University of King’s College London

Computer Science Department

FINAL YEAR PROJECT

Testing Multi Agent Based Systems

Seyed Hossein GHOTBI
Student Number: 0745951

Supervisor:
Professor Mark HARMAN

2007/2008
Contents

Acknowledgement VI

VII

Abstract 1

1 Introduction 2

1.1 Overview 2

1.2 Introduction 2

1.3 Aim 3

1.4 Objectives 3

1.5 Simplified Example 3

1.6 Problem Statement 4

1.7 Report Structure 5

2 Review of the literature 7

2.1 What Agents Offer? 7

2.2 Toolkits 8

2.3 Testing Criteria 8

2.4 Advantages of Testing 8

2.4.1 Classical Approach 9
3 Specification and Design ........................................ 10
   3.1 Overview .................................................. 10
   3.2 Chosen Toolkits .......................................... 10
   3.3 Requirements ............................................ 11
      3.3.1 Multi Agent System Requirements ................ 11
      3.3.2 Case Study : Auction ................................ 12
      3.3.3 Application Requirement ......................... 13
   3.4 Design Issues ........................................... 14
      3.4.1 Reasoning Nature of Such Systems .............. 14
   3.5 Jason and Behavioral Impact .......................... 14
      3.5.1 Used Keywords ..................................... 14
      3.5.2 Agent’s behavior .................................. 15
      3.5.3 Application Requirement ......................... 17
      3.5.4 Designed Algorithm (Application) .............. 17
   3.6 Conclusion ................................................ 21
      3.6.1 Abstract View- Designed Algorithm ............ 21
      3.6.2 Application ......................................... 21

4 Implementation .............................................. 23
   4.1 Overview .................................................. 23
   4.2 Testing Method .......................................... 23
      4.2.1 Step 1:Gathering Information .................... 23
      4.2.2 Step 2: Multiple-Multilevel Dependencies .... 24
      4.2.3 Step 3:Analyzing .................................. 28
      4.2.4 Step 4: Detecting Bugs ............................ 30
   4.3 Implementing the testing algorithm ................. 30
      4.3.1 Detecting the auctioneer .......................... 30
      4.3.2 Implemented Part related to Auctioneer And Bidders .. 31
5 Evaluation  

5.1 Overview  .................................................. 38

5.2 Evaluation  .................................................. 38

5.2.1 Part I: Auction Aspects  ................................. 39

5.2.2 Part II: Classical versus applied approach  .......... 39

5.2.3 Mutation and Test sets  ............................... 42

5.3 Testing Algorithm and different MAS cases  ............ 45

5.3.1 Testing Algorithm  ................................... 45

5.4 Conclusion  ................................................ 47

5.4.1 Hard To Kill Mutants  ................................ 47

6 Conclusion  .................................................... 49

6.1 Summary  .................................................. 49

6.2 Conclusions Based On The Project  ....................... 49

6.2.1 Toolkits  ............................................... 49

6.2.2 Testing  ................................................ 49

6.3 Contributions  ............................................. 52

6.4 Requirments and ideas for future Work  .................. 52
List of Figures

1.1 Project Gantt Chart ................................................. 5

3.1 The Bidders Testing Flow chart ................................. 19
3.2 The auctioneer testing flow chart ............................... 20

4.1 Testing algorithm steps ........................................... 24
4.2 Gathered Information and relation with code dependency .... 26
4.3 Auctioneer Dependencies ......................................... 27
4.4 Bidderps Dependencies ........................................... 28

5.1 Application output for analysing the defualt auction aspects ...... 40

6.1 Covered Area reallalated to types of agent speak terms .......... 50
List of Tables

4.1 Obtained Data related to the Auctioneer 1/3 .......................... 31
4.2 Obtained Data related to the Auctioneer 2/3 .......................... 32
4.3 Obtained Data related to the Auctioneer 3/3 .......................... 33
4.4 Obtained Data related to Bidders 1/4 .................................... 34
4.5 Obtained Data related to Bidders 2/4 .................................... 35
4.6 Obtained Data related to Bidders 3/4 .................................... 36
4.7 Obtained Data related to Bidders 4/4 .................................... 37

5.2 Mutation set number 1 .................................................... 43
5.4 Mutation set number 2 .................................................... 44
Acknowledgment

I want to thank Professor Mark Harman for spending his time for guiding me through this project. I would also thank Professor Michael luck for giving interesting lectures in multi agent systems class, that I become interested in the area.
To my teachers and family
Abstract

In some domains because of various reasons, agents approach could be a suitable replacement of object oriented paradigm. These reasons can be categorized into two different parts theoretical and practical. Although agent paradigm is a young approach but in reality the practical part is following a theoretical part. It might be common and have some advantages in terms of development efficiency of such toolkits. But it does not show an efficient way of developing multi agent systems to the developers. This project focuses on the important aspect of software development including the agent based system development, testing.

Because of lack of previous work and number of different languages and methodologies that support agent based system development; there are too many issues to tackle. And this is not achievable in one project. Therefore in one hand the goal of this project is to focus on some key points related to testing agent based systems and proposing a applicable testing algorithm. And in another to give a solid foundation for future works in this area.

First the testing algorithm will include analyzing the most possible and efficient way to test such systems. Produce a clear and structural design for each step of testing. Implement an application based on those steps. And at the end check the efficiency of testing algorithm by using mutation testing and proposed bug insertion technique.

In this project, a testing approach is proposed. An application based on testing approach developed and tested. The efficiency of approach calculated with a proved alternative mutation testing approach. The challenges for the future work discussed and some alternative possible way explained.
Chapter 1

Introduction

1.1 Overview

This chapter presents the aim, objectives and an introduction of testing agent based systems. It explain the challenges of this project with a simplified example. It gives the problem statement and the report structure at the end.

1.2 Introduction

In recent years, agents have been started to be implemented as a software paradigm [7, 5, 11]. The most important reason for this was the characteristics of the agents which is not available in the object oriented approach. These characteristics are namely autonomy (act on their own), proactivity (following their goals) and reactivity (react to the change of environment) [6]. The other reason is that the softwares, on the other hand, are getting more complex therefore higher abstraction paradigms such as agent approach, would help designers and developers to overcome the complexity problem. There are different toolkits for supporting agent implementation in different domains [7, 10, 12, 8]. These toolkits have both commercial and academic support. But one aspect of software development in agent development process is still missing - Testing. As classical software development, testing will help the developers to develop more efficient systems. The most used approach for agent development is BDI-belief, desire, intention- architecture for each agents in the multi agent systems. Agents will change their view dynamically through the run time of the MAS. Consequently testing agent systems will involve static and dynamic test-
ing. The software which would be developed in the project would be able to test a MAS in both static and dynamic domains. Static approach will involve testing the agents based on their files.

1.3 Aim

The aim of this project is to develop testing techniques for multi agent systems. It should be able to cover testing criteria of multi agent systems by considering it’s characteristics.

The project output is an application that is able to test the multi agent systems and illustrate and prove the results in a convenient way.

Because of lack of work in this field, it should be written and developed in a way that provides a solid foundation for future work, in both theory and practice.

1.4 Objectives

- Understand the concept behind the agent and multi agent systems.
- Based on those concepts select the suitable methodology and toolkit.
- Select an agent based system as a case study that adapt from requirement.
- Provide a testing methodology for testing agent based systems.
- Develop an application based on the methodology. It should be able to give enough information to the user by full explanation about the MAS code and/or if any faults in it. The reason, place and solution for the fault should be given.
- Prove the efficiency of the testing algorithm.
- Provide enough and useful information for future development related to this subject.

