$\mathcal{M}$OISE tutorial

(for $\mathcal{M}$OISE 0.7)

Jomi Fred Hübner[1]
Jaime Simão Sichman[2]
Olivier Boissier[3]

[1] Universidade Federal de Santa Catarina
[2] Universidade de São Paulo
[3] École Nationale Supérieure des Mines de Saint-Étienne

2010

Contents

Acronyms

NS	Normative Specification
FS	Functional Specification
MAS	Multi-Agent System
MOISE	Model of Organisation for multI-agent SystEms
OS	Organisational Specification
OE	Organisational Entity
SCH	Social Scheme
SS	Structural Specification

Chapter 1

Introduction

This tutorial describes how to *specify* and *program* a Multi-Agent System (MAS) organisation using the $\mathcal{M}$OISE model. Moreover, this tutorial focus on the utilisation of the $\mathcal{M}$OISE computational tools. It is assumed that the reader knows the $\mathcal{M}$OISE purpose and fundamental concepts (presented in [3, 4, 6], which can be found in the publications directory of $\mathcal{M}$OISE distribution). A complete list of related publications is available at `http://moise.sourceforge.net/related-papers.html` and more detailed documentation is also available with the distribution in the directory `doc`.

1.1 A general view of the soccer example

Throughout the text, a soccer team is used as an example. A soccer team that we will specify is formed by players with roles like goalkeeper, back player, leader, attacker, coach, etc. These role players are distributed in two groups (defense and attack) which form the main group (the team group). The team structure is specified, using the $\mathcal{M}$OISE notation, in the Figure 1.1 (the next chapter will explain the details of the notation and its interpretation).

The team also has some rehearsed plays, one of them is specified by the Social Scheme (SCH) depicted in the Figure 1.2. This scheme has three missions (m_1, m_2, and m_3) and their respective goals (the mission m_3 has, for example, the goals 'be placed in the opponent goal area', 'shot at the opponent's goal', and, a common goal, 'score a goal'). Agents playing these roles may commit to this scheme missions according to Table 1.1.

The structural, functional, and deontic specifications briefly described above form the Organisational Specification (OS) of a soccer team that, for example, 11 players can 'instantiated' building the Organisational Entity (OE) of Table 1.2.

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

Table 1.1: Normative Specification

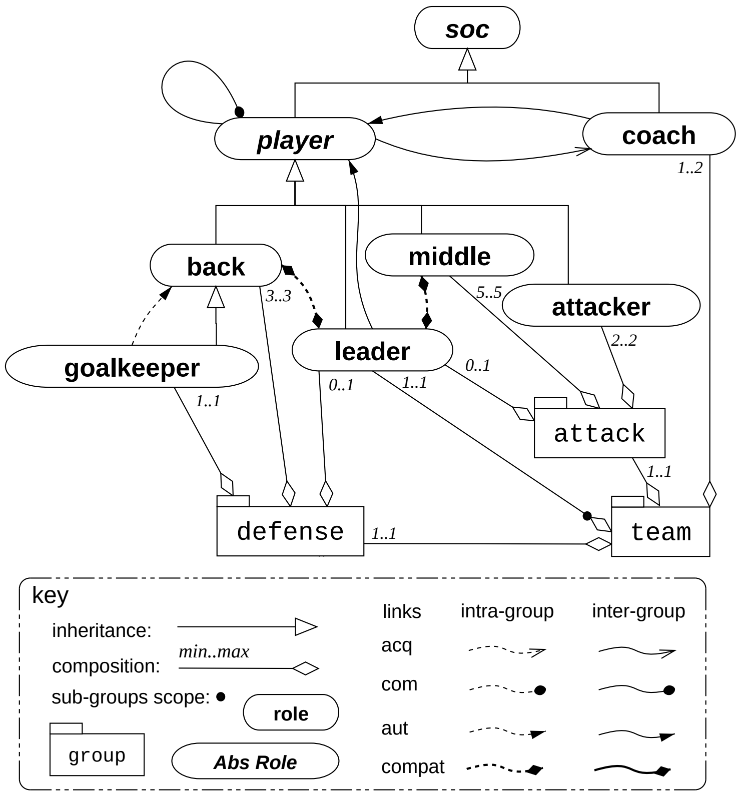
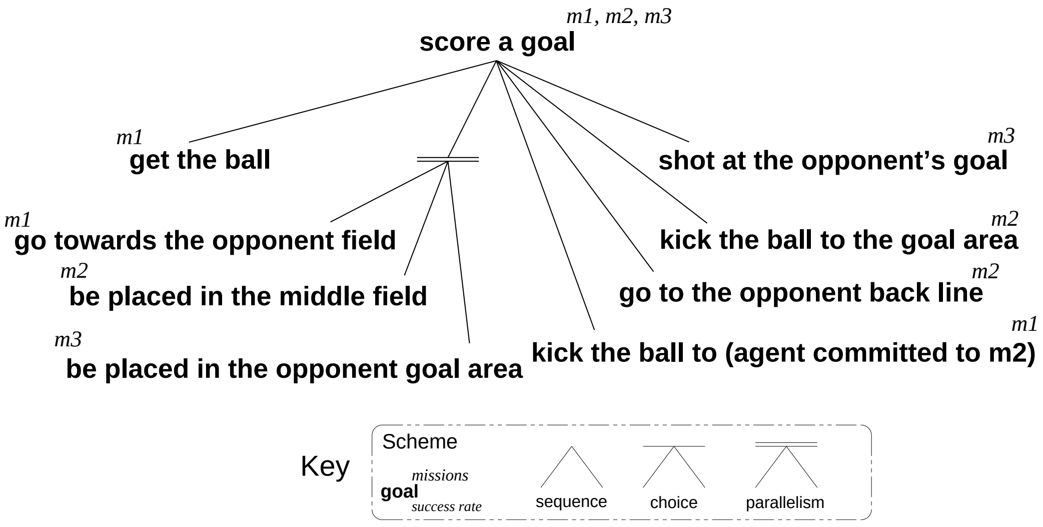


Figure 1.1: The structure of the soccer team



m1, m2, m3
score a goal
m1
get the ball
m3
shot at the opponent's goal
m1
go towards the opponent field
m2
kick the ball to the goal area
m2
be placed in the middle field
m2
go to the opponent back line
m3
be placed in the opponent goal area
m1
kick the ball to (agent committed to m2)
Key
Scheme
missions
goal
success rate
sequence
choice
parallelism

Figure 1.2: A scheme of the soccer team

agent	role	in group
Marcos	goal-keeper	defense
Lucio	back	defense
Edmilson	back	defense
Roque Jr.	back	defense
Cafu	leader and middle	attack
Gilberto Silva	middle	attack
Juninho	middle	attack
Ronaldinho	middle	attack
Roberto Carlos	middle	attack
Ronaldo	attacker	attack
Rivaldo	attacker	attack
Scolari	coach	team

Table 1.2: Organisational Entity

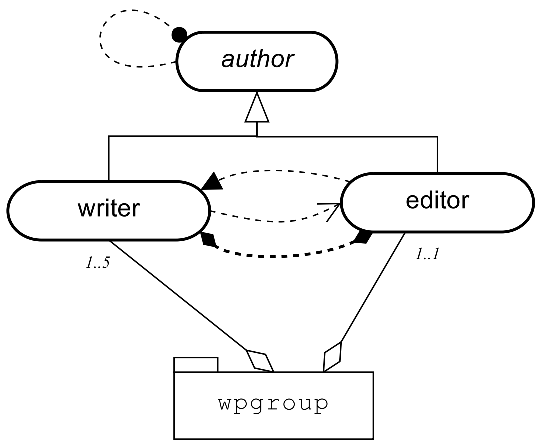
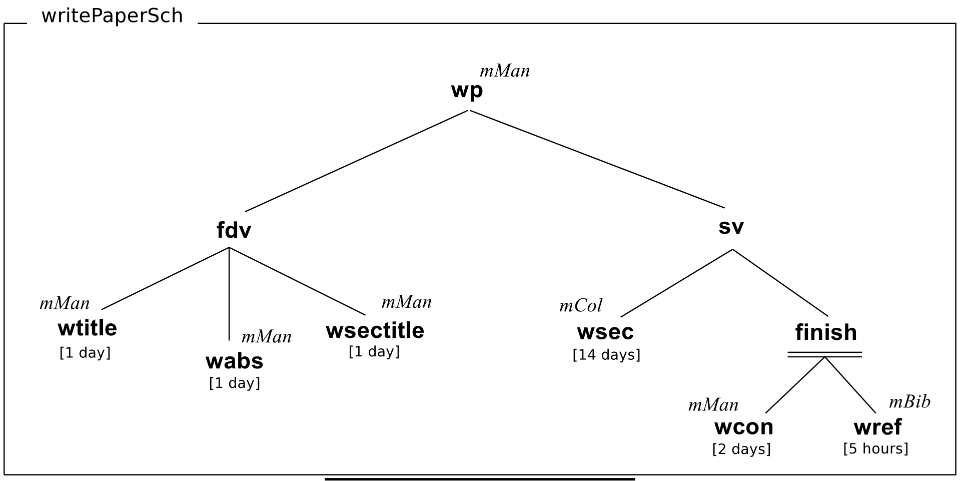


Figure 1.3: The write paper structure

1.2 The writing paper example

Another example used in this document is the 'writing paper', initially described in [7] and then extended in [2]. In this second example, we consider a set of agents that wants to write a paper and therefore define an organisational specification to help them. This MAS structure has one group (`wpgroup`) with two roles, editor and writer, both are sub-role of the author role. The links and cardinalities of this group are specified, using the $\mathcal{M}$OISE notation, in the Figure 1.3. A complete description in XML can be found in the appendix E.2.

To write a paper, they developed a scheme where initially the editor defines a draft version with title, abstract, introduction, and section names, this step is represented by the goal *fdv*. Then the writers 'fill' the gaps of the paper's sections to get a submission version of the paper, represented by the global goal identified by *sv*. This scheme is detailed in Figure 1.4, note that the goals *wcon* (write the conclusion) and *wref* (build the references) can be performed in parallel. There is a mission for the editor ($mMan$), a mission for the writers ($mCol$), and another mission for writers ($mBib$ — get the references for the paper). This relation from roles to missions is specified in Table 1.3. Note



mission	cardinality
$mMan$	1..1
$mCol$	1..5
$mBib$	1..1

Figure 1.4: The write paper scheme

role	deontic relation	mission	TTF
editor	*permission*	$mMan$	
writer	*obligation*	$mCol$	1 day
writer	*obligation*	$mBib$	1 day

Table 1.3: Normative Specification

that some goals have not a mission assigned, in this case, the achievement of the goal depends on the achievement of the sub-goals. For instance, the goal *sv* is achieved with the goals *wsec* and *finish* are achieved.

The structural, functional, and deontic specifications briefly described above form an OS that, for example, 3 agents can 'instantiated' building the OE of Table 1.4.

agent	role	in group	mission
Jaime	editor	wpgroup	$mMan$
Jomi	writer	wpgroup	$mCol$
Olivier	writer	wpgroup	$mCol$
Olivier	writer	wpgroup	$mBib$

Table 1.4: Write paper Organisational Entity

1.3 Structure of the remaining text

The creation of the soccer team entity is initially done in a $\mathcal{M}$OISE OE dynamics simulator (described in Chapter 2). The objective of this simulator is to allow the designer to test the organisational specification by performing actions like agent entrance, role adoption, scheme creation, group creation, etc. No knowledge about any programming language is required to use this simulator, it focus only on the organisational specification and simulation.

To program the agents, appendixes A and B present two tools where agents can be programmed using $\mathcal{M}$OISE concepts. While in the latter tool, the agents are programmed in Java, in the former, the AgentSpeak language is used. Note however that these tools are *outdated*, we are currently using a new platform based on artifacts, called ORA4MAS [3]. The documentation related to this platform is available in the $\mathcal{M}$OISE distribution in the directory `doc/ora4mas`.

Chapter 2

Organisational Entity Dynamics Simulator

This chapter explains both how to write an OS for the $\mathcal{M}$OISE tools and how to use the simulator to test this OS.

2.1 Installation

In order to start using the $\mathcal{M}$OISE tools you need to perform to following steps:

1. check whether the Java 1.5 is installed in your system (the `java` command must be in the PATH);
2. download the $\mathcal{M}$OISE platform from `http://moise.sourceforge.net` and ***Jason*** programming language from `http://jason.sourceforge.net`.
3. uncompress the downloaded files; and
4. test the system by running the script `.../bin/simOE` (or `ant run`). The simulator asks for a specification file, select the `example/tutorial/jojOS.xml` and a screen like Figure 2.1 should appear.

2.2 Structural Specification

The first step to build the Structural Specification (SS) of the soccer team (Figure 1.1) is to write a XML file that describes its structure.[1] The next sections present how this file is composed (all the content of this file is listed in appendix E.1). The focus is both on the XML tags and on some implementation issues.

2.2.1 Role definition (individual level)

The following lines define the role inheritance relation:

[1]The XML files have to follow the XML Schema `src/xml/os.xsd`.

