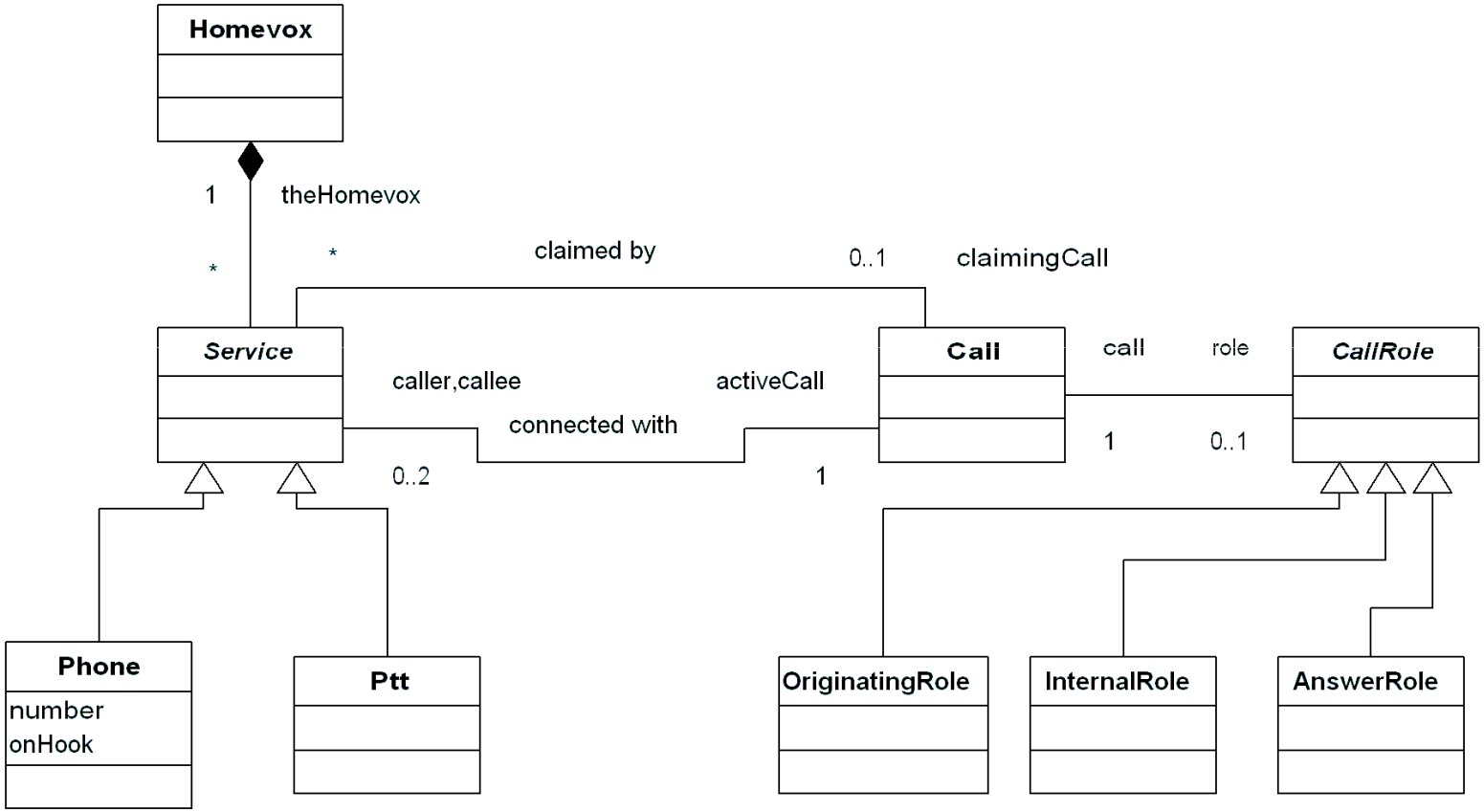
