Improving the LTI-USP Team: A New JaCaMo based MAS for the MAPC 2013

Mariana Ramos Franco, Jaime Simão Sichman

Laboratório de Técnicas Inteligentes (LTI)
Escola Politécnica (EP)
Universidade de São Paulo (USP)
mafranko@usp.br, jaime.sichman@poli.usp.br

Abstract. This paper describes the architecture and core ideas of the multi-agent system created by the LTI-USP team which participated in the 2013 edition of the "Multi-Agent Programming Contest" (MAPC). This is the third year of the "Agents on Mars" scenario, in which the competitors must design a team of agents to find and occupy the best zones of a weighted graph. The team was developed using the *JaCaMo* multi-agent framework and it is an improvement of the system used in the last year contest.

1 Introduction

The "Multi-Agent Programming Contest" (MAPC) is held every year in an attempt to stimulate research in the field of programming Multi-Agent System (MAS) [1] [1]. In the MAPC, two teams of agents are located in the same environment and compete directly in a scenario set by the organizers. By being a direct competition, it is an interesting scenario to evaluate and compare different systems, allowing to identify strengths and weaknesses, promoting the development of all participants.

The LTI-USP, located at the University of São Paulo is one of the most relevant research groups in multi-agent systems in Brazil. The group participated in the 2009 [2], 2010 [3] and 2012 [4] MAPC editions. Since our first participation, the MAPC has been used to evaluate platforms and tools, and to improve our knowledge in developing MAS.

For this year's contest the LTI-USP team was formed by only one M.Sc. student (Mariana Ramos Franco), supervised by Prof. Jaime Simão Sichman; and like last year our main motivation to participate in the Contest was to test and evaluate the *JaCaMo* [5] framework.

JaCaMo[2] is a platform for multi-agent programming which supports all levels of abstractions – agent, environment, and organisation – that are required for developing sophisticated multi-agent systems, by combining three separate

[1] `http://multiagentcontest.org`.
[2] Available at `http://jacamo.sourceforge.net/`.

technologies: *Jason*[3] [6], for programming autonomous agents; *CArtAgO*[4] [7], for programming environment artifacts; and *Moise*[5] [8], for programming multi-agent organisations.

2 System Analysis and Design

The team was developed using the *JaCaMo* multi-agent framework and it is an improvement of the system used in the last year.

In order to cope with the new rules, we changed the team, that is now composed of 28 agents. Moreover, in contrast with our team that participated last year, this new team is more decentralized. One agent, the `coordinator`, is still responsible for determining which are the best zones in the map; however, each agent decides by itself which empty vertex it will occupy in order either to create a zone or to expand it.

In our team, each agent has its own view of the world, and communicates with others for the following purposes: (i) informing the other agents about the structure of the map; (ii) informing about the agent's or the opponent's position, role and status; and (iii) asking for a repair.

The agents' communication occurs via the speech acts provided by *Jason* and, to reduce the communication overhead, agents broadcast to all others only the new percepts, i.e., only percepts received from the contest server which produces an update on the agent's world model are broadcasted. For this reason, there is a strong exchange of information between the agents in the beginning of the match due to the broadcast of new percepts, specially those related to the map, such as vertices and edges. However, the communication overhead decreases as the agents' world model starts to be more complete.

The agent architecture is based on the BDI model [9]. Each agent has its own beliefs, desires, intentions and control thread. The agents are autonomous to decide by themselves the next action to be performed, but in cooperation with each other. The agents have a proactive behaviour, for example, to find the better vertices in the map, and to move to the closest repairer when they are damaged.

At each step, the agent decides which new plan will be executed to achieve a determined goal given only the state of the environment and the results of previous steps. There are no plans that last for more than one step and the plan's priority is determined by the order in which the plans were declared, i.e., the executed plan will be the first one to have its conditions satisfied. Some high priority plans can be considered reactive, such as the one which tells the agent to perform a recharge when running low on energy.

For the development of this project, we choose to not use any multi-agent methodology, because we already had the 2012 team from where we start to

[3] Available at `http://jason.sourceforge.net/`.
[4] Available at `http://cartago.sourceforge.net/`.
[5] Available at `http://moise.sourceforge.net/`.

work, and mainly because we decided that it was better to spend our time to improve the system performance.

To achieve this goal, first we changed the team to 28 agents and the coordination mechanism to form the zones. Next, we added to the agents the possibility to perform some actions from a distance. Then, many strategies were tested, such as: to divide or not the agents in groups to occupy more zones in the map,to buy or not upgrades to the agents, and to use more or less agents to attack the opponents. Finally, based on the results of our tests, we selected the best team to use in the Contest.

