Efficient Testing of Self-Adaptive Behaviors in Collective Adaptive Systems

Yoo Jin Lim  
Server Technologies Division  
Oracle Korea  
Seoul, Republic of Korea  
yoo.jin.lim@oracle.com

Eunkyoung Jee, Donghwan Shin, Doo-Hwan Bae  
School of Computing  
KAIST  
Daejeon, Republic of Korea  
{ekjee, donghwan, bae}@se.kaist.ac.kr

Abstract—Collective adaptive systems (CAS) consist of multiple agents that adapt to changing system and environmental conditions in order to satisfy system goals and quality requirements. As more applications involve using CAS in a critical context, ensuring the correct and safe adaptive behaviors of quality-driven CAS has become more important. In this paper, we propose Collective Adaptive System Testing (CAST), a scalable and efficient approach to testing self-adaptive behaviors of CAS. We propose a selective method to instantiate and execute test cases relevant to the current adaptation context. This enables testers to focus testing on key self-adaptive behaviors while dealing with the scale and dynamicity of the system. An experimental evaluation using a traffic monitoring system is performed to validate its scalability, efficiency, and fault-detection effectiveness. The experimental results provide insights into how CAST can serve as a feasible and effective assurance technique for CAS.

Keywords—software quality attributes; system testing; collective adaptive system; agent-based adaptive system

I. INTRODUCTION

Collective Adaptive Systems (CAS) are large-scale self-adaptive systems that consist of multiple agents, which collaborate to achieve goals and adapt along the way as their operational contexts change [1]. CAS are open and dynamic—the agents are autonomous and may enter or leave the collective organization at any time. Such dynamic adaptation capability of CAS has been applied to enhance the quality of systems from various domains, including intelligent transportation, healthcare, and even space missions [2-4]. Many of the applications involve using CAS in a critical context, where failing to meet the adaptation requirements can lead to serious damages. Ensuring the correct and safe adaptive behaviors of quality-driven CAS thus remains an important research issue for this promising technology.

Recently many studies have applied model checking to verify self-adaptive system properties [5-6]. These studies utilize well-established model checking tools such as SPIN [7], NuSMV [8], and Uppaal [9] to support automated verification. However, the state explosion problem inherent in model checking remains a major limitation in applying these techniques for verification of CAS. The open and dynamic nature of CAS significantly increases the number of system states to be checked and thus exacerbates the state explosion problem.

On the other hand, several studies have proposed testing as a means of assuring the quality of adaptive systems. Some techniques use goal models to generate integration test cases [10-11]. Others work on continuous testing approaches to check functional requirements satisfaction [12-13]. These existing approaches lack explicit considerations for self-adaptive behaviors and dynamic reorganization in systems. For each change that triggers adaptation, it is vital to verify that agents adapt correctly afterwards and satisfy the quality requirements. Furthermore, it is generally infeasible to specify and check the entire system after each adaptation, which inures an excessive amount of memory and time.

In this paper, we address the aforementioned assurance needs by proposing an efficient approach to test self-adaptive behaviors of CAS in dynamic environments. The testing approach, called Collective Adaptive System Testing (CAST), provides a scalable and efficient way to verify that CAS executes correctly when adapting to dynamic environments. The CAST approach works by monitoring the running system to select and execute tests relevant to the current adaptation context. The selective instantiation of quality requirements-based test cases provides efficient assessments as to whether agents are behaving properly. Finally, test results indicate quality requirements satisfaction by self-adaptive behaviors in CAS.

CAST is evaluated by applying it to the traffic monitoring system (TMS) application [6], to validate the scalability and efficiency of CAST in practice. Experimental results indicate that CAST is efficient in testing self-adaptive behaviors in large-scale CAS configurations and effective in finding faults related to self-adaptation.

The contributions in this paper are summarized as follows:

- We propose a testing approach that explicitly considers self-adaptive behaviors and quality requirements satisfaction in CAS.
- We propose a selective method to testing parts of the system relevant to the current adaptation context, enabling testers to focus on self-adaptive behaviors
of the system when its state space is large and complex.

