SMADAS: A Team for MAPC Considering the Organization and the Environment as First-class Abstractions*

Maicon R. Zatelli, Maiquel de Brito, Tiago L. Schmitz,
Marcelo M. Morato, Kaio S. de Souza, Daniela M. Uez, and Jomi F. Hübner

Department of Automation and Systems Engineering
Federal University of Santa Catarina
CP 476, 88040-900 Florianópolis - SC - Brasil
{xsplyter,tiagolschmitz,marcelomenezes73,dani.uez}@gmail.com,
maiquel.b@posgrad.ufsc.br,kaiossouza@hotmail.com,jomi.hubner@ufsc.br

Abstract. This paper describes the SMADAS team for the Multi-Agent Programming Contest 2013. Throughout this paper we highlight the design, main strategies, tools, and results of our team. For this year we used the JaCaMo platform to develop the team, which is composed of Jason (to program the agents), CArtAgO (to program the environment), and $\mathcal{M}$oise (to program the organization). We also improved the last year team with new strategies focused on the updated "Agents on Mars" scenario.

1 Introduction

The Multi-Agent Programming Contest (MAPC) [6][1] is an important event to stimulate research in the multi-agent systems programming field. The MAPC 2013 used the "Agents on Mars" scenario, which was improved from the last year scenario, therefore the efforts must continue concentrated in cooperation, coordination, and decentralization. Our agent team, called SMADAS, acronym for our research group, named Multi-Agent Systems from Systems and Automation Department (in Portuguese, **S**istemas **M**ulti**a**gentes do **D**epartamento de **A**utomação e **S**istemas) was developed by a group formed by one PhD, four PhD students, and two undergraduate students from the Federal University of Santa Catarina (UFSC). This is our second participation in the contest and we have two main aims this year: improve our MAS developing skills and evaluate some proposals developed in our thesis.

2 System Analysis and Design

For this year's contest, we opted for developing a new team using the JaCaMo [2] platform instead of just improving the last year team. The programming model of JaCaMo

* We are grateful for the support given by CAPES and CNPq (grant numbers 140261/2013-3, 306301/2012-1).

[1] http://multiagentcontest.org

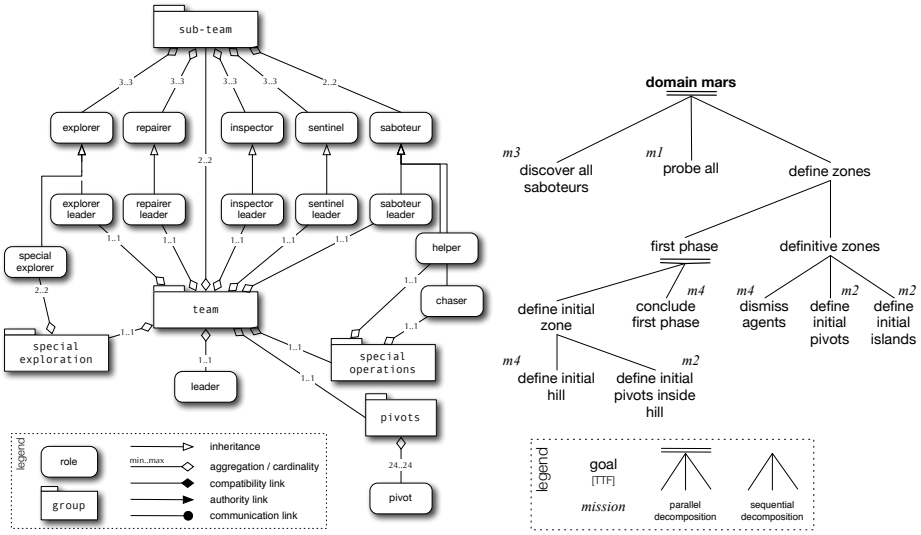
provides high-level support for developing MAS considering agents, environment, and organization as *first-class* entities. The development of these three dimensions is based on three different technologies: Jason [3], for programming the agents; CArtAgO [7], for programming the environment; and $\mathcal{M}$oise [5] for programming the organization. Thus, this year the organization and environment, which were previously implemented as part of the agent code, have now been programmed with proper organizational and environmental elements according to the aforementioned models and technologies.

