Architecting Resilience: Handling Malicious and Accidental Threats

Rogério de Lemos
University of Kent, UK

Javier Cámara
CMU, USA

Chris Bailey
University of Kent, UK

Carlos Eduardo da Silva
UFRN, Brazil

EASSy 2013
NII Shonan Village Centre. April 2013
Resilience according to Laprie
- persistence of service delivery that can justifiably be trusted, when facing changes
- attaining dependability or security considering change
  - accidental and malicious threats

“Architecting” according to Rechting
- the process of creating and building architectures
  - the architect influences the whole development process
- the art and science of creating and building complex systems
  - scope, structure and certification
Resilience: Justification of Trust

- Related to **provision of assurances**
- Based on building **arguments** about system resilience
- Development- and run-time **evidence**
  - collection
    - relies on verification and validation techniques, rigorous design, etc.
    - e.g., model checking, sat solvers, testing, etc.
  - structuring
    - in the from of resilience cases (dependability or safety cases)
  - analysis
    - build the arguments
Outline of the Talk

- Integration Testing
  - generation of plans for managing testing

- Evaluating Resilience
  - controller and system through stimulation of changeload

- Self-Adaptive Authorisation Infrastructures
  - insider threats
Let’s have break... Let’s be reflective...

- **Does it make sense to test self-adaptive system at run-time?**
  - If positive, what would be needed?

- **How effective is empirical evidence when evaluating the resilience of self-adaptive systems?**
  - Should it be done at development-time or run-time?

- **How useful is self-adaptation in dealing with insider threats?**
  - What should be monitored, analysed and controlled?
EASSy 2013
Architecting Resilience: Handling Malicious and Accidental Threats

Part 1: Integration Testing
Integration Testing in Self-adaptive Systems

- During architectural reconfiguration software testing is usually neglected
  - challenging to test all possible architectural configurations at development-time
  - new components allow configurations not envisioned at development-time

- Uncertainty and variability affects adaptation process
  - changing goals, unexpected resource conditions, and unpredictable environments
Integration Testing in Self-adaptive Systems

Architectural reconfiguration

system
controller
  P   A
  E   AM
  M

voter
SP1
SP2
SP3
SP4
SP5
comp
SP1
SP2
SP3
voter
SP4

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
Integration Testing in Self-adaptive Systems

system

controller

integration test manager

target system

Integra0n Tes0ng in Self-­‐adap0ve Systems

target system

controller

integration test manager

target system
Integration Testing in Self-adaptive Systems

- A framework and tool support for the dynamic generation of plans for integration testing [SEAMS 2011]
  - workflows
    - represent and execute the plan
  - AI planning
    - AI planner generates a sequence of tasks to achieve a goal
  - MDE transformations
    - translate domain specific models into planning problems
Integration Testing in Self-adaptive Systems

- Process for generating dynamic plans

Flowchart:
1. Obtain current configuration
2. Obtain abstract configuration
3. Calculate integration order
4. Translate into pre/post
5. Run planner
6. Translate into workflow

Exceptional outcome:
1. Extract abstract tasks
2. Obtain concrete configuration
3. Replace tasks parameters
4. Obtain test cases
5. Associate concrete tasks

Key:
- Start
- End
- Strategic activity
- Tactical activity
- Normal flow
- Exceptional flow
- From execution level
Integration Testing in Self-adaptive Systems

- Generated workflow with several sub-workflows
  - IntegrateComponent: reconfiguration
  - TestComponent: test cases

Key:
- Start
- End
- XOR-split task
- XOR-join task
Example of sub-workflow

- TestComponent: test cases selected based on goals
EASSy 2013
Architecting Resilience: Handling Malicious and Accidental Threats

Part 2: Evaluating Resilience
Evaluating Resilience

- Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems
  - Resilience evaluation based on environmental stimuli [SEAMS 2012]
  - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems [Computing 2013]
  - framework and changeload
  - Robustness evaluation of controllers [LADC 2013]
  - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation [SEAMS 2013]
  - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
  - evaluating system properties by injecting faults
Self-Adaptive Software System – Our Model

Self-Adaptive Software System

Controller

Target System

Environment
Non-controllable software, hardware, network, physical context

Effectors

Monitor

Affect

Adapt

Probes

Monitor

Inputs

Outputs

Monitor
Evaluating Resilience

- Stepwise progress for the **provision of assurances** about the resilience of self-adaptive software systems
  - Resilience evaluation based on environmental stimuli [SEAMS 2012]
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems [Computing 2013]
    - framework and changeload
  - Robustness evaluation of controllers [LADC 2013]
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation [SEAMS 2013]
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults
A **changeload** is a set of **representative** change scenarios
- specification to stimulate system and environment

A **change scenario**
- **system state**
- **environment state** under which tasks are performed
  - including hardware/software resource conditions needed to provide service
- set of **system goals**
- set of **changes** in
  - system
  - environment
  - system goals
- an implicit time frame
Framework Overview

