Why decomposition and abstraction matters in Requirements Engineering

Martin Glinz

http://www.ifi.unizh.ch/~glinz

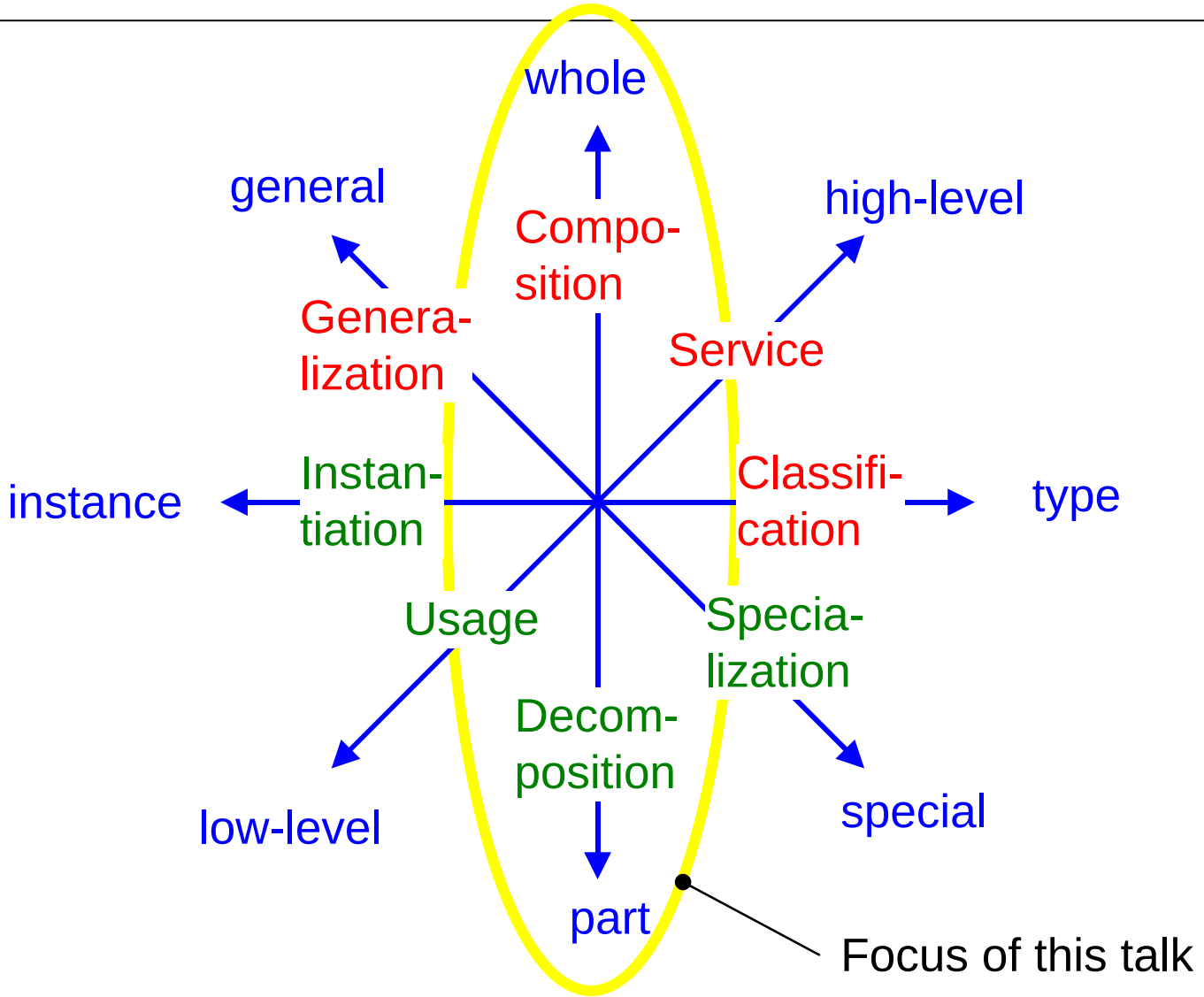
Institut für Informatik der Universität Zürich

Terminology – Abstraction and decomposition

- **Abstraction** – Extracting or emphazizing essential/characteristic/important items or properties in a set of entities with respect to some aspect of interest (and neglecting the rest)

