

# Chapter 5



Prof. Dr. Harald Störrle  
Danmarks Tekniske Universitet (DTU)

---

## Chapter 5: System Structure Modeling

---

DTU course 02264

## Abstract

- **Every system is embedded in a context that imposes massive constraints. In this chapter we will look at the context of systems and projects from different perspectives:**
  - the structure and culture of the target organization that will use the system under construction, and its processes and rules affected by the system, and
  - the technical structures in existence, i.e., a landscape of existing systems and technologies which compete and/or support the system under construction.
- **In this chapter we will refine our domain architecture models structurally, that is, we will create a tree of nested sub-systems under the main system or systems, and we will connect these systems/sub-systems with point-to-point connections over so-called ports.**
- **In order to make this model more rich and more concrete, we will also specify the behaviors the connected systems expose via their ports.**

## Contents

1. **System Context, Domain Architecture and Structural Decomposition**
2. **Level 2 –Context Diagrams**
3. **Level 3a – Domain Architecture**
4. **Level 3b – System Structure**
5. **Level 3c – System Topology**



Prof. Dr. Harald Störrle  
Danmarks Tekniske Universitet (DTU)

---

# **Chapter 5.1:**

## **System Context, Domain Architecture and Structural Decomposition**

---

DTU course 02264

# Top-Down Decomposition

- **Large and not-so-large corporations use a large number of software applications today.**
  - It is not uncommon to see catalogs of several hundreds applications, and there are reports of companies that use in excess of 2000 applications.
- **For such very large systems of systems (SoS), the main problem is getting and maintaining an overview over the many systems.**
  - The same problem arises again for each individual system, on a smaller scale.
- **Probably the only way to handle this degree of complexity is a layered top-down decomposition.**
  - Do not confuse this with the “Structured Programming” and SADT-style “Structured Methodologies” approaches advocated in the 1970’s and 1980’s.
  - There, the (strict) hierarchical structure was applied to very small units where it is both unnecessary and unhelpful.
  - Many of the concepts from SADT et al., though, are still useful.

# DFD-style Context Diagrams

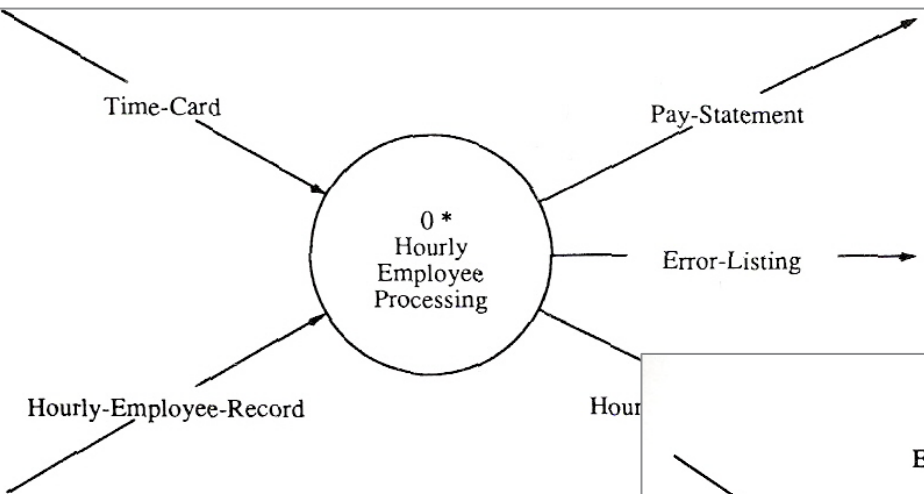


Figure 2-43. Hourly Employee Processing

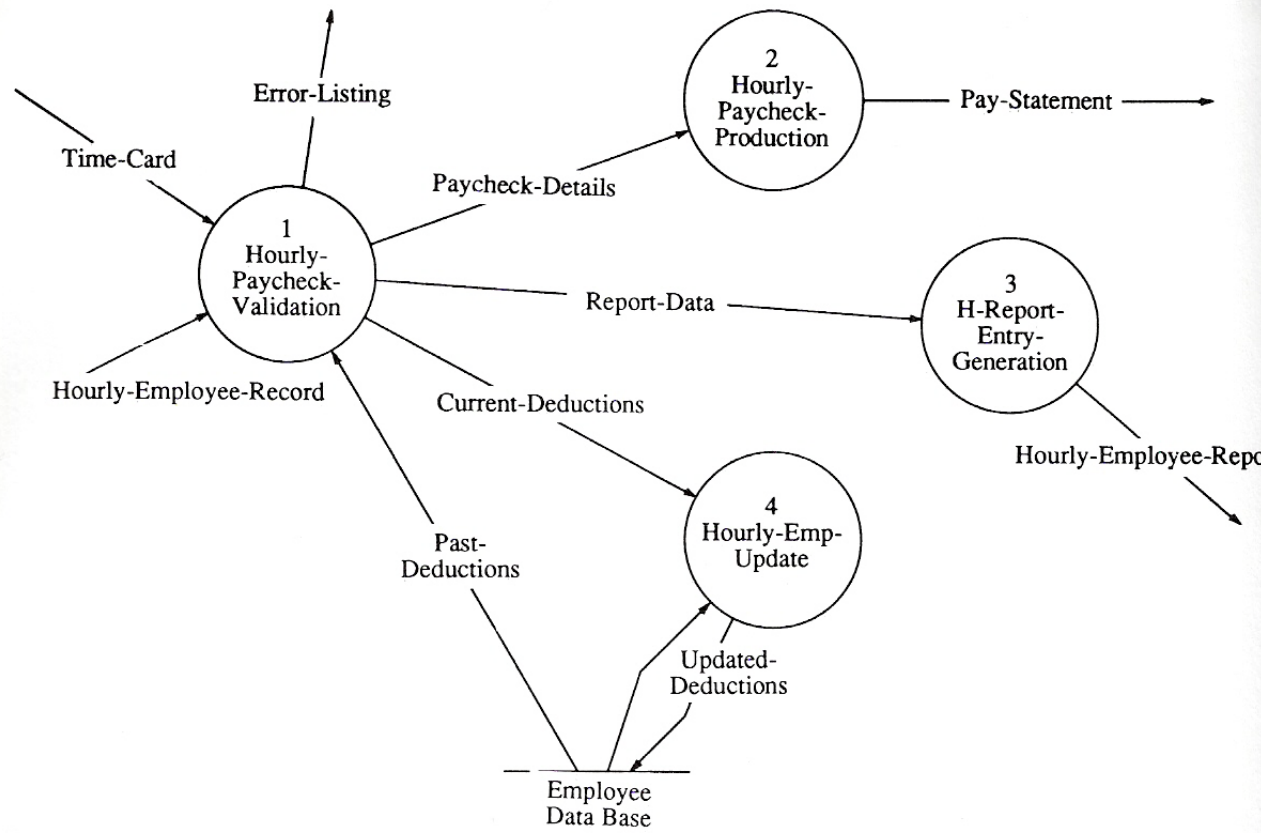


Figure 2-45. The Expanded Hourly Employee Processing DFD.

# SADT-style Context Diagrams

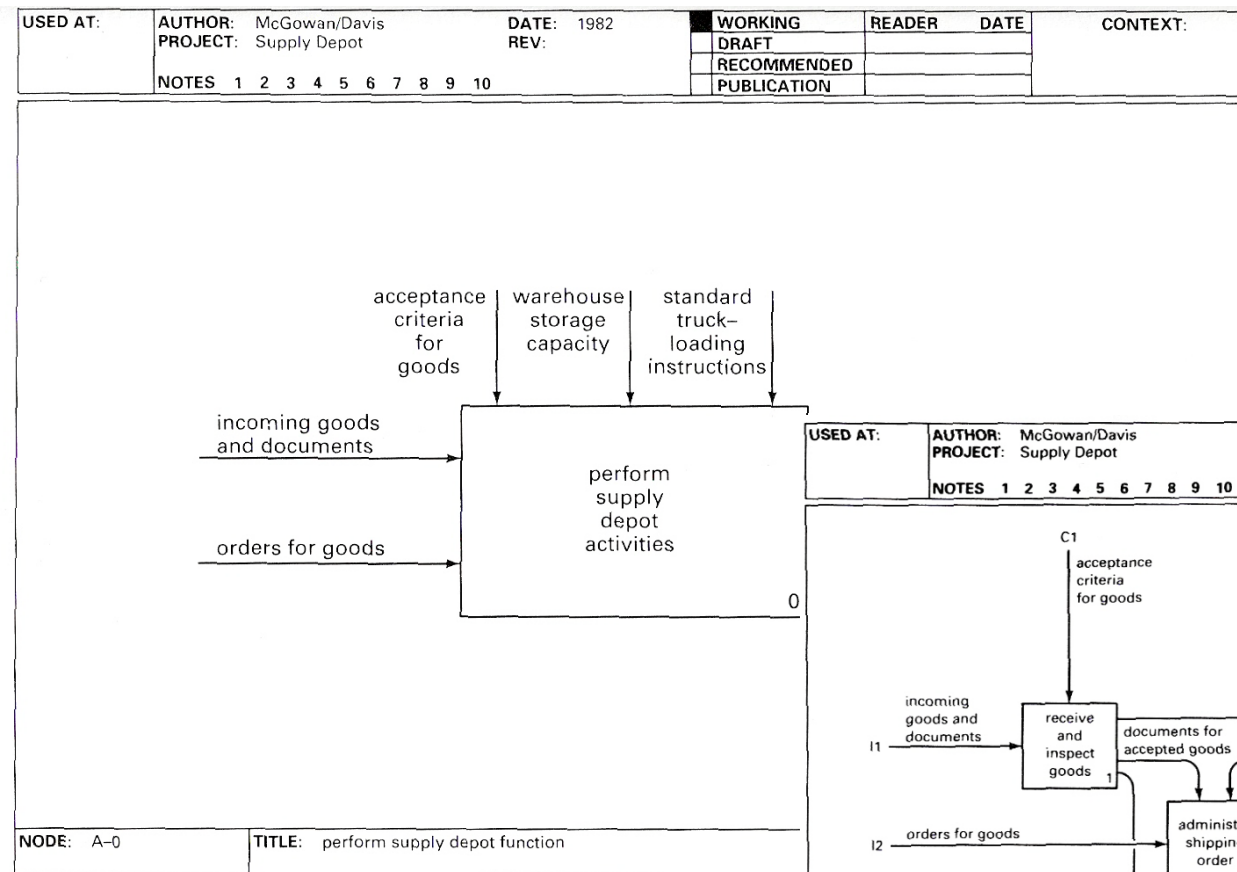


Figure 2-36. SADT Context Diagram: Army Su

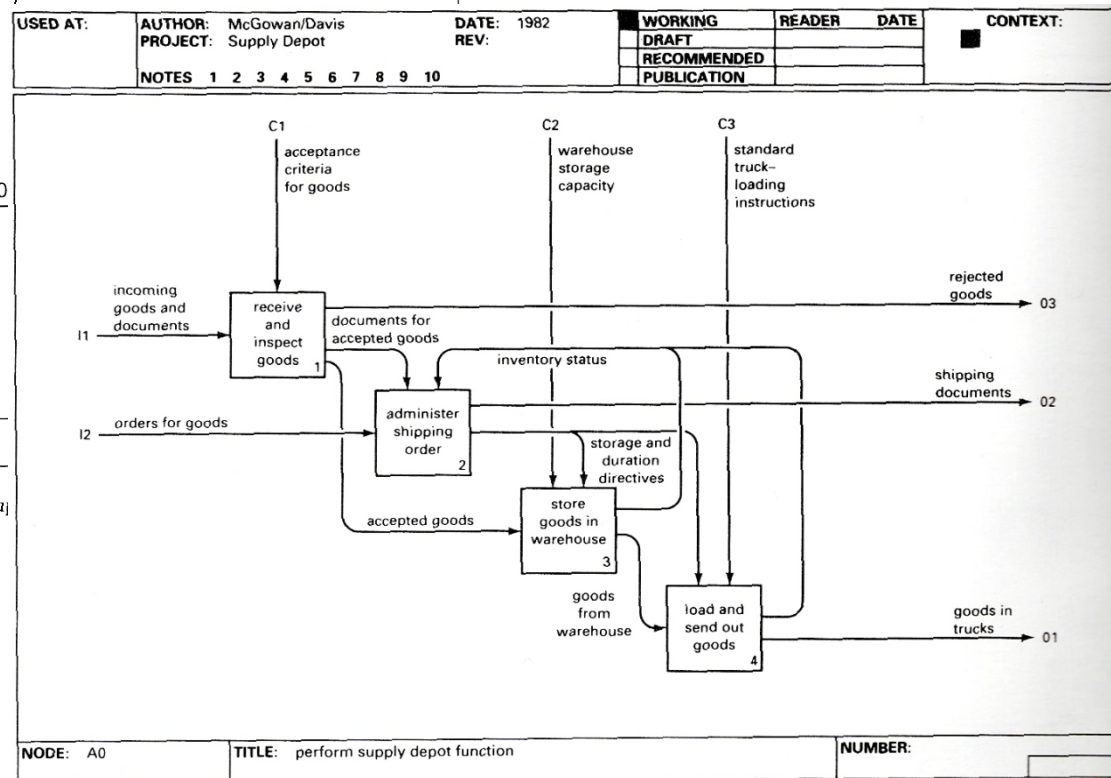
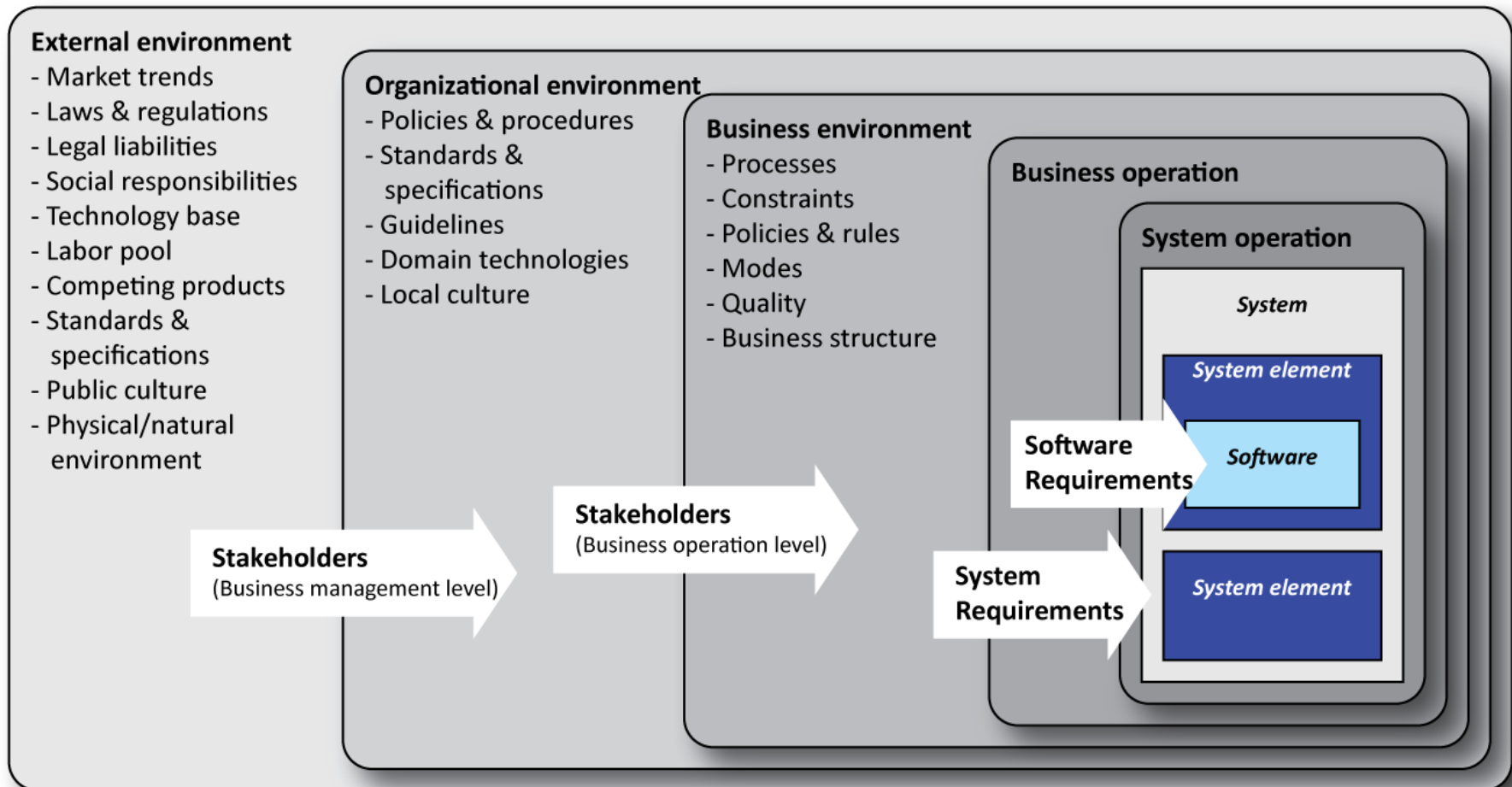


