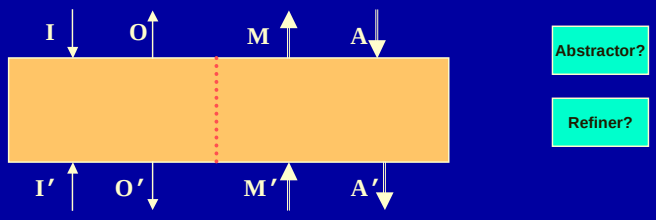
Class 5
Architecture-Based Self-Healing Systems

David Garlan
Carnegie Mellon University
www.cs.cmu.edu/~garlan/17811/

Architectures of Self-Healing Systems

Basic Component



What is its signature?

Is this a general-enough model?

What architectural styles can be used to compose such components?

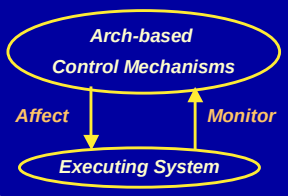
Four General Requirements for Self-Healing Systems

- Monitor: Observe the running system and abstract observed behavior.
- Detection: Continuously check design constraints via explicit run-time models.
- Resolution: Determine the cause of constraint violation and choose a repair strategy.
- Adaptation: Adapt the system using verified change strategies.

- Monitor:
  - » How can we get information out of a running system?
  - » How can we abstract it to make sense of observations?
- Detection:
  - » What kind of models are useful? What kinds of constraints?
- Resolution:
  - » What kind of repair engines are useful, efficient, flexible?
- Adaptation:
  - » How can we cause the repairs to happen safely in a running system
  - » How can we verify repair strategies?

Architecture-based Self-Repair

- Main idea
  - > Use architectural models as basis for problem detection and repair.
- Variations
  - > Externalized or internalized?
  - > Style-specific?
  - > Support multiple control models?
  - > Enforcement of linkage between architectural model(s) and system?



Taxonomy?

- Application Domains
  - > Networks, Distributed Systems
  - > Mobile Systems
  - > Ubiquitous Computing
  - > Biology Simulation
  - > User Interfaces
  - > Collaborative Computing
  - > Games
- Tools, Mechanisms, Techniques
  - > Architectural models
  - > Algorithms/code-based
  - > Formal models
  - > Genetic algorithms/alternative models
  - > Agents
  - > Economic theory
- Goals:
  - > Improve system performance/Resource usage
  - > Improve user experience; reduce user distractions
  - > Improve dependability

Arch-based adaptation

Typically, but not necessarily distributed applications

Arch models

Reusable abstractors and refiners

Repair strategies

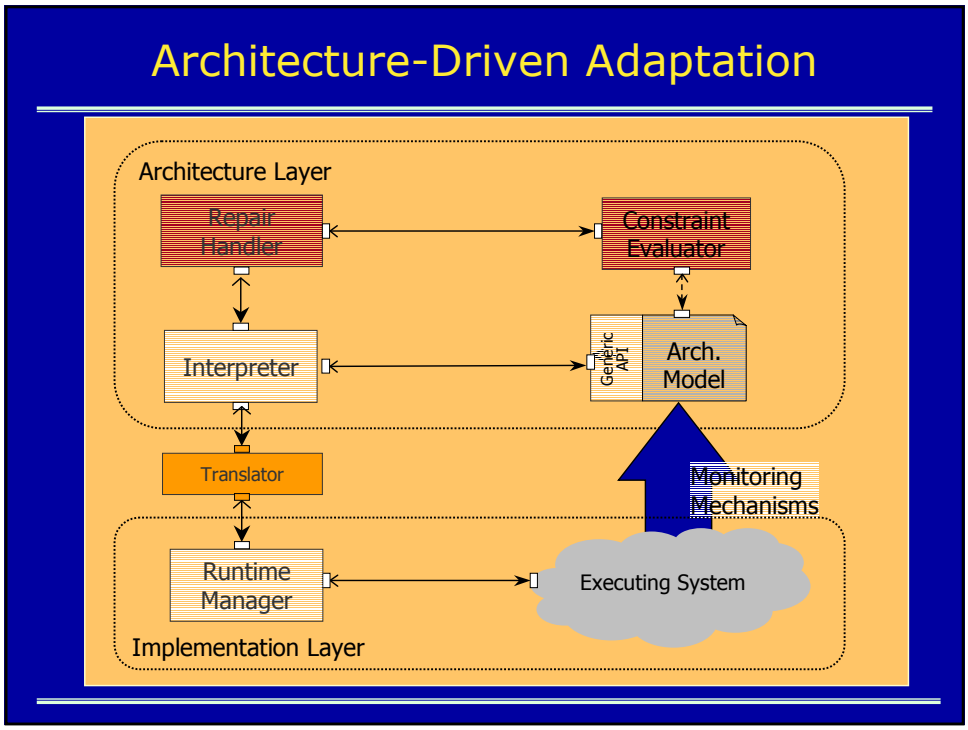
System quality attributes such as performance, reliability, etc.