- **Decomposition** – Dividing a set of entities into parts with respect to a given criterion

Four kinds of abstractions



Whole-Part decomposition of object-orientated models

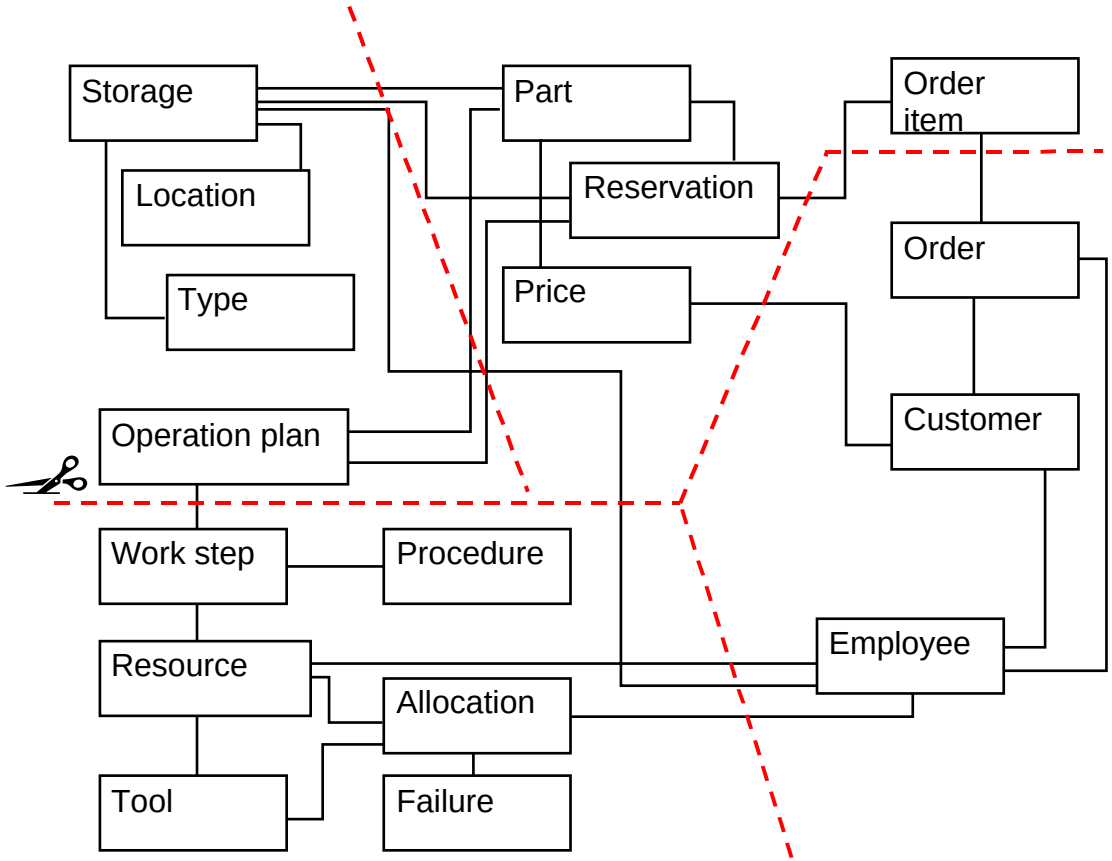
- Looking back:
  - Entity-Relationship-models never got it
  - Object-oriented models inherited the problem from ER-models
  - The early object-oriented analysis approaches did not have any model decomposition

- UML has some decomposition mechanisms
  ...but they do not work as needed

  - Packages are mere containers
  - Composition aggregation is ill-defined and not powerful enough
  - Class models principally cannot be decomposed properly