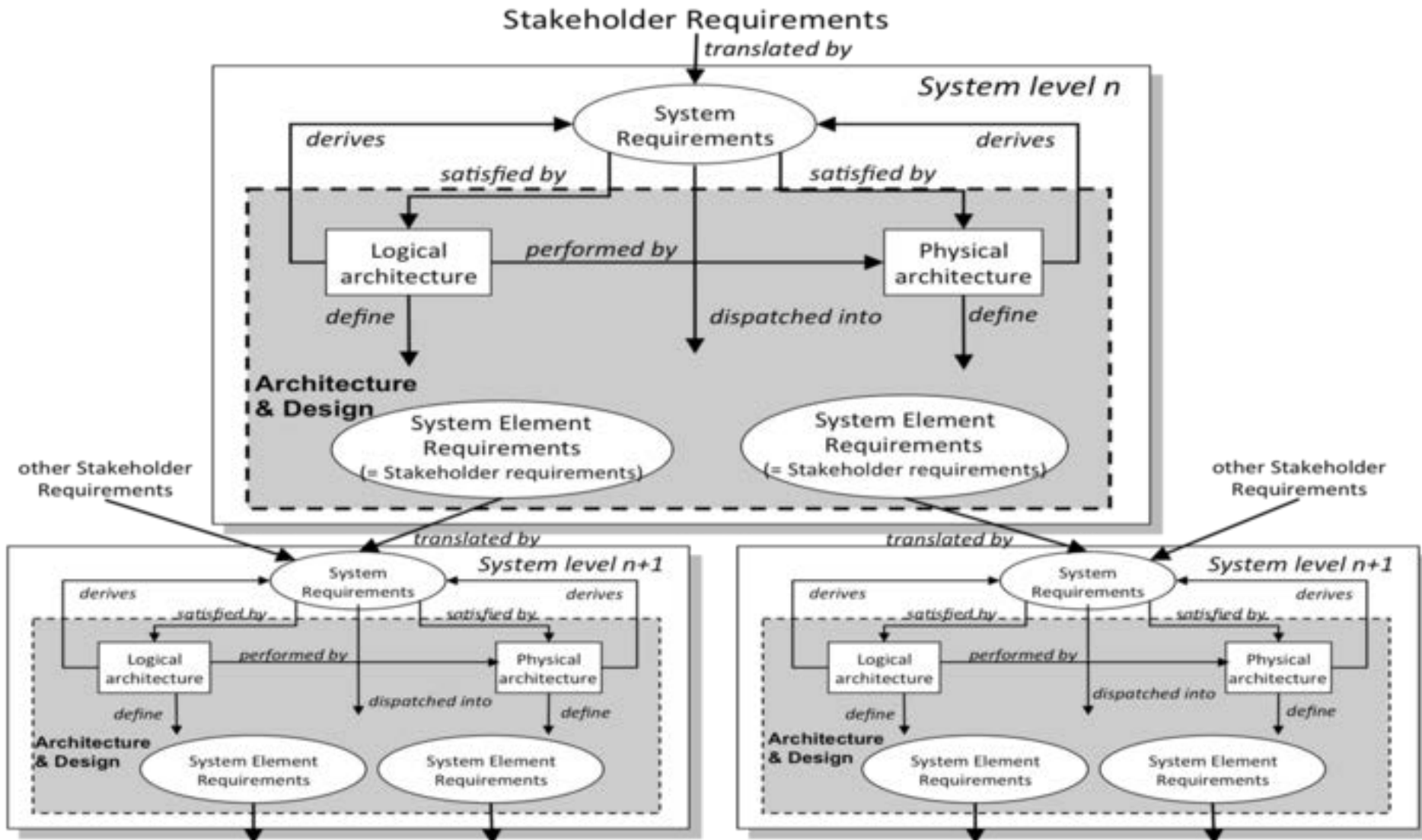
Figure 2-37. SADT Example: Refinement of Army Supply Depot Problem.

# Layered Contexts in ISO 29148

- The notions of system, context, and boundary are relative to each other and apply on several consecutive layers.
  - Consequently, “Stakeholder”, “Goal”, and “Requirement” are relative, too.