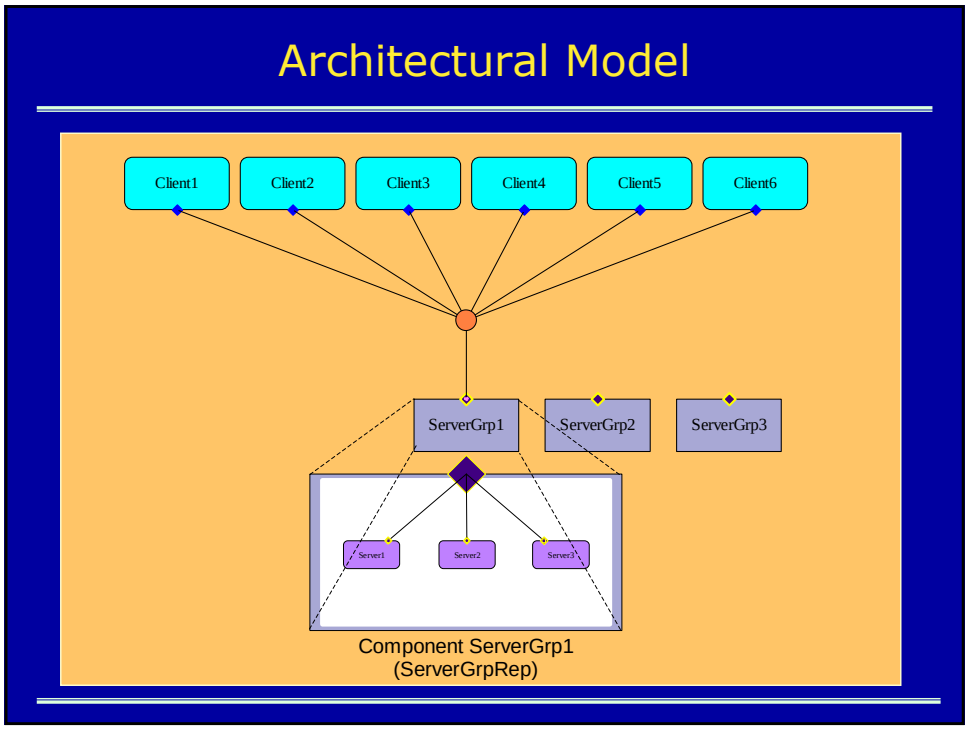
Rainbow Perspective

> Work with existing systems, and manage the adaptation of those systems
  - » Allow minimal intrusion into the system
  - » Use existing mechanisms for change
  - » Use non-intrusive monitoring techniques
> Use architectural models as basis for self-healing
  - » Externalized model
  - » High-level perspective
  - » Tailorable to specific architectural style and properties of interest

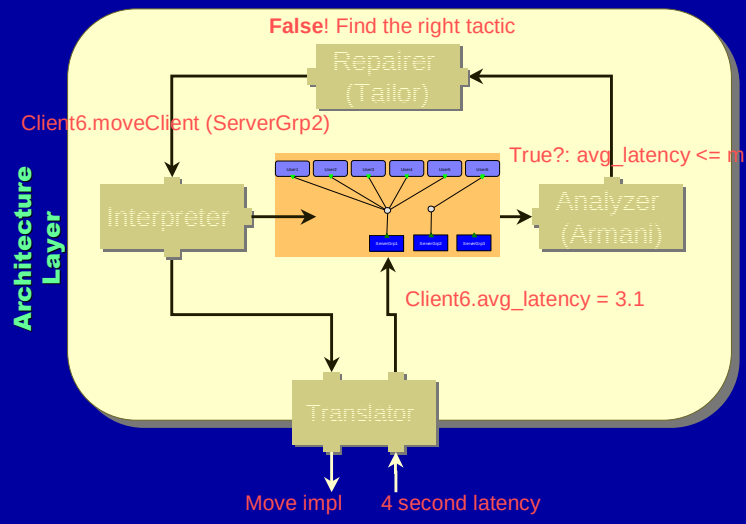
Rainbow Approach

- Monitor:
  - » Use system of “probes” to get raw information out
  - » Abstract that information using “gauges”
- Detection:
  - » Compare (abstracted) observed behavior against desired (as specified in the architecture)
- Resolution:
  - » Specify architecture-based repair policies
- Adaptation:
  - » Rely on system’s own adaptation capabilities, but provide a translator to map from arch-based adaptation to lower-level ones.


Architecture-Driven Adaptation
Architecture Layer
Repair Handler
Constraint Evaluator
Interpreter
Generic API
Arch. Model
Translator
Monitoring Mechanisms
Runtime Manager
Executing System
Implementation Layer


Architectural Model
Client1
Client2
Client3
Client4
Client5
Client6
ServerGrp1
ServerGrp2
ServerGrp3
Server1
Server2
Server3
Component ServerGrp1
(ServerGrpRep)

Making Repairs



Technical Details

- Monitoring mechanisms
  - > Reusable library of ‘gauges’ translate ‘probe’ information into model-based properties
    - » implemented as Java classes
    - » listen to a probe pub-sub bus
    - » dynamically modify properties of architectural models
  - > Example gauges:
    - » *performance gauges* convert network observations into connector latencies and throughputs
    - » *protocol gauges* detect ordering of events in component interactions

Technical Details (continued)

- Architectural models
  - > Typed components, connectors, compositions define architectural topology
    - » Acme
  - > Property annotations convey semantic details
    - » Define expected dynamic behavior
    - » Extensible, like XML
  - > Each architecture must conform to a ‘style’, which indicates
    - » Allowable types of elements, constraints on composition
    - » Constraints expressed as first-order predicates over architectural elements, topology, and properties

Technical Details (continued)

- Modifications to architectural representation needed for dynamic adaptation
  - > Define a set of architecture change operators for that style
    - » E.g., change a client-server connector, add a server
  - > For each stylistic constraint associate a repair strategy
    - » May be verified in advance
  - > For each architectural change operation define a translation to modify the running system
    - » May involve complex set of distributed, coordinated, low-level actions

Example

- Sample constraint:
  `avg_latency < max_latency`
- Sample repair tactic

```
tactic fixServerLoad (client : ClientT) ={
    let lSG : set {ServerGroupT} =
        select sgrp:ServerGroupT in self.components |
            connected (sgrp, client) and sgrp.load > maxLoad
    foreach sGrp in lSG {sGrp.addServer ()}

}
```

Adaptation Model