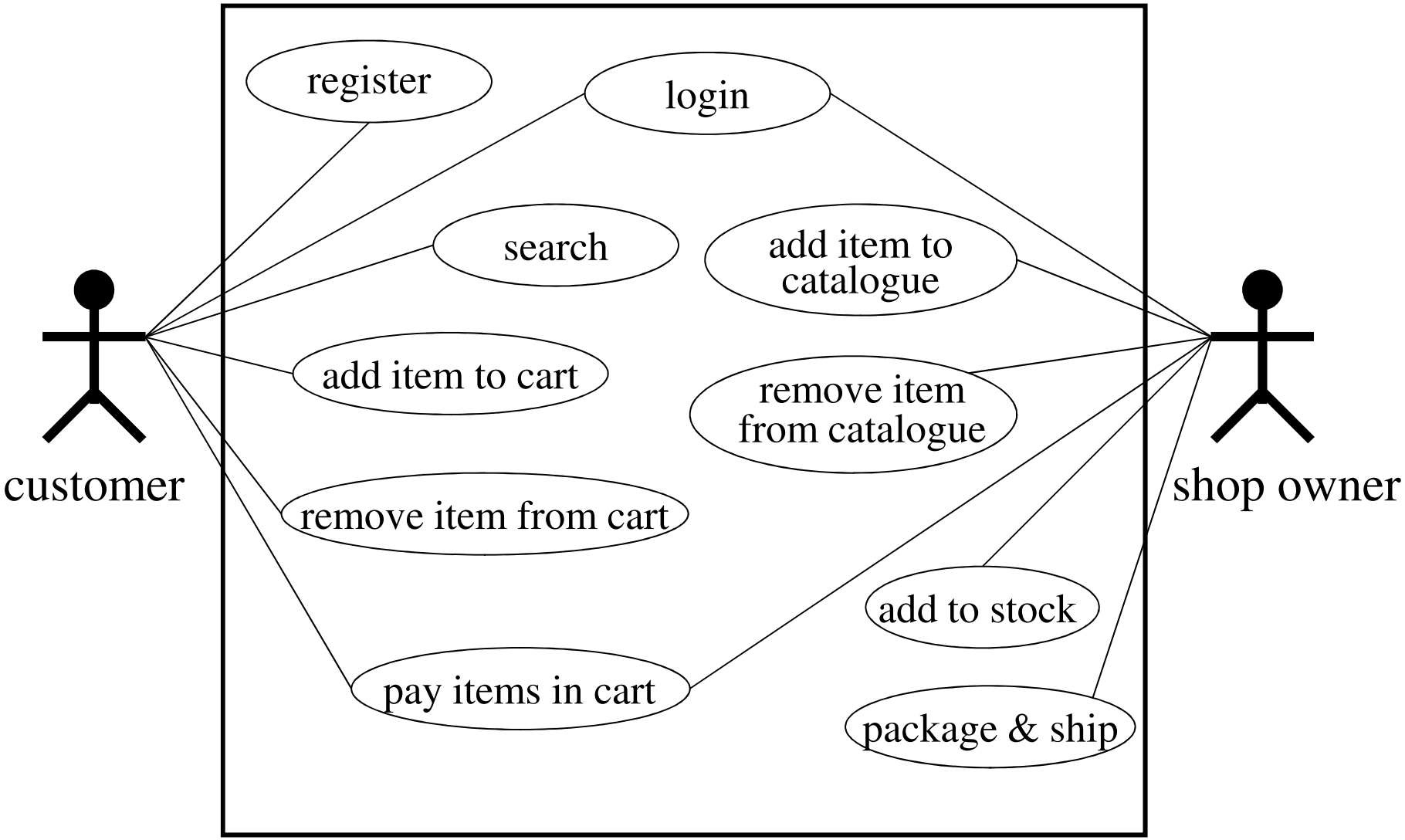



