Int. J. Accounting, Auditing and Performance Evaluation, Vol. 0, No. 00, 1–?? 1

Developing Organised Multi-Agent Systems Using the $\mathcal{M}$oise$^+$ Model: Programming Issues at the System and Agent Levels

Jomi F. Hübner,
Jaime S. Sichman, and
Olivier Boissier

GIA / DSC / FURB
Braz Wanka, 238
89035-160, Blumenau, Brazil
E-mail: jomi@inf.furb.br

LTI / EP / USP
Av. Prof. Luciano Gualberto, 158, trav. 3
05508-970 São Paulo, SP, Brazil
E-mail: jaime.sichman@poli.usp.br

SMA / G2I / ENSM.SE
158 Cours Fauriel
42023 Saint-Etienne Cedex, France
E-mail: Olivier.Boissier@emse.fr

Abstract: Multi-Agent Systems (MAS) has evolved towards the specification of global constraints that heterogeneous and autonomous agents are supposed to follow when concerning open systems. A subset of these constraints is known as the MAS organisation. This article describes a set of computational tools that supports the development and the programming of such systems. At the system level, it is provided a middleware which ensures that all agents will follow the organisational constraints. At the agent level, the AgentSpeak language is extended, using ***Jason*** features, so that the agents can perceive and act upon the organisation they belong.

Keywords: Multi-Agent Systems; MAS organisations; Engineering organisations for MAS; Programming agents; $\mathcal{M}$OISE$^+$; ***Jason***.

Biographical Notes: Jomi Fred Hübner obtained his PhD at University of São Paulo, Brazil. The subject of his thesis was how to model the reorganisation process of a MAS. Currently, he is a Professor at Univer-

sity of Blumenau, Brazil. His research interests are MAS organisation and tools to develop BDI-based systems.

Jaime Simão Sichman received his PhD at INPG, France. He is Associate Professor at University of São Paulo, Brazil. His research interests are related to agents' organisational models, multi-agent based simulation, and reputation and trust in MAS.

Olivier Boissier received his PhD at INPG Grenoble, France. He is Professor at the ENS Mines of Saint-Etienne, France. His research interests are related to organisational models, agents' architectures, autonomy and control.

1 Introduction

The autonomy of the agents is among the most important characteristics of the concept of agency [39]. However, this autonomy can lead the overall system to an undesired behaviour, since each agent does what it wants. This problem may be solved by assigning an *organisation* to the system, as it is done in Human Societies [32, 5]. In this context, the organisation can be considered as a set of behavioural constraints that an agent adopts when entering into the system, expressed, for instance, by the role it will play [13, 25]. This approach is especially useful in *open* Multi-Agent Systems (MAS) [22] where we do not know what sort of agent will enter the system and therefore some constraints should be enforced (this motivation for organised MAS is well described in [36, 11]).

The precise meaning of constraint used to describe an organisation is defined by the underlying organisational model. According to Lemaître and Excelente [30], organisational models may be conceived along two main points of view: *agent* centred and *system* centred. The former takes the agents as the 'engine' for the organisation. The organisation only exists inside the agents and an unified global view of the organisation is possible only as an observable phenomena (Figure 1 (a) and (b) depicts this view). For instance, in an ant colony, the organisational behaviour constraints are somehow defined inside the ants or in the environment and we can describe the organisation based on the collective emergent behaviour. The latter point of view considers a different perspective: the organisation exists as an explicit entity of the system and is not localised in the agents. The constraints are defined by the designer (or by the agents themselves in self-organised systems) and the agents are supposed to follow this organisation. For example, in a school there are documents that explicitly state the academic organisation by means of its structure and rules. Besides the observable organisation, in this point of view an observer can also obtain a description of the explicit organisation. Of course, the explicit and the observed organisations may differ.

These two points of view are concerned with the 'place' of the organisation in the system. Taking an agent architecture perspective, there are other situations regarding the agents' capabilities to represent and reason about their organisation. This extension to Lemaître and Excelente [30] work was proposed in [23]. In case (a) of Figure 1 the agents are unable to represent the organisation, although some external observer can see an emergent structure (e.g., an ant colony [14]). Indeed, in all four cases of Figure 1 an observer can describe, using organisational concepts

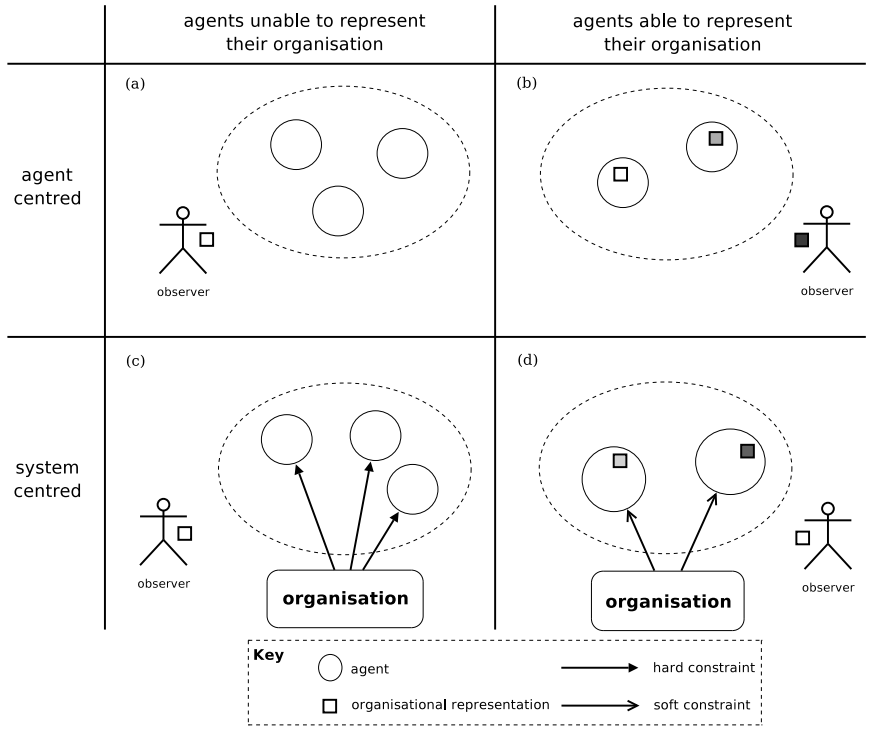

Figure 1 Four views of organisation.

and languages, what s/he is viewing. In case (b), the agents have an internal representation of the organisation and follow this when deciding what to do (e.g., coalition formations [35]). The agents' representation is obtained by perception, communication, etc., since there is no explicit representation of the organisation available. In case (c), the organisation exists but the agents do not reason about it, they simply obey as if the organisational constraints were hardwired in them (e.g., the MAS resulted from some AOSE methodologies where the agents' code is generated from specifications based on organisational concepts such as roles and responsibilities [29]). Note that in all three cases, an agent has no organisational autonomy, in (a) and (c) they do not *decide* whether to follow the organisational constraints and in (b) the organisation is defined by the agent itself, so it can not be autonomous regarding its own organisation. Finally, in case (d) – the focus of this article – we have both an explicit representation of the organisation available to the agents at runtime and agents able to read, represent, and reason about the organisation. In these kinds of systems the agents can exert organisational autonomy, since they may decide whether to follow the organisation or not.

In order to implement a system that follows organisational constraints, it is also usual to take either an agent or system centred point of view (in [38] these views are called agent and institutional perspectives). In the former, the focus is on how to develop an agent reasoning mechanism that follows the organisation. In the latter, the main concern is how to develop an MAS framework that ensures the satisfaction of the organisational constraints. This latter is important in heterogeneous

and open systems, since the agent that enters the system may have an unknown architecture. Of course these agents still need to have access to an organisational representation that enable them to reason about it. However, any agent should follow the organisation despite its organisational reasoning abilities. As far as we know, the following available frameworks are system centred: AMELI [16] (based on ISLANDER), MadKit [21] (based on AGR), KARMA [33] (based on STEAM), and $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [28] (based on $\mathcal{M}$OISE$^+$).

Although these frameworks ensure that the agents will follow the organisational constraints, they do not help the developer to program the agents to reason about the organisation. We speculate that both approaches are needed to develop systems like those in Figure 1 (d): (*i*) a framework to ensure organisational constraints and (*ii*) an agent programming language that supports the agent decision making about its organisation. The objective of this article is thus to present some software tools to take one step towards the complete implementation of organised MAS. The proposal is based on our previous work on the $\mathcal{M}$OISE$^+$ organisational model.

