Modeling a Software Architecture

Paolo Ciancarini

Agenda

- Describing software architectures
- Architectural frameworks
- Models based on architectural languages
- Models based on UML
- Main architectural views

Why document the architecture?

In the software life cycle we:

- **Create** an architecture
  - Using architectural patterns, design patterns, experience
- **Evaluate** the architecture
  - Using ATAM for example (see lecture on SA evaluation)
- **Refine**, **update**, and **refactor** the architecture along the way
- **Use** the architecture to guide the implementation
- (Try to) **enforce** the architecture during the implementation and throughout maintenance

Creating a software architecture

- The architecture of a software system is closely related to its quality attributes
- *Architectures allow or preclude nearly all of the system’s quality attributes*
- Without a proper architecture the quality of a system cannot be ensured or can be highly expensive, or even impossible, to implement

Sw architecture and function

- Qualities are attributes of the system, while the function is the purpose of the system
- Functionality describes what the system does; the quality "functional suitability" describes how well the system does its function
- *Functional suitability:* "the capability of a software product to provide functions which meet stated and implied needs when the software is used under specified conditions" (ISO 25010 *Systems and software Quality Requirements and Evaluation*)

Architecture and structure

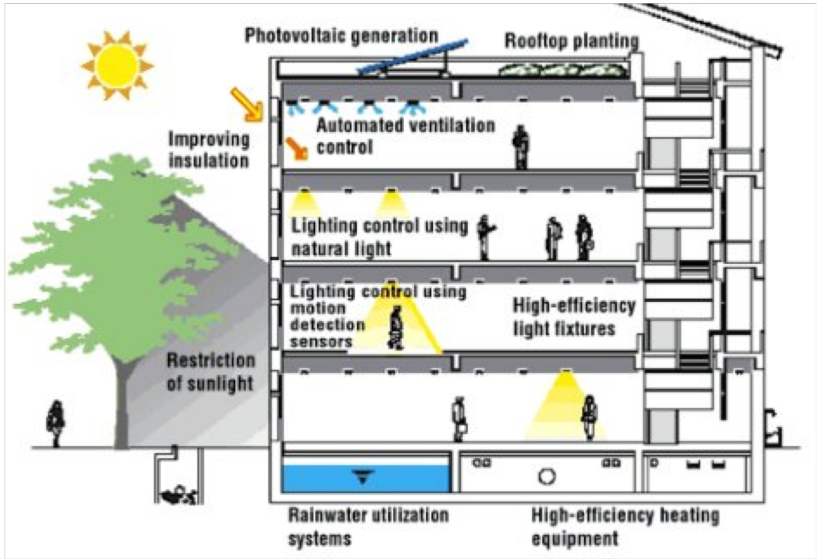
A *software architecture* includes multiples *structures*

- Each *structure* represents a different view of the system
- Each *structure* focusses on one aspect of the system

Analogy: the architecture of a building has many structures:

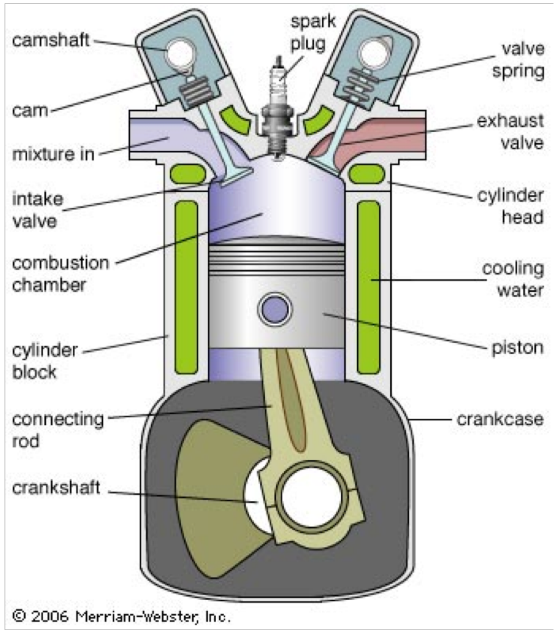
- Building structure
- Electricity supply structure
- Water supply structure

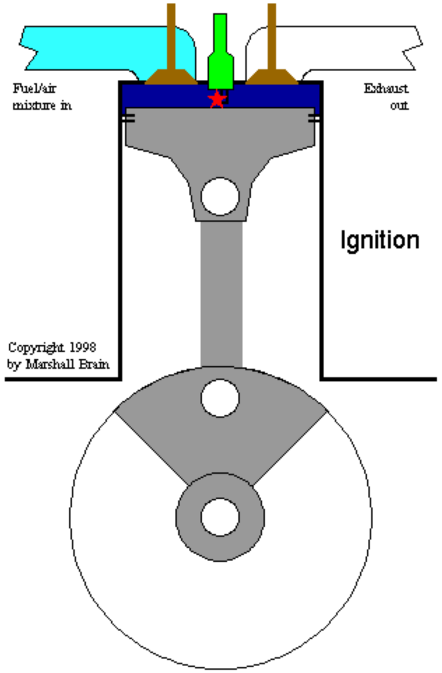
and so on


Photovoltaic generation
Rooftop planting
Improving insulation
Automated ventilation control
Lighting control using natural light
Lighting control using motion detection sensors
High-efficiency light fixtures
Restriction of sunlight
Rainwater utilization systems
High-efficiency heating equipment

Static and dynamic structures

- The static structures of a system define its internal *design-time elements* and their **arrangement**
- The dynamic structures of a system define its *run-time elements,* their **behaviors** and **interactions**

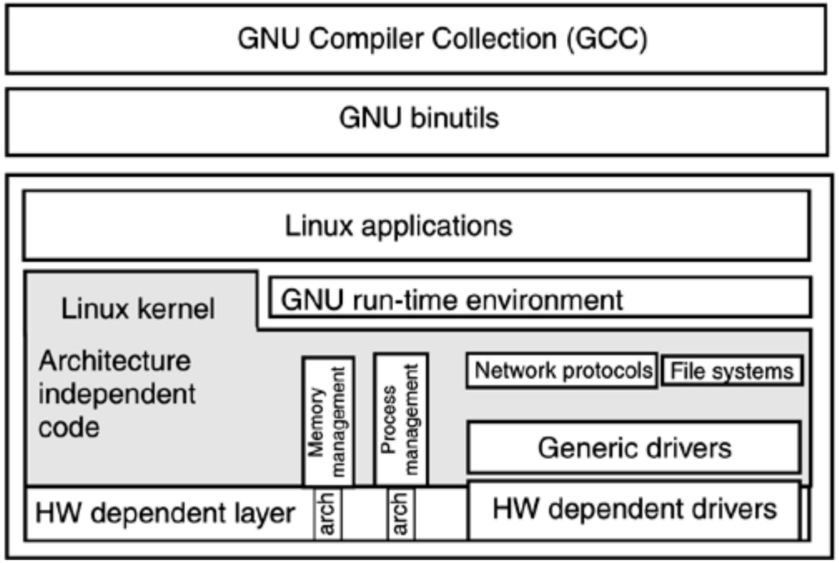


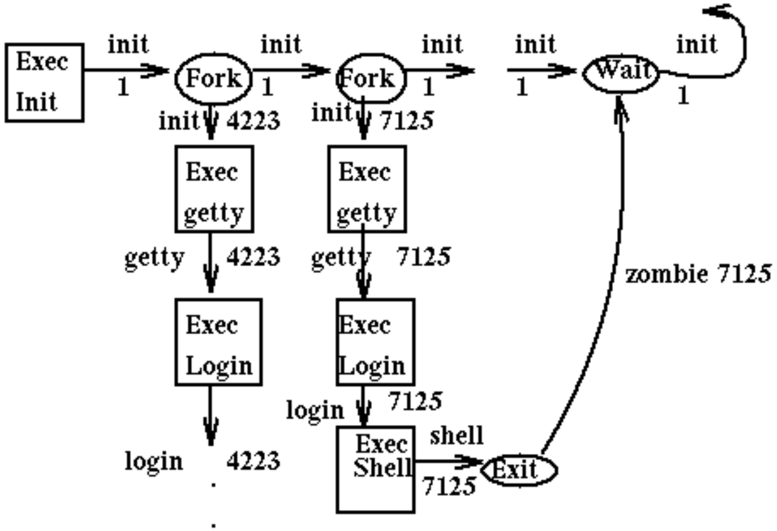


Example

- **Structure**: the source code of an operating system like Linux has a structure, that separates the kernel (eg. concerning processes and scheduling) from the services (eg. concerning the file system)
- **Behavior**: the operation of an operating system like Linux can be described as a set of concurrent processes which can invoke system calls; each call can raise events which can suspend or activate processes; we can say that a running Linux system is made of processes coordinated by events

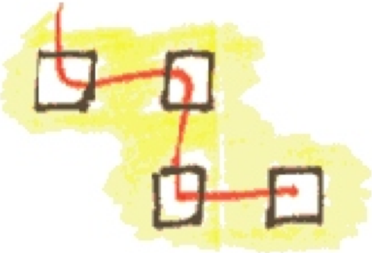
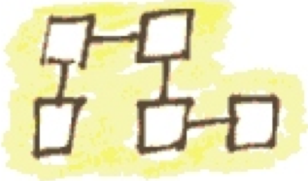
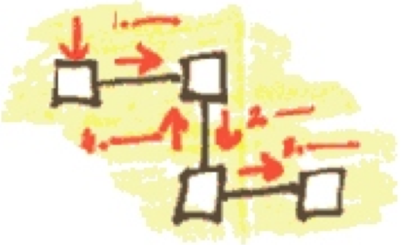
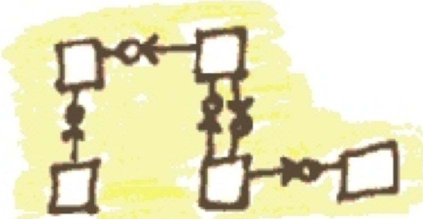
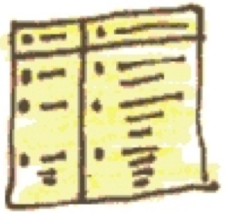
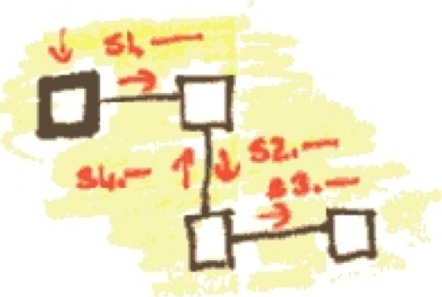
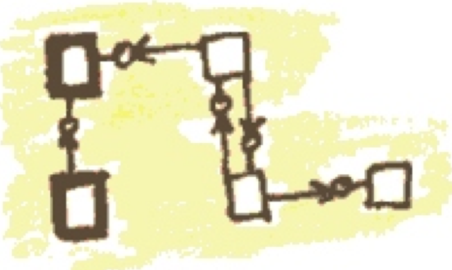
Structure vs behavior





Structure

Behavior

	Behavioral View	Structural View
Conceptual Architecture (abstract)	 Collaboration trace	 Architecture Diagram Informal Component Specs (CRC-R)
Logical Architecture (detailed)	 Collaboration Diagrams	 Architecture Diagram with I/Fs  Interface Specs
Execution Architecture (Process View and Deployment View)	 Collaboration Diagrams showing processes	 Architecture Diagram showing Active Components

Sw architecture and responsibility

- The distinction between function and quality sometimes is vague
  - for instance, if the function of a software is to control engine behavior, how can it be correctly implemented without considering timing behavior?
  - the ability to control access through requiring a user name/password combination is a function even though it is not the purpose of any system
- The word “responsibility” describes better the computations that a system must perform
  - Questions such as “What are the timing constraints on that set of responsibilities?”, “What modifications are anticipated with respect to that set of responsibilities?”, and “What class of users is allowed to execute that set of responsibilities?” make sense and have value
- Thus, the achievement of qualities induces a definition of responsibility

Responsibility-driven design

Responsibility-driven design is inspired by contracts (interfaces) between objects:

- DOING: what actions is this object responsible for?
- KNOWING: what information does this object share?

Example: who should be responsible for creating a new instance of a class? An object of class A can create objects of class B if

A aggregates objects B

A contains objects B

A can initialize objects B (A has the complete data for initialize B)

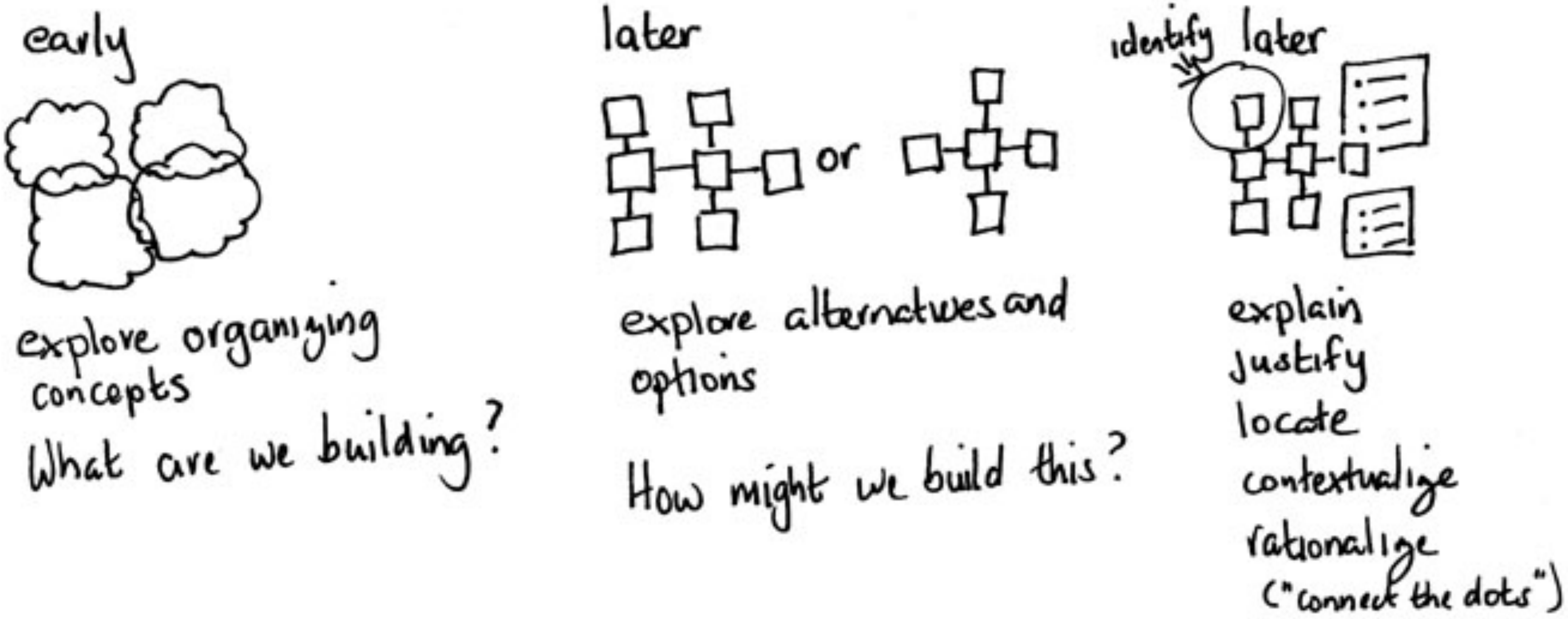
Models and views

- It is **not possible** to capture the functional features and the quality properties of a complex system in a **single model** that is understandable by and of value to all stakeholders
- A software architecture is modeled by many structures
  - Code units, their decomposition and dependencies
  - Processes and how they interact
  - How software is deployed on hardware
- A **view** is a representation of a structure
- It illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders

Architectural description

- An **architectural description** (AD) is a set of artifacts which collectively document an architecture in a way understandable by its stakeholders, and demonstrates that the architecture meets their concerns
- The artifacts in an AD include views, models, decisions, principles, constraints, etc., to present the essence of the architecture and its details, so that it can be validated and the described system can be built
- The AD can include other relevant information like business drivers, scope or requirements overview

Conceptual architecture view



explore organizing concepts

What are we building?

explore alternatives and options

How might we build this?

explain
justify
locate
contextualize
rationalize
("connect the dots")

http://ruthmalan.com/ByTopic/Architecture/ConceptualArchitecture.htm

The architecture document: template

① Architectural goals

② Significant requirements

  ① Functional

  ② Nonfunctional

③ Decisions and justifications

④ Key abstractions (Domain model)

⑤ Architectural description

  ① Logical component model

  ② Process model

  ③ Physical components and layers

  ④ Development model

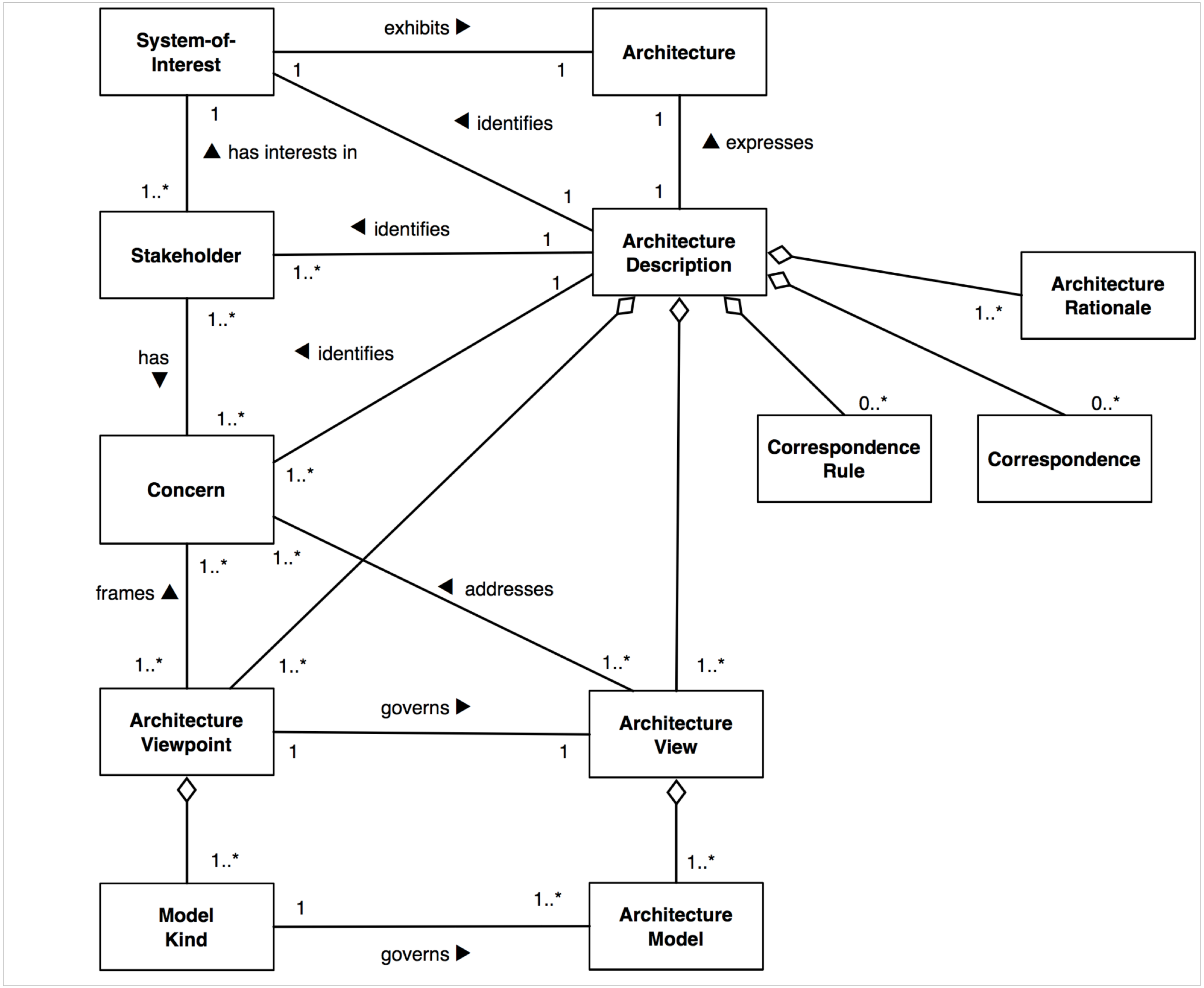
⑥ Deployment model

The ISO Standard 42010

This standard, the most recent version of IEEE 1471, makes a distinction between Architectures and Architecture Descriptions

- Architecture description, identification and overview
- Identification of stakeholders and concerns
- Selection of architecture viewpoints
- Architecture views
- Consistency and correspondences among architectural views
- Architectural rationale

A Conceptual Model of Architecture Description

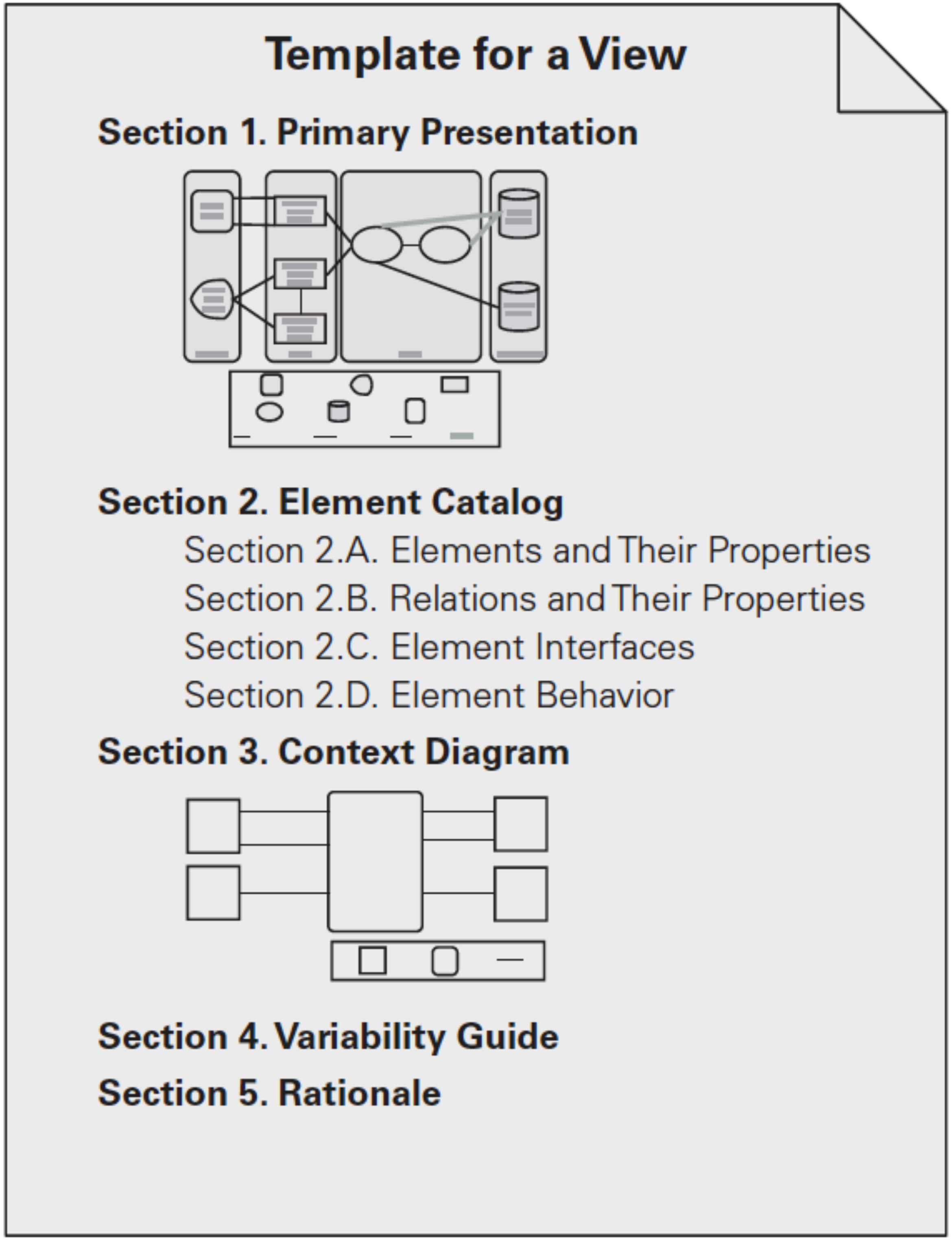


http://www.iso-architecture.org/ieee-1471/cm/

Documenting sw architectures: views

- A **view** is a description of a system according to the perspective (**viewpoint**) of some stakeholder, who has to satisfy some interest (**concern**)
- **Example**: a user view describes the typical scenarios where a system can be used
- An **architectural view** is a description of some relevant issues of a software architecture
- **Example**: the architectural view of packages necessary to install a software system, depicting their dependencies