Approximately 150 man-hours were invested in the team development and before the tournament we participated in some test matches set by the organizers to ensure the stability of our team. Only during the competition we discussed the design and strategies with the other teams, without performing any change to our team from a match to another.

3 Software Architecture

The architecture of the LTI-USP team, showed in Figure 1, is the same used in 2012 [4]. In this architecture, the agents are developed using the *Jason* MAS platform, which is a *Java*-based interpreter for an extended version of the *AgentSpeak* programming language for BDI agents. Each agent is composed of plans, a belief base and its own world model. The plans are specified in *AgentSpeak* and the agent decides which plan will be executed according to its beliefs and the local view of the world.

The world model consists of a graph developed in *Java*, using simple data structures and classes. It captures every detail received from the MASSim contest server, such as: explored vertices and edges, opponents' position, disabled teammates, etc. At each step, the agent's world model is updated with the percepts received from the MASSim server, and with the information received from the other agents.

Some of the percepts received from the MASSim server are also stored in the agent's belief base, such as the agent's role, energy, position and team's money, thus allowing the agent to have a direct access to these information without have to access its world model. Percepts about vertices, edges and other agents were not stored in the belief base so as to not compromise the agent's performance, as it could be very expensive to update and to access the belief base with so much information. Moreover, since we wanted to update a belief when a new instance was inserted (instead of adding a second one), we decided to use an indexed belief base in which some beliefs are unique and indexed for faster access.

Agents communicate with the MASSim server through the EISMASSim environment-interface included in the contest software-package. EISMASSim is based on EIS[6] [10], which is a proposed standard for agent-environment interaction. It automatically establishes and maintains authenticated connections to

[6] Available at `http://sourceforge.net/projects/apleis/`.

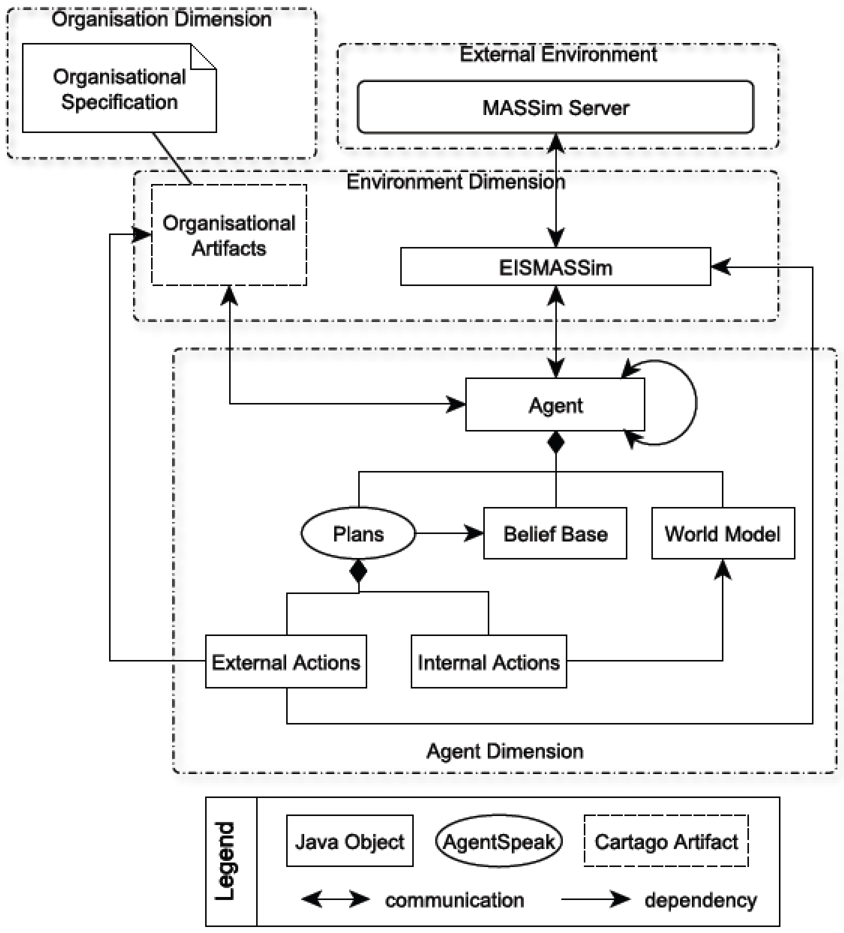

Fig. 1. LTI-USP Team Architecture [4].

the server and abstracts the communication between the MASSim server and the agents to simple Java-method-calls and call-backs. In order to use this interface, we extended the *JaCaMo* default agent architecture to perceive and to act not only on the *CArtAgO* artifacts, but also on the EIS environment as well.

