Architecture of Distributed Systems 2017-2018

Introduction

Original : J.J Lukkien

Revision: R.H. Mak

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 (as explained in the ISO/IEC/IEEE 42010 standard)

- Students know some common viewpoints, the principle concerns they address and typical models they use.

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?
  - how to 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
    - including its role in and interaction with its environment
  - analysis of design alternatives, and of general system properties
    - *before such system is built*
    - *or without having access to it*
  - communication between team members and with customers
  - communication (documentation) for handing off parts of the work
  - documentation for later reference
    - in the maintenance phase of the lifecycle
    - to guide the evolution of the system
  - decomposition and synthesis of parts
- Larger technical systems are simply too complex and long-lasting for one person

Some (a priori) architectural questions

Will it not kill the patient?

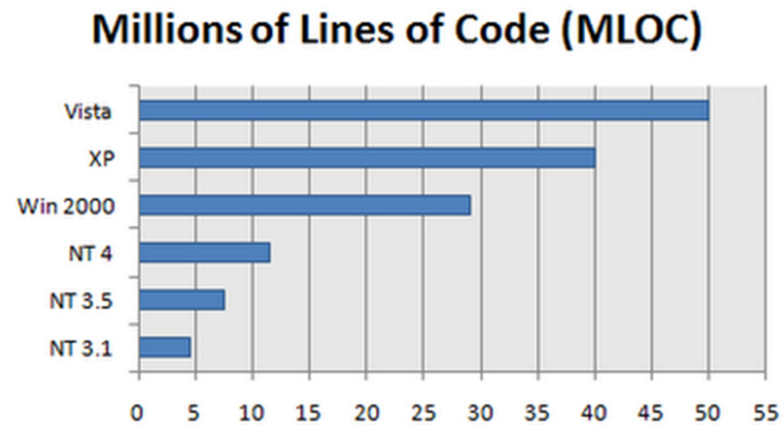
https://www.usa.philips.com/healthcare/solutions/interventional-xray/allura

Will it bring me to Paris within 3 hours?

https://en.wikipedia.org/wiki/Fyra

(Embedded) Software complexity

- Measured in MLOC
  - Million lines of code. Beware, LOC is a dubious measure
- Conventional wisdom:
  – # faults is linearly related to LOC
  – ~ 3K faults / MLOC.
- Some system sizes
  – WoW: > 5 MLOC
  – ASML wafer scanner: > 35 MLOC (2011)
  – LHC: > 50 MLOC, ergo more faults than detected hadrons!
- Google code base
  - > 2 GLOC
  - Comm. of the ACM (July 2016)
- www.informationisbeautiful.net



- Windows 7, 8: Who knows?
- Linux: 15 MLOC (2011), 20 MLOC(2015)

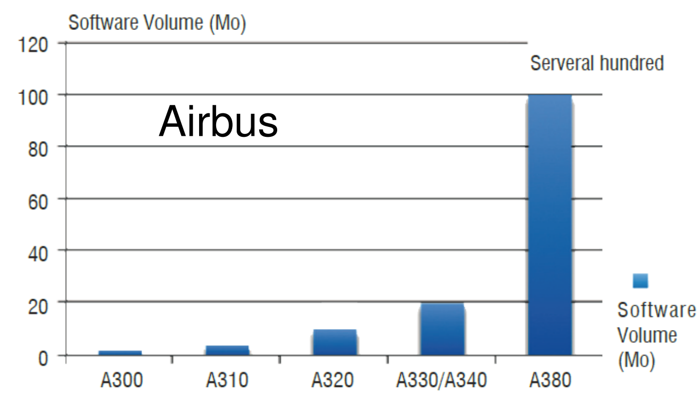


Figure 2 - Evolution of the volume of embedded software

Modern cars

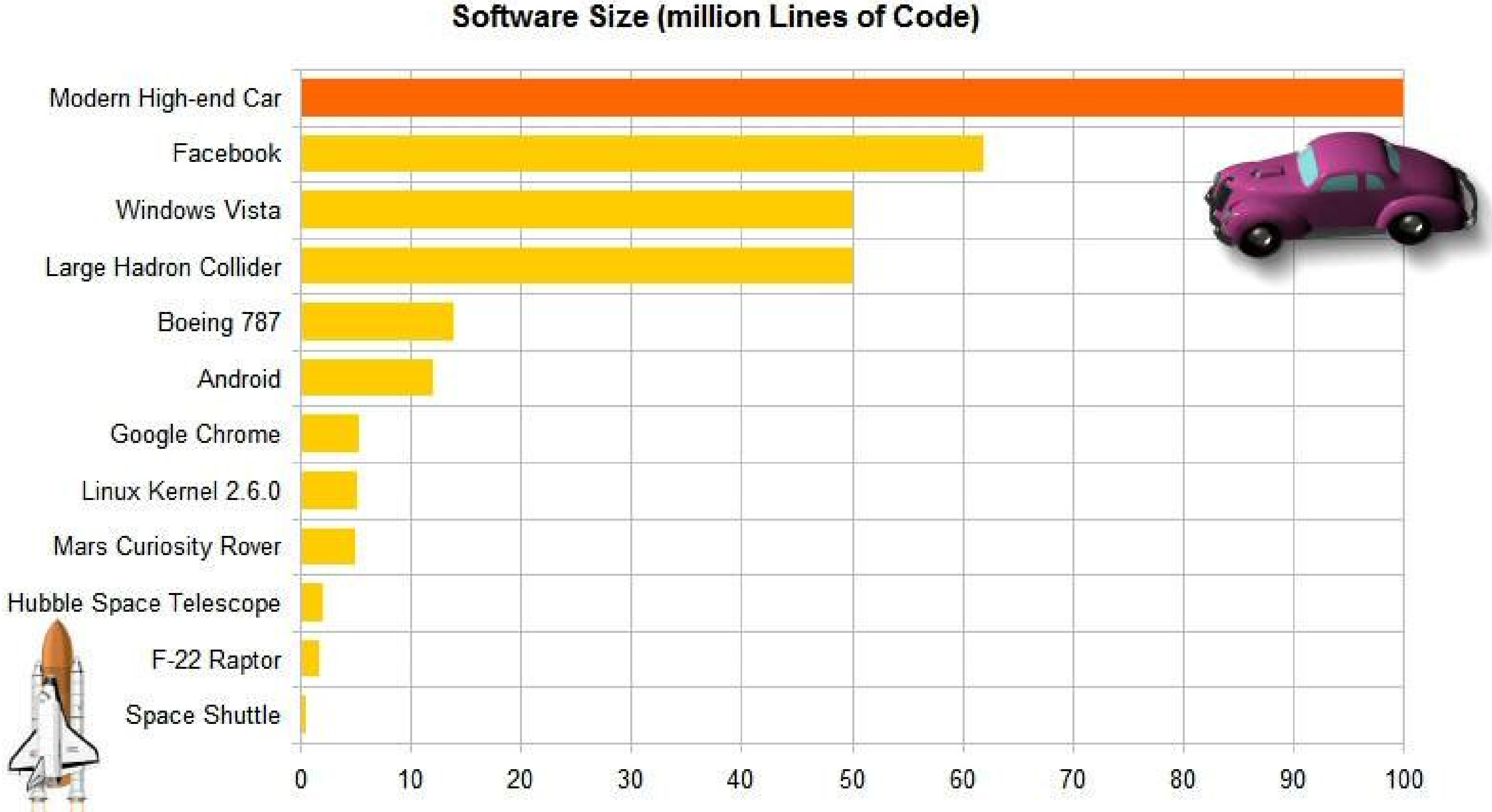
- What's Got 10M Lines of Code & An IP Address?
  - The Volt (http://gigaom.com/cleantech/whats-got-10m-lines-of-code-an-ip-address-the-volt)

- IEEE Magazine, January 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

(Embedded) Software Complexity



Purpose of the architecture

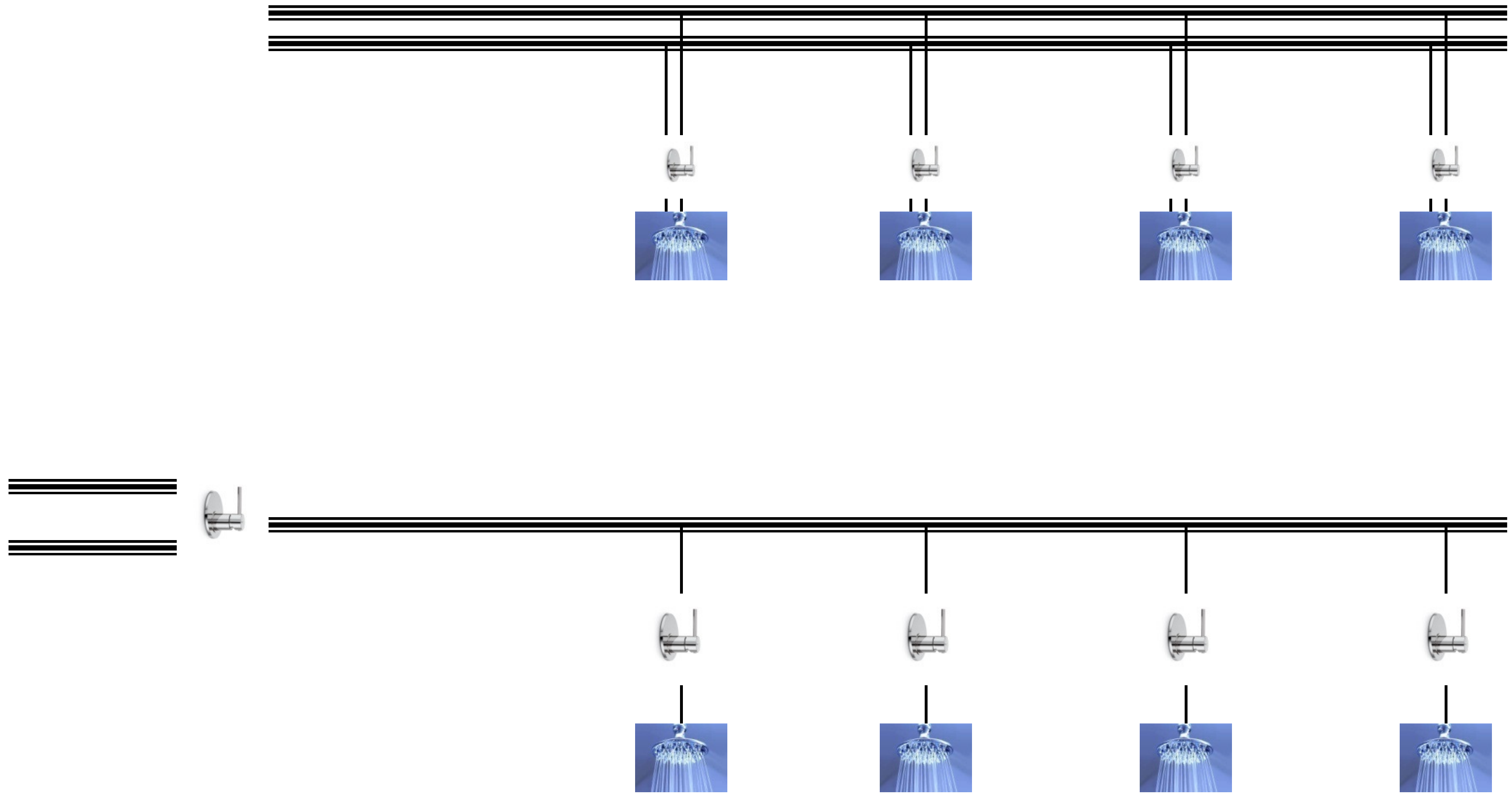
- Communication / Understanding
  - To define global meaning and scope of the system
  - With stakeholders to see whether the right system is made
  - With (teams) of developers to guide the construction
  - With users / customers to provide insight how to use the system
- Analysis
  - In general to answer questions about the system
  - To evaluate candidate architectures on an abstract level
    - W.r.t. all kinds of quality attributes
  - To make design decisions that have significant impact on the cost and performance of the system
- Construction (Synthesis)
  - Floorplan / blueprint for the overall structure
    - Complete, but at a high level of abstraction
  - Basis for detailed design by developers
    - Indication of used technologies / of-the-shelf components

Design problem

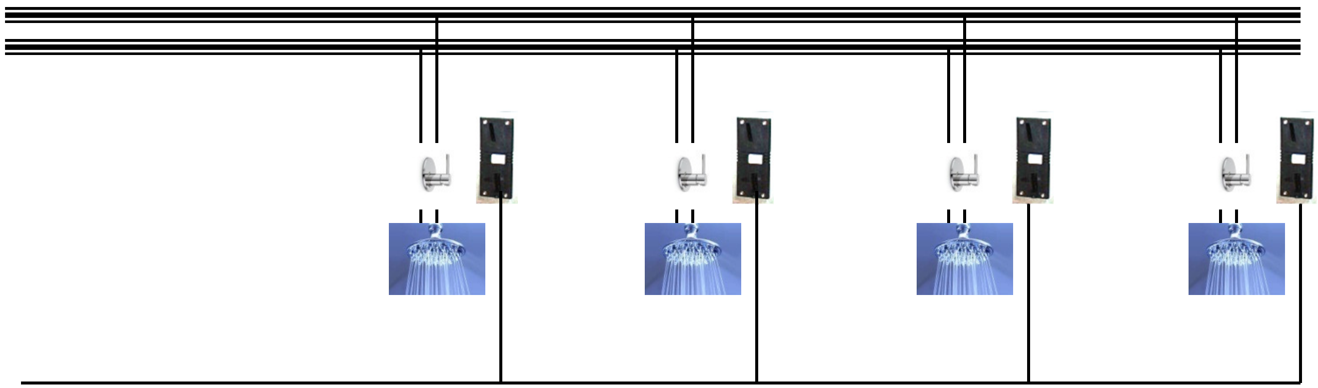
- Realize a *product design* from *building blocks* and *connectors (combinators)* satisfying *functional requirements...*
  - *functional requirements*: captured via *use cases*
  - *use cases:* interactions with the product
  - *product*: building blocks + connectors
  - *design*: drawings, blue prints
- ...subject to *boundary conditions* and *quality constraints...*
  - *rules (constraints)* for building blocks and connectors
  - *extra-functional* properties (perspectives)
    - *'...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, user,

