The First Contest on Multi-Agent Systems based on Computational Logic

Mehdi Dastani[1], Jürgen Dix[2] and Peter Novak[2]

[1]Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
`mehdi@cs.uu.nl`
[2]Clausthal University of Technology
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
`dix@tu-clausthal.de, peter.novak@in.tu-clausthal.de`

Abstract. This is a short report about the first contest of Multi-Agent Systems (MASs) that are based on computational logic. The CLIMA workshop series (which started in 1999) is a forum to discuss techniques, based on computational logic, for representing, programming, and reasoning about Multi-Agent Systems in a formal way. Now in its seventh year, it was felt that organising a competition for evaluating MASs based on computational logic is appropriate. The authors took on this task, which turned out to be quite difficult under the given time frame. We believe that this competition is a first (modest) step towards (1) collecting important benchmarks, (2) identifying advantages/shortcomings and, finally, (3) analysing promising approaches and techniques from Computational Logic that can be used in Multi-Agent Systems.

1 Introduction

Multi-Agent Systems are beginning to play an important role in today's software development: The *International Journal of Agent-Oriented Software Engineering (IDOSE)*[1], the *International Workshop on Agent-Oriented Software Engineering (AROSE)*[2] and the *International Joint Conference on Autonomous Agents and Multi Agent Systems* are just examples for that trend.
The development of MASs requires efficient and effective solutions for different problems which can be classified into two classes: the problems related to (1) the development of individual agents and (2) the development of their interactions. Typical problems related to individual agents are how to specify, design and implement issues such as *autonomy*, *pro-active/reactive behaviour*, *perception and update of information*, *reasoning and deliberation*, and *planning*. Typical problems related to the interaction of individual agents are how to *specify*, *design and implement* issues such as *communication*, *coordination*, *cooperation* and *negotiation.*

This competition is a first attempt to stimulate research in the area of MASs by

1. identifying key problems, and
2. collecting suitable benchmarks

that can serve as milestones for testing new approaches and techniques from computational logic. While there already exist several competitions in various parts of artificial intelligence (theorem proving, planning, Robo-Cup, etc.) and, lately, also in specialised areas in agent systems (Trading Agent Competition (TAC) [3] and AgentCities competitions [4]), the emphasis of this contest is on the use of *computational logic* in MASs. We believe that approaches and techniques of computational logic are essential for the development of MASs for at least two reasons: (1) logical approaches have proven to be very useful for specifying and modelling MASs in a precise manner, and (2) the specification and models can be executed.

We expect to promote the development of MASs by first identifying difficult problems and then finding solutions by comparing different approaches from computational logic for solving them. While this idea seems very appealing, it is not an easy task to come up with a particular scenario that serves as a basis for a contest. Such a scenario should be *generic* enough to be applicable for a wide range of techniques of computational logic, but it should also be *precise* enough so that different approaches can be tested and compared against each other.

2 Scenario description

This competition was organised as a part of CLIMA and consisted of developing MASs to solve a *cooperative task in a dynamically changing environment.* The environment of the MAS is a grid-like world where agents can move from one slot to a neighbouring slot if there is no agent already in that slot. In this environment, food can appear in all but one of these slots. The special slot, in which no food can appear, is considered as a depot where the agents can bring and collect their food. An agent can observe if there is food in the slot it is currently visiting. Initially, food can be placed in some randomly selected slots. During the execution, additional food can appear dynamically in randomly selected slots except the depot slot. The agents may have/play different roles (such as explorer or collector), communicate and cooperate in order to find and collect food in an efficient and effective way.

We have encouraged submissions that specify and design a MAS in terms of high-level concepts such as goals, beliefs, plans, roles, communication, coordination, negotiation, and dialogue in order to generate an efficient and effective solution for the above mentioned application. Moreover, the MAS implementations should be based on computational logic techniques (e.g., logic programming, formal calculi, etc.) and they should reflect their design in a direct and intuitive way.

We are completely aware of the fact that this scenario can also be attacked by completely different methods and approaches (e.g., based on machine learning, neural nets, etc.). In fact, we believe almost all scenarios can be modelled in various languages and programming paradigms. One important aim of this contest is to find out where exactly computational logic helps in solving particular problems and where other approaches are superior.