Template for an architectural view



Template for a View

Section 1. Primary Presentation

Section 2. Element Catalog

Section 2.A. Elements and Their Properties

Section 2.B. Relations and Their Properties

Section 2.C. Element Interfaces

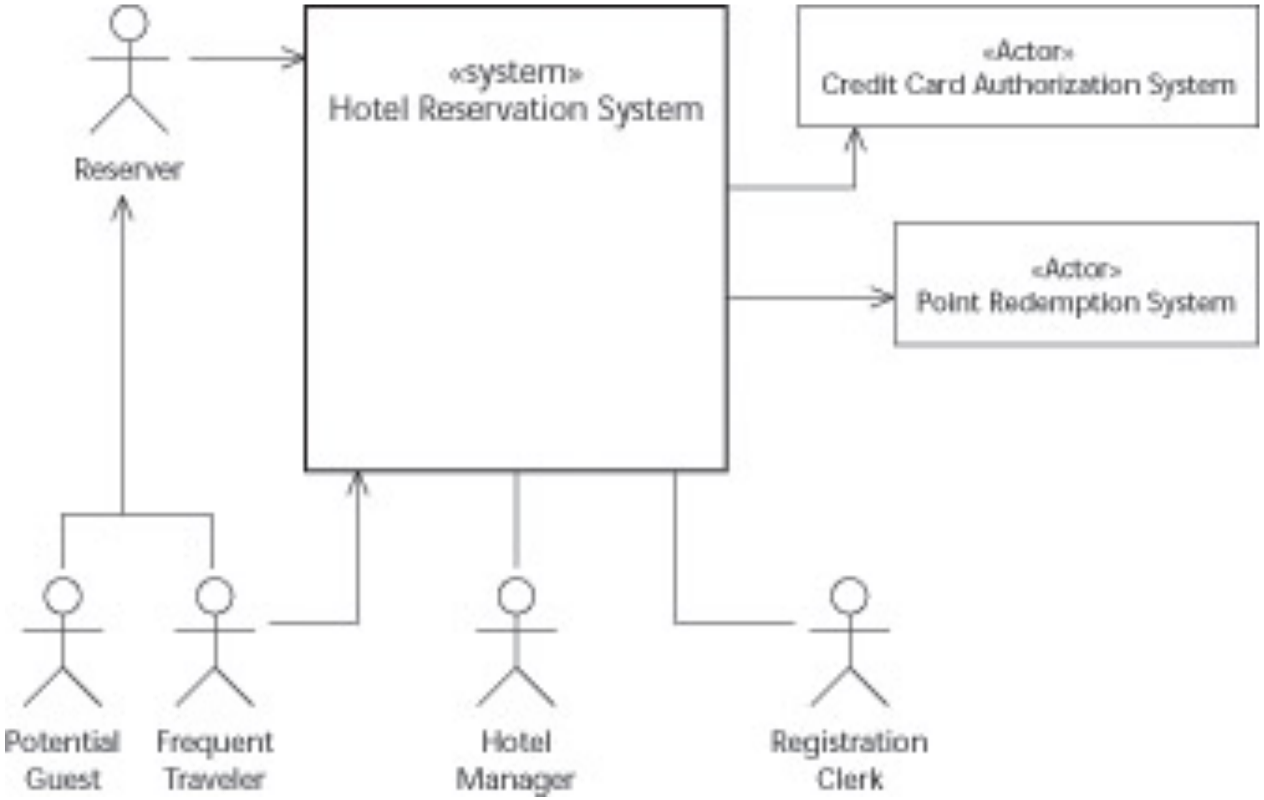
Section 2.D. Element Behavior

Section 3. Context Diagram

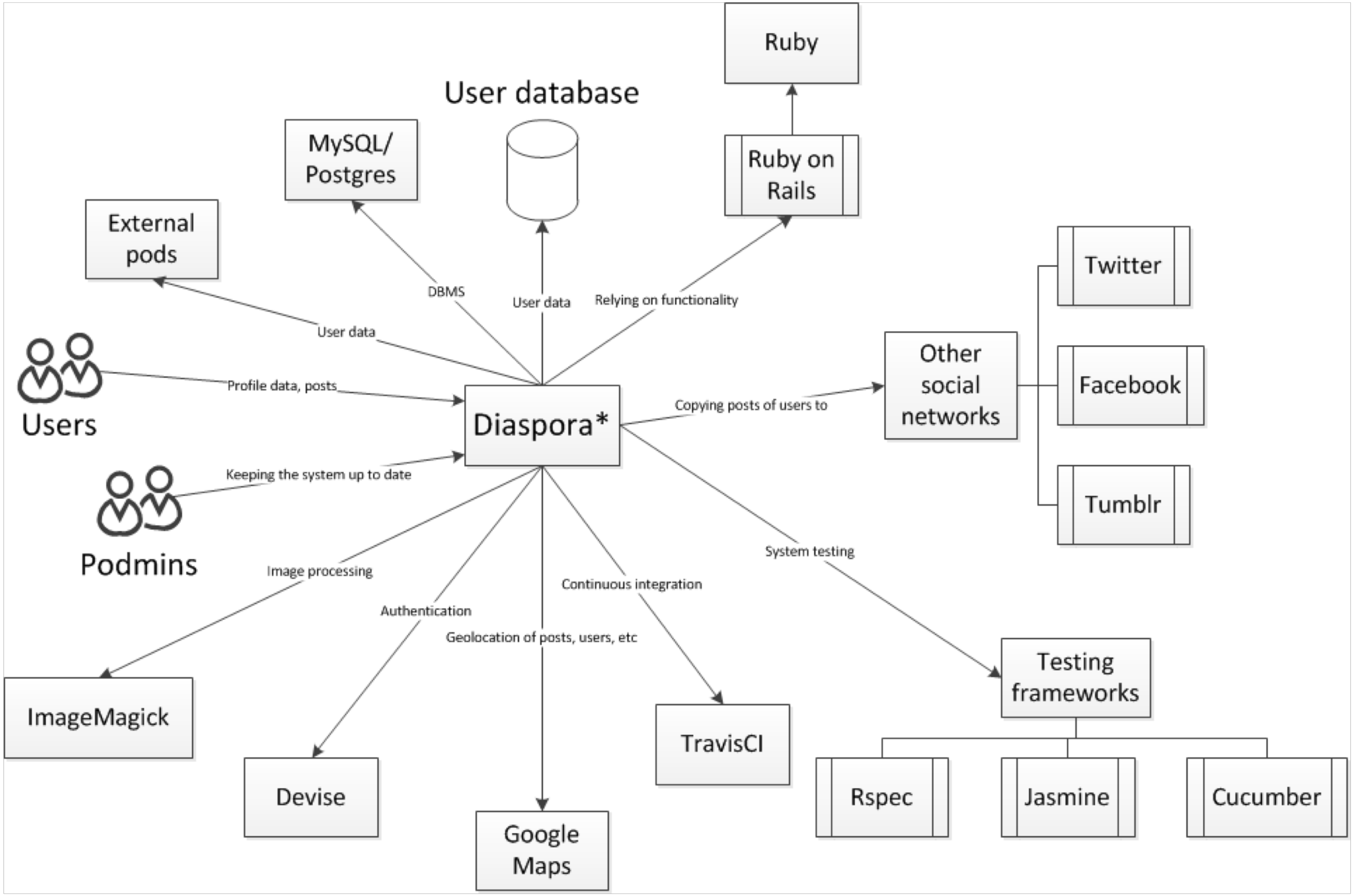
Section 4. Variability Guide

Section 5. Rationale

Example: context view

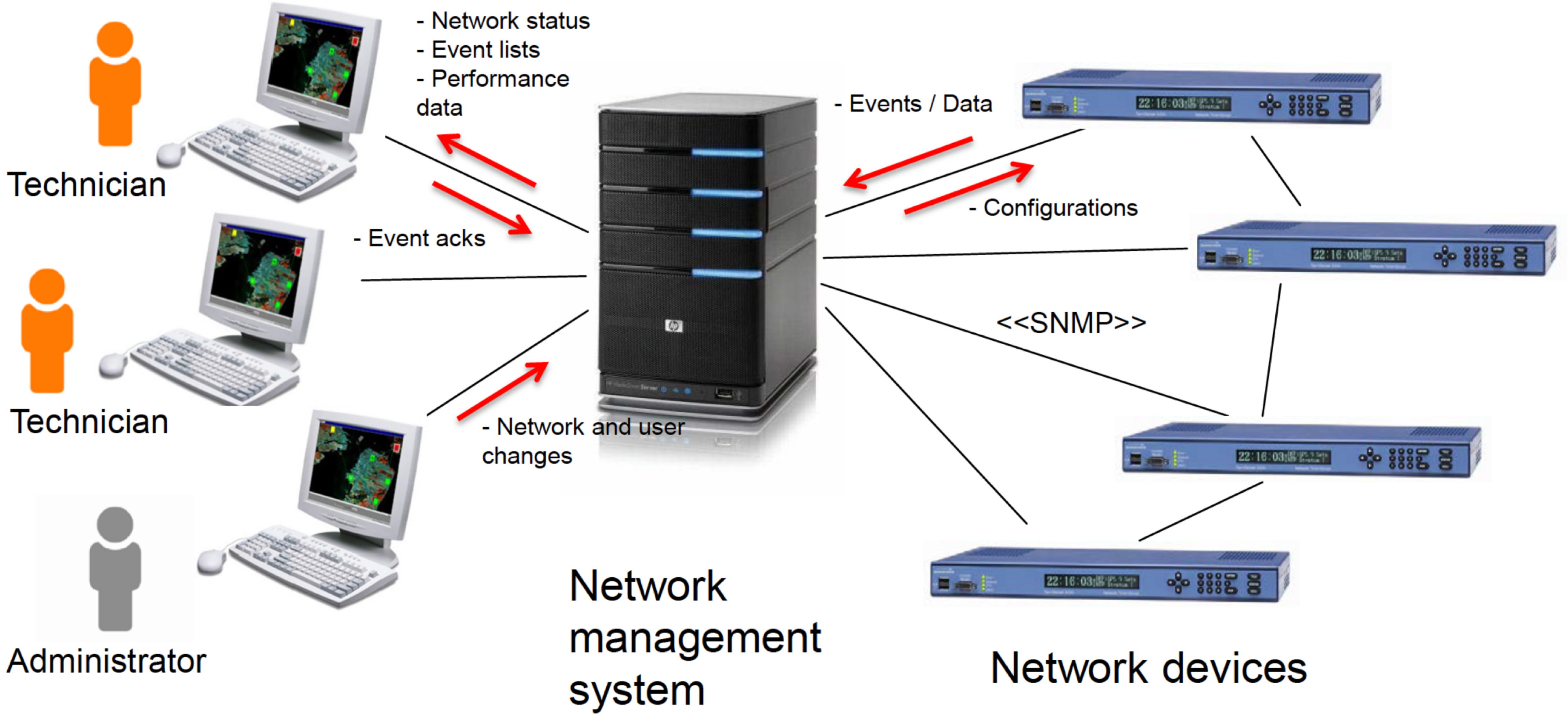


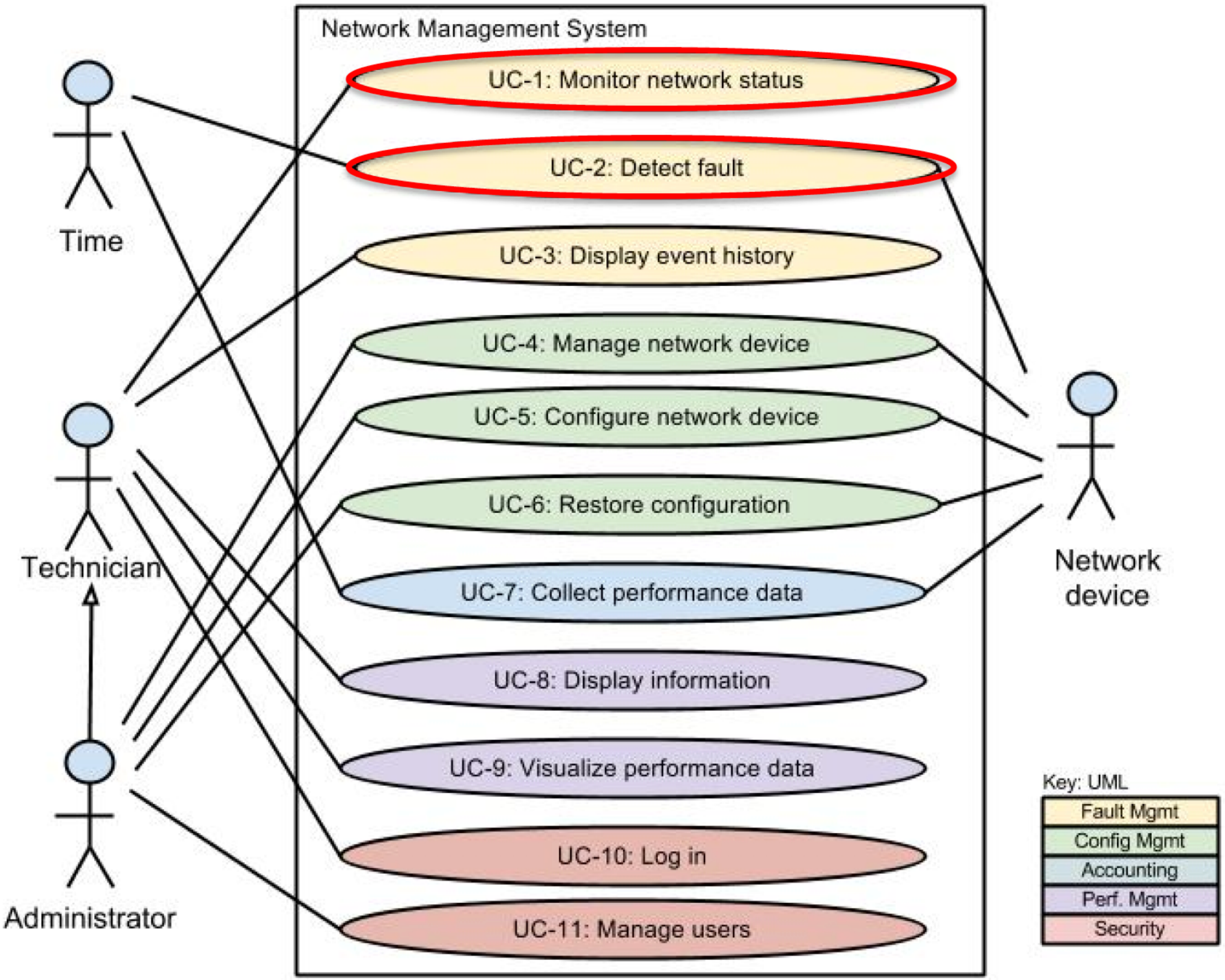
Example: context view



http://avandeursen.com/author/avandeursen/#Content

Example: context view

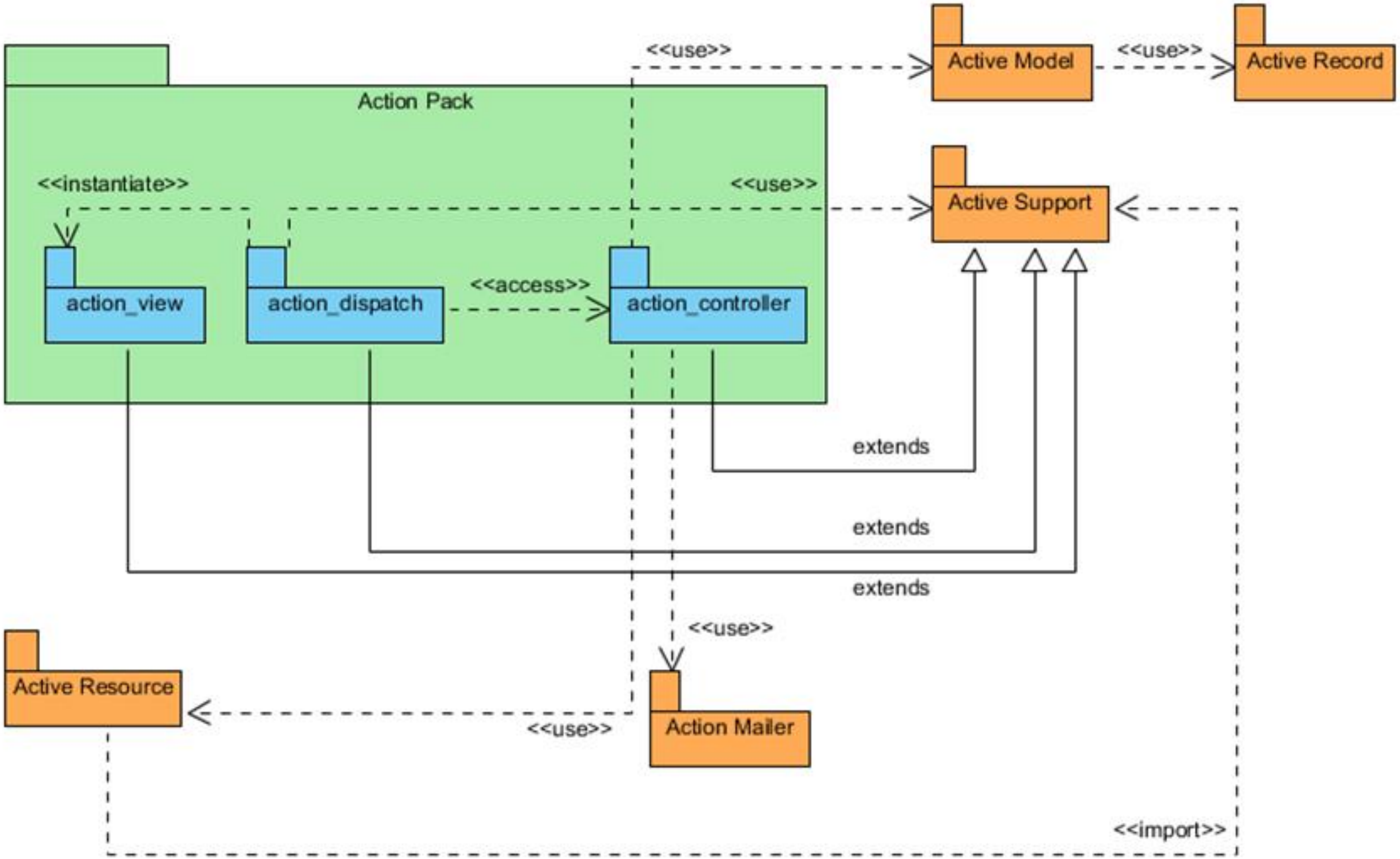



Network Management System
UC-1: Monitor network status
UC-2: Detect fault
UC-3: Display event history
UC-4: Manage network device
UC-5: Configure network device
UC-6: Restore configuration
UC-7: Collect performance data
UC-8: Display information
UC-9: Visualize performance data
UC-10: Log in
UC-11: Manage users
Time
Technician
Administrator
Network device
Key: UML
Fault Mgmt
Config Mgmt
Accounting
Perf. Mgmt
Security

Example: requirements view

- Purpose: documenting the system requirements
- Stakeholders: architects, developers, customers, management, testers, project lead, domain experts
- Concerns:
  - What does the business context of the system looks like?
  - What are the essential requirements the system must satisfy?
- Artifacts:
  - problem description and business opportunities
  - stakeholders
  - business processes
  - requirements
  - guidelines