Example: shower control system

Two alternatives



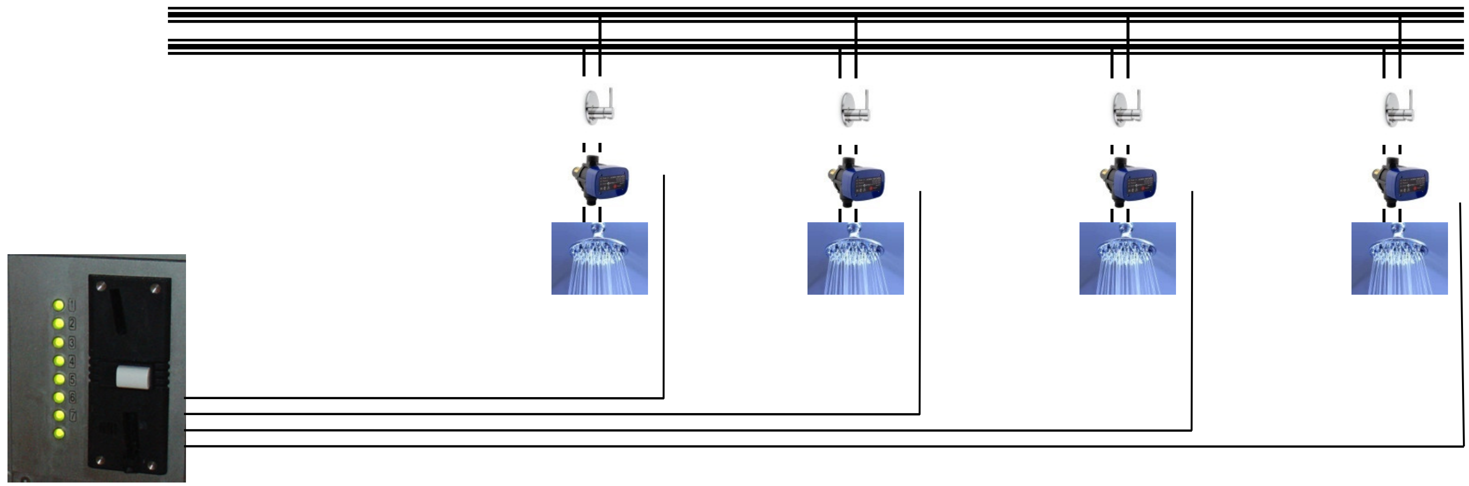
It’s not for free



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

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

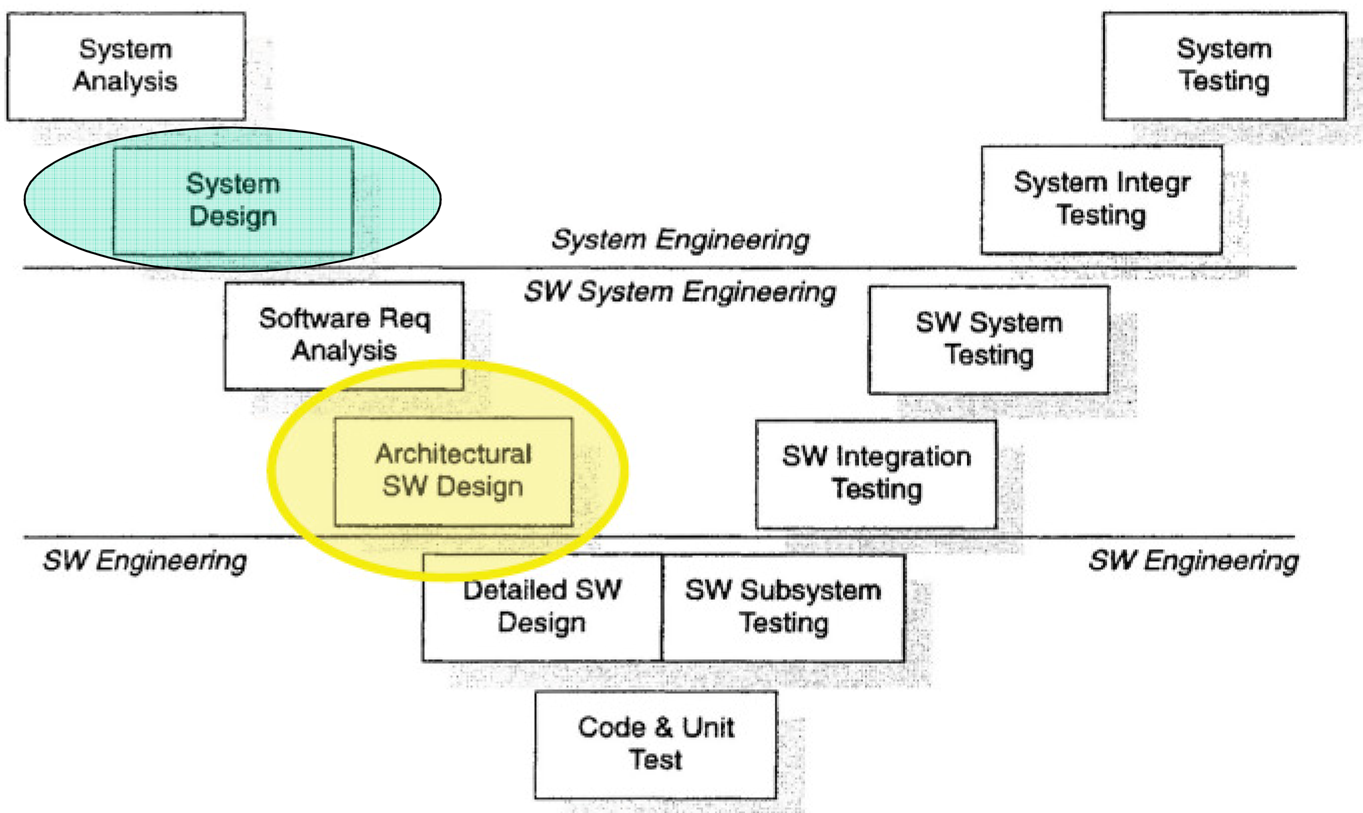
- The real (built) system:
  – schedules shower assignment
    - makes a prolonged shower embarassing
  – uses a combined maximum on time and water used
    - changes this architecture
    - probably more expensive
  – is a COTS system (Commercial Off-The-Shelf)
    - cheaper, known by (installation)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

Design problem: design shower system

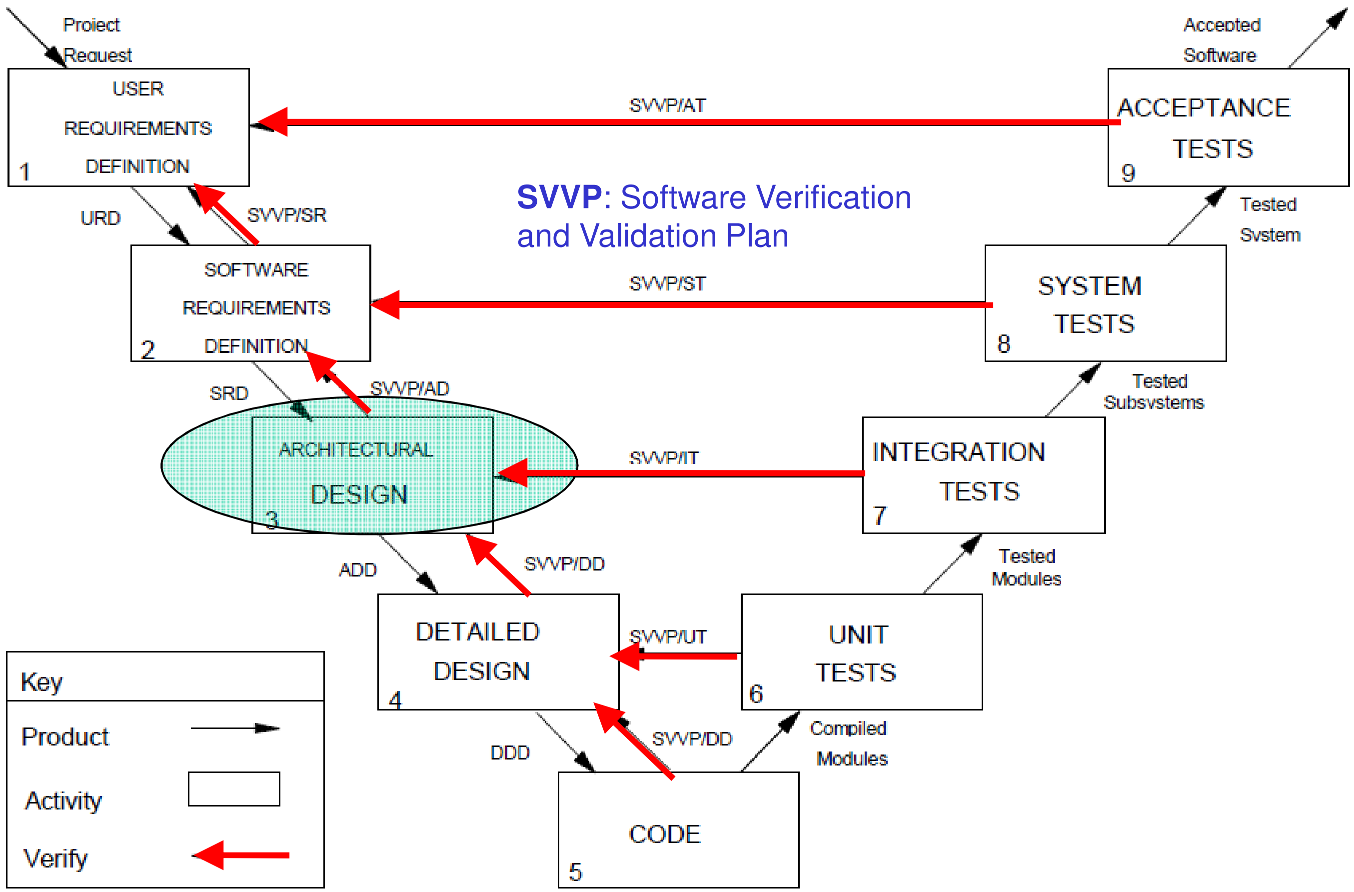
- Functional requirements:
  - shower functionality – use cases
  - cleaning functionality – use cases
  - ....
- *Building blocks*: money insert/controlling device, shower heads, switches
- *Connectors*: pipes and wires
- Constraints:
  - building block choices (perhaps), brands, regulations, ...
- 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 (concerns) and use cases*

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



Software Architecture
place in software development cycle



Esa standard: life cycle verification approach

Design *process*

- 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

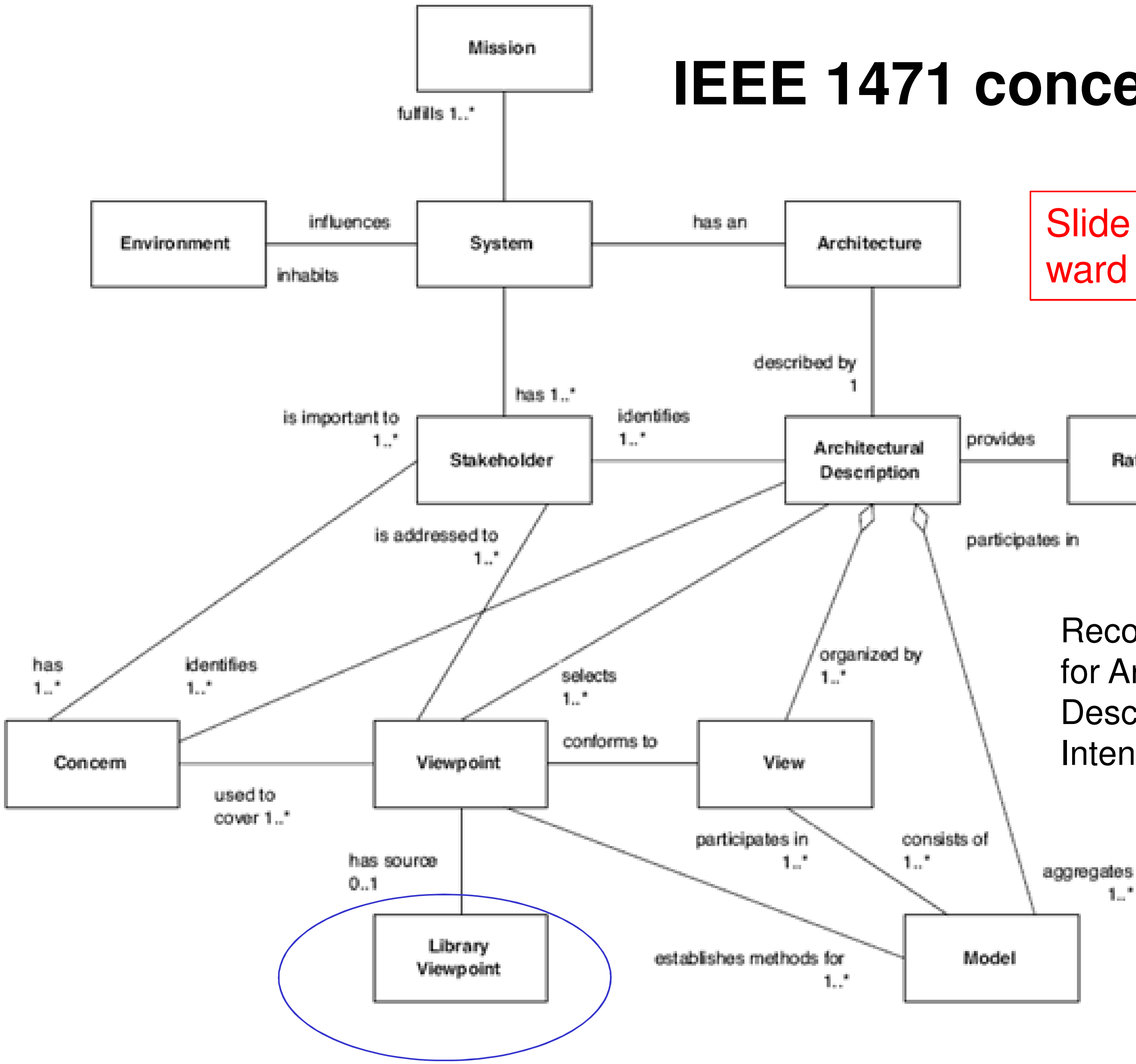
- (Domain) analysis
  - increase knowledge, make models
    - use cases, based on stakeholder viewpoints
  - feedback to stakeholders:
    - validation of requirements (“Do we solve the right problem?”)
- Apply strategies
  - hierarchical decomposition:
    - top-down (factorization): *specify* advanced 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?”)