This article is organised as follows: sections 2 and 3 present a comprehensive description of the $\mathcal{M}$OISE$^+$ model and accompanying tools that implement an organisational framework (item *i* above). These descriptions are the background knowledge necessary to present the very contribution of this article in Section 4: $\mathcal{J}$-$\mathcal{M}$OISE$^+$, which is an extension of the AgentSpeak language (initially proposed in [34] and improved in [1, 2]). $\mathcal{J}$-$\mathcal{M}$OISE$^+$ allows developers to use this high level BDI language to program organisation aware agents (item *ii* above). Our extension enables agents to perceive their organisation, especially its changes (e.g., a new group is created, an agent has adopted a role), and to act upon it (e.g., create a group, adopt a role). Finaly, before concluding, we compare our proposal to related works in Section 5.

2 $\mathcal{M}$oise$^+$ Organisational Model

The organisational models are normally focused on a single dimension of the organisation. We have identified three main organisational dimensions in the models proposed in the area. The first dimension concerns the *functioning* of the organisation, for instance, the specification of global plans, the policies to allocate tasks to agents, the coordination of plans execution, and the quality (time consumption, resources usage, ...) of a plan. In this dimension, the global purpose of the system is better achieved because the MAS has a kind of organisational memory where the best plans to achieve a global goal are stored (e.g., TÆMS [31], STEAM [37]). The second dimension addresses a more static aspect of the organisation: its *structure*, i.e., the roles, the relations among roles, the groups of roles, etc. In these models, the global purpose is accomplished while the agents have to follow the obligations and permissions that their roles entitle them (e.g., AGR [17], TOVE [18]). The third dimension focus on the definition of high level *norms* that the agents should obey (e.g., ISLANDER [15], OPERA [11]).

Figure 2 informally shows how an organisation could explain or constrain the agents behaviour in the case where we consider an organisation as having both structural and functional dimensions. In this figure, it is supposed that an MAS has the purpose of maintaining its behaviour in the set P, where P represents

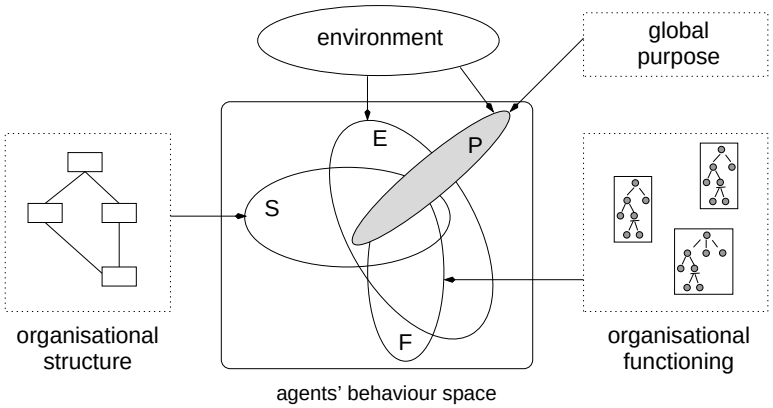

Figure 2 Organisation effects on an MAS.

all behaviours which draw the MAS's global purposes. In the same figure, the set E represents all possible behaviours in the current environment state. The organisational structure is formed, for example, by roles, groups, and links that limit the agents behaviour to those inside the set S, i.e., the set of possible behaviours $(E \cap S)$ becomes closer to P. It is a matter of the agents, and not of the organisation, to conduct their behaviours from a point in $((E \cap S) - P)$ to a point in P. In order to help the agents in this task, the functional dimension contains a set of collective plans that has been proved efficient in activating P behaviours. For example, in a soccer team we can specify both the structure (defence group, attack group, and some roles for each group) and the functioning of the team (e.g., rehearsed plays, as a kind of predefined collective plans that have already worked well).

Having only one dimension is normally insufficient for a system. If only the functional dimension is specified, the organisation has nothing to 'tell' the agents when there is no plan to execute (the set of possible behaviours is outside the set F of Figure 2). Otherwise, if only the organisational structure is specified, the agents have to reason for a collective plan every time they want to play together. Even with a small search space of possible plans (since the structure constrains the agents' options), this may be a hard problem. Furthermore, the plans developed for a particular problem are lost, since there is no organisational memory to store these plans. Thus, in the context of some application domains, if the organisation model specifies both dimensions while maintaining suitable independence, then the multi-agent system that follows such a model can be more effective in adjusting the group behaviour to its purpose. Another advantage of having both dimensions is that the agents have more information to reason about the others position in the organisation and thus better interact with them.

The definition of a proper organisation for an MAS is not an easy task. On one hand the organisation can be too flexible, and then it does not help the achievement of the global purpose. On the other hand, it can be too stiff, and then the organisation removes any advantage of the agents' autonomy. An initial adequate organisation is normally set up by the MAS designer, however this may become not suitable in dynamic environments. In this case the system should have the ability to change or adjust its organisation.

The $\mathcal{M}$OISE$^+$ organisational model is an attempt to join the three dimensions into an unified model suitable for the reorganisation process [24, 25]. Its main feature concerning the reorganisation process is the *independence* between the first two dimensions, which are linked by the third one, the deontic dimension. The MAS can therefore change its own structure without changing its functioning, and vice versa [26]. In the next three subsections, we give a brief description of these dimensions of the $\mathcal{M}$OISE$^+$ model (a more detailed description is found in [25, 23]). For each dimension we have a corresponding specification that together forms the Organisational Specification (OS). When a set of agents adopts an OS they dynamically instantiate an Organisational Entity (OE). Once created, the OE's history starts and is run by events like the entering or exiting of other agents in the OE, group creation, role adoption, mission commitment, etc.

2.1 Structural Dimension

The $\mathcal{M}$OISE$^+$ Structural Specification (SS) is built in three levels: (*i*) the behaviours that an agent is responsible for when it adopts a role (*individual* level), (*ii*) the acquaintance, communication, and authority links between roles (*social* level), and (*iii*) the aggregation of roles in groups (*collective* level). Throughout this article, the analogy of a soccer team will be used to further illustrate and clarify the model. Using the $\mathcal{M}$OISE$^+$ notation, the team structure is specified in Figure 3 (a more detailed definition is found in [25]).

At the individual level, the soccer team is defined by roles like goalkeeper, back player, leader, attacker, coach, etc., and an inheritance relation among them. For example, an agent playing as a goalkeeper inherits all properties of the roles back, player, and soc. In $\mathcal{M}$OISE$^+$ model, the adoption of roles is constrained by a *compatibility* relation between roles. An agent can play two or more roles only if they are compatible. For example, an agent playing the leader role is also allowed to play the back role since these roles are compatible. Due to the role specialisation (see the back-goalkeeper inheritance relation in Figure 3), the leader can also play the goalkeeper role.

In the collective level, the players are divided into two groups (defence and attack) which are sub-groups of the team. According to the composition relation of Figure 3, the defence group specification is formed by three roles (goalkeeper, back, and leader) and the maximum number of players in the group is constrained (one goalkeeper, three backs, and one leader). It is also defined the minimum number of players for the group to be considered *well formed* (one goalkeeper and three backs). An agent playing leader is thus optional in the defence group. The leader is also optional in the attack group. However, the group `team` must have one agent playing leader to be considered well-formed. In this case, the composition has a sub-group scope, meaning that the leader player must be in a sub-group (defence or attack). Only after one agent is playing the role leader in a sub-group, may the team be considered well-formed. More precisely, a team is well formed if it has one defence sub-group, one attack sub-group, one or two agents playing the coach role, one agent playing the leader role, and the two sub-groups are also well formed.