1.5 Simplified Example

Here is an example to describe the issues related to testing agent based systems in a basic level. There is a MAS consists of 3 agents. The aim of this MAS is to share information in times of need for each agent. Therefore each agent has 3 goals:
CHAPTER 1. INTRODUCTION

1. Get information if it does not know the answer.

2. Give information if it does know the answer the other agent is asking.

3. If necessary choose between two provided answers.

And simply three plans designed for achieving these two goals. And they are not containing any subgoal or plan. In this example the issues need to be considered for testing mechanism are:

1. Agent characteristics should be tested. As [2], discussed intelligent agents are agents with these characteristics:
   - They are situated in an environment.
   - They are autonomous (independent from the others).
   - They communicate with each other (being social).
   - They are following their goals (being proactive).
   - As discussed in the previous list, they have different plans (different ways) to achieve their goals (being flexible).
   - If failures occur they revocer from them (being robust).

2. The method can understand how each agent achieve it’s goals.

3. Although each agent is autonomous but their actions have impact in other agents by being social. Therefore at the last point of testing the MAS is consider the related plans in same and/or different agents as a set. And test this set in general.

1.6 Problem Statement

Defining and developing a testing algorithm for agent based systems, if necessary. It have to be in such a way that explains the key points in this area. Achieve the task by following a case study based on a chosen language and methodology. Prove the efficiency of the testing algorithm and highlights the challenges in this area for the future work. The figure [1,1] shows the gantt chart related to this project.
1.7 Report Structure

The project contains five major steps:

- **Background Reading** - Background reading includes materials related to the agent based systems and the areas related to testing and multi agent based systems. The output of this section is choosing a suitable toolkit and methodology for the rest of the project. Get an idea if we can use classical approach for testing software, like slicing and apply to agent based systems, and if no what key points have to be in consideration for developing such testing algorithms.

- **Specification and Design** - In this chapter the chosen case study is explained in detail. Defining the issues related to the case study. Proposing the testing algorithm. And explain each substeps in great detail. The reason for each step, flowchart and explanation is given.

- **Implementation and Testing** - This chapter covers issues and the output of the implementation based on previous chapters. The issues mainly related to development of such toolkits for MAS, like defining dependencies, and partially the toolkit itself, such as using useful comments.

- **Evaluation** - This chapter presents the mutation technique of inserting bugs into the agent based system. And to evaluate the efficiency of the proposed testing algorithm. The results will be represented in statistical graphs.

- **Conclusion** - This chapter provides the gathered conclusions and foundings from each part of the project.
CHAPTER 1. INTRODUCTION

- **Future Work** This chapter explains the challenges for future work in this area. It explains some about how the future path could look like in this area of testing.
Chapter 2

Review of the literature

In this chapter the background information related to agents, toolkits for developing MAS and testing criteria are covered.

2.1 What Agents Offer?

Softwares are getting more complex. Higher abstraction will help developers to overcome the complexity and agent as a software by using a BDI approach will fulfill these needs. Agents are autonomous, proactive, reactive [6], [3], [2]. Being autonomous makes agents to be on their own. This characteristic helps agents to be suitable for systems with changeability and modularity issues [6]. For instance, if there is a system with 300 agents, changing one agent will not effect the other parts of the system unless they are interacting with that agent. Proactive means that agents will continue to fulfill their needs, such as getting information for particular purposes. Reactiveness helps agents to react to the changes in their environment. Being proactive and reactive helps the agents to be suitable for decentralized and complex systems. Because they can react to the changes and get their required information even if the information is not broadcasting in a centralized way [6], [3], [2].
2.2 Toolkits

There are different toolkits which are mostly developed in the last 10 years. Most used ones are Jade [8], JACK[7], and more recently Jason [7] and Living Systems [10]. Most of them are for commercial use. Programming languages that they use are mostly an extension of a object oriented language. The reason for that is because the running systems in industry are now mostly written in Boolean languages. And they do not want to implement 100 percent in agent approach, however they need to use agents in some part of their systems. Consequently the coexistence is an important issue for such systems [6].

Agents are used in different application domains, some of them are in logistics [7], mobile industry [5], military [7].

2.3 Testing Criteria

Among all the mentioned toolkits only Living System providing testing library for their developers [3]. Testing of an agent should focus on the agent before and after running of the multi agent system. The reason for that is some problems are hidden from the compiler and therefore the tester should check the syntax of the code related to the agents. If there is a change in environment or agents in the environment during the runtime, runtime testing should be available. It is because agents will constantly changing their view, by being reactivity and proactivity. Here is the list of essential issues related to bugs and the tester:

- They are making problems for agents and MAS in general.
- They can not be discovered by the compiler.
- The testing case and solution is suitable for that matter.

They will be in more details in section 3.3

2.4 Advantages of Testing


2. Introducing new testing technique which may be applicable in other computer science categories, namely behavioral testing.
2.4.1 Classical Approach

It might be argued that why the other classical testing is not appropriate. The reason goes back to the point of differences between agent code and the most applicable domain for classical testing techniques-object-oriented code. The agent code (BDI model) represents set of characteristics and these characteristics form the behavior of the agent. In boolean, on the other hand, the code forms an object and then the object forms the characteristics and further more the behavior.

From abstraction point of view Agent code have higher abstract view compared to the objects. And the characteristic nature of the agent code is more close to the Boolean value compared to OO. Because either the agent has that set of behavior or do not. And what agent "do" is based on its set of characteristics. So testing those characteristics is similar to test the agent itself. On the other hand, well-known testing techniques are code based. Maybe in the future the behavioral testing would be implemented for the object paradigm.

The behavioral testing gives higher level and easier experience to the user who uses the application. But in terms of programming that application should use the keyword of the language (in this case Jason), pre-defined expected set of behaviors and the ability to make clear what each part of the agent is represented for. The ability of detection is based on finding path between sub elements of an agent and other agents. And what are the unnecessary and/or corrupted behaviors.
Chapter 3

Specification and Design

3.1 Overview

This chapter covers the reasons behind the toolkit, case study, and testing requirement that were chosen for this project. The criteria and requirement for the MAS and the application that test the MAS are also discussed. The reason of using the case study and the testing algorithm which was designed based on the requirement and design decisions are highlighted. And at the end based on foundings, it provides the flowchart for testing two sections of the case study.

3.2 Chosen Toolkits

The chosen toolkit for creating MAS is Jason [9]. Firstly because Jason is the only agent based open source, academic toolkit. But more importantly it uses Agent Speak which is a language for implementing agents. Agent Speak is not an extension of OO language such as Jack[7], Jade[8] or Living Systems [10]. Although Jason uses an editor that is written in Java. It also uses extension of Java just for GUI purposes [3]. The toolkit used for creating the agent tester is Visual Studio .Net (C#). The used platform is Windows.
3.3 Requirements

By having an abstract view, there are two aspects that need to fit together to make testing multi agent system possible.

- An application to test and analyze the MAS. more details in section 3.3.3.
- The MAS that consists of different types of agents. Each of these agents has a unique set of behaviors or partially similar compared to the other agents. Look at section 3.3.2 for more details.

3.3.1 Multi Agent System Requirements

The agents in MAS should have specific form through out the project. There will be different forms based on the phase of the project. First, the agents in MAS should not have any syntax error that could be detected by the compiler. The focus of this project is to find out the logical errors that can cause conceptual problems. The reason for that is because:

- **MAS Efficiency:**
  The logical errors are the hidden problems. Any type of programmers with various levels of skills can create these problems without being aware of them.

- **Constructive Structure:**
  Each phase from design until the evaluation will follow spiral development.

- **Approach Efficiency:**
  - More efficient approach by having Boolean test cases and results.
  - Gives a clear idea of how much percentage the test cases can cover. The reason for that is, because each language has specific number of syntaxes. And each agent will have specific number of behaviors.