History of Architecture Description

- 1995: “4+1” paper by Kruchten
  - Makes the usage of multiple views widely accepted
  - Advocates 4 specific views, plus scenarios to address system behavior
- 2000: IEEE Standard 1471
  - Presents a formal conceptual model for architectural descriptions (ADs) that standardizes terminology
  - Distinguishes between viewpoints and views
    - Kruchten’s views are in fact viewpoints
- 2011: New ISO/IEC IEEE standard 42010
  - Extension of 1471.
  - Introduces the notion of *architecture frameworks* and their support by means of *architectural description languages*

IEEE 1471 conceptual model

Slide included for backward compatibility



Recommended practice for Architecture Description of Software-Intensive Systems

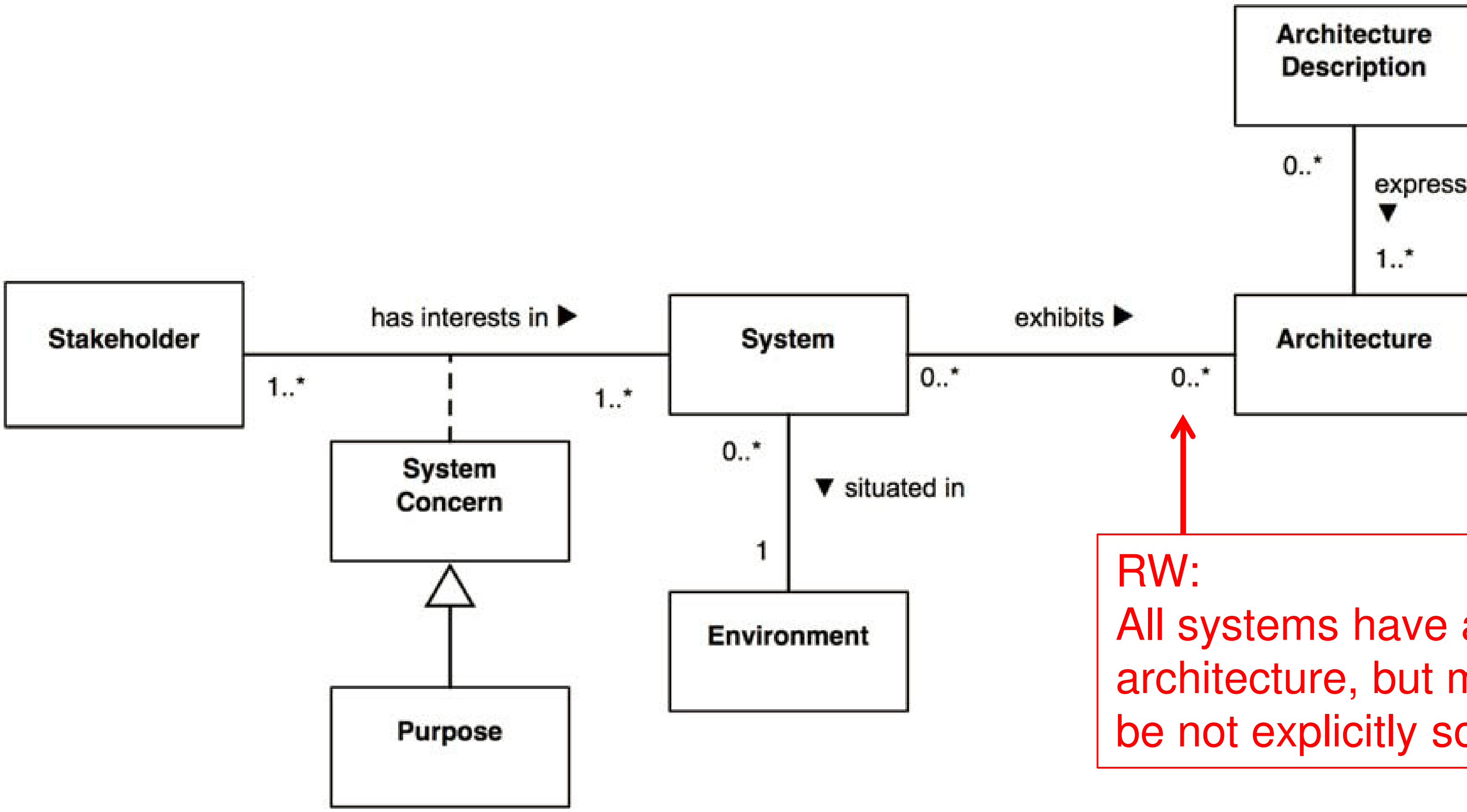
ISO/IEC/IEEE 42010 standard

- Revision/update of IEEE 1471
- Provides a core ontology for the description of architect-ures.
- Specifies the organization of these descriptions using
  - Architecture viewpoints, Architecture frameworks and Architecture description languages
- Defines and motivates concepts and presents their relationship in a set of (meta-)models
  - Architecture context model
  - Architecture description model
  - Architecture decision and rationale model
  - Architecture framework model.
  - Architecture description language model

ISO/IEC/IEEE 42010: Concepts

- Most concepts have already been defined in the IEEE 1471 standard (see previous slide) and their definition remains largely the same. The most important ones are:
  - Architecture, architecture description
  - Viewpoint, view,
  - Architecture model
  - Stakeholder, concern
- New concepts are
  - Architecture rationale
  - Architecture framework
  - Architecture description language

Architecture context model



RW:
All systems have an architecture, but may be not explicitly so.

Quiz

Taken from:
https://upload.wikimedia.org/wikipedia/commons/c/c8/De_Rat_molen_IJlst_25.JPG

- What is the system?
- What is its purpose?
- What is, *or are important aspects of*, its environment?
- What about the other entities from the context model?

Architecture (42010 definition)

The architecture of the system is

- the set of *fundamental concepts* or properties of a system in its *environment* embodied in its *elements, relationships*, and in the *principles of its design and evolution*

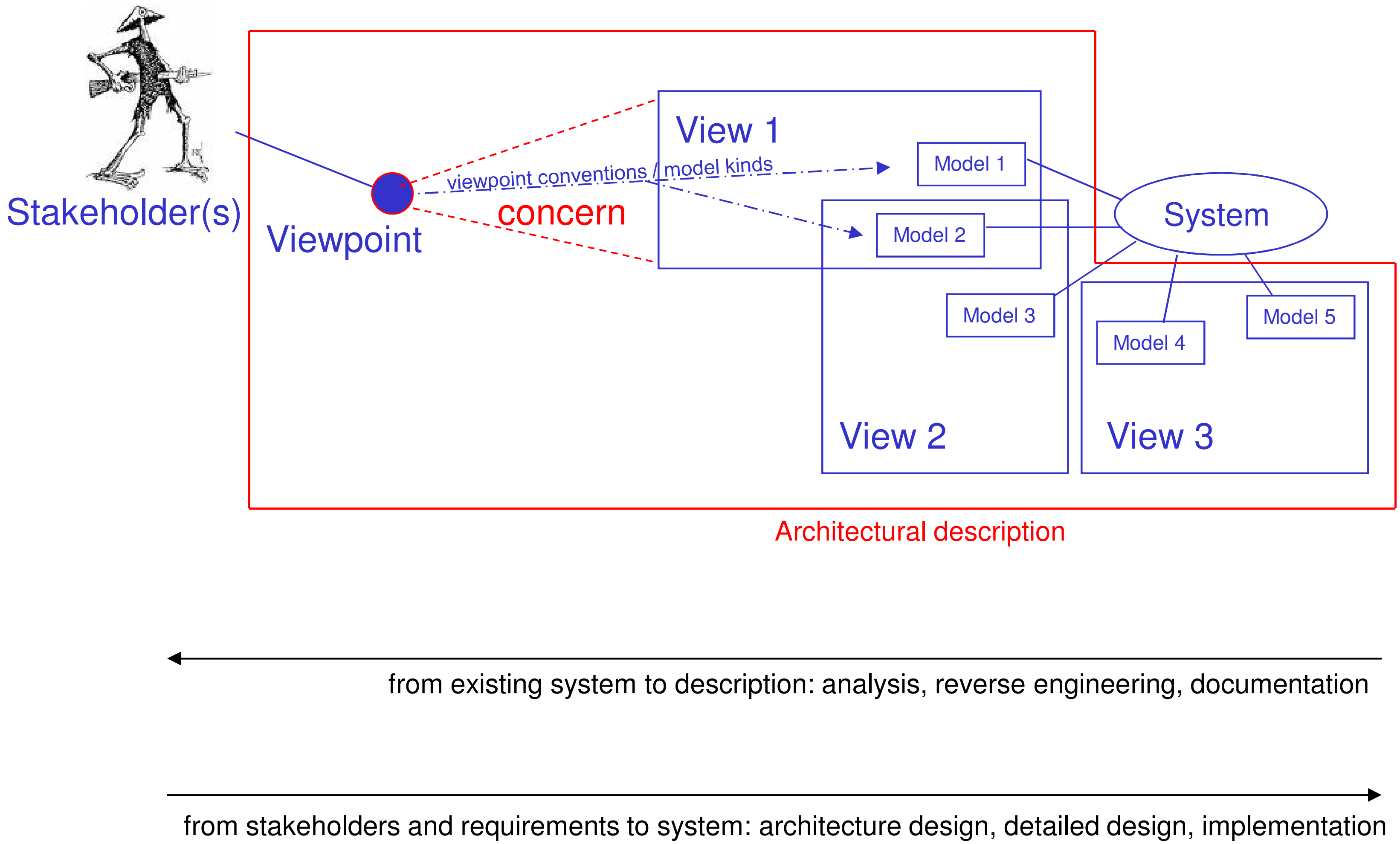
1. Architectures are created solely for the benefit of stakeholders
   - A *good* architecture is one that successfully addresses the concerns of its stakeholders and, when those concerns are in conflict, balances them in a way that is acceptable to the stakeholders
2. Every system has an architecture, but it may be implicit.
3. An architectural description makes an architecture explicit

Architecture description

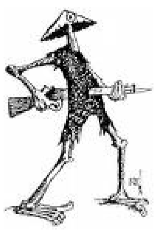
An architecture is described by a collection of models/blueprints

- 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' AD):
  - 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

Overall picture



Example

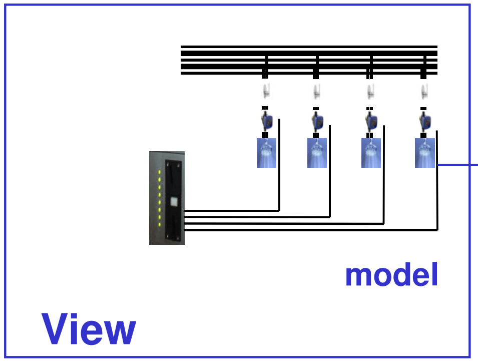


Plumber, electrician

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
- use cases, leading to scenarios in the views:
  - person taking shower, person collecting money, maintenance, installation

