Multi-Agent Programming Contest 2013: The Teams and the Design of their Systems

Tobias Ahlbrecht[1], Christian Bender-Saebelkampf[5], Maiquel de Brito[6], Nicolai Christian Christensen[3], Jürgen Dix[1], Mariana Ramos Franco[4], Hendrik Heller[5], Andreas Viktor Hess[3], Axel Hessler[5], Jomi F. Hübner[6], Andreas Schmidt Jensen[3], Jannick Boese Johnsen[3], Michael Köster[1], Chengqian Li[2], Lu Liu[2], Marcelo M. Morato[6], Philip Bratt Ørum[3], Federico Schlesinger[1], Tiago L. Schmitz[6], Jaime Simão Sichman[4], Kaio S. de Souza[6], Daniela M. Uez[6], Jørgen Villadsen[3], Sebastian Werner[5], Øyvind Grønland Woller[3], and Maicon R. Zatelli[6]

[1] Department of Informatics, Clausthal University of Technology,
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
[2] Dept. of Computer Science,
Sun Yat-sen University
Guangzhou 510006, China
[3] Algorithms, Logic and Graphs Section
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Matematiktorvet, Building 303B, DK-2800 Kongens Lyngby, Denmark
[4] Laboratório de Técnicas Inteligentes (LTI), Escola Politécnica (EP)
Universidade de São Paulo (USP)
[5] Distributed Artificial Intelligence Laboratory
Technische Universität Berlin, Germany
[6] Department of Automation and Systems Engineering
Federal University of Santa Catarina
CP 476, 88040-900 Florianópolis - SC - Brasil

Abstract. Five teams participated in the Multi-Agent Programming Contest in 2013: All of them gained experience in 2012 already. In order to better understand which paradigms they used, which techniques they considered important and how much work they invested, the organisers of the contest compiled together a detailed list of questions (circa 50). This paper collects all answers to these questions as given by the teams.

1 Introduction

One of the main aims of the Multi-Agent Programming Contest [1,2] is to test and evaluate multi-agent systems: Are they better suited for decentralized scenarios than more traditional approaches? Do they offer tools that can be easily used and are still sufficiently scalable? Compared to pure Java based approaches, what do we gain?

In the past, we have seen several teams not using multi-agent platforms. Some of them have been chosen by students who wanted to participate in the contest,

but who did not have a deep background in multi-agent programming. This year, only one team decided not to use any dedicated multi-agent programming language and to stick to C++ (as they did last year). It is interesting to note that the runner-up in 2011 and 2012, Python-DTU, did not use an agent programming language, but many concepts and techniques from multi-agent reasoning (programmed in Python).

An important point in the contest is the choice of the scenario. We do *not* want to evaluate a particular smart strategy to solve the task, we would like to evaluate the agent platform or software system that is used for the solution. Therefore the scenario has to be sufficiently complex, otherwise we risk that a smart team comes up with a clever solution that has noting to do with the tools provided by the underlying programming language.

It is also obvious that we can test only some features of agent platforms and languages. To evaluate the whole software development life-cycle, from requirements phase to deployment, we would have to evaluate all these phases, from the design to the final software code. This is not possible and therefore we decided only to test the final system. We hope that the questions and the answers that we have collected here shed some light on these phases as well.

2 The Contest in 2013

All five contestants in 2013 (see Table 1) also participated in the contest in 2012. However, for TUB the team members changed completely. Only one of the teams (AiWXX) did not use a multi-agent programming platform or language. The winner in 2013 already won in 2012: SMADAS-UFSC. TUB, who came fourth in 2013, won several times in the past (for different scenarios).

Team	**Affiliation**	**Platform/Language**
AiWXX[5]	Sun Yat-Sen University, China	C++
GOAL-DTU[6]	Technical University of Denmark	GOAL
LTI-USP[3]	University of Sao Paulo, Brazil	Jason, CArtAgO, $\mathcal{M}$oise
SMADAS-UFSC[8]	Federal University of Santa Catarina, Brazil	Jason, CArtAgO, $\mathcal{M}$oise
TUB[7]	TU Berlin, Germany	JIAC

Table 1. Participants of the 2013 Edition.

The person-hours invested to implement the teams range from 150 (LTI-USP), 250 (AiWXX), 400 (SMADAS-UFSC), 500 (GOAL-DTU) to 840 (TUB). The lines of code written for the teams range from 1300 (GOAL-DTU), 4000 (LTI-USP), 7996 (TUB), 8500 (SMADAS-UFSC), to 11000 (AiWXX).

LTI-USP and AiWXX used a centralized approach, TUB decided for a hybrid method, i.e., centralized regarding zone building, decentralized otherwise.

The remaining two teams, winner SMADAS-UFSC and runner-up GOAL-DTU implemented a purely decentralized approach.

Table 2 shows the results. SMADAS-UFSC won again (after 2012) and GOAL-DTU was again runner-up (but using this time a different agent programming language).

Pos.	Team	Score	Difference	Points
1	SMADAS-UFSC	2702948 : 1455163	1247785	36
2	GOAL-DTU	2284575 : 1614711	669864	27
3	LTI-USP	2117299 : 2083105	34194	15
4	TUB	1412702 : 2238820	-826118	6
5	AiWXX	1516760 : 2642485	-1125725	6

Table 2. Results.

When we introduced our Mars-scenario for the first time in 2011, a team from the Netherlands (headed by Koen Hindriks) won using the agent language GOAL. A detailed description of the team and the architecture of the system recently appeared in [4].

When we introduced the Mars scenario in 2011, three teams used a multi-agent platform (among them the winner and the third place) and six did not. In 2012, again three teams used a multi-agent platform, the remaining four did not. As the year before, the winner and third place in 2012 used a multi-agent platform. Interestingly, the runner-up in both years (DTU) used Python and not a dedicated agent programming language. In 2013, DTU used GOAL (the language of the winning team in 2011) but again came second.