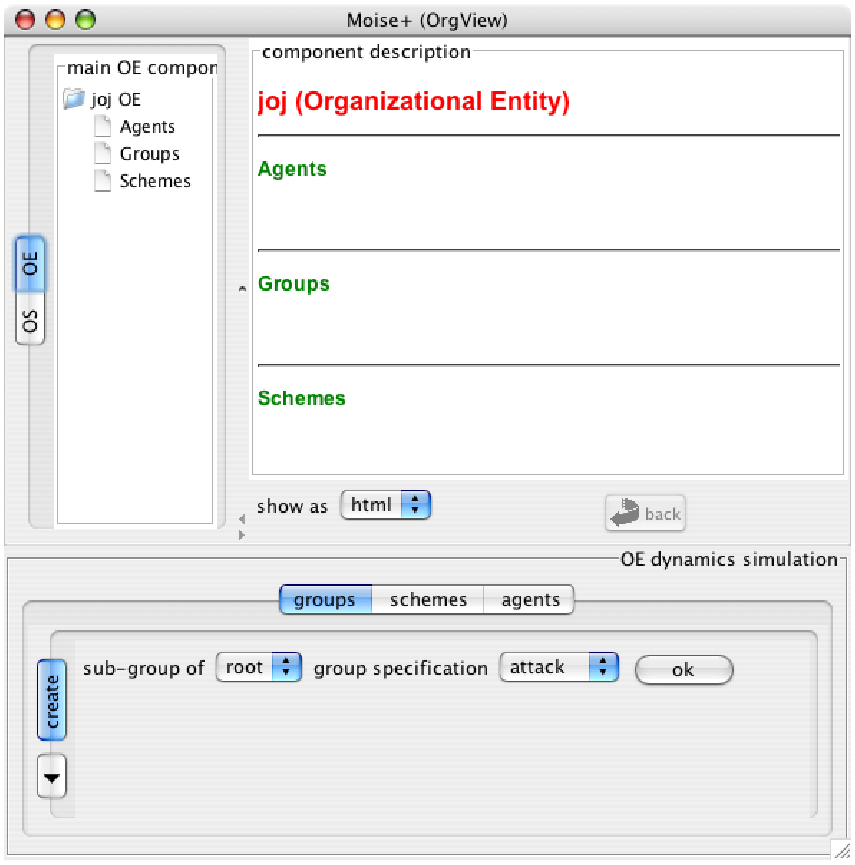


Figure 2.1: Simulator first screen

```
<role-definitions>
  <role id="player" />
  <role id="coach" />
  <role id="middle"> <extends role="player"/> </role>
  <role id="leader"> <extends role="player"/> </role>
  ...
</role-definitions>
```

Remarks.

- The role soc is the root of the role inheritance tree. All roles are sub-roles of soc, even if it is not explicitly specified, as is the case of role coach.
- A role definition does not imply that an agent is allowed to play it, only when the role is added in a group specification it can be played. The roles definition tag is used simply to state the roles hierarchy.
- To state that a role inherits properties from many other roles, the `extends` tag can be used many times, for example:

```
<role id="r1>
  <extends role="r2" />
  <extends role="r3" />
</role>
```

2.2.2 Groups definition (collective level)

A group specification is described inside the tag `group-specification`, for example:

```
<group-specification id="team">
    ...
</group-specification>
```

creates the group specification identified by `team`. Inside the group specification, we can include:

- the allowed *roles* in this group and their cardinality. For example, the following XML code states that one or two agents can play the role coach in the group `team`:

  ```
  <roles>
      <role id="coach" min="1" max="2"/>
  </roles>
  ```

  The cardinality is optional, the default value for min is 0 and for max is 'unlimited'.

  All roles included inside a group must be previously defined inside the `role-definitions` tag.

- the *links* between the group's roles, for example, to state an inter-group authority link from coach to player:

  ```
  <link from="coach"
        to="player"
        type="authority"
        scope="inter-group"
        extends-subgroups="true" />
  ```

  The values for the link `type` are 'authority', 'communication', and 'acquaintance'.

  The values for the `scope` are 'inter-group' and 'intra-group'.

  In case where `extends-subgroups` parameter is *true*, this link is also valid in all `team` subgroups. The default value is *false*.

- the *subgroups* and their cardinality:

  ```
  <subgroups>
    <group-specification id="attack" min="1" max="1">
      ...
    </group-specification>

    <group-specification id="defense" min="1" max="1">
      ...
    </group-specification>
  </subgroups>
  ```

Each subgroups also contains a group specification.

- the *constraints* formation. For example, in the group `attack` there are the following role compatibility:

```
<formation-constraints>
  <compatibility from="middle"
                 to="leader"
                 scope="intra-group"
                 extends-subgroups="false"
                 bi-dir="true"/>
</formation-constraints>
```

  If the `bi-dir` parameter is *true*, the compatibility (or link) also exists from the destination to source.

2.2.3 Organisational Entity creation

Given the XML file described in the previous sections, we can create organisational entities, groups, agents, etc. Notice however that only the structure is specified yet.

To create an OE:

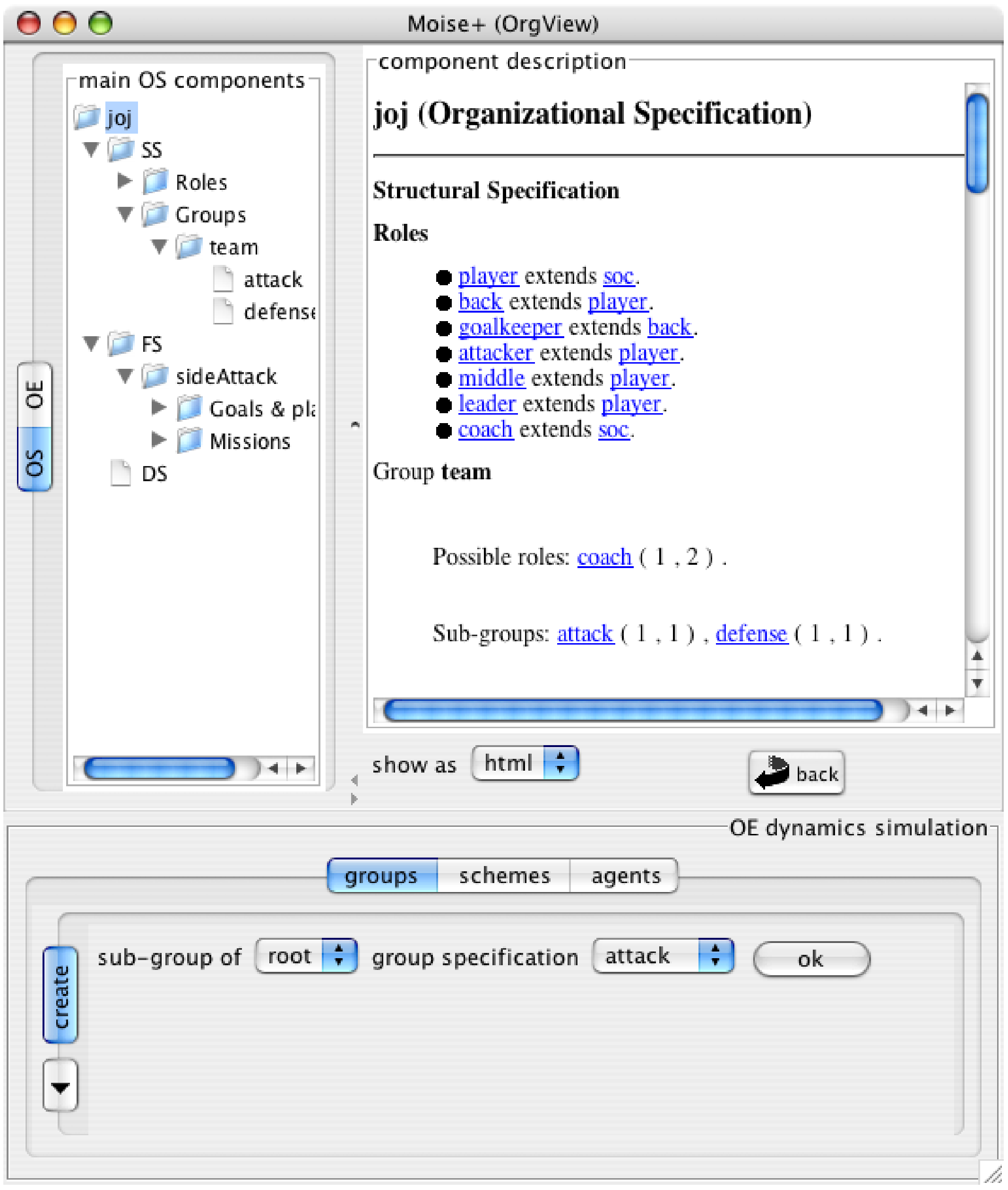
1. Run the `.../bin/simOE` script.
2. Open the soccer team SS file (.../examples/tutorial/jojSS.xml).
3. A screen like Figure 2.1 appears: an OE with an OS but without groups, agents, or schemes.
4. You can navigate through the OS specification clicking on the 'OS' tab and after in the 'joj' tree object (Figure 2.2).

2.2.4 Group creation

1. To create a group, select the 'group'/'create' tab, then select the creation of a 'root' group (some group that is not subgroup of any other group), select the 'team' specification, and finally click on 'ok' button.

   Notice that a new group (id=gr_team0)[2] was created and its well formation status is not Ok (Figure 2.3).
2. Create a `gr_team0` subgroup using the defense specification. Notice that a defense group can only be created as a `team` subgroup, since defense is specified as a `team` subgroup in the SS.
3. Create another `gr_team0` subgroup using the attack specification.

[2]The id of the group is automatically given by the simulator. In the API however, the id of the group may be defined when it is created.



joj (Organizational Specification)

Structural Specification

Roles

- player extends soc.
- back extends player.
- goalkeeper extends back.
- attacker extends player.
- middle extends player.
- leader extends player.
- coach extends soc.

Group **team**

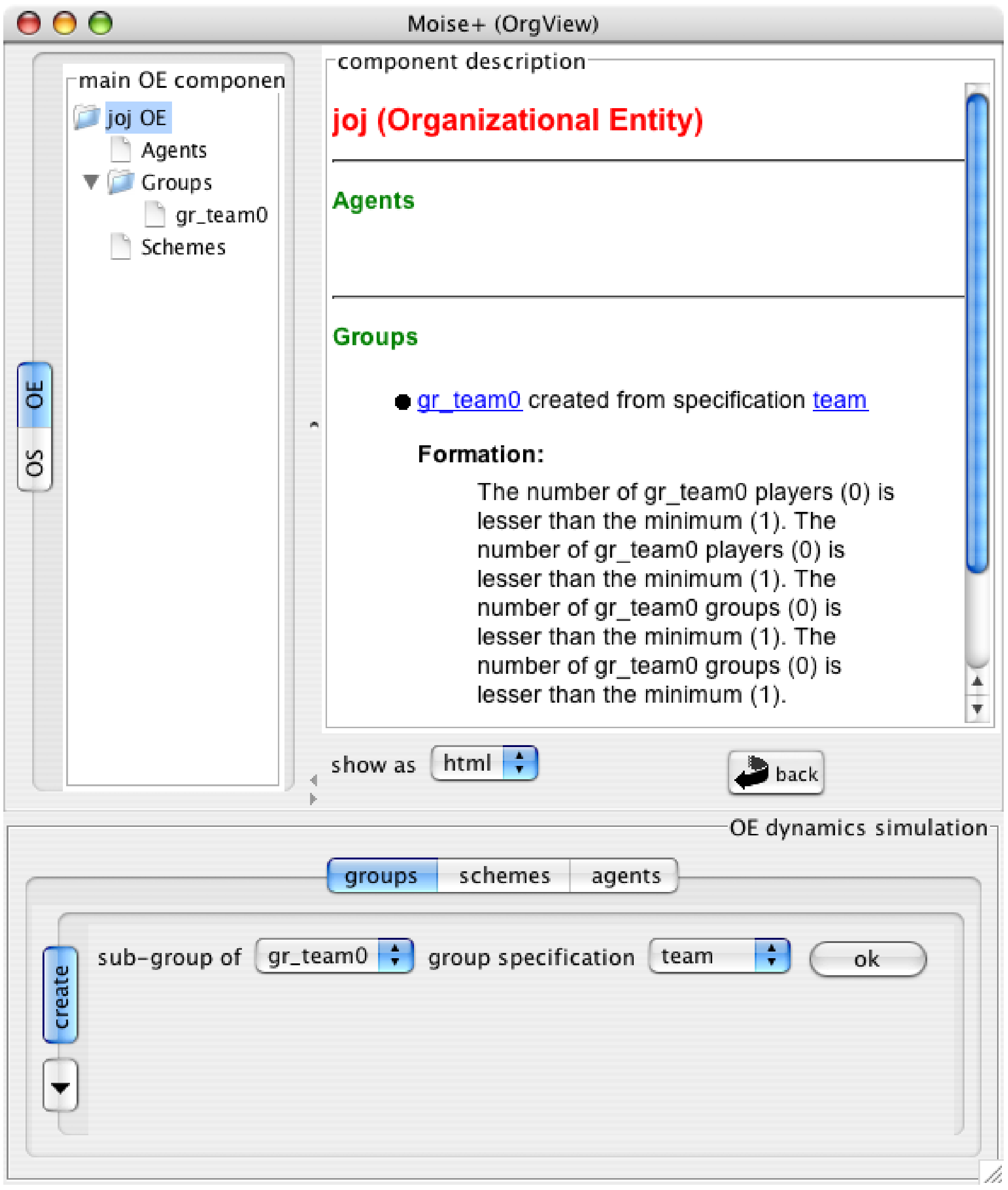
Possible roles: coach (1 , 2) .