In the social level, the roles are linked. Each link has a source and target role, for example, in Figure 3 it is specified that the coach (the source of the link) has authority on all players (the target of the link) by means of an authority

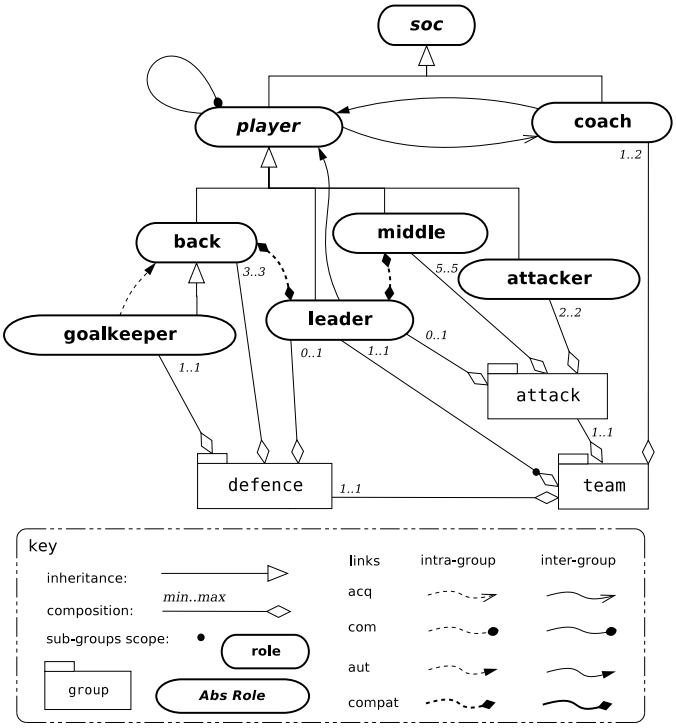

Figure 3 Soccer team structure using $\mathcal{M}$OISE$^+$.

link. Although the authority link target is the role 'player', other roles inherit this link. The same construction is used to increase the leader's authority. Besides the authority link, we can define communication and acquaintance links. Two agents can communicate if they play roles with a communication link. The acquaintance link is similar. In order to simplify the specification, for every authority link there is an implicit communication link and for every communication link there is an implicit acquaintance link.

A $\mathcal{M}$OISE$^+$ group can have intra-group and inter-group links. The intra-group links state that an agent playing the source role in a group *gr* is linked to all agents playing the target role in the *same* group *gr* or in a sub-group of *gr*. The inter-group links state that an agent playing the source role is linked to all agents playing the target role in spite of which groups these agents belong to. For example, the coach authority on player is an inter-group link (the coach and the player agents do not need to belong to the same group), while the goalkeeper authority on backs is an intra-group link (both agents must belong to the same group to 'use' this link).

2.2 Functional Dimension

The Functional Specification (FS) is composed by a set of schemes which represent how an MAS usually achieves its *global* (organisational) goals [7] stating how these goals are decomposed (by plans) and distributed to the agents (by missions).

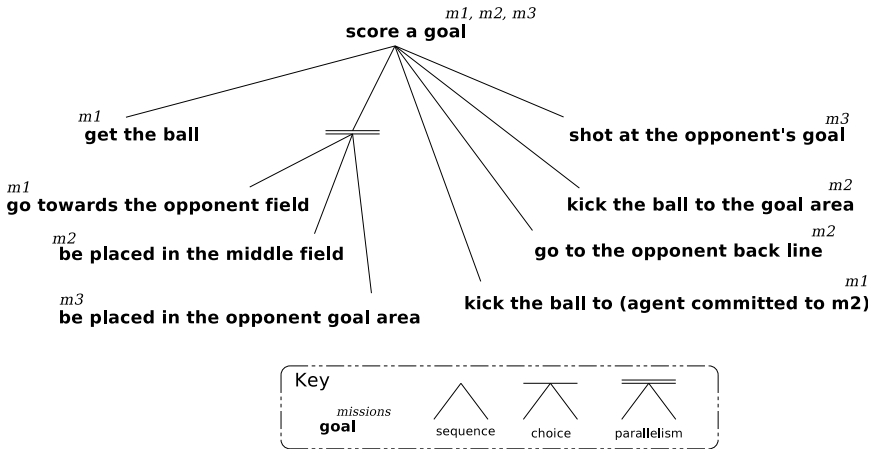

Figure 4 Soccer team attack scheme using $\mathcal{M}$OISE$^+$.

The scheme can be seen as a goal decomposition tree where the root is a global goal and the leafs are goals that can be achieved by the agents. Such decompositions may be set either by the MAS designers who specify their expertise in the scheme or by the agents that store their past (best) solutions. In the soccer example, suppose the team has a rehearsed play as the one specified in Figure 4. This scheme has three missions (m_1, m_2, and m_3) – a mission is a set of coherent goals that an agent can achieve. When an agent commits to a mission, it is responsible for all this mission's goals. For example, an agent committed to the mission m_3 has the goals 'be placed in the opponent goal area', 'shoot at the opponent's goal', and, a common goal, 'score a goal'. Each mission also has a cardinality constraint in the scheme that state how many agents should commit to it. In the soccer example, all the three missions must be committed by only a single agent.

In a scheme, each non-leaf goal $g_i \in \mathcal{G}$ (where $\mathcal{G}$ is the set of global goals) is decomposed in sub-goals through plans using three operators:

- sequence ",": the plan "$g_1 = g_2, g_3$" means that the goal g_1 will be achieved if and only if the goal g_2 and subsequently goal g_3 are achieved;
- choice "|": the plan "$g_1 = g_2 \mid g_3$" means that the goal g_1 will be achieved if one, and only one of, the goals g_2 or g_3 is achieved; and
- parallelism "||": the plan "$g_1 = g_2 \parallel g_3$" means that the goal g_1 will be achieved if both g_2 and g_3 are achieved, but they can be achieved in parallel.

2.3 Deontic Dimension

The deontic dimension addresses the autonomy of the agents by stating explicitly what is permitted and obligated in the organisation. The corresponding specification describes the roles' permissions and obligations for missions. A permission $permission(\rho, m)$ states that an agent playing the role ρ is allowed to commit to

the mission m. Furthermore, an obligation $obligation(\rho, m)$ states that an agent playing ρ ought to commit to m. For example, in the soccer team Deontic Specification (DS) (Table 1), three roles have the right to start the scheme presented in Figure 4 because they have the permission for the scheme's root missions. Once the scheme is created, the other agents (playing back, middle, ...) are obligated by their roles' deontic relations to participate in this scheme. These other agents ought to pursue their mission's goals, following the sequence specified by this scheme. For instance, when a middle agent accepts the mission m_2, it will try to achieve its goal 'be placed in the middle field' only after the goal 'get the ball' is already satisfied by a back agent committed to the mission m_1.

An important feature of $\mathcal{M}$OISE$^+$ is to avoid to link roles and goals directly. One reason is to define sets of coherent goals (the missions) which are not reducible to the concept of role. Roles are indirectly linked to missions by means of permissions and obligations. Another reason is to add some independence between the functional and the structural specifications.

2.4 Organisational Entity Dynamics

The dynamic of the OE consists of changes in its state, which is represented by the following tuple:

$$\langle os, \mathcal{A}, \mathcal{GI}, \mathcal{SI}, gt, sg, st, ar, am, gs \rangle$$

where:

- os is the initial organisational specification which is composed by a set of group specifications $\mathcal{GT}$, a set of scheme specifications $\mathcal{ST}$, a set of roles $\mathcal{R}$, a set of missions $\mathcal{M}$, and a set of global goals $\mathcal{G}$ (although an OS has more elements than these listed here, we have included only those directly linked to the entity definition);
- $\mathcal{A}$ is the set of agents in the MAS;
- $\mathcal{GI}$ is the set of created groups;
- $\mathcal{SI}$ is the set of scheme instances;
- $gt : \mathcal{GI} \rightarrow (\mathcal{GT} \times \mathcal{A})$ maps each group in $\mathcal{GI}$ to a group specification and an owner agent; the owner agent is the creator of the group;
- $sg : \mathcal{GI} \rightarrow \mathbb{P}(\mathcal{GI})$ maps each group to its sub-groups;

Table 1 Soccer team deontic relations using $\mathcal{M}$OISE$^+$.

role	deontic relation	mission
back	*permission*	m_1
middle	*obligation*	m_2
attacker	*obligation*	m_3

- $st : \mathcal{SI} \rightarrow (\mathcal{ST} \times \mathbb{P}(\mathcal{GI}) \times \mathcal{A})$ maps each scheme instance to its specification, responsible groups, and owner;
- $ar : \mathcal{A} \nrightarrow \mathbb{P}(\mathcal{R} \times \mathcal{GI})$ maps agents to the roles they are playing the groups;
- $am : \mathcal{A} \nrightarrow \mathbb{P}(\mathcal{M} \times \mathcal{SI})$ maps agents to the missions they are committed to in the schemes;
- $gs : \mathcal{SI} \times \mathcal{G} \nrightarrow \{unsatisfied, satisfied, impossible\}$ maps a goal to its state, all goals are initially mapped to *unsatisfied*.