- We evaluate our approach with a simulation-based experiment using a collective adaptive system application and validate its usefulness and fault-detection ability in practice.

The remainder of this paper is organized as follows. Section 2 provides background information, and Section 3 introduces the CAST approach with the TMS as its motivating example. Section 4 presents the experimental evaluation and results. Section 5 presents related work, and Section 6 concludes the paper with future research directions.

II. BACKGROUND

A. Adaptation Properties of Self-Adaptive Systems

Self-adaptation approaches are used to satisfy quality requirements for systems operating in dynamic environments. To consider the satisfaction of the quality requirements, we focus on the adaptation properties which can be directly mapped to quality requirements and observed in the system after adaptation. Definitions for three of the adaptation properties fundamental to self-adaptive systems as well as CAS are found below:

**Flexibility.** By IEEE definition [14], flexibility is “the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.” In the context of self-adaptive systems and CAS, it is the ability to adapt dynamically to the changing environmental conditions [6].

**Robustness.** It is defined [14] as “the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.” In the context of self-adaptive systems and CAS, it is the ability to detect, adapt to, and recover from operational problems; though it may be in a degraded system state [15].

**Openness.** In multi-agent systems, it is defined as the ability to deal with agents that enter or leave the system at will [15]. The same concept applies to CAS as well, and it is an important property for the system to meet its scalability needs.

B. Traffic Monitoring System (TMS)

1) **Functional Description:** Intelligent transportation systems aim to improve road conditions by providing reliable traffic information [6]. The TMS plays the role of monitoring and providing information about traffic congestions. The TMS with camera agents distributed over a road network can be considered and designed as a CAS. An example TMS introduced in [16] achieves the traffic monitoring function with self-adaptive behaviors. Fig. 1 shows the TMS in self-adaptation scenarios as specified in [6]. The TMS operates from time T0 and goes through adaptations from time T1 to T4.

Each camera is capable of monitoring the traffic within its viewing range. When the traffic condition is normal, each camera is a single member organization. To monitor traffic congestions, camera agents must perform collective adaptation and collaborate in organizations. This allows the system to accurately inform on where the traffic congestion starts and ends. In every organization, a camera serves as the Master (“M” in Fig. 1) reporting the monitored information to clients, while other cameras remain as Slaves. As for a single member organization, it serves as the Single Master.

2) **Quality Requirements:** The TMS is expected to achieve three key quality requirements with self-adaptation:

- **Flexibility.** In the TMS, cameras reorganize themselves depending on the traffic condition on the road. For example, T1 in Fig. 1 shows three cameras merged into one organization. Once the congestion ends, the collective organization breaks up and the cameras return to operating individually.

- **Robustness.** In the TMS, failure in a camera disrupts the service and may put the system in an inconsistent state. For example, at times T2 and T3 in Fig. 1, Camera 5 has failed in Organization 35 (Org35 in Fig. 1). To deal with this failure, the remaining cameras in the organization elect a new master, and continue its collaboration on traffic monitoring.

- **Openness.** In the TMS, openness concerns the addition of a new camera. To satisfy openness, the newly added camera is properly integrated with the existing camera agents. In T4 from Fig. 1, Camera 5 has returned to the scene as a new camera. Since the traffic congestion is still present on the road, it joins Organization 35, this time as a slave.

III. TESTING SELF-ADAPTIVE BEHAVIORS

A. Approach Overview

We propose a scalable and efficient technique for testing self-adaptive behaviors of CAS. Fig. 2 is an overview of the CAST approach applied to the TMS example. The CAST approach requires two inputs. First, an executable prototype...
of the CAS must be provided to simulate the system running in a dynamic environment. The running system is assumed to be well-instrumented, to produce system execution traces. Even though the system itself runs on decentralized control among agents, the execution traces contain information over all agents in the system.