CArtAgO is a framework for environment programming based on the A&A meta-model [11]. In *CArtAgO*, the environment can be designed as a dynamic set of computational entities called artifacts, organized into workspaces, possibly distributed among various nodes of a network [5]. Each artifact represents a resource or a tool that agents can instantiate, share, use, and perceive at runtime. For this project, we did not create any new artifact; we only made use of the organisational artifacts provided in *Moise*.

Moise[8,12] is an organisational model for MAS based on three complementary dimensions: *structural*, *functional* and *normative*. The model enables a MAS designer to explicitly specify its organisational constraints, and it can be also used by the agents to reason about their organisation. We used the *Moise* model to define the agent's roles, groups and missions.

The code of our team can be found in the MAPC website [7], and consists of approximately 2000 lines of code in Java and 1800 lines in *AgentSpeak*, and the development was all carried on using the Eclipse IDE with the *Jason* plugin. The main developer was already familiar with both the development and the runtime platforms, i.e. the Eclipse IDE and the *JaCaMo* framework.

The agents were not distributed across several machines due to time constraints, but is our intention to work in the future on a distributed team, since this is supported by *JaCaMo*.

4 Strategies, Details and Statistics

4.1 Team strategies

For this year's contest, we changed substantially the team organisation by adding different roles to the agents, as shown in Figure 2.

We kept the strategy of distributing the agents in three subgroups (`best_zone`, `second_best_zone` and `attack`), two of them in charge of occupying the best zones in the map, and the other one in charge of attacking the opponents. However, regarding the agents' roles, we decided not to map the five types specified in the scenario (explorer, inspector, repairer, saboteur and sentinel) directly to the roles in our team. Instead, we defined additional different roles in our system according to the adopted strategy. Each of these roles has a mission associated to it, and can be played by one or more type of agents. For example, the `map_explorer` role can be played only by the explorer type, while the `soldier` role can be played by all types of agents. Below we describe the missions related to each role:

- **map_explorer** (explorer): Explores the whole graph by probing every vertex and surveying all edges on its path;
- **map_explorer_helper** (explorer): Helps the `map_explorer` to explore the graph, but only in the first 250 steps. After that, the agent leaves this role to adopt the `soldier` role in the `best_zone` subgroup;
- **soldier** (all types): Tries to occupy the best (or second best) zone in the graph indicated by the `coordinator` agent. When all the vertices of the best zone are occupied the `soldier` starts to look to the neighbour vertices of the team's zone in which he can move to increase its size;
- **guardian** (saboteur): Defends the best (or second best) zone by attacking any opponent that is close to the team's zone, or trying to invade it;

[7] Available at http://multiagentcontest.org/downloads/Multi-Agent-Programming-Contest-2013/Sources/LTI-USP/.

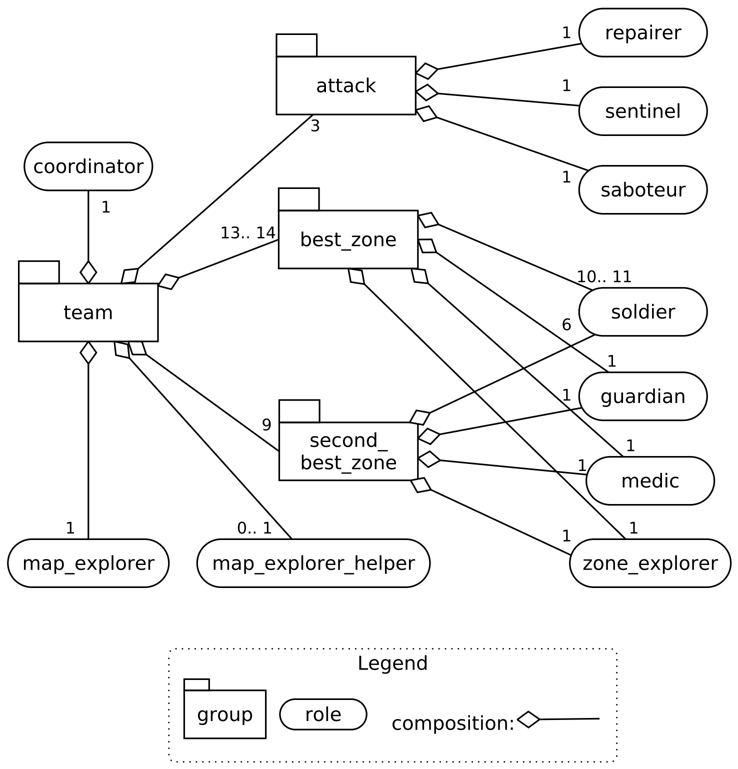

Fig. 2. *Moise structural specification* of the LTI-USP team.