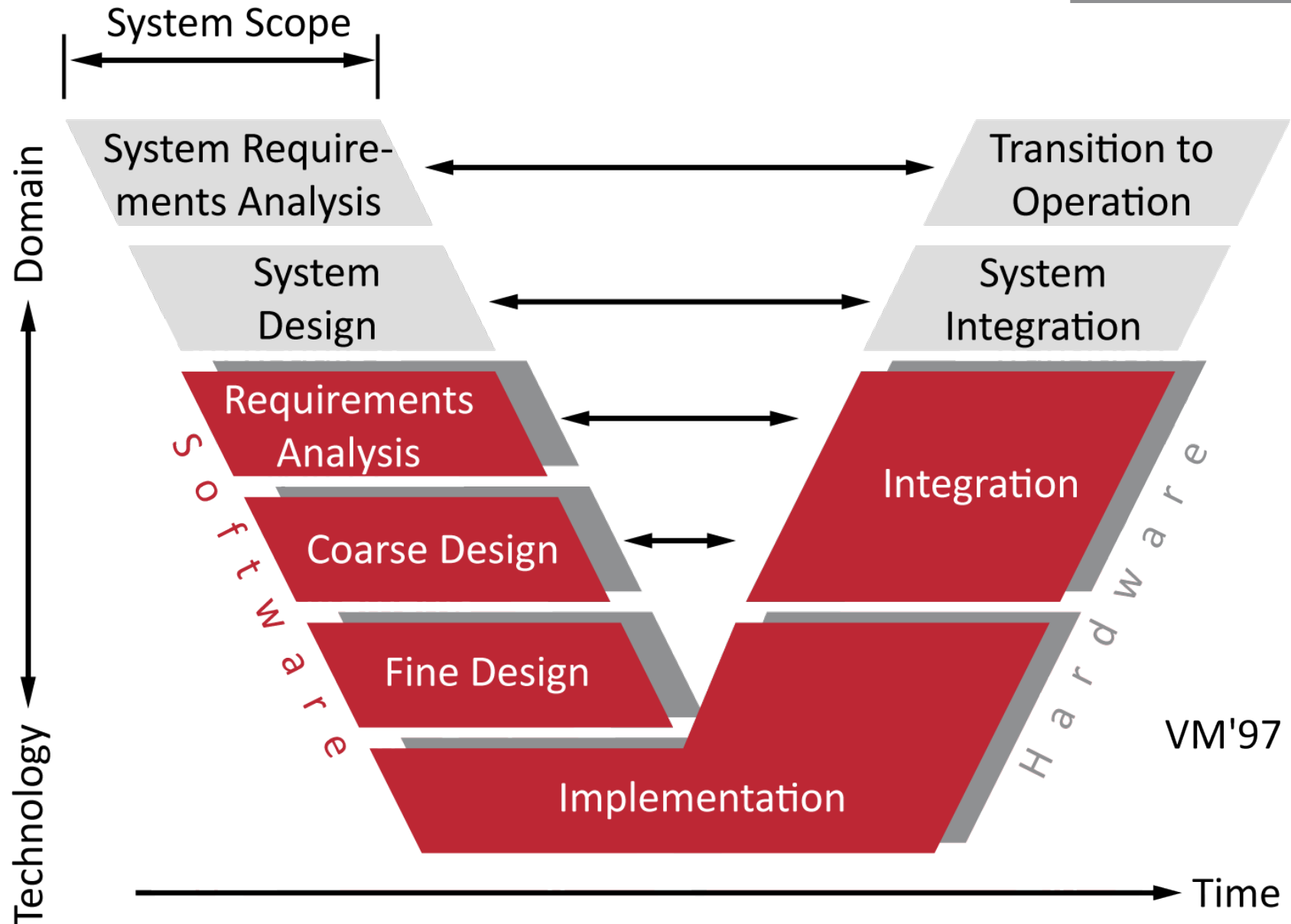


# System decomposition ~ Requirements



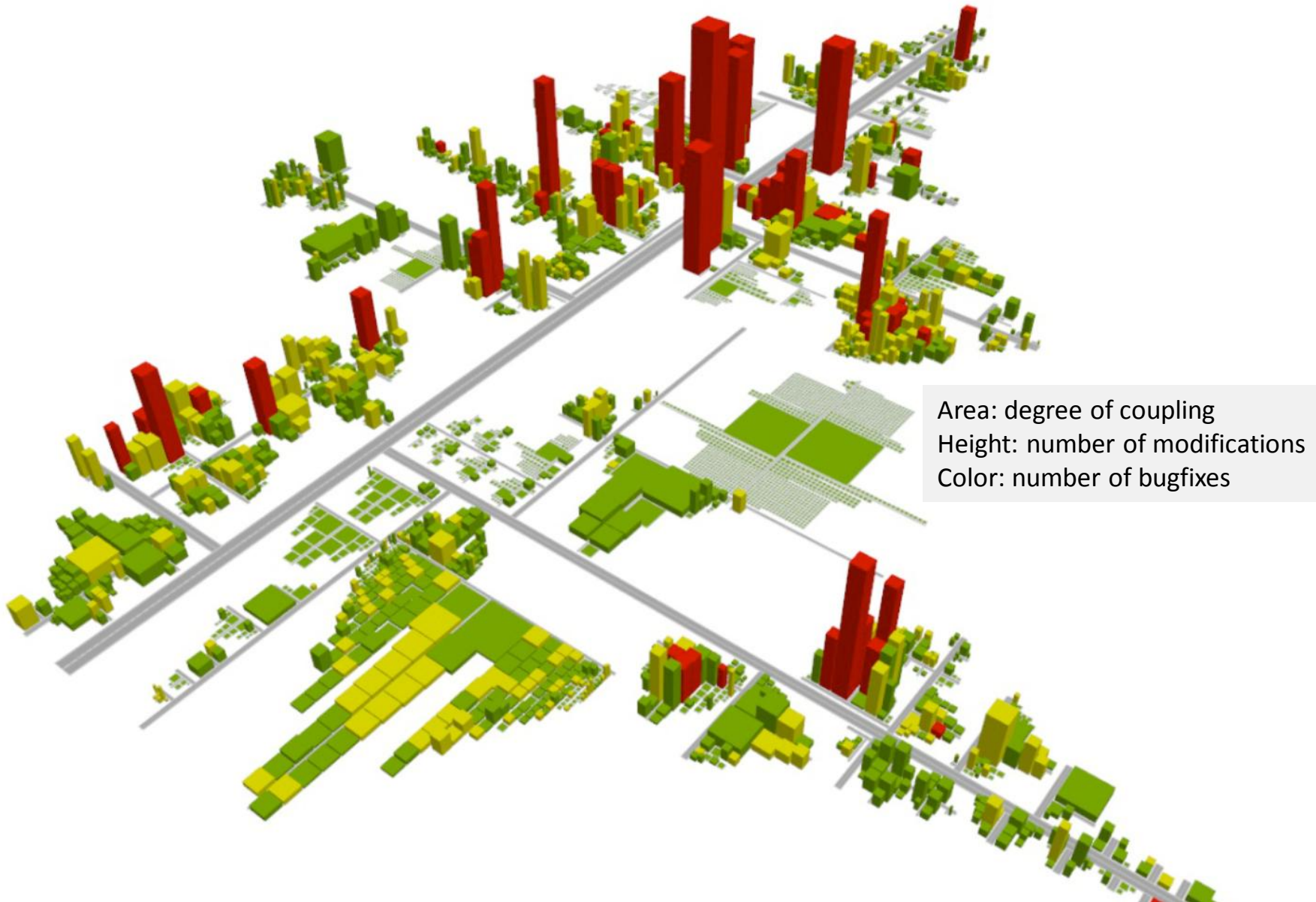


# V-Models



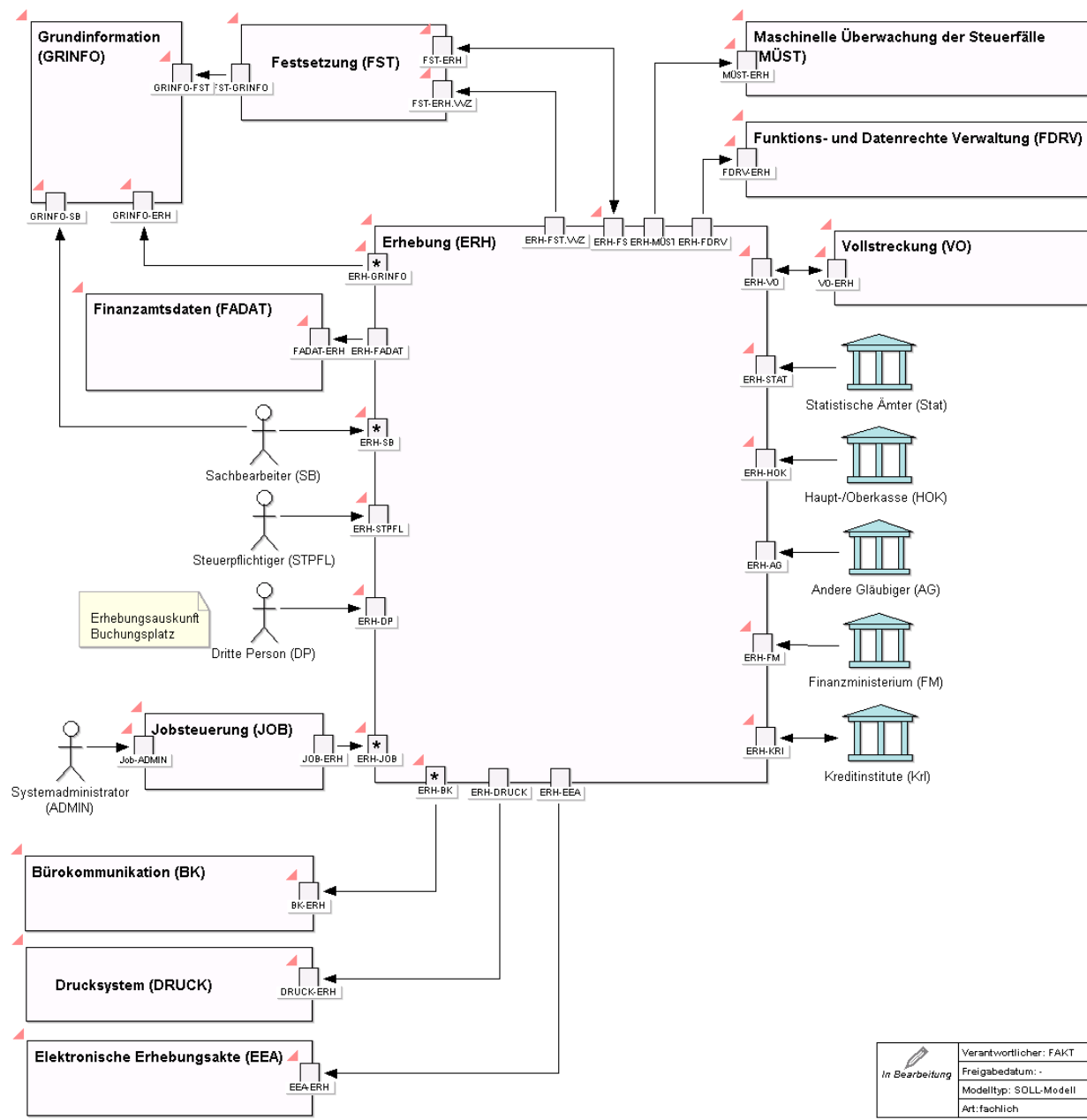
# Application Landscape

# 1

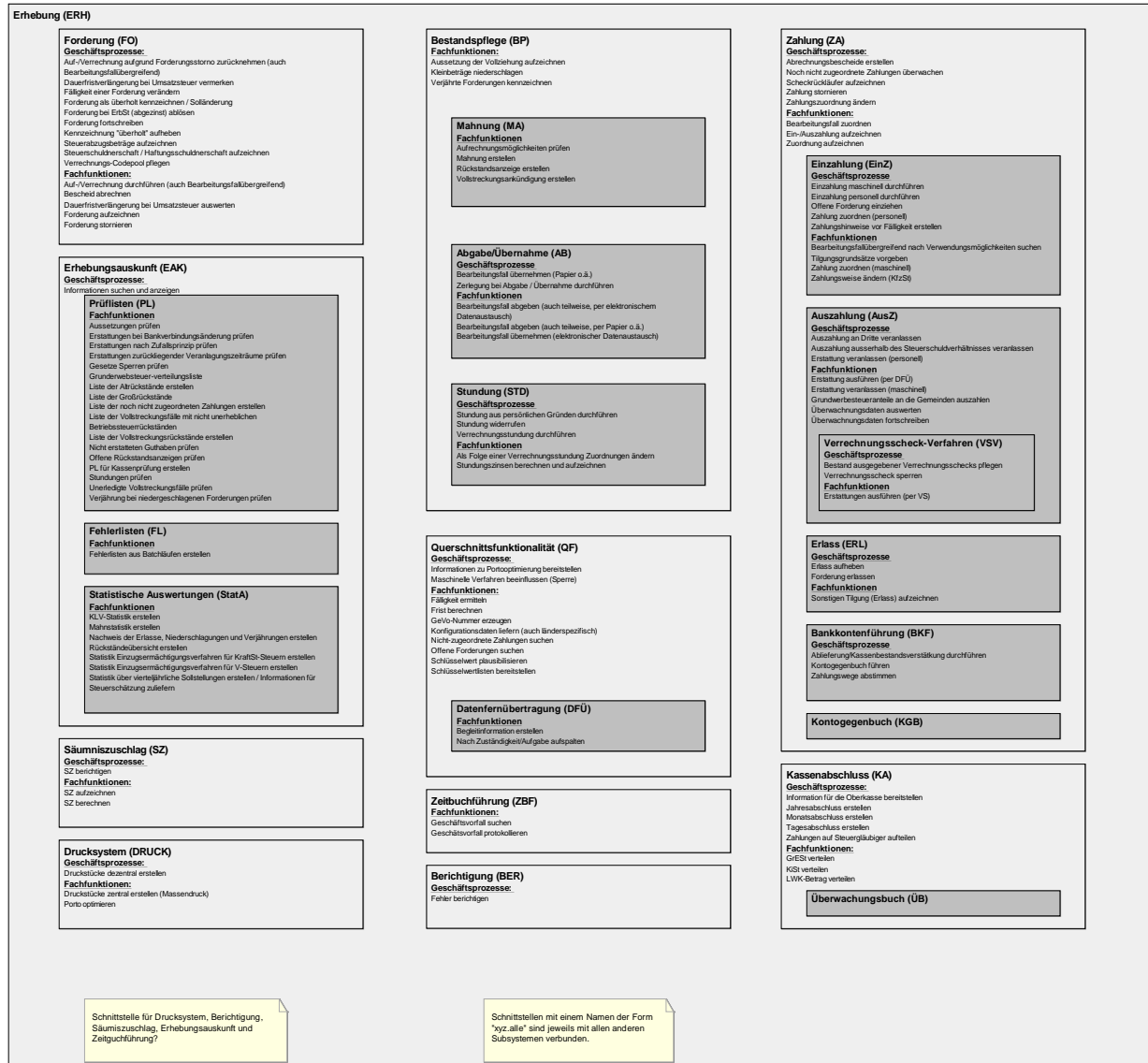


# System Context

# 2

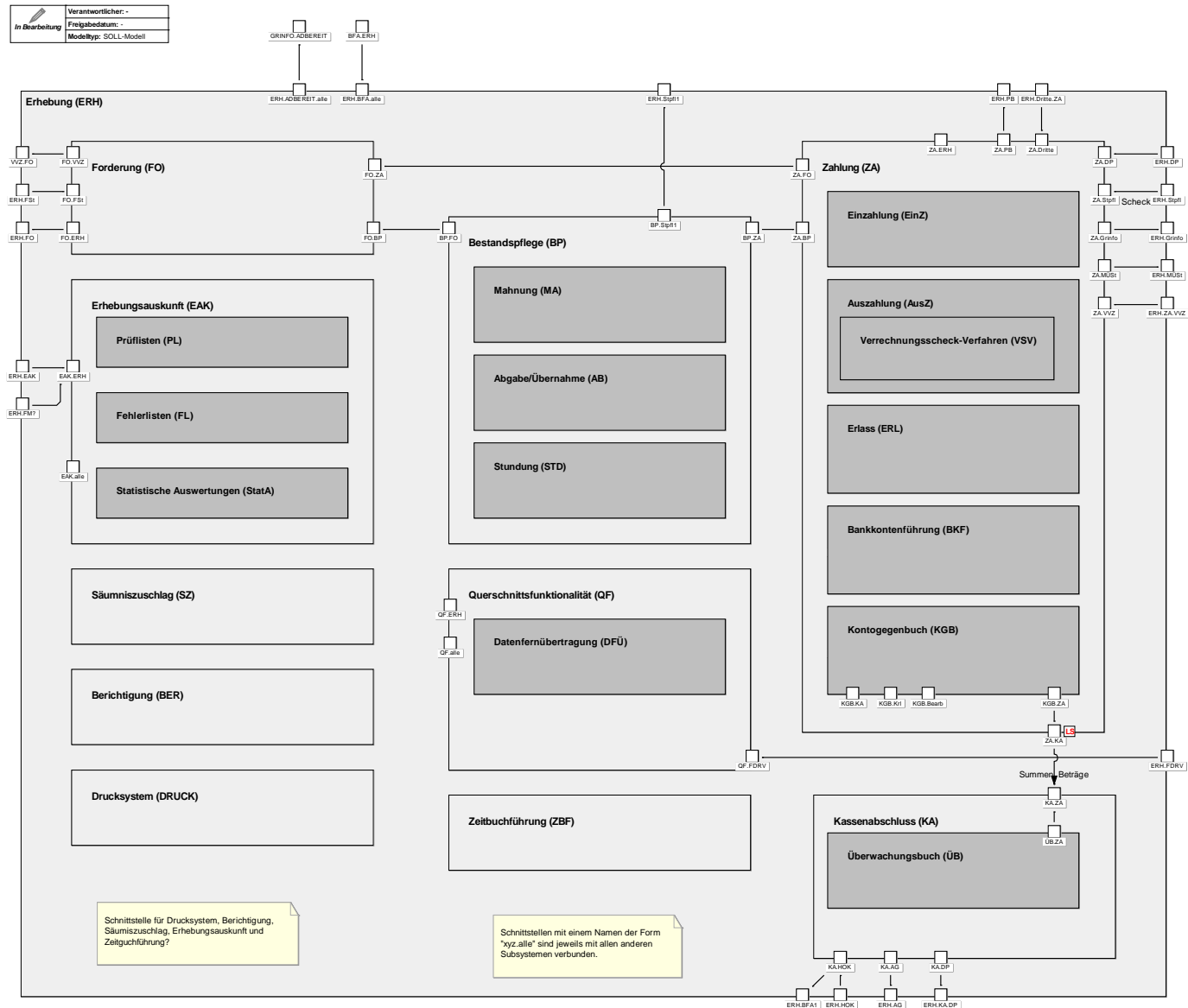


# 3a



# System Structure

# 3b

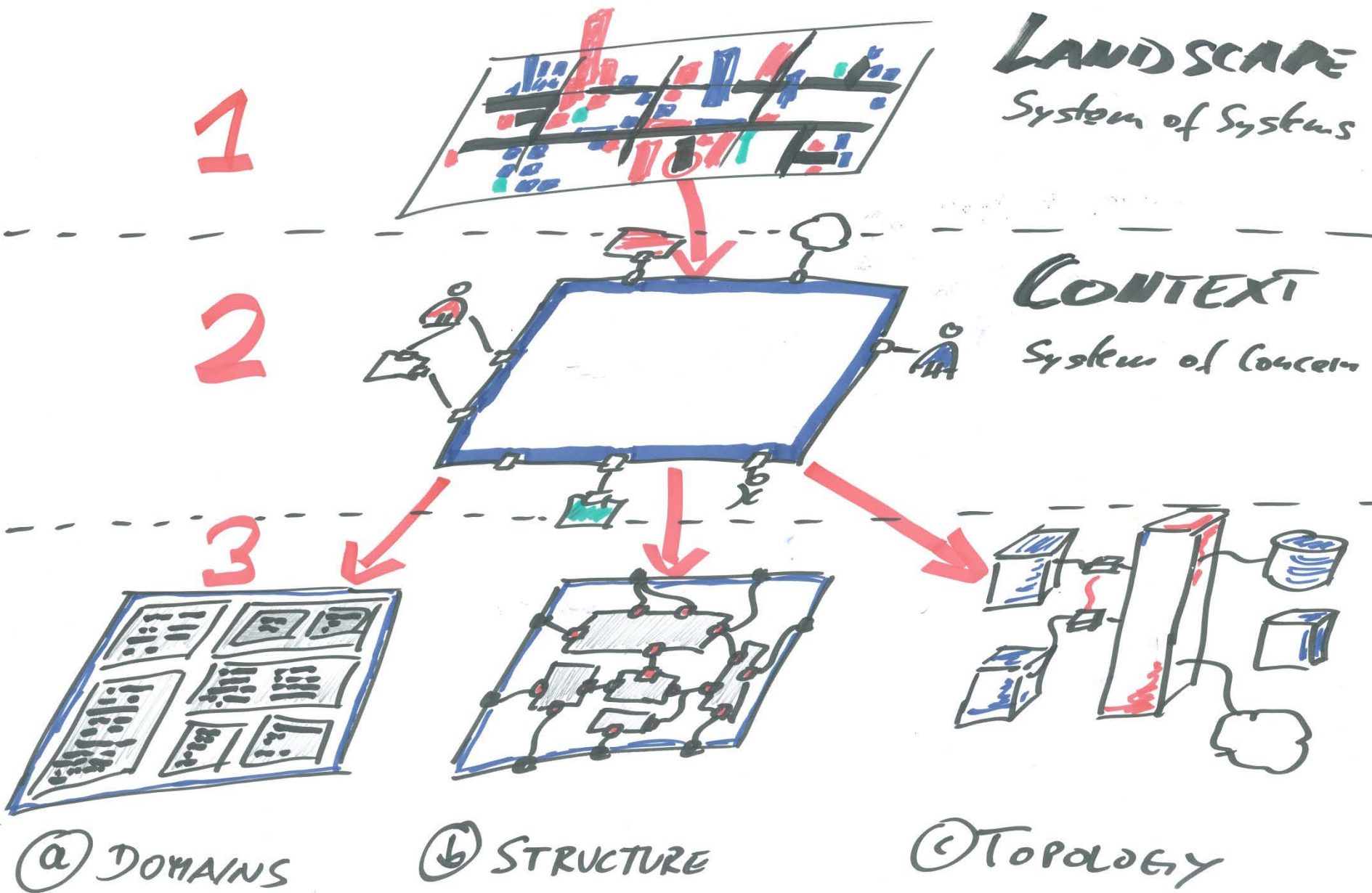


# System Topology

## 3c









Prof. Dr. Harald Störrle  
Danmarks Tekniske Universitet (DTU)

---

## **Chapter 5.2:**

### **Level 2 – Context Diagrams**

---

DTU course 02264



# Context Diagrams

- **A context diagram visualizes the usage-embedding of a given system.**
  - The context refers only to the technical and usage context, not the socio-cultural context of the system.
- **The overall system context details a black box view of the problem domain. It shows**
  - the actors (i.e. roles, organizations),
  - neighboring systems, and
  - their connections with the system.
- **Context diagrams are always normative, that is, they state exactly what is the case, no more and no less.**
  - Anything left out of the context is defined to be not relevant.
  - Leaving out an important actor, neighbor system, or connection may have catastrophic consequences.
- **Context diagrams are so simple and single-minded they may appear to be trivial, but they are also very cheap and fast to create.**
  - There is no situation, no type or size of system where a context diagram is not paying back the cost of creating it.

# Variations and Background

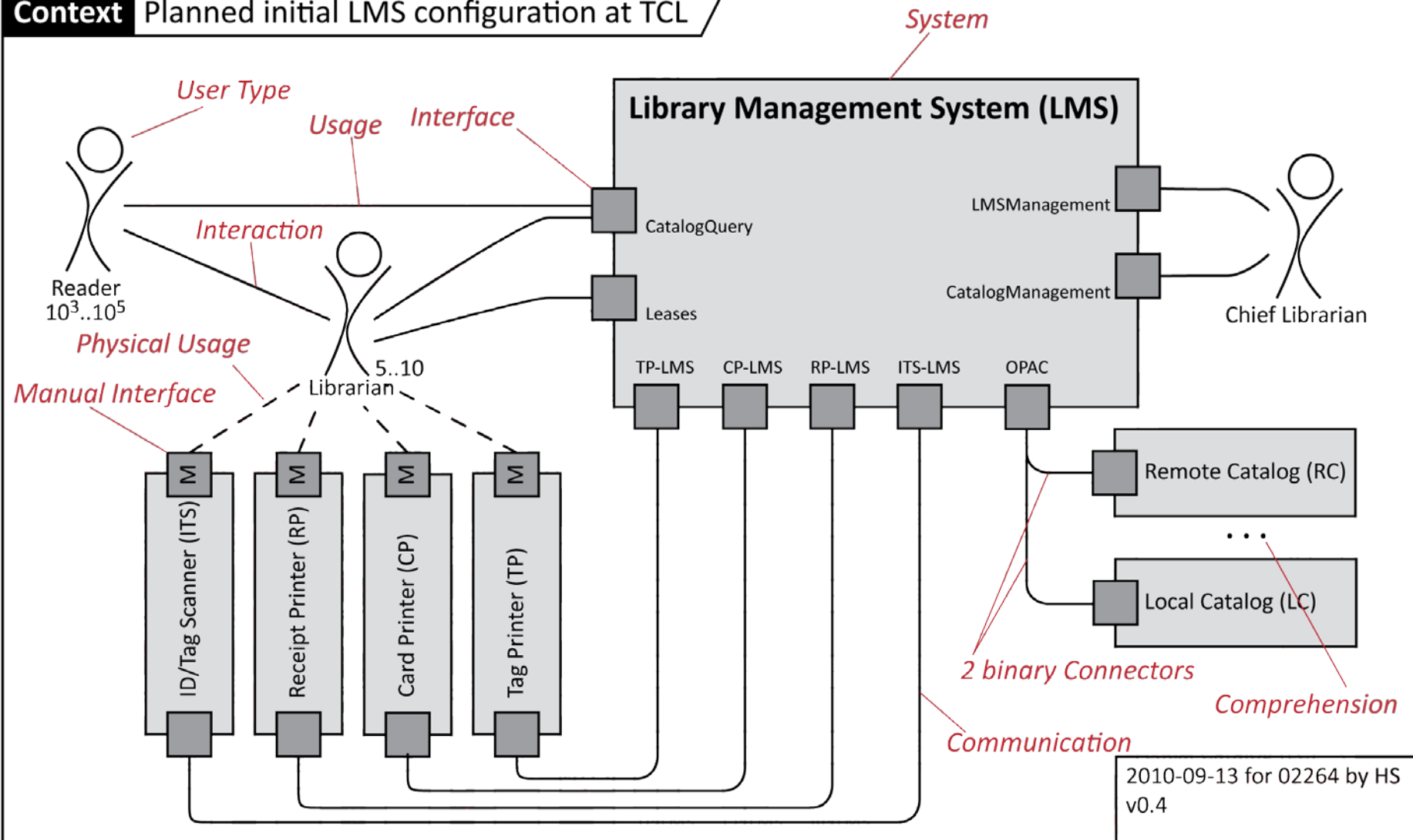
- **Context models are extremely useful in many situations, they are cheap, and they are flexible to allow various additions and extensions.**
  - Context models are indispensable for systems of every size.