The challenge of this competition is thus to use computational logic techniques to provide implemented models for the abstract concepts that are used in the specification and design of MASs. These implemented models should be integrated to implement the above-mentioned application intuitively, directly, and effectively.

3 Submission format

A submission consisted of two parts. The first part is a description of analysis, design and implementation of a MAS for the above application. Existing MASs methodologies such as Gaia[16], Prometheus[15] and Tropos[10] can be used (not demanded) to describe the analysis and design of the system. For the description of the implementation, it should be explained how the design is implemented. This can be done by explaining, for example, which computational logic techniques are used to implement certain aspects of the MAS (including issues related to individual agents).

The second part is an (executable) implementation of the application. We did not demand any particular way (data format, algorithm, mechanism) to implement the system as long as it is implemented as a MAS and as long as the environment is a 20x20 grid. Moreover, it should be possible to configure the initial state of the environment to place food in arbitrary slots. During the execution food should appear automatically every 20 secondsin a randomly selected slot. The MAS was run with 4 agents that were positioned initially at the corners of the grid. The implementation should be executable on a PC running either Microsoft Windows or Linux OS.

3.1 Received Submissions

We have received four submissions for this first edition of the CLIMA contest. From the received submissions, only one submission did use an existing multi-agent methodology to develop a running system. Moreover, some submissions did explain explicitly which techniques from computational logic are used to develop certain aspect of the MAS efficiently and effectively, while the use of computational logic techniques in other submissions seemed to be limited to the use of the language Prolog for the system implementation.

The submission from Carlos Cares [9] analyses the scenario and designs a MAS in a systematic manner using Tropos, a well-known MAS methodology. The scenario is analysed in terms of multi-agent concepts and features such as actors, roles, beliefs, goals, plans, capabilities, commitments and resources. Based on these concepts, a system is designed in terms of instantiations of these concepts resulting a set of agents. In this submission, the Tropos methodology was used to semi-automatically generate code which was then extended with a

Prolog implementation phase. This allowed the implementation of the Tropos-based architecture in terms of Prolog data structures such as lists, predicates, and rules.

The submission from Simon Coffey and Dorian Gaertner [11] does not use any existing MAS methodology to develop their system. They provide directly a system architecture consisting of BDI agents that sense the grid environment to update their beliefs, evaluate their intentions, communicate with other agents, and select and execute actions. The agents are able to negotiate over their intentions to improve the efficiency of food collection. They also introduce different roles that agents can play such as the scouting role: a role for finding food. Based on different agents roles, they discuss a second system that consists of two types of agents: the agents that can only play the scouting role and the agents that can find and collect foods. Although in the proposed system the agents are static and can play only one role, they discuss the possibility of agents that can play different roles and can change their roles dynamically. In this submission, the designed system is implemented using Qu-Prolog that allows multi-threaded execution of agents.

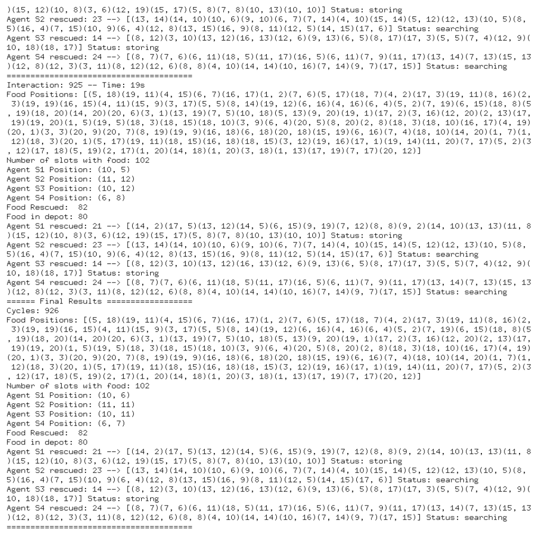
The solution of Robert Logie, Jon G. Hall and Kevin G. Waugh [13] consisted of a purely reactive system of agents with no internal representation of the current state. Their system resembled Brookes subsumption architecture and had the notion of a *role* (or policy) at its core. Agents use certain roles and can switch between them when the environment changes. They use the idea of pheromone trails in order to find interesting and successful paths (their agents do not have a memory). Although their system does not seem to use computational logic in an extensive way, it has been motivated from research on normative reasoning in deontic logic. An interesting idea is that for more complex systems, this might lead to agents that develop and create new roles (in addition to those originally specified).