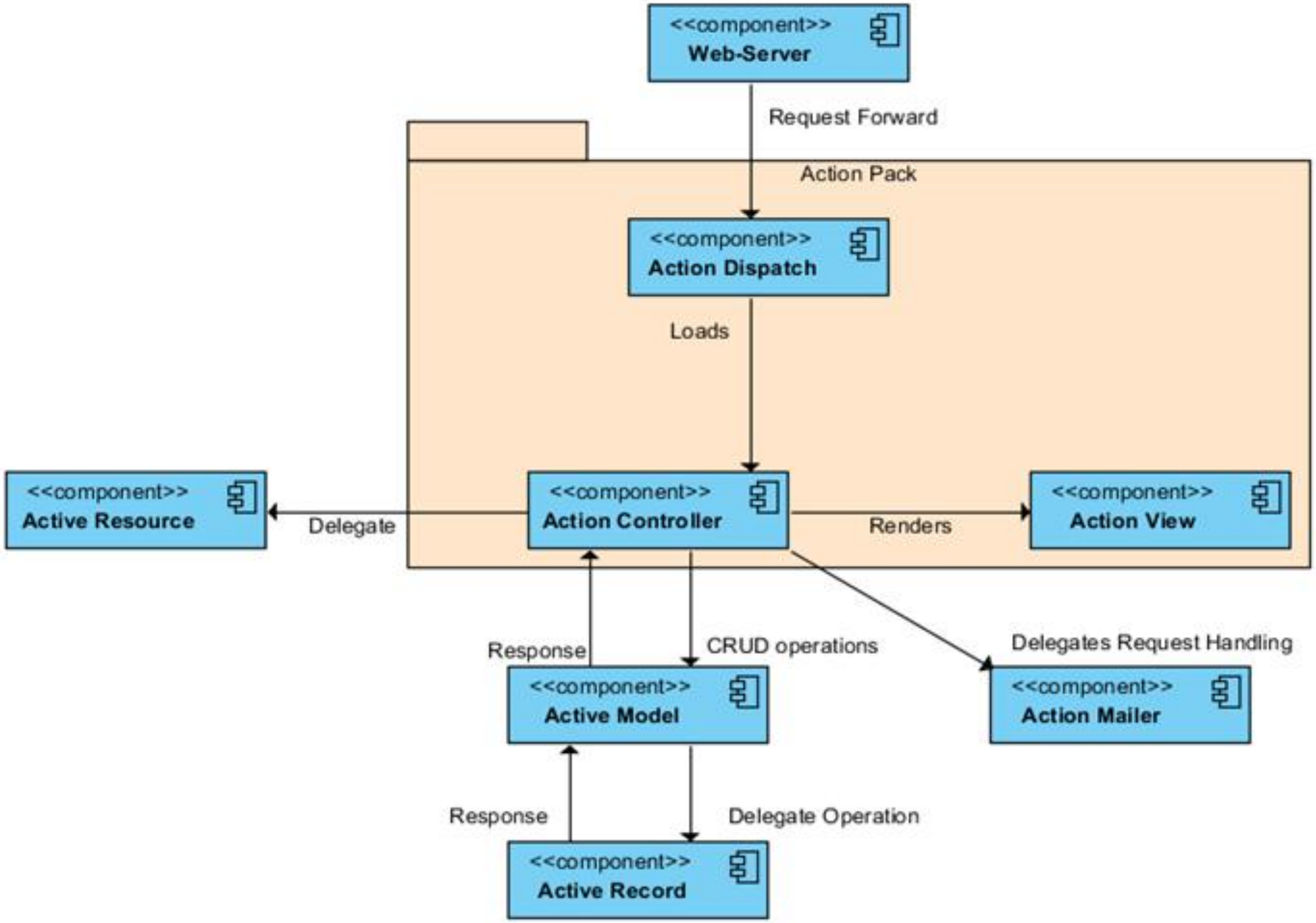
Example: a module view



Module view: summary

- Modules are implementation units, each providing a coherent set of responsibilities
- Relations
  - Is part of
  - Depends on
  - Is a
- Goals:
  - Providing a blueprint for constructing the code
  - Facilitating impact analysis
  - Supporting traceability analysis
  - Supporting the definition of of work assignments

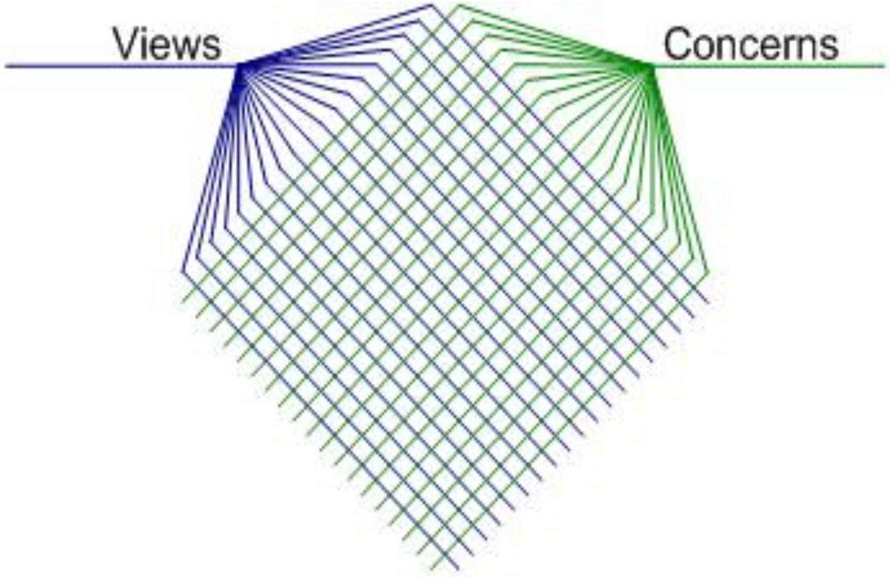
Example: a C2 view



What we put in a C&C view

- components and component types
- connectors and connector types
- component interfaces representing points of interaction between a C&C component and its environment
- connector interfaces (or roles)
- systems as graphs of components and connectors
- decomposition: a means of representing substructure and selectively hiding complexity
- properties: attributes (such as execution time or thread priority) that allow to analyze the performance or reliability of a system
- styles: defining a vocabulary of component and connector types together with rules for how instances of those types can be combined to form an architecture in a given style. Common styles include pipe-and-filter, client-server, and publish-subscribe

Architectural views and concerns



Code Distribution
Data Storage
Data Transmission
Deployment
Function/Logic/Services
Events
Hardware
Network
System Interface
User Interface
Usage

Accuracy
Availability
Concurrency
Consumability
Customization Points
Environment (Green)
Internationalization
Layering/Partitioning
Maintenance
Operations
Quality
Performance
Regulations
Reliability
Reuse
Security
Serviceability
Support
Timeliness
Usability
Validation

Pre-runtime vs runtime views

Architectural views can be classified into:

- **Pre-runtime views**: Module and some Allocation views. E.g. an implementation structure where implementation files are mapped to a file structure
- **Runtime views**: Component-and-connector (C2) and some Allocation views. E.g. a deployment view, involving mobile code (e.g. mobile agents)

Researcher vs practitioners

- The researchers’ community in the academia
- The practitioners’ community in the industry

use different approaches to describe software architectures

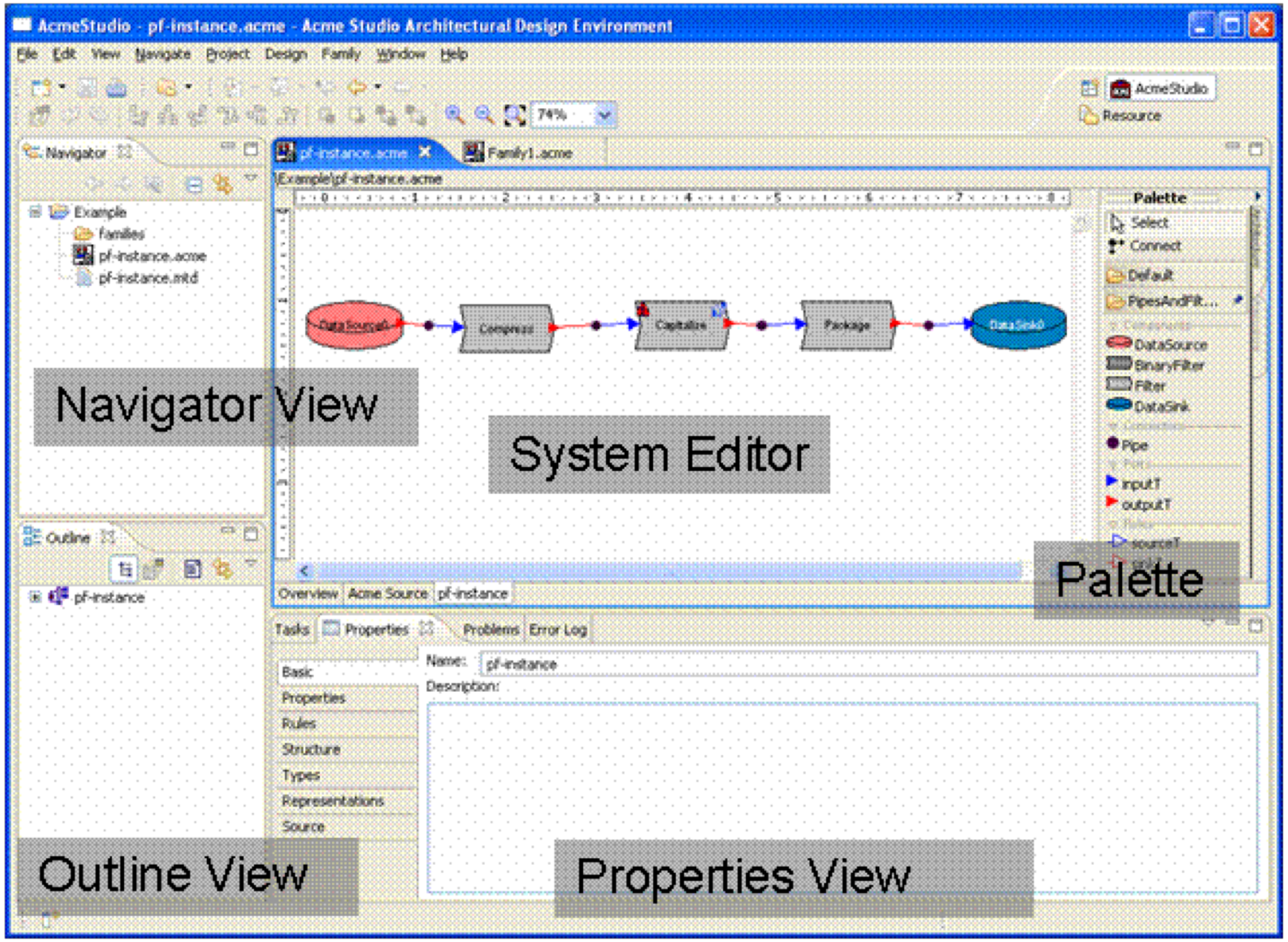
Researchers: use ADL

- Researchers advocates using architecture description languages (ADLs), which represent formal notations for describing architectures in terms of coarse-grained components and connectors

- ADLs are usually domain-specific languages
- ADLs provide solid support for formal verification and correction but it are considerably more difficult to use than UML

ADL: examples

- ACME (ACMEStudio)
- Darwin
- Archimate



Name: pf-instance

Description:

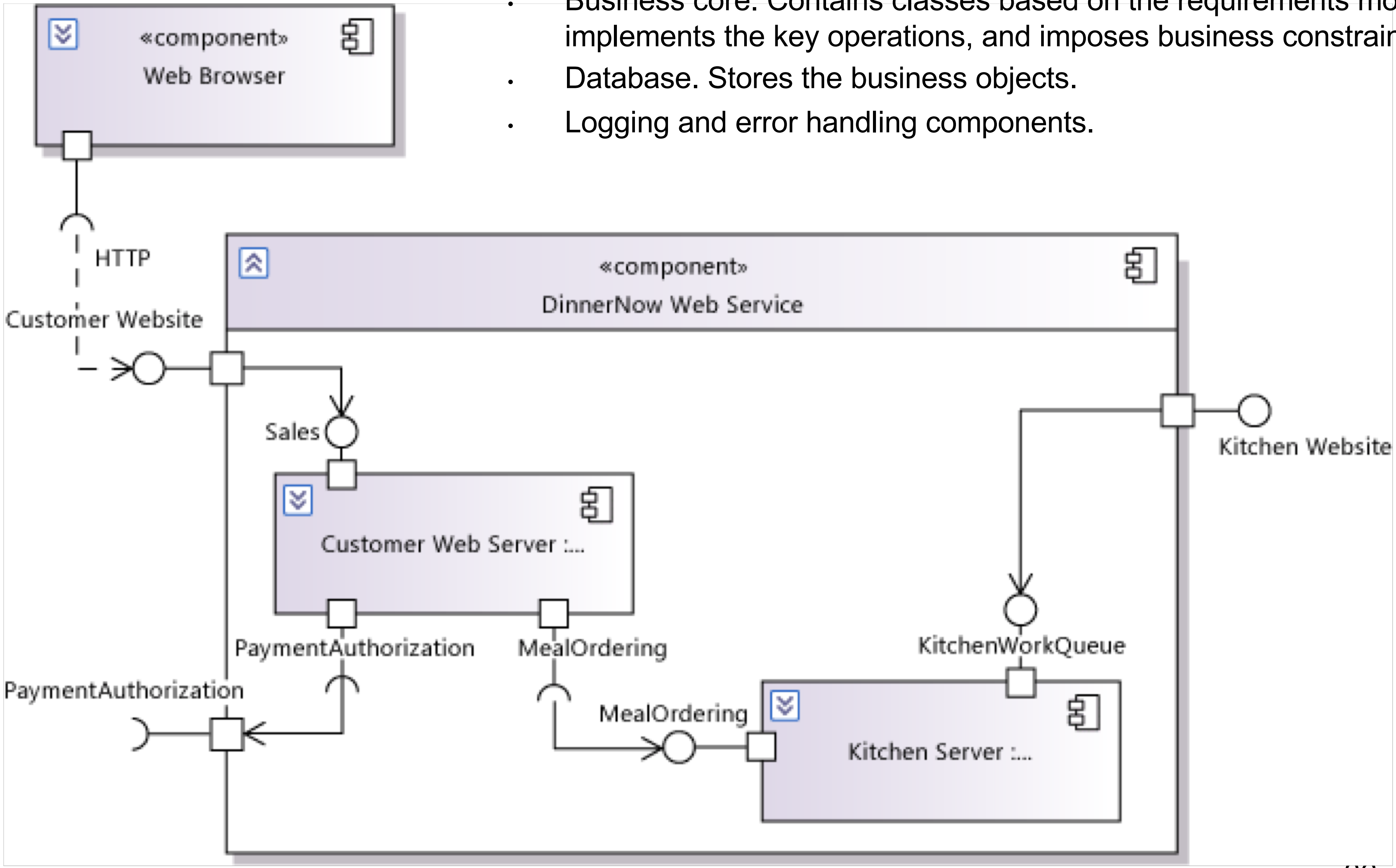
Practitioners: use UML

- UML is a very general purpose modeling language, as it can be used for any software product and even for hardware systems
- UML has become a standard de facto for documenting software systems
- UML 2.0 includes some extensions from the ADL world
- UML has been extended to SysML for documenting system engineering artfacts

Example: component diagram

A component diagram for a large system might include:

- Presentation. The component that provides access to the user, typically running on a Web browser.
- Web service. Provides connection between clients and servers.
- Use case controllers. Conduct the user through each scenario.
- Business core. Contains classes based on the requirements model, implements the key operations, and imposes business constraints.
- Database. Stores the business objects.
- Logging and error handling components.



http://msdn.microsoft.com/en-us/library/dd490886.aspx

UML: disadvantages

- The descriptions can be ambigous
- Weak tool support for detecting inconsistencies
- Inability to establish traceability between design and code
- Incomplete architectural details (most of the times reverse engineering is needed to detect them)
- Lack of an architectural theory: which views are more important to represent?