A test specification is also needed as a basis to select and instantiate test case templates. It defines elements to monitor and expected output of the adaptation, representing the quality requirements to be satisfied through self-adaptive behaviors. Thus, it contains test case templates providing full coverage of quality requirements specification. An excerpt from the test specification for the TMS is found in Table I.

In each test case, the adaptation triggering event describes a specific event to be monitored. This is considered to be the test input for each test case. The test oracle part is a template, to be instantiated with a specific identification number of the entity based on the current adaptation context. The test oracle templates specify system entities (e.g. Camera and Organization) or objects (e.g. NeighborList, CameraList, etc.) to be monitored and its expected status following the adaptation. They are written using assert statements for the convenience of test execution.

As described in Section II.B.2, three quality requirements need to be satisfied in the TMS: robustness (R), openness (O), and flexibility (F). For example, Test Cases R-1 and R-2 are selected when a camera failure occurs in the system. Test Case R-1 checks that the camera failure has been detected by other cameras. The NeighborList of a camera containing the failCamera is expected to be false. If all test cases in the Robustness section pass, then the system has satisfied its robustness requirement with its self-adaptive behavior. In a similar way, the test specification can be extended to address other quality requirements and specify complex self-adaptation scenarios.

B. System Monitoring and Analyzing

The goal of monitoring step is to identify the adaptation triggering event and the system states before and after adaptation. Once an adaptation triggering event has been detected, the analyzing step begins based on execution traces.

For analysis, execution traces from before and after adaptation are compared to identify the system entities affected by the current adaptation. This is done automatically by identifying differences in the two traces. If any change is detected, the corresponding entity is subjected to testing. A list of system entities changed by the adaptation is produced.

C. Test Planning and Executing

The first step in planning is to reason about which test cases to execute. The test case templates relevant to the current adaptation context are obtained according to the test specification. To make the test case executable, the oracle templates are instantiated for the changed system entities only. Thus, a set of test cases are selectively instantiated for parts of the system relevant to the current adaptation context.

Finally, the test cases are executed on the CAS. The test results indicate satisfaction or violation of quality requirements. This information can be used to discover faulty self-adaptive behaviors or environmental scenarios that caused quality requirements violation.

IV. EXPERIMENTAL EVALUATION

The aim of the experiments is to validate the proposed approach. The CAST approach was implemented as a prototype, and the experimental evaluation was conducted on the Traffic Monitoring System (TMS) described in Section II.B. The following three research questions are answered:

RQ1. Does CAST show better scalability than model checking in verifying self-adaptation behaviors?

RQ2. How does the method of oracle instantiations (selective vs. all) affect the efficiency of CAST?

RQ3. Can CAST detect faults in adaptation mechanisms of collective adaptive systems?

All experiments were performed on a Windows machine with a 3.4GHz Intel Core i5 CPU processor.

A. Experimental Setup and Design

The TMS test specification, newly specified in this work, contains 12 separate test oracle templates, in which one tests system invariant, five test robustness, five test openness, and one tests flexibility. The system invariant checks that every organization has a master. The robustness tests check for failure detection and continual function in organizations. The openness tests check for correct registration of the newly

---

TABLE I. TRAFFIC MONITORING SYSTEM TEST SPECIFICATION EXAMPLES

<table>
<thead>
<tr>
<th>Adaptation Triggering Event</th>
<th>Test Oracle Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera Failure</td>
<td>R-1: assertFalse(Cam(x).NeighborList.contains(failCam))</td>
</tr>
<tr>
<td>Camera Addition</td>
<td>O-1: assertTrue(Cam(x).NeighborList.contains(newCam))</td>
</tr>
<tr>
<td>Traffic Congestion</td>
<td>F-1: assertTrue(Org (x).size &gt; 1)</td>
</tr>
</tbody>
</table>
added camera. The flexibility test checks for organizational response to traffic congestion on the road.

**RQ1.** First, the scalability of model checking with respect to a property of the TMS is assessed. Based on a previous case study of formal verification with the TMS by Iftikhar and Weyns [6], this experiment uses the Uppaal model checker and the system invariant as the verification property, which checks that all cameras cannot be slaves at the same time. The amount of time required to verify the invariant property is measured as the total number of cameras in the system increases from 6 to until verification is possible without state explosion. The maximum number of cameras is recorded, to assess the scalability limits of Uppaal model checking for comparison with CAST.