Sub-groups: attack (1 , 1) , defense (1 , 1) .

sub-group of: root

group specification: attack

Figure 2.2: Organisational Specification



joj (Organizational Entity)

Agents

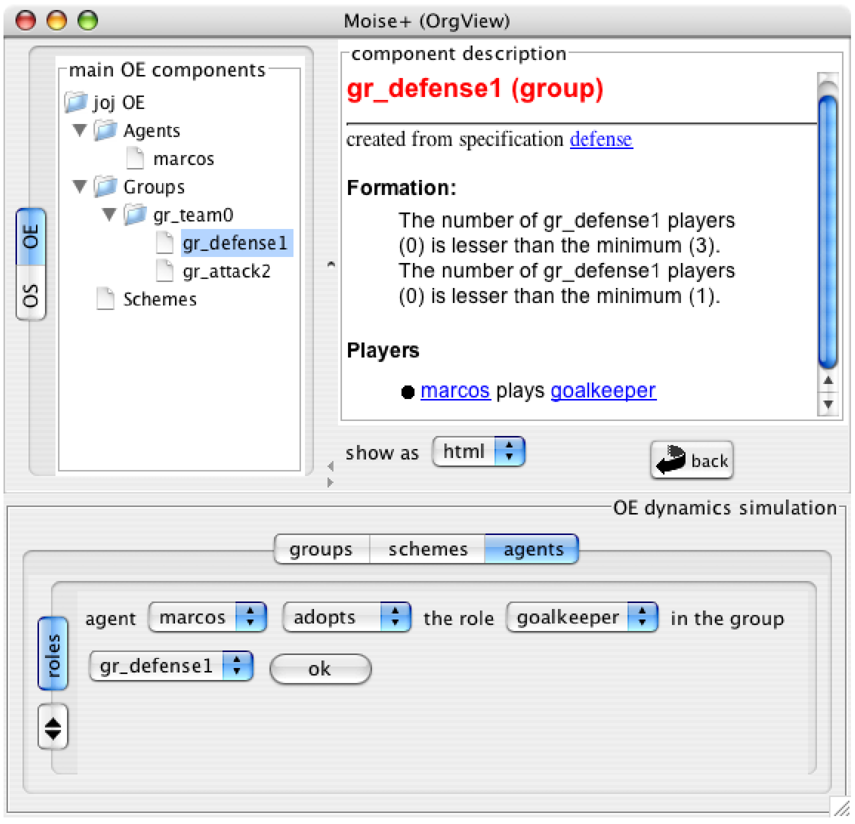
Groups

- gr_team0 created from specification team

Formation:

The number of gr_team0 players (0) is lesser than the minimum (1). The number of gr_team0 players (0) is lesser than the minimum (1). The number of gr_team0 groups (0) is lesser than the minimum (1). The number of gr_team0 groups (0) is lesser than the minimum (1).

Figure 2.3: Result of the first group creation



component description

gr_defense1 (group)

created from specification defense

Formation:

The number of gr_defense1 players (0) is lesser than the minimum (3).
The number of gr_defense1 players (0) is lesser than the minimum (1).

Players

● marcos plays goalkeeper

Figure 2.4: Role adoption result

2.2.5 Agent creation

To create an agent, select the 'agent'/'create' tab, fill 'Marcos' in the agent name field, and click on 'create' button. Do the same for the other agents enumerated at page 6.

2.2.6 Role adoption

To assign roles to agents, as suggested in Table 1.2, select the 'agent'/'roles' tab, select the agent (e.g. Marcos), select the role (e.g. 'goalkeeper'), select the group 'gr_defense1', and click on the 'ok' button (see Figure 2.4). Repeat this operation for the other players' roles and notice how the groups' well formation status is changing.

The role adoption event is constrained by the cardinality and compatibilities of each role. For example, try the following role adoption and notice the error messages:

- Marcos adopts the role back in the group `gr_defense1`.
- Marcos adopts the role back in the group `gr_team0`.
- Edmilson adopts the role back in the group `gr_defense2`.
- Edmilson adopts the role goalkeeper in the group `gr_defense2`.
- Marcos gives up the role goalkeeper in the group `gr_defense2`. No error here, just the well formation has changed.

- Edmilson adopts the role goalkeeper in the group `gr_defense2`. The Edmilson's back role are not intra-group compatible with its goalkeeper role.

The leader role has an interesting property: it has cardinality constraints in three groups. In defense and attack groups this role is optional (cardinality 0..1). In the team group, this role is mandatory (cardinality 1..1), but can not be enacted in this group! Thus, the only way to satisfy the cardinality constraint for the team group is the role leader being played either in its defense or attack subgroups. This definition could be read as 'there must be a leader either in defense or in attack group'. For example, see the `team` well formation status during the following actions:

- The agent Cafu gives up the leader role in the group `gr_attack1`. The team formation becomes not well formed, although its subgroups (defense and attack) are well formed.
- The agent Cafu adopts the leader role in the group `gr_defense2`. This causes an error since the Cafu's middle role in attack is not compatible with the leader role in the defense group. The compatibility between middle and leader is intra-group.
- The agent Cafu adopts the leader role in the group `gr_attack1`. The well formation of the team becomes ok.

2.3 Funcional Specification

In this section we will fill the `functional-specification` tag in order to specify the scheme of the Figure 1.2.

2.3.1 Scheme definition (collective level)

Briefly, a scheme is a global goal decomposition tree. Such a decomposition is done by plans, so the main elements in a scheme specification are plans and goals. For example, the plan for the scheme `sideAttack` is:

```
<scheme id="sideAttack">
 <goal id="scoreGoal" min="1" >
  <plan operator="sequence">
    <goal id="g1" min="1" ds="get the ball" />
    <goal id="g2" min="3" ds="to be well placed">
      <plan operator="parallel">
        <goal id="g7" min="1" ds="go toward the opponent's field" />
        <goal id="g8" min="1" ds="be placed in the middle field" />
        <goal id="g9" min="1" ds="be placed in the opponent's goal area" />
      </plan>
    </goal>
    <goal id="g3" min="1" ds="kick the ball to the m2Ag" >
       <argument id="M2Ag" />
    </goal>
    <goal id="g4"       min="1" ds="go to the opponent's back line" />
    <goal id="g5"       min="1" ds="kick the ball to the goal area" />
    <goal id="g6"       min="1" ds="shot at the opponent's goal" />
  </plan>
 </goal>
 ...
```

In this scheme, `scoreGoal` is the root goal, and this goal is achieved by a plan recursively composed by a sequence of other goals achievement. The `min` attribute of a goal means the number of agents that must satisfy the goal such that it is considered globally achieved. Most of the goals of the scheme should be satisfied by only one agent, but goal `g2` should be satisfied by three agents. The default value for `min` is 'all', meaning that all agents committed to this goal must set is as achieved to state is as globally achieved.

Each goal has thus a unique identification in the scheme and a description. Optionally, a goal can have an argument, e.g. the goal $g3$ has $M2Ag$ as argument. This argument must be assigned to a value in the instance scheme creation.

It is not possible to use more than one operator for a plan, thus in the case of a plan like '$g = g_1, (g_2|g3)$' it is necessary to create two plans and an auxiliary goal:

```
<goal id="g">
  <plan operator="sequence">
    <goal id="g1" />
    <goal id="gaux" >
      <plan operator="choice">
        <goal id="g2" />
        <goal id="g3" />
      </plan>
    </goal>
  </plan>
</goal>
```

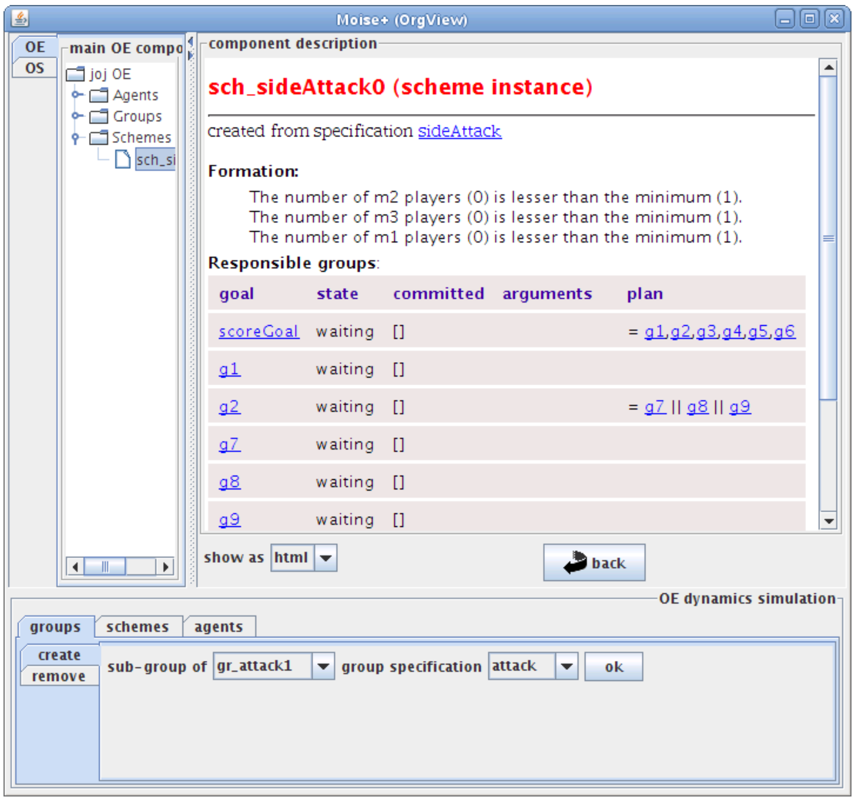
Remarks. Although not used in this example, two types of goals are considered in $\mathcal{M}$oise: achievement and maintenance goals. Achievement goals are the default type and should be declared as satisfied by the agents committed to them when they finished to achieve them. Maintenance goals are not satisfied in a precise moment, they should be pursued while the scheme is running. The agents committed to them do not need to say that they are satisfied.

2.3.2 Mission definition (individual level)

A mission is a set of goals for an agent commitment in the context of a scheme execution. The missions are defined as follow:

```
<scheme id="sideAttack">
  ... the goals ...

  <mission id="m1" min="1" max="1">
    <goal id="scoreGoal" />
    <goal id="g1" />
    <goal id="g3" />
    ...
  </mission>
  ..
</scheme>
```


Moise+ (OrgView)

OE
OS

main OE compo

joj OE
Agents
Groups
Schemes
sch_si

component description

sch_sideAttack0 (scheme instance)

created from specification sideAttack

Formation:

The number of m2 players (0) is lesser than the minimum (1).
The number of m3 players (0) is lesser than the minimum (1).
The number of m1 players (0) is lesser than the minimum (1).

Responsible groups:

goal	state	committed	arguments	plan
scoreGoal	waiting	[]		= g1,g2,g3,g4,g5,g6
g1	waiting	[]		
g2	waiting	[]		= g7 || g8 || g9
g7	waiting	[]		
g8	waiting	[]		
g9	waiting	[]		

show as html

back

OE dynamics simulation

groups schemes agents

create
remove

sub-group of gr_attack1 group specification attack ok

Figure 2.5: Scheme starting

The missions cardinality (the `min` and `max` parameters) state that only one agent can be committed to these missions. The default value for `min` is 0 and for `max` is 'unlimited'.

2.3.3 Scheme creation

Before one can generate scheme related actions, it is necessary to add the functional specification in the XML file and run the .../bin/simOE program again. There is a copy of the FS in the file .../examples/tutorial/jojFS.xml and in the appendix E.1.

To start the scheme, select the `simOE` 'scheme'/'start' tab, choose a scheme specification (there is only one: 'sideAttack'), and click on the 'start' button. An instance scheme (id='sch_sideAttack0') was started as shown in Figure 2.5. Notice the well formation status.

2.3.4 Goal state changes

Each achievement goal of a scheme instance (as shown in the Figure 2.5) has the following dynamic information:

i) state of the goal (possible states and the transitions are represented in Figure 2.6). Every goal is initially *waiting* the conditions to be pursued, when that condition is satisfied its state becomes *enabled*. For example, the goal g_7 is enabled only after the goal g_1 has been satisfied and thus before the g_1 satisfaction g_7 is in a waiting state. The algorithm 1 specify when an goal becomes enabled. In the example of the Figure 2.5 no goal is

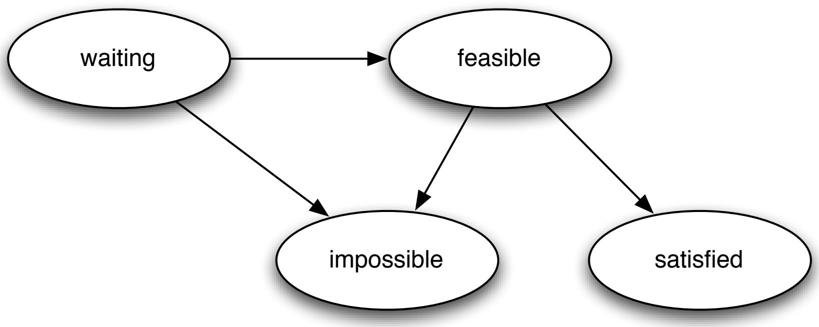


Figure 2.6: Goal states

```
function isEnabled(scheme $sch$, goal $g$)

if $sch$ is not well formed then
    return $false$;
if $g$ has no committed agent then
    return $false$;
if $g$ is the $sch$ root then
    return $true$;
else
    $g$ is in a plan that match '$g_0 = \cdots g \cdots$';
    if $g$ is in a plan that match '$g_0 = \cdots g_i$ , $g \cdots$' then
        if $g_i$ is already satisfied then
            return $true$;
        else
            return $false$;
    else
        return isEnabled($sch$, $g_0$);
```

Algorithm 1: Algorithm to verify possible goals

enabled since the scheme is not well formed yet. When the scheme is well formed, as shown in Figure 2.8, only the goal g_1 is enabled. As soon as g_1 is satisfied, the goals g_7, g_8, and g_9 becomes enabled (Figure 2.9). Once in the enabled state, the agents can pursue the goal and change its state to either satisfied or impossible. Goals without committed agents, pass from the state waiting to the state enabled/satisfied/impossible based on the satisfaction state of its sub-goals.