- **medic** (repairer): Occupies the center of the best (or second best) zone and is responsible for repairing the agents in the group, or other agents which eventually need to be repaired, such as the `map_explorer`. In our team, the damaged agents move to the repairers to be repaired;
- **zone_explorer** (explorer): Explores the team's zone by probing the vertices which value is unknown. When all vertices are probed, the `zone_explorer` helps the `soldiers` to increase the zone size;
- **saboteur** (saboteur): Attacks any close opponent, or the opponent who occupies a good vertex;
- **sentinel** (sentinel): Tries to sabotage the opponent by moving inside its zone;
- **repairer** (repairer): Follows the `saboteur`, but always staying two vertices away from it, in order to be prepared to repair the `saboteur` when necessary, but without taking too much risk;
- **coordinator** (none): Agent internal to our system which does not communicate with the MASSim server. It builds its local view of the world through the percepts broadcasted by the other agents. Whenever the world model is

updated, it computes which are the two best zones in the graph and send this information to the other agents. The `coordinator` is also responsible for creating the organisational artifacts, in the beginning of the simulation, and for distributing the groups, roles and missions among the other agents, in order to eliminate the performance issues caused by two or more agents trying to adopt the same role in a group, or trying to commit to the same mission.

The best zone in the map is obtained by calculating for each vertex the sum of its value with the value of all its direct and second degree neighbours. The vertex with the greatest sum of values is the center of the best zone. Zones with the sum of values below 10 are not considered in the calculation. The same computation is performed again to determine if there is a second best zone, but this time removing the vertices belonging to the first best zone from the analysis.

If two best zones are found, the `coordinator` agent will designate the first best zone for the `best_zone` subgroup, and the other for the `second_best_zone` subgroup. Otherwise, the same zone will be assigned for the two groups.

At each step, the team's score is computed by summing up the values of the zones and the current money. Thus the money obtained by the team through the achievement points has a big impact on its score. For this reason, we decided to limit the buy action, allowing only the agents of type saboteur and repairer to purchase an unique extension pack of `sensors`, in order to enable them to attack or repair agents in neighbour vertices.

Figure 3, taken from the beginning of the first match against the TUB team, shows the described strategies in action. It is possible to notice that the LTI-USP team (in blue) occupies two different zones in the map, while in the right bottom the `saboteur` (followed by the `repairer`) is going to attack the opponent. The `map_explorer` and `sentinel` are in the center of the map, and in the right top the `soldier` attacks an opponent in the neighbour vertex.

4.2 Results

All other teams (AiWYX, GOAL-DTU, TUB and UFSC) participated in the previous Contest as well, and their improvement was noteworthy. The LTI-USP team finished the competition in the third place, behind UFSC and GOAL-DTU, which had very strong teams.

The UFSC team won all matches against all teams thanks to their strategy of creating many small zones distributed in the map, instead of only one or two big zones. We also lost all matches against GOAL-DTU which had a very aggressive team, in which all saboteurs and repairers attacked the opponent, while the other agents were creating good scoring zones.

In the first day of the Contest, we won all the three matches against the TUB team. During these matches, our team was able to defend itself very well from the attacks of the other team, keeping a stable zone score, while the `sentinel` agent sabotages the opponent's zone, as shown in Figure 4.

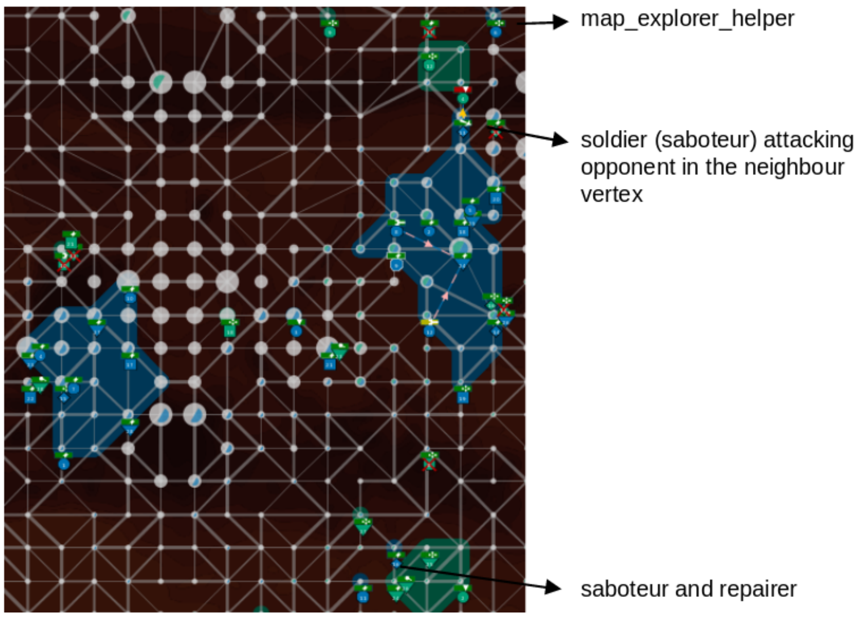
map_explorer_helper
soldier (saboteur) attacking opponent in the neighbour vertex
saboteur and repairer