figure from Rein Smedinga, RuG

# Case: Web shop



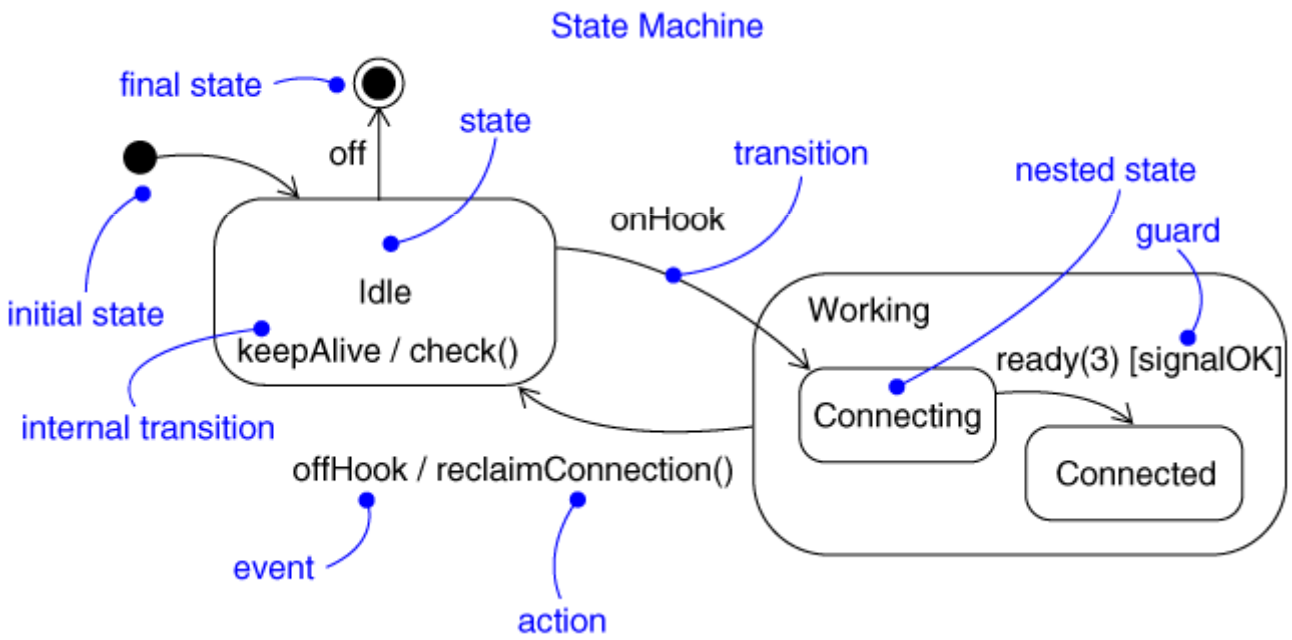
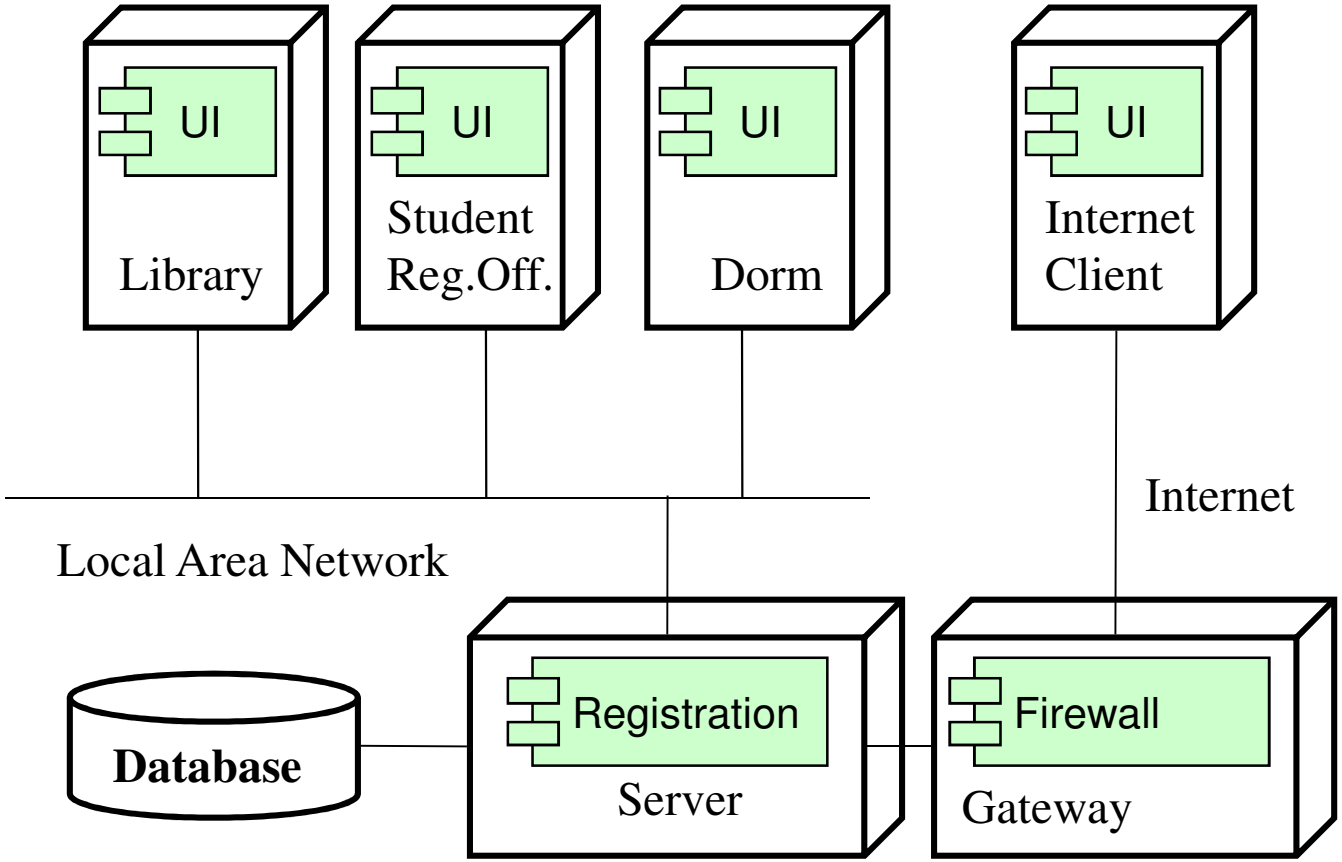



figure from Rein Smedinga, RuG


UI
Library
UI
Student
Reg.Off.
UI
Dorm
UI
Internet
Client
Internet
Local Area Network
Database
Registration
Server
Firewall
Gateway