This section presents a brief general overview of the entity dynamics. More details will be given in the next sections. The OE dynamics has the following life-cycle:

1. Creation of the OE based on an organisational specification.
2. Running the OE, which consists of the life-cycle of its groups, schemes, and agents.
3. Entity destruction to finish the organisation.

The life-cycle of a group has the following four steps:

1. Creation of the group based on a group specification. The agent which created the group becomes the group's owner.
2. Adoption of roles in the group by the agents.
3. When the group is well formed (i.e., it has a valid number of players for each role), it can be responsible for the execution of schemes.

   Being responsible for a scheme means that the agents in the group will be the agents that commit to the missions of the scheme. For instance, if we have two instances (s_1 and s_2) of the same scheme specification (sch_1) and two group instances (g_1 and g_2), such that $st(s_1) = (sch_1, \{g_1\}, \alpha_1)$ and $st(s_2) = (sch_1, \{g_2\}, \alpha_2)$, only agents playing roles in g_1 can (or have to) commit to missions in s_1 and only agents from g_2 can (or have to) commit to missions in s_2. The set of schemes that a group is responsible for is thus important to define the permissions and obligations for an agent (see Algorithm 1).
4. Removal of the group from the OE by the owner. In order to be removed, a group must have (i) no agents playing any role in it, (ii) no sub-groups, and (iii) no responsibility for any scheme.

The life-cycle of a scheme is formed by the following steps:

1. Creation of the scheme based on a scheme specification by an agent that will become its owner. The owner must have permission for a mission which includes the root goal.
2. Assignment to responsible groups.
3. Commitment of some agents.

4. When the scheme is well formed (i.e., there are agents committed to all missions), the goals can be pursued by the committed agents.

5. The scheme is finished when the root goal is set as satisfied or impossible.

6. When no agent is committed to the scheme, it can be removed from the OE by its owner.

Finally, the life-cycle of an agent inside the entity is:

1. Entering in the system.

2. Adoption of roles in groups, this adoption is constrained by the role's cardinalities and compatibilities.

3. Commitment to missions in schemes, this commitment is constrained by the deontic specification (for example, an agent can commit only to permitted missions).

4. Removal of commitments. This removal is also constrained, an agent can remove its commitments only if its obligations were already fulfilled. If an agent does not fulfil its obligations, the system should not allow it to uncommit.

5. Removal of roles if it has no more commitments.

6. Leaving the system, if the agent has no more roles to play.

3 $\mathcal{S}$-$\mathcal{M}$oise$^+$ Organisational Middleware

$\mathcal{S}$-$\mathcal{M}$OISE$^+$ is an open source implementation of an *organisational middleware*, based on a system centred view that follows the $\mathcal{M}$OISE$^+$ model. This middleware, initially described in [28], is the interface between the agent and the system levels, providing access to the communication level, to the information about the current state of the organisation (created groups, schemes, roles assignments, etc.), and allowing the agents to change the organisation entity and specification (see Figure 5). Of course these changes are constrained to ensure that the agents respect the organisational specification.

$\mathcal{S}$-$\mathcal{M}$OISE$^+$ has two main components: an OrgBox API that agents use to access the organisational layer (this component is detailed in Section 3.2) and a special agent called OrgManager. This latter agent stores the current state of the OE and maintains its consistency during its life-cycle, i.e., it ensures organisational constraints. The OrgManager receives messages from the agents' OrgBox asking for changes in the OE state (e.g., role adoption, group creation, mission commitment). The messages' contents are organisational actions, detailed in Section 3.1. The OrgManager performs the actions only if the request does not violate the organisational constraints. For example, if an agent wants to adopt a role ρ_2 but it already plays a role ρ_1 which is not compatible with ρ_2, the adoption of ρ_2 must be denied.

3.1 Organisational Actions

The OE is changed by *organisational actions* encapsulated in messages sent by the agents to the OrgManager. Each action has arguments, preconditions and effects (Table 2 summarises these actions). In this article we describe only some of the actions using the soccer example, a full formalisation can be found in [23] and also on `http://moise.sourceforge.net`.

As an example, let's suppose that we have an OE where the following actions have already been performed by an agent identified by α:

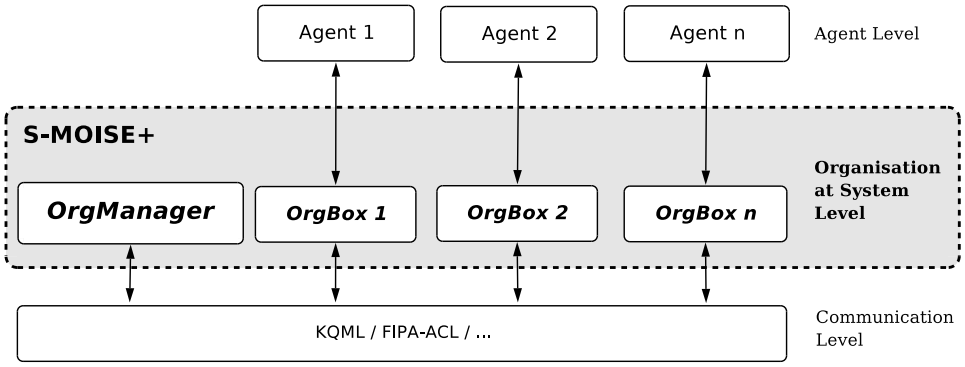

Figure 5 $\mathcal{S}$-$\mathcal{M}$OISE$^+$ components.

Table 2 $\mathcal{S}$-$\mathcal{M}$OISE$^+$ main organisational actions.

Action	*Description* and some preconditions
`create_group(`*gt* `[,` *gi*`])`	Creates a new group, or a sub-group of *gi* if *gi* argument is informed, based on specification *gt* ($gt \in \mathcal{GT}$ and *gi* identifies an instance group).
`remove_group(`*gi*`)`	Removes the group identified by *gi*.
`create_scheme(`*st*`,` *gis*`)`	Creates a new scheme instance from specification *st* ($st \in \mathcal{ST}$), *gis* ($gis \subseteq \mathcal{GI}$) is a set of groups that are responsible for the new scheme execution.
`remove_scheme(`*si*`)`	removes the scheme *si* from the OE; to be removed, the root goal of the scheme must be satisfied or impossible and no agent is still committed to it.
`set_goal_state(`α`,` *si*`,` *g*`,` *s*`)`	The goal *g* of the scheme *si* is set as *s* ($s \in \{satisfied, impossible\}$) by the agent α (α must be committed to a mission that includes *g*).
`adopt_role(`α`,` ρ`,` *gr*`)`	The agent α adopts the role ρ in the group *gr*.
`remove_role(`α`,` ρ`,` *gr*`)`	The agent α gives up the role ρ in the group *gr* (this role missions must be finished).
`commit_mission(`α`,` *m*`,` *si*`)`	The agent α commits to the mission *m* in the scheme *si*.
`finish_mission(`α`,` *m*`,` *si*`)`	The agent α finishes its mission *m* in the scheme *si* (all the mission's goal must be satisfied or declared impossible).

- `create_group(team)`: a group, identified hereafter by gr_t, was created from the `team` specification defined in Figure 3;
- `create_group(defence,` gr_t`)`: a group, identified hereafter by gr_d, was created from the `defence` specification as a sub-group of gr_t;
- `create_group(attack,` gr_t`)`: a group, identified hereafter by gr_a, was created from the `attack` specification as a sub-group of gr_t; with these three groups created, the state of the entity is $\mathcal{GI} = \{gr_t, gr_d, gr_a\}$, $gt = \{gr_t \mapsto (\text{team}, \alpha), gr_d \mapsto (\text{defence}, \alpha), gr_a \mapsto (\text{attack}, \alpha)\}$, and $sg = \{gr_t \mapsto \{gr_d, gr_a\}\}$ (other components of the state remain empty);
- `create_scheme(side_attack,` $\{gr_t\}$`)`: an instance of the side attack specification (Figure 4), identified by sch_{sa}, was created and the agents of the group gr_t are responsible for this scheme's missions; the state of the entity is $\mathcal{SI} = \{sch_{sa}\}$ and $st = \{sch_{sa} \mapsto (\text{side\_attack}, \{gr_t\}, \alpha)\}$.