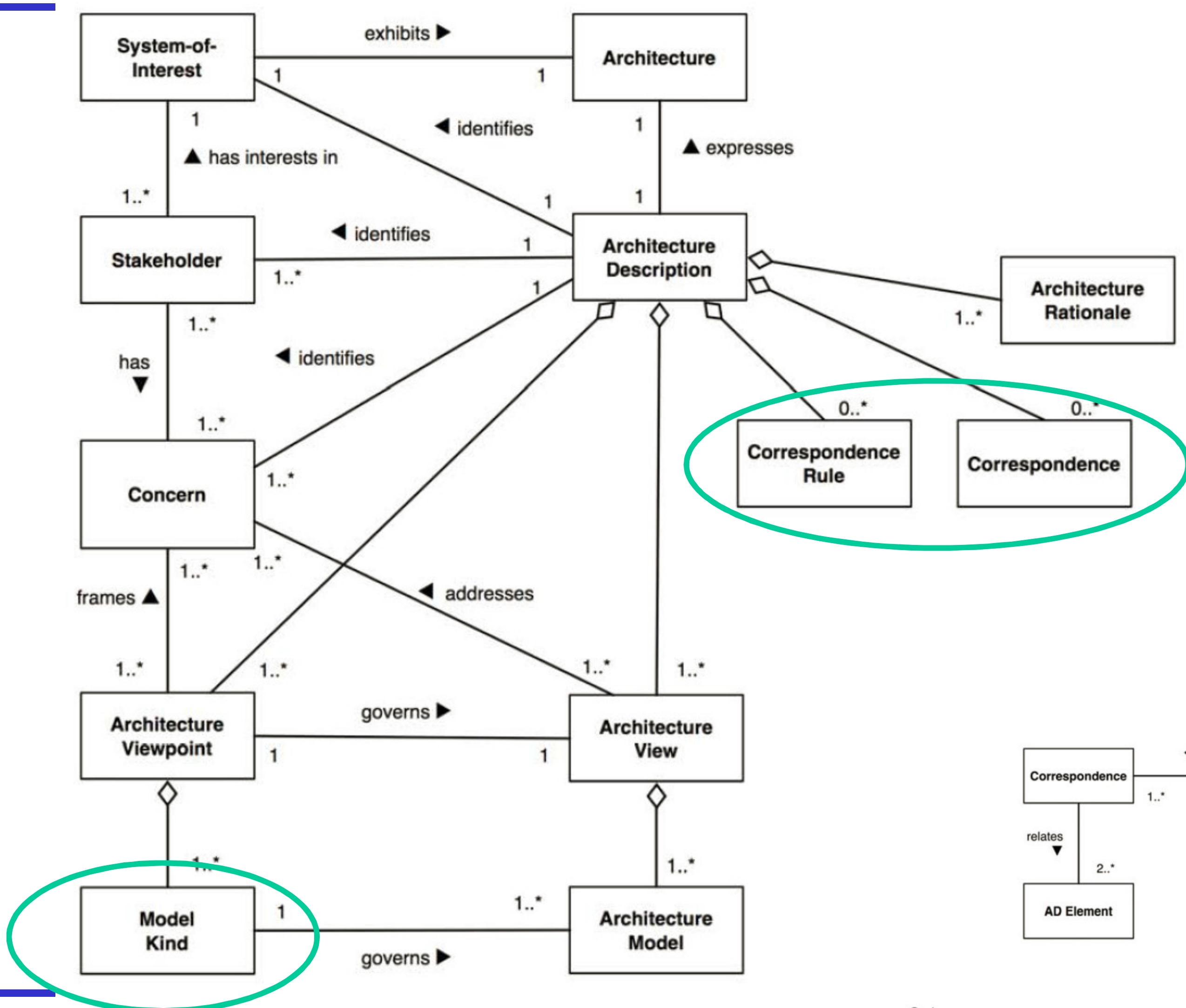
Crosscutting concerns (appearing in multiple viewpoints):

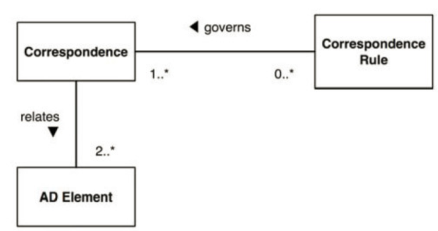
- quality properties & evaluation (concerns of several stakeholders)
  - *system properties*: pipe sizes, pressure, heater capacity
  - *metrics*: water spending, cost, user experience

Possible hierarchy

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

Architecture description model





Rozanski-Woods: Stakeholders

- Definition
  - A person, group, or entity with an interest in or concern about the realization of the architecture.
- Criteria
  - Stakeholders should be:
    informed, committed, authorized, representative.
- Responsibilities
  - Communicate concerns (theirs or those of the group they represent) to the architect.
  - Review the architectural description (AD).
  - Decide on the acceptance of the architecture and other issues about the system.

Beware. It are the roles of stakeholders that are important not their identities. The same person can hold multiple roles.

RW: Stakeholder role classification

Class	Description
Acquirers	Oversee the procurement of the system
Assessors	Oversee conformance to standards and legal regulations
Communicators	Explain the system via documentation and training
Developers	Construct and deploy the system from its specifications
Maintainers	Manage the evolution of the system once operational
Suppliers	Provide hardware/software platform on which the system will run
Support staff	Assist users to make use of the running system
System administrators	Run the system once deployed
Testers	Check whether the system meets its specifications both functional and extra-functional
Users	Define the system's functionality and use it once running

Models and views

- **Model**: abstract, simplified (partial) representation of a system
  – leaving out details irrelevant to a given set of criteria ("concerns")
  – while preserving the properties of interest with respect to those concerns
  – can be used to answer a set of questions about the system
  – a single model is often insufficient to describe a complex system

- **View:** a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses some concerns of some stakeholders
  – 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 laid-down in the model kinds of the viewpoint and represents what you 'see' from that viewpoint

Viewpoints (42010 definition)

A *viewpoint* is a

- collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines and principles, and template models for constructing its views

A *viewpoint* is characterized by

- a set of **concerns** to be addressed
- a set of **stakeholders** interested in how they are addressed
- one or more **model kinds**
- the **conventions**: concepts, notations, rules, patterns, styles and semantics to be invoked in creating, interpreting and using models of each kind;
- **correspondence rules** linking the models together.

Viewpoints libraries

Viewpoints are generic, hence they can be collected in viewpoint catalogs (or libraries) and reused.

Examples

- Kruchten 4 +1 views
  - Beware Kruchten's views are viewpoints in 42010 parlance
- Rozanski-Woods
- ISO/IEC 42010 Viewpoints Repository [42]

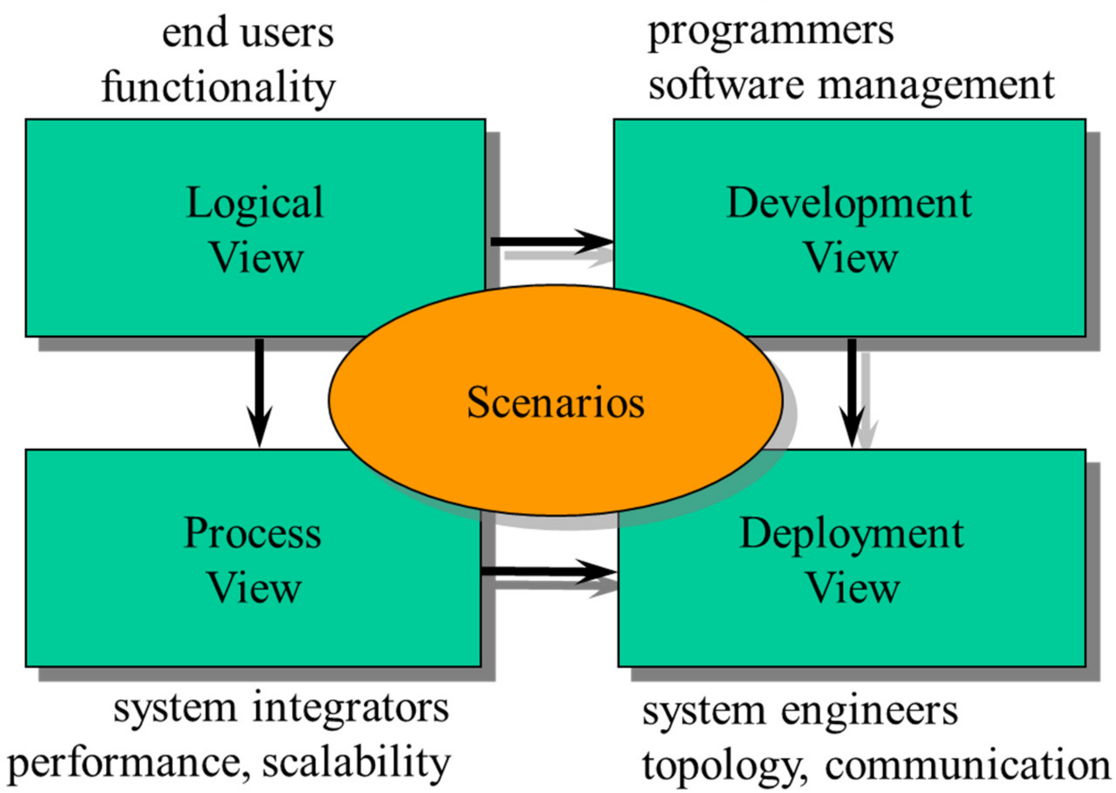
Kruchten's views and stakeholders

Article: Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12(6), Nov. 1995, pp 42—50.

Stakeholder	View	Concerns
User	Logical	Using the system; associated qualities • Externally visible structure and functionality
Programmer	Development	Implementing/modifying the system • Decomposition into subsystems • Organization into files, components and modules
System Integrator	Process	Performance aspects, interaction between parts • Units of deployment (programs, components) and concurrency (processes, threads)
System Engineer	Deployment Physical	Installing/deploying/realizing the system • Computers, networks, infrastructure, protocols, distribution, mapping of software to hardware
	Scenarios	Sets of interactions with or within the systems, integrating (the models in) the views and providing behavioral models within the views

Kruchten views (viewpoints)

A number of standard (library) views



4+1 View Model [Kruchten 95]

RW: Viewpoint catalog

- Context viewpoint
- Functional viewpoint
- (analogous to Kruchten's Logical view)
- Information viewpoint
- Concurrency viewpoint
- (analogous to Kruchten's Process view)
- Development viewpoint
- Deployment viewpoint
- Operational viewpoint

See also http://www.viewpoints-and-perspectives.info/home/viewpoints/

RW: viewpoint library (1 of 4)

- The context viewpoint
  - describes the relationships, dependencies, and interactions between the system and its environment (people, systems, external entities).
  - is relevant to all stakeholders, in particular, acquirers, users and developers.
  - addresses concerns like: system scope and responsibilities; identity and nature of external services; data used; identity, nature and characteristics of external interfaces; impact of the system on its environment; overall completeness, consistency and coherence.

RW: viewpoint library (2 of 4)

- The functional viewpoint
  - describes the system's runtime functional elements and their responsibilities, interfaces and primary inter-actions.
  - is relevant to all stakeholders
  - addresses concerns like: functional capabilities, external interfaces, internal structure and design philosophy.

- The information viewpoint
  - describes the way the architecture stores, manipulates and distributes information.
  - is primarily relevant for users, acquirers, developers and maintainers
  - addresses concerns about information structure, content and flow; data ownership, volume, validness, lifetime, and accessibility; transaction management; recovery; regulations.

RW: viewpoint library (3 of 4)

- The concurrency viewpoint
  - describes the concurrency units of the system, their functionality, and the required coordination
  - is relevant to developers, testers and some administrators
  - addresses concerns like: task structure, inter process communication, state management, synchronization and reentrance, process creation and destruction

- The development viewpoint
  - describes the architecture that supports the development process
  - is relevant to software developers and testers
  - addresses their concerns about module organization, common processing, standardizations of design and testing, code organization and instrumentation.

RW: viewpoint library (4 of 4)

- The deployment viewpoint
  - describes the environment into which the system will be deployed, including dependencies the system has on its run-time environment
  - is relevant for system administrators, developers, testers, communicators and assessors and addresses their concerns about
  - hardware (processing elements, storage elements, and network), third-party software, and technology compatibility.

- The operational viewpoint
  - describes how the system will be operated, administered, and supported when running in its production environment
  - is relevant to system administrators, developers, testers, communicator, and assessors, and addresses their concerns about
  - installation and upgrade, operational monitoring and control, configuration management, resource management.

Extra-functional properties (EFRs)

- 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
  - the real challenge of architecting, not getting the functionality right
- 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)
- Commonly also referred to as non-functional properties (NFRs)

Quality Attributes (a.k.a. '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, Openness, Composability, Interoperability, Integrability, Accountability, Completeness, Conciseness, Correctness, Testability, Traceability, Coherence, Analyzability, Modularity, Reusability, Configurability, Distributeability, Availability, Confidentiality, Integrity, Maintainability, Reliability, Safety, Security, Affordability, Serviceabilility, …

See also: http://www.thomasalspaugh.org/pub/fnd/ility.html

RW: Perspectives

- An (*architectural) perspective* is a collection of architectural activities, tactics and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require considerations across a number of system's architectural views
  - i.e., address cross-cutting concerns
- Perspectives are for QAs, what viewpoints are for views
- RW-perspective catalog (a selection)
  - Security, Performance and scalability, Availability and resilience, Evolution, ...
- See also
  - http://www.viewpoint-and-perspectives.info/home/perspectives