3 Questions and Answers

We have collected over 50 questions, arranged into five different groups: (1) information about the team (motivation, number of members and time invested, background of team members), (2) system analysis and design (centralized or not, multi-agent language or not, communication, mental states), (3) software architecture (programming language, development/runtime tools, algorithms used, reasoning of the agents, lines of code), (4) strategies and some details related to the MARS scenario (strategy, information sharing, exploring the topology, communication with the server, building zones, assigning roles to agents, changing behaviour at runtime, achievements, mental state of agents), and, finally, (5) lessons learned (reasons for the performance of the team, weak and strong points when compared with other teams, how to improve the contest in the future).

3.1 Teams and their Background

This group of questions collects information about the motivation of the teams, their size and their background.

What was the motivation to participate in the contest?

SMADAS-UFSC: Testing the JaCaMo platform in the contest scenario and evaluate some other technologies developed in our master and PhD thesis.

GOAL-DTU: We participated in the contest because we find the contest very interesting for both research and teaching.

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

TUB: The main motivation was and is to test, benchmark and improve our JIAC V agent framework; the contest was the first application of the framework; we still need to improve a lot of things and the contest then says whether it is with good performance or works reliably; the benefit of a solid agent framework can be seen in many applications of the JIAC framework in projects whenever a distributed architecture makes sense.

AiWXX: Our motivation was to gain a deeper understanding about Multi-agent Systems.

What is the history of the team? (course project, thesis, ...)

SMADAS-UFSC: Our team was formed by members from the Multi-Agent Systems research group (called SMADAS) at Federal University of Santa Catarina (UFSC).

GOAL-DTU: The team consists of both researchers/students from previous years and students taking a special course.

LTI-USP: 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.

TUB: We started participation in 2007 with the JIAC IV agent framework, which was heavily loaded with AI concepts such as own ontology language, trinary propositional calculus, first-order logic, situation calculus, reaction rules, an own agent programming language (JADL), an own component system. At this time, it was already clear that such a framework is not maintainable, too difficult to learn and to use. We then started a bottom-up approach where we used a third-party component framework and added those components that we really needed to make the agents and a programming language people really use. One year later the contest was the first touchstone where we tested if our concept holds in reality, and what nobody could envision we won. In 2008 the contest helped us to get the teething troubles out of the way and to develop really useful features that make the life of the programmer easier. We then started to offer a course on multi-agent programming where we use the JIAC framework and the contest scenarios to teach agent programming principles, usually to students that do not have experience neither in software engineering nor in agent programming. The schedule of this course is to develop a solution for the Multi-Agent-Contest and participate in the contest at the end of the semester, implicitly testing and improving our agent framework.

AiWXX: Our team consists of two members, Chengqian Li and Lu Liu. Last year Chengqian Li participated in MAPC 2012. And a Multi-Agent scenario was used in his thesis.

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

SMADAS-UFSC: Our team has seven developers and everyone was involved with the system design. We have one PhD, four PhD students, and two undergraduate students.

GOAL-DTU: We are 7 computer scientists: associate professor Jørgen Villadsen (PhD), Andreas Schmidt Jensen (PhD student), Nicolai Christian Christensen (MSc student), Andreas Viktor Hess (BSc student), Jannick Boese Johnsen (MSc student), Øyvind Grønland Woller (BSc student) and Philip Bratt Ørum (MSc student).

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

TUB: During the course we had twelve students split into two teams. During preparation to the contest our team consisted of three computer science students: Hendrik Heller, Christian Bender-Seaelkampf and Sebastian Werner. As well as the course supervisor Axel Heßler. The students start their fourth Bachelor semester when they joined the course.

AiWXX: Our team consists of two second-year postgraduate students.

What is your field of research? Which work therein is related?

SMADAS-UFSC: All team members work with Multi-Agent Systems and Artificial Intelligence.

GOAL-DTU: Our field of research is AI with an emphasis on algorithms and logic.

LTI-USP: 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 $\mathcal{M}$oise organisational model.

TUB: Main field is agent-oriented software engineering, distributed and complex systems, we usually do more projects that apply agent principles and methods than basic research.

AiWXX: Data structures, algorithms and the Boolean Satisfiability (SAT) problem.

3.2 System Analysis and Design

Moreover, we wanted to know why (if at all) an agent approach has been chosen and whether the approach was centralised or not. Did agents share some information with others? How did the agents communicate and how were autonomy and proactiveness implemented.

Did you use multi-agent programming languages? Why?

SMADAS-UFSC: We used the JaCaMo framework. Thus, we used Jason for implementing the agents, CArtAgO for the environment and the $\mathcal{M}$oise organizational model to specify the organization.

GOAL-DTU: We use GOAL, which is a dedicated multi-agent programming language.

LTI-USP: 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 $\mathcal{M}$oise for programming multi-agent organisations.

TUB: No. The choice of language is up to the students. In fact, we use Java intentionally, for several reasons: platform independence (usually students and developer use an evil mix of operating systems, versions and distributions), the JIAC framework is written in Java although we have several language ports (e.g. JADL, Python, Scala, BPMN), most multi-agent programming languages are logic-based and most students at that point in their studies are not familiar with logic programming.

AiWXX: No, we did not because we are proficient in the C++ language which is well-known for its efficiency.

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

SMADAS-UFSC: We did not use any existing AOSE method. The problem seemed too easy and there was no need to use a complete methodology.

GOAL-DTU: We have not used a multi-agent system methodology as we based our system analysis on the earlier Python-DTU system (2012) and our overall design on the HactarV2 system (2011).