The final submission, by Eder Mateus Nunes Goncalves and Guilherme Bittencourt [14], concentrated on the notion of coordination between agents in a MAS. Each agent maintains a knowledge base and updates it accordingly. The underlying notion is a high-level petri net. Agents start cooperating with the agent closest to the food (once it has been found). The cooperation ends when the food is delivered at the depot. Messages are FIPA compliant. One of the main results is the influence of the appearance of new food (and the time it takes to store food in the depot) to the impact of cooperation between the agents. If the time interval for new food to appear is small with respect to the time it takes to store it, than cooperation pays off.

4 Winning Criteria

The criteria used to evaluate submissions and to select the winners were as follows:

1. Original, innovative, and effective application of computational logic techniques in solving specific multi-agent issues identified in this application.



```
)(15, 12)(10, 8)(3, 6)(12, 19)(15, 17)(5, 8)(7, 8)(10, 13)(10, 10)] Status: storing
Agent S2 rescued: 23 --> [(13, 14)(14, 10)(10, 6)(9, 10)(6, 7)(7, 14)(4, 10)(15, 14)(5, 12)(12, 13)(10, 5)(8,
5)(16, 4)(7, 15)(10, 9)(6, 4)(12, 8)(13, 15)(16, 9)(8, 11)(12, 5)(14, 15)(17, 6)] Status: searching
Agent S3 rescued: 14 --> [(8, 12)(3, 10)(13, 12)(16, 13)(12, 6)(9, 13)(6, 5)(8, 17)(17, 3)(5, 5)(7, 4)(12, 9)(
10, 18)(18, 17)] Status: storing
Agent S4 rescued: 24 --> [(8, 7)(7, 6)(6, 11)(18, 5)(11, 17)(16, 5)(6, 11)(7, 9)(11, 17)(13, 14)(7, 13)(15, 13
)(12, 8)(12, 3)(3, 11)(8, 12)(12, 6)(8, 8)(4, 10)(14, 14)(10, 16)(7, 14)(9, 7)(17, 15)] Status: searching
=======================================
Interaction: 925 -- Time: 19s
Food Positions: [(5, 18)(19, 11)(4, 15)(6, 7)(16, 17)(1, 2)(7, 6)(5, 17)(18, 7)(4, 2)(17, 3)(19, 11)(8, 16)(2,
 3)(19, 19)(16, 15)(4, 11)(15, 9)(3, 17)(5, 5)(8, 14)(19, 12)(6, 16)(4, 16)(6, 4)(5, 2)(7, 19)(6, 15)(18, 8)(5
, 19)(18, 20)(14, 20)(20, 6)(3, 1)(13, 19)(7, 5)(10, 18)(5, 13)(9, 20)(19, 1)(17, 2)(3, 16)(12, 20)(2, 13)(17,
 19)(19, 20)(1, 5)(19, 5)(18, 3)(18, 15)(18, 10)(3, 9)(6, 4)(20, 5)(8, 20)(2, 8)(18, 3)(18, 10)(16, 17)(4, 19)
(20, 1)(3, 3)(20, 9)(20, 7)(8, 19)(19, 9)(16, 18)(6, 18)(20, 18)(15, 19)(6, 16)(7, 4)(18, 10)(14, 20)(1, 7)(1,
 12)(18, 3)(20, 1)(5, 17)(19, 11)(18, 15)(16, 18)(18, 15)(3, 12)(19, 16)(17, 1)(19, 14)(11, 20)(7, 17)(5, 2)(3
, 12)(17, 18)(5, 19)(2, 17)(1, 20)(14, 18)(1, 20)(3, 18)(1, 13)(17, 19)(7, 17)(20, 12)]
Number of slots with food: 102
Agent S1 Position: (10, 5)
Agent S2 Position: (11, 12)
Agent S3 Position: (10, 12)