1. Changeload Identification
   1a. Environment Stimulation
       - Static Analysis
         - Environmental Variables
       - Dynamic Analysis
         - Environment Changes + Impact
       - Filtering/Cutoff
         - Environmental Changeload
   1b. Architecture-driven System Stimulation
       - Static Analysis
         - System Variables + Operators
       - Dynamic Analysis
         - System and Environment Changes + Impact
       - Filtering/Cutoff
         - Changeload

2. Run-time Stimulation
   - Criteria Definition
   - Metrics
   - Experimentation
   - Execution Data
   - Scoring/Ranking
     - Evaluation of Adaptation by Comparison
**Goal:** Exercising the different adaptation alternatives in the running system to obtain evidence for comparison

**Three stages**
- criteria definition
  - metrics defined according to system goals
- experimentation
- scoring/Ranking
  - evaluation by comparison
Evaluating Resilience

- Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems
  - Resilience evaluation based on environmental stimuli [SEAMS 2012]
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems [Computing 2013]
    - framework and changeload
  - Robustness evaluation of controllers [LADC 2013]
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation [SEAMS 2013]
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults
Robustness Tests

- Based on mutation of input received from probes
- General structure of controller input coming from probes
  - Name of variable being updated
  - New variable value
  - Timestamp providing temporal context
- Rainbow: messages encoded as text strings sent back to the controller

[ timestamp ] variable name : variable value
The Rainbow Framework
Controller Failure Mode Classification

Adapted version of the CRASH* Scale

- **Catastrophic**: whole controller crashes or becomes corrupted
  - Might include the OS or machine on which the controller runs
- **Restart**: controller hangs and needs external reboot
  - Within worst case execution time of adaptation cycle
- **Abort**: abnormal behavior due to an exception raised at run-time in the controller
- **Silent**: controller fails to acknowledge an error
- **Hindering**: Controller fails to return correct error code

# Robustness Tests - Mutation Rules

<table>
<thead>
<tr>
<th>Type</th>
<th>Rule Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Message</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Rule Name</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Rule Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Timestamp</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Rule Name</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Rule Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Variable Name</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Rule Name</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Rule Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Variable Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Rule Name</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>
Changeload Identification

1. Identify workload and operating conditions
   - Able to drive the controller through its different operational stages

2. Identify set of probes used through different controller stages

3. Identify set of applicable mutation rules
   - According to probes identified

4. Instantiate changes using mutation rules
   - Incorporated into change scenarios
   - One change scenario per mutation rule+controller stage
Experimental Evaluation

- 96 tests (32 applicable mutations X 3 controller operational stages)
- 48 tests (50%) uncovered robustness issues
- No Catastrophic, Restart, or Hindering issues detected
  - Limited observability
  - Build specific error detectors
  - Additional controller input
- Similar pattern of robustness issues identified at different controller stages
  - Justified by Rainbow’s architecture
  - Some variations relevant when considering ensemble controller plus system
- Additional robustness issues raised outside of the controller

<table>
<thead>
<tr>
<th>Mutation Rule</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>MsgNull</td>
<td>1</td>
</tr>
<tr>
<td>MsgEmpty</td>
<td>1</td>
</tr>
<tr>
<td>MsgPredefined</td>
<td>1</td>
</tr>
<tr>
<td>MsgNonPrintable</td>
<td>1</td>
</tr>
<tr>
<td>MsgAddNonPrintable</td>
<td>1</td>
</tr>
<tr>
<td>TSEmpty</td>
<td>1</td>
</tr>
<tr>
<td>TSRemove</td>
<td>1</td>
</tr>
<tr>
<td>VNRemove</td>
<td>1</td>
</tr>
<tr>
<td>VVRemove</td>
<td>1</td>
</tr>
<tr>
<td>VVInvalidFormat</td>
<td>1</td>
</tr>
<tr>
<td>VVNumAbsoluteMinusOne</td>
<td>1</td>
</tr>
<tr>
<td>VVNumMax</td>
<td>1</td>
</tr>
<tr>
<td>VVNumMin</td>
<td>1</td>
</tr>
<tr>
<td>VVNumMaxPlusOne</td>
<td>1</td>
</tr>
<tr>
<td>VVNumMinMinusOne</td>
<td>1</td>
</tr>
<tr>
<td>VVNumMinRangeMinusOne</td>
<td>1</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>1</td>
</tr>
</tbody>
</table>
Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems

- Resilience evaluation based on environmental stimuli [SEAMS 2012]
  - probabilistic model-checking for obtaining levels of confidence
- Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems [Computing 2013]
  - framework and changeload
- Robustness evaluation of controllers [LADC 2013]
  - injection of faults for evaluating Rainbow
- Effectiveness of architecture-based self-adaptation [SEAMS 2013]
  - effort in evolving industrial middleware
- Robustness-driven resilience evaluation of self-adaptive software systems
  - evaluating system properties by injecting faults
Integrating Rainbow and DCAS

Remove hardwired adaptation logic from DCAS, replacing it by external control exerted by Rainbow

- **Evolution of DCAS**
  - Remove built-in adaptation logic
  - Expose part of DCAS functionality
    - Public interface to probes and effectors
  - Enable Rainbow-DCAS communication
    - Lightweight TCP Server