I.Jacobson on agile modeling

- It is not smart to model everything in UML (happened often for enterprise architecture projects).
- It is not smart to model nothing and go straight to code.
- It is however smart to find exactly that something that is of importance to model and code.

The theory of sw architecting

The theory of sw architecting studies how architectures are described or built

- Architectural models (eg. UML/SysML)
- Architectural tools (eg. Archi, ACMEstudio)
- Architectural mechanisms in programming models (eg. Coordination-based or agent-based)

Typical architectures to study

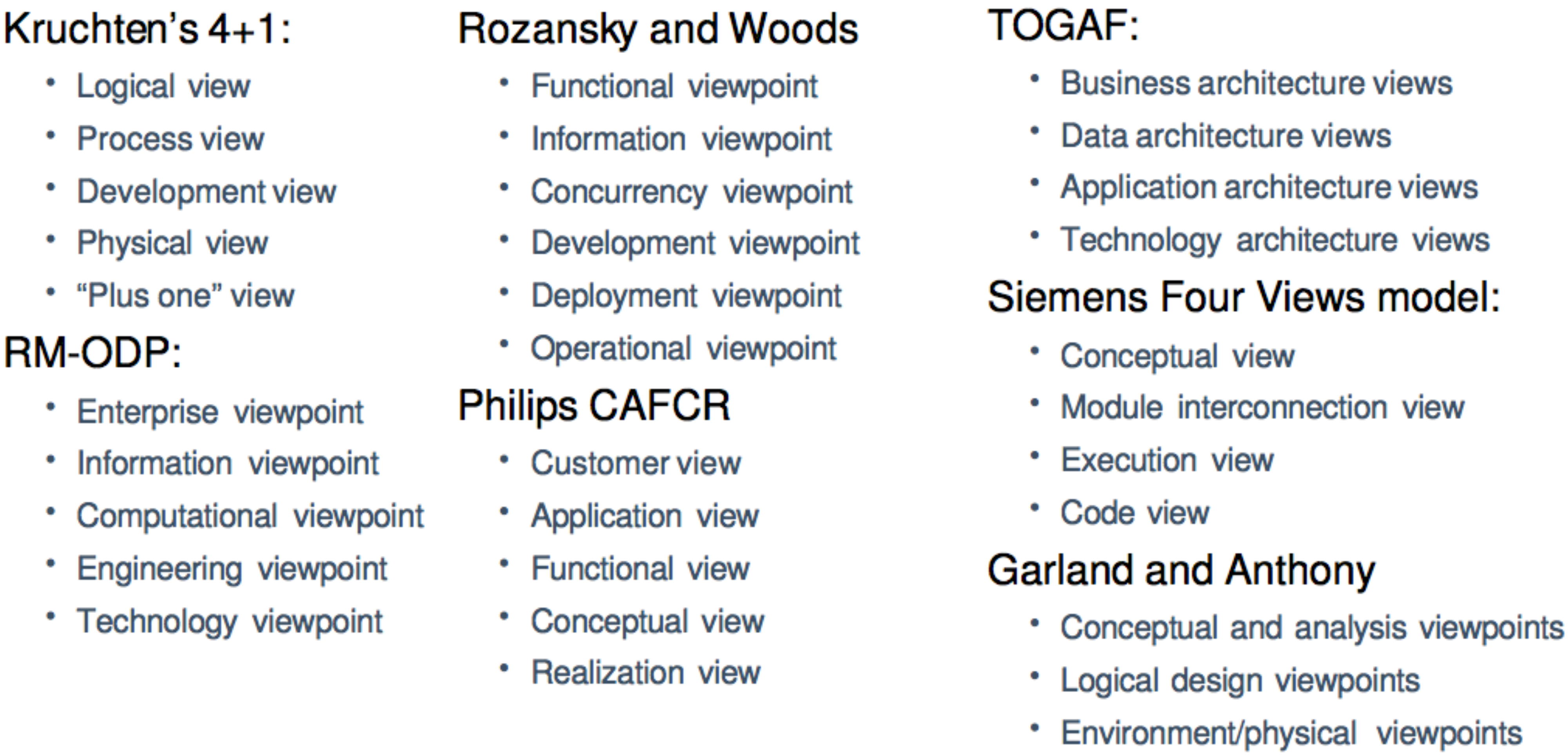
- World wide web
- Mobile applications (eg. Games + advertising)
- Cloud-based services (eg. Elastic clouds)
- Multiagent systems
- Adaptive systems
- Software ecosystems (eg. App Store)
- Social applications (eg. Facebook, Twitter)
- Embedded social software (eg. Traffic management based on social recommendations)

Architectural frameworks

Architectural frameworks are methods for creating, interpreting, analyzing and using architectural models within a domain of application or a stakeholder community

- **Kruchten’s 4+1 Views** framework
- **Hofmeister & Nord & Soni’s** framework
- **Rozanski & Woods’** framework
- **Clements & Bass’** framework
- **C4** framework by Simon Brown
- **TOGAF** framework

Which views are available?



Kruchten's 4+1:

- Logical view
- Process view
- Development view
- Physical view
- "Plus one" view

RM-ODP:

- Enterprise viewpoint
- Information viewpoint
- Computational viewpoint
- Engineering viewpoint
- Technology viewpoint

Rozansky and Woods

- Functional viewpoint
- Information viewpoint
- Concurrency viewpoint
- Development viewpoint
- Deployment viewpoint
- Operational viewpoint

Philips CAFCR

- Customer view
- Application view
- Functional view
- Conceptual view
- Realization view

TOGAF:

- Business architecture views
- Data architecture views
- Application architecture views
- Technology architecture views

Siemens Four Views model:

- Conceptual view
- Module interconnection view
- Execution view
- Code view

Garland and Anthony

- Conceptual and analysis viewpoints
- Logical design viewpoints
- Environment/physical viewpoints

Which views are more useful?

- An architect usually considers at least four perspectives of the system:

1. How is it structured as a set of code units?

   Module Views

2. How is it structured as a set of elements that have runtime presence?

   Runtime Views

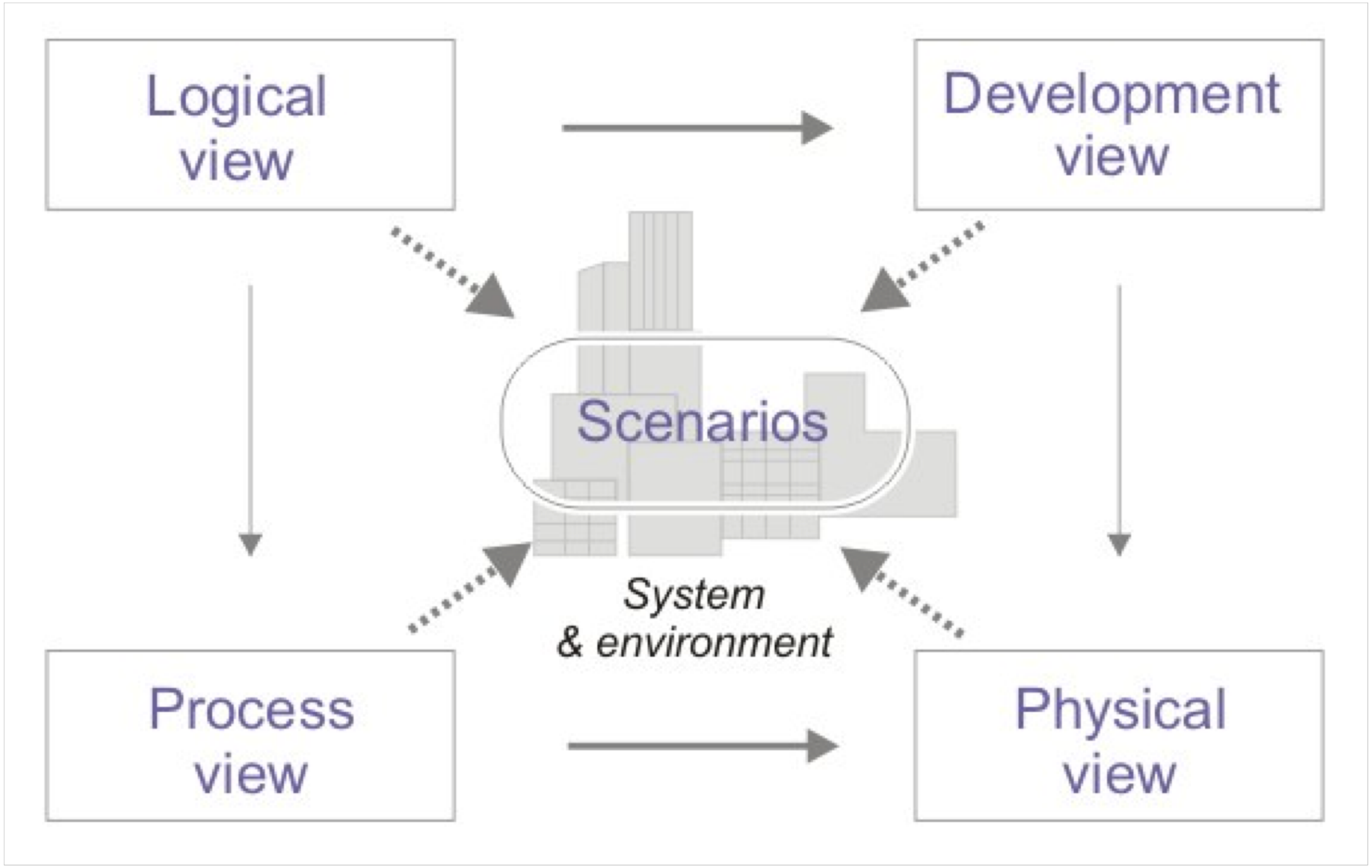
3. How are artifacts organized in the file system and how is the system deployed to hardware?

   Deployment Views

4. What are the data entities and their relationships?

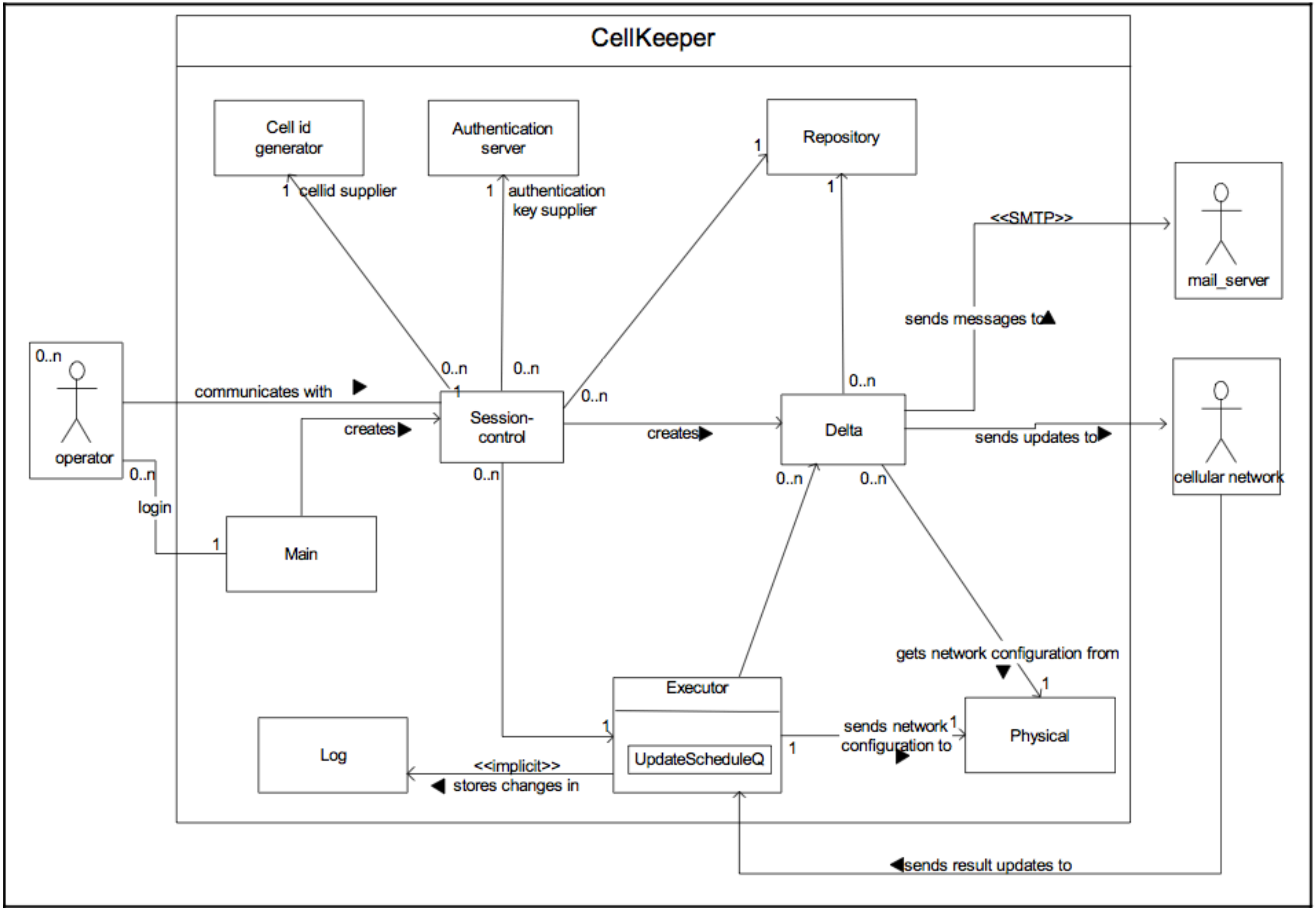
   Data Model

Views (Krutchen 4+1)