2.1 Organizational Dimension

As JaCaMo supports organizational programming, part of the coordination of team agents is modeled in the organizational dimension instead of being modeled as skills of the agents. The organization provides guidelines for the achievement of the overall system goals, but the agents remain autonomous to decide how to achieve them. For example, the organization informs that an agent is obligated to probe the vertices, but the agent is autonomous about "how" to do it, based on its local knowledge about the world. However, the autonomy can be constrained by means of organizational norms.

Fig. 1(a) shows the structural specification (SS) of the team using the $\mathcal{M}$oise notation. Notice that the SS is designed based on the roles of the contest scenario. The team is divided into two *sub-teams*. Besides, the team has three minor subgroups: *special operations*, *special exploration*, and *pivots*. An agent can play more than one role at the same time. For example, an *explorer* can also play *explorer leader* and *special explorer* roles. One agent plays the role *leader* and is responsible to manage the overall organization (e.g. designating the roles of the other agents). The functional specification (FS) (Fig. 1(b))) specifies the goals (i.e. specific states of affairs) that the agents must achieve and distributes these goals to the agents (by means of missions). The overall goal (*domain mars*) consists in a decomposition tree where the leaves are the goals that can be achieved by the agents. The goals are grouped in four *missions* ($m1$, $m2$, $m3$, and $m4$). Finally, the normative specification (NS) (Fig. 1(c)) relates roles to missions. For example, as the *explorer leader* is obligated to commit to mission $m4$, it is obligated to achieve the goals *define initial hill*, *conclude first phase*, and *dismiss agent*.

Along the development of the team, we performed some experiments introducing *count-as rules* [4]. Count-as rules changes in the organization as the result of facts occurring in the environment. For example, without count-as rules, a specific agent has to set up the organizational infrastructure and explicitly adopt the role of *leader*. With count-as rules, it is possible to state that such setup *counts-as* the adoption of the *leader* role. In another example, without count-as rules, the agents have to reason about the organizational structure, checking their roles, and committing to missions according to that roles. With count-as rules, it is possible to model that playing of a specific role *counts-as* the commitment to a specific mission. The use of count-as rules simplifies the reasoning and action of the agents, as they do not need to perform some actions on the organization (e.g. they do not need to reason and to act to commit to missions). Besides, count-as rules contribute to keep the organization in a consistent state as some organizational actions do not depend on the agents actions. Due to time constraints, the count-as rules were not added to the tournament team. However, the experiments indicate that the rules seem a suitable approach for further versions of the team.



(a) Structural specification (b) Functional specification

norm	*role*		*mission*
n1	explorer	obligation	m1
n2	sentinelLeader	obligation	m2
n3	inspector	obligation	m3
n4	explorerLeader	obligation	m4

(c) Normative specification

Fig. 1. Organizational Specification

2.2 Environment Dimension

The environment for our agents has two parts. The first part provides integration with the contest simulation by means of EISMASSim framework [1] and is well defined in the contest documentation. The second part is provided by JaCaMo artifacts that agents perceive and use to achieve their goals. In our team, the information about the inspected enemies is managed by an artifact. We also conceived an artifact responsible to manage all the graph structure. However, since we used the same graph structure and algorithms of the last year, which were based on Java shared memory, we decided to not move this previous implementation into a specific CArtAgO artifact because it would require more time. Therefore, part of the environment is developed in CArtAgO and another part was kept in "pure Java", accessed by means of Jason internal actions, as we did last year. It is a future work to unify all perceptions and actions under the CArtAgO approach.

2.3 Agent Dimension

The agents may behave proactively or reactively, in accordance with the needs. For example, a damaged agent will proactively call a repairer and all agents promptly react to the environment events, like the start of a new simulation step.

Agents share information by two mechanisms: messages and blackboards. Since it was not so appropriate to broadcast everything between the agents because we would

have 28×27 messages[2], we chose to send messages about few things. For example, when agents are disabled, they call a repairer; when enemies invade some team area, the saboteurs are notified; and when some vertex is probed, the explorers broadcast the value of the vertex. Some information is also important to exchange between agents of the same kind to avoid them performing the same action. For example, when there are two saboteurs with enemies in the same vertex, they need to communicate to decide which enemy each one will attack. We used the same solution for explorers and repairers to avoid repairing the same agent and to avoid probing the same vertex. Messages are also used to inform agents about the zones they should help to conquer.

