TDDE41 Designing for Non Functional Properties

Lena Buffoni
lena.buffoni@liu.se

LINKÖPING UNIVERSITY

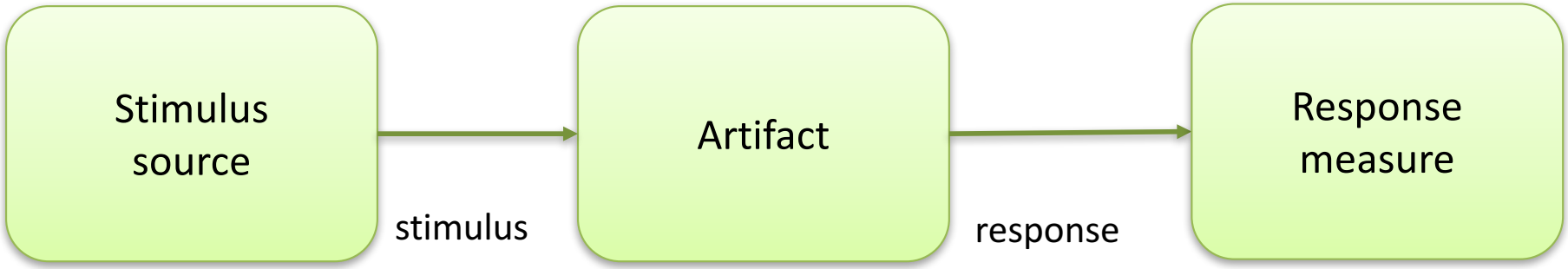
Lecture plan

- Designing for non-functional properties
- Efficiency/performance
- Complexity
- Scalability and Heterogeneity
- Adaptability
- Dependability
- Security

How do we express system properties?

- Context: System can be robust with respect to some fault and brittle wrt to others
- Responsibility: Is a failure because of a denial of service attack – Availability? Performance? Security? Usability?

A model for expressing system qualities



Guiding design decisions

- Allocation of responsibilities
- Coordination model
- Data model
- Management of resources
- Mapping among architectural elements
- Building time decisions
- Choice of technology

Efficiency

Meeting performance requirements while minimizing system resources

"95% of requests processed in under 4s and all requests processed in under 15s"

Performance measures

- Throughput
  - Number of transactions or messages processed per second (tps/mps)
- Response time
  - Guaranteed vs average response time
- Deadlines

Designing for Efficiency

- Keep components small
- Keep interfaces simple and compact
- Separation of data from meta-data
- Use asynchronous interactions
- Use location transparency
- Keep interacting components close/co-locate resources

Manage resources

- Increase resources
- Introduce concurrency
- User replicas for computation units and a load balancer
- Caching and data replication
- Schedule resources

Complexity

Degree to which a system's design or implementation is difficult to understand or verify