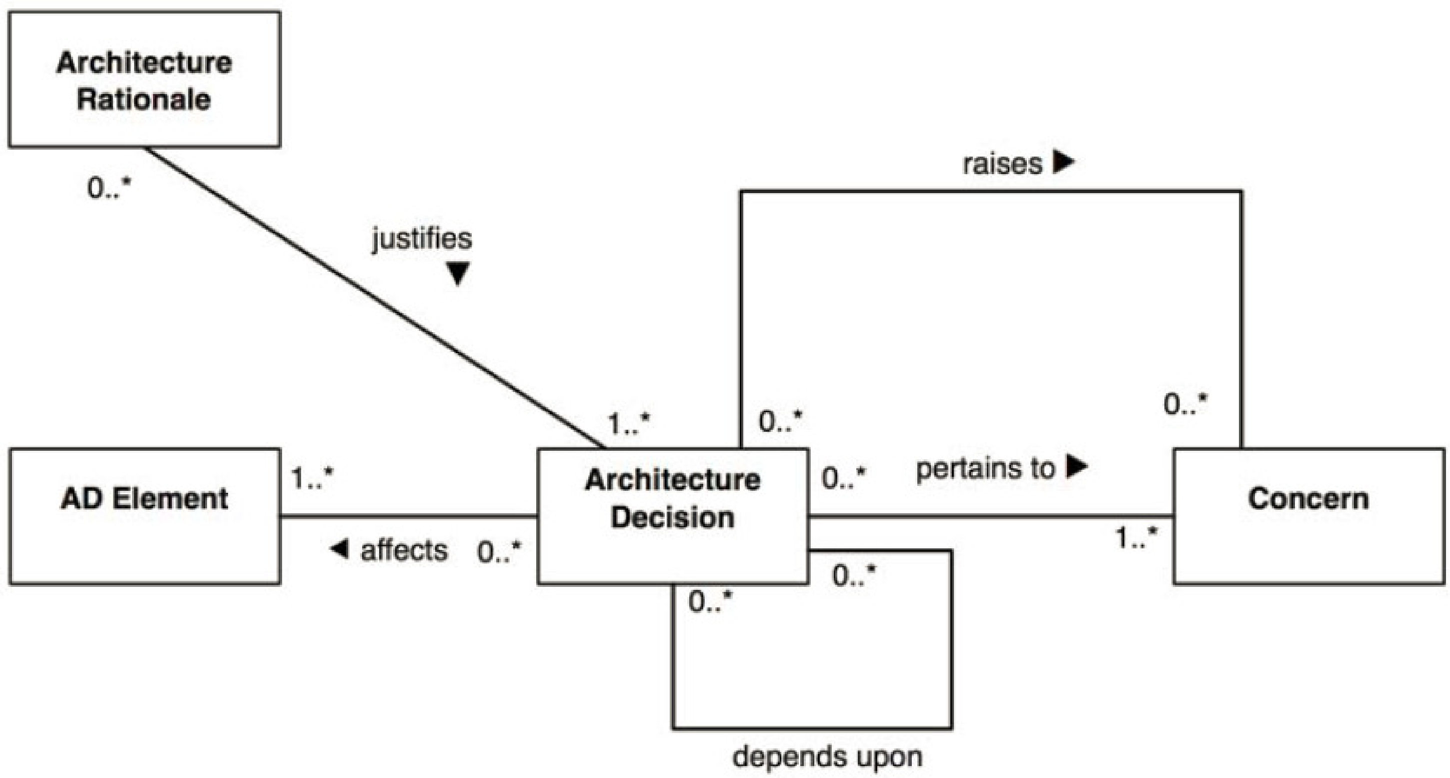
See also http://www.viewpoints-ar

ISO/IEC/IEEE 42010: New concepts

- *Architecture rationale* records explanation, justification or reasoning about architecture decisions that have been made. The rationale for a decision can include the basis for a decision, alternatives and trade-offs considered, potential consequences of the decision and citations to sources of additional information

- Decisions pertain to system concerns; however, there is often no simple mapping between the two. A decision can affect the architecture in several ways. These can be reflected in the architecture description as follows:
  - requiring the existence of AD elements;
  - changing the properties of AD elements;
  - triggering trade-off analyses in which some AD elements, including other decisions and concerns, are revised;
  - raising new concerns.

Architecture decision and rationale model



Quiz

Table 1 Wireless connectivity tradeoffs

	WiFi	ZigBee (802.15.4)	Bluetooth	NFC
Network topology	Star	Mesh	Point-to-point	Point-to-point
Range	30-100 m	10-20 m	10 m	< 0.1 m
Discovery	Broadcast	Broadcast	Broadcast	Response to field
Power	High	Low	Classic: Mid LE/Smart: Low	Tag: Zero Reader: Very low
Privacy	Low	Mid	Mid	High

Taken from: NXP's NFC for embedded application brochure.

- What are the AD elements?
- What are the concerns ?
- Give an example of a system requiring wireless connectivity, state the technology it could use, and a give a rationale for that choice.

Architectural Reasoning-diagram

- Specifies a format for documenting design decisions
- Establishes the rationale for architectural entities (AEs)
  - visible in the top layer
  - necessary for the proper execution of scenarios
  - traceable through a chain of decisions to requirements of particular stakeholders
- Documents alternatives
  - also ones that are rejected

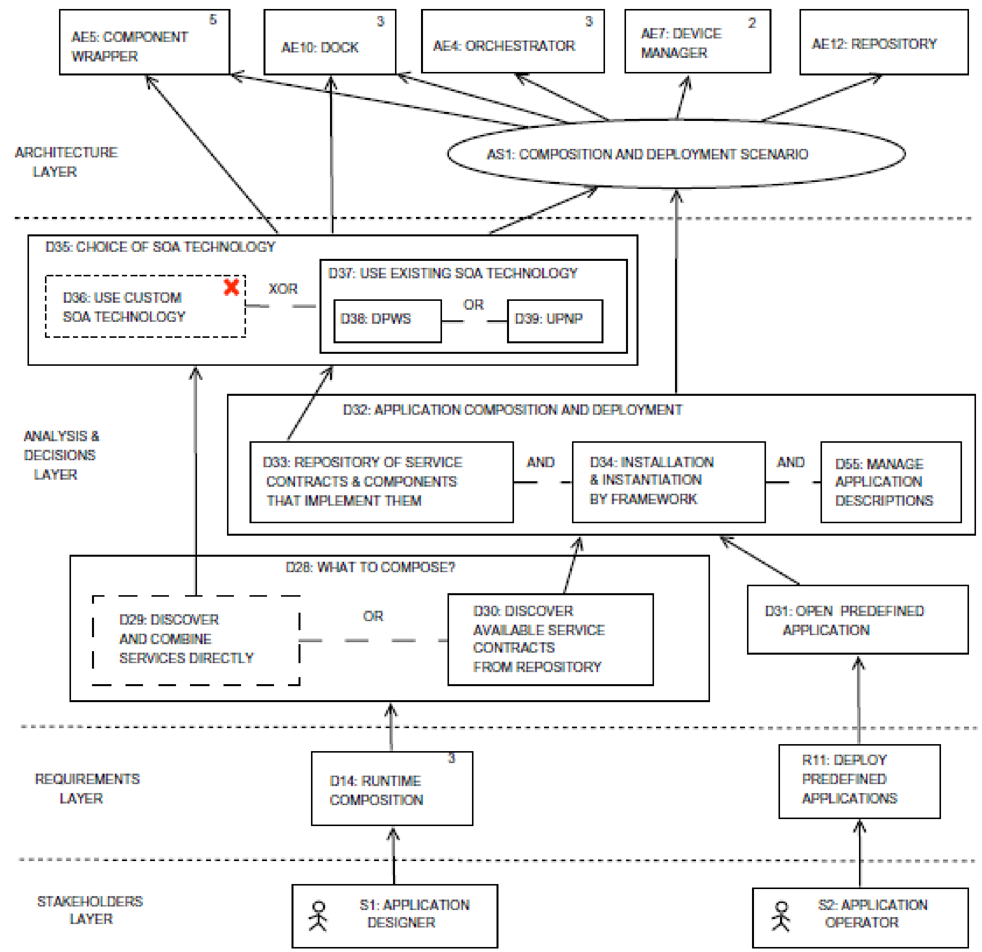
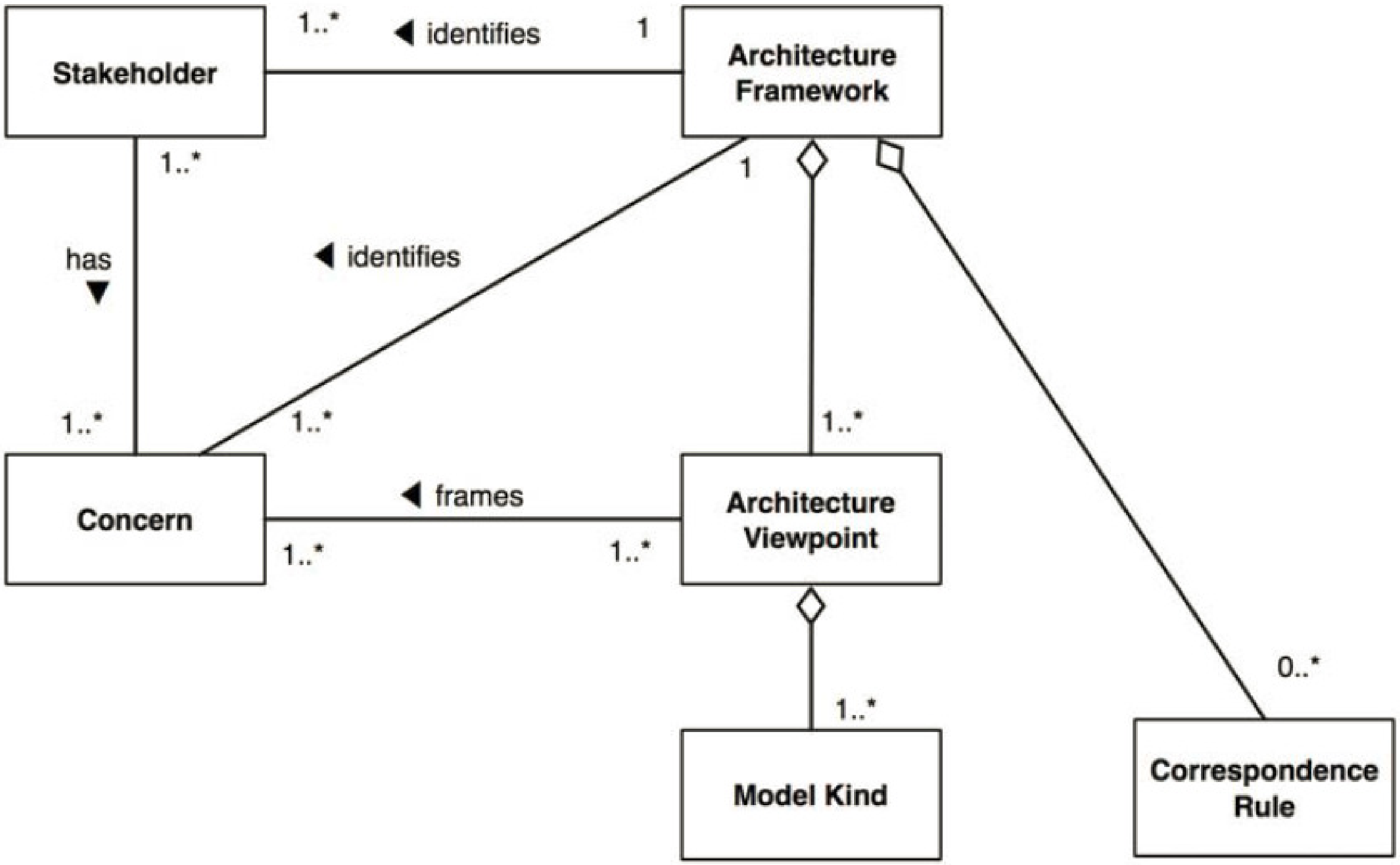


Figure 9: Decisions about usage of SOA technology

ISO/IEC/IEEE 42010: New concepts

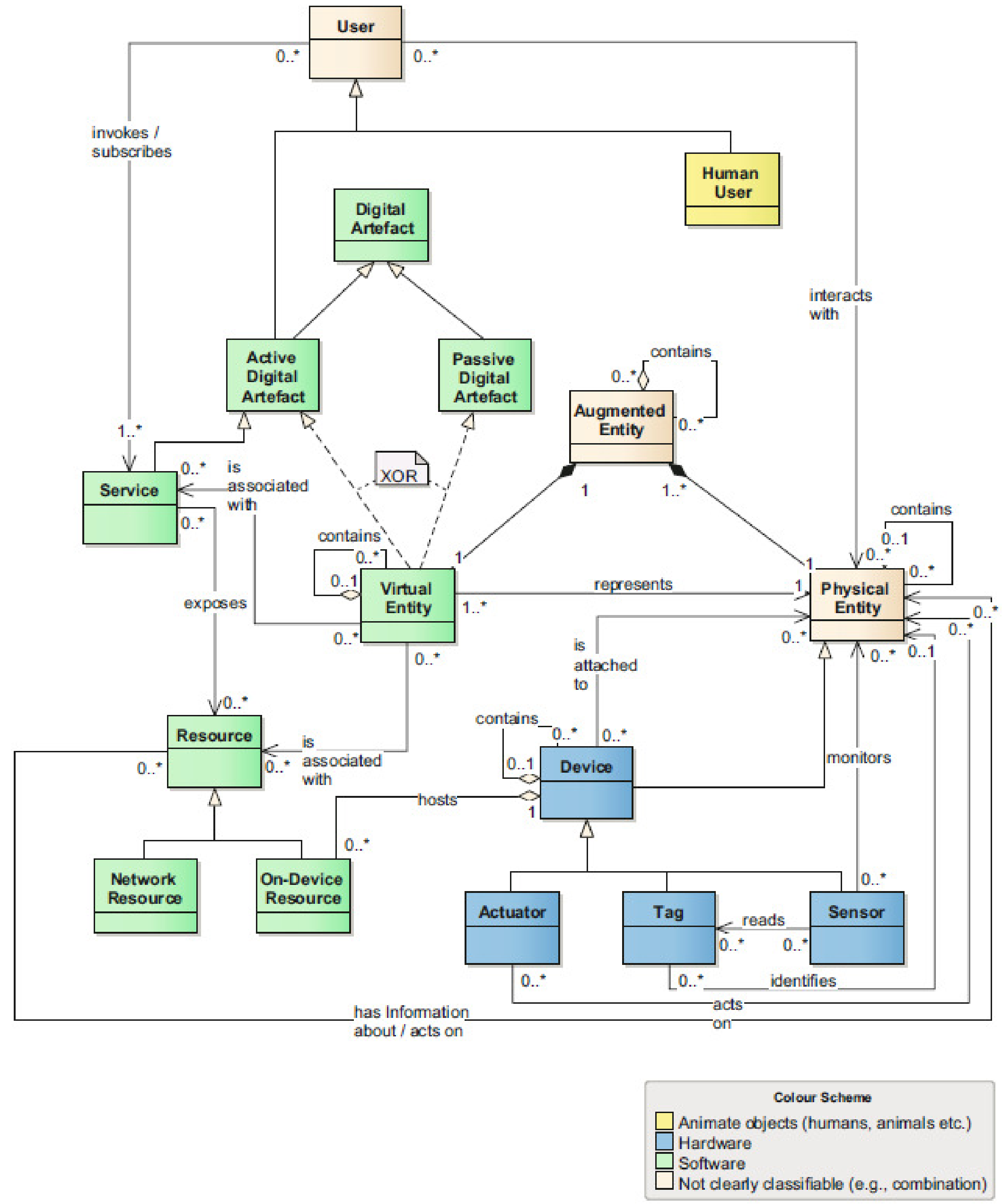
- An *architecture framework* establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community.
- An architecture framework is
  - generic
  - technology independent
  - can be defined through ADL and toolset
- Typical examples are *reference architectures,* e.g., for
  - Service Oriented Architectures
  - the Internet of Things
  - AUTOSAR