The second (indirect) communication mechanism is the use of blackboards and artifacts as commented on Sec. 2.2. In this case, the agents share the graph structure, the information about the inspected agents, and the position of the enemies and team mates. Finally, the remaining data, such as their own health, energy, zone scores, or visible vertices, is private for each agent.

We defined a priority among the agents to avoid conflicting actions (like two agents probing the same vertice). The agent with the highest priority chooses its action first and informs the others of the same role about its decision. Then the agent with the second highest priority does the same and so forth.[3] However, actions like survey and inspect do not follow this priority approach. That means two agents can inspect or survey at the same target to try to guarantee some of them will be successful.

2.4 Testing

To develop the team we used a particular incremental process. We performed weekly meetings to define the team strategies. These strategies were implemented and tested during the week and, in the next meeting, these results were considered to define new improvements. To evaluate the team strategies and ensure the competitiveness, we tested our team by simulating a great amount (more than 1000) of matches against our 2012 team [8], the 2012 Python-DTU team, and previous versions of our current team. The aim of the tests was to evaluate the overall performance of the team in different maps, adopting different strategies and facing different strategies from the opponent. In addition, we participated in all test matches during the testing phase to evaluate the connection and a couple of strategies.

3 Software Architecture

As we done in the edition of 2012 [8], in the current edition of MAPC we used the EIS-MASSim [1] to communicate with the contest server. However, while in the previous edition the team was developed essentially using Jason, in the current edition, our team has been developed with JaCaMo platform. This was the main change in the software

[2] The team is composed of 28 agents.

[3] Notice that sharing the information about the chosen action is not enough to solve the problem. Some coordination is required to efficiently solve it. Although this coordination is an organizational issue, this priority solution was coded in the agent dimension and it remains as a future work to model it in the organizational dimension.

architecture for this year. Furthermore, even with the increase number of agents, from 20 to 28, we were still able to run the agents in a single machine, therefore we decided to avoid distributing the agents between several machines. We also made several contributions for the tools that we used in this year. In Jason, we added features to handle goals with deadlines, new mechanisms for the `.wait` internal action, and we fixed some bugs. In $\mathcal{M}$oise, we added a new feature to reset organizational goals to avoid creating new schemes at runtime and we added an organizational monitor accessed via HTTP, so that we were able to watch our team organization remotely.

The source code of the team has 3794 lines of Jason code, 135 for $\mathcal{M}$oise, 96 for CArtAgO, and 4434 for Java, totaling 8459 lines. Although the implemented strategies of these year are more complex, we can notice that the number of lines coded in Jason has decreased from 5504 in last year's team to 3794 this year. It is an expected consequence of the organization and the environment programming available in JaCaMo. Coordination strategies that previously required several lines of Jason code, could now be coded in a few lines of $\mathcal{M}$oise, since $\mathcal{M}$oise is a proper language for that. Not only have we reduced the size of the programs, but the new approach has allowed us to debug and change the organization of the team quite easily. Instead of monitoring the agents internal state, we can now monitor the state of the organization, which is a more general view of the state of the team. Since the organizational program is the same as the specification, to change the team sometimes is simply reduced to update the organization. For instance, to change the order of organizational goals, we simply need to change the scheme of Fig. 1(b).

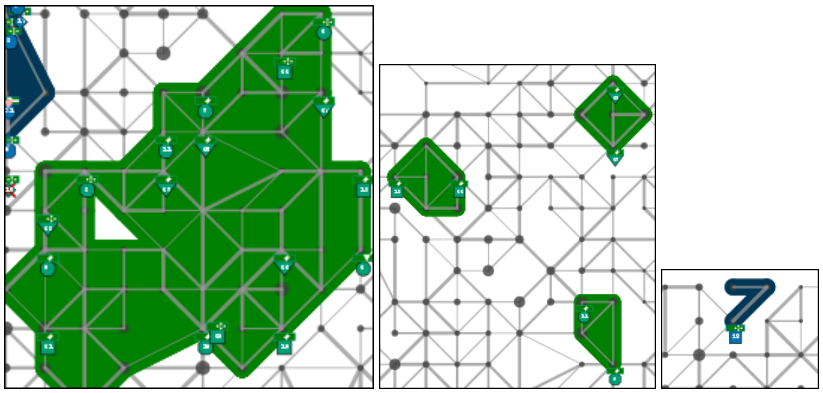
4 Strategies, Details and Statistics

In this section, we describe the main strategies of our team (Sec. 4.1) and we highlight the main results that we got (Sec. 4.2).

