Self-Adaptive Systems

David Garlan
Carnegie Mellon University

Portland State University
Portland, Oregon
January 2017
About Me

- 1987 PhD at CMU
- 3 years at Tektronix developing a software product line architecture
- 1990 joined faculty at CMU
  - Began collaboration with Mary Shaw
  - Became involved in the Master in Software Engineering program, now Director
- 1992 taught first course in Architectures for Software Systems
- 1996 published book on Software Architecture with Mary Shaw
Talk Outline

- The vision of Self-Healing Systems
  - The problem and its solution
- Architecture-based self-adaptation
  - Rainbow and Stitch
- Some current research directions
  - Applications to security
  - Run-time diagnosis and fault localization
  - Human-in-the-loop adaptation
  - Other research areas
The Problem

- An important requirement for modern software-based systems

Maintain high-availability and optimal performance even in the presence of
- changes in environment
- system faults
- attacks
- changes in user needs and context
Websites Fail to Adapt

Amazon.com disrupted due to Xbox 360 the day before Black Friday, 2006: “Scheduled Maintenance” on the busiest shopping day?

Walmart.com Scheduled Maintenance

Walmart.com is temporarily unavailable while we make important upgrades to our site. We appreciate your patience and invite you to return soon.

If you need immediate assistance, please email us at help@walmart.com or call Customer Service between the hours of 6 a.m. to 1 a.m. CST at 1-800-966-6546.
A concentrated spike in mobile traffic triggered issues that led us to shut down BestBuy.com in order to take proactive measures to restore full performance.
Cost of Downtime

- Average hourly impact of downtime by industry sector

Data from *IT Performance Engineering and Measurement Strategies: Quantifying Performance Loss*, Meta Group, Stamford, CT (October 2000).
How is this addressed today?

- **Technique 1**: Build resilience directly into application code
  - Use exceptions, timeouts, and other low-level programming mechanisms

- Unfortunately, this approach is not good for
  - Locating the cause of a problem
  - Detecting “softer” system anomalies
  - Anticipating future problems
  - Maintainability: hard to add and modify adaptation policies and mechanisms
  - Handling changing objectives
  - Legacy systems: hard to retrofit later
How is this addressed today?

- **Technique 2: Human oversight**
  - Operators, system administrators, users
  - Global oversight, intelligent response

- Unfortunately, this approach is
  - Costly
  - Error-prone
  - Slow
Cost of Human Oversight

- Estimated 1/3-1/2 total IT budget to prevent or recover from crash
- “For every dollar to purchase storage, you spend $9 to have someone manage it”—Nick Tabellion
- Administrative cost: 60-75% overall cost of database ownership
- 40% of root causes of computer system outage is attributable to operator error
Stop worrying about mastermind hackers. Start worrying about the IT guy.

the weakest link often involves the inherent fallibility of humans. ... even the most skilled system administrators struggle to keep every computer at large institutions running smoothly, with the proper software updates, security patches and configurations.
A New Approach

- **Goal:** systems automatically and optimally adapt to handle
  - faults and attacks
  - variable resources and environments
  - changes in user needs

But how?

**Answer:** Move from open-loop to closed-loop systems
Example: Google File System

The Challenge

- Provide effective engineering support for making systems self-adaptive
  - Applicable to legacy systems
  - Low development cost
  - Domain-specific adaptations
  - Multiple quality dimensions
  - Easily change/augment adaptation policies and mechanisms
  - Reason about the effects of self-adaptation actions and strategies
IBM MAPE-K

Related Disciplines

Control Systems
Fault Tolerance
Software Architecture
Biology
Human Immune System
AI
Rainbow Approach

- A framework that
  - Allows one to add a control layer to existing systems
  - Uses architecture models to detect problems and reason about repair
  - Can be tailored to specific domains
  - Separates concerns through multiple extension points: probes, actuators, models, fault detection, repair

- The framework is instantiated for specific domains, systems, mechanisms, and policies
Rainbow

System Layer

Control Layer

Adaptation Manager

Model Manager

Translation Infrastructure

Target System

System Layer
Rainbow

Strategy Executor

Adaptation Manager

Architecture Evaluator

Model Manager

Gauges

Translation Infrastructure

Effectors

System API

Resource Discovery

Probes

System Layer

Target System
Self-Adaptation Example: Znn.com
Self-Adaptation Example: Znn.com

Adaptation Condition: client request-response time must fall within threshold

Possible actions:
- enlistServers
- dischargeServers
- restartWebServer
- lowerFidelity
- raiseFidelity

Possible actions for Load Balancer:
- restartLB

© David Garlan 2017
Znn.com: Rainbow Customizations

Architecture Layer

Model Manager (MM)
- Architecture Evaluator (AE)
- Adaptation Manager (AM)
- Strategy Executor (SX)

