Design of Intelligent Agents for Collaborative Testing of Service-Based Systems

Xiaoying BAI and Bin CHEN
Dept. of CS&T, Tsinghua University, Beijing, China, 100084

Bo MA and Yunzhan GONG
Research Institute of Networking Technology, BUPT, Beijing, China, 100876
Outline

- Research motivation
- Test agent design
- Agent-based simulation testing
  - Performance testing
  - Coverage testing
- Conclusion and future work
Outline

- Research motivation
- Test agent design
- Agent-based simulation testing
  - Performance testing
  - Coverage testing
- Conclusion and future work
Service-oriented computing enables dynamic service composition and configuration
How to Test Dynamic Changes?

- To revalidate the re-composed and re-configured service-based systems
  - Re-select test cases
  - Re-schedule test execution
  - Re-deploy test runners
  - ....

- The challenges: changes occur **ONLINE**
  - Uncontrolled
  - Un-predictable
  - Distributed
New Testing Capabilities Required

- Adaptive testing
  - The ability to sense changes in target software systems and environment, and to adjust test accordingly.

- Dynamic testing
  - The ability to re-configure and re-compose tests, and to produce, on-demand, new test data, test cases, test plan and test deployment.

- Collaborative testing
  - The ability to coordinate test executions that are distributed dispersed.
The MAST Framework

- Multi-Agent based Service Testing Framework [Bai06, Xu06, Bai07, Ma10]
  - MAS is characterized by persistence, autonomy, social ability and reactivity
  - Test agents are defined to simulate distributed service testing
    - Test Runners simulate autonomous user behaviors
    - Runners are coordinated to simulate diversified usage scenarios
Agent Intelligence is Key to Test Effectiveness

- How to simulate users behavior?
- How to sense and react to changes?
- How to collaborate to simulate various scenarios?

The Needs

Environment Knowledge Representation
Change Events Capturing
Adaptation and Collaboration Rules
Architecture Overview

Multi-Agent Based Service Testing Framework

Test Coordinator

Knowledge

Interpreter

Events

Actions

Internet

Test Runner

Knowledge

Interpreter

Events

Actions

Test Runner

Knowledge

Interpreter

Events

Actions

Internet

Services
Outline

- Research motivation
- Test agent design
- Agent-based simulation testing
  - Performance testing
  - Coverage testing
- Conclusion and future work
Basic Test Agent Definition

\[ \text{TestAgent} := \langle K, E, A, \Phi \rangle \]

- **K**: the set of knowledge
- **E**: the set of events
- **A**: the set of agent actions
- **\( \Phi \)**: the interpreter that derives an agent’s action sequences based on its knowledge and triggering events
Two Agent Types

- **Test Coordinator**
  - Analyze test requirements, generate test plans, create test runners, and allocate tasks to test runners.

- **Test Runner**
  - Accept test cases, carry test tasks to target host computers, and exercise test cases on the service under test.
Test Coordinator

- Knowledge
  - <Services, TestCases, Runners, Tasks>
    - Runners:=<ID, state, task>
    - Tasks:=<sID, tcID, result>

- Actions
  - Test Preparation
    - ParseTestScenario, GenerateRunner
  - Test Execution
    - SelectRunner, SelectTestCase, AllocateTestTask, DeployRunner

- Events
  - TEST_PARSED_OK, TEST_PARSED_ERROR
  - START_TEST
  - RUNNER_OK, RUNNER_NOT_AVAILABLE, GENERATE_RUNNER_COMPLETE
  - RUNNER_REQUEST_TASK, RUNNER_SEND_RESULT, RUNNER_UPDATE
Test Runner

- **Knowledge**
  - `<Hosts, Task, Configuration>`
    - Hosts: `<URL, Resources>`
    - Configuration: `<hID, tID, Parameters>`

- **Actions**
  - Coordination
    - AcceptTask, ReturnResult, SyncState
  - Execution
    - Migrate, ExecuteTask, CollectResult
  - Decision
    - SelectHost, RequireTestTask, ConfigTest

- **Events**
  - Task_Arrival, Task_Finish
  - Resource_Error, Host_Error, Host_Update
  - Migration
Interpreter

