TDD for AOP: Test-Driven Development for Agent-Oriented Programming

Cleber Jorge Amaral
Federal Institute of Santa Catarina
São José, Brazil
cleber.amaral@ifsc.edu.br

Jomi Fred Hübner
Federal University of Santa Catarina
Florianópolis, Brazil
jomi.hubner@ufsc.br

Timotheus Kampik
Umeå University and SAP Signavio
Umeå, Sweden
tkampik@cs.umu.se

ABSTRACT

This demonstration paper introduces native test-driven development capabilities that have been implemented in an agent-oriented programming language, in particular as extensions of AgentSpeak. We showcase how these capabilities can facilitate the testing and continuous integration of agents in JaCaMo multi-agent systems.

KEYWORDS

Agent-Oriented Programming, Engineering Multi-Agent Systems, Test-Driven Development

ACM Reference Format:
Cleber Jorge Amaral, Jomi Fred Hübner, and Timotheus Kampik. 2023. TDD for AOP: Test-Driven Development for Agent-Oriented Programming. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION

Agent-Oriented Programming (AOP) [9] is a programming paradigm that provides first-class abstractions for instilling autonomous behavior into software systems. Over the past decades, research on AOP has emerged as a key direction within the domain of Engineering Multi-Agent Systems (EMAS), leading to the development of a diverse range of tools and platforms that support AOP [8]. While these technologies have not yet been widely adopted by the software engineering mainstream, from an academic perspective, the technology ecosystem can be considered thriving [7].

In modern software tool-chains, testing and Quality Assurance (QA) plays a major role. This applies in particular in the context of (at least partially) autonomous and distributed agent-oriented systems, where reliable governance is a key concern [6] and implementation or operation errors can have disastrous consequences [5]. Although first works that are concerned with the development of QA-related capabilities for AOP have recently emerged [1, 2], so far, no testing utilities for any agent-oriented programming language appear to exist. In this paper, we address this issue by presenting testing capabilities for the *AgentSpeak* AOP language, in particular for the AgentSpeak dialect that is supported by the Jason interpreter [4], which in turn is part of the JaCaMo framework [3] for developing multi-agent systems. We conceptualize the testing approach, describe its architecture, provide examples, and finally conclude the paper with a future outlook.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom. © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

2 TEST-DRIVEN DEVELOPMENT FOR AGENT-ORIENTED PROGRAMMING

In modern software engineering, developers commonly apply Test-Driven Development (TDD) approaches, in which a large portion of the tests is written during or even ahead of the implementation of the actual program code. The assumption is that specifying the exact desired behavior of a software component before or alongside the implementation facilitates a more rigorous assessment of the component and ensures testing is not cut short because of time shortage caused by incorrect or imprecise estimations. Generally, it is considered good practice to focus automated testing efforts on unit tests of small components that can and hence should be tested in a rigorous manner. The overall system (or system of systems) can then be covered with less dense integration tests and End-to-End (E2E) tests. The latter cannot cover all possible input and environment configurations because of the explosion in combinatorial options (even for fairly small systems) but they can potentially catch unexpected behavior of components that seemingly work correctly from a lower-level perspective.

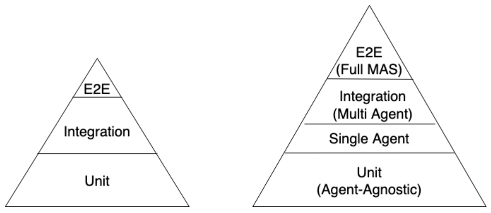

Figure 1: Overview: proposed levels of agent-oriented testing.

From an agent-oriented view, unit tests cover behavior that is independent of the *agent* abstraction and that can be tested as a function, rule, or method call without side effects: the agent's internal state, the environment and the states and behaviors of other agents do not play a role in this context. In contrast, single agent tests rely on the agent's internal state (its *beliefs*) or the state of the environment, whereas multi-agent tests consider interactions with other agents (analogous to how integration tests verify the interaction of several software components). Finally *Full MAS* (Multi-Agent System) tests cover the entire software system end-to-end (E2E), even considering non-MAS components such as reactive Web services or graphical user interfaces. Figure 1 depicts the test pyramid from a traditional software engineering perspective and contrasts it with an agent-oriented test pyramid. Note that conceptually, we consider end-to-end full MAS tests as out-of-scope for a testing technology that is tied to a specific AOP language because traditionally, end-to-end tests are not provided by programming language-specific test suits.