  - It is no problem to add e.g. new graphic symbols as long as the standard semantics is maintained.
- **The term context model was introduced with the so-called structured methods of the 1970's.**
  - There, a context was a functional specification of a system.
  - It supported functional decomposition and definition of the system boundary, but helped neither with the structural decomposition, nor the embedding in a system landscape.
- **Today, we frequently deal with systems of systems.**
  - The internal structure of a system of systems is the context of each of its parts.

# Purpose of Context Diagrams

- **Context diagrams (and the associated models) serve many purposes.**
  - **Understanding:** Creating a context is a good starting point for discussing with clients and understanding their domain.
  - **Planning:** A context model may represent the current state of the system configuration, the target state, or one of a sequence of intermediate states.
  - **Embedding:** Today, no system is an island; a context model shows how the system is embedded in a complex ecosystem of applications („system landscape”).
  - **Delimitation:** The context model defines the system border which implies what must be considered and what may be ignored in the project.
  - **System Architecture:** The context can be used to capture very-high level technical decisions that impact the overall system.
  - **Introduction:** It is also a good starting point for presenting the system to management, external consultants/auditors, new team members and so on.
  - **Decomposition:** The context affords several decompositions, including structural, functional, and technical.

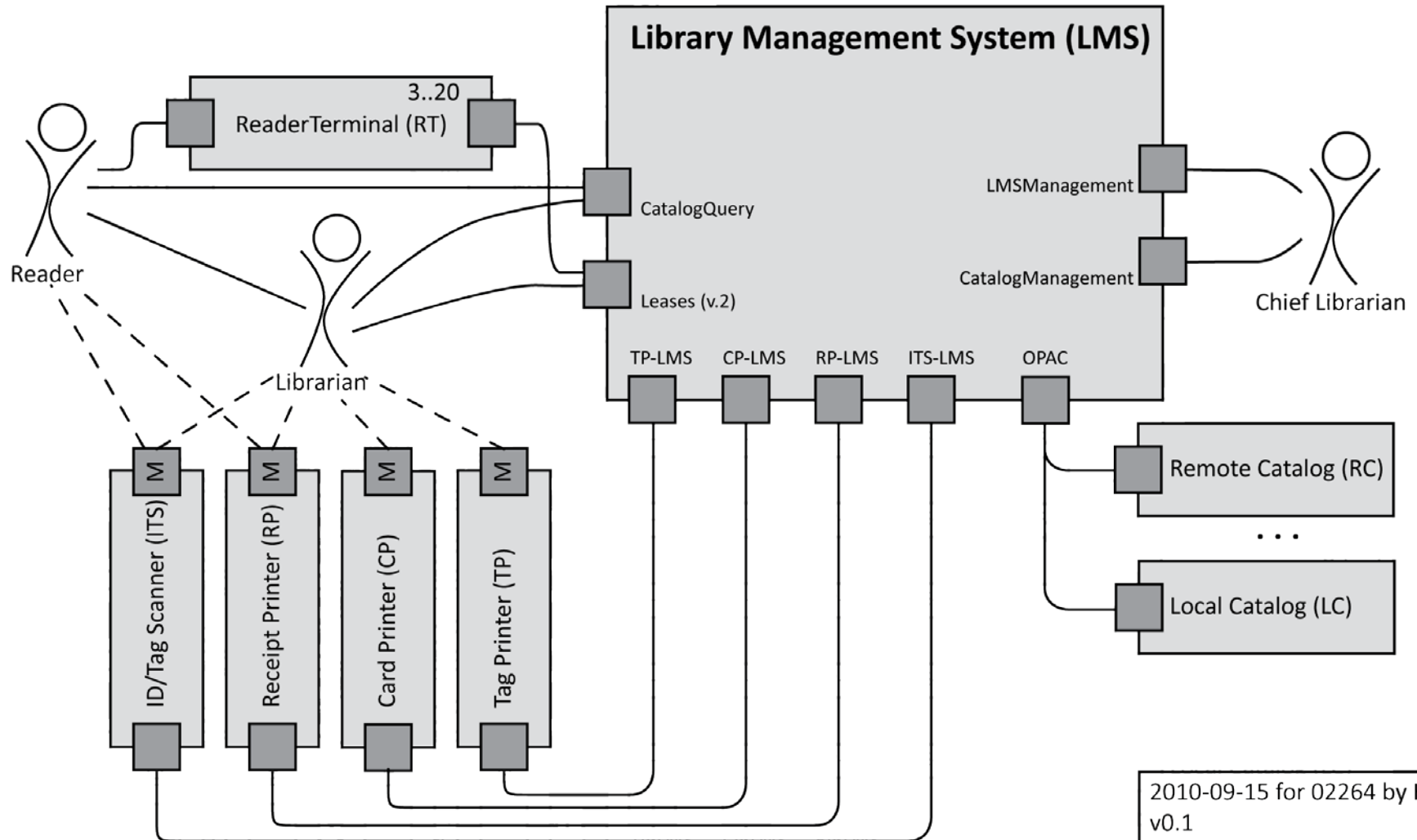
# LMS Context (TCL Stage 1)

