How to Guarantee at the Architectural Level the Dependability Requirements of a System?

Contracts, Composability and Optimized Prevention

Miroslaw Malek
Institut für Informatik
Humboldt-Universität zu Berlin
malek@informatik.hu-berlin.de

The NOMADS Republic

The Status: The largest nation on Earth

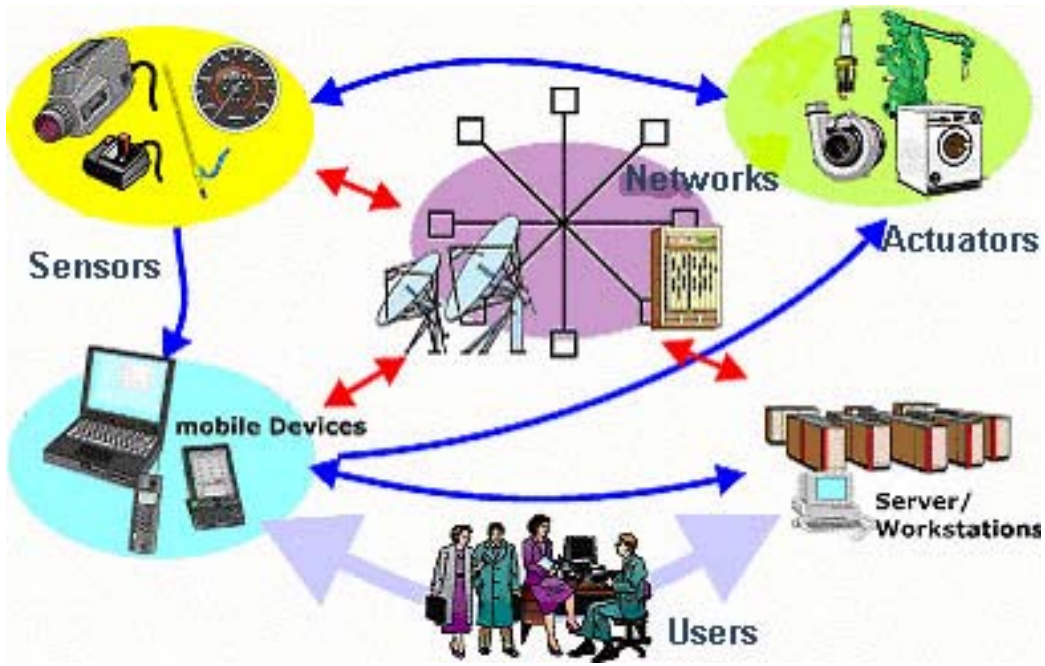
Population: 20 - 100 B citizens, maybe 1T