4.1 Team Strategies

The strategy of the team has two moments. In the former, the agents explore the map to obtain achievement points and to define good zones as soon as possible. Thus, it is possible to get a good score in the first steps. In the latter, the agents start to conquer and protect several small zones. During the whole match, the saboteurs disturb the enemy and the repairers help disabled agents.

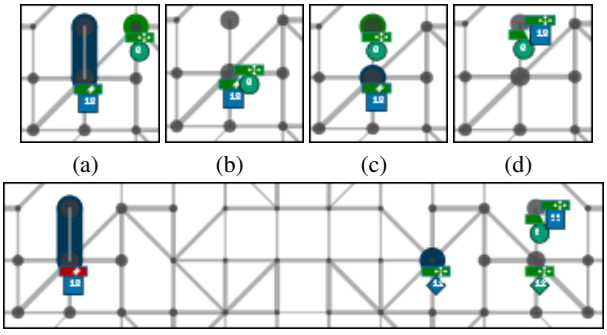
The good zones are defined in terms of *hills*, *pivots*, and *islands*. A hill (the big zone in Fig. 2(a)) is a zone formed by several vertices that have a good value and are in the same region of the map. As in the 2012 team, the agents try to discover two hills. The hills are defined as follows: for 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, and then the agents try to stay on the neighborhoods. The agents control the hills simply moving to the border of the hills in order to expand them. If they break the zone, they come back to the previous places and try to expand again. We also defined that the sentinels need to conquer the best vertices in the hills and stay over them all the time until the strategy changes. Sometimes, it



(a) Hill (b) Pivots (c) Islands

Fig. 2. Hills, pivots, and islands

induces the opponents to avoid those places and we guarantee a fixed gain of scores of the hills, even if the enemy is disturbing. Moreover, the explorers of the group *special exploration* prefer to probe first the vertices in the hills, because it increases the gain of points in the first steps.



(a) (b) (c) (d)

(e)

Fig. 3. Protecting islands

Islands (Fig. 2(c)) 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, are an island. If there are enemies on the island, the controller agent will go to the same vertex of the enemy. Thus, both teams do not get the points of that island. The figures 3(a), 3(b), 3(c), and 3(d) illustrate this situation. In addition, the controller agent notifies the saboteur leader about the invader. 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 for a further attack. Fig. 3(e) illustrates a probable call of the saboteur leader (the diamond not at the cut vertex) that is going to fight against an explorer (the circle) and a saboteur (the diamond at the cut vertex).

Pivots (Fig. 2(b)) 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 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. The agents that control pivots do not move away from their places, since most of the time the enemy does not stay fixed in those places. However, if the enemy stays in the same vertex both teams do not get the points, and so our team also cancels the enemy strategy. This is another reason to not leave the vertices. Otherwise the opponent will get the points.

The hills are defined in the first phase of our strategy, until around step 130. We chose to use hills instead of islands and pivots in the beginning of the match because most of the vertices are still unprobed and so we would not get so many points. The use of hills can keep all agents together and getting higher points because the zones are bigger. After a while the agents start conquering pivots and islands. The agents also need to decide if it is better to conquer two islands or one pivot. The decision is taken by simply summing the value of two islands and comparing with the pivots. If the two islands provide the same gain of the pivot or if they are more valuable, the agents will prefer to conquer two islands instead of conquering one pivot. The pivots and islands are very stable and so our agents are not disturbed by the enemy while our agents can disturb their zones since it is harder to protect a big zone than several small zones.

Action	Repairer	Saboteur	Explorer	Sentinel	Inspector
attack		x			
repair	x				
parry	x			x	
probe			x		
inspect					x
recharge	x	x	x	x	x
goto	x	x	x	x	x
survey	x	x	x	x	x

Table 1. Implemented strategies by agent type.

