

Unit Testing in Multi-Agent Systems using Mock Agents and Aspects

Roberta Coelho
Uirá Kulesza
Arndt von Staa
Carlos Lucena

Outline

1. Testing Multi-Agent Systems
2. Our Approach
 - Overview
 - Implementing our approach on top of JADE
3. Conclusions & Next Steps

Testing Multi Agent Systems



- Agent-Oriented methodologies, proposed so far, defines approaches to:
 - analyze,
 - design,
 - and implement MASs.
- However, little attention has been paid to how multi-agent systems can be tested.
 - see:
Cernuzzi, L., Cossentino, M., Zambonelli, F., "*Process Models for Agent-based Development*", Journal of Engineering Applications of Artificial Intelligence, 18(2), 2005.


Related Work



- Only a few of these methodologies define an explicit verification phase.
 - **MaSE** and **MAS-CommonKADs** methodologies propose a verification phase based on model checking;
 - **Desire** methodology proposes a verification phase based on mathematical proofs.
 - **AGILE** defines a testing phase based on JUnit.
 - implement a sequential agent platform, used strictly during tests.
 - **Agile PASSI** proposes a framework to support tests of single agents.
 - poorly documented.

Laboratório de Engenharia de Software

Outline




1. Testing Multi-Agent Systems
2. Our Approach
 - Overview
 - Implementing our approach on top of JADE
3. Conclusions & Next Steps

1/6/2006 © LES/PUC-Rio 5

Laboratório de Engenharia de Software

Unit Test Approach for MAS



- Our testing approach calls attention to:
 - the test of the **smallest building block** of a MAS: **the agent**.
 - **Rather than**, analyzing the system as a whole trying to devise system properties.
- The key hypothesis is:
 - if an agent taken alone is not dependable then the collection of agents will not be dependable too
- Hence the basic idea is:
 - to verify whether each agent in isolation respects its specification;
 - **under normal and abnormal conditions**

1/6/2006 © LES/PUC-Rio 6

Objects interaction x Agents Interaction



- A running MAS is a web of agents that interact asynchronously by sending messages to each other.
- This kind of interaction differs in nature from the direct method call.
- We need to devise specific techniques to test each individual agent in isolation.
- We assume that the code comprising an agent has been (unit and integration) tested
 - Hence we need to test just the agent's interface

1/6/2006

© LES/PUC-Rio

7

Unit Test Approach for MAS



- Since:
 - Nearly no agent is an island;
- Almost all agents access resources and interact with others:
 - to whom they provide services or
 - on whom they rely for services,
- One question arises:

How can we define meaningful tests to verify an agent in isolation?

1/6/2006

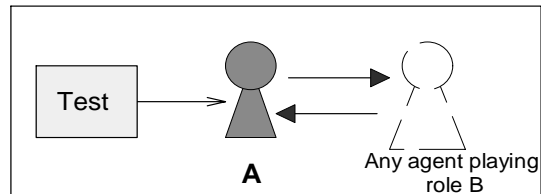
© LES/PUC-Rio

8

Unit Test Approach for MAS



- A test of agent A, which needs a service provided by another agent;



- A valuable strategy is to define a “dummy” version of B (usually called stub).
 - Fake implementations that return canned results.

1/6/2006

© LES/PUC-Rio

9

Mock Object Definition



- Mackinnon et al proposed the Mock Object test design pattern;
- A *Mock Object* is:
 - an object that acts as a stub,
 - but also includes assertions to **instrument** the interactions between the mock and its neighbors.
- Mock Objects have been recognized as a useful approach to the unit test of object-oriented software;



Mackinnon, T., Freeman, S., and Craig, P. "EndoTesting: Unit Testing with Mock Objects". *Proc. of XP2000*, 2000.
Hunt, A.; Thomas, D.; *Pragmatic Unit Test: in Java with JUnit*, Sebastopol, CA: O'Reilly; 2003; Chapter "Mock Objects"; pages 65-78

1/6/2006

© LES/PUC-Rio

10

Mock Agent Definition



- We adapted Mackinnon et al. idea to the MAS context;
- A *Mock Agent* is:
 - an agent that communicates with just one agent: the **Agent Under Test (AUT)**.
 - And has just one plan: to test a specific (or small set) interaction with the AUT.
 - there may be several mocks.
- The *Mock Agent's* plan is equivalent to a test script:
 - it defines the messages that should be sent to the AUT
 - and the messages that should be received from it.

