University of Trento
Department of Information Engineering
and Computer Science

Requirements and Architectural Approaches to Adaptive Software Systems: A Comparative Study

Konstantinos Angelopoulos, Vítor E. Silva Souza
João Pimentel

angelopoulos@disi.unitn.it
vitorsouza@inf.ufes.br
jhcp@cin.ufpe.br

Outline

- Motivation
- Comparison Process
- Case Study
- Architecture-based approach
- Requirements-based approach
- Approach Comparison
- Conclusions

Motivation

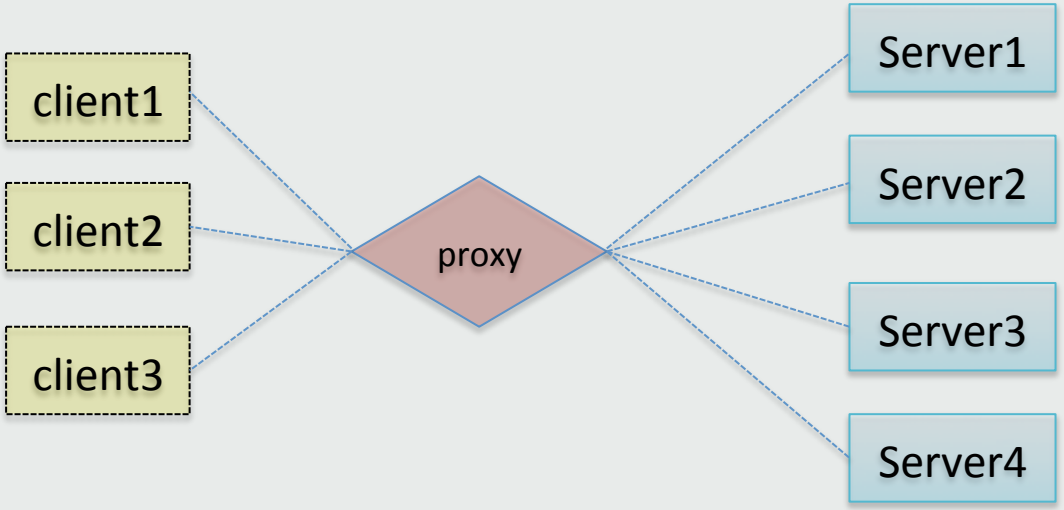
- Many approaches for software adaptation adopt *requirements* or *architectural* models.
- We propose to conduct a comparison experiment that answers questions such as:
  - What aspects of a problem/solution do these types of models capture?
  - What are their advantages and disadvantages?
  - Can we develop approaches to adaptation that use both types of models synergistically? (future work)

Comparison Process

- Use Zanshin for requirements-based adaptation and Rainbow for architecture-based adaptation.
- Use the Znn.com (news portal) case study, an exemplar for the SEAMS community.
- Apply Zanshin and Rainbow to the case study.
- Compare solutions in terms of:
  - common concepts adopted
  - models used
  - monitoring and effecting mechanisms
  - adaptation mechanisms

Znn.com Case Study

Znn.com news portal



Objectives:

1. Low Cost
2. High Fidelity
3. High Performance

Adaptation strategies for balancing traffic:

1. add/remove servers
2. increase/decrease fidelity

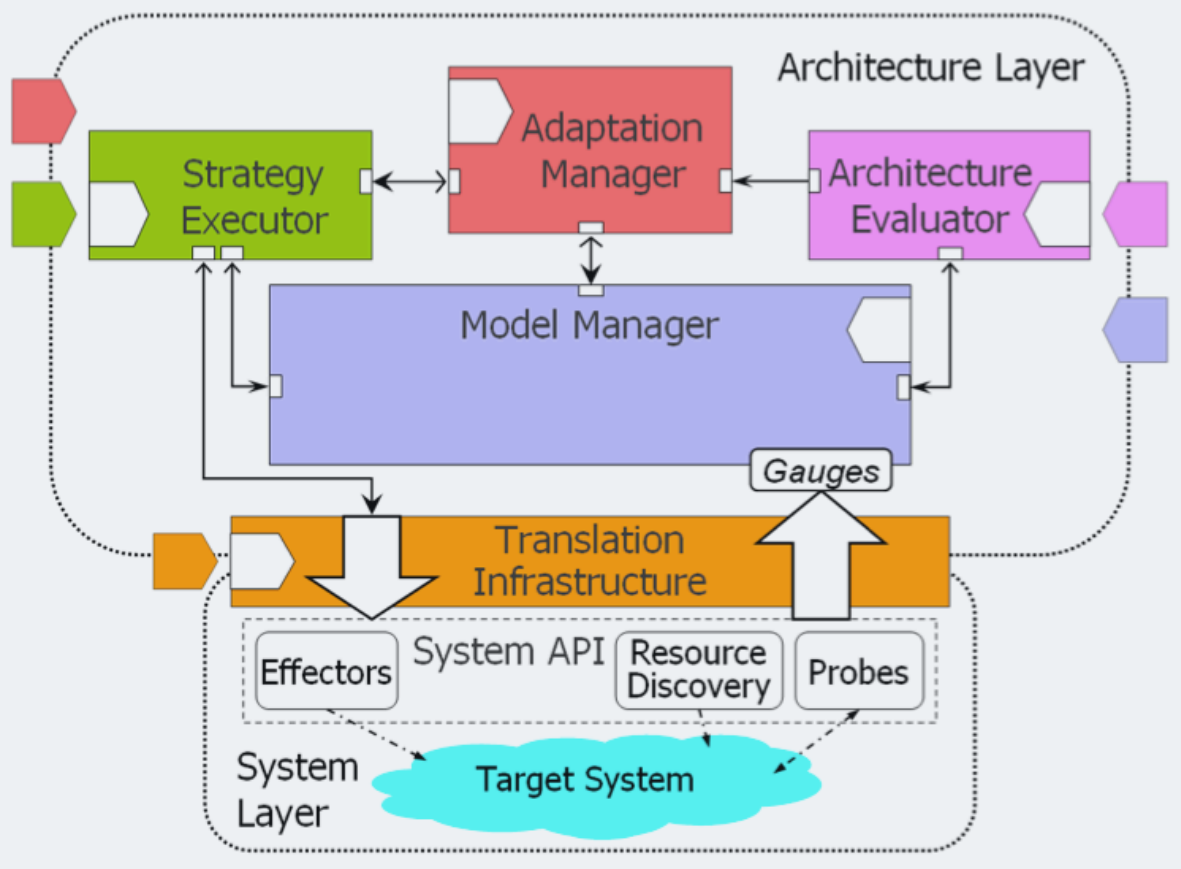
Architecture-based Adaptation (Rainbow) 1/2

Baseline:

- Adopts **feedback loop** concepts from Control Theory.
- **Architectural models** (ACME) describe target system.
- **Decision mechanisms** (based on Utility Theory) to select adaptation strategies.
- Script language (Stitch) to compose **adaptation strategies**

Architecture-based Adaptation (Rainbow) 2/2

Overview:



The components of the Rainbow framework [Cheng08]

Architecture-based Solution

- An ACME model describes the system's architecture
- Adaptation strategies in Stich:

  SimpleReduceResponseTime:
  reduce fidelity, if response time still low then reduce again

  SmarterReduceResponseTime:
  add server, add server, reduce fidelity until response time is low

  ReduceOverallCost:
  If response time low then remove servers

  ImproveOverallFidelity:
  If response time is low raise fidelity

```
 strategy SmarterReduceResponseTime
[styleApplies&&cViolation]{
  define boolean unhappy = numUnhappyFloat/
numClients > M.TOLERABLE_PERCENT_UNHAPPY;

t0:unhappy -> enlistServers(1)@[500/*ms*/]{
  t1:(!cViolation) -> done;
  t2:(unhappy) -> enlistServers(1)@[2000/*ms*/]{
    t2a:(!cViolation) -> done;
    t2b:(unhappy) -> lowerFidelity(2,100)@[2000/*ms*/]{
      t2b1:(!cViolation) -> done;
      t2b2(unhappy) -> do[1]t2;
      t2b3(default) -> TNULL; //no more steps to take
      }
   }
}
```

- Detect objective violations having as a reference the architectural model
- Select strategy to apply
- Apply Strategy