The achievements continue to be as important as in the last year. We decided do not waste money buying items and get as much achievements as possible and as soon as possible, since they accumulate in each step. We made this decision since in our tests we did not see any advantage in buying items. Finally, the specific strategies of each kind of agent are explained below while Table 1 summarizes the strategies implemented for each kind of agent.

Explorer: the explorers have an important role in the beginning of the match. They need to probe all vertices as soon as possible. To do so, the explorers avoid performing the survey action and conquering zones until they have probed all vertices. Furthermore, the *explorer leader* defines the initial two hills.

Saboteur: the main aims of the saboteurs are to protect the islands and disturb the enemy. The saboteur with the role *saboteur chaser* has the aim to attack mainly explorers, inspectors, sentinels, and repairers to avoid staying in the same vertex fighting against saboteurs all the time. The saboteurs with the roles *saboteur leader* and *saboteur helper* have the main aim to protect islands against enemies. The *saboteur leader* is also the main contact of the other agents to ask for help. In other cases, the saboteurs simply search and destroy enemies. In addition, the saboteurs attack following a priority. First of all they prefer to attack saboteurs, followed by explorers, inspectors, repairers, and sentinels. The saboteurs prefer to attack explorers and inspectors because they can not parry.

Repairer: the main aim of the repairers is to keep the agents enabled. All agents that are disabled inform the *repairer leader*. The *repairer leader* asks all the other repairers if they can help the disabled agents. If a repairer is not committed to an agent and it is not getting high points and it is enabled, then it is apt to help the other agent. All apt repairers inform the path size until the disabled agent and the *repairer leader* chooses the closest repairer to help the disabled agent. To make the repair operation faster, both the repairer agent and the disabled agent follow the same path to meet each other. If there is some repairer next to the disabled agent, that repairer will repair the agent and the agent will cancel the appointment with its repairer. Finally, when the repairers are not helping the disabled agents, they also go to the pivots or islands that they belong to.

Sentinel: the sentinels always protect the best zones, since they are harder to get disabled and they can parry. They are usually avoided by the enemy because they do not have an important role like repairers, explorers, and saboteurs. Therefore, the sentinels are not disturbed so often. The sentinels also try to survey if they find some vertex with edges with unknown value. Finally, the *sentinel leader* has the aim to define the islands and pivots and inform the agents about it.

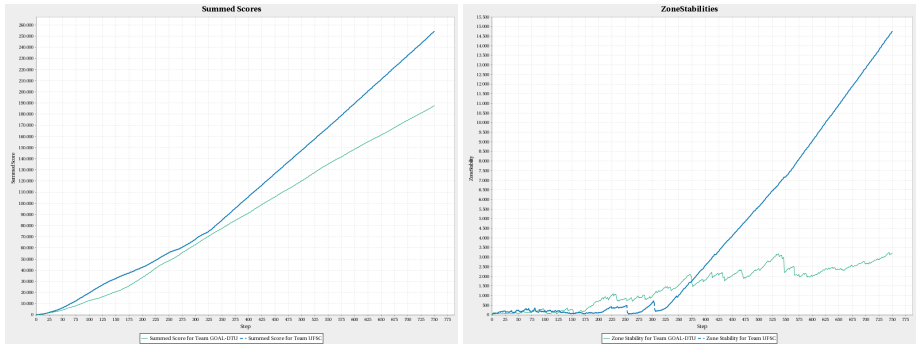
Inspector: the inspectors protect the next best zones, since they are also not so disturbed by the enemies. Their aim is to inspect all the enemy agents and they do it just once, since we do not care about what the enemy is buying. Doing so, the inspectors can stay in the same vertices until the end of the match, getting more points than by inspecting enemies.