1/6/2006

© LES/PUC-Rio

11

Outline



1. Multi Agent Systems
 - What is a Multi-Agent System?
 - When do we need to use agents?
2. Testing Multi Agent Systems
3. Our Approach
 - Overview
 - Implementing our approach on top of JADE
4. Conclusions & Next Steps

1/6/2006

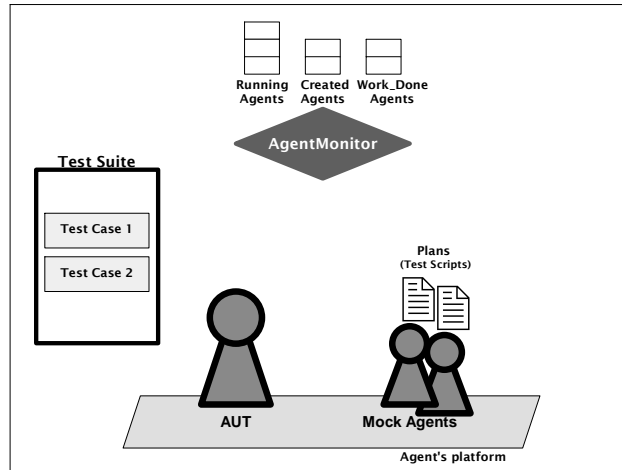
© LES/PUC-Rio

12

Approach's Overview



- Five elements take part in the unit test of an agent:



1/6/2006

© LES/PUC-Rio

13

Approach's Participants



- **Test Case:**
 - defines a scenario – a set of conditions – to which an *Agent Under Test* is exposed;
 - and verifies whether this agent obeys its specification under these conditions.
- **Test Suite:**
 - consists in a set of *Test Cases* and a set of operations performed to prepare the test environment before a *Test Case* starts.

1/6/2006

© LES/PUC-Rio

14

Approach's Participants



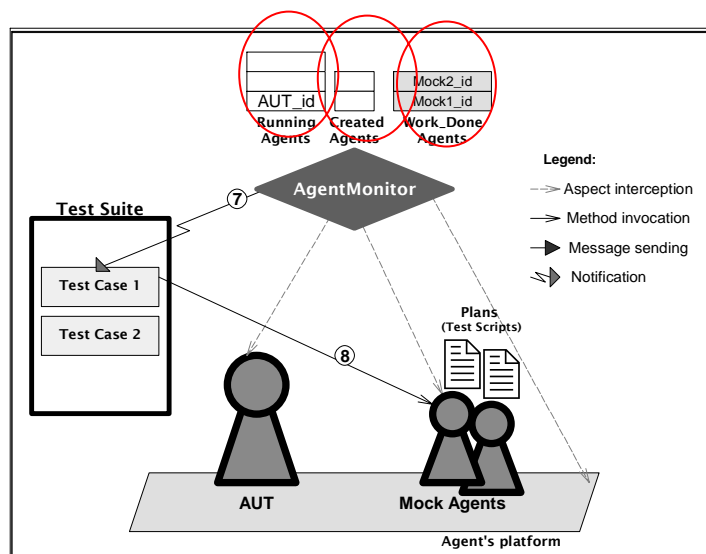
- **Agent Under Test (AUT):**
 - is the agent whose behavior is verified by a *Test Case*.
- **Mock Agent:**
 - consists in a fake implementation of a real agent (class), or part thereof, that would interact with the AUT in the operational MAS.
 - Its purpose is to simulate a real agent (class) strictly for testing the AUT.
- **Agent Monitor:**
 - is responsible for monitoring interaction of the agents in order to inform the *Test-Case* when this interaction finishes.

1/6/2006

© LES/PUC-Rio

15

Unit Tests Common Structure



1/6/2006

© LES/PUC-Rio

16

Unit Tests Common Structure



- According to our approach:
 - The plan of a *Mock Agent* comprises the logic of the test.
 - Each *Test Case* just starts the AUT and the corresponding *Mock Agent(s)*;
 - Than, the *Test Case* waits for a notification from the *Agent Monitor* informing that the interaction between the agents has finished;
 - Finally, *Test Case* asks the *Mock Agent(s)* whether or not the *AUT* acted as expected.

1/6/2006

© LES/PUC-Rio

17

Designing Effective Test Cases



- A very important consideration in program testing is the design of effective test cases;
- Testing:
 - Cannot guarantee the absence of all errors;
 - It just shows the presence of them;
 - Complete testing is impossible for most of the programs
 - certainly is for agents

What subset of all possible test cases has the highest probability of detecting most of the errors?