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

TUB: We use the JIAC development methodology (MIAC). MIAC focusses on efficient and fast development and is closely related to the concepts and architecture of the JIAC framework. Although most methodologies state to be general, but implicitly they are build on an concrete (multi-)agent models and thus they are often not applicable in another context. Many agent methodologies focus on design and thus often leave out important software engineering disciplines such as requirement elicitation, implementation and testing, deployment, management and maintenance. MIAC is still not complete in that sense but gives developers useful guidelines for intuitive understanding of what to do based on what they already know about programming and development and then they can focus on solving the problem.

AiWXX: No, we did not use any of them because we thought our framework was good enough for almost every strategy. And we are proficient in C++ language which is well-known for its efficiency.

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?

SMADAS-UFSC: The coordination is mostly based on the $\mathcal{M}$oise organizational structure. However, we use an agent - called coach, which adopts the role of *leader* - that manages the organization and performs the setup of organizational structure.

GOAL-DTU: Our solution is generally a decentralized system, though some features are centralized. Our implementation uses the new multi-threading feature of GOAL.

LTI-USP: 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.

TUB: The behaviour of the agents (i.e. the roles) is completely decentralised. Each agent has its own world model and decides on its own what to do next. The agent communicate their perception and their decision with each other, so the world model is more complete. There is one exception, the calculation of promising zones to be captured is done by an extra agent, which assigns positions to agents that are free to build a zone.

AiWXX: Our framework is a decentralised solution. However, the current implementation is restricted because we only deal with common knowledge.

What is the communication strategy and how complex is it?

SMADAS-UFSC: The agents use message exchange to call repairers, saboteurs or inform others about good vertices and map regions. Other information is shared through a blackboard.

GOAL-DTU: We aim to send as few messages between the agents as possible. Our agents communicate the status of themselves and enemy agents as well as map information.

LTI-USP: 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 other 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 produce an update of 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.

TUB: Each agent communicates its perception and decision with each other agent. Thus, the complexity is $2n \cdot (n-1)$ where n is the number of agents

in the setting and we are using multicast messaging to solve this. Additionally, there is a communication for zoning: each free agent communicates its availability to the zoning agent and is informed about the best position for zoning as a reply, so complexity is $2n$ in the worst case.

AiWXX: When any agent's knowledge state is updated, the other agents' knowledge states will be updated in precisely the same way because of the assumption of common knowledge.

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

SMADAS-UFSC: Our agents are autonomous to decide how to achieve their specific goals, but all of them have to attend to the organizational norms. Similarly, the agents may behave proactively or reactively in accordance with the needs. For instance, a damaged agent will call a repairer and all agents react to the environment events like the start of a step.

GOAL-DTU: Agents do not cooperate in making choices. One proactive feature is that *Repairers* repair agents that are likely to be attacked. Otherwise we are mostly reactive.

LTI-USP: 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.

TUB: JIAC V agents have their own thread of control and decide and act autonomously. We see the agents with low health level pro-actively seeking the repairer's help using a simple request, whereas probing or surveying has been implemented as a simple reactive behaviour: if the node is unprobed then probe.

AiWXX: Every agent chooses her action according to her state and no agent can control other agents. If the environment is changed, their knowledge state will also be changed as soon as they realize the changes. The agents are aggressive, that is, they keep exploring new areas of the world, never passively waiting for changes of the environment.

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

SMADAS-UFSC: Our team was developed as a true MAS composed by three dimensions: agents, organization and environment.

GOAL-DTU: It is truly a multi-agent system.

LTI-USP: 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.

TUB: We consider it a true multi-agent system as the agents run independently of each other in their own threads for life-cycle, sense-decide-act cycle, and can be distributed over CPUs, CPU-cores and the network without change to architecture, protocols and agent implementation.

AiWXX: We have exploited a decentralization framework in implementing various strategies, however, the implementation now is so restricted because we only deal with common knowledge.

How much time (person hours) have you invested (approximately) for implementing your team?

SMADAS-UFSC: Together, we used around 400 person hours divided between tests and programming.

GOAL-DTU: We have invested approximately 500 person hours.

LTI-USP: Approximately 150 person hours were invested in the team development.

TUB: Approximately 840 person hours.

AiWXX: About 250 person hours.

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

SMADAS-UFSC: We did not share our strategy in advance. However, we participated in all test matches provided by the contest's organization.

GOAL-DTU: We discussed our strategies with the creators of the Python-DTU system and also tested against that system.

LTI-USP: 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.

TUB: During the course we split the students into two teams that compared their solutions every week, both in discussion and in simulation. We also used the test games provided by the organizers of the contest, where we mainly tested for reliability and conformance.

AiWXX: During this period, we did not discuss the design and strategies of our agents with others because this year, we were focusing on the cooperation of agents and not on the competition with different strategies.

What data structures are shared among the agents, and which are private?

SMADAS-UFSC: Our agents share information about the graph structure, the enemy position and inspected agents. Information about health, energy, zones and others is private for each agent.

GOAL-DTU: No data structures are shared among the agents.

LTI-USP: Only the organisational artifacts are shared among the agents. Each agent has its own world model.

TUB: Each agent maintains its own world model, i.e., it updates the world model with perceptions and action results, then calculates the next step and remembers the decision. Perceptions and decisions are shared among all other agents and each agent also maintains the state of each other agent in

its own world model. The exception is again the agent that calculates the best zone. It also knows the perceptions and decisions of all other agents but it is the only agent that knows the best zone. All other agents only know their position in the best zone.

AiWXX: Each agent has only one private data structure, which stores the perceptions received from the server.