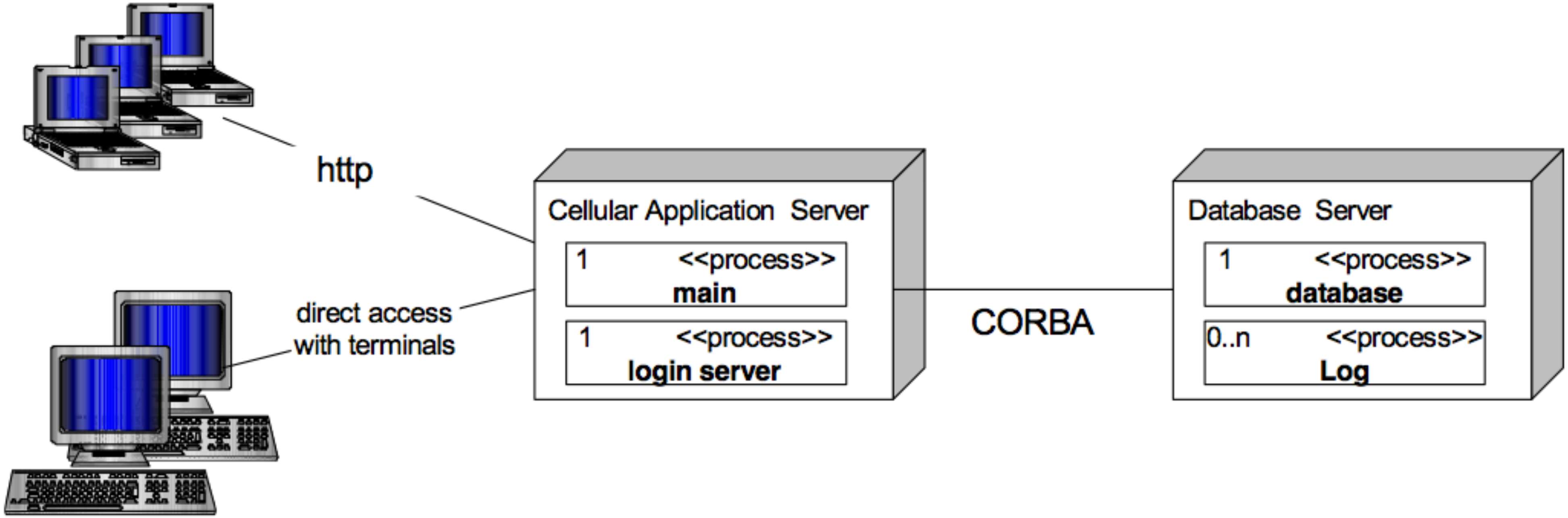


pkruchten.wordpress.com/architecture/

Logical view



Physical view



Siemens approach

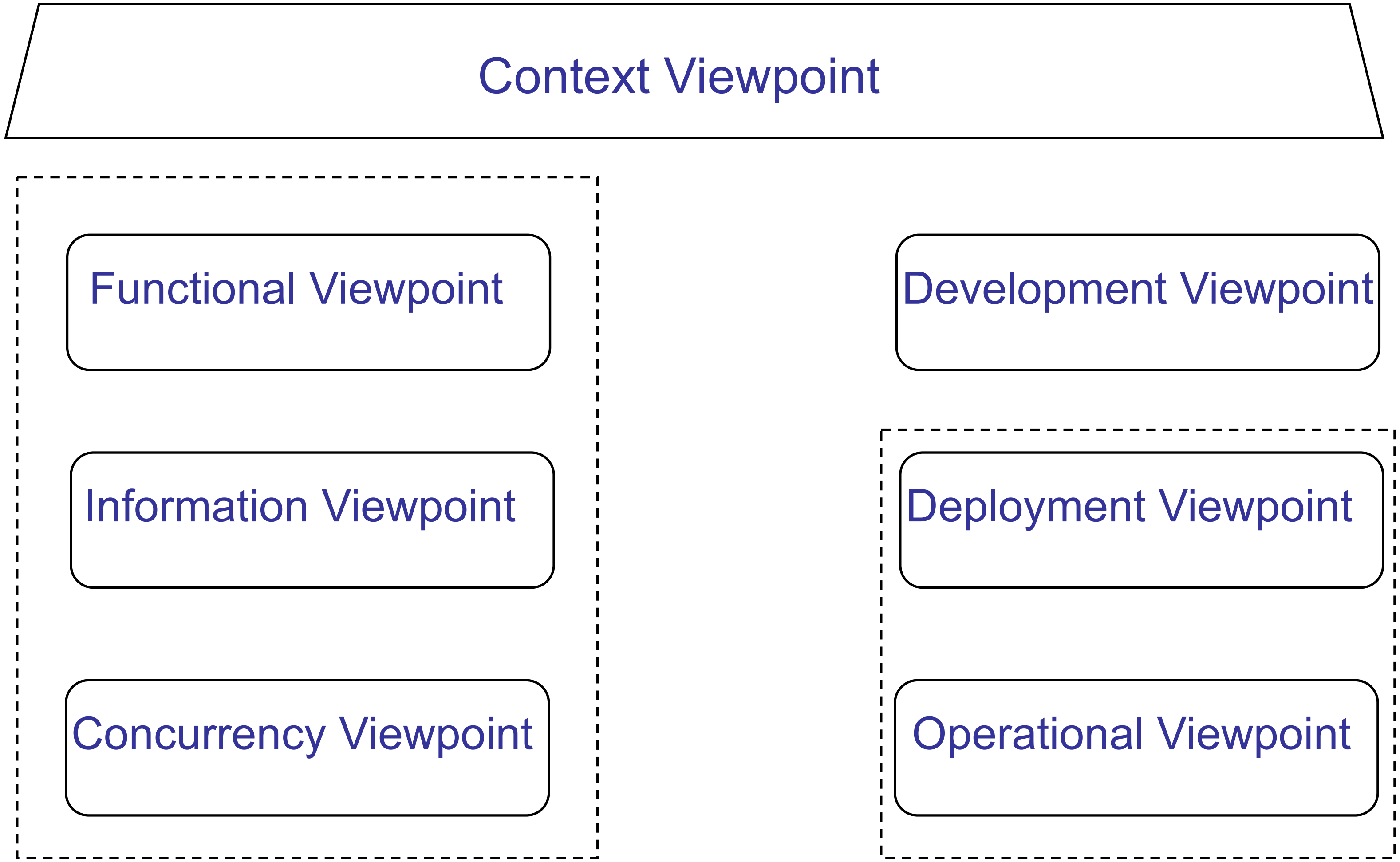
- Soni, Nord, and Hofmeister of Siemens Corporate Research described some views of software architectures they observed in use in industrial practice
- The conceptual view describes a system in terms of its major design elements and the relationships among them
- The module interconnection view combines two orthogonal structures: functional decomposition and layers
- The execution view describes the dynamic structure of a system
- Finally, the code view describes how the source code, binaries, and libraries are organized in the development environment

Rozanski and Woods approach

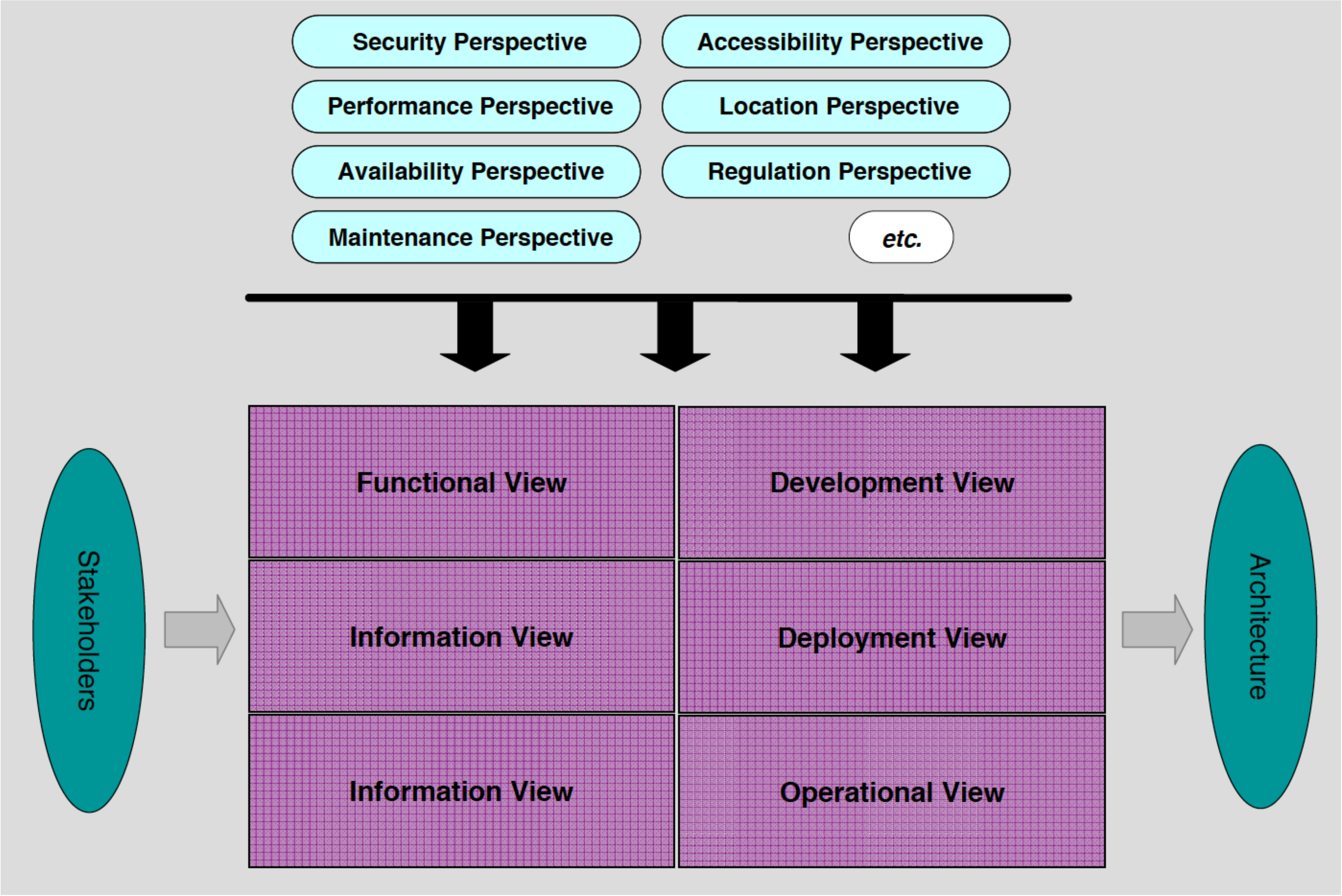
- **Context** viewpoint**:** describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts).
- **Functional** viewpoint: models the runtime elements which deliver functionality, including their responsibilities, interfaces and interactions
- **Information** viewpoint: how the architecture stores, manipulates, manages, and distributes information
- **Concurrency** viewpoint: state-related structure and constraints
- **Development** viewpoint: module organization and related tools
- **Deployment** viewpoint: physical environment in which the system runs
- **Operational** viewpoint: how the system will be operated, administered,

www.viewpoints-and-perspectives.info/

Rozanski and Woods viewpoints



Perspectives and views (Rozanski & Woods)



Impact of perspectives on viewpoints

(Rozanski & Woods)

Viewpoints	Security	Performance	Availability	Evolution
Context	High	Low	Medium	High
Functional	Medium	Medium	Low	High
Information	Medium	Medium	Low	High
Concurrency	Low	High	Medium	Medium
Development	Medium	Low	Low	High
Deployment	High	High	High	Low
Operational	Medium	Low	Medium	Low

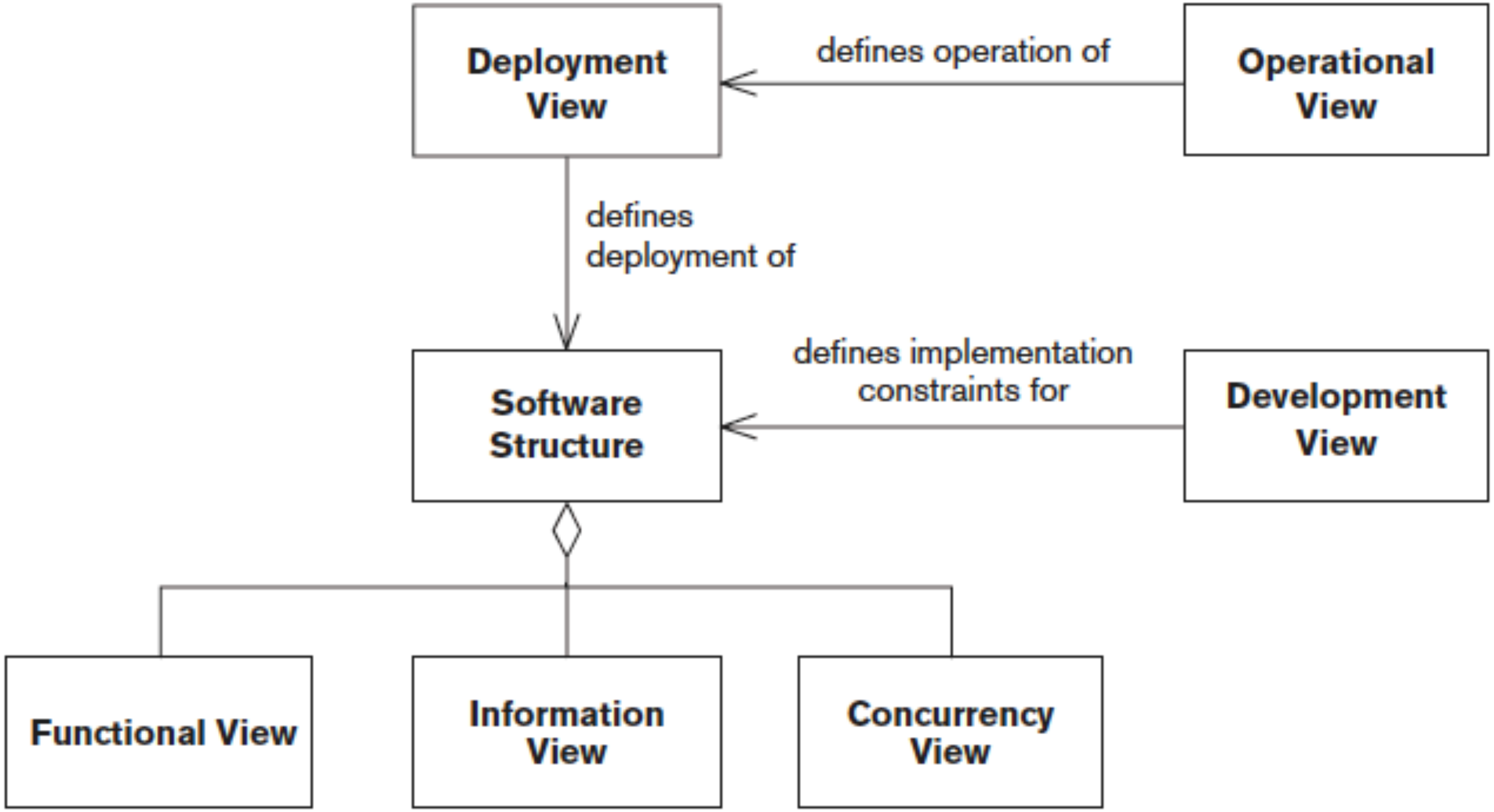
Example: most important views for typical system types