**Context** Planned initial LMS configuration at TCL



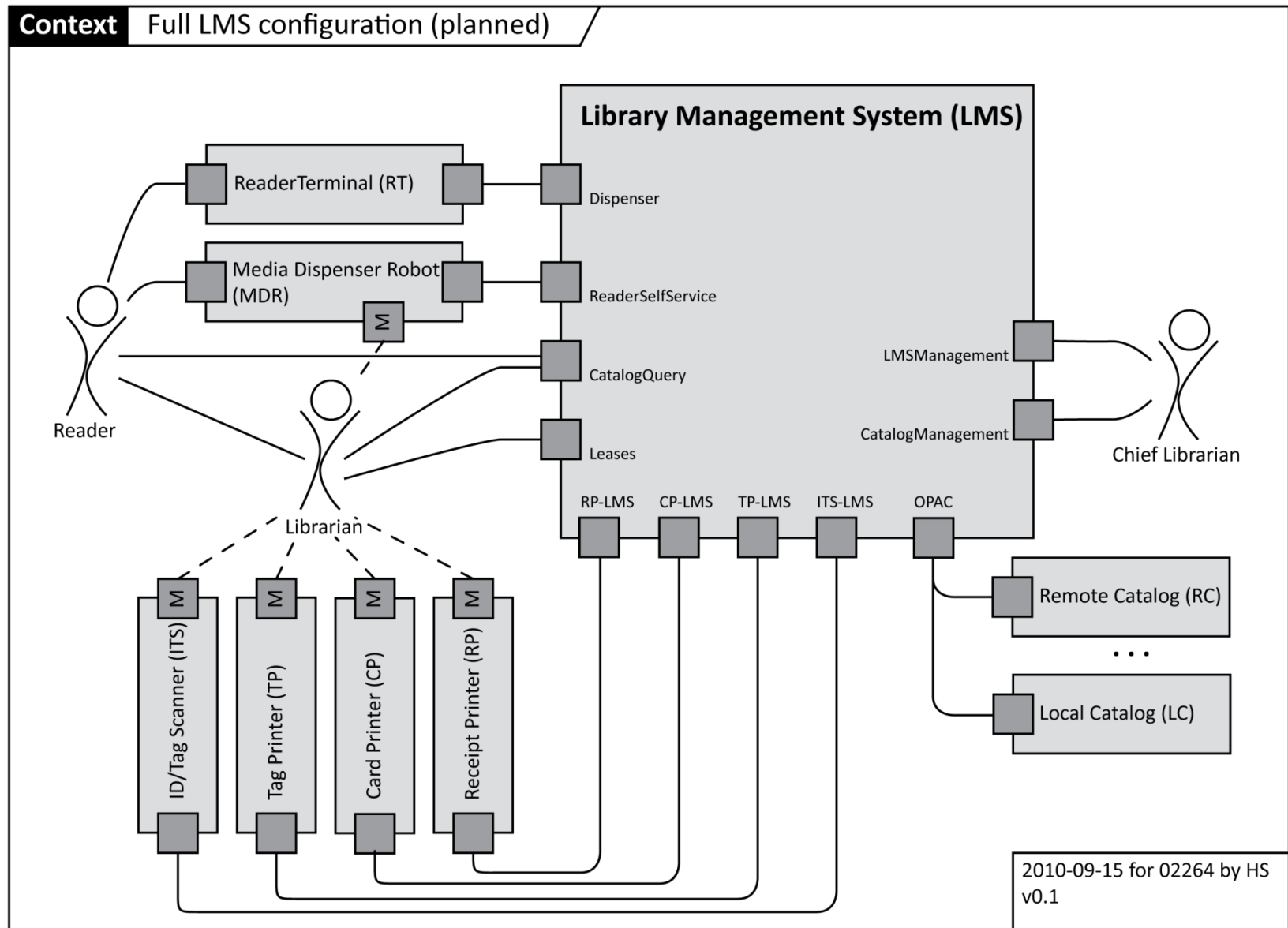
# LMS Context (TCL Stage 2)

**Context** Planned final LMS configuration at TCL



2010-09-15 for 02264 by HS  
v0.1


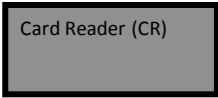
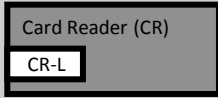
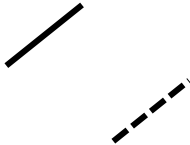

# LMS Context (Stage 3)



# Context Model Problems & Mistakes

- **A good context visualizes your story.**
  - Contexts are used to communicate your system vision and design.
  - Thus, „secondary“ visual properties such as layout, colors, annotations, and icons matter.
  - It is even more important than usual, that each diagram has a type, title, and possibly a legend.
- **Ports represent logical interfaces, not physical ones.**
- **The context model shall be complete and clear.**
- **Structure your model in different layers.**
  - Offer the right level of detail at each layer – which level you chose for which component is part of the story you tell.
  - It is customary to consider the following units (corresponding to levels of abstraction):
    - System of systems
    - System
    - Subsystem
    - Component
    - Function Block

# Syntax of Context Diagrams

Concept	Description	Presentation	Icon
<b>Actor</b>	a role played by a human or set of humans, possibly by whole organizations, e.g. Reader, Librarian, Auxiliary Librarian, Remote Lending Library	a stick person or person shape, a group shape, or a house-shape	
<b>System</b>	(1) the system to be built (2) any source of data and/or events like IT-systems, data bases, paper forms, letters, phone calls and so on. Technically, such systems are actors to the system in sense (1). e.g. Library Management System (LMS), Card Reader (CR)	a rectangle with its name at the top left corner	
<b>Port</b>	A point of interaction of a system, defining the services, signals, and data exchanges there. e.g. CR-L (Interface between Card Reader and Librarian)	a little square on the edge of a system, name inside or floating nearby	
<b>Association</b>	The two entities connected by an association share some interaction, e.g. two people talking, a person entering data to a system, or a system requesting a service from another system. e.g. Reader—Librarian, Card Reader—Librarian	a line between actors and interfaces, maybe a name floating near its middle Dashed lines for manual and/or physical interactions	
<b>Time Event</b>	Reaching a deadline or given date, e.g. monthly catalog feeds	hourglass- or watch-shapes	

- **Alternative symbols or icons for these concepts may be used liberally, as long as it is clear to which concept an entity belongs – adding a legend is a good idea.**
  - Typical examples are the drum-shaped symbol for a database system, a sheet-like symbol for a paper form, a phone symbol for telephone calls and so on.



# Interface Catalog

Neighbor Systems	Message from Neighbor →Reply from LMS	Volume (approx.)	Frequency [per day]	Remarks
Remote Catalog	QUERY → RESULT CONTENTS ← FEED	1KB 1GB..1TB	3/r 1/30	-
ID/Tag Scanner	ID → OKNOK - ← STATUS	0.1KB	8/r	-
Receipt Printer	STATUS ← MSG	1KB	2/r	Prints receipts for readers
Tag Printer	STATUS ← MSG	0.1KB	0.0005 / b	Prints ID tags for physical media
Card Printer	STATUS ← CARD	0.5KB	0.01/r	Prints reader card bodies
Remote Lending Network	CMD → STATUS	1KB	35	-
WebOPAC	QUERY → RESULT	1KB	500	-
ReaderTerminal	CMD → STATUS QUERY → RESULT	2KB	10 · r	-

# Message Catalog

Message	Purpose	Additional information contents
QUERY	query the catalog for media	any subset of media data
RESULT	prioritized list of media matching a query	a list of media identifiers a list of media data
FEED	feeding added or updated catalog data into a remote catalog	a list of media data
ID	a typed id for a medium or reader	-
MSG	a message preformatted for output	depends on type
STATUS	signal system or process status	depends on type
CARD	identify a reader	reader id, name, and status of a reader
CMD	issue a command to a system	depends on type, e.g. BUTTON for a button on a device or NUMBER(n) for a keypad
OKNOK	Confirm or reject preceding action	either OK or NOK

# Data Package Details

name	description / values
<b>Id</b>	unique identifier of the medium (not the title)
<b>DDC</b>	DDC classification code
<b>Author</b>	Last Name: String First Name: String
<b>Title</b>	Main title: String Subtitle: String
<b>Keywords</b>	List of strings
<b>Genre</b>	List of strings
<b>Medium type</b>	Book, Periodical, CD, DVD, BrailleBook, Game, Other
<b>Publication date</b>	Year
<b>publisher</b>	String
<b>lending class</b>	presence: may not be lent at all restricted: may be lent only by some readers (see Restriction field) limited: lease may be extended only a limited number of times unlimited: lease may be extended any number of times
<b>restriction</b>	String, e.g. for an age restriction or a restriction to a particular user class



Prof. Dr. Harald Störrle  
Danmarks Tekniske Universitet (DTU)

---

## **Chapter 5.3:**

### **Level 3a – Domain Architecture**

---

DTU course 02264

# Domain Architecture

- **A domain architecture (or „process landscape“) provides an overview over the business processes a system implements.**
  - It is a table of context or index to an exhaustive catalog of processes or services provided by on in an organization.
- **A domain architecture may serve several purposes.**
  - Being a table of context/index to service catalog, it allows convenient and fast access by casual users, e.g., to get an overview.
  - It depicts a past, present, or future state of the system in terms of the functionality offered and can thus be used for detailed planning and resource allocation (cf. context diagrams).
  - The domain architecture divides the system's set of features and functions into manageable chunks, reflecting the structure of the application domain.
    - Any technical structuring must not play a role, although for physical systems and very large software systems the two usually coincide.
- **For the time being, we shall express domain architecture diagrams as indented text.**
  - This may be complemented by some free-form graphics for improved visualization.
  - Technically, a domain architecture diagram can be expressed as a UML class diagram without properties.

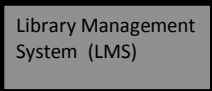
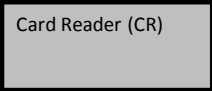
# Domain Architecture Notation

- **If visualization is indeed required, present a domain architecture as a tree of nested boxes, each of which represents a particular area of the business („domain“).**
  - The outermost domain is often called „the system“.
  - Each domain enumerates the processes and functions it contains, identified by their names.
  - Maintain the same terminology for levels that is used for context diagrams.
  - Maintain consistency between the units shown as refinements of a system (or system of systems) in the context diagram and the domain architecture.
- **Domain architectures are allowed to have any number of levels of nesting, though typically one or two are enough.**
  - Only very large scale systems may reach four or five levels.
- **Domains may contain both sub-domains (nested boxes) and processes (lines of text).**
  - If a process uses functionality from to different sub-domains, it should be located at the smallest super-domain containing all the relevant sub-domains.
- **What is not modeled in the domain architecture is not supposed to exist.**

# Domain Architecture Diagrams

- **While the system context diagram focuses on the environment of a system, the domain architecture diagram shows:**
  - the logical parts of the system (and their parts in turn);
  - The functions and/or processes that make up the characteristic behavior of these systems.
- **By defining the functions, the freedom of interpretation of what the systems actually is accomplishing is narrowed down just enough to allow for precise discussions at the domain level.**
  - This is particularly useful for all administrative functions, e.g., scoping, prioritizing, delegation, and project planning.
  - The structure does not necessarily imply technological or implementation divisions, but usually, at the highest levels, the two types of boundaries coincide.
- **Usually, only 2-3 levels of refinement are sensible.**

# Syntax of Domain Architecture Diagrams

Concept	Description	Presentation	Icon
<b>Domain</b> (top-level)	A system	A rectangle with its name at the top left corner. Names are followed by their abbreviations.	
<b>Sub-Domain</b> (level i+1)	A logical (!) part of system, not necessarily a physical part (the further down, the less likely there is a coincidence between logical and physical parts).	A rectangle with its name at the top left corner. Names are usually followed by their abbreviations. Typically, subdomains at lower levels are represented by darker shades of grey.	
<b>Process</b>	Any process or function to be executed by and/or implemented within a given system or subsystem.	The name of the process. Naming conventions apply.	<b>Admit reader</b>

- **Alternative symbols or icons for these concepts may be used liberally, as long as it is clear to which concept an entity belongs – adding a legend is a good idea.**
  - Typical examples are the drum-shaped symbol for a database system, a sheet-like symbol for a paper form, a phone symbol for telephone calls and so on.



# LMS Domain Architecture

## Domains Library Management System (LMS)

### tc!ONLINE

#### Online Lending

media search  
reservation handling  
lease handling

#### Self-Service Lending

authenticate user  
identify medium  
print receipt

#### Digital Media Shelf

get medium  
forward medium

### ReaderTerminal (RT)

#### Self-Service Lending

authenticate user  
identify medium  
print receipt

#### AccountPrinter

identify reader  
show reader account  
print account status

#### Reading & Lending

media search  
reservation handling  
lease handling

### BookStation™

unlock compartment  
lock compartment  
get contents data

## Library Management System (LMS)

### Readers

enlist reader  
unlist reader  
maintain reader profile  
lookup reader  
update reader (manual)  
update reader (batch)  
authenticate reader  
write message  
read message

### Leases & Reservations

reserve medium  
terminate reservation  
lease medium  
prolong lease  
terminate lease  
query leases  
query reservations  
compute fees

### Media Management

acquire medium  
update medium (batch)  
update medium (manual)  
feed catalog

### Catalog

search catalog (local)  
search catalog (remote)  
simple search  
lookup medium

### Reader Accounts

get account status  
display reader account  
set account capabilities  
update account (manual)  
update account  
update deadlines  
record transaction  
find transaction

### BookTip™

add connection  
reorganize suggestions  
ask for suggestion

### Policies

show policies  
set policies  
migrate data base  
maintain tasklist  
export policy

### Wishlist

enter wish  
lookup wish  
show open wishes  
comment wish

### Users

create & maintain user  
lookup user  
authenticate user  
self-reset credentials

### Statistics & Info

report on usage  
report on lending  
report on corpus  
report on readers  
report on staff  
simulate policy

### NewsWire™

send message  
send voicemail  
send letter  
send news  
read news  
create new topic  
write news