3.3 Software Architecture

Here we are interested in the specific approach. Which agent platform or programming language was used? How were agent-related concepts implemented? Which tools, which algorithms were used? How is the reasoning of an agent realized? What were the hardest problems and how many lines of code were written?

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

SMADAS-UFSC: Our Multi-Agent System is developed in JaCaMo platform, using Jason, CArtAgO and $\mathcal{M}$oise.

GOAL-DTU: We used the GOAL agent programming language to implement the multi-agent system. As a knowledge representation language we used SWI-Prolog.

LTI-USP: Java and AgentSpeak.

TUB: Java.

AiWXX: The C++ language.

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?

SMADAS-UFSC: We used an environment and organizational multi-agent framework, which provides abstractions to develop specific agent-oriented concepts, environmental artifacts and organizational rules.

GOAL-DTU: We used the inherent architecture in the GOAL language since it is a dedicated agent programming language.

LTI-USP: 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.

TUB: Functionality in JIAC is implemented in components (AgentBeans), so each function is an AgentBean: e.g. the communication with the contest server (ServerCommunicationBean), each role is implemented as a different Strategy managed by the DecisionAgentBean, the game concepts are reflected by the ontology where we mapped them to Java classes.

AiWXX: Each of the agents runs a separate program which is designed at four different levels, from the coordination level to the physical level.

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

SMADAS-UFSC: We used Eclipse platform with Jason 1.3.8 plug-in. These tools were known by all team members. So we spent just a few hours learning new features.

GOAL-DTU: We use the GOAL IDE as a development platform as well as Eclipse as a code editor/IDE. We were already familiar with the platforms from earlier projects.

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

TUB: We used the JIAC V framework. The frame implementation was given by the course organizer, the rest was implemented by the students starting from a message parser. We additionally used a Swarming approach for the visual reconstruction of the graph, developed by our colleague Tobias Küster, as the MarsViewer maps the graph to a grid, but this information is not available to the agents.

AiWXX: Just Gedit Text Editor. We invested little time in learning it.

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

SMADAS-UFSC: We used EISMASSim framework to communicate with the environment, Jason centralized infrastructure for communication among the agents and ORA4MAS, a CArtAgO and $\mathcal{M}$oise based platform.

GOAL-DTU: We used Linux running the newest version of GOAL from the GOAL SVN repository as the runtime platform for the competition.

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

TUB: We used the JIAC runtime platform, which usage is fairly easy and straight forward. The platform manages the life cycle of the agents and the communication infrastructure. With the ASGARD agent management tool we can remotely control the life cycle and state of each agent.

AiWXX: Just the GCC compiler. We invested little time in learning it.

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

SMADAS-UFSC: The JaCaMo framework provided most of the features needed. To build graph algorithms we used Java because it is a powerful language and it is quite simple to integrate with JaCaMo.

GOAL-DTU: Even though GOAL has debugging features these were not fully functional at the time of the contest. For this reason we developed our own debugging tools.

LTI-USP: The JaCaMo framework provided all the necessary features that we needed to develop our team.

TUB: We still miss an easy agent language at all, our approach to JADL++ was to combine powerful features of a logic language with C-like surface syntax, which is not finished yet. A second point is the BDI decision cycle which is implemented in the framework but is not been used and tested, although we see every year that the decision component's implementation always produces similar solutions, so a generalizing concept is overdue.

AiWXX: We have implemented all proposed features efficiently due to the flexibility of the C++ language.

Which algorithms are used/implemented?

SMADAS-UFSC: We implemented some graph algorithms like Dijkstra, breadth-first search and identification of cut vertices.

GOAL-DTU: We use our own implementation of the A* algorithm for pathfinding, but since there is no usable heuristic this is basically Dijkstra's algorithm.

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

TUB: Path finding is based on Dijkstra, breadth-first search for finding agents and other targets from a given node.

AiWXX: The breadth-first search, the dijkstra algorithm, the minimum cost flow algorithm and the hungarian algorithm.

How did you distribute the agents on several machines? If not why?

SMADAS-UFSC: We did not distribute the agents on several machines. Our agents run fast enough on a single machine for the contest.

GOAL-DTU: We did not distribute our agents on several machines since this feature was not fully functional in GOAL.

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

TUB: No, we didn't. There was no need to. During the contest we used a multi-core machine with huge RAM connected to a GigaBit switch.

AiWXX: We did not distribute the agents on several machines because sharing memory in one computer is almost the most efficient way for communication between agents.

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?

SMADAS-UFSC: While waiting for the server, our agents reason about some information which is not used to perform an action, like the good zones definition, graph synchronization, repairer allocation, etc.

GOAL-DTU: Our agents do not perform any reasoning while waiting for the server since they are synchronized with the receive-percepts/send-action cycle.

LTI-USP: 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

reasoning agent is completely synchronized with the receive-percepts/send-action cycle.

TUB: Almost all agent are synchronized to the server cycle, only the agent that calculates the best zone is doing this all the time.

AiWXX: If an agent receives a new percept, any other agent will perform no actions until this percept is updated to the knowledge base.

What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?

SMADAS-UFSC: The most difficult part was to decide which strategy to use for the contest. We implemented several strategies and tested each one a lot. As we used $\mathcal{M}$oise and CArtAgO technologies, we also found issues to improve on this technologies.

GOAL-DTU: The most difficult/time consuming part of development was fixing bugs in both the GOAL system and our own code.

LTI-USP: Due to the modularity of the JaCaMo framework, it was not complicated to change our team for this year's 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.

TUB: the most challenging part is to predict the behaviour of the enemy team and then to derive the best strategy against it, e.g. in last years discussion many participants believed holding a huge area was key to success, when analysing the winners matches we realized that they maintained many small zones only held by two agents on high-weighted nodes.