System API
- activateServer.pl
- deactivateServer.pl
- setFidelity.pl

Translation Infrastructure
- PingRTTLatency
- Bandwidth
- Load
- Fidelity
- Cost

Znn.com

ClientT.reqRespLatency <= MAX_LATENCY

Adaptation Manager (AM)
- addServer
- removeServer
- setFidelity

Effectors
- ClientT.reqRespLatency
- HttpConnT.bandwidth
- ServerT.load
- ServerT.fidelity
- ServerT.cost

Probes

System Layer

Translation Infrastructure

Znn.com
Rainbow Adaptation Decision Overview

- **Selection from a set of adaptation strategies**
  - Multiple strategies may be applicable in a particular system context

- **Language for expressing strategies as a decision tree**
  - Conditions: determine which branches are applicable
  - Actions: tactics that modify the system

- **Tree is annotated with properties that**
  - Permit selection of strategy with highest utility
  - Support formal reasoning about time, uncertainty cost and benefit
Stitch: A Language for Specifying Self-Adaptation Strategies

- **Control-system model:** Selection of next action in a strategy depends on observed effects of previous action

- **Uncertainty:** Probability of taking branch captures non-determinism in choice of action

- **Asynchrony:** Explicit timing delays to see impact

- **Value system:** Utility-based selection of best strategy allows context-sensitive adaptation
Strategy Selection

Given:
- Quality dimensions and weights (e.g., 4)
- A strategy with
  - N nodes
  - Branch probabilities as shown
  - Tactic cost-benefit attributes

Propagate cost-benefit vectors up the tree, reduced by branch probabilities
Merge expected vector with current conditions (assume: \([1025, 3.5, 0, 0]\))
Evaluate quality attributes against utility functions
Compute weighted sum to get utility score

Score = 0.58

Algorithm
Given tree \(g\) with node \(x\) and its children \(c\):
\[
EAAV(g) = \text{sysAV} + \text{AggAV(root}(g))
\]
\[
\text{AggAV}(x) = \text{cbav}(x) + \sum_c \text{prob}(x,c) \text{AggAV}(c)
\]

\(u_{\text{latency}}(), u_{\text{quality}}(), u_{\text{cost}}(), u_{\text{disruption}}() \)
\((w_{\text{latency}}, w_{\text{quality}}, w_{\text{cost}}, w_{\text{disruption}}) = (0.5, 0.3, 0.1, 0.1) \uparrow 1\)
System Adapts

Data shows that our adaptation approach improves overall system performance.

Control run

Adaptation run

Latency = 2 secs
System Administration Evaluation

- **Sys-admin interviews**
  - **Results**: Stitch concepts seem *natural* fit for sys-admin routines
  - Methodology: priming, interview, compose Stitch script from scenarios
  - White problem scenarios: scripts represented in Stitch
  - Actual problem scenarios: structure matches Stitch strategies

- **Analysis using CMU sys-admin example: Netbwe**
  - **Results:**
    - Rainbow captured adaptation concerns
    - Stitch *hoisted* policies buried in Perl code
  - Distinguishable adaptation tasks
    - Core commands as operators
    - Coarser-grained sequence of commands (step) with conditions of applicability and intended effects
    - Adaptations with intermediate condition-actions and observations
Self-adaptive Systems Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation
4. Combining Reactive and Deliberative Adaptation
5. Proactive and latency-aware adaptation
6. Architecting for Adaptability
7. Systems of systems
Application to Security

- **Application-layer Denial of Service Attacks**
  - Assume an N-tiered model similar to Znn.com

- **Quality objectives**

<table>
<thead>
<tr>
<th>Quality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Request-response time for legitimate users</td>
</tr>
<tr>
<td>Cost</td>
<td>Number of active servers</td>
</tr>
<tr>
<td>Maliciousness</td>
<td>Percentage of malicious clients</td>
</tr>
<tr>
<td>Annoyance</td>
<td>Disruptive side-effects of tactics</td>
</tr>
</tbody>
</table>

© David Garlan 2017
# Tactics

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add capacity</td>
<td>Activate additional servers to distribute the workload</td>
</tr>
<tr>
<td>Blackhole</td>
<td>Blacklist clients; requests are dropped</td>
</tr>
<tr>
<td>Reduce service</td>
<td>Reduce content fidelity level (e.g., text vs. images)</td>
</tr>
<tr>
<td>Throttle</td>
<td>Limit the rate of requests Accepted by the system</td>
</tr>
<tr>
<td>Captcha</td>
<td>Forward requests to Captcha processor to verify that the requester is human</td>
</tr>
<tr>
<td>Reauthenticate</td>
<td>Force clients to reauthenticate</td>
</tr>
</tbody>
</table>
## Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgun/Absorb</td>
<td>Combines Add capacity and Reduce service</td>
</tr>
<tr>
<td>Eliminate</td>
<td>Combines Blackholing and Throttling</td>
</tr>
<tr>
<td>Challenge</td>
<td>Combines Captcha and Reauthenticate</td>
</tr>
</tbody>
</table>