(Rozanski & Woods)

	Information system	Middleware	Military inf. system	High volume Web portal	Entreprise package
Context	High	Low	High	Medium	Medium
Functional	High	High	Low	High	High
Information	Medium	Low	High	Medium	Medium
Concurrency	Low	High	Low	Medium	variable
Development	High	High	Low	High	High
Deployment	High	High	High	High	High
Operational	variable	Low	Medium	Medium	High

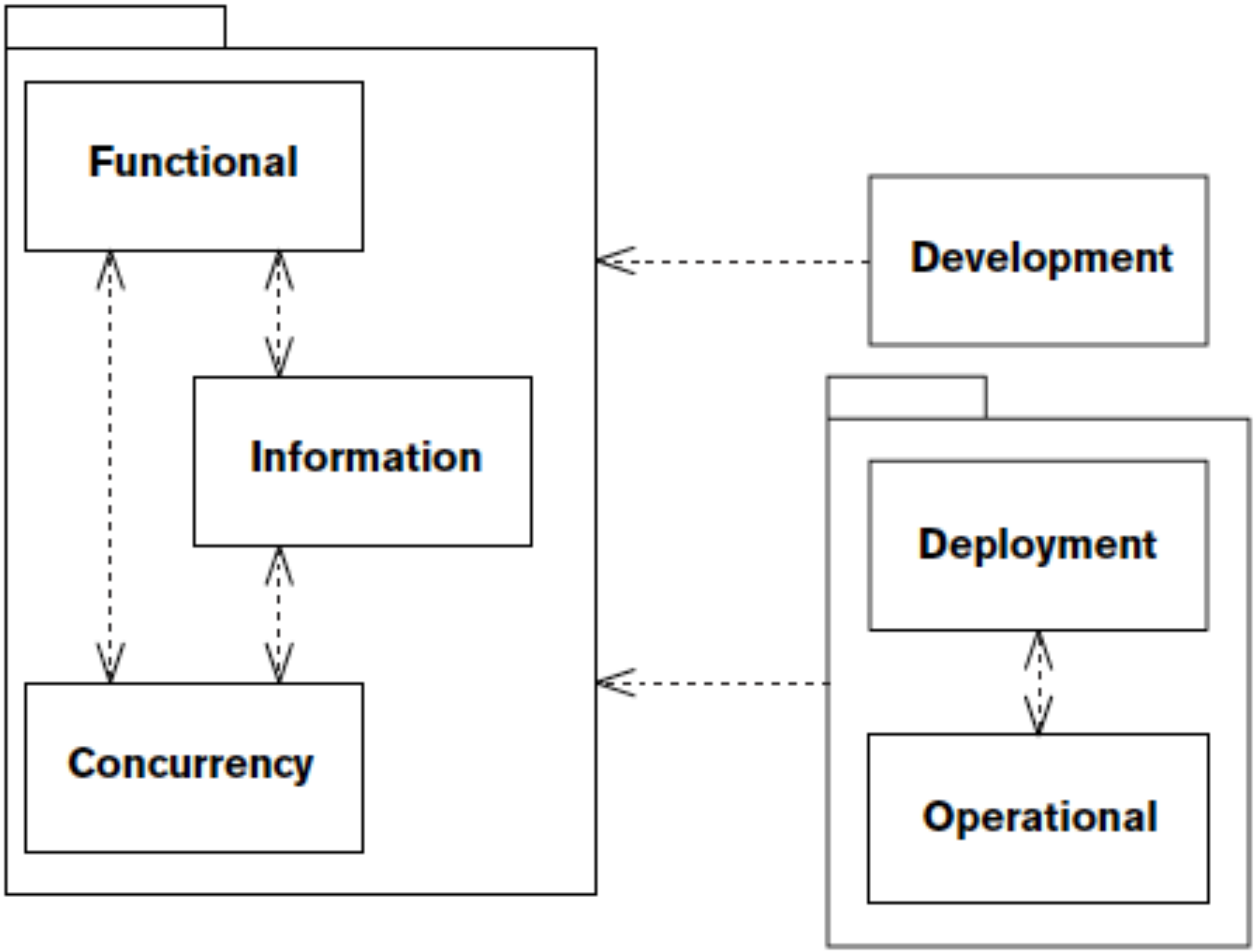
Relationships among views

(Rozanski & Woods)



Dependencies among views

(Rozanski & Woods)

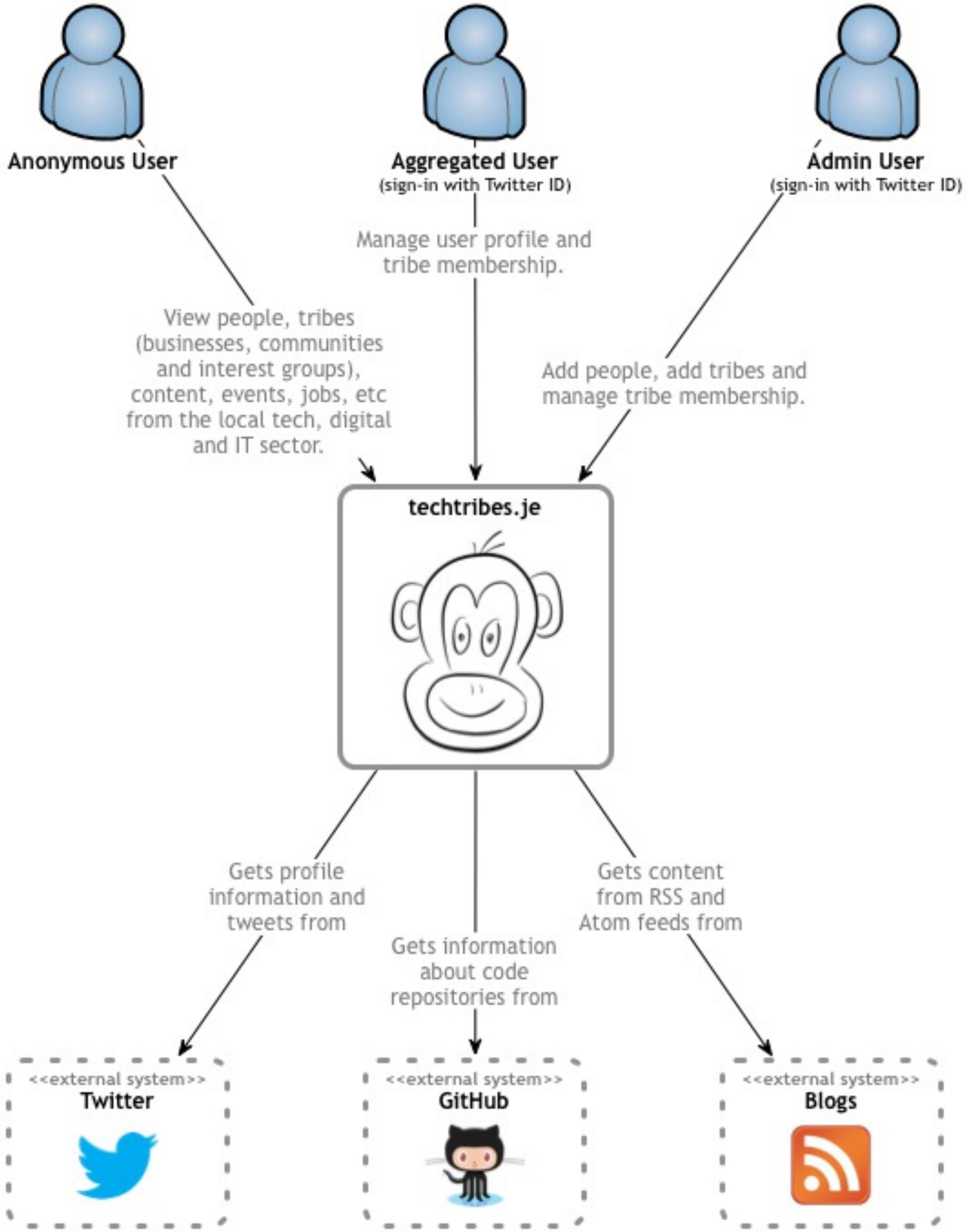


C4 (contexts containers components classes)

- Context: a simple block diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interfaces with
- A container is anything that can host code or data; it’s an execution environment or data storage
- The components diagram is used to zoom in and decompose a container
- The class diagram is an optional level of detail used to explain how a particular pattern or component will be (or has been) implemented.

www.voxxed.com/blog/2014/10/simple-sketches-for-diagramming-your-software-architecture/

Context



Anonymous User

Aggregated User
(sign-in with Twitter ID)

Admin User
(sign-in with Twitter ID)

View people, tribes (businesses, communities and interest groups), content, events, jobs, etc from the local tech, digital and IT sector.

Manage user profile and tribe membership.

Add people, add tribes and manage tribe membership.

techtribes.je

Gets profile information and tweets from

Gets information about code repositories from

Gets content from RSS and Atom feeds from

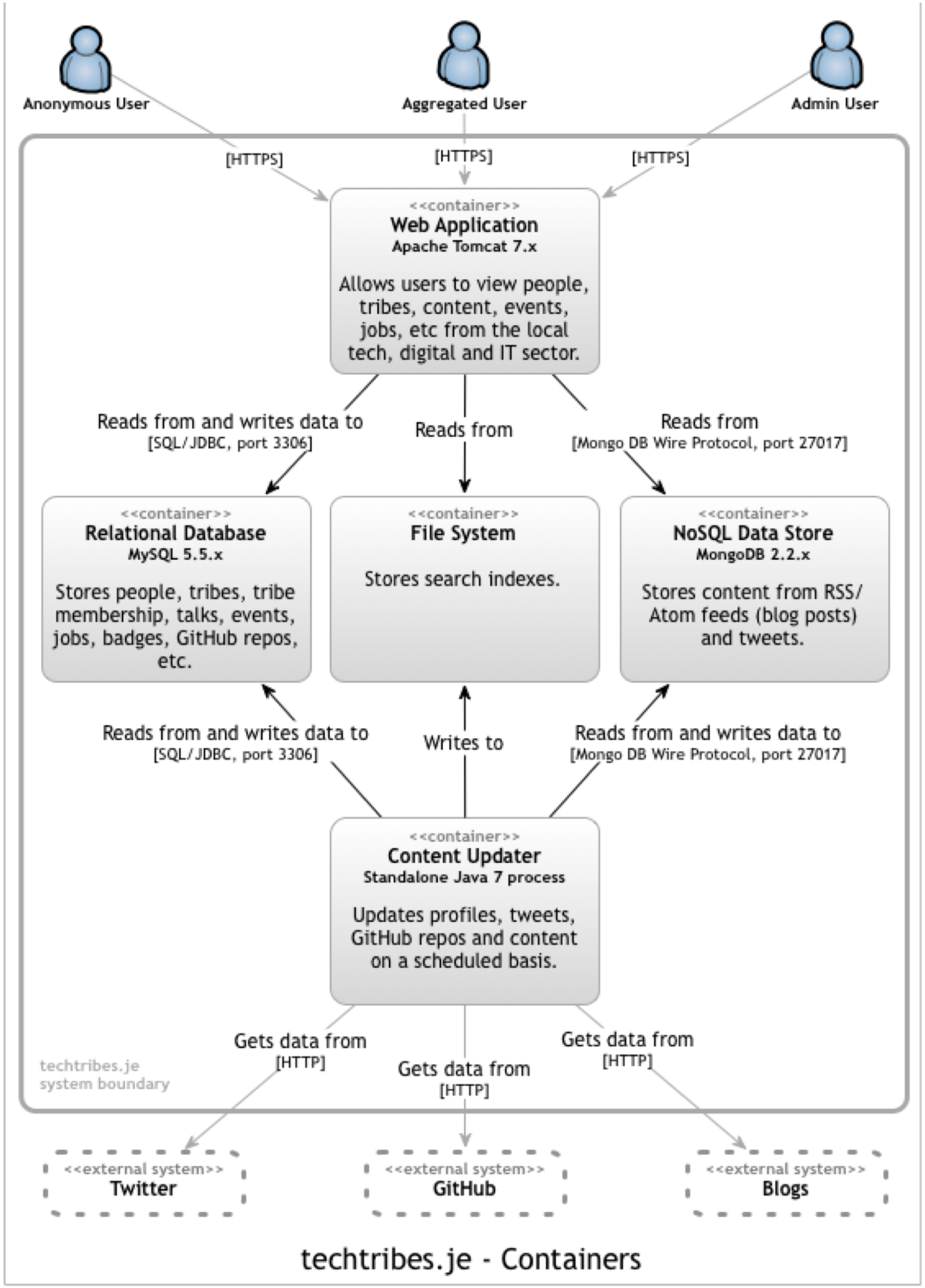
<<external system>>
Twitter

<<external system>>
GitHub

<<external system>>
Blogs

techtribes.je - Context

Containers



<<container>>
Web Application
Apache Tomcat 7.x
Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.

<<container>>
Relational Database
MySQL 5.5.x
Stores people, tribes, tribe membership, talks, events, jobs, badges, GitHub repos, etc.