AiWXX: The most difficult part of the whole development process was the optimization of team strategies against different strategies.

How many lines of code did you write for your software?

SMADAS-UFSC: We used 8459 lines to implement our team: 3794 for Jason agents; 135 for $\mathcal{M}$oise organization; 96 for CArtAgO environment and 4434 lines in Java.

GOAL-DTU: We wrote 1288 lines of code (not counting comments and blank lines).

LTI-USP: Approximately 2000 lines in Java and 1800 lines in AgentSpeak.

TUB: About 7996 lines.

AiWXX: About 11,000 lines.

3.4 Strategies, Details and Statistics

The questions in this part are related to the particular approach used by each team. How are the roles of the agents implemented, which strategies do they follow? How are zones computed or conquered and defended? Is the buying-mechanism considered important? Are achievements? Does the agent behaviour emerge on an individual or the team level?

What is the main strategy of your team?

SMADAS-UFSC: Our main strategy is to acquire achievement points and define good zones as soon as possible. After that, we spread the agents in the map and keep the agents in their places until the end of the game. We also use the saboteurs to disturb the enemy and the repairers to help disabled agents.

GOAL-DTU: The main strategy is as follows. In the first part of a simulation the agents explore the map to find the most valuable nodes. In the second part our agents establish a zone of control on the most valuable clusters of nodes. Meanwhile, our *Saboteurs* defend our zone as well as harass the enemy to disrupt their zones.

LTI-USP: The main strategy was to divide 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.

TUB: The main strategy of our team is twofold: First, individual agents follow a simple, straightforward achievement collection strategy based on their roles. And, second, there is an additional agent that calculates the best zone that is free of enemies.

AiWXX: The whole team explores the entire map for available areas and then tries to occupy several stable zones with higher values.

How does the overall team work together (coordination, information sharing, ...)?

SMADAS-UFSC: We use an explicit organizational structure to coordinate the agents. It defines the role for each agent and the goals they have to achieve. In addition, we use an artifact where the information about the graph structure is shared.

GOAL-DTU: Our agents work together by coordinating their behavior when they have to establish a zone of control. They also communicate the status of themselves and enemy agents as well as map information.

LTI-USP: One agent is responsible for determining which are the best zones in the map. 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 produce an update of the agent's world model are broadcasted.

TUB: The basis of the team work is information sharing: perceptions and decisions are communicated among all agents. There are simple conventions on how a broken agent finds the repairer. The calculation of the best zone is done by an extra agent on behalf of all interested agents (that are free to zone).

AiWXX: We allocate to each agent a unique task. When any agent receives a new percept, any other agent will not perform any actions until this percept is passed to all of them. This ensures that all agents share a synchronized knowledge base.

How do your agents analyze the topology of the map? And how do they exploit their findings?

SMADAS-UFSC: We do not try to find a map topology. However, we identify the cut vertices, which usually represent good zones that can be conquered by a single agent.

GOAL-DTU: Our agents share information about probed and surveyed nodes with each other. This way they know the structure of the whole map and can find their way around more easily.

LTI-USP: The explorers probe all unknown vertices and the results of the map analysis are exploited to find the best zones to be occupied.

TUB: Our first approach was to use a heat map to calculate the best zones. The current implementation calculates about 50 to 100 paths, where the best one is selected that separates the biggest enemy free zone from the rest of the map depending of the number of free agents.

AiWXX: Each time an agent arrives at an unexplored location, it collects all information about edges and nodes. Her strategy then is to move to those nodes on the boundary, survey them and repeat this process again and again.

How do your agents communicate with the server?

SMADAS-UFSC: We used the EISMASSim libraries to communicate with the server.

GOAL-DTU: Our agents communicate with the server using the provided EISMASSim library (version 2.1).

LTI-USP: Using the EISMASSim interface.

TUB: Via IP sockets.

AiWXX: The agents generate multiple threads and use the TCP/IP protocol to communicate with the server.

How do you implement the roles of the agents? Which strategies do the different roles implement?

SMADAS-UFSC: Explorers are responsible to probe all vertices and they define which is a good place to conquer. Saboteurs are responsible to protect the zones and to attack enemies. Repairers are responsible to help damaged agents. Inspectors are responsible to protect the best places and inspect the enemies. Sentinels are responsible to protect the best places and the whole team is responsible to survey the map.

GOAL-DTU: All agents share the same basic behavior. Depending on the role given to them by the server, they access a part of the code that is specific to their role. The agents implement strategies relevant to their role.

LTI-USP: We decided to not map the five types specified in the scenario (explorer, inspector, repairer, saboteur and sentinel) directly 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 types 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.

TUB: A role has at least one strategy, if it has more then one strategy role switching is possible dynamically.

Explorer Find unprobed nodes and probe them, recharge when necessary go to the closed unprobed node first. Avoid other Explorers from our team to avoid duplicated behaviour. Seek repair if health is zero. Switch Strategy if the graph has been probed completely.

Repairer There are two strategies for the repairer.

Simple Repairer Search for hurt team members and go to the closest one to heal. If no team member is hurt survey and zone.

Craven Repairer Same as the Simple Repairer but avoid enemy attackers at all cost. Using a breath first search and a range that repairer would search for a enemy attacker and if it would find one it would create a path to avoid that agent.

Sentinel used for Zoning see Zoner

AggressiveSaboteur As the name suggests the Saboteur tries to attack as many agents as possible.

ZoneDefender This Agent is part of the Zone and upon a detected intrusion into the Zone, this agent tries to disable the intruder.

AnnoyInspector This strategy tries to find enemy zones and plans paths trough those zones to destroy them while inspecting the enemy.

ZoneInspector The inspector is part of the zone and if an intruder is detected it tries to inspect the intruder.