Architecture framework model

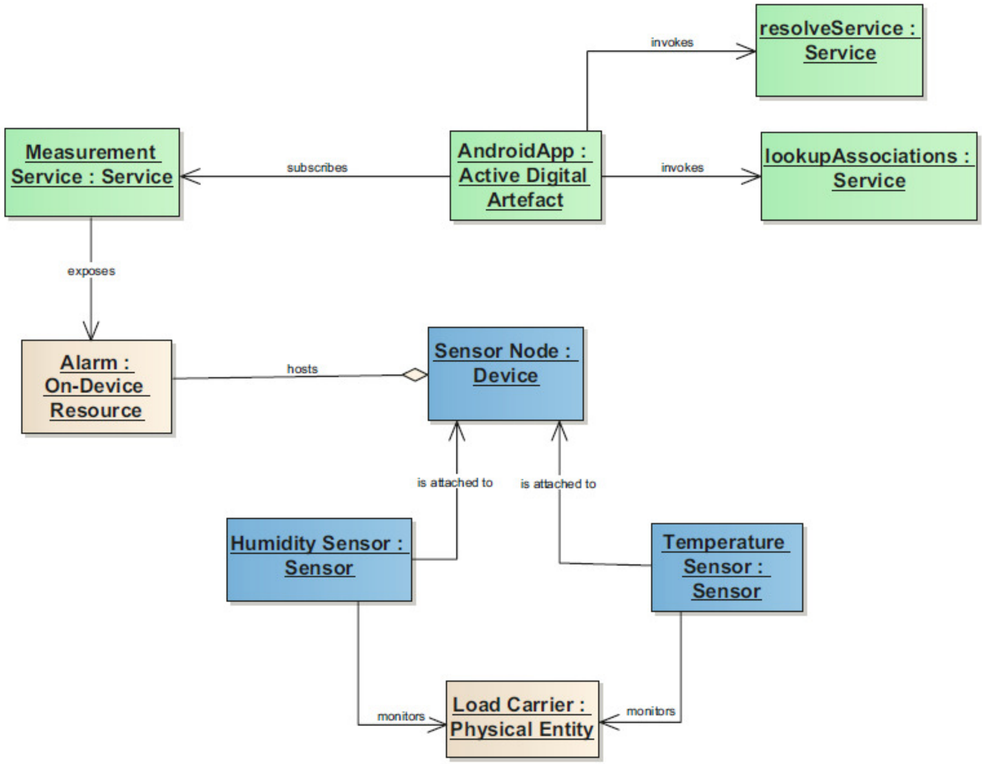


IoT domain model

- Mandatory part of the reference architecture
- Generic
  - within the domain
- Technology indepen-dent
  - such as RFID
- Establishes a vocabu-lary
- Establishes standard practices
  - access of resources via services

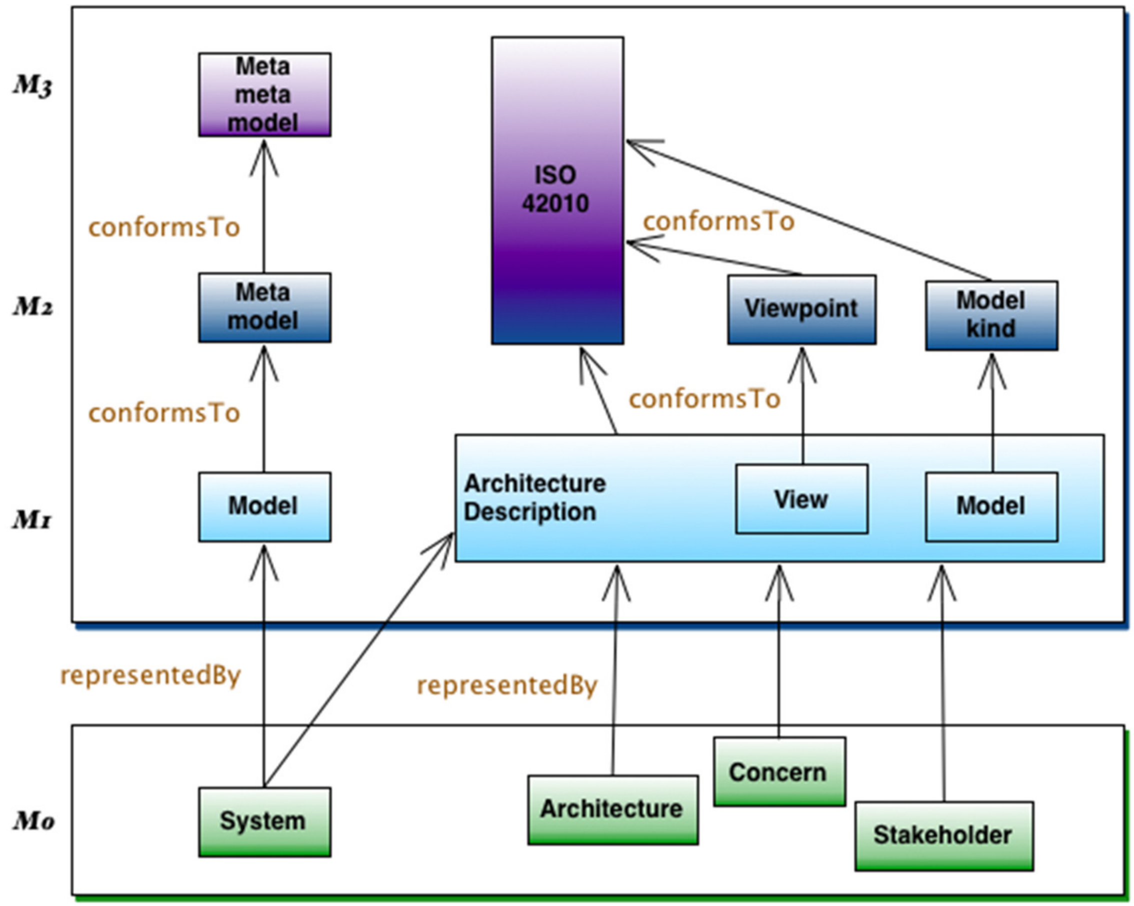


IoT domain model instantiation



Logistics scenario:: monitoring cargo

Models at various levels of abstraction



Taken from: http://www.iso-architecture.org/ieee-1471/meta/

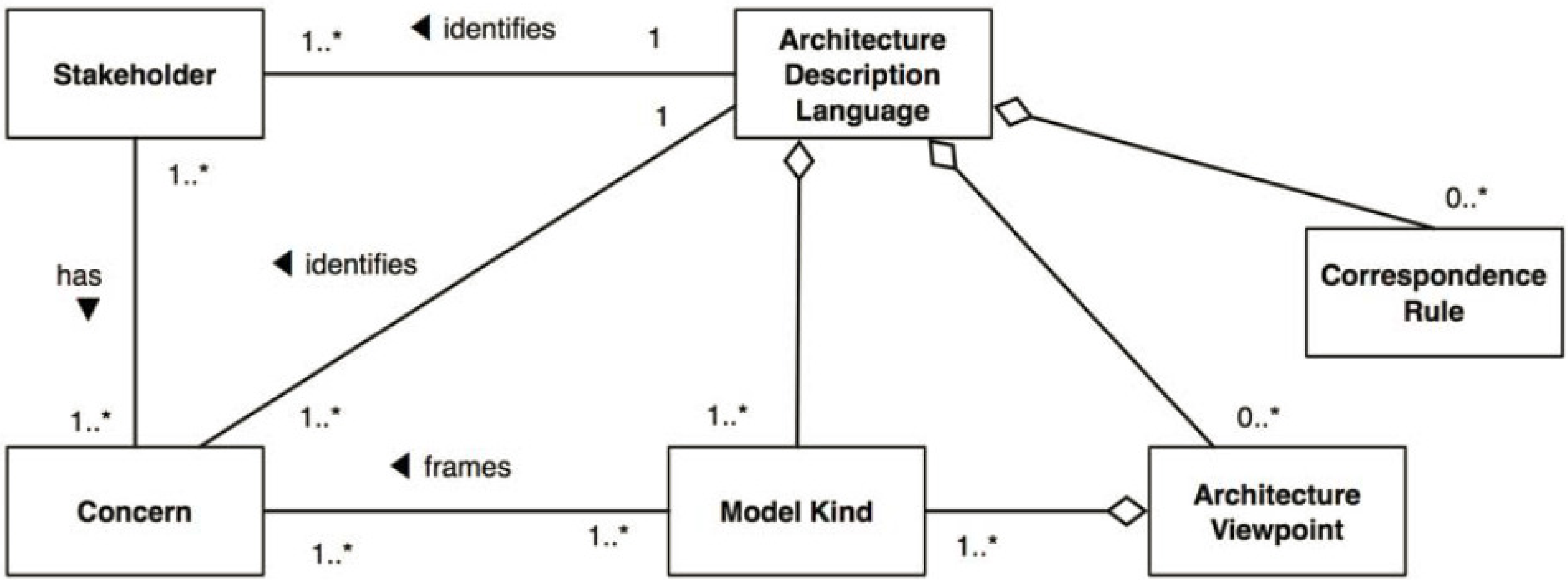
ISO/IEC/IEEE 42010: New concepts

- An *architecture description language* (ADL) is any form of expression for use in architecture descriptions.

- An ADL provides one or more model kinds as a means to frame some concerns for its audience of stakeholders. An ADL can be narrowly focused, defining a single model kind, or widely focused to provide several model kinds, optionally organized into viewpoints. Often an ADL is supported by automated tools to aid the creation, use and analysis of its models.

- Examples: UML, SysML, AADL

Architecture description language model



- Makes explicit which concerns are framed by what model kind
- Model kinds may have associated analysis methods and tools

Twitter example

From presentation Raffi Krikorian at Qcon nyc 2012

Some numbers (outdated by now) to get an idea of the performance and scalability required of the Twitter architecture

Tweet input

- 340 Mtw/day
- daily avg 4 Ktw/sec
- daily peak 6 Ktw/sec
- extreme peaks 10Ktw/sec

Timeline deliveries

- 26G deliveries/day
  - = 18M/min
  - = 300K/sec
- 3,5 sec (@p50) to deliver to 1M

How did they achieve this?

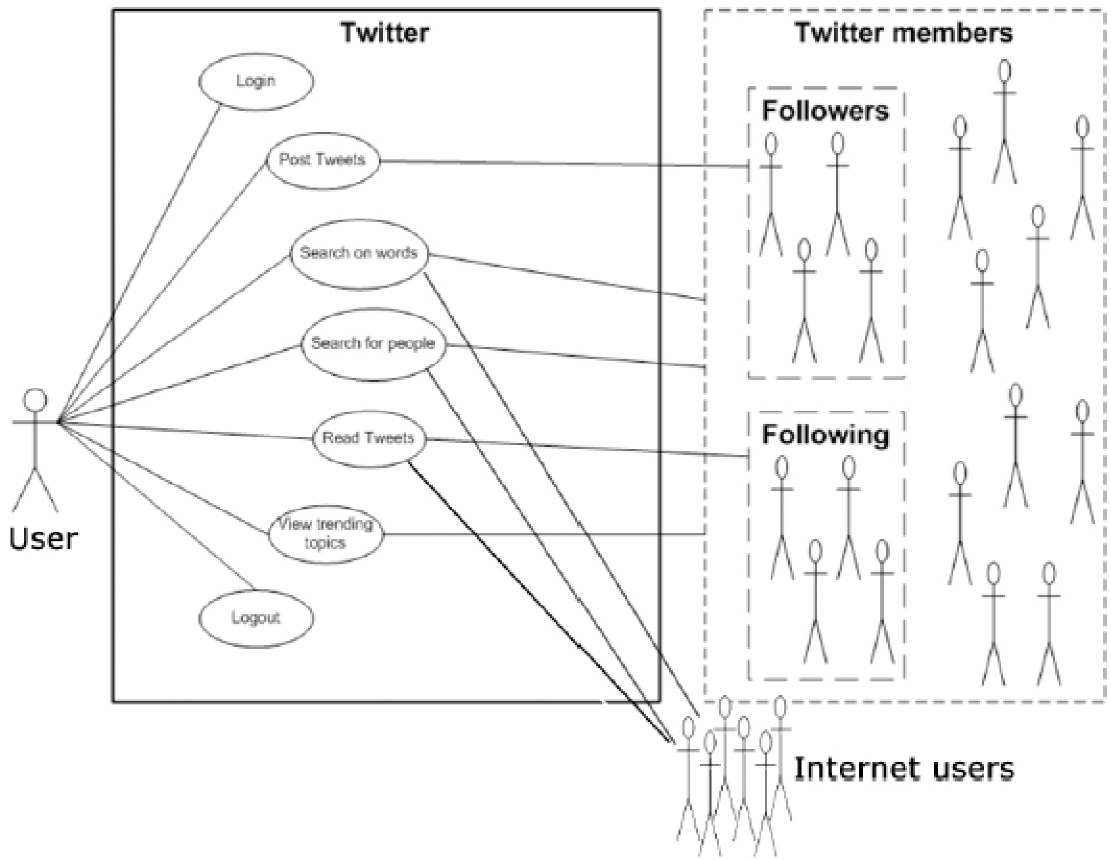
https://blog.twitter.com/2013/new-tweets-per-second-record-and-how