Fig. 3. Strategies in action.

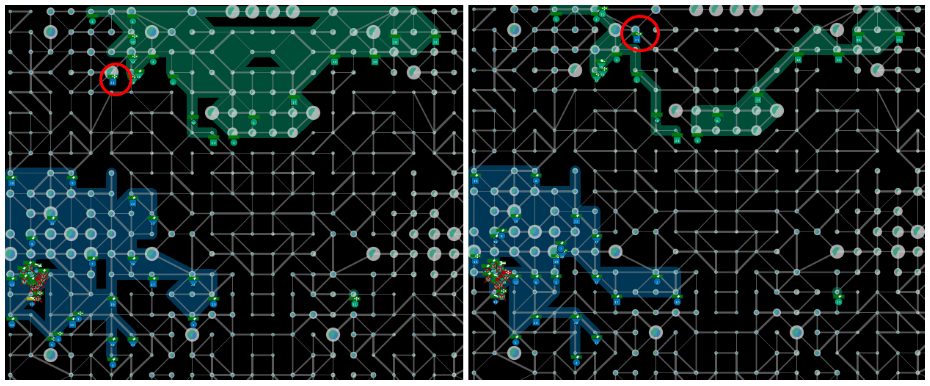

Fig. 4. LTI-USP vs TUB (third match): `Sentinel` invading the TUB's zone.

The AiWYX team came to the last day of the tournament with a very good strategy of finding the map corners to create big zones. Despite this, even with smaller zones, our team won the second match against them (ensuring the third place), thanks to the stability of our team's zone, and because we started to score early (cf. Figure 5).

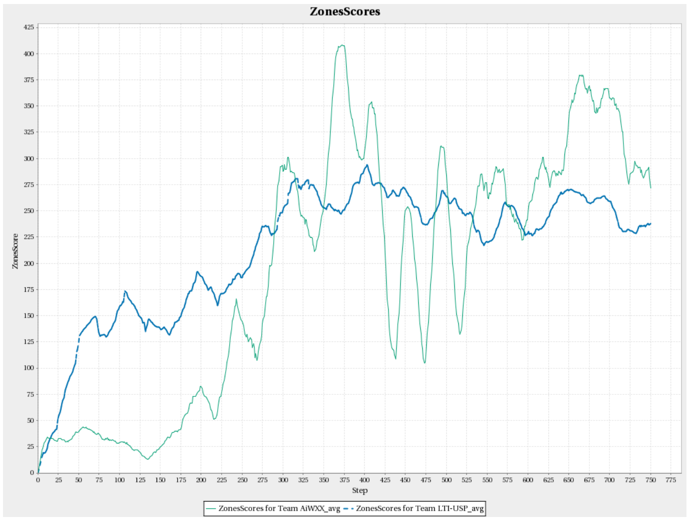

Fig. 5. LTI-USP vs AiWYX (second match): Zone score by step.

5 Conclusions

Participating in the MAPC was a great opportunity to improve our knowledge on developing MAS, and on the *JaCaMo* framework.

Due to the modularity of the *JaCaMo* framework it was not complicated to change our team for this year Contest. We could reuse all the architecture built to communicate with the MASSim server, and to capture the agent's local view of the world. The main changes were in the team's organisation, through the *Moise* specifications, and in creating the plans for the new roles. We believe that *JaCaMo* proved to be a flexible platform, allowing us to easily change our strategy and to test some of its variations.

In summary, given the effort put to the development of the team, we were pleased with the final result. Comparing with last year, we reached a better zone stability by (i) moving to a more decentralized approach, with the agents deciding by their own were to move to create or expand the team's zone, and by (ii) adopting a defensive strategy, with the `guardian` ready to attack any close opponent, and the `medic` in the center of the zone focused on repairing the agents.