Agent S4 Position: (6, 8)
Food Rescued:  82
Food in depot: 80
Agent S1 rescued: 21 --> [(14, 2)(17, 5)(13, 12)(14, 5)(6, 15)(9, 19)(7, 12)(8, 8)(9, 2)(14, 10)(13, 13)(11, 8
)(15, 12)(10, 8)(3, 6)(12, 19)(15, 17)(5, 8)(7, 8)(10, 13)(10, 10)] Status: storing
Agent S2 rescued: 23 --> [(13, 14)(14, 10)(10, 6)(9, 10)(6, 7)(7, 14)(4, 10)(15, 14)(5, 12)(12, 13)(10, 5)(8,
5)(16, 4)(7, 15)(10, 9)(6, 4)(12, 8)(13, 15)(16, 9)(8, 11)(12, 5)(14, 15)(17, 6)] Status: searching
Agent S3 rescued: 14 --> [(8, 12)(3, 10)(13, 12)(16, 13)(12, 6)(9, 13)(6, 5)(8, 17)(17, 3)(5, 5)(7, 4)(12, 9)(
10, 18)(18, 17)] Status: storing
Agent S4 rescued: 24 --> [(8, 7)(7, 6)(6, 11)(18, 5)(11, 17)(16, 5)(6, 11)(7, 9)(11, 17)(13, 14)(7, 13)(15, 13
)(12, 8)(12, 3)(3, 11)(8, 12)(12, 6)(8, 8)(4, 10)(14, 14)(10, 16)(7, 14)(9, 7)(17, 15)] Status: searching
====== Final Results ==================
Cycles: 926
Food Positions: [(5, 18)(19, 11)(4, 15)(6, 7)(16, 17)(1, 2)(7, 6)(5, 17)(18, 7)(4, 2)(17, 3)(19, 11)(8, 16)(2,
 3)(19, 19)(16, 15)(4, 11)(15, 9)(3, 17)(5, 5)(8, 14)(19, 12)(6, 16)(4, 16)(6, 4)(5, 2)(7, 19)(6, 15)(18, 8)(5
, 19)(18, 20)(14, 20)(20, 6)(3, 1)(13, 19)(7, 5)(10, 18)(5, 13)(9, 20)(19, 1)(17, 2)(3, 16)(12, 20)(2, 13)(17,
 19)(19, 20)(1, 5)(19, 5)(18, 3)(18, 15)(18, 10)(3, 9)(6, 4)(20, 5)(8, 20)(2, 8)(18, 3)(18, 10)(16, 17)(4, 19)
(20, 1)(3, 3)(20, 9)(20, 7)(8, 19)(19, 9)(16, 18)(6, 18)(20, 18)(15, 19)(6, 16)(7, 4)(18, 10)(14, 20)(1, 7)(1,
 12)(18, 3)(20, 1)(5, 17)(19, 11)(18, 15)(16, 18)(18, 15)(3, 12)(19, 16)(17, 1)(19, 14)(11, 20)(7, 17)(5, 2)(3
, 12)(17, 18)(5, 19)(2, 17)(1, 20)(14, 18)(1, 20)(3, 18)(1, 13)(17, 19)(7, 17)(20, 12)]
Number of slots with food: 102
Agent S1 Position: (10, 6)
Agent S2 Position: (11, 11)
Agent S3 Position: (10, 11)
Agent S4 Position: (6, 7)
Food Rescued:  82
Food in depot: 80
Agent S1 rescued: 21 --> [(14, 2)(17, 5)(13, 12)(14, 5)(6, 15)(9, 19)(7, 12)(8, 8)(9, 2)(14, 10)(13, 13)(11, 8
)(15, 12)(10, 8)(3, 6)(12, 19)(15, 17)(5, 8)(7, 8)(10, 13)(10, 10)] Status: storing
Agent S2 rescued: 23 --> [(13, 14)(14, 10)(10, 6)(9, 10)(6, 7)(7, 14)(4, 10)(15, 14)(5, 12)(12, 13)(10, 5)(8,
5)(16, 4)(7, 15)(10, 9)(6, 4)(12, 8)(13, 15)(16, 9)(8, 11)(12, 5)(14, 15)(17, 6)] Status: searching
Agent S3 rescued: 14 --> [(8, 12)(3, 10)(13, 12)(16, 13)(12, 6)(9, 13)(6, 5)(8, 17)(17, 3)(5, 5)(7, 4)(12, 9)(
10, 18)(18, 17)] Status: storing
Agent S4 rescued: 24 --> [(8, 7)(7, 6)(6, 11)(18, 5)(11, 17)(16, 5)(6, 11)(7, 9)(11, 17)(13, 14)(7, 13)(15, 13
)(12, 8)(12, 3)(3, 11)(8, 12)(12, 6)(8, 8)(4, 10)(14, 14)(10, 16)(7, 14)(9, 7)(17, 15)] Status: searching
=======================================
```