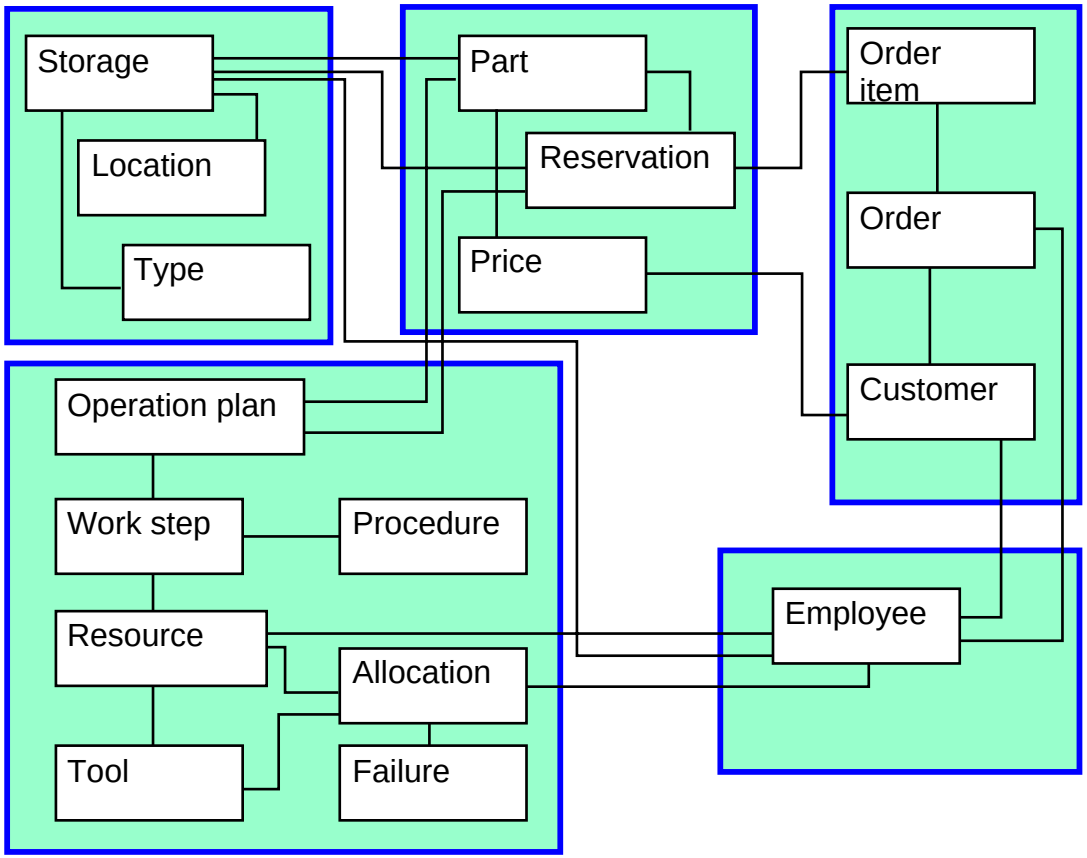
A decomposition example (1)