Myers, G. J. The Art of Software Testing. Wiley, second edition. 2004.

1/6/2006

© LES/PUC-Rio

18

Test Case Design



- The least effective technique of all is: to arbitrarily choose a set of test cases.
 - As current MASs testing approaches usually do.
- We propose an error-guessing technique for Test Case Design.
 - similar to risk based testing
- The basic idea of an error-guessing technique is:
 - to enumerate a list of possible error-prone situations;
 - And then write test cases based on this list.



Myers, G. J. The Art of Software Testing. Wiley, second edition. 2004.

1/6/2006

© LES/PUC-Rio

19

Test Suite Design Technique



1. For each agent to be tested
 - List the set of roles that it plays.
2. For each role played by the AUT
 - List the set of roles of other agents that interacts with it.
3. For each interacting role:
 - Implement in a *Mock Agent* a "plan" that codifies a successful scenario.
 - List possible exceptional scenarios in which the *Mock Agent* can take part.
 - Implement in the *Mock Agent* an extra plan that codifies each exceptional scenario.

1/6/2006

© LES/PUC-Rio

20

Test Suite Design

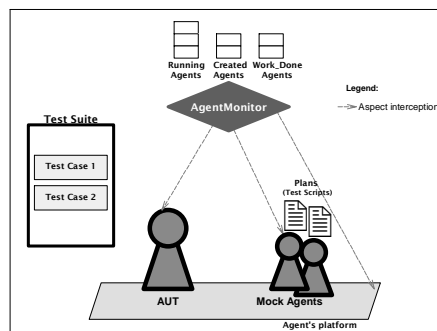


- In order to define unit tests according to this technique, useful sources of information are:
 - sequence diagrams;
 - and the specification of protocols that regulate the interaction between MAS roles.
- Each *Mock Agent* exercises **just one role** of the AUT, rather than the **wide interface** that comprises all the features provided by it:
 - We call this approach “**Role Driven Unit Testing**”.

Agents Monitor Aspect



- To prevent **monitoring concern** from becoming:
 - scattered across multiple platform modules;
 - and tangled with other application concerns.
- The *Agent Monitor* is built upon the facilities of Aspect Oriented Software Development.
 - Represented as an Aspect.
 - That crosscuts platform components to access specific information.



Outline

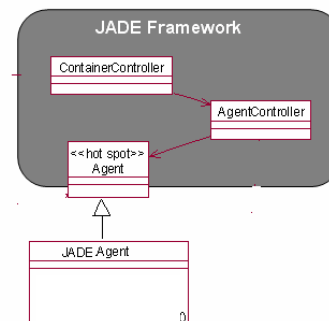


1. Multi Agent Systems
 - What is a Multi-Agent System?
 - When do we need to use agents?
2. Testing Multi Agent Systems
3. Our Approach
 - Overview
 - Implementing our approach on top of JADE
4. Conclusions & Next Steps

Applying our Approach on Top of JADE



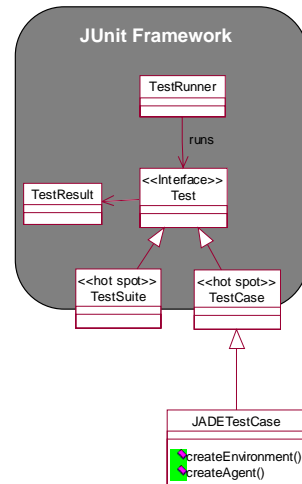
- JADE is an object-oriented framework for developing agent applications;
- An agent in the JADE platform:
 - Extends the base `Agent` class (hot spot);
 - Contains its own thread of execution;
 - Defines a set of *behaviors* (equivalent to a *plan*);



Applying our Approach on Top of JADE



- Instead of creating a unit testing tool from scratch:
 - We decided to extend JUnit framework to support JADE agents' tests.
- The reason for that is:
 - to lower the developers' learning curve providing a simple, and widely used testing framework architecture.
 - possibly used while unit testing the components of the agent.