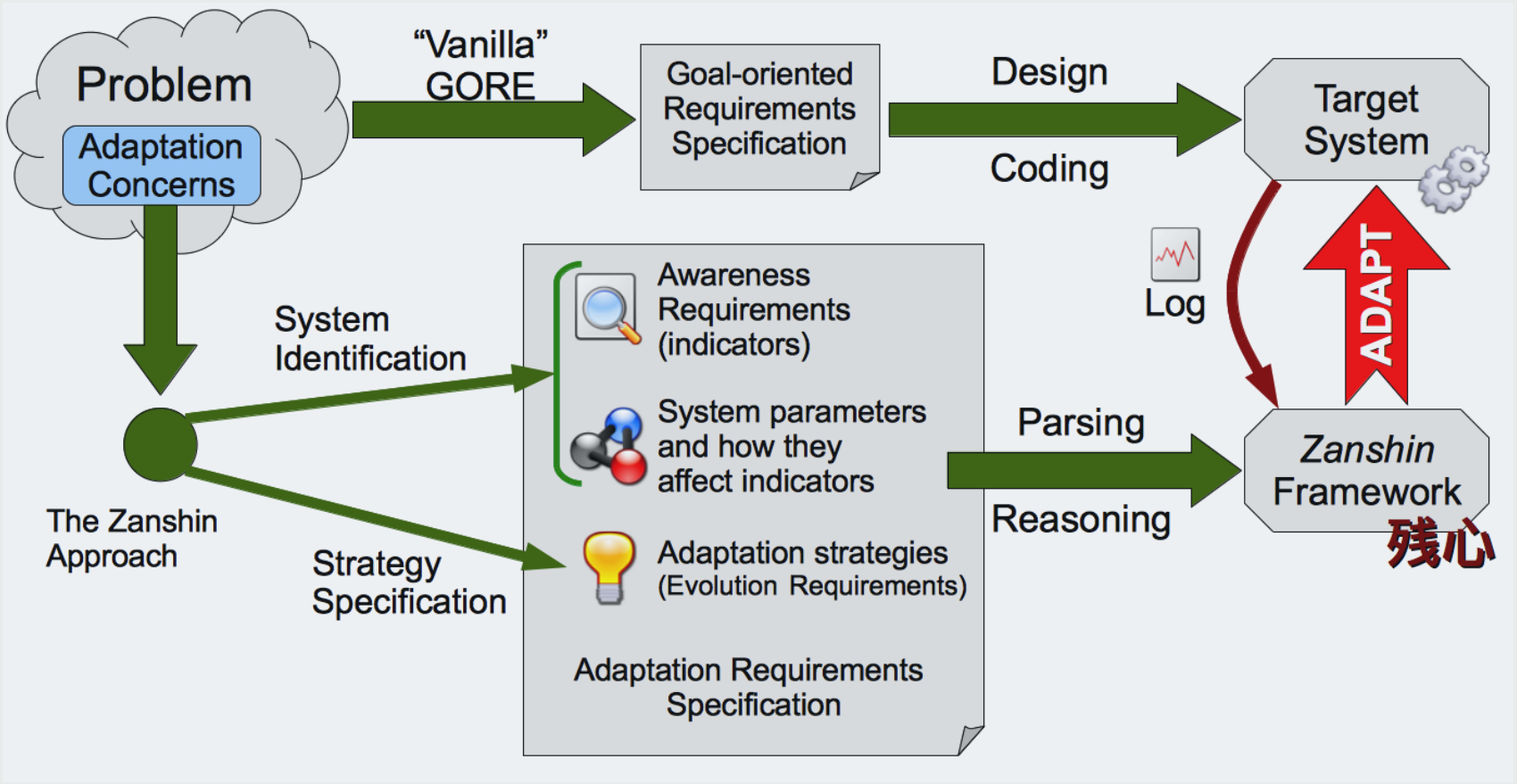
Requirements-based Adaptation (Zanshin) 1/2

Baseline:

- **Awareness requirements**: Define allowable thresholds on the success/failure of other requirements
- **System Identification**: define the parameters of the system (CV and VP) and the impact over indicators (e.g. servers↑ then performance↑)
- **Adaptation**: a) Reconfiguration by changing parameter values or b) Evolution requirements (e.g. relax a constraint from 2.5sec to 3sec)