After the execution of these actions, the groups are not well formed, since there is no agents playing their roles. The defence group, for instance, needs one agent playing goalkeeper. If an agent α wants to adopt the role ρ in the group gr, it must send the action `adopt_role(`α`,` ρ`,` gr`)` to the OrgManager.[a] As every organisational action in $\mathcal{S}$-$\mathcal{M}$OISE$^+$, the role adoption action has some preconditions to ensure that no organisational constraint is violated:

1. the role ρ must belong to gr's group specification;
2. the number of ρ players in gr must be lesser than the maximum number of ρ players defined in the gr's compositional specification;
3. for all roles ρ_i that α already plays, the roles ρ and ρ_i must be intra-group compatible in the gr's group specification;
4. for all roles ρ_i that α already plays in groups other than gr, the roles ρ and ρ_i must be inter-group compatible.

The effect of adopting a role is a new mapping $(\alpha \mapsto (\rho, gr))$ in the ar function. Notice that a role is always adopted inside a group of agents, since role is a relational concept [6].

In our example, let's suppose that eleven agents have already adopted roles such that the three groups are well formed and that the goal 'get the ball' of the scheme sch_{sa} is satisfied. Among these agents, 'Lucio' has adopted the role middle in the gr_d group (since gr_d is a sub-group of gr_t, Lucio also belongs to gr_t). Is this agent following its organisational obligations? No, because it plays a middle role in the group that created the side attack scheme and this role obligates it to commit to mission m_2 (Algorithm 1 describes how to get all missions an agent is obligated to). In order to be organisation compliant, Lucio commits to the m_2 mission through the action `commit_mission('Lucio',` m_2`,` sch_{sa}`)`. From the OrgManager point of view, this action also has some preconditions:

1. the scheme sch_{sa} must not be finished yet;

[a] The reasons for an agent to adopt a role is not covered by the $\mathcal{M}$OISE$^+$ model, for more details regarding motivations for role adoption, the reader is referred to [20, 17, 10, 9].

2. the agent must play a role in the scheme's responsible groups;

3. this role must be permitted or obligated to the mission, as defined in the DS.

The action effect is so that the function *am* is changed to add the mapping 'Lucio' $\mapsto (m_2, sch_{sa})$.

After its commitment, Lucio will likely pose the question: which are the global goals I have to achieve? In the case of its m_2 goals, only the goal 'be placed in the middle field' is permitted (see Figure 4). Its second goal 'go to the opponent back line' is not permitted by the current state of sch_{sa}. This second goal should be pursued only after another global goal is satisfied, since it depends on 'kick the ball to' achievement. Algorithm 2 is used in the OrgManager implementation to identify permitted global goals. Thus, while some goals are becoming satisfied (by the action `set_goal_state`), others become permitted. When a goal becomes permitted, the agents committed to it are informed by the OrgManager. This mechanism is very useful to *coordinate* the agents in the scheme execution. The agent developer does not need to program messages that synchronise the agents in the scheme execution.

In the $\mathcal{S}$-$\mathcal{M}$OISE$^+$ middleware, the OrgManager both constrains the agents' actions and provides useful information for the agents' organisational reasoning and coordination (e.g., missions they are supposed to commit to and goals they can pursue). The agents can get this information either by their OrgBox API (presented in the next section) or by some other language resources (as shown in the Section 4).

3.2 Agents' OrgBox

The OrgBox is the interface the agents use to access the organisational layer and thus the communication layer. When an agent intends to (*i*) change the organisational entity (adoption of a role, for instance), (*ii*) send a message to another agent, or (*iii*) get the organisational entity state it has to ask this service for its OrgBox. The OrgBox will therefore interact with the OrgManager or another agent using the communication layer. We have developed a protocol in the communication layer that is followed by the OrgManager and the OrgBox to exchange information and to

```
function getObligatedMissions(agent α)

all ← empty list  // list of obligated missions
forall role ρ ∈ ar(α, gr) do
    // gr is the group where ρ is being played
    forall scheme si that gr is responsible to do
        if si is not finished then
            forall mission m in the scheme si do
                if obligated(ρ, m) is in the deontic specification then
                    all ← append(all, m);
return all;
```

Algorithm 1: Algorithm to compute the missions an agent is obligated to.

inform about organisational events. We can see the OrgBox as a component that encapsulates this protocol. In current implementation, the communication layer is implemented by SACI system (`http://www.lti.pcs.usp.br/saci`) – a KQML compliant multi-agent communication infrastructure.

When an agent asks OrgManager for a 'copy' of the current state of the OE, it will not receive exactly what is in the OrgManager's memory. According to the $\mathcal{M}$OISE$^+$ model, an agent is allowed to know another agent α only in case it plays a role ρ_1, α plays ρ_2, and these roles are linked by an acquaintance link. For example the *player* role of Figure 3 has an acquaintance link to the *coach* role, thus an agent playing this role is allowed to know the agents who are playing the *coach* role. Indeed, since *player* is an abstract role, no agent will adopt it, however other roles (like *back*, *leader*, etc.) inherit this acquaintance link from the *player* role. The copy of the OE received by an agent contains therefore only information related to acquainted agents.

The OrgBox interface is hence invoked by the agents to send messages, to ask for information, ant to change the organisation. However, this service is also invoked by the OrgManager. When there is a state change of a scheme to which some agent is committed, the OrgManager informs this agent's OrgBox about the new permissions, obligations, and goals it can pursue. The OrgBox then notifies the agent about this event. Obviously, the OrgBox only informs the agent about its organisational goals, it is a matter of the agent to achieve them (by plans, behaviours, etc.). The fact that is stated in the organisational system level is that the agent is responsible for achieving such a goal.

3.3 Organisational Constraints

In the $\mathcal{S}$-$\mathcal{M}$OISE$^+$ middleware, we have two kinds of constraints: hard constraints and soft constraints. Hard constraints are those that must be enforced to maintain the organisational entity in a consistent state. Since these constraints can not be violated by any agent, they should be implemented in the middleware. From all organisational constraints defined in the $\mathcal{M}$OISE$^+$ model, the following are hard constraints: (i) maximum number of role players in a group as defined

```
function isPermitted(scheme sch, goal g)

if g is the sch root then
    return true;
else
    g is in a plan that match 'g_0 = ··· g ···';
    if g is in a plan that match 'g_0 = ··· g_i , g ···' then
        if g_i is already satisfied then
            return true;
        else
            return false;
    else
        return isPermitted(sch, g_0);
```

Algorithm 2: Algorithm to verify permitted goals.

in the cardinality relation between a role and a group; (ii) role compatibility; (iii) commitment allowed only to permitted or obligated missions; (iv) acquaintance and communication links; and (v) creation of groups and schemes that are previously specified.

Soft constraints are related to the deontic dimension and are not guaranteed by the middleware, since the agents are supposed to autonomously decide to follow them or not. These constrains can therefore be violated. A sanction system may be used to enforce them; such a system is not covered in this paper, but one proposal is presented in [19]. Among the $\mathcal{M}$OISE$^+$ specification elements, the authority link, the commitment to obligated missions, and the achievement of goals are soft constraints. For example, if an agent is obligated to achieve some goal and is not pursuing it, the middleware can not know it because it has no access to the internal state of the agents. It could also be the case when the middleware may know that the agent is not respecting the constraint (e.g., it is not committing to an obligated mission), but there is no instruments to force the agent to commit. Soft constraints are thus normally enforced in the agent reasoning capabilities.

An important feature of $\mathcal{S}$-$\mathcal{M}$OISE$^+$ middleware concerning open systems is that it can ensure hard organisational constraints despite the architecture or language used to develop the agents that use the middleware. The requirements to enter the system are (i) to use the OrgBox to interact with the organisational level and (ii) to understand $\mathcal{M}$OISE$^+$ specifications.

Once this middleware constrains the agents' behaviour based on a $\mathcal{M}$OISE$^+$ organisational specification, we can see this specification as a kind of *declarative organisational programming language* at the system level. Obviously, this specification does not define the exact behaviour of the agents, as they remain autonomous with respect to this organisation. However, as pointed out in the introduction, in order to develop a multi-agent system we also need to program the agents: in our proposal, we need to program agents which are organisation aware. The next section will describe a proposal for organisational programming at the agent level.