2014-03-31 for 02264 & 02341

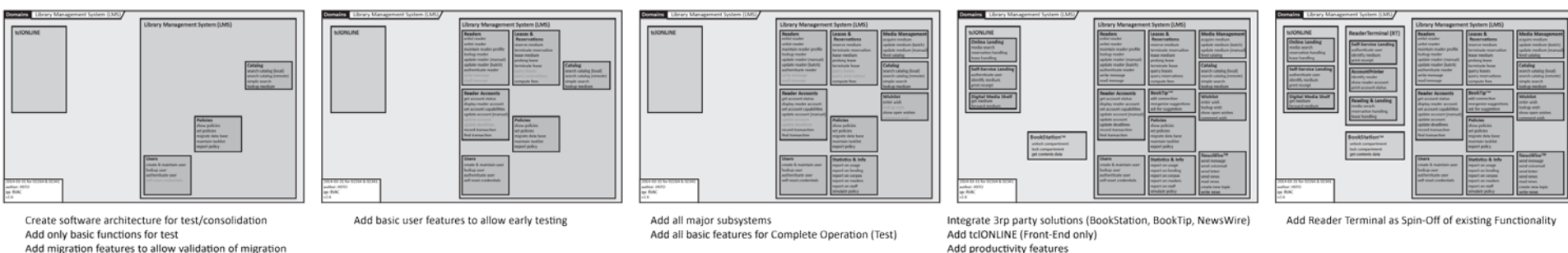
author: HSTO

qa: RVAC

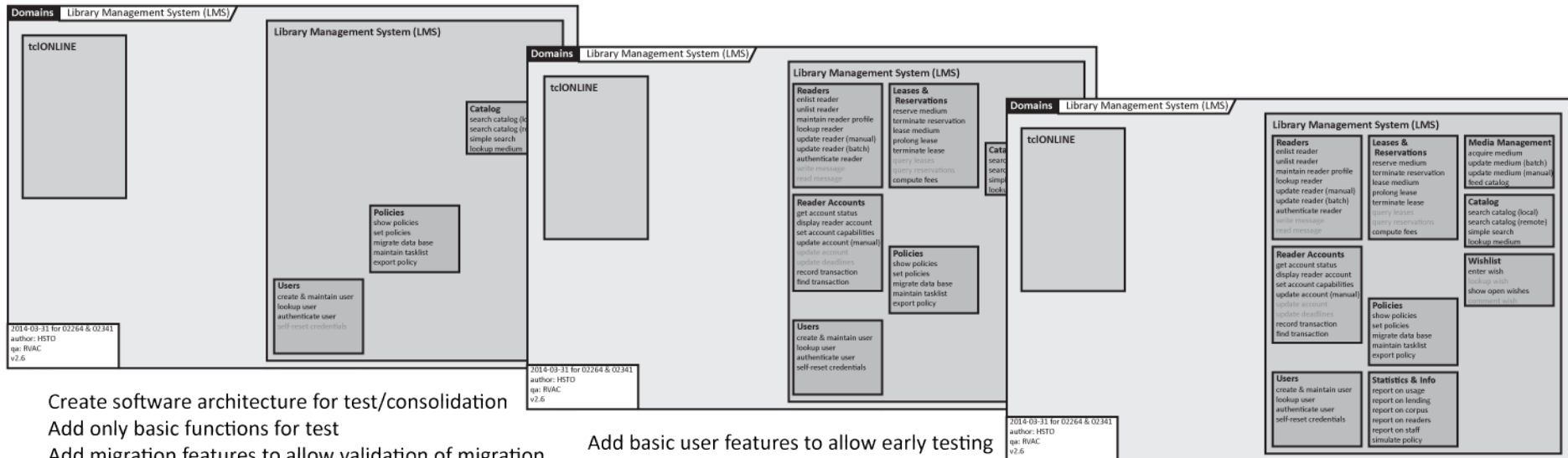
v2.6

# Planning with Domain Architectures

- **Domain architecture diagrams are very helpful in all activities that require a complete overview.**
  - **Prioritization:** discuss and decide the relative importance of processes.
  - **Effort estimation:** calculate compound efforts for subsystems or releases.
  - **Release planning:** decide on a packaging of features/processes over time.
- **The key to obtain the overview are the minimalistic notation and succinct names for systems and processes.**
  - Together they allow to handle large amounts of information synoptically.
  - Initially, the diagrams often appear to be quite trivial.
  - However, Domain Architecture diagrams are also very cheap to create.



# Planning with Domain Architectures



Create software architecture for test/consolidation

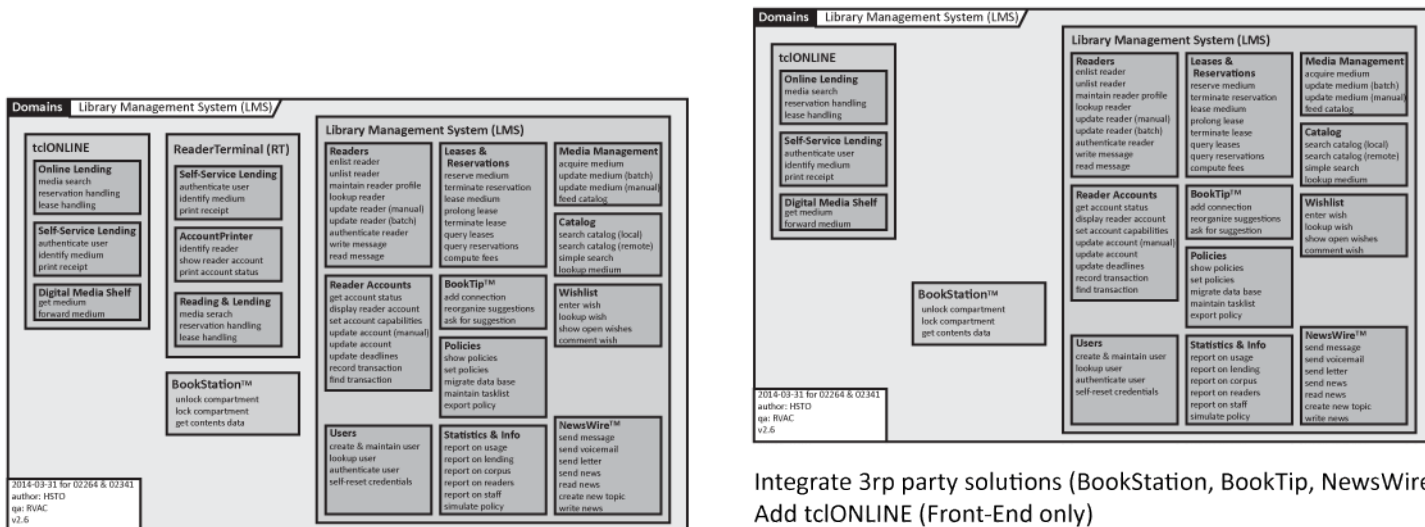
Add only basic functions for test

Add migration features to allow validation of migration

Add basic user features to allow early testing

Add all major subsystems

Add all basic features for Complete Operation (Test)



Integrate 3rp party solutions (BookStation, BookTip, NewsWire)

Add tc!ONLINE (Front-End only)

Add productivity features

Add Reader Terminal as Spin-Off of existing Functionality

# Creating a Domain Architecture

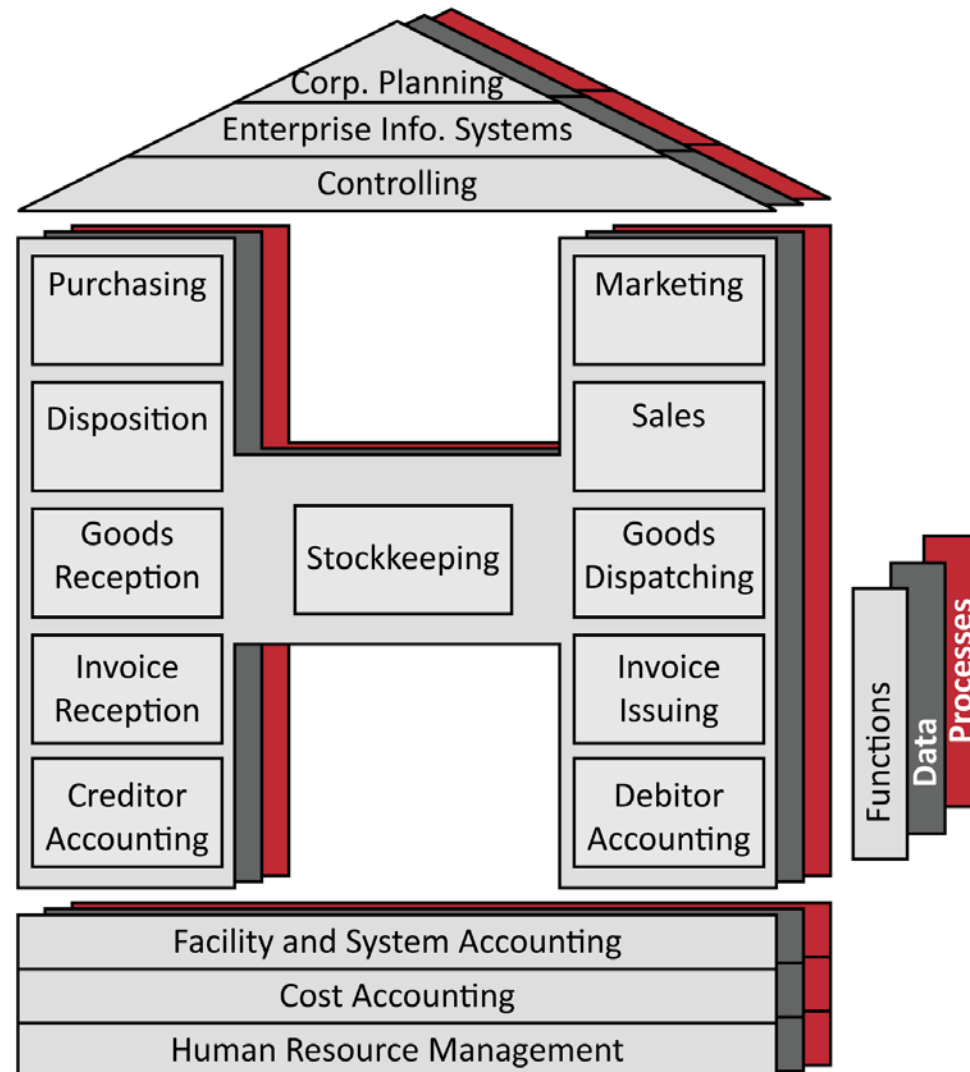
- **As with other model types, there are several sources to inform us about the domain architecture.**
  - Ask the experts: probably the fastest and most reliable way of getting to know existing processes, but not necessarily a good way to learn about new processes.
  - Existing lists of forms/procedures, organizational structures and task assignments: sanity check to ensure everything is covered
  - Industry reference models: aligning terminology and abstraction level with high quality reference models will often improve the model quality, and it may help settle disputes in the development team.
  - Previous models (e.g. personas and goals): particularly important for new processes that are not implemented (as IT or as organization) yet.
- **Doing all of these steps and cross-checking the results increases the confidence.**
  - Recall that small changes at early stages may have large repercussions later on.

# Reference Models

- **References models are idealized abstractions serving as a template and common vocabulary across a whole industry.**
- **Reference models are widely used in different areas.**
  - Analysis/Design patterns are (small) reference models.
  - Other well known reference models in software development include the ISO/OSI 7 layer model and the RM-ODP model.
- **Here, we are interested in reference models of businesses and business information system.**
  - Probably the two best known reference models in this area are the Handels-H model and the VAA model.
  - An extensive catalog of approx. 80 business information reference models is found at <http://rmk.iwi.uni-sb.de>.

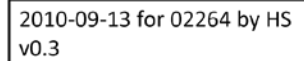
# The Trading Reference Model

- In the trading industry, the “Handels-H”\* is widely accepted to represent the business at a high level.
- This diagram shows only domains and not processes.
  - Typically, such pictures are also used as the entry point to interactive repositories.
  - See [www.semture.de](http://www.semture.de) and [prolix.iwi.uni-sb.de/index.php/Business\\_Process](http://prolix.iwi.uni-sb.de/index.php/Business_Process) for some examples.



\* Handel means Trade in German.

- One way of creating a domain architecture is by looking at the organizational processes and document the, e.g. by a data flow diagram.





Prof. Dr. Harald Störrle  
Danmarks Tekniske Universitet (DTU)