Requirements-based Adaptation (Zanshin) 2/2

Overview:



Requirements-based Solution

- Elicit goals
- System Identification:

AwReq AR1: softgoal *Cost efficiency* should never fail
- Checked at: every second
- Adaptation Strategy 1.1: Reconfigure(∅)
  - Applicability Condition: there are no active sessions for AR3

Δ(AR 1/NoS) [0, maxServers] < 0 (1)

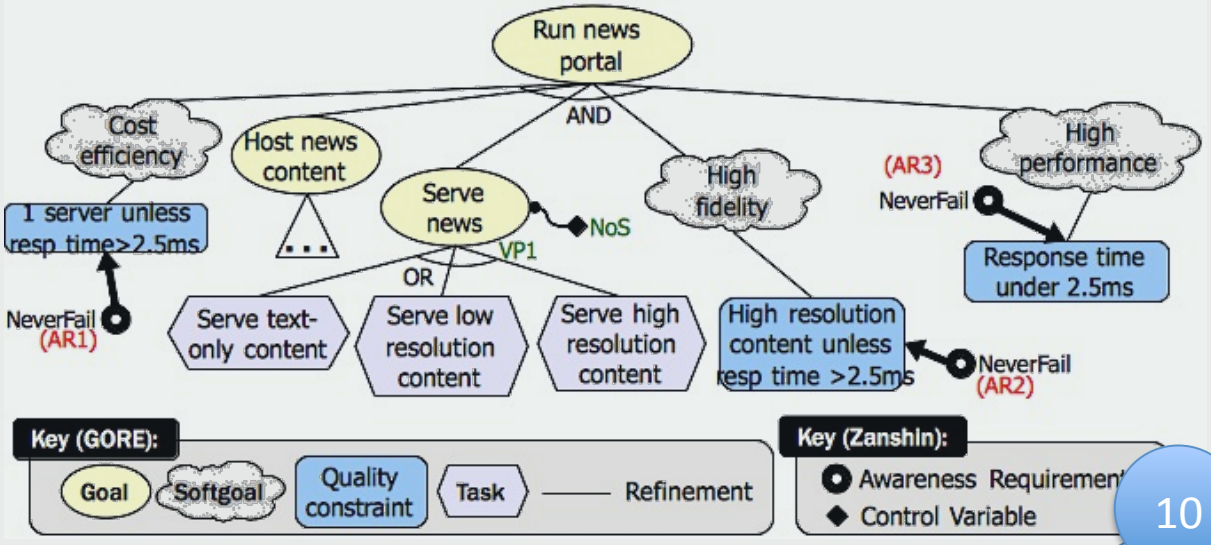
Δ(AR 3/NoS) [0, maxServers] > 0 (2)

Δ(AR 2/VP1) > 0 (3)

Δ(AR 3/VP1) < 0 (4)

AwReq AR2: softgoal *High fidelity* should never fail
- Checked at: every request
- Adaptation Strategy 2.1: ChangeParam(VP1, high)
  - Applicability Condition: there are no active sessions for AR3

- Define Strategies



Experiment and Results

Infrastructure:

- 5 Apache servers (4 hosts, 1 proxy)
- 1 MySql db server
- Apache Jmeter load tester tool

We run 2 trials of a high traffic scenario (Slashdot effect) with and without adaptation mechanisms:

- Rainbow [Cheng09]:
  - Improved the response time by 75%
  - The throughput by 7%
  - Utilities of the objectives were also increased
- Zanshin:
  - Response time improved by 67.4%
  - The throughput by 8.7%
  - Awareness requirements failures were reduced

Comparison Overview

Both Approaches
work well in the study, adopt **feedback loop** concept, apply **external control, pre-conditions** and **post-conditions** for adaptation strategies

Rainbow (Architecture-based)	Zanshin (Requirements-based)
• Capture technical properties and constraints (reusable models) • Requirements are embedded in adaptation strategies • Hierarchic adaptation language (automates administrative processes) • Quantitative adaptation using utilities (human experience)	• Capture strategic goals (stakeholders needs) • Requirements are explicitly captured in a model • Evolution requirements and reconfiguration (offers dynamic strategy composition) • Qualitative adaptation using control theory