4 $\mathcal{J}$-$\mathcal{M}$oise$^+$ Organisational Agent

This section describes how an agent programming language can be used to make both organisational information and actions available to agents. Among several languages for agent programming, we have chosen the AgentSpeak language [34] and its open source interpreter ***Jason*** (`http://jason.sf.net` [2, 3, 4]) to program the organised agents. This choice was made because the language is based on the well known BDI architecture and the interpreter is easily customised to include the organisational support. This tool is called $\mathcal{J}$-$\mathcal{M}$OISE$^+$, since it joins ***Jason*** and $\mathcal{M}$OISE$^+$.

The $\mathcal{J}$-$\mathcal{M}$OISE$^+$ is built upon the $\mathcal{S}$-$\mathcal{M}$OISE$^+$ middleware, thus including the OrgManager and OrgBox components. We have only extended the communication level options for the communication level available in ***Jason***: SACI, JADE, standalone, or any other infrastructure available in future releases. Figure 6 illustrates how these components are integrated into ***Jason***. The $\mathcal{J}$-$\mathcal{M}$OISE$^+$ basically offers to the AgentSpeak programmer (i) a set of actions to change the organisa-

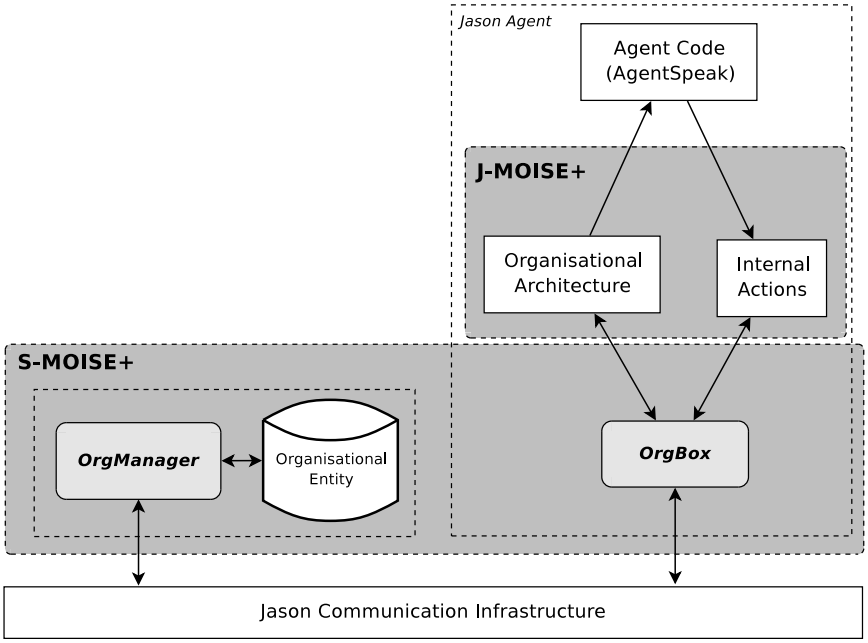

Figure 6 General view of the $\mathcal{J}$-$\mathcal{M}$OISE$^+$ architecture.

tion (Section 4.1) and (*ii*) produce some events so that agents may react to the organisational changes (Section 4.2).

4.1 Organisational Actions

The syntax of AgentSpeak is based on the notion of plans. A plan is triggered by some event and is guarded by some context, the syntax is

```
<event> : <context> <- <body>.
```

In the case where the event happens and the context holds, the plan's body is executed. For instance, the plan

```
+ball(X,Y) : i_am_near(X,Y) <- action1; action2; ...
```

is triggered when the agent perceives the ball at location X,Y (`+` means that something was added in the belief base, as a perception in the above example). If the agent can prove from its beliefs that `i_am_near(X,Y)` holds, the plan is selected for execution and becomes an intention. Every intention is executed by performing the actions of the body.

The set of available actions in ***Jason*** can be extended by means of *internal actions*. We thus create a library of internal actions called `jmoise` (see Figure 6) that implements all the organisational actions shown in Table 2. These actions simply ask the OrgBox to send messages to the OrgManager. For example, the following plan uses the internal action `jmoise.create_group` to create a new group based on the specification `team`:

```
+some_event : true  <- jmoise.create_group(team).
```

4.2 Organisational Events

The ***Jason*** programmer may customise several components of the system. In $\mathcal{J}$-$\mathcal{M}$OISE$^+$, we customise the agent architecture that is responsible to link the agent to its environment and other agents. We particularly change the agent perception to include *organisational events*. An agent thus perceives when a group is created, when a scheme is started, when it has an organisational obligation, and so on. For example, the creation of a new group is perceived by the agent with an event like `+group(<GrSpecId>, <GrInstanceId>)`. The agent can therefore handle this event with plans like the following:

```
+group(attack,GId) : true  <- jmoise.adopt_role(leader,GId).
```

In this example, whenever a group from specification `attack` is created, the agent adopts the role *leader* in the group, this new group identification is the value of the variable `GId`. Of course the plan context (`true` in above example) may constrain the role adoption. For instance, in the following plan, the agent only adopts the role in case the group creator is a friend and the agent is capable of being the leader (the terms enclosed in square brackets form the set of annotations of the predicate):

```
+group(attack,GId)[owner(O)]
   :  my_friend(O) & capable(leader)
   <- jmoise.adopt_role(leader,GId).
```

When, for example, a group is removed from the organisational entity, a belief deletion event is generated (they start with `-`). In this case, the event is `-group(<GrSpecId>, <GrInstanceId>)` and it can be handle by plans like:

```
-group(attack,GId) : true
   <- .print("The attack group ",GId," was removed!").
```

The enumeration of some events provided by the organisational architecture is described in Table 3. Among those events, an important one is `+obligation(`si`,` m`)`. This event is generated by the architecture whenever the agent should commit to the mission m in the scheme instance si. The OrgManager notifies the agent architecture about its obligation when it plays a role ρ in a group gi that is responsible for the scheme si and this role is obligated to the mission m that belongs to si (Algorithm 1 is used by OrgManager to identify obligations). In order to program an agent that always commit to its obligations, we can simply add the following plan into the agent code:

```
+obligation(Sch, Mission) : true
   <- jmoise.commit_mission(Mission,Sch).
```

As in the previous example, the context of the plan can be used to constraint the commitment, as in the following example:

```
+obligation(Sch, Mission)[group(GId)]
   :  .my_name(Me) &
```

```
      group(_,GId)[owner(O)] &
      jmoise.link(authority, O, Me)
   <- jmoise.commit_mission(Mission,Sch).
```

The example shows how the authority link can be enforced by the developer at the agent level, since, as pointed out at the end of Section 3.3, at the system level we can not guarantee that this kind of link is followed by the agents. In the above plan, the agent commits to the mission only in case the owner of the group responsible for the scheme has an authority link to the agent. The internal action `jmoise.link` is used to check whether the owner has the link or not.

The agent architecture also generates goal achievement events (represented in AgentSpeak by `+!<goal>`) when an agent's organisational goals become permitted in the current state of a scheme execution. The programmer can thus write plans to deal with these events enabling the agent to achieve its organisational goals. For example, when the goal 'bePlacedInMiddleField' is permitted, since the goal 'getTheBall' was satisfied (see Figure 4), the agent committed to this goal's mission will get the event `+!bePlacedInMiddleField` and a plan like the following may be triggered:

```
+!bePlacedInMiddleField[scheme(Sch)] : true
   <- <the actions to achieve the goal>;
      jmoise.set_goal_state(
           Sch,
           bePlacedInMiddleField,
           satisfied).
```

Note that when the goal is achieved, the agent has to notify the OrgManager. It can thus change the state of the scheme execution and coordinate its execution. If the goal is not achieved, the OrgManager should also be notified. For example, we can use failure events, that were generated when some plan to achieve a goal fails, to inform the OrgManager that some goal was not achieved (failure events are represented by `-!<goal>`):

```
-!bePlacedInMiddleField[scheme(Sch)] : true
   <- jmoise.set_goal_state(
           Sch,
           bePlacedInMiddleField,
           impossible).
```

Figure 7 contains the AgentSpeak code of an agent that plays the *back* role in the defence group. The creation of the groups of the team is performed by the coach agent, so the corresponding code is not included in the Figure. The first plan of this agent adopts the role *back* whenever a group `defence` is created. The second plan starts an attack scheme (Figure 4) when the agent perceives the ball[b]. After creating the scheme, it commits to the mission m_1, since it is permitted to it (see Table 1). The code has one plan for each organisational goal, however the actions to achieve these goals were omitted. When the root goal of the scheme is satisfied, the agent removes its commitment. Finally, when all agents have removed their commitments in the scheme (the number of players is zero), the scheme is finished.