**RQ2.** Next, we compared the efficiency of testing the TMS with the CAST-selected test cases and all-selected test cases in terms of the testing time. The CAST-selected test cases refer to the test cases that result from instantiating test oracle templates selectively for system entities relevant to the current adaptation. The all-selected test cases refer to the test cases that result from instantiating the templates for all system entities without considering the current adaptation. We applied the CAST-selected and all-selected methods, respectively, to test 10 different TMS versions with various sizes, which ranged from containing 17 to 340 cameras in the system. In each experiment setting, a simulation scenario was randomly generated and simulated on the TMS with CAST running in parallel for testing. The average values of testing time taken by the CAST-selected and all-selected methods are compared as the number of cameras increases in the TMS.

**RQ3.** To further evaluate the efficacy of CAST, we also injected artificial faults into the system under test. We specifically targeted parts of the source code related to adaptation mechanisms. For example, a statement was deleted to prevent updating the organization’s camera list following after a camera was newly added. This fault should affect the self-adaptive behavior responsible for satisfying openness. We created a total of six copies of the TMS, with each version containing one fault in self-adaptive behaviors. The goal of this experiment is to assess the fault detection effectiveness of CAST with both its selected and all-selected methods of test oracle instantiation. After fault injection, we check if testing with CAST can detect quality requirements violations as the faulty version of the TMS runs.

**B. Experimental Results**

This section presents experimental results and analysis to answer the aforementioned three research questions.

**RQ1.** The verification time with Uppaal is recorded, and Fig. 3 shows the growth of time. As Fig. 3 shows, the verification time sub-exponentially with the number of cameras until the count reaches 62. Clearly, the rapid growth of verification time limits the applicability of Uppaal model checking to only a small system, scaling up to only 62 cameras in the TMS case. Further experiments with larger numbers of cameras were not feasible due to the state explosion problem. While exhaustive exploration through model checking guarantees the correctness of the system, it

Fig. 3. Uppaal Verification Time for Increasing Number of Cameras

poses a severe scalability limitation to verifying self-adaptive behavioral properties of CAS.

While the scalability of Uppaal model checking was limited to 62 cameras in the system, the following experiments show that the CAST approach makes verification possible for TMS up to 340 cameras. It should be noted that with the CAST approach, the TMS can be tested at a scale larger than 340, by focusing on parts of the system related to self-adaptation

**RQ2.** Fig. 4 presents plots of the average time accumulated for testing the TMS. Testing with the CAST-selected method takes significantly less time than testing with the all-selected method, as the number of cameras in the system increases. The plot shows that the testing time with the CAST-selected method remains constant (i.e., around 107 milliseconds) with the increasing number of cameras in the TMS. This is because the CAST-selected method instantiates test oracles only for the system entities affected by adaptation, regardless of the total number of agents in the system. On the other hand, the testing time with the all-selected method grows linearly with the number of cameras in the TMS. This is because the CAST-selected method instantiates test oracles only for the system entities affected by adaptation, regardless of the total number of agents in the system. The difference in testing time between the two methods is negligible for system sizes with fewer than 100 cameras, but it grows significantly wider as the number of cameras increases. These observations complete the scalability evaluation from RQ1 and confirm that the CAST approach can scale well to large systems.

**B. Experimental Results**

This section presents experimental results and analysis to answer the aforementioned three research questions.

**RQ1.** The verification time with Uppaal is recorded, and Fig. 3 shows the growth of time. As Fig. 3 shows, the verification time sub-exponentially with the number of cameras until the count reaches 62. Clearly, the rapid growth of verification time limits the applicability of Uppaal model checking to only a small system, scaling up to only 62 cameras in the TMS case. Further experiments with larger numbers of cameras were not feasible due to the state explosion problem. While exhaustive exploration through model checking guarantees the correctness of the system, it

Fig 4. Comparison of Testing Time between CAST-selective and All-selected Methods

poses a severe scalability limitation to verifying self-adaptive behavioral properties of CAS.