The state of the goal can be changed in the 'scheme'/'goal state' tab.

ii) committed: a list of agents committed to this goal;

iii) argument (a String): it is the values for the goal arguments (e.g. a value for the argument $M2Ag$ of the goal g_3).

We have developed our soccer team organisation in two independent dimensions of the $\mathcal{M}$OISE model: the structure and the functioning. Since they are not linked yet, we can not create an agent with a role and also a mission. Thus, the next section will explain how to link these two dimensions.

2.4 Normative Specification

The Normative Specification (NS) states both the required roles for missions and missions obligations for roles. The SS (Section 2.2) gives the roles and the Functional Specification (FS) (Section 2.3) gives the missions.

The NS of our example is described in Table 1.1 and its specification in the XML file is simple:

```
<normative-specification>
   <norm id="n1" type="permission" role="back"     mission="m1" />
   <norm id="n2" type="obligation" role="middle"   mission="m2" />
   <norm id="n3" type="obligation" role="attacker" mission="m3" />
</normative-specification>
```

From the point of view of simulation, this new OS allows us to create an agent, assign to it a role (e.g. attacker), and after a mission (e.g. m3). From the point of view of the agent, if it

i) adopts a role (e.g. attacker)

ii) in a group responsible for an instance scheme (e.g. sch_sideAttack0),

iii) this role has obligations for some of this scheme's mission, and

iv) the cardinality of this mission is not satisfied (i.e. the minimum number of agents committed to this mission is not achieved),

then it is obligated to commit to this mission (e.g. m3).

The next sections will exemplify how the scheme well formation status and the agent obligations status may change according to the agents commitments to missions.

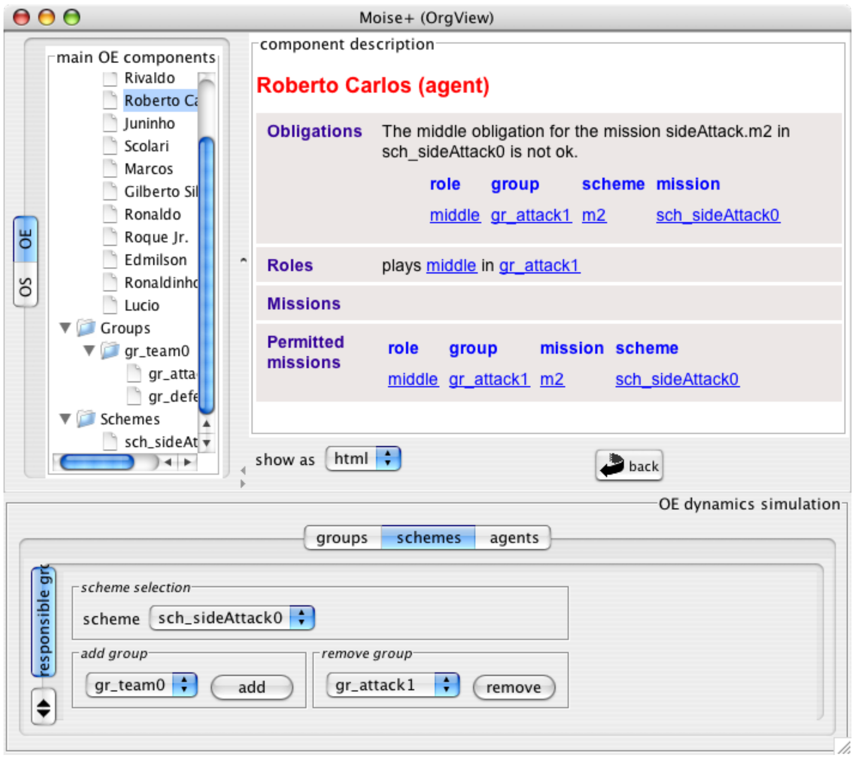
2.4.1 Responsible groups

Each scheme has a set of responsible groups, agents from these groups will perform the scheme. Therefore, only agents from these groups can (or have to) commit to the missions of the scheme. Thus, the first step is to add a responsible group for our scheme `sch_sideAttack0`:

1. run the program …/examples/tutorial/tutorialDS. This program creates an entity that already has the 11 agents, 3 groups, and 1 scheme as we had built in the previous sections.

2. In the OE tree, select the agent Roberto Carlos and notice that its obligations are Ok.

3. Select the 'scheme'/'responsible groups' tab, select the scheme `sch_sideAttack0`, and add the groups `gr_attack1` and `gr_defense2`. Notice how the Roberto Carlos's obligations changed (Figure 2.7).



Roberto Carlos (agent)

Obligations The middle obligation for the mission sideAttack.m2 in sch_sideAttack0 is not ok.

role	group	scheme	mission
middle	gr_attack1	m2	sch_sideAttack0

Roles plays middle in gr_attack1

Missions

Permitted missions

role	group	mission	scheme
middle	gr_attack1	m2	sch_sideAttack0

scheme selection: scheme sch_sideAttack0

add group: gr_team0 add

remove group: gr_attack1 remove

Figure 2.7: Agent obligations status

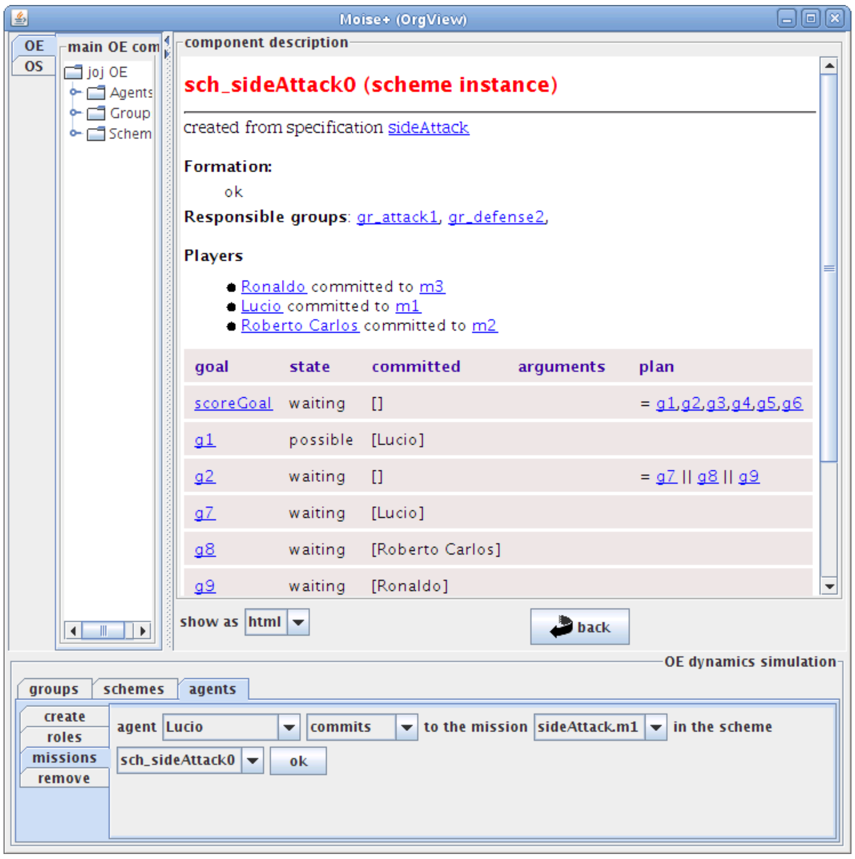
2.4.2 Mission commitment

The commitment to a mission is originated either by and agent's role *obligation* to the mission or by an agent own interest in the mission. In the latter case, the agent must have a role that gives it the *permission* for the mission. For example, the player Roberto Carlos can commit to the mission m2 since its middle role in the `gr_attack1` gives him permission[3] to this mission.

To practice this event, try the following:

- select the 'agent'/'missions' tab, select the agent 'Roberto Carlos', select the mission 'm2', and click on the 'ok' button. Notice that the Roberto Carlos obligation status becomes ok. But the sch_sideAttack0 is not well formed yet.
- Commit the agent Ronaldo to the mission m3.
- Commit the agent Lucio to the mission m1. The sch_sideAttack0 is now well formed (Figure 2.8).
- Try to commit the agent Rivaldo (an attacker) to the mission m1.
- Try to commit the agent Edmilson (a back) to the mission m1.
- Try to commit the agent Marcos (the goal-keeper) to the mission m1. Notice that the Marcos is allowed to the mission m1 since goal-keeper is a back sub-role and back is permitted to m1. Thus the error is about the cardinality of the mission m1.

[3]Indeed, this role gives obligation for the missions, but all obligation is also a permission.



sch_sideAttack0 (scheme instance)

created from specification sideAttack

Formation:
ok

Responsible groups: gr_attack1, gr_defense2,

Players

- Ronaldo committed to m3
- Lucio committed to m1
- Roberto Carlos committed to m2

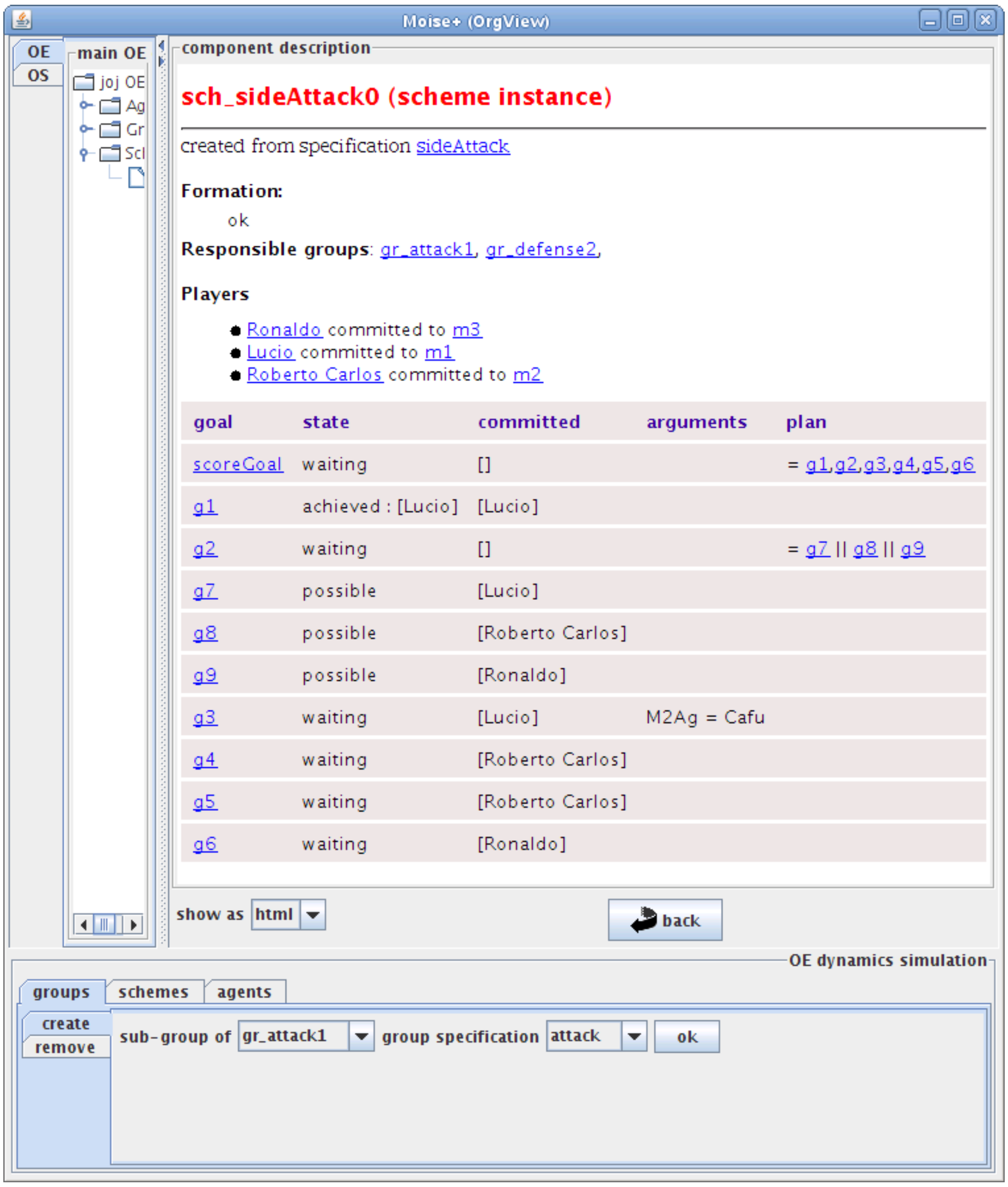
goal	state	committed	arguments	plan
scoreGoal	waiting	[]		= g1,g2,g3,g4,g5,g6
g1	possible	[Lucio]		
g2	waiting	[]		= g7 || g8 || g9
g7	waiting	[Lucio]		
g8	waiting	[Roberto Carlos]		
g9	waiting	[Ronaldo]		

agent Lucio commits to the mission sideAttack.m1 in the scheme sch_sideAttack0 ok

Figure 2.8: Scheme well formed

2.5 Entity de-construction

We have build a well formed OE in the previous sections. Now select the 'group'/'remove' tab and try to remove the gr_team0. You will realize that there are many constraints for the remotion of any OE element. For example, to remove a group, the group must have no players; to remove a group player, the player must play no role in it; to remove a player's role, the role must not be necessary for some player's mission, etc. The Figure 2.10 shows these dependencies.