Zoner The Zone is a strategy which will build a Zone using the zoning algorithm to identify a good Zone and coordinates all Zoner to build that zone.

AiWXX: The agent considers her role as a state. We have designed a particular strategy for each of the five roles in the game. When an agent realizes that it is acting in a certain role, say, repairer, it will follow the respective strategy. Only explorers accept the mission of exploring the map and probe the value of the newly encountered node. Only sentinels, inspectors and explorers will occupy the zones. Repairers will run to the injured and repair their teammates while saboteurs will go to the front line and fight with enemies.

How do you find good zones? How do you estimate the value of zones?

SMADAS-UFSC: The good zones are defined in terms of *hills*, *pivots* and *islands*. A hill is a zone formed by several vertices that have a good value and are in the same region of the map. As in the 2012 team, the agents try to discover two hills. The hills are defined as follows: for each vertex v of the graph, the algorithm sums the values of all vertices up to two hops away from v, including v. The two vertices with the highest sums are defined as the center of the hills. Then, the agents try to stay on their neighborhoods. Islands are regions of the map that can be conquered by a single agent. An island is a zone that has only one vertex (a *cut vertex*) in common with the remaining graph. They are found by disconnecting the edges of each cut vertex of the graph. It produces two disconnected subgraphs, and the smallest one, plus the cut vertex, is an island. Pivots are regions of the map that can be conquered by just two agents. For each pair of vertices (u,v) we search all vertices w connected to u and v. For all vertices w (including u and v)

we also search for all vertices only connected to these vertices. For example, if there is a vertice k connected only to the vertice w, then k also belongs to the pivot. Furthermore, if there is an island connected to some of these vertices, we consider all the vertices of the island. The best pivots are chosen considering the sum of all vertices.

GOAL-DTU: Our agents find several of the most valuable nodes on the map, and they calculate the total value of the area around each of those nodes.

LTI-USP: The best zone is obtained by calculating for each vertex the sum of its value with the value of all of 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.

TUB: see above.

AiWXX: First, we will choose each node as a point zone with no enemies standing at and then repeat the following: find the boundary node P such that after expanding P the boundary increases the least (possibly by a negative number), and then, expand it. During the expanding process, we maintain the optimal zone ever found.

How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

SMADAS-UFSC: The agents which control islands do the following: if there are enemies in the island, they go to the same vertex as the enemy, so both teams do not get scores from that island. In addition, they call the saboteur leader to fight against the enemy agent. If the saboteur leader is already busy protecting another island, the saboteur leader calls the saboteur helper of the group special operations. If both are busy, the saboteur leader keeps a list of islands with enemies. The agents that control pivots do not move away from their places, since most of the times, the enemy would not stay in the same place. Therefore, we defined that these agents do not need to move. If the enemy also continues in the same vertex, both teams do not get scores, so our team also cancels the enemy strategy. The agents which control the hills are simply moving to the border of the big zone in order to expand it. If they break the zone, they come back to the previous places in order to try to expand it again. We also defined the sentinels to stay in the same places all the time in the big zones (hills) because it can make the enemies avoid those places and we can get some fixed scores of the hills. Finally, our saboteurs disturb the enemy all the time.

GOAL-DTU: The agents will move towards the most valuable zones regardless of any enemy presence. Our *Saboteurs* engage enemy *Saboteurs* that get near our zones. We try to find enemy zones and attack them with one dedicated *Saboteur*.

LTI-USP: Given that the coordinator has assigned a zone for a group, all agents of the group move 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, with the `guardian` agent ready to attack any close opponent, and the `medic` in the center of the zone focusing

on repairing other agents.
We also developed a group to attack the opponent's zone.

TUB: the main strategy here is to avoid enemy agents while zoning, only the AnnoyInspector is trying to destroy enemy zones, the only approach to defence at the moment is inspecting the intruding agent.

AiWXX: Our agents will compute several promising zones and then move to the boundary and conquer it. Among them, the saboteurs will attack enemies they found because this may attack the area occupied by the rival.

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

SMADAS-UFSC: The agents change their behavior in some pre-defined steps. For instance, in step 7 the agents start to search for a good zone in order to get as much achievement points as possible. In the step 130 they look for the smallest ones. When all vertices are probed, all the agents start to participate in conquering pivots and islands.

GOAL-DTU: Several of our agents change behavior by adopting new goals during runtime. These changes can be triggered by reaching a specific step, enemy behavior, disabled agents etc.

LTI-USP: 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.

TUB: Yes, role changing is possible, among all agents that have a server identity. They are configured as such that they can play all roles. But as the setting is static according to the assigned role by the server it is not yet necessary. Where we already use it is when switching between different role implementations.

AiWXX: Yes. We set some random target to change their behaviors with a relatively small probability at each step.

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

SMADAS-UFSC: We used the Dijkstra algorithm to find the shortest path between two vertices.

GOAL-DTU: We use an A* algorithm without heuristic for agent path planning.

LTI-USP: Breadth-first search algorithm.

TUB: Dijkstra.

AiWXX: Breadth-First Search Algorithm and our own algorithm mentioned in the paper.

How do you make use of the buying-mechanism?

SMADAS-UFSC: We did not use the buying-mechanism.

GOAL-DTU: We buy strength and health for our *Saboteurs* when necessary.

LTI-USP: We decided to limit the buy action, allowing only the agents of type saboteur and repairer to purchase a unique extension pack of `sensors`, in order to be able to attack or repair agents in neighbouring vertices.

TUB: Not at all (this aspect was given a lower priority during development).

AiWXX: Our agents will not buy anything.

How important are achievements for your overall strategy?

SMADAS-UFSC: The achievements are important. We try to get most achievements as soon as possible, since they accumulate in each step. However, we guess the achievements did not make the difference for our team in this year.