- Action rules identify the actions to be triggered when certain events occur.
  - assertion $\rightarrow$ action
    - assertion: predicates of system status after event occurs

- To dynamic adjust behavior according to pre-defined rules and strategies
  - Agent decision making
  - Reactive to changes
  - Adaptive behavior
Interpreter

- Event Capturing
- Rule Extraction
- Action Planning
- Action Identification
- Rule Matching
- Action Execution

Rules
Outline

- Research motivation
- Test agent design
- Agent-based simulation testing
  - Performance testing
  - Coverage testing
- Conclusion and future work
Agent-Based Simulation Testing

- The generic agent design can be applied to various testing tasks with specially designed domain knowledge, events, actions, and rules.
- Test agents automatically adjust test plans and test cases to meet test objectives.
Case Study 1: Performance Testing

- Performance testing analyzes system behavior under different usage scenarios and workloads.
  - E.g. upper limit of capacity and bottlenecks under extreme load

- Two key parameters
  - Test scenarios, the complexity of test cases
  - Workloads, the number of concurrent requests

- Case study objective
  - Try-and-test manual approach → Agents autonomous decision for adaptive selection of scenarios and workloads
## Case Study 1: Agent Design

### Rules

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td></td>
<td>FindComplexity (cMin, cMax)</td>
</tr>
<tr>
<td>Over Load</td>
<td>greater(this.cCur, cMin)∧null(runner.lCur)</td>
<td>FindComplexity (cMin, cCur)</td>
</tr>
<tr>
<td></td>
<td>greater(this.cCur, cMin)∧equals(runner.lCur, lMin)</td>
<td>Decrease-Complexity (cCur)</td>
</tr>
<tr>
<td>Light Load</td>
<td>less(this.cCur, cMax)∧null(runner.lCur)</td>
<td>FindComplexity (cCur, cMax)</td>
</tr>
<tr>
<td></td>
<td>less(this.cCur, cMax)∧equals(runner.lCur, lMax)</td>
<td>Increase-Complexity (cCur)</td>
</tr>
</tbody>
</table>

**Workload Equation**

\[
\text{workload} = \sum f(\text{complexity}_i) \times \text{load}_i
\]
Case Study 1: Experiments

- Analyze the SUT’s memory usage: read file and merge data in memory
  - Services deployed on Tomcat application server.
  - Scenario #1
    - Service is implemented using Java “StringBuilder” data type with little extra memory space.
  - Scenario #2
    - Service is implemented using Java “String” data type which takes up extra memory space for object construction.
  - Scenario #3
    - Simulate changes in server memory configuration of memory restrictions.
Case Study 1: Results

Test Result of Scenario #1

Test Result of Scenario #2

Test Result of Scenario #3
Case Study 2: Coverage Testing

- Coverage testing is to select a subset of test cases to cover as many as software features.
- The problem
  - TestEfficiency = number of features covered / number of test cases selected
- Case study objective
  - To coordinate test agents working in parallel with complementary coverage achievements
Case Study 2: Agent Design

- **Coverage Matrix**
  \[ CM = [\text{cov}_{ij}]_{m \times n}, \]
  \[ \text{cov}_{ij} = \begin{cases} 1, & b_j \in \text{Cov}(tc_i) \\ 0, & b_j \notin \text{Cov}(tc_i) \end{cases} \]

- **Similarity algorithm** is used to calculate the distance between any two coverage sets.
  \[ \text{Dis}(s_i, s_j) = 1 - \frac{|s_i \cap s_j|}{|s_i \cup s_j|} \]
Case Study 2: Experiments

- Two SUTs are exercised, each has 100 code blocks and 1000 test cases.
  - Scenario #1: test cases are sparsely overlapped, and each case has a low coverage (2%)
    \[ |Cov(tc_i) \cap Cov(tc_j)| \leq 1\% \]
  - Scenario #2: test cases are densely overlapped
    \[ |Cov(tc_i) \cap Cov(tc_j)| \geq 20\% \]
- 10 runners are deployed for each test.
  - Initialized with a randomly selected set of test cases
  - Runner cache result threshold: 3
  - Coordinator synchronize threshold: 9
## Case Study 2: Results