- **Future Work:** Based on section 4, it makes it clear resource and guideline for the future work.
3.3.2 Case Study: Auction

The focus of the project is to find solutions to test multi agent systems. Due to the magnitude of this area and the lack of previous work, structural testing approach is needed. The structural approach is achievable by choosing a case study and the testing will be applied on that. The case study gives syntactical and consequently behavioral boundaries to the MAS. Therefore, the testing will focus on some aspects of MAS, and will be a useful background information for achieving a generic solution.

The MAS, itself should be suitable for testing. Here are the characteristics of a MAS that was considered for testing:

- Each agent in the MAS has a variety of behaviors. These behaviors can be unique or if in concept they are the same, they have different behavioral reactions.
- They should be able to communicate and respond to each other.
- They should consist of essential elements of agents, such as initial belief, initial goal, triggering events, contexts, etc. The description of all of these elements is covered more in the section [3.5].

Abstract View

In Auction there are 4 agents, one auctioneer and three types of bidders.

- Auctioneer announces the auction. There are a number of auction rounds. It announce the auction, receive the bids, picks and announce the winner.
- Type one bidder. Reply to the auctioneer by constantly bidding through all the auctions, with the same bid value.
- Type two and three bidders. They reply to the auctioneer, and help each other in a way that one will bet less and the other will bet with the help of the agent type two (Using alliance protocol).

Behavioral Testing is the core element of testing. The agent tester should go through the agent codes, and check the codes. These are the main steps:

1. Scan the code.
2. Check it against the list of expected behaviors. In this step the testing should not just check the code "phrases". It should read the syntax and "think" what does it mean. For example if an agent has a variable that is a number, first the tester should know which type of agent it is. It might be the auctioneer and that number is the number of rounds. Or it might be a bidder and the number shows the bid of that agent. In fact this is the main stage.

3. If there is a problem, points it out and desirably give the solution to it.

Here are the main steps of the testing the case study:

1. Identifying which agent is auctioneer and which one is bidder.

2. auctioneer
   (a) How many rounds the auctioneer announce the auction.
   (b) How it picks the winner for each round.

3. bidders
   (a) What type of bidder it is?.
   (b) What is/are the bid(s) proposed from this auction?.
   (c) What is the strategy of each agent?.
   (d) Does it make an alliance with the other agent and what is the reaction of this agent and that helper agent?.

4. Other issues related to the agents, such as if they response to the upcoming messages the way it should be.

3.3.3 Application Requirement

General Issues
Ability of connecting elements of each other with its own sub elements and other agents elements and sub elements. This is very useful because MAS is a society. In society the actions and reactions of agents with each other is very important aspect. The individuality is as important as social behavior of the agent. And also this area is the area that identifies the behavior of the agent as well. And if there is a number of logical errors this gray area should be in more focus compared to the rest.
3.4 Design Issues

Here the design issues related to the MAS and Jason as a chosen toolkit.

There are theoretically and programmatically different aspects of agents that effect the direction of testing of such systems.

3.4.1 Reasoning Nature of Such Systems

There is a close relation between architecture and the agent code. The architecture handle some aspects of the MAS, Jason is based on belief, desire, intention (BDI) architecture. Therefore as [9] mentioned these types of systems are not based on computing and then terminating. Instead continually responds to the events by executing plans to achieve their goals and have an impact on their MAS environment. This reasoning cycle is achieved by the agent architecture and it’s the nature of agents created in Jason. This cycling is not approachable by the outside applications. Therefore for testing the agents the focus is on the code and not the use of the reasoning cycles.

First the project starts with a case study. This MAS is a provided example from the official Jason website[7]. Before designing of the application there is a need to understand the facts about keywords in Jason and their impact on the behavior of the system.

3.5 Jason and Behavioral Impact

The testing is based on the Jason code. This section explains the different aspects of the agent code. The code syntax and explanation of each part of the code such as rule or plan is based on provided explanation [3].

3.5.1 Used Keywords

The only pre-defined section in the program are the keywords (such as .send) and the notation(such as +,-). The rest is based on conclusion and relation between patterns. Here are the parts each agent should have. This information is required for syntax detection of the application.
3.5.2 Agent’s behavior

The other important fact is also the relation between each part of the agent and the behavior of the agent.

These two essential factors are the starting point of implementation of testing of agent-based systems. The following section is the core summary of each part of the agent, their behavioral impact on the agents and MAS in general. And the considering "notes" for the extra functionality of the agent tester application.

Relation between Beliefs and Behavior

Beliefs are represented as predicates. They are acting as an information source for an agent and as [2] mentioned they express particular property such as name (x).

Share Facts: It can show a fact that is based on relation between individuals.
Each individual represents and refer to other part of the agent or any other agents in MAS. Ex: error (bug, computer), can be represented as: bug creates an error in a computer. They can also be used as a negation (which is not covered in this testing approach).

Behavioral Impact: Because they represent the information that a plan might need. There is a close relation between agent’s belief and the plans it executes. Although not all the plans need the information from the belief, they can also have another source to make the plan executable, such as another source of information inside other agents. These are part of dependency issues that explained more in section 4.2.2.

Note: The expression are not absolute true; it’s the current state of belief, and/or the belief is wrong anyway. Take to consider for application: contradiction in belief between different agents. For example if an agent has the belief that sky(blue). In this case the current agent belief is that the sky is blue. It might be a case that the sky is/is not blue, as soon as agent knows that the sky is not blue it will update its state of belief.

Rules

To define conditions for the plans and they are located in the belief base of the agent. Abstract simple form look like this, Conclusion: - if true.
CHAPTER 3. SPECIFICATION AND DESIGN

Behavioral Impact
The rule conclusion is based on the thing that agent already knows. And rules are
going use for certain conditions in plans therefore although it is not directly change
the behavior but because of plan and knowledge dependency, it act as an order for
certain action which executed from the plan of the agent.

Goals

There is declarative use of goal in Jason

- Initial Goal
  - Start with !: Is goal to be achieved, it does change the behavior of the
    agent.
  - Start with ?: Test goals: to retrieve the information from agent’s
    belief base. Its just information gathering and has no direct impact to
    the behavioral direction of the agent.

Plan

Without plan(s) the agent cannot achieve anything. The plan shows how and why
the agent achieves the state of the goals it wanted to. It has three parts as follows:

Triggering Event

- Plan Activation
  - Start with +: When you acquire that belief then the plan becomes
    active. One goal can be achieved by different plans and consequently trig-
    gering events. The context is used -second part of the plan- to decide
    which plan should be executed

Context

Determines the success of a plan compared to the alternative ones. Logically speak-
ing it contains expressions. There are 4 types of literals. The agent believe in is
true, the agent believes m is false, the agent does not believe l is true, the agent does not believe l is false.

If the plan has no context it means there is no condition that plan will not execute.

Body

- Actions

  - Internal Actions
    * Starting with the . and not change the environment such as .print and .send
    * Additional goals inside the body. These add the sub goals for that plan.

- Achievement Goals (! and !!) The agent should wait till achieved that subgoal and then continue. the !! Shows as soon as the agent has that goal it will move forward. It does not wait till the time to achieve that sub goals.

- Test Goals(\text{name(x)}); For reminding the value of that belief. Connect it to the belief and the application should take it into the account. Important Fact: If it fails the whole plan fails

- Mental Notes -+If exists remove the former one

- Expressions Expressions in the plan will be executed if they are true.

- Plan Labels
  Can contain meta level information, not covered. Used to refer to that plan.

3.5.3 Application Requirement

3.5.4 Designed Algorithm (Application)

Based on the given MAS location by the user, the program should provide the following main functionalities:

- Ability to gather enough information from the user.
• To provide enough information related to the MAS, in a convenient way (producing reports).

• To give the testing results related to the MAS. In case of bugs give adequate information, by giving name, cause and location of the error.