GOAL-DTU: Achievements are fairly important for our strategy, since we need achievement points for our *Saboteur* buying strategy.

LTI-USP: The achievements were very important in the team score, which is why we limited the buy action.

TUB: Not at all at the moment.

AiWXX: In the contest, we use achievements only for the score. So we save them and do not buy anything.

Do your agents have an explicit mental state?

SMADAS-UFSC: Yes, our agents use a BDI architecture and use their beliefs to decide on their actions.

GOAL-DTU: Our agents have explicit mental states.

LTI-USP: The agents' mental states consist of internal beliefs, desires, intentions, and plans.

TUB: In a sense, yes. The agents maintain their own world model on the basis what they perceive and what they are told by other agents. The model consists of the own status, the environment, and the own intention. To a small extend they can predict what other agents are going to do based on the role and their possibilities.

AiWXX: No.

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

SMADAS-UFSC: The agents communicate by message exchange and a blackboard. They use message exchange to call a repairer or a saboteur, to inform about the good zones and vertices, to exchange information about probed vertices and to communicate their current action (which prevents two agents from performing the same action). The blackboard is used to share information about the graph structure and the agents' positions.

GOAL-DTU: Our agents communicate using the built-in messaging system in GOAL. They communicate agent status and map information.

LTI-USP: 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.

TUB: Agents share their perception and the next action using multicast communication.

AiWXX: The agents communicate by sharing memory. They communicate about the map, enemies, status of teammates and proposals of team work missions.

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

SMADAS-UFSC: To organize our agents we use a $\mathcal{M}$oise organization model. Also, the agents follow a hierarchy, since we defined a leader for each agent kind and an overall leader.

GOAL-DTU: We do not organize our agents. We sometimes perform actions based on a ranking system in order to prevent that several agents perform the same action with the same target.

LTI-USP: We used the $\mathcal{M}$oise 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.

TUB: Organization is implicit as there is no explicit organisational concept except for roles. A role is the function that the agent plays in the MAS.

AiWXX: They share the same knowledge base at any time and act by themselves. They are all at the same status. Hence, we organize our agents explicitly and no hierarchy is exploited.

Is most of your agents' behavior emergent on an individual or team level?

SMADAS-UFSC: A team behavior is important for our agents' strategy. Thus, our agents' behavior is mostly on the team level.

GOAL-DTU: The behavior of the *Saboteurs* and *Repairers* is mostly emergent on an individual level. For the rest of the team it is on a team level.

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

TUB: No, mainly. We saw clotting as negative effect of the repairers' and saboteurs' behaviour.

AiWXX: The behavior of our agents is mostly emergent on an individual level because we think autonomy is the key idea of MAS.

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

SMADAS-UFSC: The agents do not plan in advance. Since the environment is dynamic, the agents choose their action in each step.

GOAL-DTU: Our agents do not plan ahead.

LTI-USP: Our agents do not plan ahead.

TUB: A planned path holds path length steps unless something went wrong, e.g. action fails or health status is down.

AiWXX: The agents do not perform any planning because we think that the current planning technology is not efficient enough.

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

SMADAS-UFSC: We used EISMASSim to communicate with the server. After getting the percepts, the agents reason about it and decide what action to do. Both percepts and actions are performed by EISMASSim.

GOAL-DTU: Our perceive-think-act cycle is synchronized with the server by preventing agents from performing more than one action each step.

LTI-USP: After sending an action, the agent waits until it receives new percepts from the server and then starts a new perceive-think-act cycle.

TUB: The cycle must complete within the server cycle.

AiWXX: To synchronize with the server, agents listen to the message from the server and the respective agent will decide which action to perform. Furthermore, they will send the action to the server. Our program is so efficient that any agent is always able to send its action to the server before the next percept arrives.

3.5 And the Moral of it is ...

Finally, it is important to find out lessons learned. What are positive and negative experiences of the particular approach? Given the performance in this contest, critically evaluate your team. What was good, what was bad in the current scenario? How can it be improved.

What have you learned from the participation in the contest?

SMADAS-UFSC: We learned more from MAS development and about the technologies we used, like $\mathcal{M}$oise and CArtAgO. Also, the contest allowed us to evaluate and improve these technologies.

GOAL-DTU: From the participation in the contest we gained further experience with the GOAL agent-programming language.

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

TUB: First, to have a simple implementation of each relevant aspect in the contest that works reliable, then improve it. Second, to make more and smaller zones: they are easier to form and to maintain.

AiWXX: The participation in this contest has greatly improved our knowledge of multi-agent systems and stimulated our interest in conducting research in this area.

Which are the strong and weak points of the team?

SMADAS-UFSC: The strategy to get many small zones was the strongest point of our team and it turned out to be hard for the opponents to disturb our zones since our agents were spread over the whole map while our saboteurs were able to disturb the opponent zones. However, our team can be improved to perform better in maps where there are too many good vertices gathered in the same place. In that case, the best strategy seems to be to build a big zone and defend it instead of building just small zones.

GOAL-DTU: One of the strong points of the team is our ability to control a zone. Another is our preemptive repairing; our *Repairers* anticipate attacks on our *Saboteurs*. One of the weak points is the harass strategy for our *Saboteurs* because of unresolved bugs.

LTI-USP: 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.

TUB: The strong point is the easy exchange between strategies that makes an easier and faster development cycle, the weakest point was the one best zone strategy.

AiWXX: One strong point of our team is that it only costs about 0.2 seconds to make all decisions, on a 500-node map, in a perfect network. Our framework is compatible enough to develop more complex strategies in future contests. The weaknesses of our team are that we do not observe the enemy and we are not familiar with the strategies of the other teams.