The testing features introduced in this paper contributes to the state-of-the-art technology-wise and conceptually. Regarding the former, they are the first testing abstractions introduced to an AOP language (unit level). Regarding the latter, they elevate the relevance of *goals* for test-driven development, checking whether an agent will eventually (or given a specific time constraint) reach a goal, either in a single-agent or multi-agent setup (single agent and multi-agent level). A demonstration video is available at https://youtu.be/395OHpuCILs.

3 TESTING CAPABILITIES OF JACAMO

To cover the three lower levels of the agent-oriented test pyramid as displayed in Figure 1, an AOP testing feature has been implemented for JaCaMo. JaCaMo's agent-oriented testing feature allows for the instantiation of one main agent per test file, as well as for the instantiation of several mock agents; in addition to mock agents, the main (to-be-tested) agent may interact with other agents (and artifacts) that have been implemented in a traditional manner. The tester agent inherits the abilities of the main agent, which enables it to have testing plans that check whether the main agent can reach an expected internal state (for example: belief adoption) or specific goals. Whenever a corresponding assertion fires, the tester agent reports the result (pass or fail). Figure 2 depicts the architecture of the testing capabilities as described above. Note that a more comprehensive tutorial is available online at https://github.com/jacamo-lang/jacamo/tree/master/doc/tutorials/tdd.

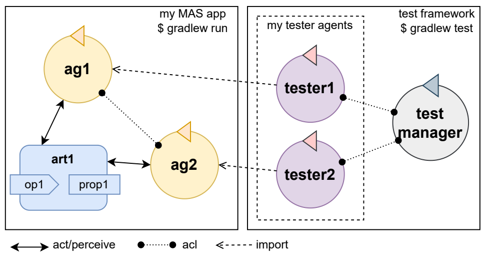

Figure 2: Architecture: testing agents in JaCaMo.

Tests in JaCaMo are plans that have a test annotation (such as `@[test]`) and that can test anything from a simple inference rule to the interaction of several agents and artifacts. Let us provide some examples that give an intuition of how tests are specified, starting with the test of a rule that determines the Manhattan distance, as provided below:

```
distance(X1,Y1,X2,Y2,D) :-
    D = math.abs(X2-X1) + math.abs(Y2-Y1).
```

As the Manhattan distance between points `(0,0)` and `(3,3)` is known to be 6, this rule can be straightforwardly tested using the following testing plan:

```
@[test]
+!testDistance : distance(0,0,3,3,D) <-
    !assert_equals(D0,6).
```

The testing of multi-agent systems, in which several agents interact with each other and act upon artifacts, requires additional features, e.g. for the instantiation of mock agents. For example, we may want to instantiate several agents, pass messages between them, and evaluate how the environment changes as a result of the agents' actions. Below, we test an agent that manages the temperature of a room, considering the preferences relayed to it by several *assistant agents*. We first instantiate the agents, then have the assistant agents communicate their preferences, and finally check if the result on the environment (i.e., on the room temperature) is the average of their preferences:

```
@[test]
+!test_multiple_preferences <-
    .create_agent(mock_ra, "mock_room_agent.asl");
    .create_agent(tims_assistant, "assistant.asl");
    .create_agent(clebers_assistant, "assistant.asl");
    .send(tims_assistant,tell,preferred_temperature(23));
    .send(tims_assistant,achieve,send_preference(mock_room_agent));
    .send(clebers_assistant,tell,preferred_temperature(25));
    .send(clebers_assistant,achieve,send_preference(mock_room_agent));
    !!pollTemperature;
    .wait(temperature(24), 200, EventTime);
    ?temperature(T);
    !assert_equals(24,T).
+!pollTemperature: temperature(24).
+!pollTemperature  <-
    .send(mock_room_agent,askOne,temperature(T));
    !pollTemperature.
```

As an initial application of the JaCaMo testing features, parts of the Jason code base were covered with new, agent-oriented tests. Because some of the tests have Java-based counterparts, a comparison of the testing approaches is possible. Below is a test of Jason's feature for setting random seeds, written in AgentSpeak:

```
@[test,atomic]
+!test_set_random_seed <-
    .set_random_seed(20);
    RET = math.random(10);
    !assert_equals(7.320,RET,0.01).
```