Fig. 1. Screenshot from Concalves et al.

2. The performance of the executable implementation. The performance of the executable implementation will be measured based on the amount of food that is collected by the MAS in a certain period of time. All programs will be run on the same machine (Windows/Linux double boot machine).
3. The quality of the description of analysis, design and implementation of the MAS, the elegance of its design and implementation, and the ease of installation and execution of the program.

5 How to determine the winning system?

To determine a winner turned out to be a very difficult task. Three of the four systems were very close and we finally decided to select two of the three as winners. It should be noted that our decision was based on the packages we got, the problems we had to install them and our impression from the description of these packages and their underlying theory. We decided not to go into lengthy discussions with the authors as to the *why* and *how* of their systems.

While we did the best to achieve a fair evaluation, there are certainly no perfectly objective criteria we could have used: even the performance of the systems was not comparable. We believe that by using a server architecture for the second contest, we can at least measure the performance of the different approaches in a fair manner.

As an example, one of the criteria was the use of computational logic. Thus a system that seemed to be based on a simple reflex-architecture (as Logie et al. [13]) was not as highly ranked as others. But Logie-agents did use only a local view and no memory, so they might be much more efficient in unknown or changing environments.

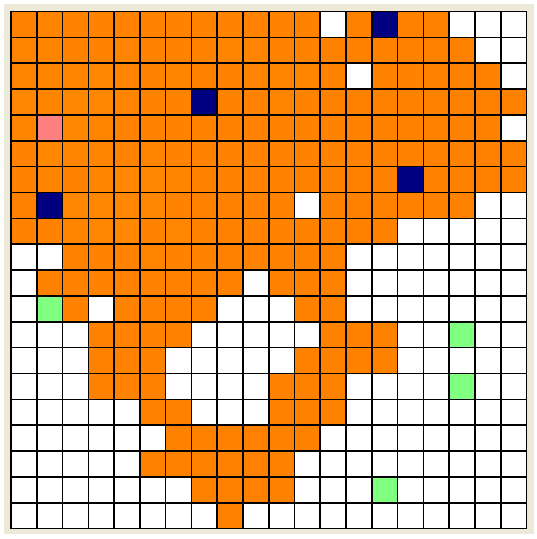


Fig. 2. Screenshot from Logie et al.

5.1 Technical Issues

In the call for submissions we did not impose any technical requirements on the implementation and installation procedure of the submitted softwares. Unfortunately, this turned out to be a serious bottleneck of the evaluation process. In fact, there were many technical problems varied from missing files for visualizing the simulation results (Cares et al.), using obsolete C++ compiler version (Concalves et al.), to using a number of not very well integrated external packages and libraries (Coffey et al.). The only submission without such technical problems was from Logie et. al. Fortunately, all the problems were finally solved in cooperation with the corresponding authors.

The authors used various supporting technologies for their implementations. While submissions of Logie et al. and Cares et al. were standalone MS Windows executables, submissions of Concalves et al. and Coffey et al. were source code packages for Linux OS. Although not required, most submitted softwares, except Concalves et al., did include a visualization component. Figures 2, 3 and 4 illustrate the screenshots of the visualization component of the submissions from Logie et al., Cares et al. and Coffey et al., respectively. While Logie et al. and Coffey et al. were visualizing the running system on-the-fly, the program from Cares et al. was generating a HTML file with embedded JavaScript code, which