Conclusions

- The architecture-based approach:
  - ✓ captures better the properties of the target system
  - x requirements are implicitly represented
  - ✓ captures precisely human administration process
  - x only automates control
  - ✓ Utility Theory allows a quantitative control
- The requirements-based approach:
  - ✓ captures explicitly the goals of the system
  - x doesn't capture the technical limitations of the system
  - ✓ allows the dynamic composition of adaptation strategies
  - ✓ Qualitative control (useful when numbers are not available)
- The approaches include complementary features

References

- [Cheng08] S.-W. Cheng, “Rainbow: Cost-Effective Software Architecture-based Self-adaptation,” Ph.D. dissertation, Carnegie Mellon University, 2008.
- [Souza12] V. E. S. Souza, “Requirements-based Software System Adaptation,” PhD Thesis, University of Trento, Italy, 2012.
- [Cheng09] S.-W. Cheng, D. Garlan, and B. Schmerl, “Evaluating the Effectiveness of the Rainbow Self-Adaptive System,” in Proc. of the ICSE 2009 Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE, 2009, pp. 132–141.

Thank You!

Questions?

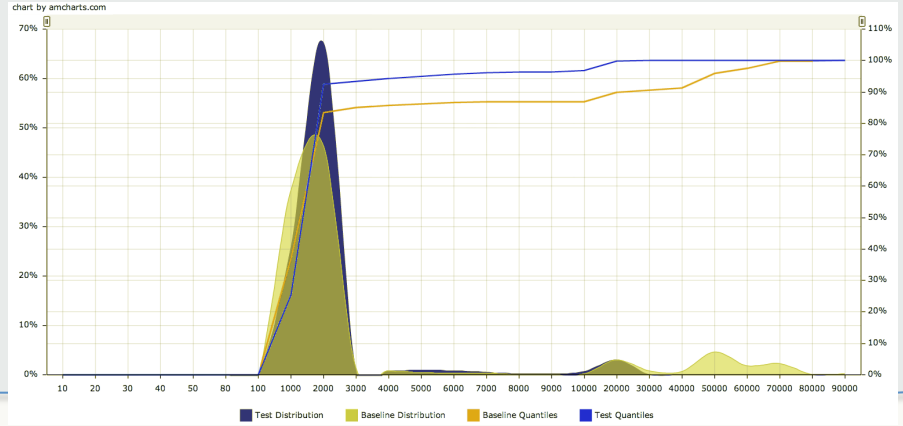
```
rterReduceResponseTime
&& cViolation ] {
ean unhappy = numUnhappyFloat/numClients > M.TOLERABLE_PERCENT_UNHAPPY;

) -> enlistServers(1) @[500 /*ms*/] {
ion) -> done;
y) -> enlistServers(1) @[2000 /*ms*/] {
lation) -> done;
ppy) -> lowerFidelity(2, 100) @[2000 /*ms*/] {
Violation) -> done;
happy) -> do[1] t2;
fault) -> TNULL;  // in this case, we have no more steps to take
```