While the scalability of Uppaal model checking was limited to 62 cameras in the system, the following experiments show that the CAST approach makes verification possible for TMS up to 340 cameras. It should be noted that with the CAST approach, the TMS can be tested at a scale larger than 340, by focusing on parts of the system related to self-adaptation

**RQ2.** Fig. 4 presents plots of the average time accumulated for testing the TMS. Testing with the CAST-selected method takes significantly less time than testing with the all-selected method, as the number of cameras in the system increases. The plot shows that the testing time with the CAST-selected method remains constant (i.e., around 107 milliseconds) with the increasing number of cameras in the TMS. This is because the CAST-selected method instantiates test oracles only for the system entities affected by adaptation, regardless of the total number of agents in the system. On the other hand, the testing time with the all-selected method grows linearly with the number of cameras (i.e., from 117 milliseconds for 17 cameras to 247 milliseconds for 340 cameras). The difference in testing time between the two methods is negligible for system sizes with fewer than 100 cameras, but it grows significantly wider as the number of cameras increases. These observations complete the scalability evaluation from RQ1 and confirm that the CAST approach can scale well to large systems.

**B. Experimental Results**

This section presents experimental results and analysis to answer the aforementioned three research questions.

**RQ1.** The verification time with Uppaal is recorded, and Fig. 3 shows the growth of time. As Fig. 3 shows, the verification time sub-exponentially with the number of cameras until the count reaches 62. Clearly, the rapid growth of verification time limits the applicability of Uppaal model checking to only a small system, scaling up to only 62 cameras in the TMS case. Further experiments with larger numbers of cameras were not feasible due to the state explosion problem. While exhaustive exploration through model checking guarantees the correctness of the system, it...
A. SLAVE NODES

RQ3. We analyze the fault detection effectiveness of CAST based on its ability to detect artificial faults injected in the adaptation mechanisms of the TMS. Table II shows that all of the injected faults were detected by both CAST-selected and all-selected methods of test oracle instantiation. For example, in version F1, a fault was injected into a method that elects a new master in the organization, to affect the ability of the system to ensure that all organizations have a master camera. When F1 of the TMS is running and a slave camera fails, both the CAST-selected and all-selected methods return failed test cases for the system invariant requirement.

Based on the results of the experiment, we can notice that the CAST-selected method is just as effective as the all-selected method in detecting faults related to self-adaptive behaviors. Since CAST is shown to detect all faults injected in the adaptation mechanism of CAS, we can conclude that CAST can selectively instantiate test oracles to take less time for testing while maintaining the necessary fault detection effectiveness.

C. Threats to Validity

The TMS served as an example CAS by describing cameras as agents that self-reorganize in the road environment with changing traffic conditions. One threat to validity is the ability of CAST to achieve its efficiency and fault detection effectiveness in testing adaptive systems from other domains or with different types of faults. Another threat is the quality of test oracle templates in the test specification, which requires considerable effort from the system or test engineer. The representativeness of the system quality requirements is important with respect to the test result accuracy. Lastly, our experimental evaluation was based on simulation which does not show performance impact of testing a live, running system.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Adaptation-Triggering Event</th>
<th>Requirement Violated</th>
<th>Detected (CAST-selected)</th>
<th>Detected (All-selected)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>SlaveNode Fail</td>
<td>Invariant</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F2</td>
<td>SlaveNode Fail</td>
<td>R: Continual Function by Subject Org</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F3</td>
<td>MasterWith-Slaves Fail</td>
<td>Invariant</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F4</td>
<td>SlaveNode Addition</td>
<td>O: Addition Detection by Cameras</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F5</td>
<td>SingleMaster Addition</td>
<td>O: Addition Detection by Subject Org</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F6</td>
<td>SlaveNode Addition</td>
<td>O: Continual Function by Subject Org</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

V. RELATED WORK

To complement the strides made in research for adaptation mechanisms, the research community has begun to place a greater emphasis on developing the assurance techniques as well [5]. A comprehensive survey [2] on assurance of adaptive systems has identified key properties specifically found in adaptive systems. In recent years, different methods have been used to verify those properties.