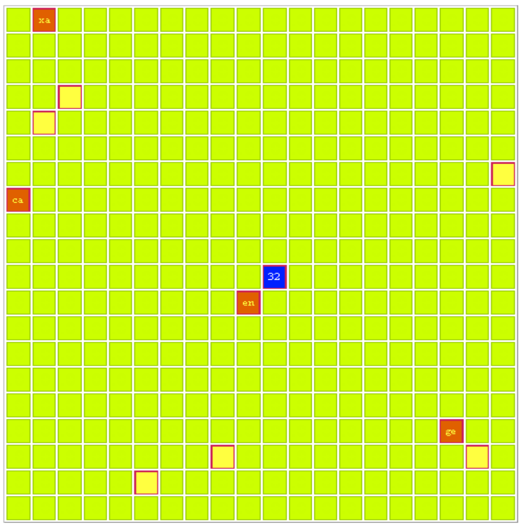


Fig. 3. Screenshot from Cares et al.

could be later viewed using standard HTML browser. The output generated by the submission from Concalves et. al. is illustrated in Figure 1.

5.2 Evaluation of Submitted Programs

Since participants of the contest used different approaches to implement the scenario, performance could not be considered as the most important criterion to compare them. Clearly, the systems in which each agent has a global view of the grid and is aware of its relative position with respect to the other agents (e.g. Cares et al.), has an advantage over the systems in which agents have only a local view and no memory (e.g. Logie et al.). Therefore, we rather evaluated the systems based on the originality of the idea and the strategy to collect food. The amount of collected food is then considered as a second-class criterion.

Overall duration of a simulation run	3 min.
Interval of random food generation	5 sec.
Amount of food seeded in one food generation	1 piece
Amount of food at starting configuration	0 pieces
Depot position	$\langle 10, 10 \rangle$
Number of agents	4
Starting positions of agents	grid corners

Table 1. Parameters of simulation used in the final evaluation.

We executed simulations with the parameters presented in Table 1 on the same double-boot Windows XP/Linux computer with Intel Pentium 4 CPU with tact-frequency 2.80 GHz and 1GB of RAM. In some cases, it was not possible to comply with the simulation parameters set by the authors. For example, in the case of Logie et al., the depot position was not configurable. It was placed randomly at the beginning of the simulation. Also the simulation by Concalves et al. did not allow us to configure the details of the simulation and finished automatically after approximately 20 seconds with the exit status OK.

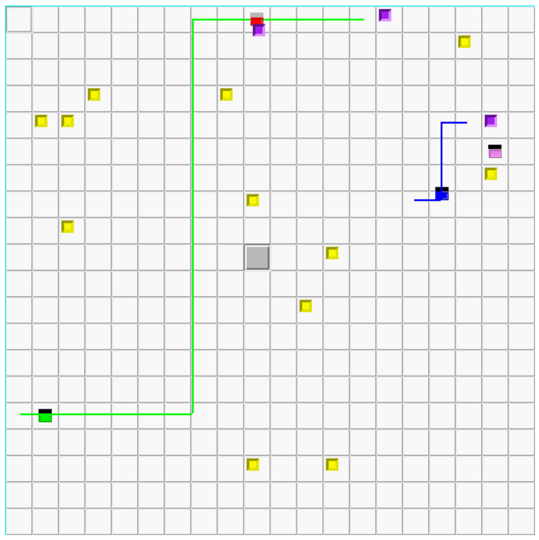


Fig. 4. Screenshot from Coffey et al.

5.3 Results and Winners

In the following, we describe the submitted multi-agent systems from a *black box* perspective. This means that we describe the behavior of the multi-agent simulations without considering their system descriptions. Note that the system descriptions are briefly summarized in subsection 3.1. Table 2 presents the results of evaluated simulation runs.

The best performance in terms of collected food was achieved by Concalves et al. However, this system exited unexpectedly before the allowed three minutes run time without any error message. Since it wasn't possible to configure the starting amount of the food in the grid and the interval of its generation, according to the simulation log, agents were able to collect 165 food items, although this is not in a correlation with the required interval of food generation (e.g. food item every 20 seconds as specified in the call for submissions). It was

submission	**collected food**	**food picked up**	**performed steps**
Cares et al.	25	25	1066
Coffey et al.	15	47	not reported
Concalves et al.	165	165	1826
Logie et al.	4	6	3600

Table 2. Results of evaluated simulation runs.

also not easy to verify the details of the simulation run because the program generated only a text stream to the standard output. Figure 1 shows the few lines of the simulation log. Because of these reasons and the fact that the use of computational logic techniques were quite limited, we decided not to consider this submission for the first rank.