---

## **Chapter 5.4:**

### **Level 3b – System Structure**

---

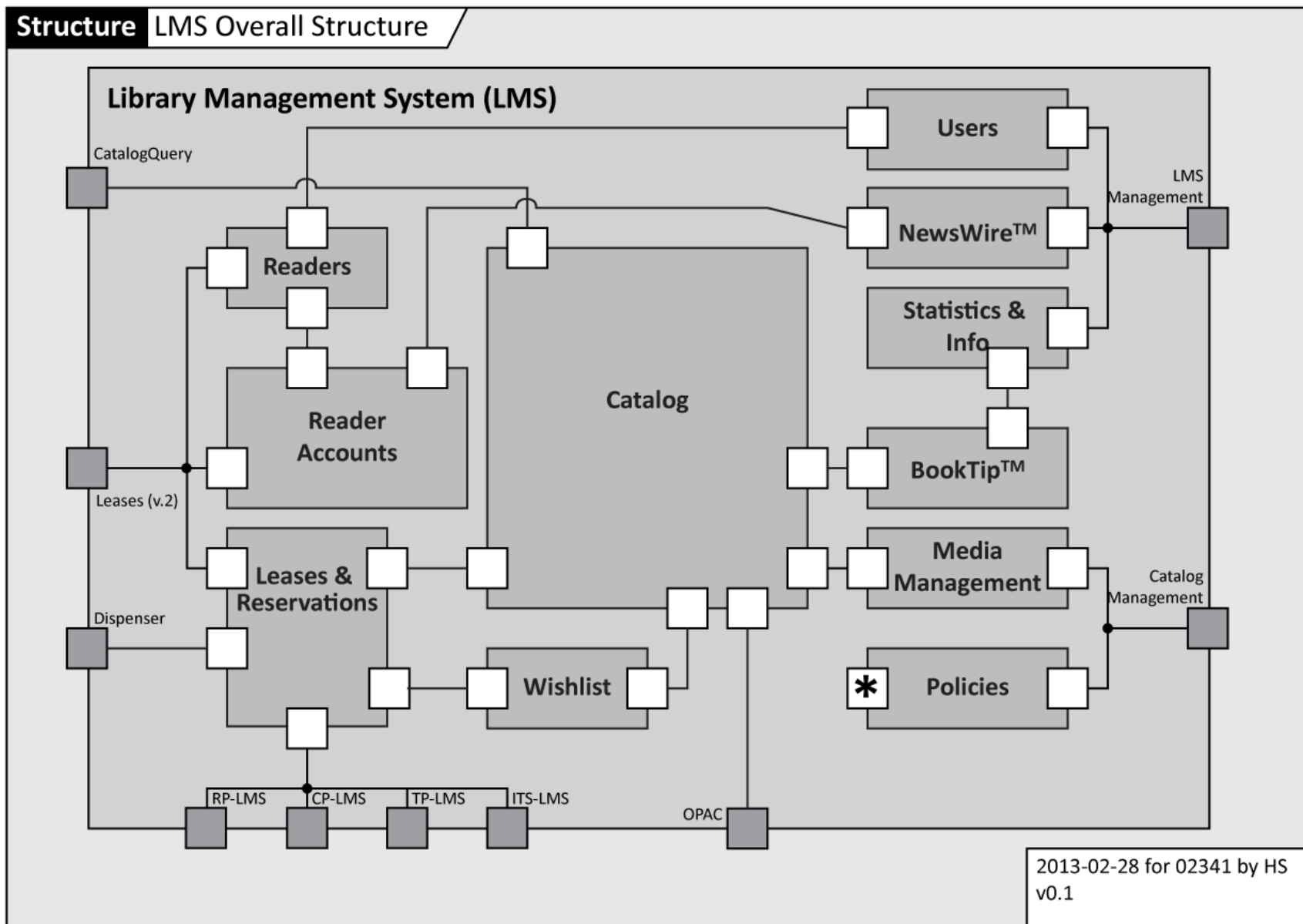
DTU course 02264



# System Structure

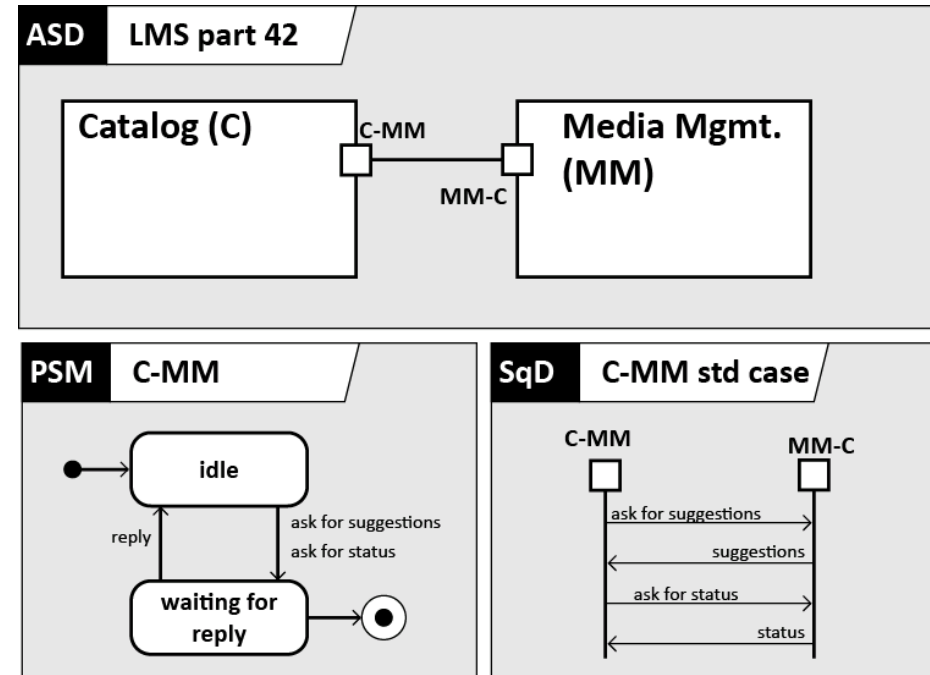
- **The point of Domain Architecture Diagrams is to limit the expressiveness to allow handling even the largest systems.**
- **System Structure Diagrams, on the other hand, allow for detailed specifications of structure and behavior.**
  - It uses the same notation as the Context Diagram, applied recursively.
  - It adds signatures and behavior on the ports, and interactions on the connectors.
- **In a perfect world, such descriptions can be analyzed formally and quantitatively, executed, and animated to support architectural design.**

# System Structure Example (LMS)



# System Interface Models

- The complexity of System Interface Models derives from the tight integration between their parts.
  - The Protocol State Machines for Ports specify the protocol roles of the interaction between two Parts (here: C and MM).
  - The Interactions of connected Parts (their protocol) are specified by interaction diagrams.
  - Thus, the signatures of Ports, their protocol roles and the interactions they are taking part in must conform.
  - Also, the behavior of protocol roles and protocols must conform.
  - Fortunately, these checks are automatable. Unfortunately, almost no tools do it.



# Scenarios in RED

- **In RED, scenarios are modeled as recursive expressions.**
  - Corresponding diagrams (e.g., UML-type interaction diagrams) are currently not available in RED.
- **The scenario expressions appear as a tree where**
  - inner nodes carry operators to combine sub-expressions (e.g., for sequential or parallel composition, loops, exceptions etc.); and
  - leaves are actions or meta-level instructions.
- **In RED, such scenarios may be enriched by pictures and descriptions.**
  - Rich descriptions can be used as a play book for an enactment, or they can be enacted automatically.
  - If UI-sketches are used as pictures, we effectively have an animated paper prototype with automatic voice over.

# Interaction operators (overview)

- **action**
  - do something
- **send**
  - send a message
- **receive**
  - receive a message
- **meta level**
  - text
  - direction
  - start
  - stop
- **ref**
  - macro-expansion of fragment
- **par**
  - interleaving of events
- **alt**
  - alternative execution
- **opt**
  - optional execution
  - syntactic sugar for alt
- **break**
  - exceptional behavior
- **strict**
  - operand-wise sequencing
- **seq**
  - lifeline-wise sequencing
- **loop**
  - repeated seq

# Protocol Roles

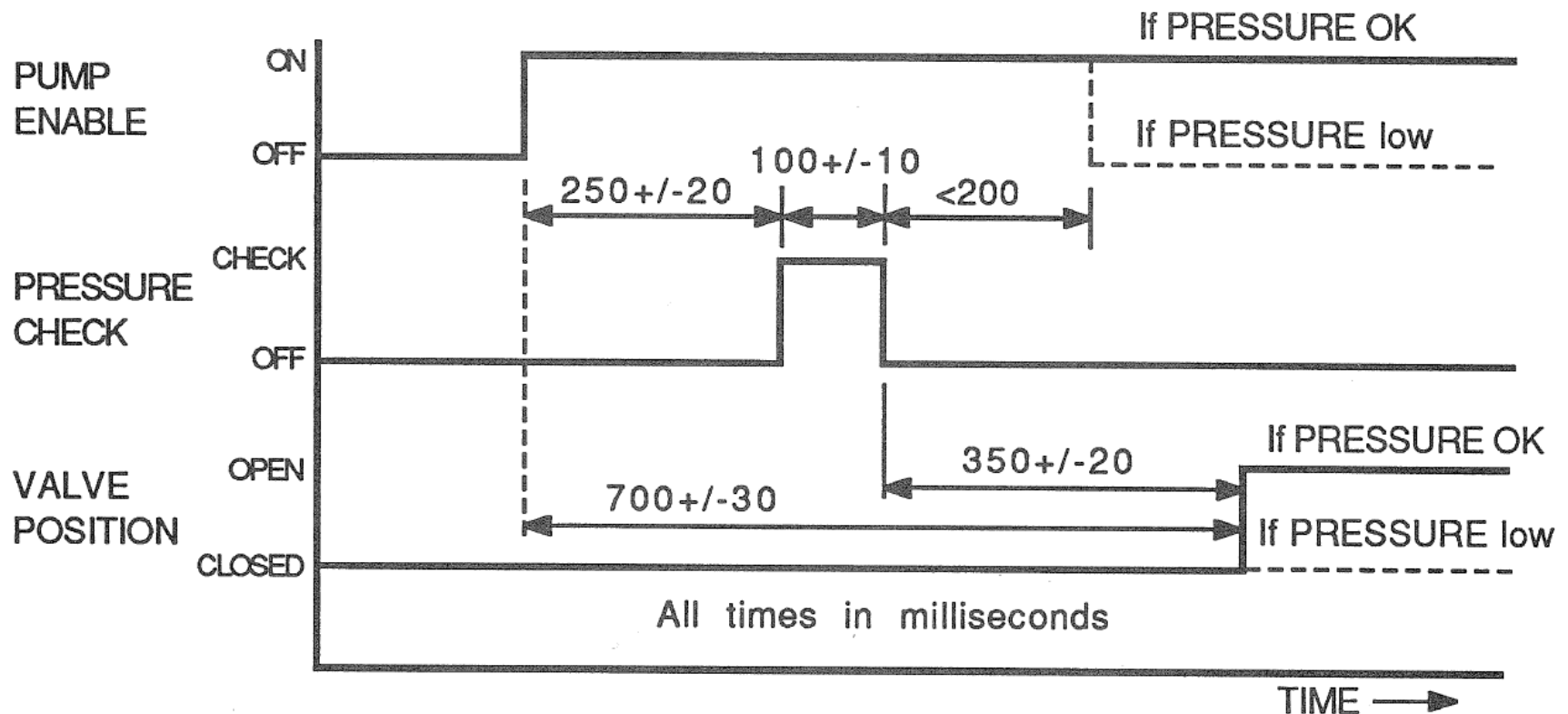
- **Instead of highlighting the interactions between ports, we could also specify the ports in isolation.**
  - This would indeed be required to ship a shrink-wrapped component.
- **A protocol role consists of**
  - sets of used and provided interfaces (“signature”),
  - a state machine to describe their behavior.
- **As before, these model types are not yet available in RED.**
- **Variants**
  - A consistent connection between two ports must then result in “inverse” or “dual” ports (and thus, signatures and protocol roles).
  - Other notable special cases of ports are “relay” ports (vias), underspecified ports, and broadcast ports.

# Timing Constraints

- **Timing constraints are most relevant for systems that interact with the physical world such as real time and embedded systems.**
  - Typical examples include the traffic violation camera explained in the previous section or the door control unit (DCU) introduced in chapter.
- **One thing the DCU controls is the motor to open and close the windows.**
  - Of course it must make sure that the motor shuts off rather quickly when the window is being blocked, e.g. by the driver sticking out his hand.
- **This constitutes a deadline that may be captured as a requirement like this.**
  - R1: When the power consumption of the motor rises above 5 watts while closing the window, the motor must be reversed within 0.1 seconds, and shut off after 0.5 seconds.

# Timing requirements

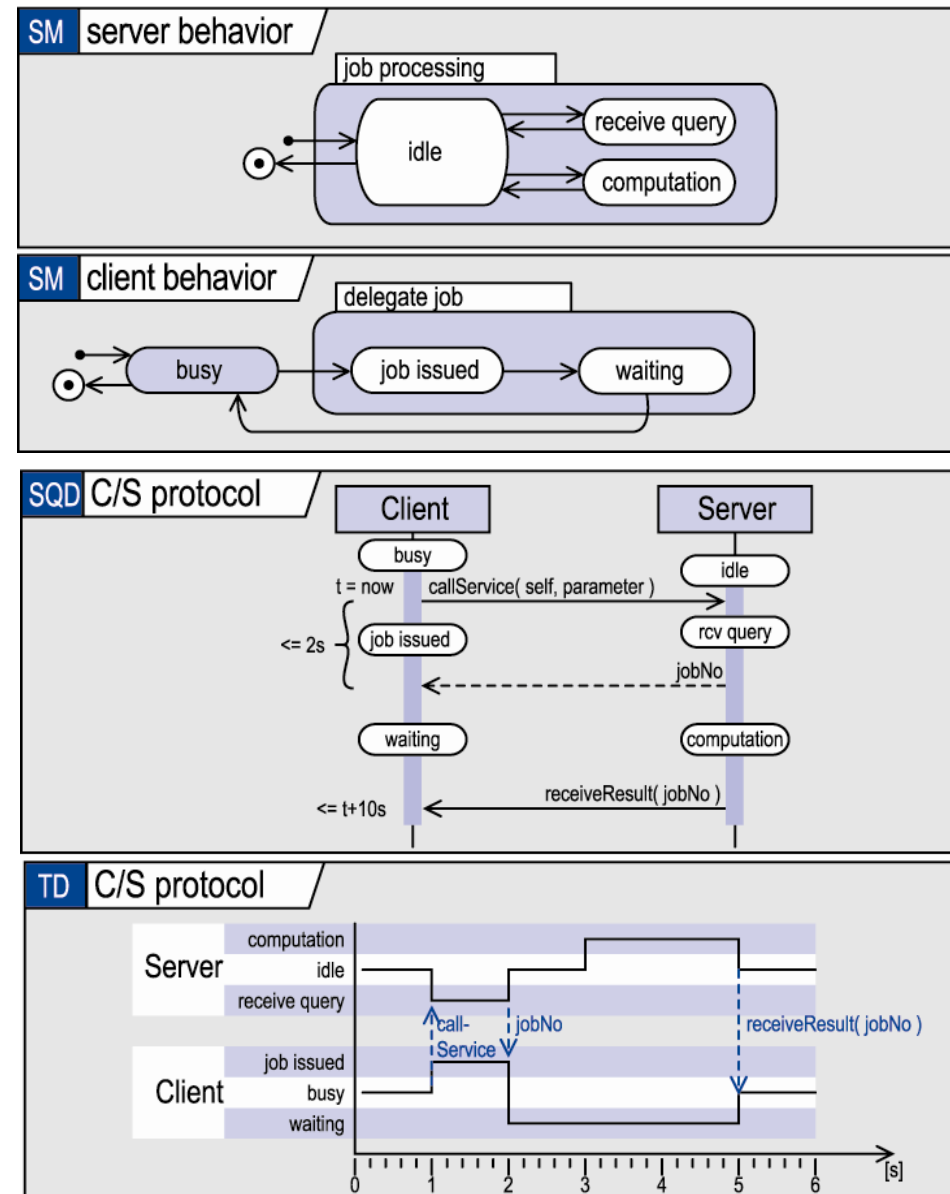
- A timing diagram (Hatley/Pirbai, p. 96) used to specify timing requirements of a part of a hydraulic cruise-control system.





# Using Interactions for timing

- For embedded and real-time systems, it may be important to specify absolute timings and state evolution over time.
- This is not readily expressed in sequence diagrams, much less communication diagrams.
- UML 2.0 introduces timing diagrams for this purpose.
  - Or rather: it reintroduces the notation abolished in OMT 2.