# Example from HSN

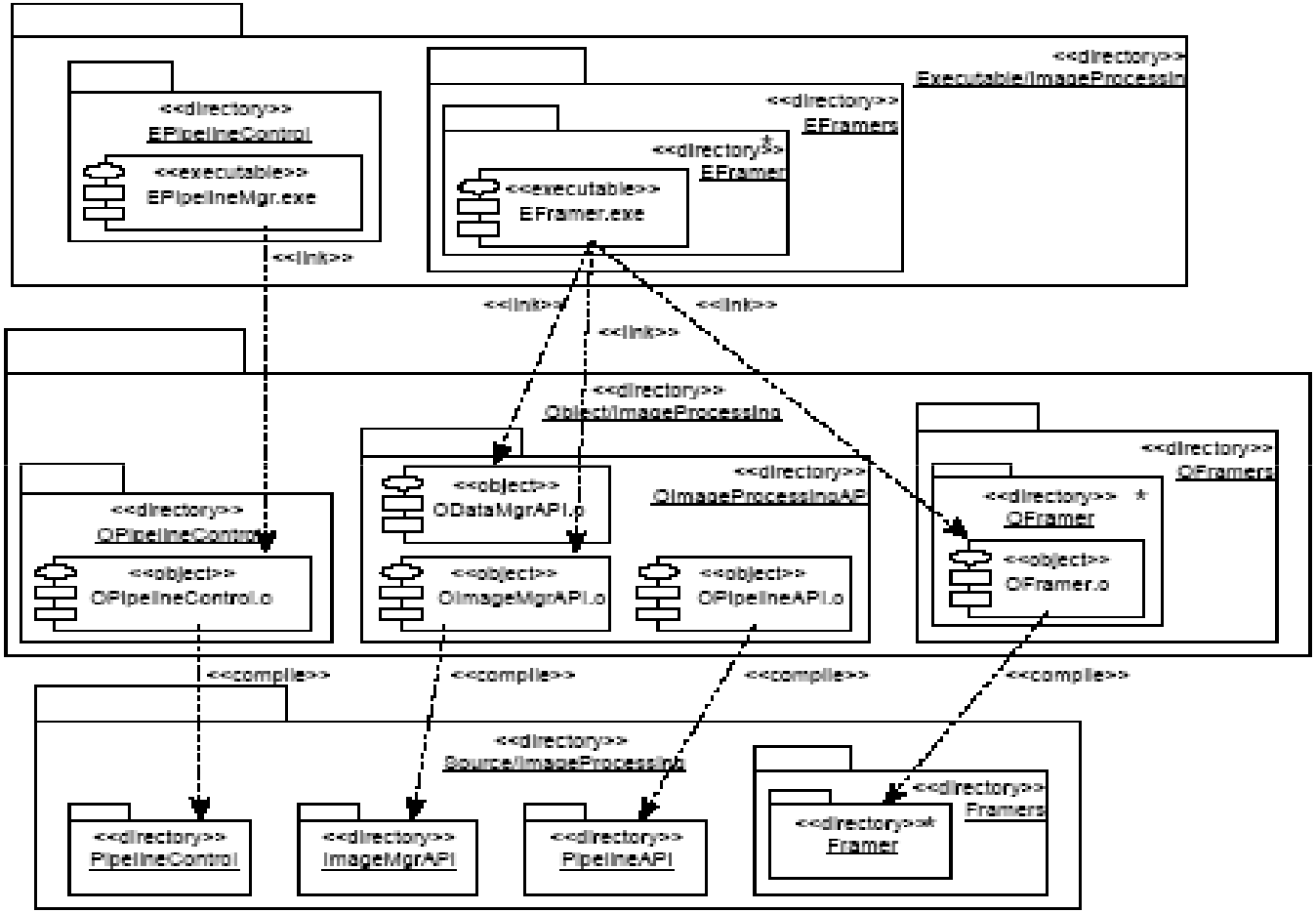



Figure 8: Dependencies among Source, Object, and Executable Files