1/6/2006

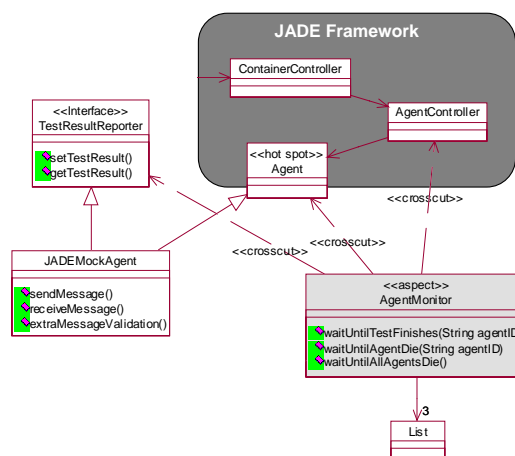
© LES/PUC-Rio

25

JADE Agent Monitor



- The Agent Monitor was developed in AspectJ - an aspect-oriented extension to the Java programming language.



1/6/2006

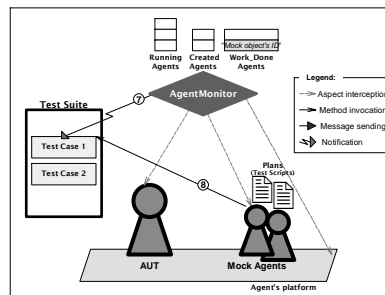
© LES/PUC-Rio

26

JADE Mock Agent



- According to our approach:
 - The plan of a *Mock Agent* comprises the logic of the test.
 - The Mock Agent needs to report the result of a test to the Test Case (step 8)

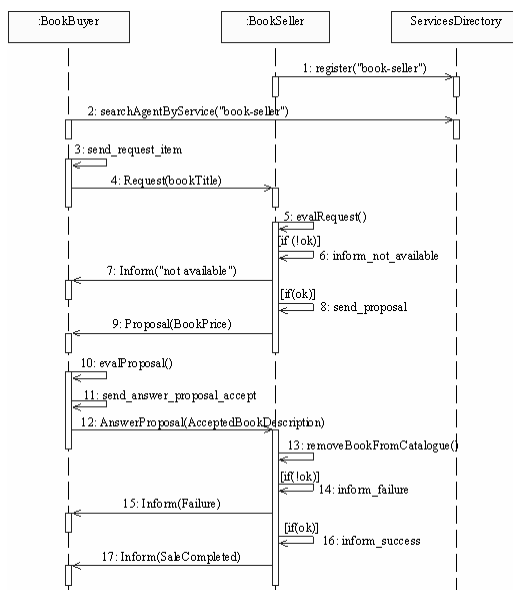


1/6/2006

© LES/PUC-Rio

27

Worked Example



- An Application of book-trading;
- Two roles:
 - BookSeller
 - BookBuyer
- In JADE:
 - BookSellerAgent
 - BookBuyerAgent
- Let`s Test:
 - BookSellerAgent

1/6/2006

© LES/PUC-Rio

28

Worked Example



- Following the unit test case design technique:

Agent	BookSellerAgent
Roles	BookSeller
Interacting Roles	BookBuyer
Successful Scenario	BookSellerAgent sells a book to an agent playing BookBuyer role.
Exceptional Scenario	A BookBuyer agent can send a "cfp" message requesting a specific book, and afterwards send a purchase message trying to buy a different book.

Partial Code of a JADE Mock Agent



```
1. public class MockBookBuyerAgent extends JADEMockAgent {
2.     ...
3.     protected void setup() {
4.         ...
5.         addBehaviour(new TestScenario());
6.     }
7. }
8. private class TestScenario extends OneShotBehaviour {
9.     public void action(){
10.        try {
11.            ...
12.            sendMessage(msgType.CFP,sellerID, bookTitle);
13.            reply = receiveReply(6000, msgType.PROPOSE);
14.            sendMessage(msgType.Accept,sellerID,otherTitle);
15.            reply2 = receiveReply(6000, msgType.FAIL);
16.        } catch (ReplyReceptionFailed e) {
17.            setTestResult( prepareMessageResult(e));
18.        }
19.        setTestResult("OK");
20.    }
```

Partial Code of a JADE Test Case



```
1. public class BookSellerTestCase extends JADETestCase {
2.     ...
3.     public void testBookSelling_Success(){
4.         ...
5.         createAgent("seller", "BookSellerAgent", argS);
6.         createAgent("buyer", "MockBookBuyerAgent", argB);
7.         AgentsManager.waitUntilTestFinishes("buyer");
8.         mockAg=environment.getLocalAgent("buyer");
9.         res=((TestReporter) mockAg).getTestResult();
10.        if(!res.equals("OK")){
11.            fail(res);
12.        }
13.    }
14. }
```

Outline



1. Multi Agent Systems
 - What is a Multi-Agent System?
 - When do we need to use agents?
2. Testing Multi Agent Systems
3. Our Approach
 - Overview
 - Implementing our approach on top of JADE
4. Conclusions & Next Steps

Conclusions & Next Steps



- We presented a unit testing approach for MASs.
- Our approach aims at helping MASs developers in testing each agent individually.
- It relies on the use of *Mock Agents* to guide the design and implementation of agent unit test cases.
- In order to monitor and control the execution of tests cases we used the facilities provided by Aspect OSD.