Regarding possible improvements for the current scenario, we would propose to increase the probability of success for the ranged actions, since we noticed during the competition that these actions has a huge chance to fail, not being worth to use them. Another idea is to change the score computation to consider only the zones values. In this way, the buying strategy will not impact directly the team score and it will be interesting to see how each team would invest their achievement points.

Acknowledgements

Jaime Simão Sichman is partially supported by CNPq and FAPESP/Brazil.

References

1. Köster, M., Schlesinger, F., Dix, J.: The Multi-Agent Programming Contest 2012. In Dastani, M., Hübner, J.F., Logan, B., eds.: Programming Multi-Agent Systems. Volume 7837 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 174–195
2. Bordini, R.H., Gouveia, G.P., Pereira, R.H., Picard, G., Piunti, M., Sichman, J.S.: Using Jason, Moise, and CArtAgO to Develop a Team of Cowboys. In: Proceedings of 10th International Workshop on Computational Logic in Multi-Agent Systems. (2009) 203–207
3. Gouveia, G., Pereira, R., Sichman, J.: The USP Farmers herding team. Annals of Mathematics and Artificial Intelligence **61** (2011) 369–383 10.1007/s10472-011-9238-x.
4. Franco, M., Rosset, L., Sichman, J.: LTI-USP Team: A JaCaMo Based MAS for the MAPC 2012. In Dastani, M., Hübner, J., Logan, B., eds.: Programming Multi-Agent Systems. Volume 7837 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 224–233
5. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming (2011)
6. Bordini, R., Hübner, J., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. (2007)
7. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Autonomous Agents and Multi-Agent Systems **23**(2) (June 2010) 158–192
8. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Autonomous Agents and Multi-Agent Systems **20**(3) (April 2009) 369–400
9. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away. MAAMAW '96, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (1996) 42–55
10. Behrens, T.M., Dix, J., Hindriks, K.V.: The Environment Interface Standard for Agent-Oriented Programming - Platform Integration Guide and Interface Implementation Guide. Department of Informatics, Clausthal University of Technology, Technical Report **IfI-09-10** (2009)
11. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems **17**(3) (December 2008) 432–456
12. Hübner, J., Sichman, J., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering (2007) 1–27

Short Answers

A Introduction

1. What was the motivation to participate in the contest?

A: The main motivation to participate in the contest was to test and evaluate the JaCaMo framework.

2. What is the (brief) history of the team? (MAS course project, thesis evaluation, ...)

A: The LTI-USP participated in the 2012 edition of the MAPC and also in previous years. Since our first participation, the MAPC has been used to evaluate platforms and tools, and to improve our knowledge in developing MAS. The previous Cows and Cowboys scenario was used in the last two years of the Multi-Agent course held at the Department of Computer Engineering and Digital Systems of the University of São Paulo.

3. What is the name of your team?

A: LTI-USP.

4. How many developers and designers did you have? At what level of education are your team members?

A: The LTI-USP team was formed by Mariana Ramos Franco (M.Sc. Student) and Jaime Simão Sichman (Professor).

5. From which field of research do you come from? Which work is related?

A: The LTI-USP, located at the University of São Paulo is one of the most relevant research groups in multi-agent systems in Brazil. In cooperation with other research groups in DAS/UFSC (Brazil) and ISCOD/LSTI/ENSMSE (France), our group is one of the responsibles for the development and maintenance of the Moise organisational model.

B System Analysis and Design

1. Did you use multi-agent programming languages? Please justify your answer.

A: We developed our team using the JaCaMo framework. JaCaMo is a platform for multi-agent programming which supports all levels of abstractions - agent, environment, and organisation - that are required for developing sophisticated multi-agent systems, by combining three separate technologies: Jason, for programming autonomous agents; CArtAgO, for programming environment artifacts; and Moise, for programming multi-agent organisations.

2. If some multi-agent system methodology such as Prometheus, O-MaSE, or Tropos was used, how did you use it? If you did not, please justify.

A: For the development of this project, we choose to not use any multi-agent methodology, because we already had the 2012 team from where start to work, and mainly because we decided that it was better to spend our time improving the system than learning a methodology.

3. Is the solution based on the centralisation of coordination/information on a specific agent? Conversely if you plan a decentralised solution, which strategy do you plan to use?

A: Our team is decentralised, each agent decides by itself which empty vertex it will occupy in order to create the zone or expand it. There is no centralisation of information, each agent has its own view of the world.

4. What is the communication strategy and how complex is it?

A: In our team, each agent has its own view of the world, and they communicate with each other for the following purposes: (i) informing the others agents about the structure of the map; (ii) informing about the agent's or the opponent's position, role and status; (iii) asking for repair.