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*
    - *domain concepts and their relationships*
  - 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

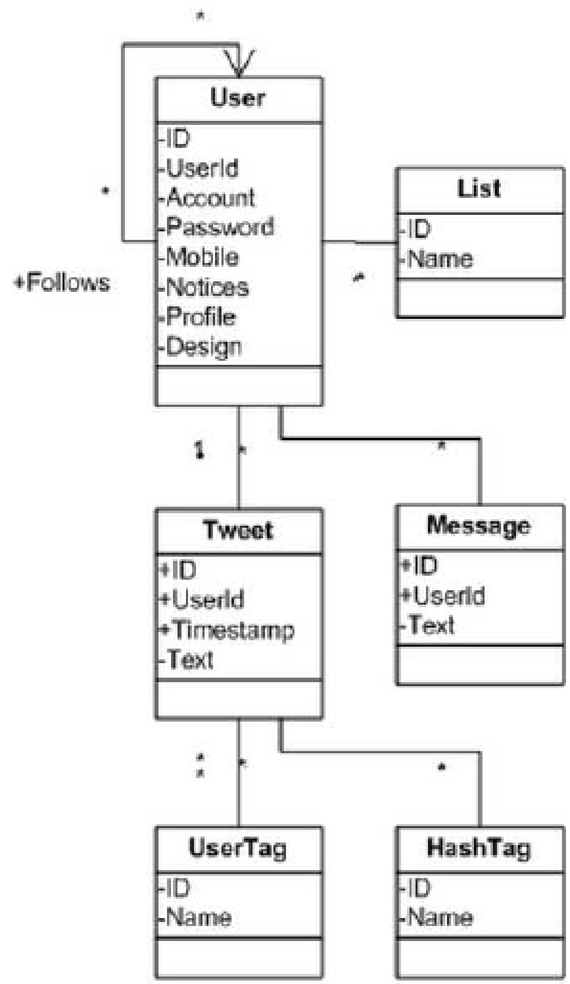
Twitter example: Logical view

- A behavior model (use case) within the logical view
- More models possi-ble; consider, e.g., data miners, (including Twitter itself) that extract information for commercial purposes from the complete stream of tweets (the Twitter Firehose)



Twitter example: Logical view

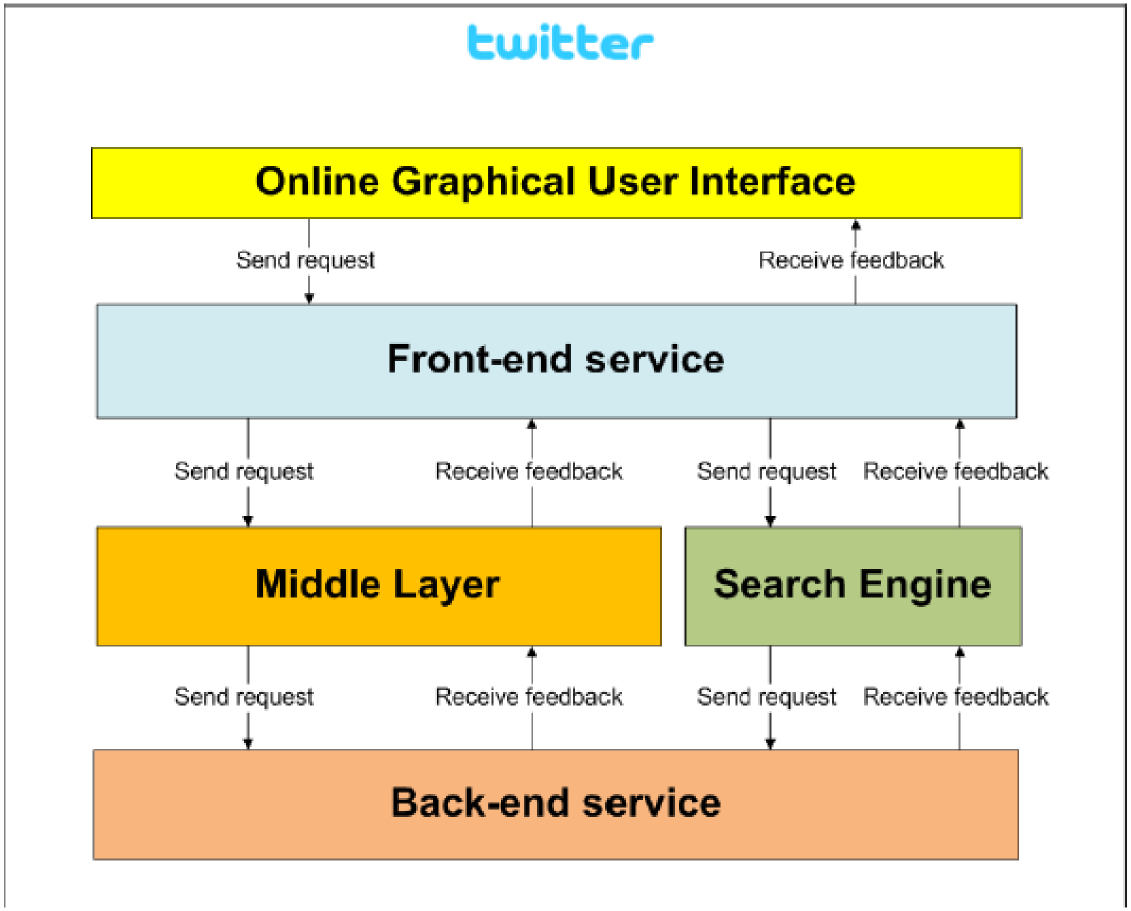
- A domain model in the form of a class diagram that captures important concepts from the application domain and their relationships
- Incomplete
  - E.g. timeline is missing



Development 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

Twitter example: Development view



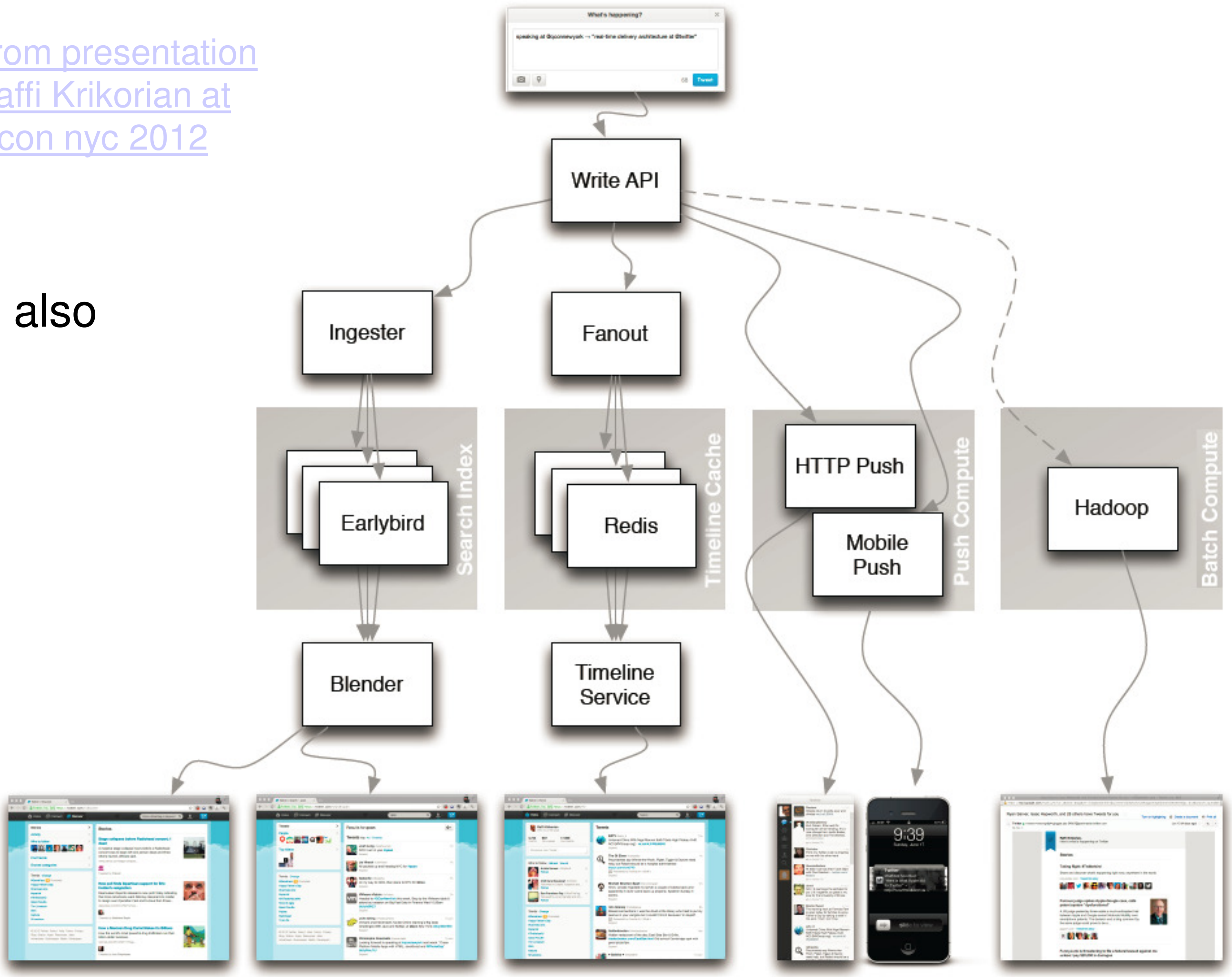
Process view

- 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 performance concerns, mapping of functionality (components) to execution environment and information flow through the system

Twitter example: process view

From presentation Raffi Krikorian at Qcon nyc 2012

See also



Physical / deployment view

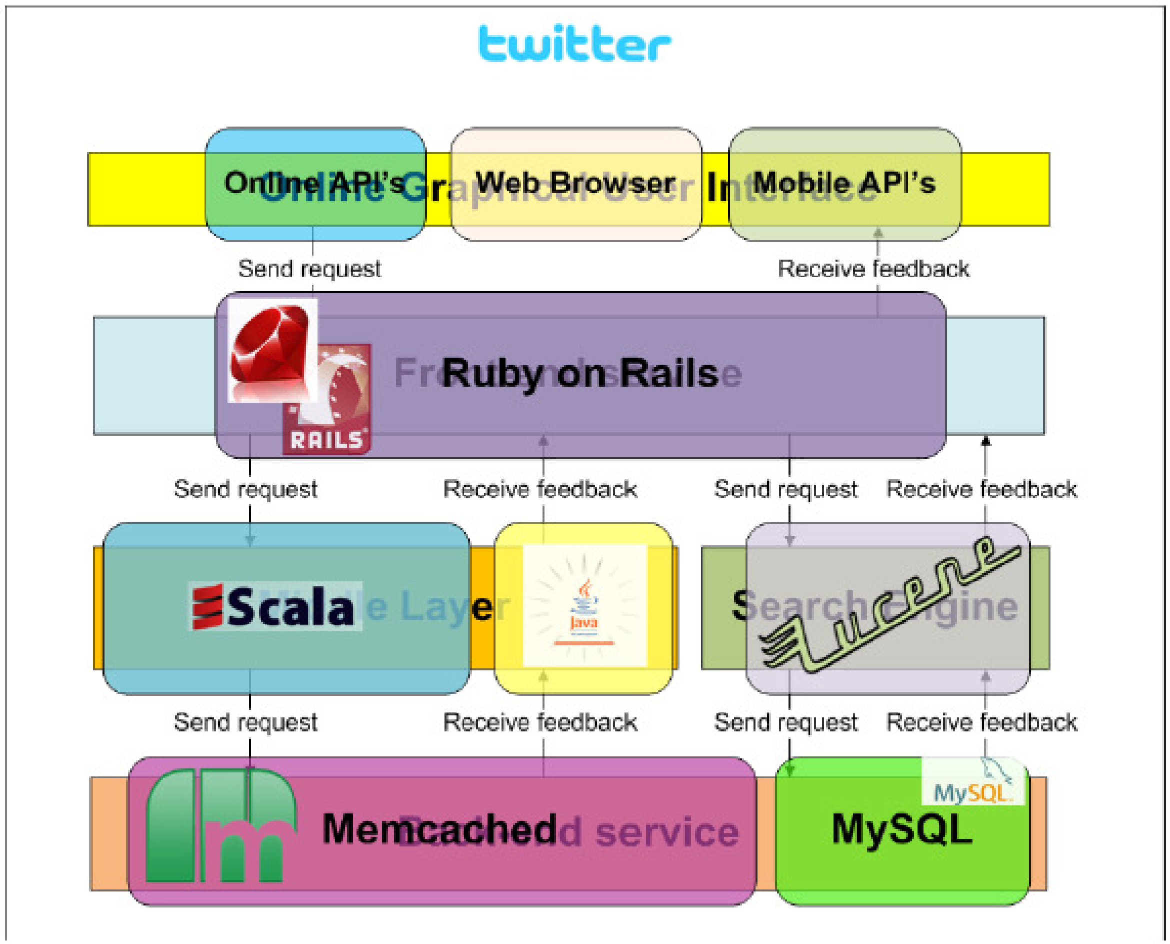
- Machines (processors, memories), networks, organization of interconnect
  - including specifications, e.g. speeds, sizes
  - specific technologies
- Deployment view: mapping of elements of other views to machines