The poor man‘s way: cutting it into pieces



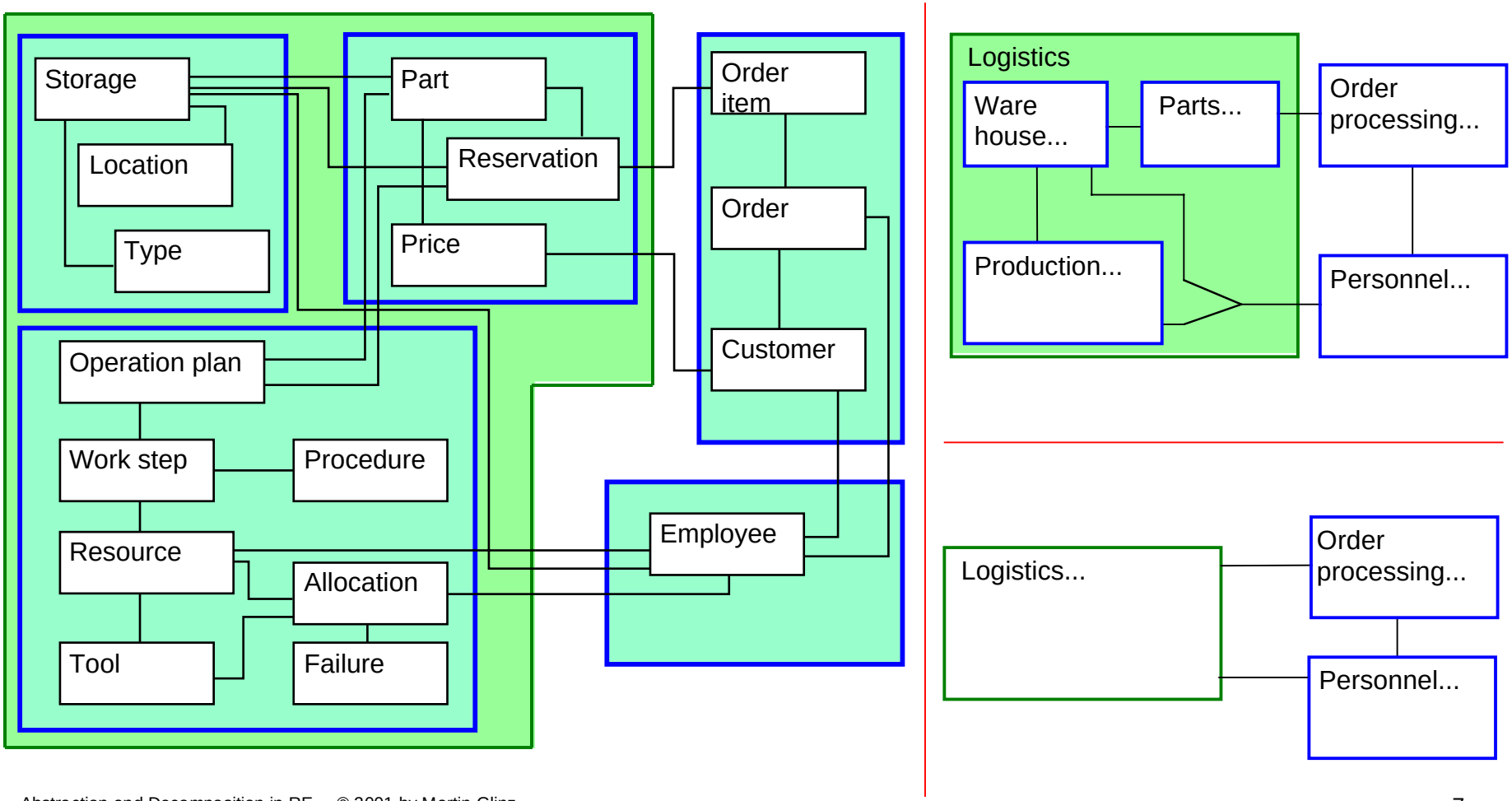
A decomposition example (2)

A more intelligent way: decompose into autonomous parts



A decomposition example (3)

What we really need: decomposition plus abstraction



A good decomposition involves an abstraction

A *good decomposition* divides a problem recursively into parts such that

- every part is a sub-problem that can be treated and understood separately and locally
- every composite gives an abstract overview of the parts and their interrelationships

- Follows the principles of separation of concern and information hiding
- Is a composition (whole-part) abstraction when viewed in the opposite direction

Decomposing models – this is design!?

- Requirements and design models are inevitably intertwined
  - Requirements models must reflect design decisions that have previously been taken on a higher level
  - The structure of an existing system context must be reflected in requirements models
- Information-hiding guided decomposition makes sense for requirements models
  - Understanding large models
  - Distributing work
  - Making large specifications manageable

Example: TRMCS – Goals

The Teleservices Remote Medical Care System (TRMCS)