### Scenario #1

<table>
<thead>
<tr>
<th>Test Rounds</th>
<th>Runner1</th>
<th>Runner2</th>
<th>Runner3</th>
<th>Runner4</th>
<th>Runner5</th>
<th>Runner6</th>
<th>Runner7</th>
<th>Runner8</th>
<th>Runner9</th>
<th>Runner10</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>#2</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>#3</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Coordinator Sync

| Test Rounds | 96 | 96 | 96 | 96 | 96 | 96 | 96 | 96 | 96 | 96 |

### Scenario #2

<table>
<thead>
<tr>
<th>Test Rounds</th>
<th>Runner1</th>
<th>Runner2</th>
<th>Runner3</th>
<th>Runner4</th>
<th>Runner5</th>
<th>Runner6</th>
<th>Runner7</th>
<th>Runner8</th>
<th>Runner9</th>
<th>Runner10</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>#2</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>#3</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
</tbody>
</table>

Coordinator Sync

| Test Rounds | 77 | 77 | 77 | 77 | 77 | 77 | 77 | 77 | 77 | 77 |

### Scenario #3

<table>
<thead>
<tr>
<th>Test Rounds</th>
<th>Runner1</th>
<th>Runner2</th>
<th>Runner3</th>
<th>Runner4</th>
<th>Runner5</th>
<th>Runner6</th>
<th>Runner7</th>
<th>Runner8</th>
<th>Runner9</th>
<th>Runner10</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>80</td>
<td>80</td>
<td>81</td>
<td>81</td>
<td>81</td>
<td>81</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>#2</td>
<td>83</td>
<td>83</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>83</td>
<td>83</td>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td>#3</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>87</td>
<td>87</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>85</td>
</tr>
</tbody>
</table>

Coordinator Sync

| Test Rounds | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

### Scenario #4

<table>
<thead>
<tr>
<th>Test Rounds</th>
<th>Runner1</th>
<th>Runner2</th>
<th>Runner3</th>
<th>Runner4</th>
<th>Runner5</th>
<th>Runner6</th>
<th>Runner7</th>
<th>Runner8</th>
<th>Runner9</th>
<th>Runner10</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>#2</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>#3</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
## Case Study 2: Results

<table>
<thead>
<tr>
<th>Test Rounds</th>
<th>Runner1</th>
<th>Runner2</th>
<th>Runner3</th>
<th>Runner4</th>
<th>Runner5</th>
<th>Runner6</th>
<th>Runner7</th>
<th>Runner8</th>
<th>Runner9</th>
<th>Runner10</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>#2</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>#3</td>
<td>26</td>
<td>26</td>
<td>29</td>
<td>27</td>
<td>27</td>
<td>29</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>24</td>
</tr>
<tr>
<td>#4</td>
<td>33</td>
<td>31</td>
<td>36</td>
<td>35</td>
<td>34</td>
<td>35</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#28</td>
<td>97</td>
<td>93</td>
<td>95</td>
<td>96</td>
<td>95</td>
<td>98</td>
<td>95</td>
<td>93</td>
<td>93</td>
<td>96</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#35</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>98</td>
<td>96</td>
<td>98</td>
<td>98</td>
<td>96</td>
<td>98</td>
<td>99</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#55</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Outline

- Research motivation
- Test agent design
- Agent-based simulation testing
  - Performance testing
  - Coverage testing
- Conclusion and future work
Conclusion

- SOA systems impose new requirements of automatic and collaborative testing.
- Agent-based simulation provides a new way for SOA testing
  - Distributed deployment and dynamic migration
  - Autonomous user behavior
  - Collaborative usage scenario
  - Adaptive to environment changes
- Abstract agent model to be instantiated to address different testing tasks
- Experiments show promising improvements compared with conventional approaches
Future Work

- Agent design
  - Joint intention model for agent collaboration
- Improvement of experiments
  - Scale and complexity
- Simulation on the cloud infrastructure
Thank you!

Xiaoying Bai

Ph.D, Associate Professor
Department of Computer Science and Technology,
Tsinghua University
Beijing, China, 100084
Phone: 86-10-62794935
Email: baixy@tsinghua.edu.cn