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

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

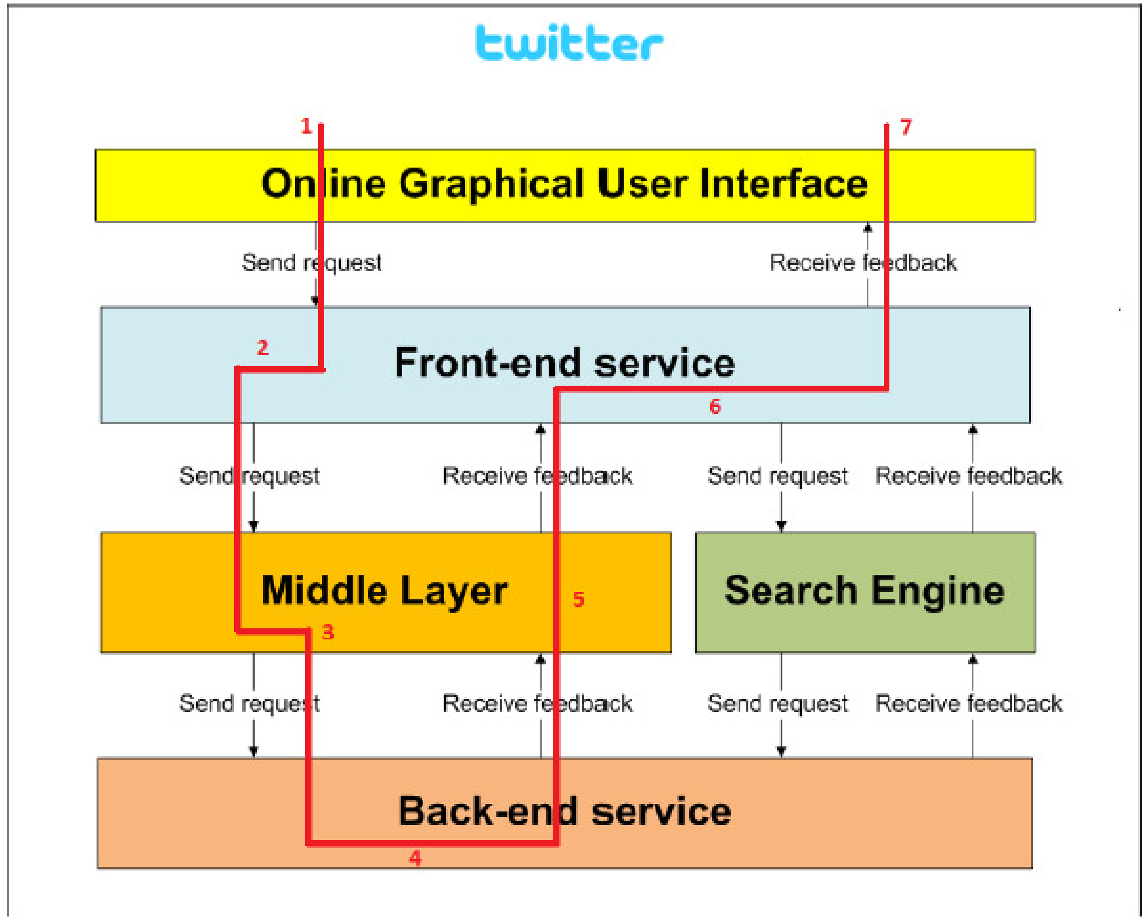
Twitter example: technologies

- Outdated Ruby on Rails is abandoned
- Earlybird on top of Lucene for real-time search
- Kestrel as the message queue

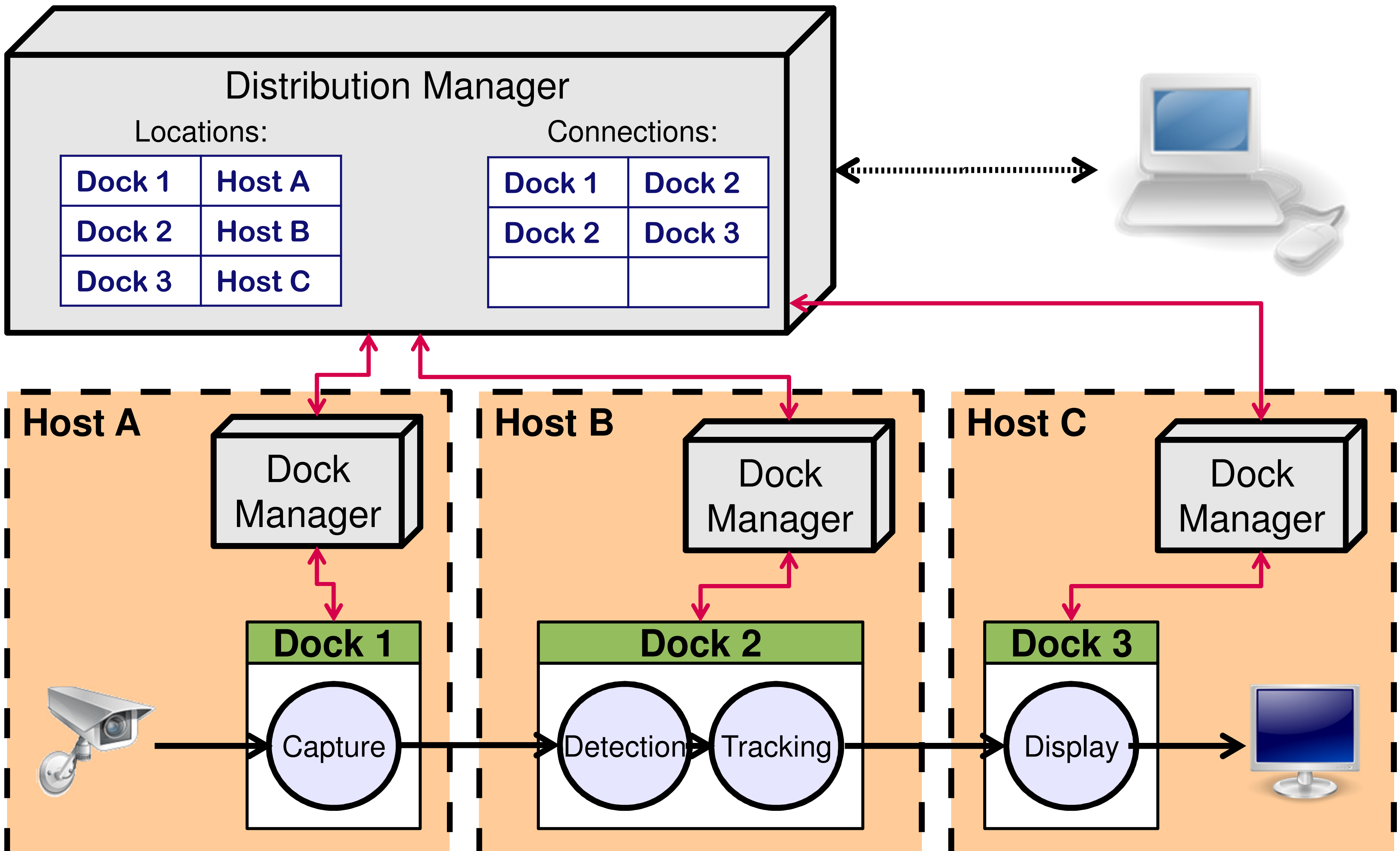


Twitter scenario example: insertion of a tweet

1. Logon to GUI and send tweet
2. Process the insertion request
3. Put the request in the queue system
4. Get request from front of queue and insert tweet into memcached memory. Update the search engine with an index entry
5. Send feedback to queuing system
6. Notify GUI



ViFramework



Architecture, views and models

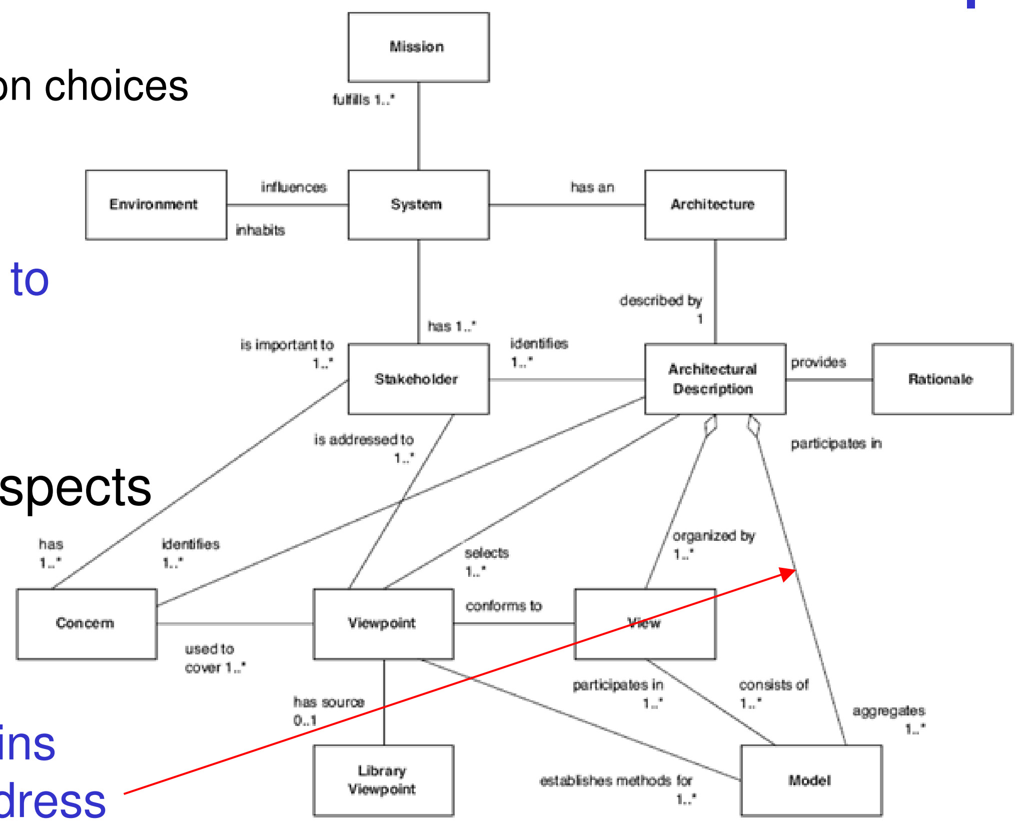
- Views need to be systematically related, typically by mappings
  - processes execute components, connectors are mapped to networks IPC or calls, objects use methods from their classes, etc.
  - scenario’s are instrumental here
    - e.g. a behaviorial model can be part of both logical view and process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

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



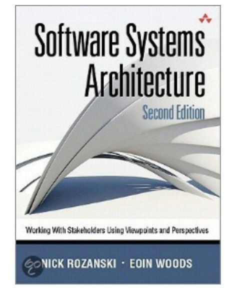
- The views do not cover all aspects
  - e.g. architecture design decisions may not be relevant in the logical view
  - hence, the architecture contains models that do not clearly address a stakeholder concern

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?

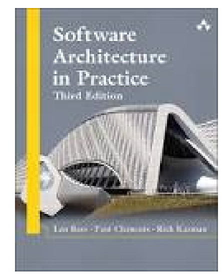
Literature

- Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2ed, *Rozanski and Woods*, Barnes and Noble, 2011

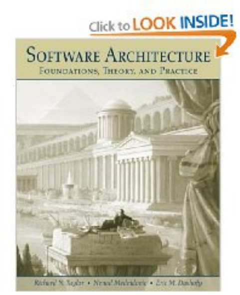


http://www.viewpoints-and-perspectives.info/

- Software Architecture in Practice, Third Edition, L. Bass, P. Clements, R. Kazman, SEI Series in Software Engineering, Addison-Wesley, 2013

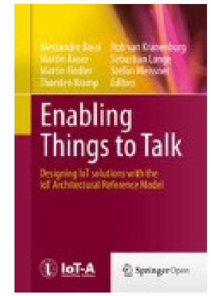


- Software Architecture, Foundations, Theory, and Practice, R.N. Taylor, N. Medvidovic, E.M. Dashofy, Wiley & Sons, 2009

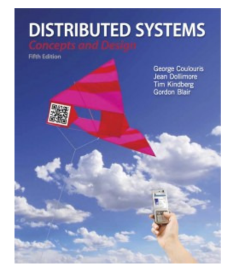


Literature

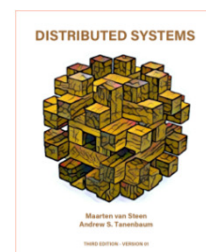
- Enabling Things to Talk, *Designing IoT solutions with the IoT Architectural Reference Model*, Eds. Alessandro Bassi, Martin Bauer, Martin Fiedler, Thorsten Kramp, Rob van Kranenburg, Sebastian Lange, Stefan Meissner, Springer, 2013



- Distributed Systems: Concepts and Design, 5ed, Coulouris, Dollimore, Kindberg and Blair, Addison Wesley, 2011.



- Distributed Systems, 3rd ed., version 01, Maarten van Steen, Andrew Tanenbaum, 2017



Literature

- Recommended Practice for Architectural Description,
  IEEE STD 1471-2000, 23 pages
- Systems and software engineering – Architecture description
  ISO/IEC/IEEE STD 42010-2011, 35 pages

Twitter references

- Matthijs Neppelenbroek, Matthias Lossek, Rik Janssen, Tim de Boer, *Twitter An Architectural Review*
- Twitter presentations at Qcon