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

SMADAS-UFSC: All the technologies we used are suitable to MAS development. However, during tests we created some new features to improve these technologies.

GOAL-DTU: The GOAL system performed excellent during the contest and we only lost to USFC who used better algorithms. However, we encountered a number of bugs and other issues in GOAL during the development of our system.

LTI-USP: 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.

TUB: The easy thing is that the student do not need to know about agent theory, framework implementation details to solve the contest problem. The agent metaphor is intuitive as such and the framework delivers the implementation so the student developers can concentrate on the domain specific parts.

AiWXX: Our framework can support almost every strategy we can imagine. The C++ language we used is suitable to MAPC, because we are proficient in this language which is well-known for its efficiency and flexibility, supporting various data structures and algorithms.

What can be improved in the contest for next year?

SMADAS-UFSC: The contest scenario should be released earlier and new features should not be made after the release. In this contest, the scenario changed (the thinning was added) after the first releases and it made us change our strategies before the contest.

GOAL-DTU: The contest could be improved by presenting the description of the specific scenario and the requirements for the contest as early in the year as possible. Furthermore, test-matches should not be 750 steps but rather 300-400 steps.

LTI-USP: 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.

TUB: Dynamic role assignment and role switching to a certain extend, in principle, nearly every human can do nearly every job, and one can use a pencil as a weapon, too.

AiWXX: Next year, we are going to observe the enemy and analyze the strategy of the other teams.

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

SMADAS-UFSC: Our team performed very well, except in maps with low thinning (below 20%) with most of the good vertices located in the same regions. The other teams performed worse most of times because they tried to conquer big zones, which are harder to protect.

GOAL-DTU: We believe that our team had strong strategies for zone control, *Saboteurs*, and *Repairers*. Some of the other teams did not have as stable zone control strategies as we did, and performed worse. UFSC, who won, had a very strong zone control strategy.

LTI-USP: 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.

TUB: We build the team from scratch, although there was an contest implementation of last year. Time is always a limiting factor, i.e. the overall time to the contest and also the time that is then used for the implementation and testing. We did not know where we are standing. And we also had an avoidable bug in the first match, we should have won this.

AiWXX: The performance this year was not so satisfactory and there are many reasons. Our VPS was down during the final contest and we did not have enough time to implement all the ideas.

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

SMADAS-UFSC: Machine learning is one interesting field to improve our next team.

GOAL-DTU: Other research fields such as algorithms, logic, game theory and AI might be interested in the contest.

LTI-USP: Algorithms, Game development, Game theory, AI, Robotics.

TUB: Even within our institution people always produce central server solutions in the following fields: information retrieval, HCI, home control.

AiWXX: Distributed algorithms, Game Theory.

How can the current scenario be optimized? How would those optimizations pay off?

SMADAS-UFSC: The ranged actions should be revised in order to balance the fail probability.

GOAL-DTU: The current scenario could be optimized by making upgrades more viable. Furthermore, ranged actions should have fewer drawbacks.

LTI-USP: 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 have a huge chance

to fail and it is not worth it to use them. Another idea is to change the score computation to consider only the zones' values. This way, the buying strategy will not directly impact the team score and it will be interesting to see how each team will invest their achievement points.

TUB: The very interesting part could be how can agents from different teams work *together*? Could be interesting for the interoperability part of different frameworks, toolkits, languages and libraries.

AiWXX: The perception should be compressed so as to relieve the pressure of network communication. And the organizers should offer VPS for the participants.

4 Conclusion

In this paper we have tried to put together detailed information about how the participants of this years agent contest approached the Mars scenario. We did this through a series of concrete questions and requested brief answers. By listing for each question the answers of all teams one after another, we get a good comparison of the similarities and distinctions of the individual systems. We believe this information is helpful not only for future participants of the contest, but also for other people who are interested to apply multi-agent technology to similar problems.

References

1. Tobias Ahlbrecht, Michael Köster, Federico Schlesinger, and Jürgen Dix. Multi-Agent Programming Contest 2013. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 296–323. Springer, 2013.
2. Tristan Behrens, Mehdi Dastani, Jürgen Dix, Jomi Hübner, Michael Köster, Peter Novák, and Federico Schlesinger. The multi-agent programming contest. *AI Magazine*, 33(4):111–113, 2012.
3. Mariana Ramos Franco and Jaime Simão Sichman. Improving the LTI-USP Team: A New JaCaMo based MAS for the MAPC 2013. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 344–353. Springer, 2013.
4. Koen Hindriks and Jürgen Dix. Goal: A multi-agent programming language applied to an exploration game. In Onn Shehory and Arnon Sturm, editors, *Research Directions Agent-Oriented Software Engineering*, pages 112–137. Springer, 2013.
5. Chengqian Li and Lu Liu. Prior State Reasoning in Multi-agent systems and Graph-Theoretical Algorithms. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 362–371. Springer, 2013.
6. Jørgen Villadsen, Andreas Schmidt Jensen, Nicolai Christian Christensen, Andreas Viktor Hess, Jannick Boese Johnsen, Øyvind Grønland Woller, and Philip Bratt Ørum. Engineering a Multi-Agent System in GOAL. In Michael

Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 334–343. Springer, 2013.

7. Sebastian Werner, Christian Bender-Saebelkampf, Hendrik Heller, and Axel Heßler. Multi-Agent Programming Contest 2013: TUB Team Description. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 354–361. Springer, 2013.
8. Maicon R. Zatelli, Maiquel de Brito, Tiago L. Schmitz, Marcelo M. Morato, Kaio S. de Souza, Daniela M. Uez, and Jomi F. Hübner. SMADAS: A Team for MAPC Considering the Organization and the Environment as First-class Abstractions. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 324–333. Springer, 2013.