The Java-based pendant to the test follows below:

```
public void testRandom() throws Exception {
    Agent ag = new Agent();
    ag.initAg();
    new jason.stdlib.set_random_seed().execute(
        ag.getTS(),
        new Unifier(),
        new Term[] { ASSyntax.parseNumber("20") }
    );
    InternalAction ia_r = (InternalAction) ag.getIA(".random");
    double retFunc = new jason.functions.Random().evaluate(
        ag.getTS(),
        new Term[] { ASSyntax.parseNumber("10") }
    );
    assertEquals(7.320, retFunc, 0.01);
}
```

As can be seen, the AgentSpeak test is substantially more concise. The Java-based test requires 15 lines of code and 521 characters (without spaces). In contrast, the test written in AgentSpeak requires five lines of code and merely 110 characters.

4 CONCLUSION

In this paper, we described the introduction and application of test-driven development capabilities to the JaCaMo framework for developing multi-agent systems and in particular to JaCaMo's AgentSpeak interpreter. Considering that AgentSpeak is not only an agent-oriented, but also a logic programming language, future work could, for example, study the introduction of testing capabilities to other logic programming approaches.

REFERENCES

[1] Cleber Jorge Amaral and Jomi Fred Hübner. 2020. Jacamo-Web is on the Fly: An Interactive Multi-Agent System IDE. In *Engineering Multi-Agent Systems*, Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance (Eds.). Springer International Publishing, Cham, 246–255.

[2] Cleber Jorge Amaral, Timotheus Kampik, and Stephen Cranefield. 2020. A Framework for Collaborative and Interactive Agent-oriented Developer Operations. In *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems* (Auckland, New Zealand) *(AAMAS '20)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 3.

[3] Olivier Boissier, Rafael H Bordini, Jomi Hubner, and Alessandro Ricci. 2020. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo.* MIT Press, Cambridge, Massachusetts, United States.

[4] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology).* John Wiley & Sons, Inc., Hoboken, NJ, USA.

[5] Timotheus Kampik, Cleber Jorge Amaral, and Jomi Fred Hübner. 2022. Developer Operations and Engineering Multi-agent Systems. In *Engineering Multi-Agent Systems*, Natasha Alechina, Matteo Baldoni, and Brian Logan (Eds.). Springer International Publishing, Cham, 175–186.

[6] Timotheus Kampik, Adnane Mansour, Olivier Boissier, Sabrina Kirrane, Julian Padget, Terry R. Payne, Munindar P. Singh, Valentina Tamma, and Antoine Zimmermann. 2022. Governance of Autonomous Agents on the Web: Challenges and Opportunities. *ACM Trans. Internet Technol.* 22, 4, Article 104 (nov 2022), 31 pages. https://doi.org/10.1145/3507910

[7] Viviana Mascardi, Danny Weyns, Alessandro Ricci, Clara Benac Earle, Arthur Casals, Moharram Challenger, Amit Chopra, Andrei Ciortea, Louise A. Dennis, Álvaro Fernández Díaz, Amal El Fallah-Seghrouchni, Angelo Ferrando, Lars-Åke Fredlund, Eleonora Giunchiglia, Zahia Guessoum, Akin Günay, Koen Hindriks, Carlos A. Iglesias, Brian Logan, Timotheus Kampik, Geylani Kardas, Vincent J. Koeman, John Bruntse Larsen, Simon Mayer, Tasio Méndez, Juan Carlos Nieves, Valeria Seidita, Baris Tekin Teze, László Z. Varga, and Michael Winikoff. 2019. Engineering Multi-Agent Systems: State of Affairs and the Road Ahead. *SIGSOFT Softw. Eng. Notes* 44, 1 (March 2019), 18–28. https://doi.org/10.1145/3310013.3322175

[8] Constantin-Valentin Pal, Florin Leon, Marcin Paprzycki, and Maria Ganzha. 2020. A Review of Platforms for the Development of Agent Systems. arXiv:2007.08961 https://arxiv.org/abs/2007.08961

[9] Yoav Shoham. 1993. Agent-oriented programming. *Artificial Intelligence* 60, 1 (1993), 51 – 92. https://doi.org/10.1016/0004-3702(93)90034-9