© David Garlan 2017
Tactics and Strategies

```java
1 tactic addCaptcha () {
2 condition {exists lb:D.ZNewsLBT in M.components | lb.captchaEnabled;}
3 action {
4   set lbs = {select l : D.ZNewsLBT in M.components | l.captchaEnable}
5   for (D.ZNewsLBT l : lbs) {
6     M.setCaptchaEnabled (l, true);
7   }
8 }
9 effect {forall lb:D.ZNewsLBT in M.components | lb.captchaEnabled;}
10 }
```

```
strategy Challenge [unhandledMalicious || unhandledSuspicious] {
1 t0: (cNotChallenging) -> addCaptcha () @[5000] {
2   t0a: (success) -> done;
3   t0b: (default) -> fail;
4 }
5 t1: (lNotChallenging) -> forceReauthentication () @[5000] {
6   t1a: (success) -> done;
7   t1b: (default) -> fail;
8 }
9 }
```

```
[addCaptcha()]
-250,-80,+0.25,+50

[forceReauthentication()]
-250,-70,0,+50
```

```
addCaptcha() 0.5
-250,-90,+0.5,+50

forceReauthentication() 0.5
-250,-70,0,+50
```

```
fail [0,0,0,0]
done [0,0,0,0]
fail [0,0,0,0]
done [0,0,0,0]
```
Formal Model – Utility Profile

- Utility profile encodes functions and preferences as reward structures

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Priority</th>
<th>$w_{UR}$</th>
<th>$w_{UM}$</th>
<th>$w_{UC}$</th>
<th>$w_{UA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Minimizing number of malicious clients.</td>
<td>0.15</td>
<td>0.6</td>
<td>0.1</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>Optimizing good client experience.</td>
<td>0.3</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>Keeping cost within budget.</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>

### DoS utility profile encoding

**Formula:**

```plaintext
formula uM = (mc>=0 & mc <=5? 1:0)
  + (mc>5 & mc <=20? 1+(0.80-1)*((mc-5)/(20-5)):0)
  + (mc>20 & mc <=50? 0.80+(0.40-0.80)*((mc-20)/(50-20)):0)
  + (mc>50 & mc <=70? 0.40+(0.00-0.40)*((mc-50)/(70-50)):0)
  + (mc>70? 0:0);
```

**Rewards:**

```
rewards "rGU" // Global Utility
  leaf & scenario=1 : 0.15*uR + 0.6*uM + 0.1*uC + 0.15*uA;
... endrewards
```
Results

- Different security strategies are picked in different contexts
  - Not hardwired into the system
- Allows combinations of security repair tactics
  - Can create many strategies from the same tactics
- Supports formal reasoning and model checking
  - We use the PRISM probabilistic model checker to analyze strategies
- Allows future addition of security strategies as new tactics become available
Evaluation

No Adaptation

Minimize Malicious Clients

Prioritize Performance
Strategy Selection Analysis

- Based on quantifying expected utility after strategy execution
- Different preferences result in different strategy selections
- Choices are consistent

Minimize malicious clients

Optimize good client experience

© David Garlan 2017
Self-adaptive System Technical Challenges

1. **Self-securing systems**
2. Fault diagnosis and localization
3. **Human-in-the-loop adaptation**
4. **Combining Reactive and Deliberative Adaptation**
5. **Proactive and latency-aware adaptation**
6. **Architecting for Adaptability**
7. **Systems of systems**
Fault Diagnosis and Localization

- Successful adaptation requires detecting when there is a problem and locating the source of it

- This is a hard problem because:
  - Many possible causes for an observed problem
  - We have incomplete knowledge of the system
  - Many concurrent execution threads
  - Problems may be intermittent
  - May involve combinations
  - Must be done in real time
Approach

Five step pipeline:
1. Detect *transactions* that map interleaved, concurrent system events to distinct paths in the system
2. Determine whether transactions are successful
3. Create a set of transactions that can be used for analysis
4. Use spectrum-based multiple fault localization to diagnose problems
5. Pass this information to consumers for further action
Example

- Web-based system using multiple servers and dispatchers to serve clients
- Multiple concurrent communication threads
Transaction Families

- **Transaction families** define a parameterized pattern of behaviors
  - Light-weight specification of behavior
  - Define the finite executions
  - Criteria for success/failure
Detecting Transactions