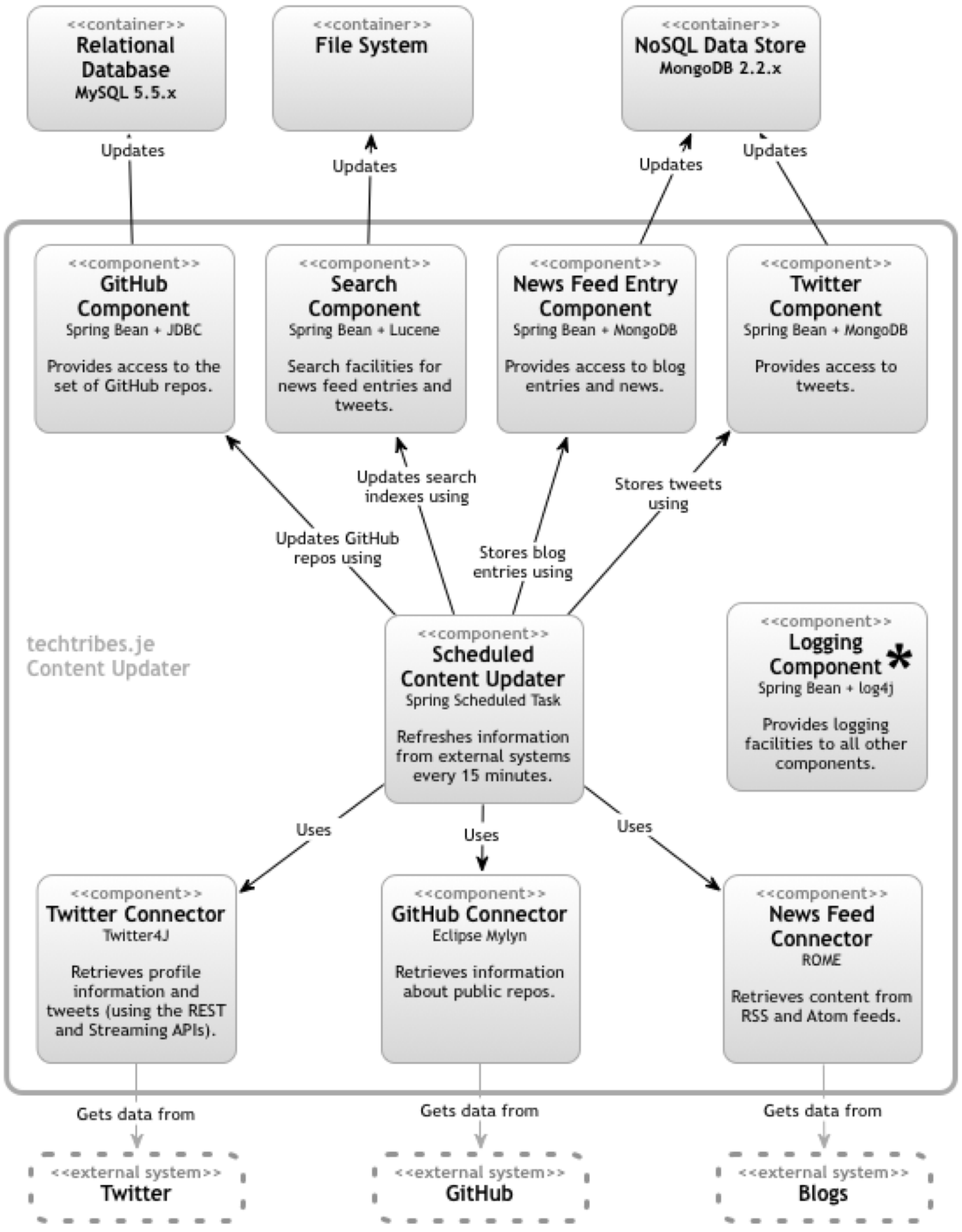
<<container>>
File System
Stores search indexes.

<<container>>
NoSQL Data Store
MongoDB 2.2.x
Stores content from RSS/ Atom feeds (blog posts) and tweets.

<<container>>
Content Updater
Standalone Java 7 process
Updates profiles, tweets, GitHub repos and content on a scheduled basis.

techtribes.je - Containers

Components



<<container>>
Relational Database
MySQL 5.5.x

<<container>>
File System

<<container>>
NoSQL Data Store
MongoDB 2.2.x

Updates

Updates

Updates

Updates

<<component>>
GitHub Component
Spring Bean + JDBC
Provides access to the set of GitHub repos.

<<component>>
Search Component
Spring Bean + Lucene
Search facilities for news feed entries and tweets.

<<component>>
News Feed Entry Component
Spring Bean + MongoDB
Provides access to blog entries and news.

<<component>>
Twitter Component
Spring Bean + MongoDB
Provides access to tweets.

Updates search indexes using

Stores tweets using

Updates GitHub repos using

Stores blog entries using

techtribes.je
Content Updater

<<component>>
Scheduled Content Updater
Spring Scheduled Task
Refreshes information from external systems every 15 minutes.

<<component>>
Logging Component *
Spring Bean + log4j
Provides logging facilities to all other components.

Uses

Uses

Uses

<<component>>
Twitter Connector
Twitter4J
Retrieves profile information and tweets (using the REST and Streaming APIs).

<<component>>
GitHub Connector
Eclipse Mylyn
Retrieves information about public repos.

<<component>>
News Feed Connector
ROME
Retrieves content from RSS and Atom feeds.

Gets data from

Gets data from

Gets data from

<<external system>>
Twitter

<<external system>>
GitHub

<<external system>>
Blogs

techtribes.je - Components - Content Updater
Standalone Java Process

* Used by all components

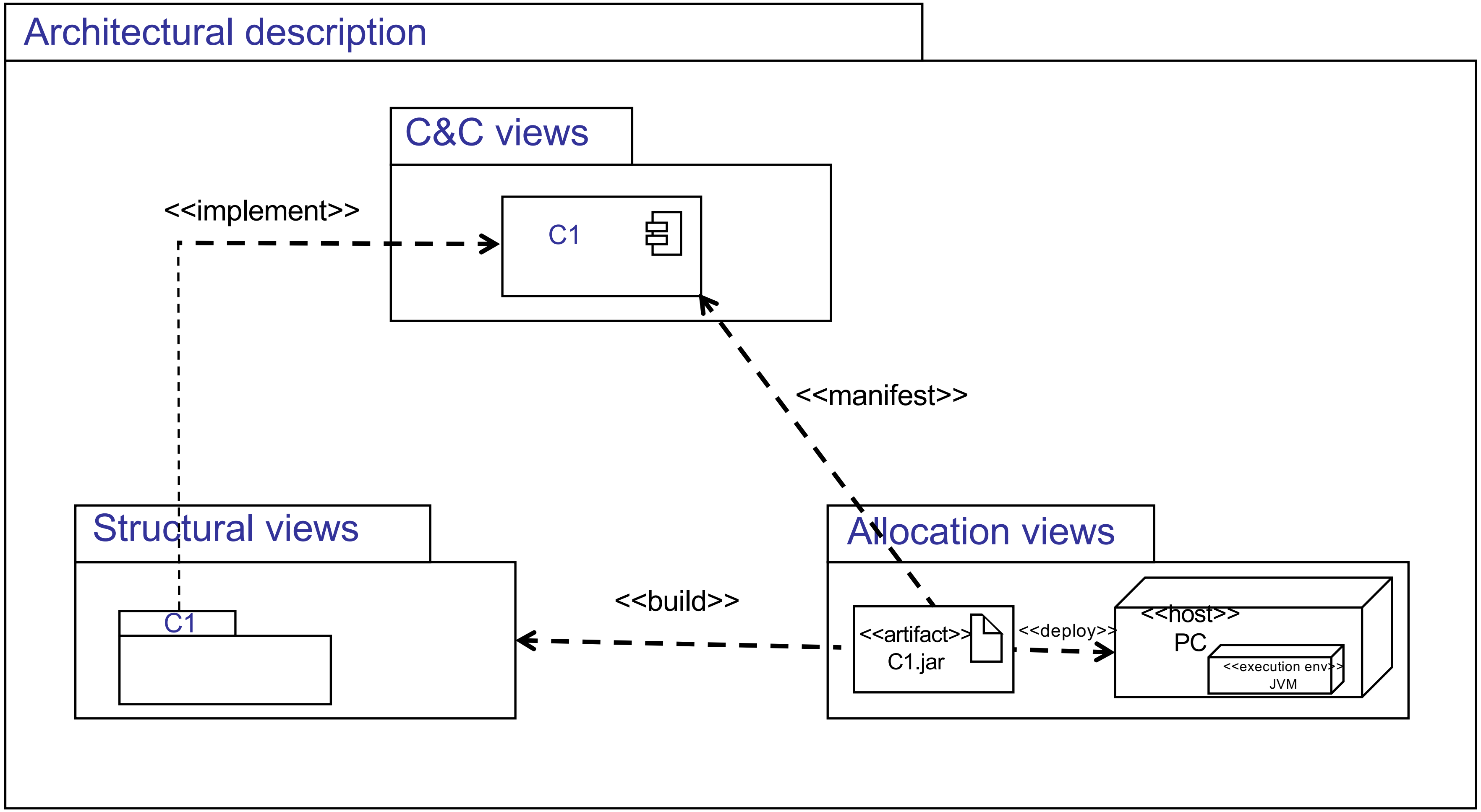
Clements and Bass approach

- **Module** views: organization of source code
- **Components-and-connectors** views: description of the main functional parts of a system and of their dependencies
- **Allocation** views: mapping between software and physical components

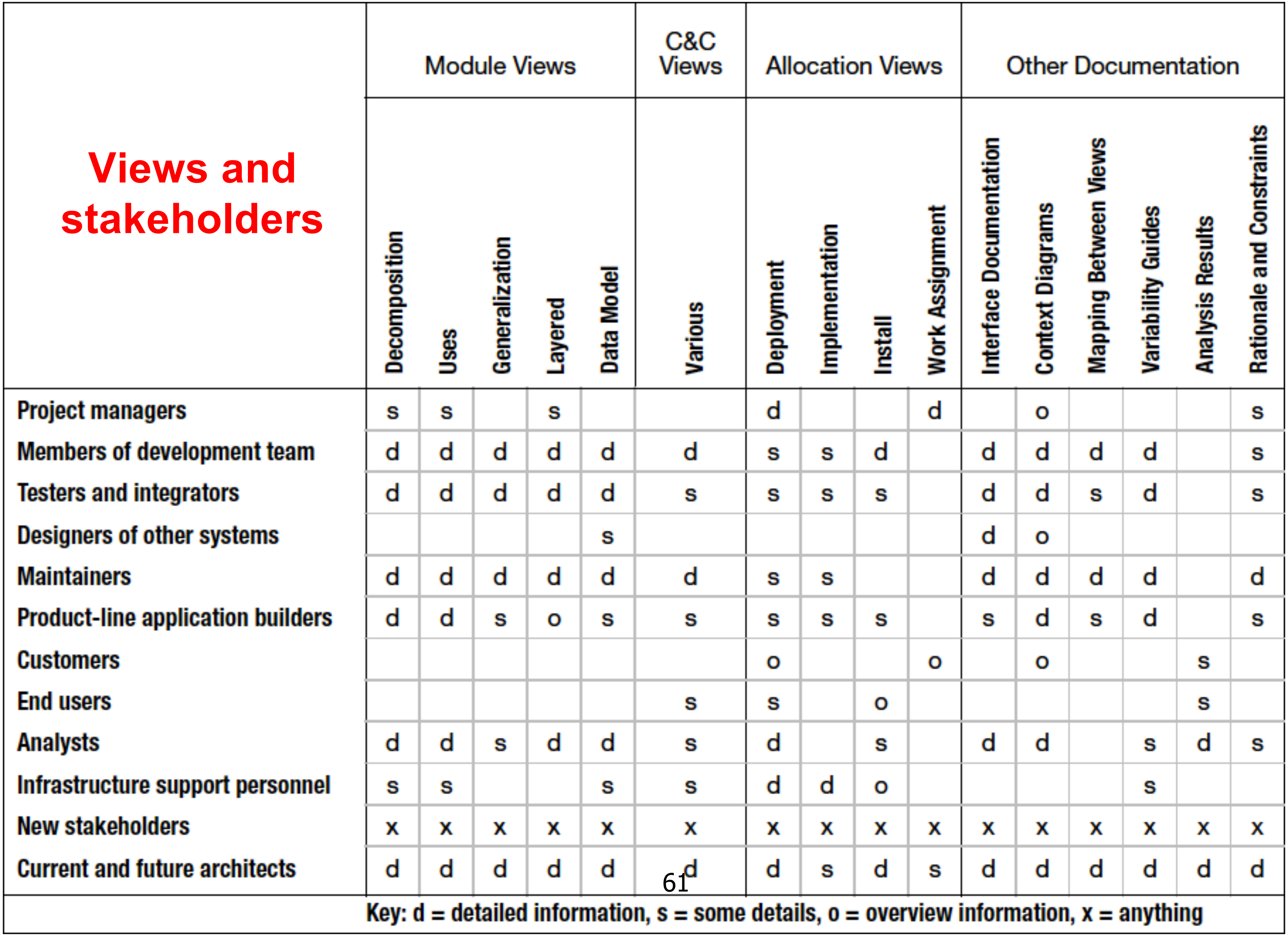
wiki.sei.cmu.edu/sad

Dependencies among views

(Clements and Bass)



Views and stakeholders



	Module Views					C&C Views	Allocation Views				Other Documentation					
	Decomposition	Uses	Generalization	Layered	Data Model	Various	Deployment	Implementation	Install	Work Assignment	Interface Documentation	Context Diagrams	Mapping Between Views	Variability Guides	Analysis Results	Rationale and Constraints
Project managers	s	s		s			d			d		o				s
Members of development team	d	d	d	d	d	d	s	s	d		d	d	d	d		s
Testers and integrators	d	d	d	d	d	s	s	s	s		d	d	s	d		s
Designers of other systems					s						d	o				
Maintainers	d	d	d	d	d	d	s	s			d	d	d	d		d
Product-line application builders	d	d	s	o	s	s	s	s	s		s	d	s	d		s
Customers							o			o		o			s	
End users						s	s		o						s	
Analysts	d	d	s	d	d	s	d		s		d	d		s	d	s
Infrastructure support personnel	s	s			s	s	d	d	o					s		
New stakeholders	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Current and future architects	d	d	d	d	d	d	d	s	d	s	d	d	d	d	d	d

Key: d = detailed information, s = some details, o = overview information, x = anything

Conclusions

Views can be organized in three different *categories*:

- **Module**: module views represent structures including units of functionality as elements. E.g. module decomposition and the class inheritance view
- **Component-and-connector**: represent structures containing runtime elements such as processes and threads. E.g. concurrency and communication processes views
- **Allocation**: represent structures involving assignment relations. E.g. In a deployment structure, software elements are assigned to hardware elements

Summary

- Creating a software architecture starts from non functional qualities, then we deal with features and functions
- Different architectural theories prescribe different views and approaches to the description of software architectures
- Software architecture is a matter of social consensus, its description has the goal to facilitate the consensus

Self test

- What is an architectural view?
- Which are the main approches to describing a software architecture?
- What is the C2 view?
- Describe the approach by Rozanski and Woods
- Describe some dependencies among views

References

- Bass, *Sw architecture in practice*, 3ed. AW 2013
- Rozanski, *Software Systems Architecture*, 2ed. AW 2012
- Bass, *Documenting Software Architectures*, 2ed. AW 2010
- Taylor, *Foundations of Software Architecture*, Wiley 2009
- Spinellis, *Beautiful Architecture*, O’Reilly 2009

Examples of software architecture models

- www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm
- aosabook.org/en/index.html Architecture of open source systems
- delftswa.github.io Delft students on Software architecture

Sites

- www.sei.cmu.edu/architecture
- wiki.sei.cmu.edu/sad
- www.iso-architecture.org
- www.viewpoints-and-perspectives.info
- www.softwarearchitectureportal.org
- aosabook.org/en/index.html
- enterprise-architecture-wiki.nl
- www.bredemeyer.com/definiti.htm
- www.booch.com/architecture
- www.ivencia.com/index.html?/softwarearchitect/
- stal.blogspot.com
- softwarearchitecturezen.blogspot.com
- www.gaudisite.nl

Questions?