First, there are several works for model checking of adaptive systems. They include well-established model checking tools such as Uppaal [9] and SPIN [7]. For example, in a recent work by Iftikhar and Weyns [17], the adaptation components are modelled as a network of timed-automata and then Uppaal is used for verification. In addition, the SPIN model checker was used to verify a nurse agent system with self-adaptive behaviors [18]. However, these approaches have restricted applicability to CAS due to its openly large scale and dynamic operating environment. To that end, a testing approach is necessary to realize scalable verification for self-adaptive behaviors in CAS.

Many studies have also focused on testing adaptive systems. Nguyen et al. [13] introduced a continuous testing framework called eCAT for multi-agent systems. In the framework, a tester agent checks the agent interactions and continual satisfaction of system goals. Ramirez et al. [19] proposed an approach to discover environmental conditions that produce requirement violations in a self-adaptive system. Akour et al. [20] presented an approach in which they update runtime test models following dynamic adaptation. The test model contains the test information necessary for running the test cases, such as the test stubs, instead of capturing the adaptive system behavior. Welsh and Sawyer [21] proposed a technique that uses model analysis to reduce the set of test cases in testing for emergent behavior in adaptive systems. They recognize the limited testing resources at runtime and focus on a reduced number of key scenarios.

In particular, Fredericks et al. [12] presents a requirements-aware testing framework, MAPE-T, which monitors, analyzes, plans, and executes test cases at runtime. They take an evolutionary approach to realizing the framework. During test case adaptation, the test case parameters are mutated to be more aligned with the operating context. However, their approach is demonstrated using a component-based adaptive system and does not consider the dynamic addition of agents. The test oracles did not describe explicitly the self-adaptive behaviors of the subject system.

The existing testing approaches in general do not acknowledge the continuous adaptations enabled by dynamic reorganization of multiple agents. Our CAST approach, on the other hand, has elaborated the testing process with a focus on adaptations, specifically for self-adaptive behaviors that satisfy quality requirements. The scale of the subject adaptive system considered also differs. Unlike the aforementioned case studies, we additionally incorporate a multi-agent based adaptation in our subject system. It is evident that our testing approach is capable of addressing a class of adaptive systems with larger scalability. Overall, in spite of sharing the common limitations inherent to testing,
when compared to existing work in the area, our CAST approach has definitely addressed more challenging issues with respect to the scalability and efficiency of CAS testing.

VI. CONCLUSION

Given the increasing use of CAS in high-assurance domains, providing assurance for the correct and safe adaptive behavior in such systems is an essential topic. This paper proposed CAST, a scalable and efficient technique for testing self-adaptive behaviors of CAS. CAST systematically carries out testing in four steps: monitoring, analyzing, planning, and executing. Agents in the system are continuously monitored for adaptation triggering events, at which point the system states are analyzed to select test cases that correspond to the current context. The selectively instantiated test cases are then executed on the system. The experimental evaluation using a traffic monitoring system demonstrated the scalability, efficiency, and fault-detection ability of CAST. The presented testing technique has effectively addressed the dynamic aspects of CAS operations while assuring the correctness of its self-adaptive behaviors. Furthermore, the analysis of artificial faults showed how CAST effectively detects several types of faults related to self-adaptive behaviors in CAS.

For future work, the CAST approach will be elaborated to support automated derivation of test oracles from adaptation quality goal models. To further improve efficiency of the approach, we also plan to investigate other test case selection techniques.

ACKNOWLEDGMENT

This research was partly supported by the ICT R&D program of MSIP/IITP[10044457, Development of Autonomous Intelligent Collaboration Framework for Knowledge Bases and Smart Devices] and Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education(NRF-2013R1A1A2058618).

REFERENCES