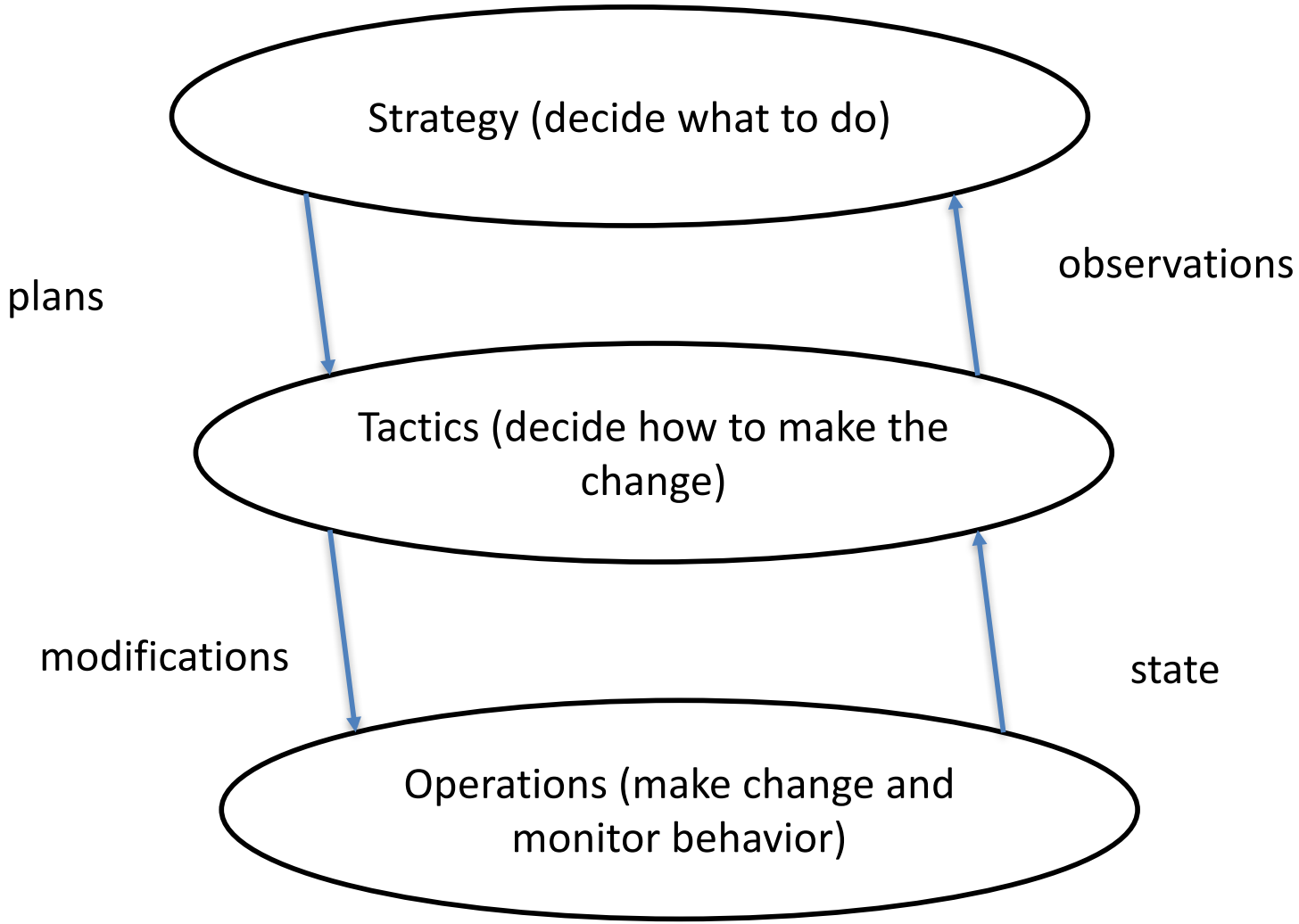
Designing to control Complexity

- Separation of behavioral concerns
- Separation of interaction concerns
- Reduce interaction responsibilities on components
- Cohesive components
- Select off the shelf components carefully
- Decouple software logic from data format
- Limit dependencies
- Explicit dependencies
- Avoid unnecessary heterogeneity

Adaptability

System's ability to adjust to new requirements and operating conditions during it's lifetime

- What can change?
- What is the likelihood of change?
- When is the change made?
- What is the cost of change?


Strategy (decide what to do)
plans
observations
Tactics (decide how to make the change)
modifications
state
Operations (make change and monitor behavior)

Impact of a change

- Broken links in the new architecture?
- Are all interfaces type consistent?
- Any licensing concerns?
- Are security requirements respected?
- Is every component and connector mapped?
- Is the deployment package complete?

Dynamic changes

- Service strategy
- Timing
- State restoration or transfer

Even further : autonomous changes?

Planing for Adaptability/ Modifiability

- Each component has a single, clear function
- Minimize interdependencies
- Decouple interaction from component logic
- Decouple processing from data
- Decouple data from meta-data

Techniques

- Plug-ins
- Interfaces
- Component frameworks
- Scripting languages
- Event-based communication

Defer binding

- Compile time binding
- Deployment-time binding
  - Configuration files
- Runtime binding
  - Runtime registration
  - Dynamic lookup
  - Plug-ins
  - Publish-subscribe

Pitfalls

- High modularity ->
  - Complex design
  - Performance overhead

  - Predicted cost is only an estimate

Dependability aspects

- Reliability
- Availability
- Robustness
- Fault-tolerance
- Survivability
- Safety
- Security

Dependability principles

- Control external dependencies
- Provide reflection capabilities
- Handle exceptions
- Define state invariants
- Interaction guarantees
- Avoid single points of failure
- Provide backups for critical parts of the system
- Support monitoring
- Support dynamic adaptation

Availability

- MTBF / (MTBF + MTTR)
- Impact of failures (no effect – catastrophic)

Availability	Downtime/90 days	Downtime/Year
99.0%	21h, 36m	3d, 15.6h
99.9%	2h, 10m	8h, 46s
99.99%	12m, 58s	52m, 34s
99.999%	1m, 18s	5m, 15s
99.9999%	8s	32s

High availability

Detect faults

- Ping/echo
- Monitor components
- Heartbeat
- Time stamp
- Sanity check
- Voting
- Replication
- Redundancy (functional, analytic)

Handling faults

- Redundancies – active/passive
- Spare
- Exception handling
- Rollback
- Software upgrade/ patch
- Retry
- Ignore
- Degradation mode
- Reconfiguration

Scalability

Capacity of system to adapt in size and scope

- Number of simultaneous connections

Eg: 2000 connections max/ computer -> 100.000 users -> 50 computers

Scalability and Heterogeneity

- Each component has a clear purpose
- Each component has a simple interface
- Avoid unnecessary heterogeneity
- Distribute data sources
- Use replication sparingly
- Avoid system bottlenecks
- Parallelize things
- Use topology strategically
- Use transparent distribution

Use case: multi-player online games

- Need to scale up and down – distributed computation
- Event based model
- Thick client
- Server holds the shared state
- Resource distribution – geographical/sharding…
- System structured as set of services with small interfaces

Security

- Confidentiality
- Integrity
- Availability

Trust management

- Reputation based systems
- Decentralized trust management
  - Authentication
  - Internal beliefs vs externalized information
  - Explicit trust relationships
  - Comparable trust

Security tactics

Detect attacks

- Detect intrusion
- Detect service denial
- Verify message integrity
- Detect message delay

Resist attacks

- Identify actors
- Authenticate actors
- Authorize actors
- Limit access
- Limit exposure
- Encrypt data
- Separate entities
- Change default settings

React to attacks

- Revoke access
- Lock computer
- Inform actors

Recover from attacks

- Restore

Other attributes

- Testability
- Usability
- Portability
- Mobility
- Marketability
- ….

Discuss

Component reuse and non-functional properties?

Design trade-offs

- Highly secure system – open environment
- Real time system – asynchronous communications
- Small components or off the shelf components?
- Abstraction or efficiency in implementation and in location?

Better/Cheaper/Faster ?

Summary

- Different architectural decisions have different quality properties
- Be aware of trade-offs
- Augment the architectural design patterns with tactics for non-functional properties

Questions?

www.liu.se

LINKÖPING UNIVERSITY