• To produce an agent manager based on gathered information, for further information.
Figure 3.1: *The Bidders Testing Flow chart*
Figure 3.2: The auctioneer testing flow chart
3.6 Conclusion

Based on design and specification there are some issues related to the application and the testing algorithm.

3.6.1 Abstract View- Designed Algorithm

- Finding the Auctioneer.
- Go through Auctioneer to find number of rounds, pick up technique, and associated issues.
- Go through each agent and find out the number of bids and their strategy.
- Take to the consideration the flowchart from this section as a basic point for developing a testing algorithm to test the MAS.

3.6.2 Application

Based on decisions made the application should include the following aspects :

- First, the software should know the location of agents. In Jason all the agents are initialized in the container therefore by knowing where the container is, the software can find the name of other agents and the location of them, as the agents are in the same folder as the container.
- After getting the name of the agents the software will go through each of them one by one and find the syntaxes which are related to the communication.
- Then it finds out the number of times each agent communicates with other agents.
- It will add up the numbers and show in the graph, the agents and the numbers of times they are called.

In terms of programming the application should use these issues :

- **Data Structure and file handling** This two aspects is the back bone of this program. The reason for that is the program will decide the behavior and different aspects of agents based on their code by using sequence of steps [12].
• **Comments** For every single aspect of the program there is a comment. It makes it more readable and easier for future work.

• **Variables** the very important ones like the belief of agents was used globally. But the variables that are needed for limited time was programmed locally and got nullled after use. It reduced the number of unnecessary code and make the code reusable throughout the project. It also makes efficient consumption of memory.

• **GUI** The GUI of this program was designed for easier user use. Different aspects designed within different forms. It helps to categorize the functionality of the program, for example the different Windows forms for agent and MAS as a whole.
Chapter 4

Implementation

4.1 Overview

This chapter presents two important aspects of this project. Proposed testing approach that is based on the previous findings, and the implemented MAS tester. It describes four different parts of the testing method and related implemented part. For the first step (gathering information), it explains the important aspects of the code and the part to take into consideration. It also shows the implementation based on the design, and programming issues. At the end of this chapter it describes the testing methodologies for testing the application itself.

4.2 Testing Method

Based on previous chapters the testing method consists of four different steps. It uses agent code as a source, for gathering information for checking the dependencies and analyzing the data.

4.2.1 Step 1: Gathering Information

Gathering information is needed, because testing is based on the agent code written in Jason. It helps for detecting the each part of the code such as belief or rules.
The gathering information itself is based on some substeps. These substeps reflect the different part of the code, more details in section code-gathering information and section 3.5.2. It has close relation with the next step (dependency check), illustrated in figure 4.2.

4.2.2 Step 2: Multiple-Multilevel Dependencies

The next step 'dependency between agent code' can not be achieved without gathering the required information. The agent code has three main parts. Initial belief, rule, and plan. In each section there might be a variable that gets the value from some other part(s) of the agent itself or any other agent within MAS. For mapping the values, there are some alphabetical variables to be replaced with the numerical values. There are three types of variable value replacement:

- Some replacement occurs once based on mapping one variable to one numerical value, such as sending bids based on default initial belief.

- Some replacement happens once but based on more than one variable replacement. For example in ag3 in the plan number 3, it gets the bid value based on its own bid (that got from initial belief) and agent2 bid, that got from agent2’s sending information that came originally from its initial belief.

- For multiple number of times, such as initiating the auctions based on initial belief that defines the lower bound and an upper bound that is defined in a context of a plan in auctioneer.
CHAPTER 4. IMPLEMENTATION

**Level One Internal Dependencies (Bidders and Auctioneer)** The first step of gathering the information is by splitting the code into right order to check the corresponding values. After this step the values of first level of internal dependency can be found. For example by splitting the name and variable of the triggering event in one hand and the name and initial value of initial belief in another. If both names matches then the value of the variable will be mapped to initial belief’s numerical value.

Auctioneer has more elements in its first level of internal dependency because:

1. It has rule to check the contexts against it.
2. It has additional goals to add and subtract for each round of the auction.

**Level Two Internal Dependency (Bidders and Auctioneer)** In some cases the value can be mapped and passed to the chain of corresponding variables. For example agent

**Level One External Dependencies (Bidders and Auctioneer)** This part refers to the part of the agent that relates to the call from other agents in the system. Different aspects of the auctioneer and bidders could be highlighted; by defining this level of dependency.

- **Auctioneer**
  1. Plans related to the broadcasting.
  2. Plans related to the calculating the winner.
  3. Plans related to the announcing the winner.

- **Bidders**
  1. Plans related to the placing of the bids.
  2. Plans related to make alliances between agent2 and agent3.
Figure 4.2: Gathered Information and relation with code dependency
Figure 4.3: Auctioneer Dependencies

Level Two External Dependencies (Only Bidders) So far all the dependencies related to the some part of the agent code itself or from other agents code in the environment-level 1 external dependencies-. So normal mapping without calculation would be applicable. But there are some variables that relate to the calculation during the runtime. Some plans contexts in Agent3 are using the second level of external dependencies. Here is the scenario which is possible to happen during auctions. Agent3 knows it is losing for 3 rounds from auctioneer. Based on the conditions during these 3 rounds it fires plan #1 -for first 2 rounds- and plan2 for the third round. These conditions - plan contexts- are not all related directly to the written code. plan2 and plan1 in agent3 have a condition that checks the current winner. Therefore they get the winning data during run time from the auctioneer.
4.2.3 Step3: Analyzing

Each dependency generates a list of results. For having a complete understanding of the MAS. These results should be analyzed. The following shows important issues related to the auction case study and responsible dependencies.

1. Placing Bids
   Using all four kinds of dependencies needed for defining in each particular round the values of the bids related to each bidder. The reason is to find the answer to the question “which plan in each agent in a particular round is responsible for placing a bid?”.

   - **Level 1 internal dependencies**
     Defines the initial values related to the initial belief or rule, used for mapping triggering event variables to the initial belief values.

   - **Level 2 internal dependencies**
     In this level, the plans are executed based on the values of the context.

   - **Level 1 external dependencies**
     Defines which plans are called from each bidder. Match the plan name
to the responsible plan name for calculating the winner. And finally find
the value that sent from a particular bidder to the auctioneer. This value
can be a number like for agent1 therefore no need for mapping. Or in
agent2 and agent3 the sent alphabetical variable should be mapped to
corresponding values, by using level1 internal dependencies.

- **Level 2 internal and external dependencies**
  shows which plan should be run.

2. **Alliance Protocol**
   Alliance behaviour can happen between agent2 and agent3. By looking at the
   plans related in this scenario, like placing bids, there is a mapping in all four
   kinds of dependencies.

- **Level 1 internal dependencies**
  Defines the initial values related to the initial belief or rule

- **Level 1 external dependencies**
  Defines which plans are called from each bidder. Match the plan name
to the responsible plan name for calculating the winner. And finding the
value that sent from a particular bidder to the auctioneer.

- **Level 1 and 2 internal and external dependencies**
  Shows which plan should be run.

- **Triggering Events and Initial Belief** The triggering events can get the
  initial value from the belief.

- **Context Vs the rest of the code** Context of a plan are the conditions so
  that the set of actions in the body of plan will execute. They can depend on
  the initial belief, triggering events, other context related to the same plan.

- **Trigerring events with different source** In this case the plan will be ex-
  ecuted if the other agent require or ask for the information.

- **Communication** Communication is the important aspect because it shows
  what the agents are broadcasting or what sort of information transferred with
  the ability of tracking down the source and destination.

The importance of these dependencies are to firstly know where are potential
bugs in terms of testing and also understanding the real value of a statement,
for example the number of rounds in an auction and the errors that can occur.
4.2.4 Step 4: Detecting Bugs

In this part of the testing, the bugs are collected based on findings of first three steps. These bugs can not be find by the compiler. They are logical bugs that can only be found by the understanding the overall behaviour of the MAS and each individual agent in the system.