The agents' communication occurs via the speech acts provided by Jason and, to reduce the communication overhead, an agent broadcasts to all the other agents only the new percepts, i.e., only percepts received from the contest server which produces an update on the agent's world model are broadcasted. For this reason, there is a strong exchange of information between the agents in the beginning of the match due to the broadcast of new percepts, specially those related to the map, such as vertices and edges. However, the communication overhead decreases as the agents' world model starts to be more complete.

5. How are the following agent features considered/implemented: *autonomy*, *proactiveness*, *reactiveness*?

A: The agents are autonomous to decide by themselves the next action to be performed, but in cooperation with each other, particularly with the coordinator agent. The agents have a proactive behaviour, for example, to find the better vertices in the map, and to move to the closest repairer when they are damaged.

At each step, the agent decides which plan will be executed given only the state of the environment and the results of previous steps. The plan's priority is determined by the order in which the plans were declared, and the executed plan will be the first one to have its conditions satisfied. Some high priority plans can be considered reactive, such as the one which tells the agent to perform a recharge when running low on energy.

6. Is the team a truly **multi**-agent system or rather a centralised system in disguise?

A: Our system is a true multi-agent system. Each agent has its own beliefs, desires, intentions and control thread. Each agent decides by itself its next action.

7. How much time (man hours) have you invested (approximately) for implementing your team?

A: Approximately 150 man-hours were invested in the team development.

8. Did you discuss the design and strategies of you agent team with other developers? To which extend did your test your agents playing with other teams.

A: Only during the competition did we discuss the designs and strategies with the other participants, and before the tournament, we participated in some test matches set by the organizers to ensure the stability of our team.

9. What data structures are shared among the agents, and which are private of each agent?

A: Only the organisational artifacts are shared among the agents. Each agents has its own world model.

C Software Architecture

1. Which programming language did you use to implement the multi-agent system?

A: Java and AgentSpeak.

2. How have you mapped the designed architecture (both multi-agent and individual agent architectures) to programming codes, i.e., how did you implement specific agent-oriented concepts and designed artifacts using the programming language?

A: The agents are developed using the Jason MAS platform, which is a Java-based interpreter for an extended version of the AgentSpeak programming language for BDI agents. Each agent is composed of plans, a belief base and its own world model. The plans are specified in AgentSpeak and the agent decides which plan will be executed according to its beliefs and the local view of the world. The world model consists of a graph developed in Java, using simple data structures and classes.

3. Which development platforms and tools are used? How much time did you invest in learning those?

A: All our code was written using the Eclipse IDE with the Jason plugin. All members were familiar with Eclipse.

4. Which runtime platforms and tools (e.g. Jade, AgentScape, simply Java, ...) are used? How much time did you invest in learning those?

A: We have used the JaCaMo platform to run our team. The main developer was already familiar with JaCaMo.

5. What features were missing in your language choice that would have facilitated your development task?

A: The JaCaMo framework provided all the necessary features that we needed to developed our team.

6. Which algorithms are used/implemented?

A: We used the breadth-first search algorithm to find the minimum path between two vertices in the graph.

7. How did you distribute the agents on several machines? And if you did not please justify why.

A: We did not distribute the agents in several machines due to time constraints, but is our intention to work after the tournament on a distributed team, since the JaCaMo framework facilitates this.

8. Do your agents perform any reasoning tasks while waiting for responses from the server, or is the reasoning synchronized with the receive-percepts/send-action cycle?

A: At each step, the agent decides which action will be executed given only the state of the environment and the results of previous steps. So the agent reasoning is completely synchronized with the receive-percepts/send-action cycle.

9. What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?
A: Due to the modularity of the JaCaMo framework it was not complicated to change our team for this year Contest. The most difficult part was to remove the centralized coordination and define the rules that the agents must obey to create the zone or expand it.
10. How many lines of code did you write for your software?
A: Approximately 2000 lines in Java and 1800 lines in AgentSpeak.