- Map system events to architecture observations
  - Adapt work from dynamic architecture reconstruction* to map events and monitor transaction family instances

- Determine whether the transaction passes or fails
  - Success criteria defined with transaction family

Evaluating Transactions

<table>
<thead>
<tr>
<th>Txn</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Evaluating Transactions

<table>
<thead>
<tr>
<th>Txn</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>T2</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>

…and so on
Localization

- Use a technique called “Spectrum-based Fault Localization”
Samsung Case Study

- Sponsored by Samsung
- Diagnosis for Manufacturing Control Systems
  - Stringent requirements for up-time
- Key challenge is scalability and performance
  - High volume of monitored events
  - Many components
  - Must diagnose problem quickly
- Successful demonstration
  - Simulated system
  - Can handle thousands of events and find real failures
Target System

- Large scale industrial system for manufacturing of semiconductors.
  - System controls wafer manufacture, deciding which systems are used to process what.
  - System is divided into multiple components exchanging messages over an event bus.

- Typical failures
  - Messages are lost (or not sent at all)
  - Messages are sent too late
  - Unexpected messages are sent
  - Database performance slowdowns, affecting overall system performance
Samsung Challenges

Why is it difficult to diagnose failures in this system?

- Protocols work correctly most of the time.
- Problems are serious, but rare: a lot needs to be monitored to see a failure happening.
- Given the volume of data (~2000 messages / second) it is not possible for human operators to identify problems quickly.
- The complexity of the system makes it difficult for developers to figure out where the true source of a problem is.
Simulated System
The TKIN Protocol
Results

- **Can handle high volume in real time**
  - Thousands of events

- **Diagnosis time is low**
  - Under 20 seconds for all classes of failure modeled

- **Accuracy is high**
  - Rankings are consistent with actual faults
Additional challenges

- **Diagnose faults even when we can’t monitor all components**
  - Monitoring is expensive
  - Examples: (a) Samsung doesn’t directly monitor its databases, which must have maximum performance; (b) Power for sensor nets.

- **Determine an optimal placement of probes**
  - Depends on many factors: cost of probe, ability to repair, likelihood of failure

- **Adapt probes at run-time**
  - Example: increase visibility when you think there’s a problem
Self-adaptive System Technical Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation
4. Combining Reactive and Deliberative Adaptation
5. Proactive and latency-aware adaptation
6. Architecting for Adaptability
7. Systems of systems
Human-in-the-loop Adaptation

- Real systems often require humans and automated systems to collaborate
- Humans may be involved in different ways

© David Garlan 2017
Challenges for Human “Actuation”

- Different humans have different capabilities, permissions, roles, and mental states
  - Varying human attention and readiness to be involved
- The same effect may be accomplished with an automatic mechanism
  - Time-scale differences
  - Effectiveness differences
- Implies the need for a way to determine when to involve the user
Model for Human Involvement

- **Opportunity-Willingness-Capability Model (OWC)**
  - Inspiration from human-cyber design

**Opportunity:**
- Is the human in a position to carry out an action
- E.g., Physically located on site? Access to the room? Has permissions?

**Capability:**
- How likely the human is to succeed at the task
- E.g., level of training, seniority, experience.

**Willingness:**
- How likely the human is to do the task if asked
- E.g., level of attention, stress, incentives

---

Integration with Rainbow

- Some tactics are enacted by humans
- **Opportunity** is captured in strategy conditions
- **Willingness** and **Capability** affect probabilities
- **Timing** captured by delay -- human tactics usually have longer delays than automated execution
- Normal strategy evaluation and execution can then be used
Some Additional Self-adaptive System Technical Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation
4. Combining Reactive and Deliberative Adaptation
5. Proactive and latency-aware adaptation
6. Architecting for Adaptability
7. Systems of systems
Conclusion

- Today’s systems must adapt to meet dynamically changing environments, failures, attacks, requirements
- **Architecture models and an adaptation language** can be combined for effective self-adaptation
- **Rainbow**: a framework for MAPE-based adaptation
  - Uses architecture models and a provides a language for self-adaptation
  - Supports the ability to evolve and reason about self-adaptation capabilities
- Self-adaptation is an active area of research with many challenges, but huge potential to impact the design and implementation of systems.
To get involved ...

- **SEAMS**: International Symposium on Software Engineering for Adaptive and Self-managing Systems
  - Co-located with ICSE
  - Conference: May 22-23 in Buenos Aires

- Other conferences
  - **ICAC**: International Conference on Autonomic Computing, mid-July
  - **SASO**: Foundations and Applications of Self* Systems, September
References

Rainbow

Diagnosis


Proactivity

Planning


Human-in-the-loop


Security