Conclusions & Next Steps



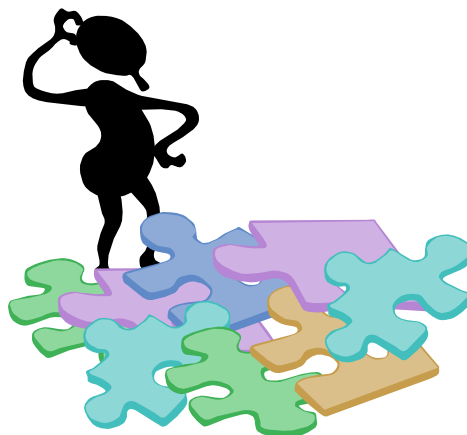
- Our work represents an initial step in the definition of a complete MAS testing process:
 - which will provide strategies to the integration and system testing levels.
- We also intend to address, in future work, the integration of this testing process with existing development methodologies.

Conclusions & Next Steps



- Finally, we are also investigating the complete specification of a generative approach:
 - which can generate from interaction protocols part of the source code of *Mock Agents*, *Test Suites* and *Test Cases*.
- The definition of this generative approach can:
 - improve the productivity of our agent unit testing approach;
 - and motivate even more MAS developers to use it.

Questions?



References



- Avizienis, A; Laprie, J-C; Randell, B; Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1(1), pp. 11-33, 2004.
- Beck, K. Extreme Programming Explained. Reading, MA: Addison-Wesley, 2000
- Bellifemine, F., Poggi, A., Rimassa, G. JADE - A FIPA2000 Compliant Agent Development Environment. In Proc. Agents Fifth International Conference on Autonomous Agents, pp. 216-217, 2001.
- Binder, R. Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman Publishing Co., Inc., 1999
- Caire, G. et al. Multi-agent systems implementation and testing. In Proc. Of 4th International Symposium - From Agent Theory to Agent Implementation (AT2AI-4), 2004.
- Cernuzzi, L., Cossentino, M., Zambonelli, F. Process Models for Agent-based Development, Journal of Engineering Applications of Artificial Intelligence, 18(2), 2005

1/6/2006

© LES/PUC-Rio

37

References



- Czarnecki, K. and Eisenecker, U. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000
- DeLoach, S., Wood, M. and Sparkman, C. Multiagent Systems Engineering. International Journal of Software Engineering and Knowledge Engineering, vol. 11, No. 3, pp. 231-258, 2001.
- Filman, R., Elrad, T., Clarke, S., Aksit, M. Aspect-Oriented Software Development. Addison-Wesley, 2005.
- Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
- Gamma, E. and Beck K. JUnit: A regression testing framework. <http://www.junit.org>, 2000.
- Garcia, A., Lucena, C., Cowan D. Agents in Object-Oriented Software Engineering. Software Practice & Experience, Elsevier, 34 (5), pp. 489-521, 2004.
- Iglesias, C. et al. Analysis and Design of Multiagent Systems using MAS-CommonKADS. Springer, LNCS 1365, pp. 312-328, 1997.

1/6/2006

© LES/PUC-Rio

38

References



- Jonker, C.M., and Treur, J. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. Proc. of COMPOS'97, Springer, LNCS 1536, 1998.
- Kiczales, G. et al. Aspect-Oriented Programming. European Conference on Object-Oriented Programming (ECOOP), Springer, LNCS (1241), 1997.
- Kiczales, G. et al. Getting Started with AspectJ. Communication of the ACM, 44(10), pp. 59-65, 2001.
- Knublauch, H. Extreme programming of multi-agent systems. In Proc. 1st International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 704 – 711, 2002.
- Mackinnon, T., Freeman, S., and Craig, P. EndoTesting: Unit Testing with Mock Objects. Proc. of XP2000, 2000.
- McConnell, Code Complete, 2nd Ed., Microsoft Press, 2004.
- Myers, G. J. The Art of Software Testing. Wiley, second edition. 2004.