[b]This code is merely illustrative and not the code for a really good player.

Table 3 $\mathcal{J}$-$\mathcal{M}$OISE$^+$ main organisational events.

Event	*Description*
`+/- group(`gt`,`$gi`$`)[owner(`$\alpha$`)]`	perceived by all agents when a gt group is created (event `+`) or removed (event `-`) by the agent α ($gt \in \mathcal{GT}$ and gi identifies an instance group).
`+/- scheme(`st`,`$si`$`)[owner(`$\alpha$`)]`	perceived by all agents when a st scheme is created (`+`) or finished (`-`) by the agent α ($st \in \mathcal{ST}$ and si identifies an instance scheme).
`+ scheme_group(`si`,`$gi`$`)`	perceived by gi players (the agents playing role in gi) when their group becomes responsible for the scheme si.
`+ sch_players(`si`, `$n`$`)`	perceived only by the owner of the scheme si when the number of players changes (the number of players is the number of agents committed to the scheme).
`+ goal_state(`si`, `g`, `$s`$`)`	perceived by si players when the state of the goal g has changed to the state $s \in \{satisfied, impossible\}$.
`+/- play(`α`, `ρ`, `$gi`$`)`	perceived by the agents of group instance gi when an agent α adopts (event `+`) or remove (event `-`) a role in the group.
`+/- commitment(`α`, `m`, `$si`$`)`	perceived by the si players (the agents committed to some mission in si) when an agent α commits or removes a commitment to a mission m in the scheme si.
`+ obligation(`si`, `$m`$`)` `[role(`$\rho$`), group(`$gi`$`)]`	perceived by an agent when it has an organisational obligation for a mission m in the scheme si. It has a role ρ in a group gi responsible for a scheme si and this role is obligated to a mission in the scheme.
`+ permission(`si`, `$m`$`)` `[role(`$\rho$`), group(`$gi`$`)]`	perceived by an agent when it has an organisational permission for a mission m in the scheme si. It has a role ρ in a group gi responsible for a scheme si and this role has permission to the mission in the scheme.

```
// when a defence group is created, adopt
// the role back of Figure 3 in it
+group(defence,GId) :  true
   <- jmoise.adopt_role(back,GId).

// when I see the ball, creates the scheme of Figure 4
// with the team instance as responsible group
+see(ball)
   :  group(team,GId) // get the id of the team group
   <- jmoise.create_scheme(sideAttack, [GId]).

// when I has permission (see Table 1), commit to
// mission m1 of the scheme I have created
+permission(Sch, m1)
   :  .my_name(Me) &
      scheme(sideAttack,Sch)[owner(Me)]
   <- jmoise.commit_mission(m1,Sch).

/* Plans for the organisational goals of mission m1 */

+!getBall[scheme(Sch)] :  true
   <- <actions to bet the ball>;
      jmoise.set_goal_state(Sch,getBall,satisfied).

+!goOpField[scheme(Sch)] :  true
   <- <actions to go to the opponent field>;
      jmoise.set_goal_state(Sch,goOpField,satisfied).

+!kickBall[scheme(Sch)]
   :  // get the agent committed to m2
      commitment(Ag, m2, Sch)
   <- <actions to kick the ball to Ag>;
      jmoise.set_goal_state(Sch,kickBall,satisfied).

// when the root goal of the scheme is satisfied,
// remove my missions
+goal_state(Sch, G[root], satisfied) :  true
   <- jmoise.remove_mission(Sch).

// removes the scheme if it has no more players
// and it was created by me
+sch_players(Sch,0)
   :  .my_name(Me) & scheme(_, Sch)[owner(Me)]
   <- jmoise.remove_scheme(Sch).
```

Figure 7 AgentSpeak code of an agent that plays back.

Thanks to the customisation facilities of ***Jason*** and the AgentSpeak language, it is quite simple to support the organisational programming at the agent level. Although $\mathcal{J}$-$\mathcal{M}$OISE$^+$ enables the agent to perceive and act upon its organisation, the problem of reasoning about the organisation and deciding whether to follow the organisation or not is not covered in this article and remains for future work (a good starting point is the proposal presented in [12]). However, $\mathcal{J}$-$\mathcal{M}$OISE$^+$ is one step towards the solution of this problem in the context of BDI programming with AgentSpeak.

5 Contributions and Future Work

In this article, we described a proposal towards declarative organisational programming both at the agent level (with $\mathcal{J}$-$\mathcal{M}$OISE$^+$) and at the system level (with $\mathcal{S}$-$\mathcal{M}$OISE$^+$). In our proposal, while a BDI based programming language can be used to program those agents, a middleware ensures that the agents will follow some organisational constraints. These constraints are declared by the developer (or even by the agents themselves) according to an organisational model (the $\mathcal{M}$OISE$^+$ model) and a point of view that enables the organisational autonomy. The organisational model used in our proposal enables the declaration of MAS organisational structure (role, groups, links), functioning (global goals, global plans, missions), obligations, and permissions. The main features of this proposal are:

- $\mathcal{S}$-$\mathcal{M}$OISE$^+$ follows a system centred point of view where the organisational specification is interpreted at runtime, it is not hardwired in the agents' code;
- It is suitable for heterogeneous and open systems, since $\mathcal{S}$-$\mathcal{M}$OISE$^+$ is an exogenous approach and therefore does not require a special agent architecture or programming language;
- Despite this openness, a programming support is given in a particular high level declarative language and architecture (AgentSpeak);
- It provides a synchronisation mechanism for scheme execution;
- It is suitable for reorganisation where the declaration of the organisation can dynamically change. Since the organisational specification is available to the agents and they follow it, the fact of simply changing the specification also changes the agents behaviour (as in Figure 1 (d)). We have used this approach both to develop a soccer team that change its $\mathcal{M}$OISE$^+$ organisational at runtime [26] and to specify contract dynamics in an electronic business alliance [27].

Regarding related frameworks, our proposal at the system level is quite complementary to AMELI [16], MadKit [21], and KARMA [33]. Many implementation solutions proposed by these frameworks were adopted in $\mathcal{S}$-$\mathcal{M}$OISE$^+$ (like the OrgBox which is very similar to Teamcore proxy from KARMA and governor from AMELI). Although the general implementation architectures are similar, the underlying organisational model and purpose are different. AMELI has a good support for communication and protocols that $\mathcal{S}$-$\mathcal{M}$OISE$^+$ does not have. However, it does not stress the structural and deontic dimensions like $\mathcal{S}$-$\mathcal{M}$OISE$^+$. MadKit is focused on

the structural dimension and does not include the functional and deontic dimensions. KARMA is concerned with both the structure and the functioning and has an excellent support for coordination of global plan execution, however it lacks an explicit deontic dimension.

Besides the contributions related to the underlying organisational model and the middleware, as far as we known, the $\mathcal{J}$-$\mathcal{M}$OISE$^+$ architecture is the first proposal to bring the organisational resources to the AgentSpeak programming language. Of course, $\mathcal{J}$-$\mathcal{M}$OISE$^+$ is not related to open systems, since it is used only for agents programmed with ***Jason***. Nevertheless the set of tools described in this article ($\mathcal{S}$-$\mathcal{M}$OISE$^+$ combined with $\mathcal{J}$-$\mathcal{M}$OISE$^+$) allows developers to implement an organised MAS both at the system and agent levels.

As a future development, we intend to formalise the model and extend it with new features like communication dimension, detection of violation of an agent obligation, and a sanction system. We also plan to define an organisational meta level, independently of the organisational model adopted, to create a (i) generic ontology of organisational terms and (ii) to provide translation to and from a particular organisational model to other (an initial proposal is presented in [8]). At the agent level, an internal reasoning mechanism that deals with an organisation specification is the next step.