Prof. Dr. Harald Störrle  
Danmarks Tekniske Universitet (DTU)

---

## Chapter 5.5: Level 3c – System Topology

---


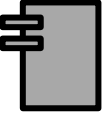
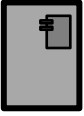


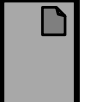
DTU course 02264

# System Topology Model

- **A system topology model (“topology”) defines and visualizes the distribution of system parts of logical and/or physical space.**
  - Observe that there are also low-level aspects to this facet of a system design, which we will ignore here.
- **Topological concerns not just implementation details: they are powerful design constraints, and many of them they come in early.**
  - Frequently, topological aspects implement high-level business decisions or goals, e.g., “Make LMS available online” or “Make this a distributed system” affect the system topology with necessity.
  - Therefore, we need to pick up and document such constraints at the earliest opportunity to inform our design process adequately.
- **Part of the topology has already been documented in the system context.**
  - While it is not always possible to completely separate out each aspect into a separate view, we should strive towards this goal.
- **System Topology is also at least as elemental as software architecture, since the hardware/software division is expressed by the topology.**
  - Many required quality attributes are affected by both concerns, and are best addressed by balancing cost and benefit of both factors (e.g., scalability, performance, availability).
  - Thus, deciding on system topology and software architecture usually go hand in hand.

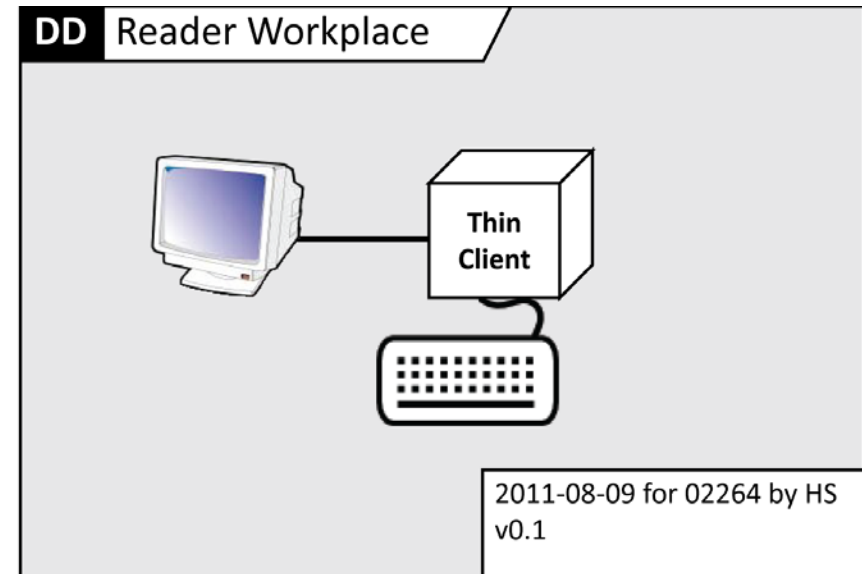
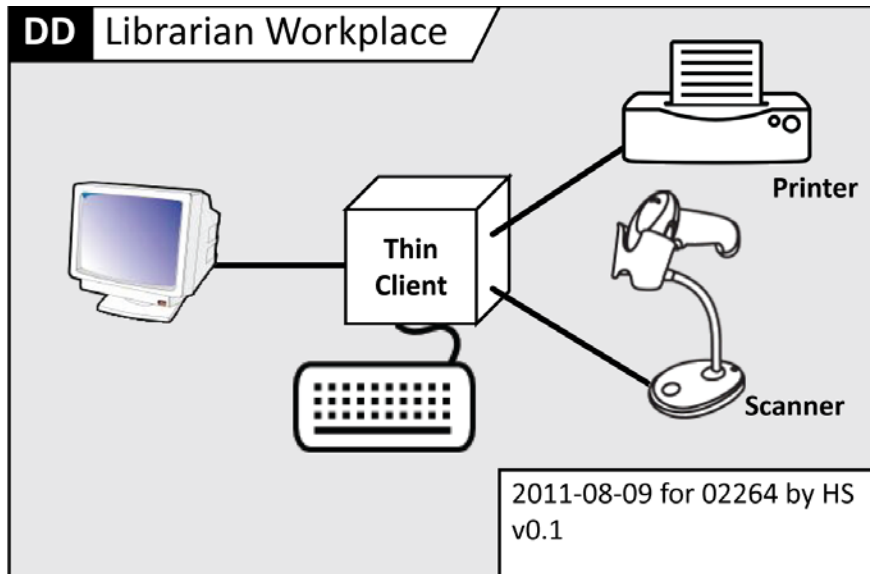
# System Topology Model Notation

- UML offers Deployment Diagrams (aka. Implementation Diagrams) to describe system topologies.
  - These names are somewhat misleading, in that they highlight only one of two applications of the concepts and notations.

Concept	Description	Presentation	Icon
<b>Node</b>	A node is computational resource, which may be either (a) a device (i.e., a computer); or (b) a software system providing some service, such as a mail server	A 3d-box with inscriptions	
<b>Component</b>	The two entities connected by an association share some interaction, e.g. two people talking, a person entering data to	Rectangle with component icon	 
<b>Connector</b>	Connectors connect nodes. If the nodes are (a) devices: then a connector corresponds to a physical connection; (b) programs: then a connector corresponds to an interfacing mechanism such as RPC, or sockets, say; (c) device & program: such connectors would correspond to HW/SW interfaces such as interrupts or flag-registers.	line	
<b>Artifact</b>	Artifacts are physical pieces of information, such as source files and binaries, models, configuration data and so on.	Rectangle with earmark	 

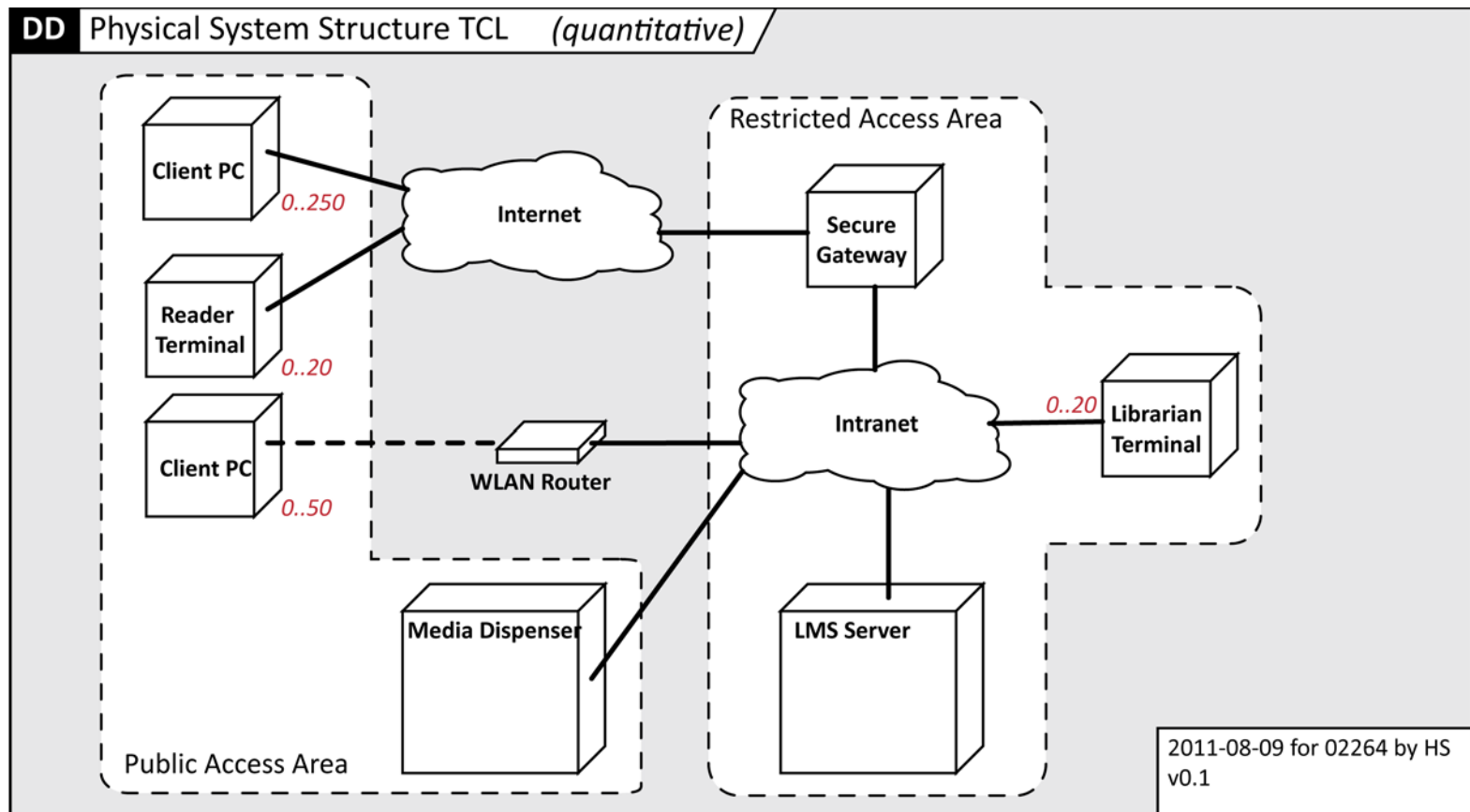
# System Topology Model Notation

- While UML is perfectly capable of expressing system topologies in general, the notation offered is visually somewhat poor.
  - Thus, in practice, we frequently find colorful free-form diagrams.



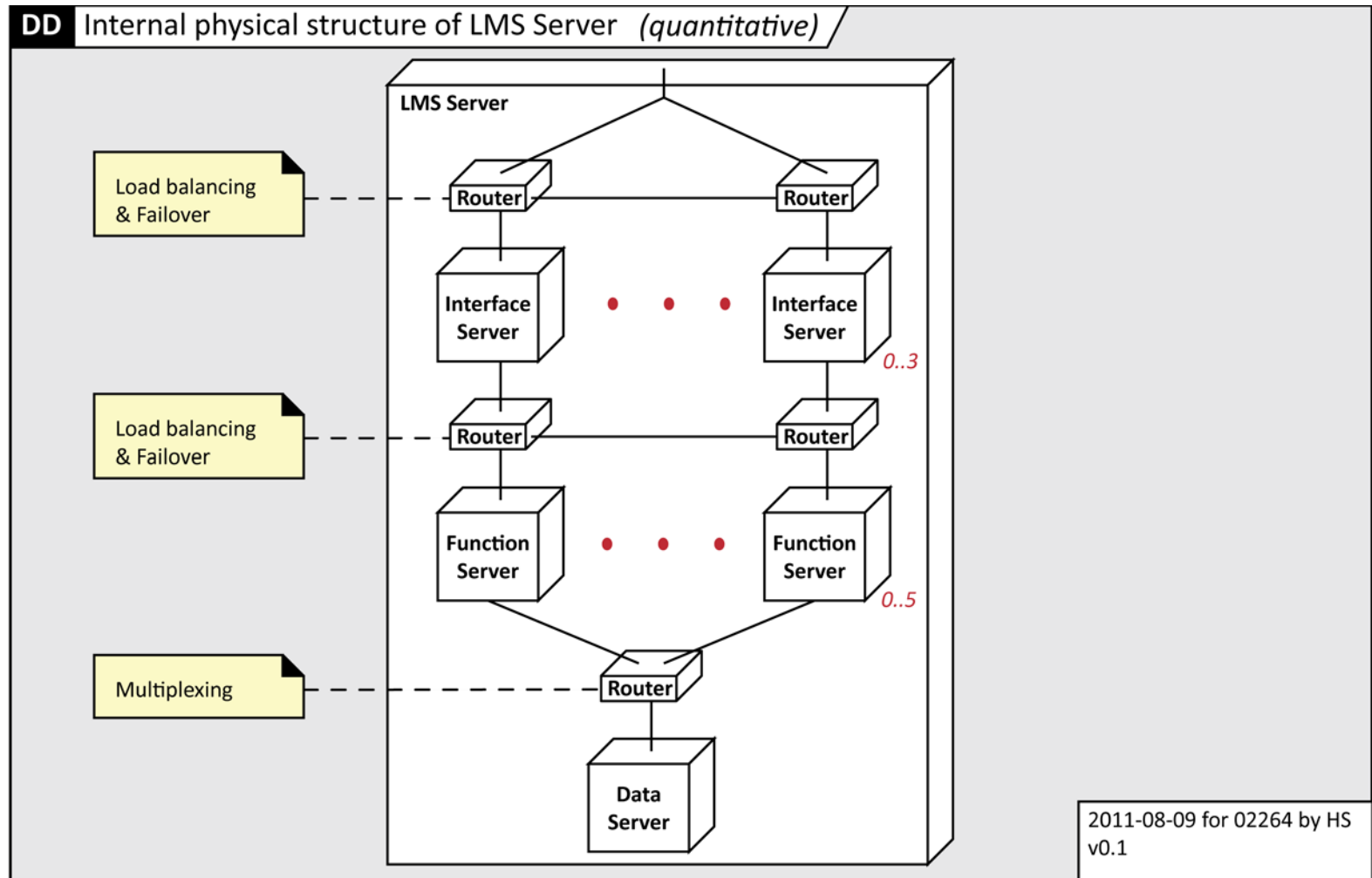
# System Topology Model Notation

- Another important aspect of system topology models is that they allow to naturally capture crucial quantitative information.
  - This kind of information frequently arises rather early in the development process, but has high impact.



# Topology Refinements

- In order to document design decisions that influence architecture and requirements, it might become necessary to include a detailed system topology, including quantitative data.





**Prof. Dr. Harald Störrle**

Software Engineering Section  
Department of Informatics and Mathematical Modelling  
Technical University of Denmark  
Richard Petersens Plads  
Building 322, Room 024  
DK-2800 Kgs. Lyngby

**`hsto@imm.dtu.dk`**  
**`www.imm.dtu.dk/~hsto`**