component description

sch_sideAttack0 (scheme instance)

created from specification sideAttack

Formation:
ok

Responsible groups: gr_attack1, gr_defense2,

Players

- Ronaldo committed to m3
- Lucio committed to m1
- Roberto Carlos committed to m2

goal	state	committed	arguments	plan
scoreGoal	waiting	[]		= g1,g2,g3,g4,g5,g6
g1	achieved : [Lucio]	[Lucio]		
g2	waiting	[]		= g7 || g8 || g9
g7	possible	[Lucio]		
g8	possible	[Roberto Carlos]		
g9	possible	[Ronaldo]		
g3	waiting	[Lucio]	M2Ag = Cafu	
g4	waiting	[Roberto Carlos]		
g5	waiting	[Roberto Carlos]		
g6	waiting	[Ronaldo]		

Figure 2.9: Scheme with goal g_1 achieved

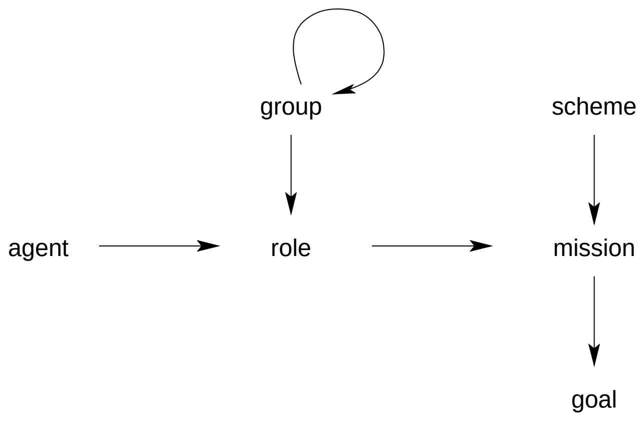

$$\begin{tikzcd}
 & \text{group} \arrow[loop above] \arrow[d] & \text{scheme} \arrow[d] \\
\text{agent} \arrow[r] & \text{role} \arrow[r] & \text{mission} \arrow[d] \\
 & & \text{goal}
\end{tikzcd}$$

Figure 2.10: Dependence for deletion

Appendix A

Developing Organised Agents with $\mathcal{J}$-$\mathcal{M}$OISE$^+$

This chapter describes an example of a simple MAS composed by agents that are aware of its organisation. These agents are developed with $\mathcal{J}$-$\mathcal{M}$OISE$^+$ which is based on ***Jason***, an interpreter used to program BDI agents (`http://jason.sf.net`, [1]). $\mathcal{J}$-$\mathcal{M}$OISE$^+$ is very similar to $\mathcal{S}$-$\mathcal{M}$OISE$^+$ (appendix B) regarding the overall system concepts (e.g. OrgManager and OrgBox components). The main difference is how the agents are programmed, in $\mathcal{S}$-$\mathcal{M}$OISE$^+$ agents are programmed in Java (using a very simple agent architecture), while in $\mathcal{J}$-$\mathcal{M}$OISE$^+$ they are programmed in AgentSpeak, a programming language based on BDI concepts and thus more suitable for agents programming.

> NOTE: The ***Jason*** $\mathcal{M}$OISE$^+$ integration was changed when we moved to ORA4MAS platform. However, the concepts, from an agent perspective, are the same. So we leave the chapter in this tutorial. Refer to `doc/ora4mas` for the current programming proposal.

The next section describes how we have customised the ***Jason*** agent architecture to enable agents to perceive and reason about its organisation. It is described from an user point of view and no implementation issues are therefore given (a more detailed description of $\mathcal{J}$-$\mathcal{M}$OISE$^+$ was published in [6]). The Section A.2 exemplifies the use of $\mathcal{J}$-$\mathcal{M}$OISE$^+$ in the application described in Section 1.2.

A.1 Organisational agent architecture in ***Jason***

In $\mathcal{J}$-$\mathcal{M}$OISE$^+$ an agent changes its organisation using organisational actions and perceives it back by organisational events.

A.1.1 Organisational actions

The overall proposal is based on the addition of a special agent called OrgManager that maintains the current organisational entity (OE) state [5] (see Figure A.1). The agents then may send messages to OrgManager, using ***Jason*** communication acts, to produce *actions* that change the OE. For example, to

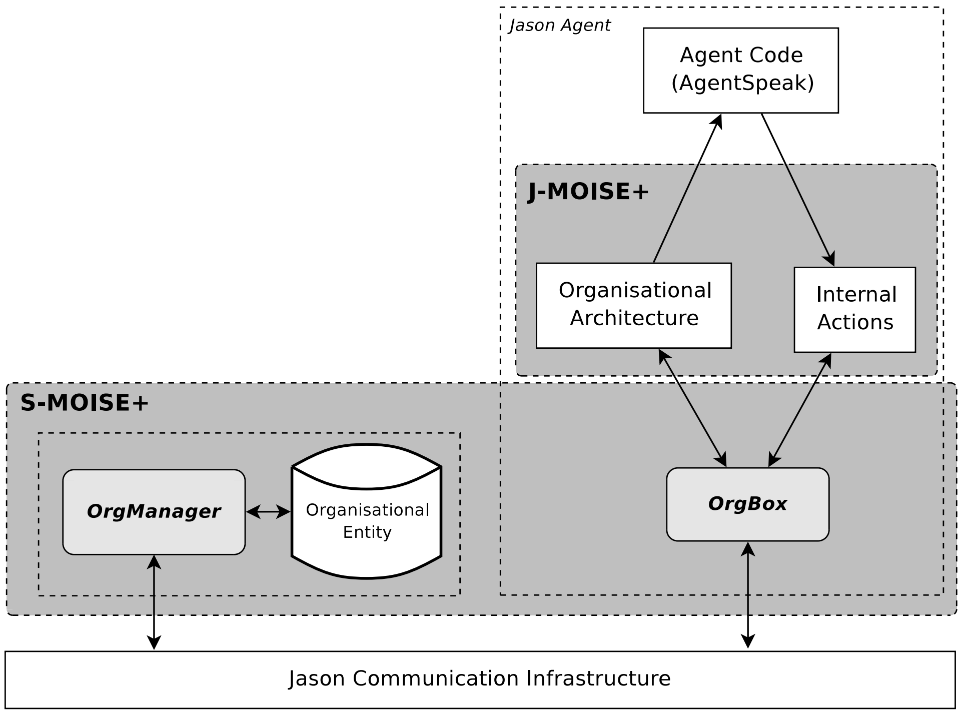


Figure A.1: General view of the $\mathcal{J}$-$\mathcal{M}$OISE$^+$ architecture

create a new group from the specification `wpgroup` (see Section 1.2), an agent should send an achieve message with content `create_group(wpgroup)` to the agent called orgManager[1]:

```
+some_event : true
   <- .send(orgManager, achieve, create_group(wpgroup)).
```

It is also possible to use an organisational action in the plan using the $\mathcal{J}$-$\mathcal{M}$OISE$^+$ internal actions that sends the corresponding message to OrgManager (these actions start with `jmoise.`), for example:

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

The actions that OrgManager can handle are[2] [3]:

- `create_roup(<GrSpecId>[,<GrId>])`: creates a new group instance based on GrSpecId specification. To create a subgroup, the super group identification must be informed as the second argument.[4]
- `remove_group(<GrId>)`: removes the group identified by GrId, this groups must be empty (no players) to be removed.
- `create_scheme(<SchSpecId> [, <ListOfRespGr>])`: creates a new scheme instance based on scheme specification identified by SchSpecId. If the second optional parameters is used, the initial set of responsible groups of the new scheme is defined by ListOfRespGr.

[1] `.send` is the ***Jason*** internal action used to send messages to another agent.
[2] These actions correspond to the organisational actions described in Chapter 2.
[3] A detailed explanation and examples are found in API documentation of $\mathcal{J}$-$\mathcal{M}$OISE$^+$.
[4] More arguments can be used to define the identification of the new group or obtain the automatically given identification (see API for more information about these arguments).

- `add_responsible_group(<SchId>,<GrId>)`: add the group GrId as responsible group for scheme SchId.
- `remove_scheme(<SchId>)`: removes the scheme SchId from the OE.
- `abort_scheme(<SchId>)`: removes the scheme SchId from the OE (does not requires that the scheme has no players).
- `set_goal_arg(<SchId>,<Goal>,<ArgId>,<value>)`: set an argument's value for some goal in a scheme.
- `set_goal_state(<SchId>,<Goal>,(satisfied|impossible))`
- `adopt_role(<RoleId>,<GrId>)`: adopts the role RoleId in the group GrId.
- `remove_role(<RoleId>,<GrId>)`: removes the role RoleId in group GrId.
- `commit_mission(<MisId>,<SchId>)`: commit the agent to mission MisId in scheme SchId.
- `remove_mission([<MisId>,] <SchId>)`: if MisId is not informed, all missions in the SchId will be removed.
- `broadcast( <GrpId/SchId>, P, C)`: broadcast a message with content C and performative P to all agents of a group or scheme.

You can see the API documentation for a complete list of actions, more details, and examples.

A.1.2 Organisational events

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

```
+group(wpgroup,Id) : true
   <- jmoise.adopt_role(writer,Id).
```

In this example, whenever a group from specification `wpgroup` is created, the agent adopts the role `writer`. 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 its friend

```
+group(wpgroup,Id)[owner(O)] : my_friend(O)
   <- jmoise.adopt_role(writer,Id).
```

The events which start with + represent a belief addition. However when, for example, a group is removed from the organisational entity, a belief deletion event is generated. In this case, the event is `-group(<GrSpecId>, <GrId>)` and it can be handle by plans like:

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

The events perceived by the agent are the following:

- `+/- group(<GrSpecId>,<GrId>)[owner(<AgName>)]`: perceived by all agents when a group is created (event `+`) or removed (event `-`) by `AgName`.
- `+/- play(<AgName>, <RoleId>, <GrId>)`: perceived by the agents of `GrId` when an agent adopts (event `+`) or remove (event `-`) a role in group `GrId`.
- `+/- commitment(<AgName>, <MisId>, <SchId>)`: perceived by the `SchId` players when an agent commits or removes a commitment to a mission `MisId` in scheme `SchId`.
- `+/- scheme(<SchSpecId>,<SchId>)[owner(<AgName>)]`: perceived by all agents when a scheme is created (`+`), finished (`-`), or aborted (`-`) by `AgName`.
- `+ scheme_group(<SchId>,<GrId>)`: perceived by `GrId` players when this group becomes responsible for the scheme `SchId`.
- `+ sch_players(<SchId>,<NumberOfPlayers>)`: perceived only by the owner of the scheme when the number of players changes (agents commit or remove a commitment in the scheme).
- `+ goal_state(<SchId>, <GoalId>, <State>)`: perceived by `SchId` players when the state of some goal changes.
- `+/- obligation(<SchId>, <MisId>)[role(<RoleId>), group(<GrId>)]`: perceived by an agent when is has an organisational obligation for a mission. It has a role (`RoleId`) in a group (`GrId`) responsible for a scheme (`SchId`) and this role is obligated to a mission in this scheme.
- `+/- permission(<SchId>, <MisId>)[role(<RoleId>), group(<GrId>)]`: perceived by an agent when is has an organisational permission for a mission. It has a role (`RoleId`) in a group (`GrId`) responsible for a scheme (`SchId`) and this role has permission to a mission in this scheme.

The agent architecture also generate goal achievement events when an agent's organisational goals becomes possible in the current state of the scheme execution. The programmer can thus write plans to deal with these events to enable to agent to achieve its organisational goals. For example, when the goal to write the paper conclusion is permitted, the following plan will be executed:

```
+!wconc[scheme(Sch)] : true
   <- .print("Writing the conclusion!");
      jmoise.set_goal_state(Sch, wconc, satisfied).
```

The `[scheme(Sch)]` in the plan's trigger event represents a set of annotations of the goal (only one annotation in this case). Differently than arguments (enclosed by '(' and ')'), annotations may not be included in the predicate. The above plan can thus simply be:

```
+!wconc : true
   <- .print("Writing the conclusion!");
      // obtain the scheme id from the belief base
      ?scheme(writePaperSch, Sch);
      jmoise.set_goal_state(Sch, wconc, satisfied).
```

Of course, in place of printing a message the plan should have action that achieve the goal. Note that when the goal is achieved, the agent have to notify the OrgManager, so it can change the state of the scheme execution and coordinate its execution. If the goal is not achieved, the OrgManager also have to be notified, for instance:

```
+!wconc[scheme(Sch)] : true
   <- .print("Writing the conclusion!");
      jmoise.set_goal_state(Sch, wconc, satisfied).
// the plan to achieve the goal failed
-!wconc[scheme(Sch)] : true
   <- jmoise.set_goal_state(Sch, wconc, impossible).
```

Other annotations of goals events are:

- `mission(MissionId)`: the mission of the goal;
- `type(Type)`: the type of the goal (achievement or maintenance);
- `source(orgManager)`: the source of the goal, always the orgManager in case of organisational goals;
- `role(RoleId)`: the role assigned to the mission of the goal;
- `group(GrpId)`: the group where the role is being played.

The plan may add these annotation if the corresponding information is necessary for the execution of the plan, for example:

```
+!wconc[scheme(Sch), role(R), group(G)] : true
   <- .print("Writing the conclusion for the scheme ",Sch);
      .print("because I play ",R," in group ",G);
      ...
```

A.2 The writing paper agents in AgentSpeak

In this section, the write paper application (as presented in Section 1.2) is implemented using AgentSpeak and ***Jason***.

A.2.1 MAS2J configuration

Projects in ***Jason*** are defined in a `.mas2j` configuration file. In this file we set which agents will run and some parameters for these agents. The write paper application is composed by the following agents:

- OrgManager: this agent is the same as presented in the $\mathcal{S}$-$\mathcal{M}$OISE$^+$, but customised to provide organisational services in ***Jason*** systems.
- Jaime: this agent will play the editor role in the application.
- Olivier: this agent will play the writer role.
- Jomi: this agent will also play the writer role.