- **Customizing the Rainbow Framework**
  - Model DCAS architecture
  - Implementing probes, effectors, gauges
  - Scripting adaptation
## Evaluation – Implementation Effort

### Rainbow Customization

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (h)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture modeling</td>
<td>20</td>
<td>21.9</td>
</tr>
<tr>
<td>Implementing client probes and gauges</td>
<td>22</td>
<td>24.1</td>
</tr>
<tr>
<td>Implementing client effectors</td>
<td>12</td>
<td>13.1</td>
</tr>
<tr>
<td>Scripting adaptation (tactics and strategies)</td>
<td>35</td>
<td>38.4</td>
</tr>
<tr>
<td>Miscellaneous configurations</td>
<td>2</td>
<td>2.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>91</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

### DCAS Evolution

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (h)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementing TCP server</td>
<td>15</td>
<td>10.3</td>
</tr>
<tr>
<td>Identifying and removing built-in adaptation</td>
<td>40</td>
<td>27.5</td>
</tr>
<tr>
<td>Implementing probes</td>
<td>45</td>
<td>31</td>
</tr>
<tr>
<td>Implementing effectors</td>
<td>35</td>
<td>24.1</td>
</tr>
<tr>
<td>Miscellaneous configurations</td>
<td>10</td>
<td>6.8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>145</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>
Evaluating Resilience

- Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems
  - Resilience evaluation based on environmental stimuli [SEAMS 2012]
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems [Computing 2013]
    - framework and changeload
  - Robustness evaluation of controllers [LADC 2013]
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation [SEAMS 2013]
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults
EASSy 2013
Architecting Resilience: Handling Malicious and Accidental Threats

Part 3: Self-Adaptive Authorisation Infrastructures
Authorization infrastructures protect, control and monitor access to electronic resources
- federated authorization infrastructures become increasingly difficult to manage

Malicious behavior can be seen as abuse of access caused by insider threat within federated authorization infrastructures

Self-adaptation is seen as a means to improve the management of malicious behavior, by adapting authorization policies and access rights
- Adapt to mitigate malicious behavior, and prevent future attacks
Federated Authorization

Kent University + 3\textsuperscript{rd} Party Services
United Kingdom Access Management Federation (UKAMF)
Challenges of Managing Federations

- Heavily dependent on manual processes to identify and resolve malicious behaviour
- Involves multiple management domains
- Privacy protected users and large unknown user base
- Built on authorization infrastructures which are not typically designed to reflect on user behaviour
Self-Adaptive Authorization

SP Management Domain

SAAF Controller
- Plan
- Analyze
- Execute
- Model
- Monitor

Effector

Probe

Authorization Service

Resource

1. Access resource

2a. Request

2b. Decision

3. Monitor use

4. Adapt policies

System

Environment

1. Access resource

2a. Request
Cross-Domain Adaptation

- Federations imply distributed authorization infrastructures with multiple management domains.

- Identity Providers (IdPs) and Service Providers (SPs) have to work together to resolve malicious activity, else jeopardise trust.

- Solution: Deploy an Identity Provider (IdP) Effector
  - facilitates adaptation of IdP assets
  - allows IdP organisation to govern who can adapt and the extent of adaptation.
Self-Adaptive Federated Authorization

1. Access resource

2a. Request

2b. Decision

3. Monitor use

4a. Adapt policies

4b. Adapt identities

0. Authenticate and obtain privacy protected identity

Effector

Authentication / Identity Services

IdP Management Domain

SP Management Domain

SAAF Controller

Plan

Analyze

Execute

Model

Monitor

Authorization Service

Effector

Probe

Resource

System

Environment

System

Environment

Information and Communication Technologies Institute

Carnegie Mellon | PORTUGAL
Simulating Malicious Behaviour

- The SAAF controller classifies malicious behaviour with a behaviour rule
  - No single subject should access the payroll service with a greater rate of 10 access requests per minute

- User is allowed to access the payroll resource using his/her IdP identity assigned attribute ‘permisRole=Contractor’

- User ‘Bob Doe’ breaks this rule by executing a high rate of requests to the payroll resource
Self-Adaptation affecting IdP Service

1. Monitor use
2. Identify and act on malicious behavior
3. Send SOAP request over SSL requesting resolution
4. Obtain user identity
5. Authorize and update LDAP directory

User: Bob Doe

IdP Management Domain:
- SimpleSAMLphp IdP
- SimpleSAMLphp Effector
- LDAP

SP Management Domain:
- SAAF Controller
- PERMIS
- Payroll Service
- Policy Enforcement Point (PEP)

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
In Summary...

- **Progress on the resilience evaluation of self-adaptive systems**
  - evidence can be obtained by testing (run-time) or stimulations (development-time)
  - challenge is how to collect, structure and analyse evidence

- **Protection against insider attacks based on self-adaptation**
  - evaluation of SAAF through malicious changeload
  - look into more sophisticated ways of detecting malicious behaviour
Questions?

Thank you!

Thanks to
- Javier Camara
- Chris Bailey
- Carlos Eduardo da Silva