The worst performer was the simulation submitted by Logie et al. In our opinion, the poor performance was due to the fact that the agents neither had an internal representation of the grid environment nor could communicate directly. Agents were randomly walking around in the grid and once they found a depot cell, they started to leave pheromone trail. This could help other agents to find the depot. At the beginning of the simulation, the paths of individual agents to the depot was not very direct, but after a while, a pheromone gradient could be observed in the grid. This was visualized as an orange gradient around the depot, where the saturation of yellow color in a particular cell was proportional to a strength of a pheromone marking in the cell. In this state of the simulation, agents were able to find their ways to the depot, after picking up a food item, more directly than at the beginning of the simulation. Unfortunately, this strategy was not performing very well in the short simulation runs, which was also a reason why this program did not qualify for higher ranking in the contest. Figure 2 shows a screenshot of the visualization component of the simulation by Logie et al. The four darkest cells are agents and the brightest cells (green) are available food. Since the depot position was not a configurable parameter of this simulation it was randomly placed to the position $\langle 2, 5\rangle$ at the beginning of the simulation. Around the depot cell, the area marked by pheromone is visible.

The programs submitted by Cares et al. and Coffey et al. achieved approximately the same performance results using quite similar strategies. In the simulation by Cares et al., there were two types of agents: scout agent and ordinary agent. In this simulation, there was only one scout agent who did only explore the grid by columns to look for food. All the other agents in the simulation were ordinary agents. They were searching for food exploring rows one by one. When they found food, they delivered it directly to the depot cell. All agents knew the position of the depot cell. The agents in this simulation were informed about the position of the depot cell at the beginning of the simulation. The most interesting behaviour could be observed when a scout agent found food and later met an ordinary agent. The ordinary agent started immediately to walk directly toward the food, picked it up, and delivered it to the depot. Obviously the scout agent communicated the position of the found food to the ordinary agent. Figure 3 displays a screenshot of the visualization component of the simulation by Cares et al. Agents are depicted as red squares and the scout agent is marked as *ca* (position $\langle 1, 8 \rangle$). Dark blue cell is a depot displaying the number of already collected food items. Available food is depicted as yellow squares.

The approach by Coffey et al. did also use the idea of a scout agent. Here the scout agent as well as the agents that were not able to load any food (i.e. they were already carrying an item to the depot), marked the food they found and broadcasted its position to all other agents. The agents that did not load any food yet, started a negotiation and the winner of the bidding walked to pick up the food and deliver it to the depot. The interesting aspect of this submission is that when another food item was found, the negotiation was started all over again and agents possibly rearranged their claims on particular food items they were approaching. By this the agent team was able to optimize the overall cost of food delivery (i.e. overall number of steps to perform in order to deliver given set of food items to the depot) and thus improve its overall performance.

Contrary to the submission by Cares et al., agents from simulation by Coffey et al. were not informed about the position of the depot right from the beginning of the simulation. They spent a considerable amount of time until the depot was found by one of them. After that, agents were able to clear the grid of food quite quickly starting from the closest neighborhood of the depot to the borders of the grid. Figure 4 shows a screenshot of the visualization component of the simulation by Coffey et al. Lines show the connection between an agent and food it currently claims. Yellow boxes depict undiscovered food, dark purple boxes mean that the food item on a given position was discovered but not picked up yet. Finally agents carrying a food item are depicted as a double-box (e.g. the one in the first row of the grid). The scout agent (position $\langle 19, 6 \rangle$) recently found food in the cell above it. The blue agent (nearest one to the scout) won the bidding and gave up his intention to pick up the food item in the first row of the grid. This allowed the green agent to claim the abandoned food in the first row. The agent is now heading towards it.

Because of the elegance of the implemented approach, the use of MAS technology, the non-trivial amount of computational logic and, finally, their overall performance, both submissions by Cares et al. and Coffey et al. were chosen for the CLIMA Contest 2005 prize.