Goals

The TRMCS shall provide medical assistance to at-home or mobile patients

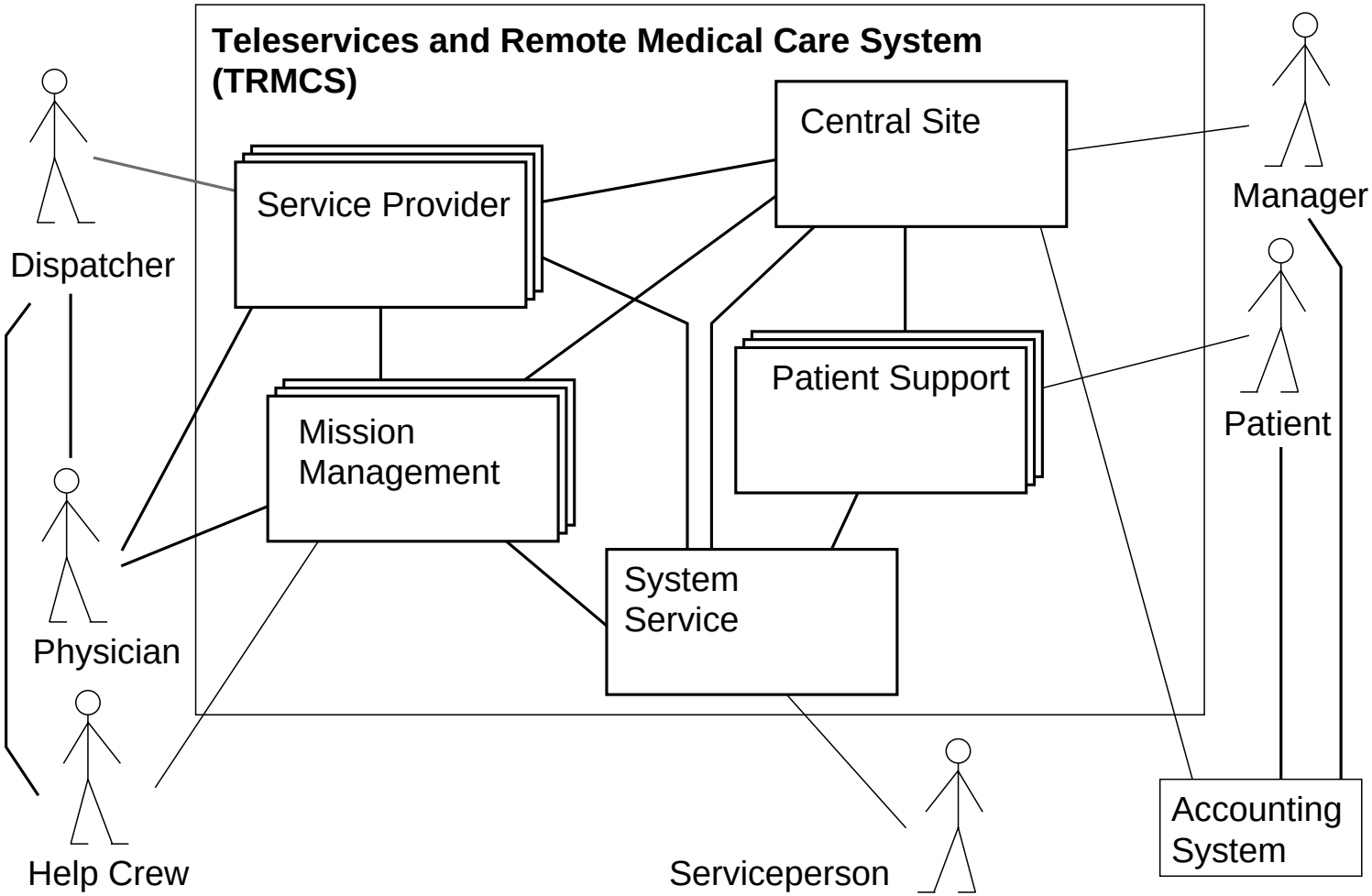
- providing two main services for patients:
  - adequately service help calls issued by a patient
  - continuously telemonitor a patient's health condition and automatically generate a help call when necessary.
- Providing services regardless of the geographic location of the patient
- supporting and coordinating multiple and geographically distributed service providers
- having the same level of reliability, safety, security, accessibility and medical ethics as a local service provided by humans would have.

Example: TRMCS – High level design decisions

System design decisions

- The TRMCS will be implemented as a distributed system with the following subsystems
  - Patient Support at every patient site
  - Service Provider at every provider site
  - Mission Management
  - Central Site
- All events generated by patients are directed to the Central Site. The Central Site routes them to an appropriate Service Provider.
- The TRMCS leaves all decisions about help or treatment to humans.

Example: TRMCS – A decomposed requirements model



The flip side of the coin

Problems introduced by decomposition

- Specification is no longer purely descriptive
- Hiding requirements – a recipe for disaster?
- Proving global global properties might become difficult
- Compositional reasoning required