Thank Y

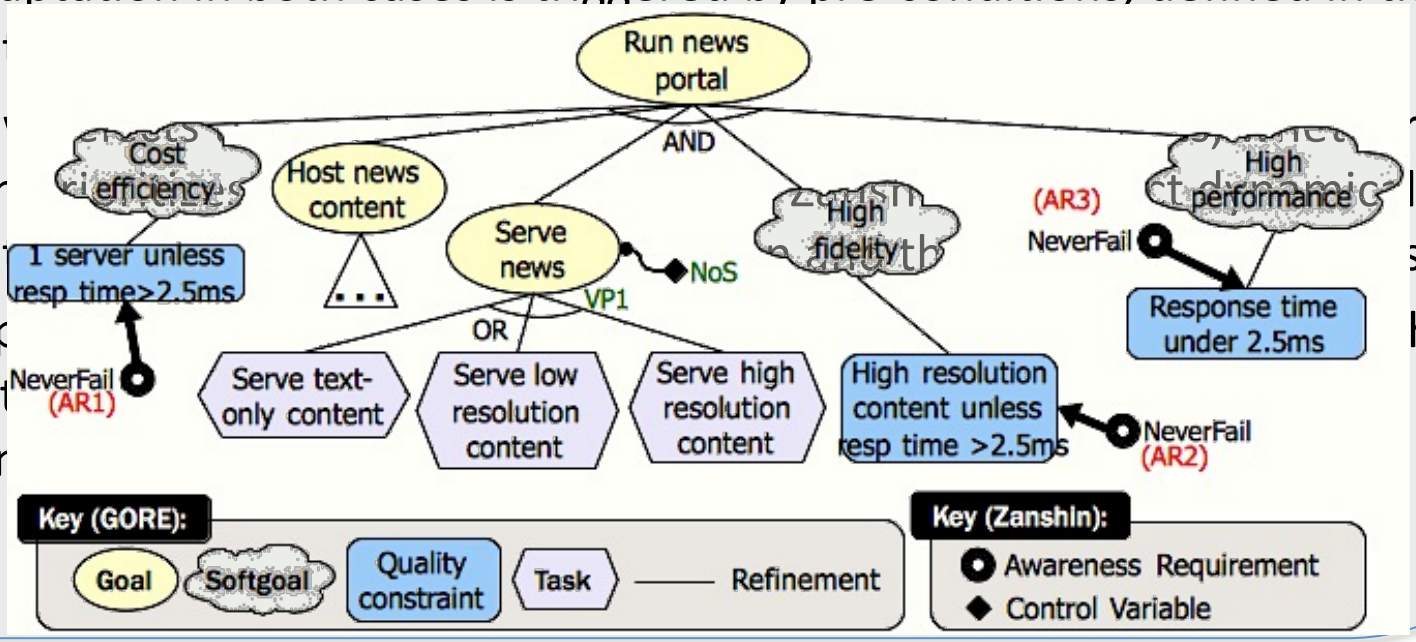
```
strategy SmarterReduceResponseTime
[styleApplies&&cViolation]{
  define boolean unhappy = numUnhappyFlo
> M.TOLERABLE_PERCENT_UNHAPPY;

t0:unhappy -> enlistServers(1)@[500/*ms*/
  t1:(!cViolation) -> done;
  t2:(unhappy) -> enlistServers(1)@[2000/*m
    t2a:(!cViolation) -> done;
    t2b:(unhappy) -> lowerFidelity(2,100)@[2
      t2b1:(!cViolation) -> done;
      t2b2(unhappy) -> do[1]t2;
      t2b3(default) -> TNULL; //no more steps
      }
  }
}
```



Comparison Overview

1. Both approaches adopt a closed loop model and apply external control.
2. Rainbow exploits architecture models that represent all the technical details, while Zanshin exploits goal models that capture tasks and strategic goals.
3. Rainbow uses hierarchically composed strategies (strategies ⊇ tactics ⊇ operators), while Zanshin uses reconfiguration and evolution requirements.
4. The adaptation in both cases is triggered by pre-conditions, defined in the adapta
5. Rainbo Zanshir adapta
6. Both ap instruct probler



AwReq AR1: softgoal *Cost efficiency* should never fail
- Checked at: every second
- Adaptation Strategy 1.1: Reconfigure(∅)
  - Applicability Condition: there are no active sessions for AR3

AwReq AR2: softgoal *High fidelity* should never fail
- Checked at: every request
- Adaptation Strategy 2.1: ChangeParam(VP1, high)
  - Applicability Condition: there are no active sessions for AR3

Req AR3: softgoal *High Performance* should never fail
- Checked at: every request
- Adaptation Strategy 3.1: ChangeParam(VP1, low)
  - Applicability Condition: this is the first failure
- Adaptation Strategy 3.2: Do Nothing
  - Applicability Condition: AS3.1 applied last, less than 1s ago
- Adaptation Strategy 3.3: ChangeParam(VP1, text-only)
  - Applicability Condition: AS3.1 applied last, more than 1s ago
- Adaptation Strategy 3.4: Do Nothing
  - Applicability Condition: AS3.3 applied last, less than 3s ago
- Resolution Condition: AR3 was satisfied AND:
  - AS3.1 was applied last, more than 1s ago OR
  - AS3.3 was applied last, more than 3s ago