4.3 Implementing the testing algorithm

Because of the efficiency of the program, the steps implemented all in once for each section. For example if the program needs to check the contexts related to the plans in auctioneer, it reads the auctioneer file and gathered data related to context, check the dependencies (if available), analyze and detect bugs. This type of implementing is also useful for future work. Because it can get use as off the shelf component.

4.3.1 Detecting the auctioneer

As [2] mentioned one characteristic of intelligent agents is being autonomous. Therefore they are individuals that can not be controlled by an external software entity. Based on this criteria the testing application should go through the agent codes to have a full understanding on different parts of the agents. For that the name and the location of the agents are needed. Therefore the first step is collecting the names and location of agent files.

In Jason the agent names are located in the container (a file with .mas2) extension. The other agent names are mentioned inside the container and be available in the same folder as the container. Therefore the shortest way to get the name and location of agents is when user locate the container. The application will go through it and get the name and location of agents.

Before starting taking to consideration the dependency between each part of an agent within itself and other agents, the program should detect which agent is an auctioneer. Because the auctioneer detection helps to define other aspects of the auction, and conclusions based on those aspects. For example, what plan the bidders should reply to. And then by going through bidders code any information sent to that plan would be considered as submitted bids-more details in Appendix.

The detection is based on one of the auctioneer characteristic—broadcasting. The auctioneer use broadcasting to announce the auctions and the winners. So the
CHAPTER 4. IMPLEMENTATION

broadcasting criteria is checked to detect the auctioneer.

Explanation related to implementation divided into two sections. First the part related to auctioneer and second the part related to bidders.

4.3.2 Implemented Part related to Auctioneer And Bidders

The following tables are showing the obtained data from the auctioneer and bidder files, description and responsible method.

<table>
<thead>
<tr>
<th>Obtained Data</th>
<th>Description</th>
<th>Responsible Method(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of added plans and the number of responsible plans</td>
<td>Calling and adding other plans so that agent should achieve them</td>
<td>auc_t_addition()</td>
</tr>
<tr>
<td>1) List of goals that added and the number of responsible plans</td>
<td>Goals can be added/deleted inside the agent reasoning cycle. Can make logical problems for agent if they did not use in a correct way</td>
<td>auc_goal_addition()</td>
</tr>
<tr>
<td>2) List of goals that deleted and the number of responsible plans</td>
<td></td>
<td>auc_goal_del()</td>
</tr>
<tr>
<td>List of incremented triggering events and the number of responsible plans</td>
<td>Each plan can be incremented inside another plan. The information is useful for finding the number of times a plan can fire</td>
<td>auc_incr_plan()</td>
</tr>
</tbody>
</table>

Table 4.1: Obtained Data related to the Auctioneer 1/3
<table>
<thead>
<tr>
<th>Obtained Data</th>
<th>Description</th>
<th>Responsible Method(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left and ride hand side of a context with signed</td>
<td>Comparing the signed context with the other context in the same plan</td>
<td>auc_cont_cont_checkup()</td>
</tr>
<tr>
<td>(&gt; , &lt; , =) value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List of Triggering event and their initial value</td>
<td>Initial triggering event value regarding to the initial belief</td>
<td>i_plan_1_value()</td>
</tr>
<tr>
<td>List of variable and their integer interval value</td>
<td>From the signed contexts and regarding to initial belief finding the interval value of used variable inside the context</td>
<td>auc_detect_interval_variable()</td>
</tr>
<tr>
<td>1) Triggering Event</td>
<td>They used for finding dependencies and initial values for their annotations by mapping them to initial belief and other plans of the auctioneer or other agents in the MAS</td>
<td>auc_t_e_a()</td>
</tr>
<tr>
<td>2) Triggering Event Annotations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Obtained Data related to the Auctioneer 2/3
<table>
<thead>
<tr>
<th>Obtained Data</th>
<th>Description</th>
<th>Responsible Method(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of 12 boolean checks</td>
<td>The syntaxes that define each line of agent speak code, and the program understand what every line suppose to do based on the values gathered from these checks</td>
<td>line_check_up()</td>
</tr>
<tr>
<td>Initial Belief and its values</td>
<td>Dependency check against triggering event, contexts, and reminders in auctioneer @MISC parunak, author = H. Van and Dyke Parunak, title = Practical and Industrial Applications of Agent-Based Systems Practical and Industrial Applications of Agent-Based Systems, year =</td>
<td>auc_initial_b_val()</td>
</tr>
<tr>
<td>Left hand side, and right hand side( the keyword, plan, and occurrence) of the rule.</td>
<td>For context substitution</td>
<td>Auctioneer_Rule_Checkup()</td>
</tr>
<tr>
<td>All auctioneer contexts and the number of plan they belong to</td>
<td>Picking all the contexts, for further mapping and analyzing against the other members of the agent</td>
<td>auc_contexts()</td>
</tr>
<tr>
<td>List of dependent plans that has signed contexts</td>
<td>Checking if the plan related to the “signed” context dependent or not</td>
<td>signed_context_dependent()</td>
</tr>
<tr>
<td>List of agent names and plans that related to the signed context</td>
<td>Plans that related to the signed context</td>
<td>plan_context_signed()</td>
</tr>
</tbody>
</table>

Table 4.3: Obtained Data related to the Auctioneer 3/3
<table>
<thead>
<tr>
<th><strong>Obtained Data</strong></th>
<th><strong>Description</strong></th>
<th><strong>Responsible Method</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean checks for determining each line of the code</td>
<td>Line check ups for bidders</td>
<td>b_line_checks()</td>
</tr>
<tr>
<td>1) Initial belief name 2) Initial belief value</td>
<td>Getting the name of initial belief and its value for further mapping against other part of the agent.</td>
<td>b_i_belief_values()</td>
</tr>
<tr>
<td>1) Called triggering Event in Auctioneer 2) The passed data</td>
<td>Information send from the bidders</td>
<td>full_b_sent_info()</td>
</tr>
<tr>
<td>List consists of name of the agent, name of the reminder inside the plans and the associated value related to the reminder</td>
<td>Checking the dependency between the reminder and the initial belief an mapping the name of the reminder with the initial belief and store the data</td>
<td>b_rem_names_values()</td>
</tr>
<tr>
<td>List of variables and their mapped value based on agent's initial belief</td>
<td>Reminders against the agent's initial belief</td>
<td>b_r_against_i_b()</td>
</tr>
<tr>
<td>List consists of name of the agents their plans number and associated contexts.</td>
<td>Grouping the contexts base on the logical notations (&amp; and</td>
<td>), storing with their associated agent names and the plan.</td>
</tr>
</tbody>
</table>

Table 4.4: Obtained Data related to Bidders 1/4
<table>
<thead>
<tr>
<th>Obtained Data</th>
<th>Description</th>
<th>Responsible Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) A list consists of agent name, plan number, triggering event name.</td>
<td>The plans which are dependent to other agents in MAS and the values they send in response to the auctioneer</td>
<td>b_reply_info_auc()</td>
</tr>
<tr>
<td>2) A list consists of agent name, plan number and the response variable (integer, expression) back to the auctioneer.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1) Agent name 2) The variable that relates to the initial belief</td>
<td>1) Reminders against the agent's initial belief. The ones that later get use for sending bids to the auctioneer. 2) They are one variable and not several variables with mathematical sign between them 3) It does not contain the send bid from agent1 to the auctioneer because the value is not coming from the agent belief.</td>
<td>b_r_against_i_b()</td>
</tr>
<tr>
<td>3) Bid values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List of agent names and the data they replied to the auctioneer, the format and the type of communication.</td>
<td>Grouping the send data to the auctioneer, for finding if the bids are integer and if they are not later based on these data find the corresponding value</td>
<td>full_b_sent_info()</td>
</tr>
</tbody>
</table>