All agents have a customised architecture (as depicted in Figure A.1) that bind them to the organisational infrastructure. The OrgManager has two special parameters: the XML file with the organisational specification and whether the a graphical interface will be shown. In the specification of the organisation, the user must start all identifiers with **lower case** characters, since identifiers that start with upper case are considered as variables in AgentSpeak.

```
MAS write_paper {
   infrastructure: Centralised
   agents:
       orgManager [osfile="wp-os.xml",gui=yes]
                  agentArchClass jmoise.OrgManager;

       jaime      agentArchClass jmoise.OrgAgent;
       olivier    agentArchClass jmoise.OrgAgent;
       jomi       agentArchClass jmoise.OrgAgent;

   classpath: "../../lib/moise.jar";"../../lib/jmoise.jar";

   aslSourcePath: ".";"../../src/asl";
}
```

A.2.2 Jaime's code

```
/*  Beliefs */

// I want to play "editor" in "wpgroups"
// (this belief is used by the moise common plans included below)
desired_role(wpgroup,editor).

// I want to commit to "mManager" mission in "writePaperSch" schemes
desired_mission(writePaperSch,mManager).


/* Initial goals */

!create_group. // initial goal

// create a group to write a paper
+!create_group : true
   <- //.send(orgManager, achieve, create_group(wpgroup)).
      jmoise.create_group(wpgroup,G);
      .print("Group ",G," created").
-!create_group[error_msg(M),code(C),code_line(L)]
   <- .print("Error creating group, command: ",C,", line ",L,", message: ",M).


/* Organisational Events  */

/* Structural events */

// when I start playing the role "editor",
// create a writePaper scheme
+play(Me,editor,GId)
   :  .my_name(Me)
   <- jmoise.create_scheme(writePaperSch, [GId]).


/* Functional events */

// when a scheme has finished, start another
-scheme(writePaperSch,_)
   :  group(wpgroup,GId)
   <- jmoise.create_scheme(writePaperSch, [GId], SchId);
      .print("Scheme ",SchId," created").

// include common plans for MOISE+ agents
{ include("moise-common.asl") }


/* Organisational Goals' plans */

+!wtitle[scheme(Sch)] : true
   <- .print("Writing title!");
      jmoise.set_goal_state(Sch,wtitle,satisfied).

+!wabs[scheme(Sch)] : true
   <- .print("Writing abstract!");
      jmoise.set_goal_state(Sch,wabs,satisfied).

+!wsectitles[scheme(Sch)] : true
   <- .print("Writing section titles!");
      jmoise.set_goal_state(Sch,wsectitles,satisfied).

+!wconc[scheme(Sch)] : true
   <- .print("Writing conclusion!");
      jmoise.set_goal_state(Sch,wconc,satisfied).

+!wp[scheme(Sch)] : true
   <- .print("***** FINISH! *****");
      jmoise.set_goal_state(Sch,wp,satisfied).
```

The included file (`moise-common.asl`) is:

```
// Common plans for organised agents based on MOISE+ model.
//
// These plans use the beliefs:
//     . desired_role(<GrSpec>,<Role>) and
//     . desired_mission(<SchSpec>,<Mission>).

/*
   Organisational Events
   --------------------
*/

/* Structural events */
// when a group is created and I desire to play in it,
// adopts a role
+group(GrSpec,Id)
   :  desired_role(GrSpec,Role)
   <- jmoise.adopt_role(Role,Id).


/* Functional events */

// finish 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).
*/

/* Deontic events */

// when I have an obligation or permission to a mission
// and I desire it, commit
+obligation(Sch, Mission)
   :  scheme(SchSpec,Sch) & desired_mission(SchSpec, Mission)
   <- jmoise.commit_mission(Mission,Sch).
+permission(Sch, Mission)
   :  scheme(SchSpec,Sch) & desired_mission(SchSpec, Mission)
   <- jmoise.commit_mission(Mission,Sch).

// when the root goal of the scheme is achieved,
// remove my missions and the scheme
+goal_state(Sch, _[root], achieved)
   <- jmoise.remove_mission(Sch);
      .my_name(Me);
      if (scheme(_,Sch)[owner(Me)]) {
         if (not sch_players(Sch,0)) {
            .wait( { +sch_players(Sch,0)} , 1000, _)
         };
         jmoise.remove_scheme(Sch)
      }.

// if some scheme is finished, drop all intentions related to it.
-scheme(_Spec,Id)
   <- .drop_desire(_[scheme(Id)]).

+error(M)[source(orgManager)]
   <- .print("Error in organisational action: ",M); -error(M)[source(orgManager)].
```

A.2.3 Olivier's code

```
/* Beliefs */

refs([boissier04,sichman03]). // refs used in the paper

// I want to play "writer" in "wpgroups"
desired_role(wpgroup,writer).
```

```
// I want to commit to "mColaborator" and "mBib" missions
// in "writePaperSch" schemes
desired_mission(writePaperSch,mColaborator).
desired_mission(writePaperSch,mBib).



// include common plans for MOISE+ agents
{ include("moise-common.asl") }

/* Organisational Goals' plans */

// a generic plan for organisational goals (they have scheme(_) annotation)
+!X[scheme(Sch)] : true
   <- .print("Doing organisational goal ",X, " in scheme ",Sch);
      jmoise.set_goal_state(Sch,X,satisfied).

// when I receive a tell message from S and
// S plays writer in a scheme, change the belief of
// used refs
+use_ref(NewRef)[source(S)]
   :  play(S, writer, _) & refs(R)
   <- .print("adding ref ",NewRef, " to ", R);
      -refs(R); +refs([NewRef|R]).

+play(Me,R,GrInst)
   : .my_name(Me) & group(GrSpec,GrInst)
  <- jmoise.group_specification(GrSpec,Roles);
     .member(role(R,Min,Max,Compat,Links),Roles);
     .print("I am starting playing ",R);
     .print(" -- cardinality of my role (Min,Max): (",Min,",",Max,")");
     .print(" -- roles compatible with mine: ", Compat);
     .print(" -- all roles of the group are ",Roles).
```

A.2.4 Jomi's code

```
/* Beliefs */

// I want to play "writer" in "wpgroups"
desired_role(wpgroup,writer).

// I want to commit to "mColaborator" mission in "writePaperSch" schemes
desired_mission(writePaperSch,mColaborator).

// include common plans for MOISE+ agents
{ include("moise-common.asl") }


/* Organisational Goals' plans */

+!wsecs[scheme(Sch)]
   :   commitment(Ag, mBib, Sch)
   <- // send a message to the agent committed to mission mBib
      .send(Ag, tell, use_ref(bordini05));
      .print("Writing sections!");
      jmoise.set_goal_state(Sch, wsecs, satisfied).
```

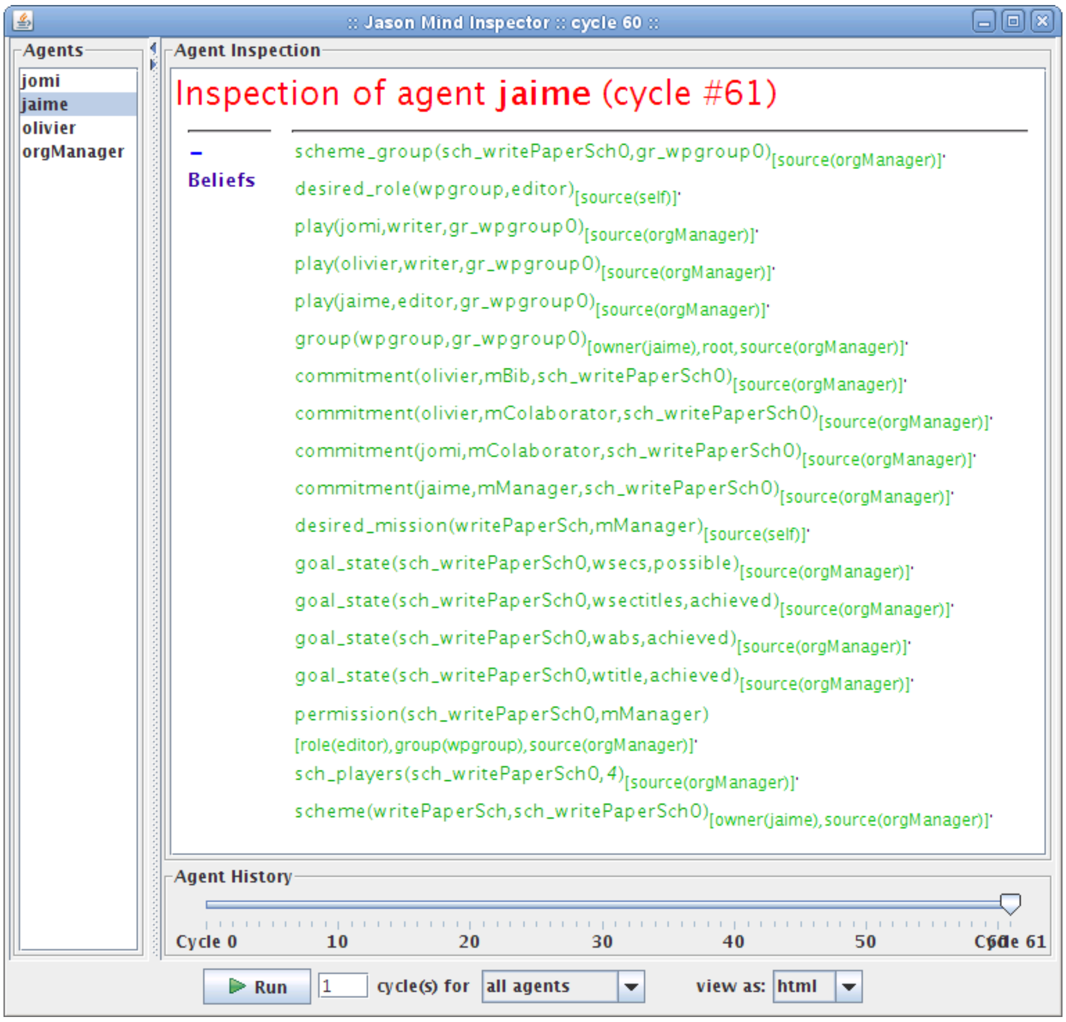
```
// the plan to achieve the goal failed
-!wsecs[scheme(Sch)] : true
   <- jmoise.set_goal_state(Sch, wsecs, impossible).
```

A.2.5 Execution

```
[jaime]   Writing title!
[jaime]   Writing abstract!
[jaime]   Writing section titles!
[jomi]    Writing sections!
[olivier] Doing organisational goal wsecs in scheme sch_writePaperSch0
[jaime]   Writing conclusion!
[olivier] Doing organisational goal wrefs in scheme sch_writePaperSch0
[jaime] ***** FINISH! *****
... continue with next paper ...
```

A.2.6 Screen shots

The following screen is the mind of the Jaime agent after the execution of the write paper scheme.

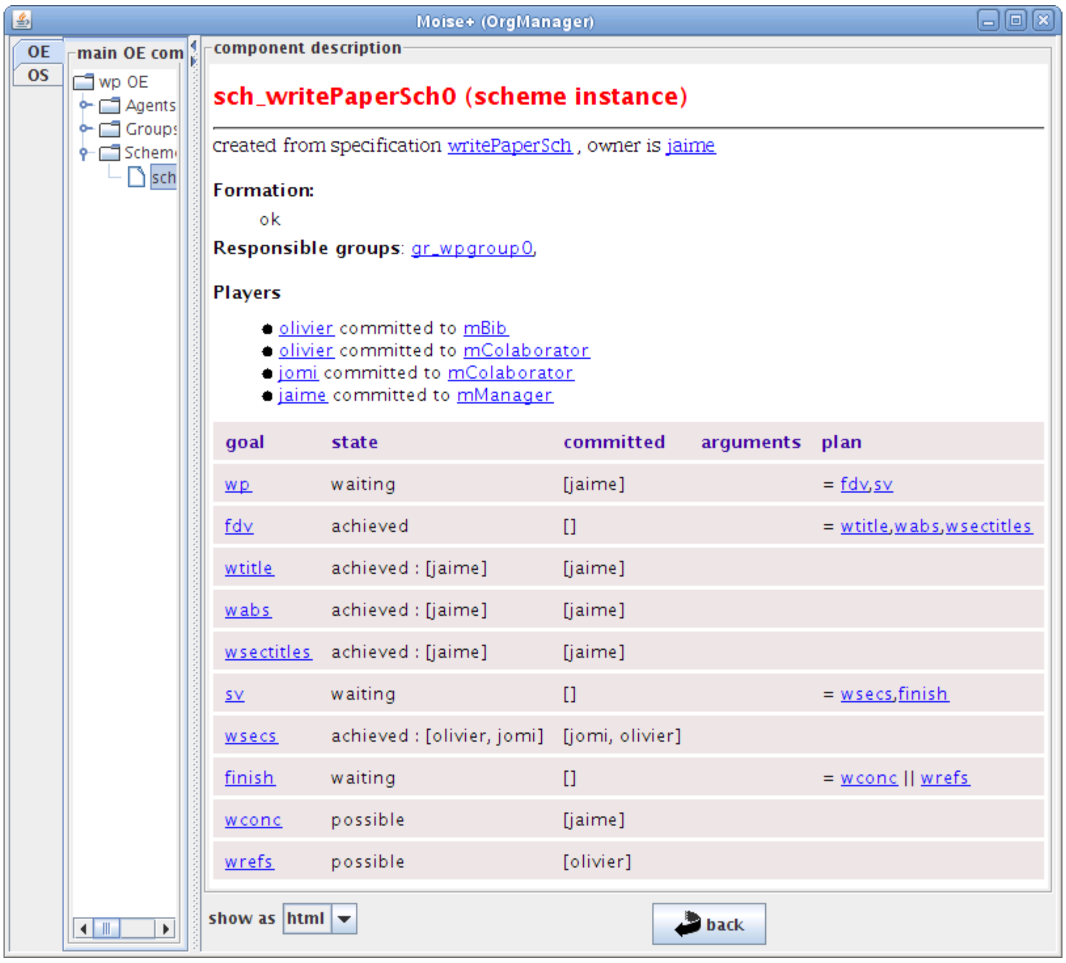


Inspection of agent jaime (cycle #61)