4.2 Comparison to Other Teams

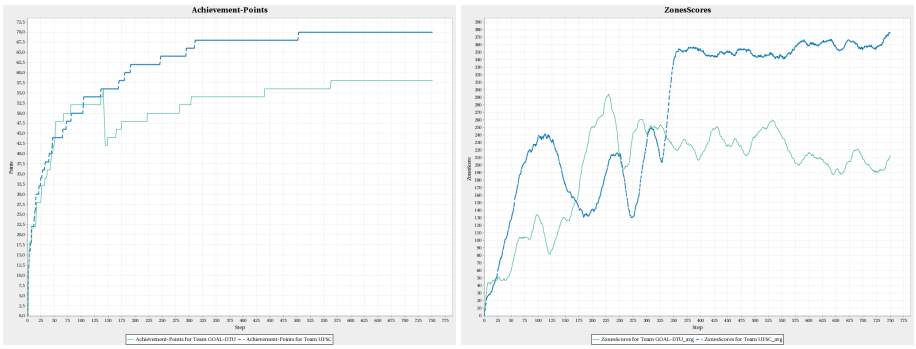
In order to highlight the main results of our strategy we chose to use some statistics of the second match against the GOAL-DTU team [4], which got the second place. Due our strategy of small zones, we can verify in the Fig. 4(b) that after step 325 our team (blue) kept almost the same zones until the end of the match. This behavior is usual in all matches against the other teams. The reason is that no matter what the opponent does, our agents will rarely leave their positions.

Another interesting result can be drawn from the zones scores plot (Fig. 4(d)). We can see that our team (blue) kept getting almost the same zone scores after step 350 and always more than the opponent. The phase of hills can also be noticed in the beginning

[4] `http://multiagentcontest.org/downloads/func-startdown/1716/`



(a) Scores (b) ZoneStabilities



(c) AchievementPoints (d) ZonesScores

Fig. 4. Statistics

of the match, between step 25 until around step 130, where the team is getting high scores because of the big zones. After step 130 our team started to conquer small zones (pivots and islands) and, therefore, spreading the agents out over the whole map. It is also possible to see that, sometimes, our zone scores were lower than the enemies. This was an expected behavior when the agents were still changing their positions because the explorers were still probing new vertices. We can see it after step 250 and after around step 315, where our team decreased the gain of zone scores. After probing all vertices (around step 325), the agents started to get higher scores because they defined the fixed zones and all agents were participating.

We can see the same behavior in Fig. 4(a), where the opponent score gets closer and then the difference of scores increases again. Notice that after step 325 the difference of scores increased continuously because all vertices were probed. On the other hand, in the Fig. 4(c) we can see that our team always has more achievement points than the opponent after step 125. It means we are getting more points because the opponent was buying items while our team was saving money.

5 Conclusion

In our second participation in the MAPC we had again a worthy experience. Our team performed very well and we won the MAPC for the second consecutive year. The strat-

egy to get many small zones was the strongest point of our team and it became more difficult for the opponents to disturb our zones because our agents were spread out 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 with low thinning (less than 20%) and with too many good vertices gathered in the same area. The best strategy for it seems to be to conquer a big zone and defend it instead of building small zones.

We also had the opportunity to use new tools, such as the JaCaMo and to test some issues related to our research topics, such as *count-as rules*. It was a good challenge and we got positive results. The main results were (i) the contributions for the improvement of the used tools and (ii) the concrete verification that considering the organization and the environment as first-class entities has improved the team program quality.

Finally, as suggestions to improve the current scenario, we suggest that ranged actions be revised to balance the fail probability. So far, it is not a good strategy to use ranged actions, since the agents need to buy several sensors to decrease this probability.

References

1. Tristan M. Behrens, Koen V. Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, April 2011.
2. Olivier Boissier, Rafael Bordini, Jomi Fred Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761, 2013.
3. Rafael H. Bordini, Michael Wooldridge, and Jomi F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
4. Maiquel de Brito, Jomi F. Hübner, and Rafael H. Bordini. Programming institutional facts in multi-agent systems. In Huib Aldewereld and Jaime Simão Sichman, editors, *COIN@AAMAS 2012*, pages 31–25, 2012.
5. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.
6. Michael Köster, Federico Schlesinger, and Jürgen Dix. The multi-agent programming contest 2012. In *Programming Multi-Agent Systems*, volume 7837 of *LNCS*, pages 174–195. Springer Berlin Heidelberg, 2013.
7. Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *AAMAS*, 23:158–192, 2011.
8. Maicon R. Zatelli, Daniela M. Uez, José R. Neri, Tiago L. Schmitz, Jéssica P. C. Bonson, and Jomi F. Hübner. SMADAS: A cooperative team for the multi-agent programming contest using jason. In *Programming Multi-Agent Systems*, volume 7837 of *LNCS*, pages 196–204. Springer Berlin Heidelberg, 2013.