Key qualities: **M**obility, **A**daptivity and **D**ependability

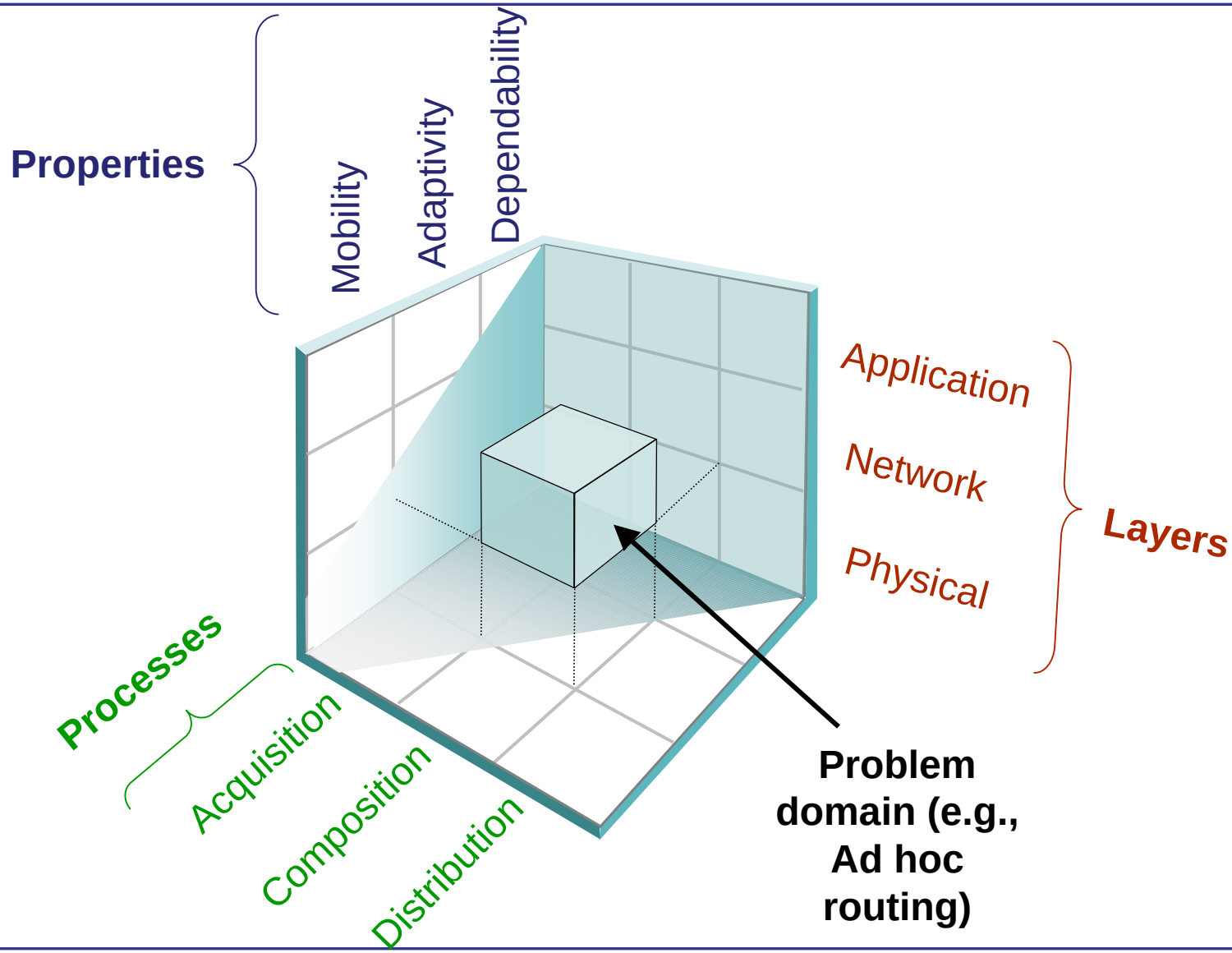
The Need

Need for a unifying paradigm that covers:

- embedded systems
- sensor networks
- personal computing
- server farms
- GRID computing

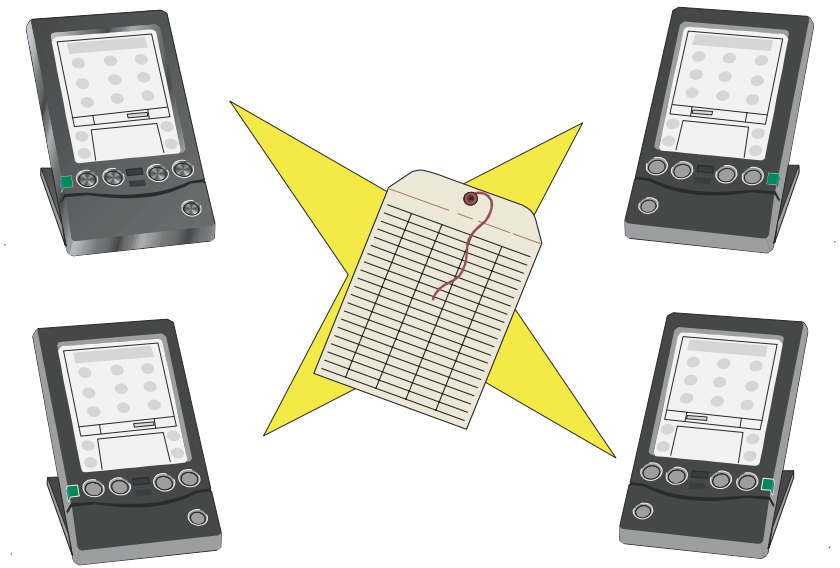


Approach


Properties
Mobility
Adaptivity
Dependability
Application
Network
Physical
Layers
Processes
Acquisition
Composition
Distribution
Problem domain (e.g., Ad hoc routing)

Our Experience at Architectural Level

- Consensus:
  - Unstoppable orchestra, robots, security
- **Service**
  - **Contracts**
  - **Reuse**
  - **Composability**
- **Failure Prediction and Recovery**
- Communication
  - Ad hoc routing
  - Remote experiment
- Resource Allocation
  - Dynamic scheduling



NOMADS Services

Hiding complexity behind services:

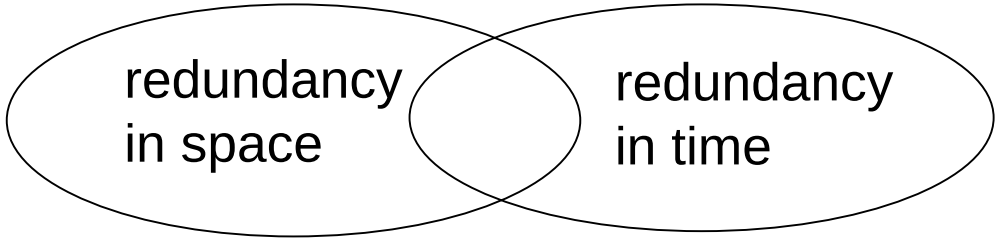
- “everything” is a service paradigm
- W-questions:
  - Who are you?
  - Where are you?
  - What do you offer?
- services/contracts expose extended interfaces:
  - functional properties
  - non-functional properties
  - semantics
- **contracts extraction**
- interoperation between systems
- composable architectures
- composition, decomposition, adaptation

Existing Approaches:

- WSCI (Web Service Choreography Interface)
- WSFL (Web Service Flow Language)
- BPEL (Business Process Execution Language)

Dependability

Fundamental principle:
- redundancy in space
- redundancy in time



Service replication vs. service retry:
- tradeoffs
- cost
- applications

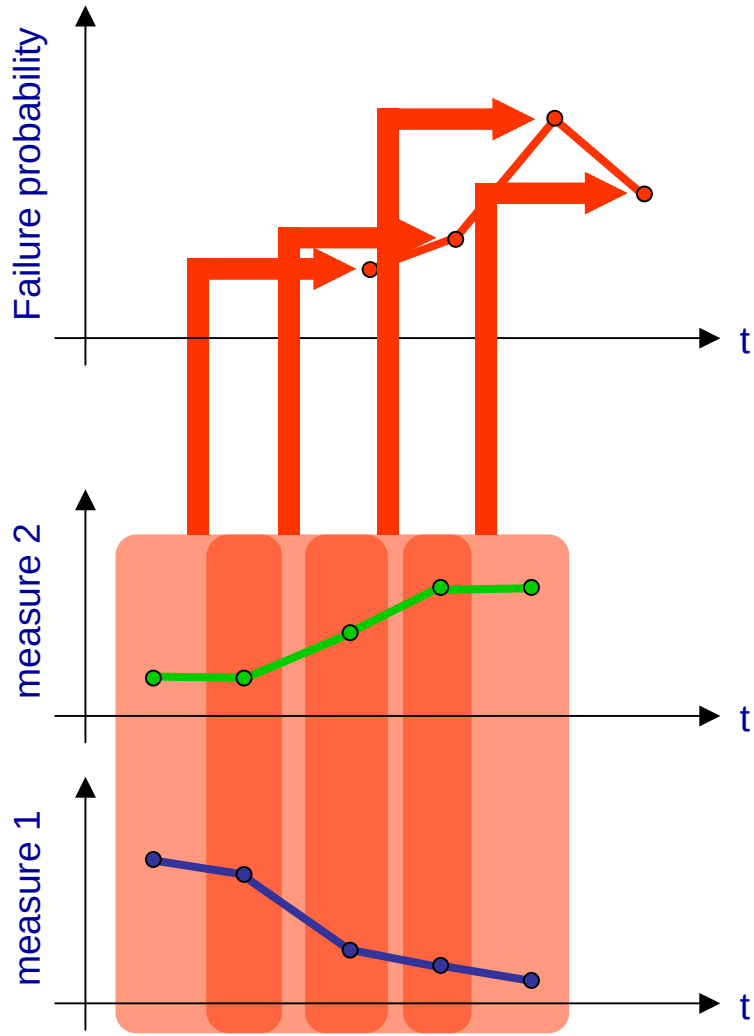
Conflicting requirements:
- fault tolerance
  - multiple copies
- security
  - single secure copy

Two-stage approach

- Step 1: Failure Prediction
  - Pattern recognition with extended Markov chains
  - Function approximation with universal basis functions (UBF)
- Step 2: Preventive action(s)
  - Load lowering
  - State clean-up
  - Garbage collection
  - Establishing recovery point (checkpointing)
  - Process restart
  - Failover
  - System restart

Universal Basis Functions Model

- **Suited to continuous data, e.g.,**
  - **Workload**
  - **Memory usage**
  - **Process starts per second**
- **Function approximation of failure probability**
- **Universal basis functions:**
  - **linear mixtures of bounded and unbounded activation functions such as Gaussian, sigmoid and multi quadratics**



Example: Self-Rejuvenation

- Preventive restart before a failure occurs
- System-state-dependent, automatic restart
- Full-featured Self-Rejuvenation comprises:
  - When to restart
  - What to restart
  - Which method to choose, e.g.,
    - Going back to recovery block
    - Redeploying an entire component

Today and Tomorrow

- Service Oriented Computing:
  - frameworks are evolving
  - reuse
  - dependability
  - productivity

- NOMADS and others will provide an infrastructure for service architectures

- Failure prediction and prevention

will be widely applied

- Self-X functionality and flexibility will be a must