Beliefs

```
scheme_group(sch_writePaperSch0,gr_wpgroup0)[source(orgManager)].
desired_role(wpgroup,editor)[source(self)].
play(jomi,writer,gr_wpgroup0)[source(orgManager)].
play(olivier,writer,gr_wpgroup0)[source(orgManager)].
play(jaime,editor,gr_wpgroup0)[source(orgManager)].
group(wpgroup,gr_wpgroup0)[owner(jaime),root,source(orgManager)].
commitment(olivier,mBib,sch_writePaperSch0)[source(orgManager)].
commitment(olivier,mColaborator,sch_writePaperSch0)[source(orgManager)].
commitment(jomi,mColaborator,sch_writePaperSch0)[source(orgManager)].
commitment(jaime,mManager,sch_writePaperSch0)[source(orgManager)].
desired_mission(writePaperSch,mManager)[source(self)].
goal_state(sch_writePaperSch0,wsecs,possible)[source(orgManager)].
goal_state(sch_writePaperSch0,wsectitles,achieved)[source(orgManager)].
goal_state(sch_writePaperSch0,wabs,achieved)[source(orgManager)].
goal_state(sch_writePaperSch0,wtitle,achieved)[source(orgManager)].
permission(sch_writePaperSch0,mManager)
[role(editor),group(wpgroup),source(orgManager)].
sch_players(sch_writePaperSch0,4)[source(orgManager)].
scheme(writePaperSch,sch_writePaperSch0)[owner(jaime),source(orgManager)].
```

The screen below is the status of the write paper scheme in the end of the execution.



sch_writePaperSch0 (scheme instance)

created from specification writePaperSch , owner is jaime

Formation:

ok

Responsible groups: gr_wpgroup0,

Players

- olivier committed to mBib
- olivier committed to mColaborator
- jomi committed to mColaborator
- jaime committed to mManager

goal	state	committed	arguments	plan
wp	waiting	[jaime]		= fdv,sv
fdv	achieved	[]		= wtitle,wabs,wsectitles
wtitle	achieved : [jaime]	[jaime]		
wabs	achieved : [jaime]	[jaime]		
wsectitles	achieved : [jaime]	[jaime]		
sv	waiting	[]		= wsecs,finish
wsecs	achieved : [olivier, jomi]	[jomi, olivier]		
finish	waiting	[]		= wconc || wrefs
wconc	possible	[jaime]		
wrefs	possible	[olivier]		

Appendix B

Developing Distributed Organised Agents with $\mathcal{S}$-$\mathcal{M}$OISE$^+$

This chapter describes an example of a simple MAS composed by distributed agents that follows an organisational specification. These agents are developed with $\mathcal{S}$-$\mathcal{M}$OISE$^+$, an extension to SACI (http://www.lti.pcs.usp.br/saci) where the agents have an organisational aware architecture. This tool is proposed in [5] (this paper is available in the publications directory of the $\mathcal{M}$OISE$^+$ distribution). While the paper focus on the organisation framework, this chapter focus on the agents development. Thus the next section describes a simple architecture for organised agents and Section B.2 explains how this application agents could be developed.[1]

B.1 A simple organisational agent architecture

The proposed architecture is very simple and is just a starting point towards organisation oriented programming. The base idea is an agent that always do what its organisation needs, it does not have personal goals, and thus there is no conflict between goals.

The following algorithm summarises the agent functioning cycle:

```
1 while true do
2     $g \leftarrow$ choseGoal();
3     $p \leftarrow$ makePlan($g$);
4     execute($p$);
```

The agent firstly chooses an organisational goal, plans a sequence of actions to achieve it, and then executes the plan. MakePlan and execute are domain dependent, whereas choseGoal function is general and could be

[1]As remarked in the introduction, this tool is not supported anymore. Refer to `doc/ora4mas` for the current programming proposal.

```
function choseGoal() : Goal;
if there is an organisational goal permitted to be achieved then
    returns it;
if I have no role then
    adopts a role;
    returns choseGoal();
if try to commit to an obligated mission then
    returns choseGoal();
if try to commit to a permitted mission then
    returns choseGoal();
if try to uncommit to finished schemes then
    for all mission $m$ I am committed to do
        if the scheme of $m$ is already finished then
            uncommit($m$);
returns no goal;
```

According to this algorithm, in case the agent has no organisational goal (first if), it firstly tries to adopt a role, then tries to commit to an obligated mission, and lastly it tries to commit to a permitted mission. After its commitments, it eventually will get an organisational goal. The last **if** remove the agent commitments when a scheme are already finished. Note that this algorithm assumes that the agent will enact only one role in the organisation.

B.2 The write paper agents

The $\mathcal{S}$-$\mathcal{M}$OISE$^+$ API has three main classes to access the organisational layer (as defined in [5]):

- OrgBox: this class has methods to generate organisational events like role adoption, mission commitment, group creation, etc. (See the API documentation for a detailed documentation).
- OEAgent: this class represents the agent inside the organisation, it stores the agents roles, missions, etc.
- BaseOrgAgent: this class implements the architecture described in Section B.1. It has two attributes currentGoal (from class GoalInstance) and currentPlan (a Java List). currentGoal is initialised in the choseGoal method and currentPlan is initialised in the user's plan method. The currentPlan is a list of strings where each element is an action description.

Using these classes, it is quite easy to code agents in the $\mathcal{S}$-$\mathcal{M}$OISE$^+$ framework. The programmer needs to override the adoptRole, plan, and executeAct methods of the BaseOrgAgent class. For example, the Jomi agent program is[2]:

```
public class JomiAg extends BaseOrgAgent {

    public static void main(String[] args) {
```

[2]The code for exceptions handling is omitted to increase readability.

```
        JomiAg a = new JomiAg();

        if (a.enterSoc("jomi", "writePaperSoc")) {
            a.initAg(null);
            a.run();
        }
    }

    protected boolean adoptRole() {
        String roleId = "writer";
        String grTeamId = getOrgBox().getRootGroupInstance( "wpgroup" );
        if (grTeamId != null) {
            getOrgBox().adoptRole(roleId, grTeamId);
            print("adopted the role "+roleId);
            return true;
        } else {
            print("plan aborted: can not identify/create a group team");
        }
        return false;
    }

    protected void plan() {
        currentPlan = null;
        if (currentGoal != null) {
            // create a plan that only prints the current goal!
            currentPlan = new ArrayList();
            currentPlan.add("print("+currentGoal+")");
        }
    }

    protected void executeAct(String action) {
        if (action.startsWith("print"))
            print(action);
   }
}
```

The main method just creates an JomiAg instance, enter this agent in the society, and runs it. The default run method is the one proposed in the architecture (a while true loop).

When the choseGoal method do not find an organisational goal for the agent, it first calls adoptRole. This Jomi's method gets the identification of the wpgroup instance[3] and adopts the 'writer' role in this group.

Jomi's plan method is very simple, itcreates a plan with only one action that is to print the goal! Having a plan, the architecture calls executeAct for each action of the current plan. Finally the executeAct method executes the print actions.

The Jaime program is:

```
public class JaimeAg extends BaseOrgAgent {

    public static void main(String[] args) {
        JaimeAg a = new JaimeAg();

        if (a.enterSoc("jaime", "writePaperSoc")) {
            a.initAg(null);
            a.run();
        }
```

[3]In case where there is no instance, the getRootGroupInstance method creates one.

```
}

protected boolean adoptRole() {
    String roleId = "editor";
    String grTeamId = getOrgBox().getRootGroupInstance( "wpgroup" );
    if (grTeamId != null) {
        getOrgBox().adoptRole(roleId, grTeamId);
        print("adopted the role "+roleId);
        return true;
    } else {
        print("plan aborted: can not identify/create a group team");
    }
    return false;
}

protected void uncommit(MissionPlayer mp) {
    super.uncommit(mp);

    // it is the case my scheme is finished
    SCH schWP = findSch();
    if (schWP != null) {
        if (schWP.getRoot().isSatisfied()) {
            getOrgBox().finishSCH(schWP.getId());
            destroy(); // kill myself
        }
    }
}

protected void plan() {
    currentPlan = null;
    if (currentGoal == null) { // there is no goal
        // create a Write a Paper scheme
        if (findSch() == null)
           createSch();
        choseGoal();
    }
    if (currentGoal != null) {
       // create a plan that only prints the current goal!
       currentPlan = new ArrayList();
       currentPlan.add("print("+currentGoal+")");
    }
}

protected void executeAct(String action) {
    if (action.startsWith("print"))
       print(" * doing * "+action);
}

SCH findSch() {
    // find my group scheme
    Group myGroup = getRolePlayer().getGroup();

    Iterator iSch = getOrgBox().getOE().
           findInstancesOfSchSpec( "writePaperSch" ).iterator();
    while (iSch.hasNext()) {
        SCH sch = (SCH)iSch.next();
        if (sch.getResponsibleGroups().contains( myGroup ))
            return sch;
    }
    return null;
    }

    boolean firstSchAlreadyCreated = false;
    SCH createSch() {
       if (firstSchAlreadyCreated)
          return null;

       String schId = getOrgBox().startSCH( "writePaperSch" );
       // set the responsible group
       getOrgBox().addResponsibleGroup(schId, getRolePlayer().
                                         getGroup().getId() );
       firstSchAlreadyCreated = true;
       return getOrgBox().getOE().findSCH(schId);
    }

    public RolePlayer getRolePlayer() {
       return (RolePlayer)getOrgBox().getMyOEAgent().
              getRoles().iterator().next();
    }
}
```

This agent planner creates a write paper scheme in case it could not find an organisational goal (chose goal equals null). It also overrides the uncommit method. If Jaime uncommits its *mMan* mission, it means that the root goal of the write paper scheme is achieved, and thus Jaime can finish the scheme (remove it from the organisational entity). Since a scheme could be finished only when it has no players, Jaime waits that other agents uncommit and then finishes the scheme.

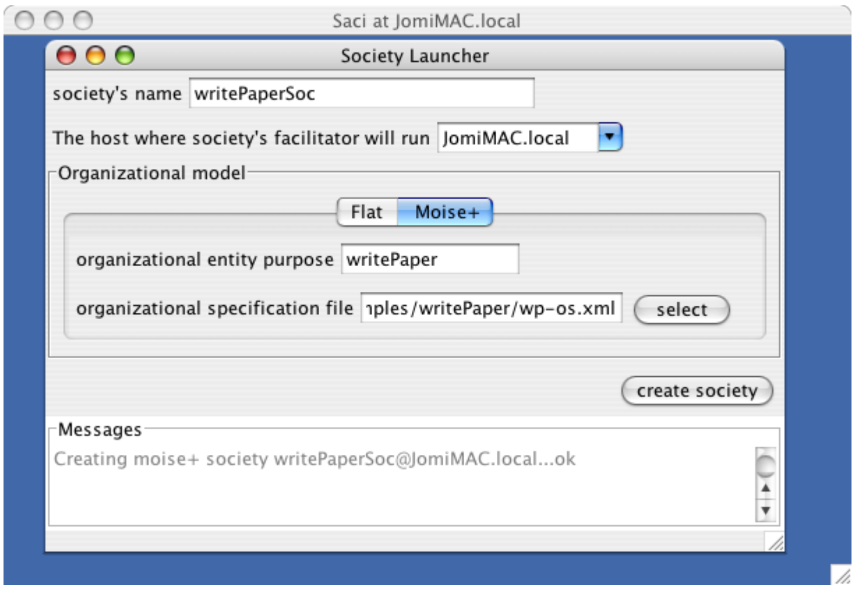
The following steps describe how to run this MAS with Ant scripts[4]:

1. Go to `saci/examples/moise/writePaper` directory

2. Start Saci

   ```
   ant saci
   ```

3. Start the OrgManager. In the Saci window, menu Launcher/Start Societies, fill the fields as shown in the figure below.



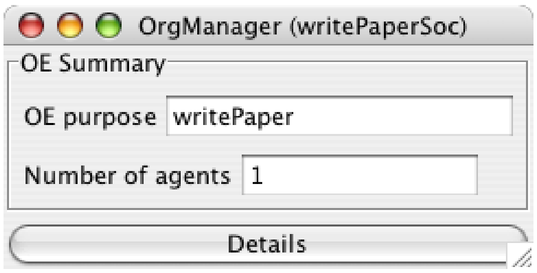
society's name: writePaperSoc
The host where society's facilitator will run: JomiMAC.local
organizational entity purpose: writePaper
organizational specification file: ıples/writePaper/wp-os.xml
Creating moise+ society writePaperSoc@JomiMAC.local...ok

[4]You need to install Ant to run this example as described here, it is available at ant.apache.org.

Alternatively, start the OrgManager by Ant

```
ant orgManager
```

The OrgManager will create a new window where the current organisational specification and entity can be consulted.