Table 4.5: Obtained Data related to Bidders 2/4
<table>
<thead>
<tr>
<th><strong>Obtained Data</strong></th>
<th><strong>Description</strong></th>
<th><strong>Responsible Method(s)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Agent name</td>
<td>Some bids are mathematical expressions consisting of mathematical expression. Therefore each variable should mapped its values from the agent or other agents on the MAS. It is important to mark the dependent values, for further search.</td>
<td>For mapped values and mathematical operators in between 1)m_sent_val() 2)m_sent_val_2() 3)m_sent_val_3() 4)m_sent_val_4()</td>
</tr>
<tr>
<td>2) Plan number</td>
<td>Find variable's value that mapped to the other agents</td>
<td>m_ext_var()</td>
</tr>
<tr>
<td>3) Variable name</td>
<td>By using previous fundings, find responsible plans of each agent that send a value to the other part of the agent</td>
<td>mas_send_var()</td>
</tr>
<tr>
<td>4) Mapped value</td>
<td>A list that defines each agent plans dependency and the sorted context expressions</td>
<td>g_con()</td>
</tr>
</tbody>
</table>

Table 4.6: Obtained Data related to Bidders 3/4
<table>
<thead>
<tr>
<th>Obtained Data</th>
<th>Description</th>
<th>Responsible Method(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of agents and their bid values for each round</td>
<td>Based on previous findings find agents bid values for each round and add notes for further analysis</td>
<td>round_info()</td>
</tr>
<tr>
<td>List of values that each three (found based on number of bidders) indicate for each agent and round</td>
<td>After finding agent bid values for each round based on mapped values and context expressions, make a list that contains the values. This ease the further analysis, by grouping them based on the number of bidders in the MAS</td>
<td>b_p_r()</td>
</tr>
</tbody>
</table>
| 1) winners for each round  
2) percentage of winning overall | Based on list of bid values find the winner. Based on number of winning each round, calculate the percentage of winning of an agent by | w_f() |

Table 4.7: Obtained Data related to Bidders 4/4
Chapter 5

Evaluation

5.1 Overview

This chapter presents the evaluation of the developed application that based on the developed testing algorithm. The evaluation consists of two main parts. First the ability of understanding case study without changing the code. This section shows the evaluation of application based on characteristics that make a software an intelligent agent and understanding of patterns in default MAS. The second part is an alternative approach to mutation testing for testing MAS. Theoretical and practical explanation and reasoning for each section is shown by different test sets. And at the end a conclusion that focuses on the efficiency of the testing algorithm and the ability of the application to focus on different patterns of the code and generate the sufficient amount of data to the tester.

5.2 Evaluation

Evaluation is based on three different issues related to the MAS. The first section of the evaluation should focus on the original case study. This shows the efficiency of the program to deal with default issues. And the focus on the characteristic of agents. Because based on these characteristic a piece of software can be consider as an agent. After the first step an alternative mutation approach is used to evaluate the efficiency of proposed testing algorithm and the application that build upon on.
5.2.1 Part I: Auction Aspects

The auction is based on the English auction therefore the higher bids in each round wins. and here are the possibilities based on default data[??].

- **Number of rounds**: 6 **Winners**: Agent1 wins for the first 3 rounds. Agent3 wins the last 3 rounds because of alliance protocol between agent2 and agent3.

As [6.1] shows the application can predict the right results, without running the MAS inside Jason. Therefore the testing algorithm and the application can understand the following abstract list of functionalities.

- agent names and MAS location
- Auction Protocol For initiating the auction and reply of a bid from each agent, for each round.
- Alliance Protocol Between agent 2 and agent3. This helps the agent3 to win.
- Bidding values from each agent
- Strategy of each agent Consists of different plans for achieving a goal. Placing a bid for agent1, agent2 and agent3. And using alliance protocol between agent2 and agent3.

This shows the application has the understanding of the case study. And also the efficiency of the testing algorithm to test the default auction MAS.

Alternative approach of mutation testing used for evaluation of the testing algorithm. The reason is based on differences is based on the logical representation of agent code in Jason, which explained more in [3.3]. The following section focus on the differences between classical and applied approach.

5.2.2 Part II: Classical versus applied approach

In classical mutation testing approach, the testing algorithm efficiency calculated based on two main parts:
Figure 5.1: Application output for analysing the default auction aspects

- **Part 1: Set of Mutans**
  Each mutant is a set, representing alternative once at a time change of the code.

- **Part 2: Test Set**
  Each test case contains different values. They get use with conjunction with a testing algorithm to kill the mutants. Each test case use the previous test case hard to kill mutants and try to kill them. In the ideal case the chain of test cases will achieve 100 percentage to kill the set of mutants.

- **Finally the efficiency of testing algorithm calculated by using mutation score formula.**

\[
\text{Mutation Score} = \frac{\text{Number of killed mutants}}{\text{Number of mutants}} \tag{5.1}
\]

Applied approach uses the first two parts of classical approach. But there is a difference in hard to kill mutants and efficiency calculation. Because of boolean value of the agent code[1], there is two categories of hard to not logical and logical errors. They further subdivided into hard to kill mutants and killed mutants. And
in each section there is an explanation about why they will or partially will part
of efficiency calculation. **Note:** In applied approach not all the cases are numerical
expressions ex: x+5. Therefore in these type of cases the test case and mutation set
are the same thing.

**Not Logical Errors** In this category although the mutants and test cases change
the scenarios output of MAS. But they do not change the **logical** representation
of any aspect of MAS. Example: changing the bids and auction rounds
boundary value by using different mutant and test sets.

1. **Hard To Kill Mutants** The testing algorithm does not need to take
to the consideration the alternatives that does not make MAS to operate
differently. Therefore these mutants do not necessary relate to the errors.
For example if the default value of the number of rounds is 9. and if
the number of rounds will change to 7 based on a mutant and a test case.
The value of boolean statement is true and it does not change the aspects
of MAS. Having 9 rounds is as much as true as having 7 rounds.
Although they are not errors but they change the output of the MAS,
therefore the application should be able to change its output too. And
consequently part of evaluation.
However there might be a case when the non logical-hard to kill mutants
represents errors. Therefore they should be part of efficiency calculation.
Examples errors such as incorrect interval for number of rounds.

2. **Killed Mutants** Non-logical killed mutants repnets the mutants that
killed by test cases. They are from the second sector of non-logical hard
to kill mutants. Therefore they are part of the evaluation and efficiency
calculation. The number of killed mutants in this category are always
equal or smaller than the number of hard to kill non-logical mutants.

**Logical Errors** This category represents to the mutants that make logical error.
Although each mutant might cause an error in the MAS. But it can not dis-
covered by the Jason compiler. The logical values related to the behaviour of
the case study. For example if a bidder has no plan to reply to the auctioneer.
have a plan to terminate the MAS which is not its responsibility.

1. **Hard To Kill Mutants** Because it is logical errors, all the hard to kill
mutants are bugs. All of them considered to be killed by various test
cases. And the number of hard to kill mutants of this category is a part
of mutation score formula.
2. **Killed Mutants** Logical-killed mutants are the mutants that killed from logical error section. In best ideal case these two numbers are the same.

**Modified Mutation score**

Here is the pre-applied alternative representation of the mutation scope code by taking to the account the discussed matter so far.