6 What did we learn for the second contest

As we already mentioned in Subsection 5.1, in the course of evaluation of submitted programs, we faced a number of difficulties. On the one hand we had to deal with various technical issues and, on the other hand, we recognized that since we had no common basis for evaluation, it was very hard (almost impossible) to compare the submitted programs exclusively on the basis of their performances. Difficulties with comparison are probably more serious, because a fair evaluation of submitted programs is only possible if conditions are equal for all participating simulations.

After carefully considering all the possibilities to solve the problems mentioned above, we decided to implement a supporting infrastructure for the next edition of the CLIMA Contest. This supporting infrastructure is a server system that runs the multi-agent system environment. Teams of agents can connect to this environment after which they can perform actions (including sense action). We allow multi-agent systems, which participate in the contest, to run on their own local platforms. The involved agents can then communicate with the server system, which runs the environment, via Internet using TCP/IP protocol.

Using this approach, participating multi-agent systems will be able to use their own specialized communication technology and infrastructure. Also implementing the simulation environment centrally will take off the burden of implementing the simulation from shoulders of contest participants and allow them to focus more on the strategy for the contest scenario. Moreover, the supporting infrastructure will allow us to evaluate participating MASs on a fair basis. It will also clearly divide a simulation scenario from the team of participating agents. We believe that such a supporting infrastructure also solve the technical issues related to installation and correct execution of submitted multi-agent systems.

In order to introduce an objective evaluation criterion, we decided to allow teams of agents to compete with each other in the simulation scenario. This allow participants to explore possibilities of more complex coordination strategies. We hope that all the mentioned factors will help to allow more flexibility on participants side and also to improve the fun-factor of the competition scenario.

7 Conclusion

Given the very tight schedule (from the announcement to the submission deadline) we were quite satisfied with the four submissions. We believe this contest will promote the use of techniques and approaches from computational logic to the development and implementation of MASs. Although the contributions for this contest may propose computational logic techniques and approaches that are specific for this particular scenario (application), they may be generalised and adopted to other MAS methodologies and programming languages. In particular, we believe that this contest will stimulate the use of computational logic techniques and approaches for research and design of programming languages that support the implementation of MASs in an effective and efficient manner.

There are several existing activities that aim at stimulating research and design of programming languages for MASs. Example of such activities are the *International Workshop on Programming Multi-Agent System* [5], the *AgentLink Technical Forum Groups on Programming Multi-Agent Systems* [6], and the various seminars and books dedicated to Multi-Agent Programming [7, 12, 8]. Our experience is that many of the existing programming languages for implementing MASs, such as IMPACT, 3APL, CLAIM, JACK, Jason, and Jadex [7], are based on techniques and approaches from computational logic. In these programming languages, computational logic techniques are used to model various mental attitudes of agents such as beliefs and goals, planning components, and reasoning components.

References

1. http://www.inderscience.com/browse/index.php.
2. http://www.agentgroup.unimore.it/aose05.
3. http://www.sics.se/tac.
4. http://www.agentcities.org/EUNET/Competition.
5. http://www.cs.uu.nl/ProMAS.
6. http://www.cs.uu.nl/ mehdi/al3promas.html.
7. R. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Multi-Agent Programming: Languages, Platforms, and Applications.* Number 15 in MASA. Springer, Berlin, 2005.
8. R. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Programming Multi-Agent Systems*, volume 3346. LNAI, Springer Verlag, 2005.
9. C. Cares, X. Franch, and E. Mayol. Extending tropos for a prolog implementation: A case study using the food collecting agent problem. In *this volume*. 2005.
10. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the TROPOS project. *Information Systems*, 27:365–389, 2002.
11. S. Coffey and D. Gaertner. Implementing pheromone-based, negotiating forager agents. In *this volume*. 2005.
12. M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Programming Multi-Agent Systems*, volume 3067. LNAI, Springer Verlag, 2004.
13. R. Logie, J. G. Hall, and K. G. Waugh. Reactive food gathering, clima vi contest submission. In *this volume*. 2005.
14. E. Mateus, N. Goncalves, and G. Bittencourt. Strategies for multi-agent coordination in a grid world. In *this volume*. 2005.
15. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Agent-Oriented Software Engineering III: Third International Workshop (AOSE'02)*. Springer, LNAI 2585, 2003.
16. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.