OE purpose writePaper

Number of agents 1

4. Run the Jaime agent

```
ant jaime
```

The output should be something like:

```
Agent jaime is inside society writePaperSoc
[jaime] adopted the role editor
[jaime] committed to permitted mission writePaperSch.mManager
        in sch_writePaperSch0 [jaime] my goal is wtitle
[jaime] Executing plan [print(wtitle)]
[jaime]  * doing * print(wtitle)
[jaime] Setting wtitle as satisfied.
[jaime] my goal is wabs
[jaime] Executing plan [print(wabs)]
[jaime]  * doing * print(wabs)
[jaime] Setting wabs as satisfied.
[jaime] my goal is wsectitles
[jaime] Executing plan [print(wsectitles)]
[jaime]  * doing * print(wsectitles)
[jaime] Setting wsectitles as satisfied.
[jaime] my goal is fdv
[jaime] Executing plan [print(fdv)]
[jaime]  * doing * print(fdv)
[jaime] Setting fdv as satisfied.
```

Note that Jaime adopted the editor role, committed to $mMan$ mission and satisfied the goal *fdv* (first darft version).

5. Run the Jomi agent

```
ant jomi
```

The output should be something like:

```
Agent jomi is inside society writePaperSoc
[jomi] adopted the role writer
[jomi] committed to writePaperSch.mColaborator
[jomi] my goal is wsec
[jomi] Executing plan [print(wsec)]
[jomi] print(wsec)
[jomi] Setting wsec as satisfied.
```

Now Jaime can continue and satisfy the goal *wconc*, Jomi commits to *mBib* mission and satisfies *wref* and *sv* goals. Then Jaime satisfy the scheme root goal. Since the scheme is satisfied, both uncommit their missions and finish their work.

Appendix C

Organisational Entity API

The same events that we can produced on an OE by the simulator can be produced by calling Java methods. Indeed, the simulator only encapsulates these calls in a graphical interface. This chapter will therefore briefly introduce the utilization of the Java API for maintaining the state of an OE.

C.1 A kind of 'hello world'

The simplest program we can write using the $\mathcal{M}$OISE$^+$ Java API is[1]:

```
import moise.oe.*;

class HelloMoise {
  public static void main(String[] args) {
    try {
      OE currentOE = OE.createOE("winGame", "jojOS.xml");

      new moise.tools.SimOE(currentOE);

    } catch (Exception e) {
        System.exit(1);
} } }
```

- the first line import the $\mathcal{M}$OISE$^+$ OE API;
- the line 5 creates an OE with the goal 'winGame' and OS as state in the file 'jojOS.xml';
- the line 8 calls the simulator interface.

C.2 Program examples

In the directory . . . /examples/tutorial there are commented examples of Java programs that:

- Creates the groups, agents, and roles: TutorialSS.java

[1]Before compiling and running this program, you must add the `moise.jar` file in the CLASSPATH.

- Creates the scheme and goals: TutorialFS.java
- Creates commitments: TutorialFS.java

Appendix D

Properties of the organisational specification

The organisation constraints used in $\mathcal{S}$-$\mathcal{M}$OISE$^+$ and $\mathcal{J}$-$\mathcal{M}$OISE$^+$ may be turned off for some applications. For example, normally a group can be removed only when empty. To turn this constraint off, the SS have to include:

```
<structural-specification>
  <properties>
     <property id="check-players-in-remove-group" value="false" />
  </properties>
  ...
```

The list of reserved properties in the SS are:

- `check-players-in-remove-group` (default value is *true*): whether a group can be removed when some agent is playing a role inside the group.
- `check-subgroup-in-remove-group` (default value is *true*): whether a group can be removed when some subgroup is attached.
- `check-missions-in-remove-role` (default value is *true*): whether an agent may remove a role if the role is obliged to that mission.

The list of reserved properties in the FS are:

- `check-players-in-remove-scheme` (default value is *true*): whether a scheme can be removed when some agent is comitted to the scheme.
- `check-players-in-remove-responsible-group` (default value is *true*): whether a responsible group can be removed from a scheme when some agent of the group is committed to a mission in the scheme.
- `only-owner-can-remove-scheme` (default value is *true*): whether only the owner of a scheme can remove it.
- `check-goals-in-remove-mission` (default value is *true*): whether an agent can remove a mission when some of the mission's goals were not achieved.

Appendix E

XML files

E.1 Organisational Specification for the JojTeam

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet href="os.xsl" type="text/xsl" ?>

<organisational-specification

    id="joj"
    os-version="0.7"

    xmlns='http://moise.sourceforge.net/os'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://moise.sourceforge.net/os
                        http://moise.sourceforge.net/xml/os.xsd'>

    <structural-specification>
        <role-definitions>
            <role id="player" />
            <role id="coach" />
            <role id="middle">     <extends role="player"/> </role>
            <role id="leader">     <extends role="player"/> </role>
            <role id="back">       <extends role="player"/> </role>
            <role id="goalkeeper"> <extends role="back"/>   </role>
            <role id="attacker">   <extends role="player"/> </role>
        </role-definitions>

        <group-specification id="team">
            <roles>
                <role id="coach" min="1" max="2"/>
            </roles>
            <links>
                <link from="leader" to="player" type="authority"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
                <link from="coach"  to="player" type="authority"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
                <link from="player" to="player" type="communication"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
                <link from="player" to="coach" type="acquaintance"
                      scope="inter-group" extends-subgroups="true" bi-dir="false"/>
            </links>
            <subgroups>

                <group-specification id="attack" min="1" max="1">
                    <roles>
                        <role id="middle"   min="5" max="5" />
                        <role id="leader"   min="0" max="1"/>
                        <role id="attacker" min="2" max="2" />
                    </roles>
                    <formation-constraints>
                        <compatibility from="middle" to="leader" type="compatibility"
                                       scope="intra-group" extends-subgroups="false"
                                       bi-dir="true"/>
                    </formation-constraints>
                </group-specification>

                <group-specification id="defense" min="1" max="1">
                    <roles>
                        <role id="leader"     min="0" max="1" />
                        <role id="goalkeeper" min="1" max="1" />
                        <role id="back"       min="3" max="3" />
                    </roles>
                    <links>
                        <link from="goalkeeper" to="back" type="authority"
                              scope="intra-group" extends-subgroups="false"
                              bi-dir="false"/>
                    </links>
                    <formation-constraints>
                        <compatibility from="back" to="leader" type="compatibility"
                                       scope="intra-group" extends-subgroups="false"
                                       bi-dir="true"/>
                    </formation-constraints>
                </group-specification>
            </subgroups>

            <formation-constraints>
              <!-- subgroups scope cardinality -->
                <cardinality min="1" max="1" object="role" id="leader"/>
            </formation-constraints>
        </group-specification>
    </structural-specification>


    <functional-specification>
        <scheme id="sideAttack" >
            <goal id="scoreGoal" min="1">
              <plan operator="sequence">
                <goal id="g1" min="1" ds="get the ball" />
                <goal id="g2" ds="to be well placed">
                  <plan operator="parallel">
                    <goal id="g7" min="1" ds="go toward the opponent's field" />
                    <goal id="g8" min="1" ds="be placed in the middle field" />
                    <goal id="g9" min="1" ds="be placed in the opponent's goal area" />
                  </plan>
                </goal>
                <goal id="g3" min="1" ds="kick the ball to the m2Ag" >
                  <argument id="M2Ag" />
                </goal>
                <goal id="g4" min="1" ds="go to the opponent's back line" />
                <goal id="g5" min="1" ds="kick the ball to the goal area" />
                <goal id="g6" min="1" ds="shot at the opponent's goal" />
              </plan>
            </goal>

            <mission id="m1" min="1" max="1">
                <goal id="g1" />
                <goal id="g3" />
                <goal id="g7" />
            </mission>
            <mission id="m2" min="1" max="1">
                <goal id="g8" />
                <goal id="g4" />
                <goal id="g5" />
            </mission>
            <mission id="m3" min="1" max="1">
                <goal id="g9" />
                <goal id="g6" />
            </mission>
        </scheme>
    </functional-specification>

    <normative-specification>
        <norm id="n1" type="permission" role="back"     mission="m1" />
        <norm id="n2" type="obligation" role="middle"   mission="m2" />
        <norm id="n3" type="obligation" role="attacker" mission="m3" />
    </normative-specification>
</organisational-specification>
```

E.2 Organisational Specification for the Write Paper application

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xsl" type="text/xsl" ?>

<organisational-specification

    id="wp"
    os-version="0.8"

    xmlns='http://moise.sourceforge.net/os'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://moise.sourceforge.net/os
                        http://moise.sourceforge.net/xml/os.xsd' >

    <structural-specification>

        <role-definitions>
            <role id="author" />
            <role id="writer"> <extends role="author"/> </role>
            <role id="editor"> <extends role="author"/> </role>
        </role-definitions>

        <group-specification id="wpgroup" monitoring-scheme="monitoringSch">
            <roles>
                <role id="writer" min="1" max="5" />
                <role id="editor" min="1" max="1" />
            </roles>
            <links>
                <link from="writer" to="editor" type="acquaintance"
                      scope="intra-group" extends-subgroups="true" bi-dir="false"/>
                <link from="editor" to="writer" type="authority"
                      scope="intra-group" extends-subgroups="true" bi-dir="false"/>
                <link from="author" to="author" type="communication"
                      scope="intra-group" extends-subgroups="true" bi-dir="false"/>
            </links>

            <formation-constraints>
                <compatibility from="editor" to="writer" type="compatibility"
                               scope="intra-group" extends-subgroups="false"
                               bi-dir="true"/>
            </formation-constraints>
        </group-specification>
    </structural-specification>



    <functional-specification>
        <scheme id="writePaperSch" monitoring-scheme="monitoringSch">

            <goal id="wp" ttf="5 seconds">
                <plan operator="sequence" >
                    <goal id="fdv" ds="First Draft Version">
                        <plan operator="sequence">
                            <goal id="wtitle"     ttf="1 day" ds="Write a title"/>
                            <goal id="wabs"       ttf="1 day" ds="Write an abstract"/>
                            <goal id="wsectitles" ttf="1 day" ds="Write the sections' title" />
                        </plan>
                    </goal>
                    <goal id="sv" ds="Submission Version">
                        <plan operator="sequence">
                            <goal id="wsecs"  ttf="7 days" ds="Write sections"/>
                            <goal id="finish" ds="Finish paper">
                                <plan operator="parallel">
                                    <goal id="wconc"  ttf="1 day"  ds="Write a conclusion"/>
                                    <goal id="wrefs"  ttf="1 hour" ds="Complete references and link them to text"/>
                                </plan>
                            </goal>
                        </plan>
                    </goal>
                </plan>
            </goal>
            <mission id="mColaborator" min="1" max="5">
                <goal id="wsecs"/>
            </mission>

            <mission id="mManager" min="1" max="1">
                <goal id="wabs"/>
                <goal id="wp"/>
                <goal id="wtitle"/>
                <goal id="wconc"/>
                <goal id="wsectitles"/>
            </mission>

            <mission id="mBib" min="1" max="1">
                <goal id="wrefs"/>
                <preferred mission="mColaborator"/>
                <preferred mission="mManager"/>
            </mission>
        </scheme>

        <scheme id="monitoringSch">
            <goal id="monitor">
                <plan operator="choice">
                    <goal id="sanction" ds="Sanction the agent that is not doing its job!"/>
                    <goal id="reward"   ds="Reward some agent for doing a good job!"/>
                </plan>
            </goal>
            <mission id="ms" min="1" max="1" >
                <goal id="sanction"/>
            </mission>
            <mission id="mr" min="1" max="1" >
                <goal id="reward"/>
            </mission>
        </scheme>
    </functional-specification>

    <normative-specification>
        <norm id = "n1"
                          type="permission"
                          role="editor" mission="mManager" />
        <norm id = "n2"
                          type="obligation"
                          role="writer" mission="mBib"
                          time-constraint="1 day"  />
        <norm id = "n3"
                          type="obligation"
                          role="writer" mission="mColaborator"
                          time-constraint="1 day"  />
        <norm id = "n4"
                          type="obligation"
                          condition="unfulfilled(obligation(_,n2,_,_))"
                          role="editor" mission="ms"
                          time-constraint="3 hours"/>
        <norm id = "n5"
                          type="obligation"
                          condition="fulfilled(obligation(_,n3,_,_))"
                          role="editor" mission="mr"
                          time-constraint="3 hours"/>
        <norm id = "n6"
                          type="obligation" condition="#goal_non_compliance"
                          role="editor" mission="ms"
                          time-constraint="3 hours"/>
        <norm id = "n7"
                          type="obligation" condition="#role_compatibility"
                          role="editor" mission="ms"
                          time-constraint="30 minutes"/>
        <norm id = "n8"
                          type="obligation" condition="#mission_cardinality"
                          role="editor" mission="ms"
                          time-constraint="1 hour"/>
        <!-- norm id = "n9"
                          type="obligation" condition="#role_cardinality"
                          role="editor" mission="ms"
                          time-constraint="30 minutes"/-->
    </normative-specification>
</organisational-specification>
```

Bibliography

[1] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[2] Jomi F. Hübner, Olivier Boissier, and Rafael H. Bordini. Normative programming for organisation management infrastructures. In Axel Polleres and Julian Padget, editors, *Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN09@MALLOW) Torino, Italy, 7th–11th September*, volume 494. CEUR, 2009.

[3] Jomi F. Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents: "giving the organisational power back to the agents". *Journal of Autonomous Agents and Multi-Agent Systems*, 2009.

[4] 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.

[5] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. S-MOISE+: A middleware for developing organised multi-agent systems. In Olivier Boissier, Virginia Dignum, Eric Matson, and Jaime Simão Sichman, editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer, 2006.

[6] 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.

[7] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: "giving the power back to the agents". In Jaime Sichman, P. Noriega, J. Padget, and Sascha Ossowski, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, volume 4870 of *LNCS*, pages 171–186. Springer, 2008. Revised Selected Papers.