Acknowledgement

Most of the work described in this article was developed during the PhD of the first author [23] and the research was supported by CNPq, CAPES, and FURB. The second author is partially supported by CNPq, grants 304605/2004-2, 482019/2004-2 and 506881/2004-0. The second and third authors are partially supported by USP-COFECUB 98-04. We specially would like to thank to Rafael H. Bordini for valuable suggestions in the integration of $\mathcal{M}$OISE$^+$ with ***Jason***.

References

[1] Rafael H. Bordini, Ana L. C. Bazzan, Rafael O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002)*, pages 1294–1302. ACM Press, 2002.

[2] Rafael H. Bordini, Jomi F. Hübner, and Renata Vieira. ***Jason*** and the Golden Fleece of agent-oriented programming. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms, and Applications*, number 15 in Multiagent Systems, Artificial Societies, and Simulated Organizations, chapter 1. Springer, 2005.

[3] Rafael H. Bordini and Jomi Fred Hübner. BDI agent programming in AgentSpeak using Jason. In Francesca Toni and Paolo Torroni, editors, *Computational*

Logic in Multi-Agent Systems: 6th International Workshop, CLIMA VI, London, UK, volume 3900 of *LNCS*, pages 143–164. Springer, 2006.

[4] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using* **Jason**. Wiley, 2007.

[5] Jean-Pierre Briot and Yves Demazeau, editors. *Principes et architecture des systèmes multi-agents*. Hermes, Paris, 2002.

[6] Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In Toru Ishida, editor, *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS'96)*, pages 41–48. AAAI Press, 1996.

[7] Cristiano Castelfranchi. Modeling social action for AI agents. *Artificial Intelligence*, (103):157–182, 1998.

[8] Luciano Coutinho, Jaime S. Sichman, and Olivier Boissier. Organizational modeling dimensions in multiagent systems. In Jomi F. Hübner and Rafael H. Bordini, editors, *Proceeding of Sixth Iberoamerican Workshop on Multi-Agent Systems (IBERAGENTS 2006)*. 2006.

[9] M. Dastani, B. van Riemsdijk, F. Dignum, and J.J. Meyer. A programming language for cognitive agents: Goal directed 3APL. In *Proc. of the First Workshop on Programming Multiagent Systems: Languages, frameworks, techniques, and tools (ProMAS03)*, volume 3067 of *LNAI*, pages 111–130, Berlin, 2004. Springer.

[10] Mehdi Dastani, Virginia Dignum, and Frank Dignum. Role-assignment in open agent societies. In Jeffrey S. Rosenschein, Tuomas Sandholm, Wooldridge Michael, and Makoto Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2003)*, pages 489–496. ACM Press, 2003.

[11] Maria Virgínia Ferreira de Almeida Júdice Gamito Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Universiteit Utrecht, 2003.

[12] Frank Dignum, David Kinny, and Liz Sonenberg. From desires, obligations and norms to goals. *Cognitive Science Quarterly*, 2(3–4):407–430, 2002.

[13] Virginia Dignum and Frank Dignum. Modelling agent societies: Co-ordination frameworks and institutions. In Pavel Brazdil and Alípio Jorge, editors, *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA'01)*, LNAI 2258, pages 191–204, Berlin, 2001. Springer.

[14] Alexis Drogoul, Bruno Corbara, and Steffen Lalande. MANTA: New experimental results on the emergence of (artificial) ant societies. In Nigel Gilbert and Rosaria Conte, editors, *Artificial Societies: the Computer Simulation of Social Life*, pages 119–221. UCL Press, London, 1995.

[15] Marc Esteva, Juan A. Rodriguez-Aguiar, Carles Sierra, Pere Garcia, and Josep L. Arcos. On the formal specification of electronic institutions. In Frank Dignum and Carles Sierra, editors, *Proceedings of the Agent-mediated Electronic Commerce*, LNAI 1191, pages 126–147, Berlin, 2001. Springer.

[16] Marc Esteva, Juan A. Rodríguez-Aguilar, Bruno Rosell, and Josep L. AMELI: An agent-based middleware for electronic institutions. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2004)*, pages 236–243, New York, 2004. ACM.

[17] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agents systems. In Yves Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press, 1998.

[18] Mark S. Fox, Mihai Barbuceanu, Michael Gruninger, and Jinxin Lon. An organizational ontology for enterprise modeling. In Michael J. Prietula, Kathleen M. Carley, and Les Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, chapter 7, pages 131–152. AAAI Press / MIT Press, Menlo Park, 1998.

[19] B. Gâteau, O. Boissier, D. Khadraoui, and E. Dubois. Controlling an interactive game with a multi-agent based normative organizational model. In *Proc. of the 2nd. International Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (COIN-ECAI 2006)*, 2006.

[20] Norbert Glaser and Philippe Morignot. The reorganization of societies of autonomous agents. In Magnus Boman and Walter Van de Velde, editors, *Multi-Agent Rationality*, LNAI 1237, pages 98–111, Berlin, 1997. Springer.

[21] Olivier Gutknecht and Jacques Ferber. The MadKit agent platform architecture. In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55, 2000.

[22] Carl Hewitt. Open information system semantics for distributed artificial intelligence. *Artificial Intelligence*, (47):79–106, 1991.

[23] Jomi Fred Hübner. *Um Modelo de Reorganização de Sistemas Multiagentes*. PhD thesis, Universidade de São Paulo, Escola Politécnica, 2003.

[24] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. $\mathcal{M}$OISE$^+$: Towards a structural, functional, and deontic model for MAS organization. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002)*, pages 501–502. ACM Press, 2002.

[25] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In Guilherme Bittencourt and Geber L. Ramalho, editors, *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02)*, volume 2507 of *LNAI*, pages 118–128, Berlin, 2002. Springer.

[26] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the $\mathcal{M}$OISE$^+$ for a cooperative framework of MAS reorganisation. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, volume 3171 of *LNAI*, pages 506–515, Berlin, 2004. Springer.

[27] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Specifying E-Alliance contract dynamics through the MOISE+ reorganisation process. In Ana Cristina Bicharra Garcia and Fernando Santos Osório, editors, *Anais do V Encontro Nacional de Inteligência Artificial (ENIA'2005)*, pages 434–443, São Leopoldo, RS, Brasil, 2005. SBC. (best paper).

[28] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. $\mathcal{S}$-$\mathcal{M}$OISE$^+$: A middleware for developing organised multi-agent systems. In Olivier Boissier, Virginia Dignum, Eric Matson, and Jaime Simão Sichman, editors, *Proceedings of the International Workshop on Organizations in Multi-Agent Systems, from Organizations to Organization Oriented Programming in MAS (OOOP'2005)*, volume 3913 of *LNCS*. Springer, 2006.

[29] Carlos Iglesias, Mercedes Garrijo, and José Gonzalez. A survey of agent-oriented methodologies. In *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories*, pages 317–330, Heidelberg, 1999. Springer-Verlag.

[30] Christian Lemaître and Cora B. Excelente. Multi-agent organization approach. In Francisco J. Garijo and Christian Lemaître, editors, *Proceedings of II Iberoamerican Workshop on DAI and MAS*, 1998.

[31] M.V. Nagendra Prasad, Keith Decker, Alan Garvey, and Victor Lesser. Exploring organizational design with TÆMS: A case study of distributed data processing. In Toru Ishida, editor, *Proc. of the 2nd International Conference on Multi-Agent Systems (ICMAS'96)*, pages 283–290. AAAI Press, 1996.

[32] Michael J. Prietula, Kathleen M. Carley, and Les Gasser, editors. *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press / MIT Press, Menlo Park, 1998.

[33] David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, 7(1–2):71–100, 2003.

[34] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perram, editors, *Proc. of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), The Netherlands*, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.

[35] Jaime Simão Sichman, Rosaria Conte, Yves Demazeau, and Cristiano Castelfranchi. A social reasoning mechanism based on dependence networks. In Tony Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 188–192, 1994.

[36] Carles Sierra, Juan Antonio Rodríguez-Aguilar, Pablo Noriega, Marc Esteva, and Josep Lluís Arcos. Engineering multi-agent systems as electronic institutions. *European Journal for the Informatics Professional*, V(4), August 2004.

[37] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Reseearch*, 7:83–124, 1997.

[38] J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Norms in multiagent systems: some implementation guidelines. In *Proceedings of the Second European Workshop on Multi-Agent Systems (EUMAS 2004)*, 2004.

[39] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2002.