Self-Adaptive *Stitch-ing*
(Architecture-based Self-Adaptation in the Presence of Multiple Objectives)

Shang-Wen Cheng, David Garlan, Bradley Schmerl
*Carnegie Mellon University*

SEAMS 2006 – ICSE Workshop
May 21-22, 2006
Key Points

1. Rainbow is an architecture-based approach for adding self-adaptation to computing systems

2. Self-adaptation capabilities address the kinds of repair strategies that are today performed manually by system administrators

3. Utility theory helps make tradeoffs between strategies where multiple objectives must be considered
Research Context

Problem motivation → change
- Dynamically changing user needs
- Unpredictable environment
- Resource variability
- System faults
- Rising IT operation cost due to system complexity

Domains of applicability
- Systems with high uptime requirements, e.g.,
  - E-commerce system
  - Mobile, ubiquitous computing
  - Large enterprise systems
  - Potentially: Mission-critical system and critical infrastructure
Approaches to Adaptation

- **Self-adaptation techniques today**
  - Low-level, embedded mechanisms
    - Effective and timely
    - Localized and costly to modify later
  - High-level, human management
    - Global perspective, can keep high uptime
    - Error-prone and also costly to upkeep

- **Ideally, an alternative new way that**
  - Maintains global perspective & provides high uptime
  - Automates the management tasks
  - But realized with much less effort and lower cost
News Website Z.com

Top-level constraint: client response time must fall within threshold

Load

ResponseTime

C₁

... 

Cₙ

Server pool

WebServer 1

WebServer k

Load

Liveness

- retrofitLB
- initiateIDS

-enlargeServerPool
-reduceServerPool
-switchToTextual
-switchToGraphical
-restartWebServer

-segregateCriticalApp
-initiateAuditing

May 21, 2006

Cheng, Garlan, Schmerl © 2006
Rainbow Self-adaptation

Architectural model & Adaptation mechanism

- Global system perspective
- Important system-level behaviors and properties
- Explicit system integrity invariants
- Proven design and tradeoff analysis techniques

System Z

Architecture-based self-adaptation
Empower the Software Engineer

- **Overall objective**
  - Empower software engineers with tool to add self-adaptation capability to existing systems

- **Critical research problem**
  - How to represent adaptation expertise and automate adaptation to satisfy multiple objectives
  - Essentially, how can we reduce the mundane and routine tasks of the system admin?
    - Disclaimer: we’re not eliminating the system admin profession...

- **Core issues**
  - Designing an expressive adaptation language
    - What kinds of information needs to be expressed?
  - Designing the supporting infrastructure
Harold, the Wise Sys Admin

- Task: keep the news website operational
  - Assume high up-time requirement
  - Dynamically changing conditions
- What takes place?

What knowledge, model, cognitive tasks are involved?
Operating Assumptions

- Monitoring mechanism provides
  - high-level properties
  - constraint evaluation
  - associated with an architectural model

- Harold has various levels of control into the system to cause changes
  - Simple restarts, start/kill process, swap process, etc.
  - Frequently performed sequence of steps are arranged into scripts

- Harold has certain company objectives in mind
  - Smooth operation
  - Performance goals
  - Security goals
Harold in Action

- Based on initial monitoring info, Harold chooses a strategy
  - Is it a performance problem?
  - Or is it a security compromise?

- Say Harold chooses a performance strategy…
  - Determines that the culprit is peaking requests within a short time frame
  - Enlarges server pool size
  - Restarts load balancer if failed

- After taking each action, Harold observes system to determine next step
  - Success ::= initial problem resolved

Based on initial monitoring info, Harold chooses a strategy

- Is it a performance problem?
- Or is it a security compromise?

- Determine that the culprit is peaking requests within a short time frame
- Enlarges server pool size
- Restarts load balancer if failed

- After taking each action, Harold observes system to determine next step
- Success ::= initial problem resolved
Harold Juggles Many Factors

- At each decision point, Harold considers several factors
  - How much resource will this action require?
  - How long will the action take?
  - How will the action affect the system?
  - This action may briefly make the system vulnerable, while the other action requires more resources, which one should I take?
Self-adaptation Concepts

- Individual action on system
  - E.g., kill process
- Script of actions
  - E.g., `enlargeServerPool`
- Trees of possible actions that Harold can take
- Harold’s decision to select a particular strategy
- Harold navigates thru a strategy
- Company objectives and operational requirements
- Factors Harold considered in choosing action
- Architectural **operators**
  - Provided by architectural style
- **Tactics**
  - Sequence of arch operators
- **Strategy**
  - Tree of tactics
  - Intermediate observations
- **Strategy selection**
- **Tactic execution** within strategy
- Adaptation **objectives** and **preference** functions
- Tactic **meta-information**, incl. cost and effect
Example Tactic

tactic switchToTextualContent (...) {

condition(s) of applicability
response time exceeds threshold

sequence of actions
change all active servers to textual mode

effect(s)
response time falls back within threshold AND all active servers are delivering textual content

}

Execution semantics
- Each cycle, adaptation engine chooses one tactic within a strategy to execute
  - Tactic is executed from beginning to end, unless exception encountered
  - Success ::= execution completed, effects matched
  - Failure ::= exception occurred mid-execution, or effects NOT matched after
- Adaptation engine observes system and repeats cycle
Example Strategy

- Strategy *improveClientLatency* (performance-oriented)
  - Strategies tackle particular problem scopes
  - Another strategy to remedy client latency might be security-oriented

```
Latency above threshold
   ↓
  enlargeServerPool

Latency below threshold          Latency above threshold
  ↓                               ↓
  done                           switchToTextual

Latency below threshold
  ↓
  done
```
Strategy Selection

How do we pick the appropriate strategy?
- “Best” remedy for system conditions while balancing multiple objectives
- Score and compare strategies on equal footing

Meta-information
- Tactic’s attribute vector
  - e.g., [latency: “low”; quality: “textual”; cost: 1; disruption: 3]
- Strategy’s branching probabilities
  - e.g., 25% chance latency remains undesirable
- Stakeholder utility preferences
  - $u_{latency}(), u_{quality}(), u_{cost}(), u_{disruption}()$
  - $w_{latency}=0.5, w_{quality}=0.3, w_{cost}=0.1, w_{disruption}=0.1$ (sum = 1)
Strategy Selection

- Apply utility theory to
  - Score strategies
  - Tradeoff between multiple dimensions

1. Score = 25

2. Score = 42

Given X, children A, B, w/ $p_A$, $p_B$...

$E_A(X) = \text{Agg}_AV(X) = p_A \times \text{Agg}_AV(A) + p_B \times \text{Agg}_AV(B) + \ldots$

3. Score = 17
Conclusion

Self-adaptation research
- Objective: empower software engineers with tool necessary to add capabilities to keep a system operational amidst change
- Concepts: strategy, tactic, objectives, selection, failure handling
- Approach: reusable infrastructure tailored to specific problems

3 takeaways
- Rainbow is an architecture-based system for adding self-adaptation to computing systems
- Self-adaptation capabilities addresses the kinds of repair strategies that are today performed manually by system administrators
- Utility theory helps make tradeoffs between strategies where multiple objectives must be considered
The End

Questions?