If $A = \text{Non-Logical} \ (\text{Number of}(|\text{hard to kill type1} - \text{hard to kill type2}|))$

If $B = \text{Non-Logical} \ (\text{Number of Killed Mutants})$

If $C = \text{Logical} \ (\text{Number of hard to kill mutants})$

If $D = \text{Logical} \ (\text{Number of killed mutants})$

Then:

$$\text{Mutation Score} = \frac{A + C}{B + D} \quad (5.2)$$

But in the other hand because of multiple use of a value, even if there is initially one error in the code, that wrong error might get use multiple times. In this project the auction is based on number of rounds. Therefore the number of rounds that error should get catched in also into consideration. Therefore the following formula is the updated version of mutation score. Used to check the efficiency of the testing algorithm.

If $A = \text{Non-Logical} \ (\text{Number of}(|\text{hard to kill type1} - \text{hard to kill type2}|))$

If $B = \text{Non-Logical} \ (\text{Number of Killed Mutants})$

If $C = \text{Logical} \ (\text{Number of hard to kill mutants})$

If $D = \text{Logical} \ (\text{Number of killed mutants})$

If $E = \text{Number of rounds the mutant killed}$

IF $F = \text{Number of rounds the mutant should be killed then}$

$$\text{Mutation Score} = \frac{A + C + E}{B + D + F} \quad (5.3)$$

**5.2.3 Mutation and Test sets**

Here are the characteristices of agents in the MAS. The case study is based on auction and the important aspects are:

The auctionner should be able to:

- **Announce the auctions** Mutation set 1

Based on pla#1 of the auctioneer, the broadcast message inside the plan. And the initial belief related to the plan. **Original Statements:**
auction(1) + auction(N): True

<table>
<thead>
<tr>
<th>Mutant Number(s)</th>
<th>Mutants</th>
<th>Number of harmful mutants/Number of Killed Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..8</td>
<td>auction(m). m is the set of numbers{-10, -5, -2, -1, 0, 1, 5, 10}</td>
<td>4/4</td>
</tr>
<tr>
<td>9</td>
<td>+auction(N): False</td>
<td>1/1</td>
</tr>
<tr>
<td>10</td>
<td>+auction(N): [source(S)]</td>
<td>1/1</td>
</tr>
<tr>
<td>11</td>
<td>+auction(10): [source(S)]</td>
<td>1/1</td>
</tr>
<tr>
<td>12</td>
<td>broadcast(tell, auction(10))</td>
<td>1/1</td>
</tr>
<tr>
<td>13</td>
<td>broadcast(tell, auction(-10))</td>
<td>1/1</td>
</tr>
</tbody>
</table>

Table 5.2: Mutation set number 1

**Efficient use of receiving bids** is based on two parts in auctioneer: Plan #2 in auctioneer: +plan_bid(N,V)[Source(s)] RRule to calculate the right number of received bids. The calculation located in the right hand side of the rule. .count(place_bid(N,V1),3). That the number of receiving bids should be equal to the number of bidders.

The table 5.2.3 shows the mutations set.
### Table 5.4: Mutation set number 2

<table>
<thead>
<tr>
<th>Mutant Number(s)</th>
<th>Mutants</th>
<th>Number of harmful mutants/Number of Killed Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>place_bid(N,V)</code></td>
<td>1/1</td>
</tr>
<tr>
<td>2</td>
<td><code>place_bid(N,N)[Source(s)]</code></td>
<td>1/1</td>
</tr>
<tr>
<td>3</td>
<td><code>.count(place_bid(N,V1),4)</code></td>
<td>1/1</td>
</tr>
<tr>
<td>4</td>
<td><code>.count(place_bid(N,V1),2)</code></td>
<td>1/1</td>
</tr>
<tr>
<td>5</td>
<td><code>.count(place_bid(N,V1),3+1)</code></td>
<td>1/1</td>
</tr>
<tr>
<td>6</td>
<td><code>.count(place_bid(N,V1),3+1-2)</code></td>
<td>1/1</td>
</tr>
</tbody>
</table>

**Calculate and announce the winner** Is based on the right calculation for the winner. Originally the statement was `winner(N, CurWin, CurV1) & V > CurV1`. Here is the set of mutants: `(N, CurWin, CurV1) & V < CurV1, (N, CurWin, CurWin) & V > CurV1, (N-1+1, CurWin, CurV1) & V > CurV1, (N, CurWin, CurV1) & V = CurV1, send(tell, winner(w)), broadcast(tell, winner(10)), broadcast(tell, winner(I)), broadcast(tell, winner(N)),

**Efficient use of number of rounds** In auctioneer a plan and a rule is responsible for the number of rounds. Auctioneer initial belief and plan are responsible for the number of rounds. Inside the plan(!!checkend) the context and one command in plan’s body can effect the number of rounds. Therefore there are 3 main part to focus on. Here is the the mutation set with number of mutants = Beleif: `auction(m)` when m is a set of numbers (-100, -50, -10, 0, 1, 3, 5, 10) Context: n<1, n>1, n=7, n>8 + auction : ++auction(k), originally the k is N+1 and the mutants related to this section. When K is N+2, N-2, N, N+10 or N+2-2.
Bidders Response to auctionner by placing bids in right order Mutation set in this category consists of In this category the consideration is on the plans that if they execute the agent will send the

Response to each other if necessary This section evaluates the alliance protocol between agent2 and agent3. Corresponding sections are 4 plans in agent3 and 2 plans in agent 4. And also the winning mechanism in aucioneer. But during auctioneer evaluation section the winning mechanism and related aspects were evaluated therefore they are not part of evaluation here. Therefore 6 plans to consider to evaluate in this section. Each plan consists of 3 parts, therefore at least 18 mutants needed.

5.3 Testing Algorithm and different MAS cases

This section focus on the challenges for using the proposed testing method and in different type of MAS. As discussed in section 4.2 the testing method consists of different steps. Here is the list of those steps and similarities and differences that should be take into consideration for future work

5.3.1 Testing Algorithm

Step 1: Gathering Information-similarities The algorithms used in this step remains in same order. The reason is because of BDI(belief, desire, intention) model and agent coding infrastructure. Any type of agents should follow their goals, that could be formulated in 3 different part of the code (initial belief, rules, plans). The plan will further divided into triggering event, context and body. The structure of the coding remains the same and so the issues to tackle in this step as illustrated in figure 4.2.

Step 2 and 3:Dependency check and analyzing Dependency check should be always presents in testing any MAS. Initially, reasons that discussed in section , are remaining the same for any type of MAS. Like in first step because of model and language infrastructure. But levels of dependency are different and specific to the MAS. The issues related to dependency levels are based on number of times the plan is executing its actions and relation between each plan and other plan. The level of dependencies are incrementing if the number of subgoals and plans are incrementing. As [13] discussed it is because of number of ways a plan can achieve a goal and relation between each path to the agent
itself and the MAS it is situated in. The second part that makes difference in each testing is the range of keywords that are used. Because of their impact to the MAS. Therefore the application that is testing the MAS should be able to detect these keywords, and find the dependency that is created by each keyword. For example if an agent is terminating another agent in MAS (by using .kill(agent_name) ) the testing program should detect the keyword and see if this is necessary. If the answer is yes from that point, there is no need to test the terminated agent. And finding its dependencies with the rest of the MAS.

Analyzing the MAS consists of two parts. One is based on the basic characteristics of the agents situated in MAS. For example if the agents can communicate with each other or if an agent can follow some paths to achieve its goals. But the second part is almost unique for testing each MAS. It because of specified characteristic that each agent can hold. One important issue in some MAS could be not important in others. For example in the case study the auctioneer announce the winner, by using broadcasting message. It is important because each agent should know who is the winner and act based on that if necessary. This characteristic might not be important in a MAS that sends information to specified agents. And in fact it should be an error if broadcasting occurs in such systems.

Step 4: Detecting Logical Errors Clearly the output of any testing algorithm is to detect bugs (if any) and give relative results. Relative information is based on the previous steps. And the task is announcing and remain the same for any type of MAS.