D Strategies, Details and Statistics

1. What is the main strategy of your team?
A: The main strategy was to distribute the agents into three subgroups: two in charge of occupying the best zones in the map, and the other one in charge of sabotaging the opponents.
2. How does the overall team work together? (coordination, information sharing, ...)
A: One agent is responsible for determining which are the best zones in the map, and then each agent decides by itself what to do to create a zone in the specified location. Each agent has its own world model, and only percepts received from the contest server which produces an update on the agent's world model are broadcasted.
3. How do your agents analyze the topology of the map? And how do they exploit their findings?
A: The explorers probe all unknown vertex and the results of map analysis are exploited to find the best zones to be occupied.
4. How do your agents communicate with the server?
A: Using the EISMASSim interface.
5. How do you implement the roles of the agents? Which strategies do the different roles implement?
A: We decided to not map the five types specified in the scenario (explorer, inspector, repairer, saboteur and sentinel) direct to the roles in our team. Instead, we defined different roles in our system according to the adopted strategy. Each role has a mission associated to it, and each role can be played by one or more type of agents. For example, the `map_explorer` role can be played only by the explorer type, while the `soldier` role can be played by all types of agents.
6. How do you find good zones? How do you estimate the value of zones?
A: The best zone is obtained by calculating for each vertex the sum of its value with the value of all its direct and second degree neighbors. The vertex with the greatest sum of values is the center of the best zone. Zones with the sum of values below 10 are not considered in the calculation.
7. How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

A: Given that the coordinator has assigned a zone for a group, all agents of the group moves to the specified location and then each agent decides by itself which empty vertex it will occupy in order to create the zone or expand it. We have implemented a defense strategy, whith the `guardian` agent ready to attack any close opponent, and the `medic` in the center of the zone focused on repair the agents.
We also developed a group to attack the opponents'zone.

8. Can your agents change their behavior during runtime? If so, what triggers the changes?

A: Yes. In the beginning, one `map_explorer_helper` has the mission of helping the `map_explorer` to explore the graph. After the step 250, the agent leaves this role to adopt the `soldier` role in the `best_zone` subgroup.

9. What algorithm(s) do you use for agent path planning?

A: Breadth-first search algorithm.

10. How do you make use of the buying-mechanism?

A: We decided to limit the buy action, allowing only the agents of type saboteur and repairer to purchase an unique extension pack of `sensors`, in order to they be able to attack or repair agents in neighbour vertices.

11. How important are achievements for your overall strategy?

A: The achievements were very important in the team score, because of that we limited the buy action.

12. Do your agents have an explicit mental state?

A: The agent's mental state consists of internal beliefs, desires, intentions, and plans.

13. How do your agents communicate? And what do they communicate?

A: The agents' communication occurs via the speech acts provided by Jason. They communicate with each other for the following purposes: (i) informing the others agents about the structure of the map; (ii) informing about the agent's or the opponent's position, role and status; (iii) asking for a repair.

14. How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

A: We used the Moise model to explicitly specify the organizational constraints of our team. We organized our agents in three groups: two in charge of occupying the best zones in the map, and the other one in charge of attacking the opponents.

15. Is most of you agents' behavior emergent on and individual and team level?

A: Each agent acts individually and they are autonomous to decide by themselves the next action to be performed, but in cooperation with each other.

16. If you agents perform some planning, how many steps do they plan ahead.

A: Our agents do not plan ahead. Plans are recalculated in each step.

17. If you have a perceive-think-act cycle, how is it synchronized with the server?

A: After send an action, the agent stay in wait until receive new percepts from the server to start a new perceive-think-act cycle.

E Conclusion

1. What have you learned from the participation in the contest?

A: Participating in the MAPC was a great opportunity to improve our knowledge on developing MAS, and on the JaCaMo framework.

2. Which are the strong and weak points of the team?

A: We believe that the strong point of our team was the defensive strategy, since it resulted in more stable zones. The weak point was the size of our zones.

3. How suitable was the chosen programming language, methodology, tools, and algorithms?

A: The JaCaMo framework proved to be a very complete platform for the development of sophisticated multi-agent systems, by providing all the necessary features that we needed to developed our team.

4. What can be improved in the context for next year?

A: Besides the test matches, the organization could leave a server running set up with a dummy team, so that the participants could test the connection and communication with the server at any time. We believe also that the early release of the software package, given more time for the development of the teams, can bring more participants for the contest.

5. Why did your team perform as it did? Why did the other teams perform better/worse than you did.

A: Given the effort put to develop the team (only 150 hours and one developer), we were pleased with final result. The two teams that performed better, had much more human resources to test different strategies.

6. Which other research fields might be interested in the Multi-Agent Programming Contest?

A: Algorithms, Game development, Game theory, AI, Robotics.

7. How can the current scenario be optimized? How would those optimization pay off?

A: Regarding possible improvements for the current scenario, we would propose to increase the probability of success for the ranged actions, since we noticed during the competition that these actions has a huge chance to fail, not worth to use them. Another idea is to change the score computation to consider only the zones values. In this way, the buying strategy will not impact directly the team score and it will be interesting to see how each team will invest their achievement points.