Related Issues

- Environment-
  Agents can get use in an unpredictable environments. In this case if an agent’s belief is updating based on the state of the system. The full testing steps are not applicable. In this case the gathering information step should be updated while the system is running. Therefore in this type of situation the testing steps should start by finding the dependencies and leave the gathering information section to agent manager. Agent manager is created by agent tester and fill the gap of the first step. Use the dependencies information from the agent tester and finally can analyze the system.

- Mutation Score Calculation-
  Like section 5.2.2, alternative used mutation score could be used for calculating the testing efficiency. Although some mutations can be killed
without test cases because they do not present a value. For example for checking if a plan context can be true, the 'true' expression should be checked.

5.4 Conclusion

This chapter showed the evaluation of the testing algorithm in a partially different way compare to classical mutation testing. There are different aspects that founded compare to the classical approach:

- **Mutation and test sets** Like classical approach, mutation sets consists of different mutants that each represent one change at the time. The difference between the test sets in classical approach and here is not all the expressions need numerical test sets. Because the expressions do not contain numerical expressions. Therefore these type of mutation sets are checked without test sets.

- **Choosing Mutants** Some mutation sets are based on the characteristic of the system that represented by lines of code. For example to test if the bidders can reply back to the auctioneer, the plans and related part in each bidder should be checked.

- **Hard to kill mutants** There is a set of hard to kill mutants that do not effect the efficiency of the testing algorithm. The reason for that is because the jason is a logical programming language. Therefore some set of mutants should be left in the category of hard to kill. For example the range of the auction rounds. Even if the default value was 6 rounds, if the mutation changes the number of rounds to two it does not effect the behaviour of the MAS and should be left as a hard to kill mutants. And consequently not be counted during the mutation score calculation.

- **Mutation Score** Based on discussed points the alternative mutation score provided to test the efficiency of the testing algorithm. It pointed out that mutation testing score is not just dependent on the mutants but also to the behaviour characteristic of the MAS.

5.4.1 Hard To Kill Mutants

Here is the list of mutants that made them hard to kill:
• **Value Representations** If a part of the agent requires a value, and initially the value is a number. By adding or subtracting that number and inserting them as a mutant, essentially does not get killed as a mutant unless they are changing the behaviour of the agent dramatically or making other elements in that agent change the behaviour or dependency of the agent or other agents in the MAS. For example if agents want to executing a plan in a limited number of times. And created an environment for itself which creates an interval, for example 1 to 10. In this case the agent will executes a plan 9 times. by adding one or subtracting one from lower or upper bound the agent behaviour does not change dramatically unless another part of the agent or some other agents depends on the value of 0 or 10 value. In the case study - Auction the auctioneer announce an auction 7 times. And because there is no dependency for the upper and lower bound, adding and subtracting the value will not kill the mutants. But if the interval corrupted by false interval such as n<1 and n>8 the testing algorithm will detect this bug and mutant will be killed.

• **Higher level dependencies** If the plans and subplans are more than 2 levels, the developed application can not find the dependencies. Although the testing algorithm itself considered these type of situation but because in the case study auction the level of dependencies are not more than 2, the application does not cover the higher level of dependencies.
Chapter 6

Conclusion

6.1 Summary

This chapter explains about the conclusions that made by doing this project. It also presents the contribution points in the area of testing multi agent systems.

6.2 Conclusions Based On The Project

6.2.1 Toolkits

Although agent approach has its own benefits compare to the OO paradigm. The way the toolkits are implemented are denying this approach. Expect Jason (which used in this project) all of them most used ones (Jack, Jade, Living Systems) are using extension of Java. Although it has some benefits such as coexistence issues with current systems. But they do not use the full benefits of agent approach. For this type of toolkits in other hand because they are using an OO code, it is possible to use classical testing approach to test the code, such as slicing or mutation testing.

6.2.2 Testing

For having an efficient testing approach for testing the Agent Speak code there are different points to consider:

- Gathering the data
CHAPTER 6. CONCLUSION

Is the first step. For this part different algorithm developed and proposed to gather:

- **Dependency check**
  Agent speak code has a high level of dependency. Although The agent in the MAS are autonomous.

- **Analysing the data**
  After gathering the data and dependency check, analyzing needed for collecting the errors.

- **Implementation Area** For implementation the testing steps combined for each part of the code. The algorithm and flowchart and explanation of each part was developed and available in section Appendix B.

The covered Area Table [6.2.2] shows the covered types of agentSpeak terms [??].

---

![Diagram](image)

**Figure 6.1:** Covered Area related to types of agent speak terms
CHAPTER 6. CONCLUSION

These are the areas of MAS that are covered

• Initial belief
• Initial goal
• Initial rules
• Plans
• Triggering event
• The one with normal: true and the one with sources
• Context
• Fully covered with the ability of detection of logical and/or between them.
• Internal actions
• Communication Using mental notes

Uncovered Area

• Graphical GUI which sometimes can be represented as the environment of the agent. The reason is the focus of this project is about testing the agent code. The agent code does not have an OO approach. Therefore an entity of this nature (such as graphical GUI) is not covered.

• Not covered—Strong negation and the degree of certainly p36-37.

• Communication
  The communication has six forms in Jason. The only logical problem is when an agent say to the agent to achieve or do something and that will change the state of that agent. In the case study the communicatio type 1 used. That is one agent say to other agent what plan to acheive(if applicable) and pass a value to the target agent.

• Rest of KeywordsThere are 52 keywords that are applicable inside th agent body. In the case study the focus was in the agent plan’s triggering event and context. Inside the body of those plans the additional goal, plan and communication type 1 used. The rest of the keywords can be applied for the future work.
6.3 Contributions

There are various points that achieved through this project, they are:

- Introducing new testing approach for multi agent systems. This can provide more efficient way of developing MAS systems.
- Highlight the negative aspects of using agent approach based on OO approach.
- Defining the testing algorithm, understanding the dependency layers beyond the code. And elements that define the dependency levels. Proposing the testing algorithm for testing multi agent systems and showing the challenges in this area. Showing the efficiency of the testing algorithm by using alternative approach of mutation testing.

6.4 Requirements and ideas for future Work

The future work in this area could include the following aspects:

- This project covered more the belief, triggering event and context. More work should be done for agent actions. Therefore use of more keywords inside the plan of the agents. Adding their behavioural effect to this project testing algorithm.
- In dynamic and unpredictable environments, changing the testing algorithm value gathering.
- Using agent manager inside the manager that can update its beliefs and test the MAS. This approach also could be applied inside the dynamic and unpredictable environments.
- As mentioned in conclusion there might be a solution for code transformation. And using the old testing technique such as slicing. Finding the errors, and transform the errors into agent oriented approach. For example using if statements instead of triggering event and the condition instead of context. After transformation and testing if there is an error will occur inside the condition of the if statement. The tester transform the error to the corresponding element of the agent context.
- **Code generation** can be used to tackle the dynamic environments. The program can gather the data and generate the agent manager. Agnet
CHAPTER 6. CONCLUSION

manager updates itself as other agents. So as updating happens simultaneously it can check the other agents based on those results. Although the response mechanism should be added into each agent in the MAS, so they understand when to respond to the agent manager.
Bibliography

Appendix A

Installing and using the program

- Agent tester application: Open the application folder. Click on install and follow the instructions.
- Using Jason: Open the Jason folder and run the jason.exe.
Appendix B

Programming Code

This section contains the programming code that is written for agent tester. It does not contain the windows Form Designer generated code\textsuperscript{[1]}. The codes represents based on their functionality. For some part because they represent a flowchart, the flowchar is also given.

\begin{quote}
Gathered information
\end{quote}

\textsuperscript{1}For seeing them open the application by Microsoft visual studio and click on files with .designer.cs extension