

# Jason and MOISE<sup>+</sup>

Organisational Programming in the Agent Contest 2008

Jomi F. Hübner<sup>†</sup>, Rafael H. Bordini<sup>\*</sup>  
Gauthier Picard<sup>†</sup>

<sup>†</sup> ENS Mines Saint Etienne, France  
{hubner,picard}@emse.fr

<sup>\*</sup> University of Durham, UK  
r.bordini@durham.ac.uk

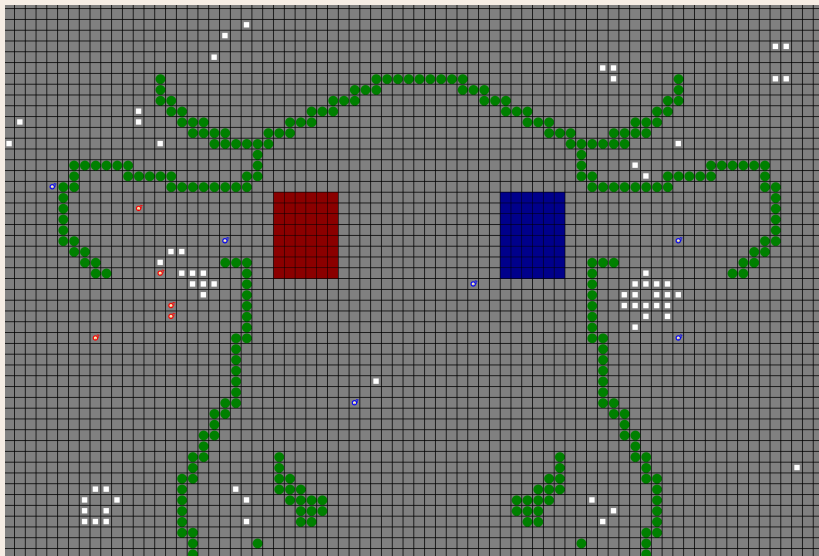
Dagstuhl Seminar on Programming Multi-Agent Systems

# Outline

- 1 Introduction
- 2 Design
- 3 Implementation
- 4 Conclusions

# Agent Contest 2008

## the Cows and Herders scenario



# Objectives of our participation

- In 2006: program our agents using **plans**
  - ∴ reactive agents
- In 2007: program our agents using **goals**
  - ∴ goal directed agents
- In 2008: program our agents using **organisation**
  - ∴ join agent and system levels
  - ↪ use **Jason** for the agents
  - ↪ use *MOISE*<sup>+</sup> for the organisation
- Test and improve **Jason** and *MOISE*<sup>+</sup> software
- Evaluate the use of organisational constructors in the development of the team

# Objectives of our participation

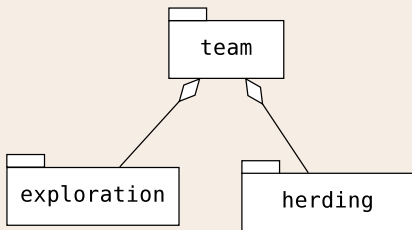
- In 2006: program our agents using **plans**
  - ∴ reactive agents
- In 2007: program our agents using **goals**
  - ∴ goal directed agents
- In 2008: program our agents using **organisation**
  - ∴ join agent and system levels
  - ↪ use **Jason** for the agents
  - ↪ use  $\mathcal{MOISE}^+$  for the organisation
- Test and improve **Jason** and  $\mathcal{MOISE}^+$  software
- Evaluate the use of organisational constructors in the development of the team

- 1 Introduction
  - context
  - objectives
- 2 Design
  - specification
  - dynamics
  - goals
- 3 Implementation
  - *MOISE*<sup>+</sup>
  - Jason
  - Java
  - tools
- 4 Conclusions
  - results
  - discussion

# Team specification — Groups

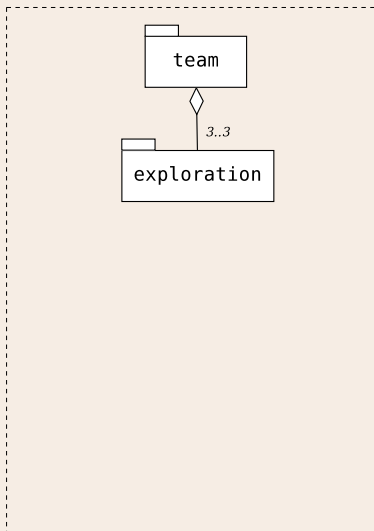
Our agents are **organised** in two types of groups:

- Exploration group: find cows
- Herding group: push cows into the corral



# Exploration group

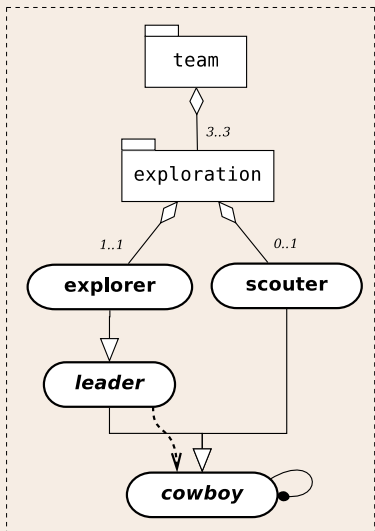
- Three instances of exploration group
- Each group is allocated to an area of the scenario
- One **explorer**: decides where to go
- One **scouter**: follows explorer





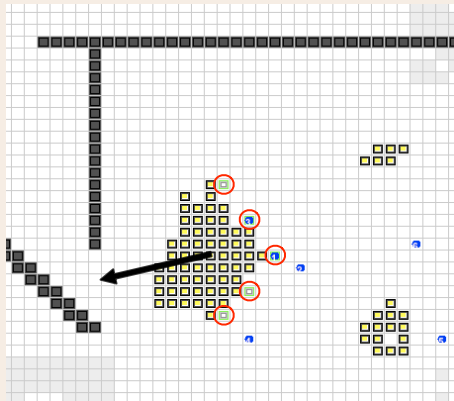
# Exploration group

- Three instances of exploration group
- Each group is allocated to an area of the scenario
- One **explorer**: decides where to go
- One **scouter**: follows explorer



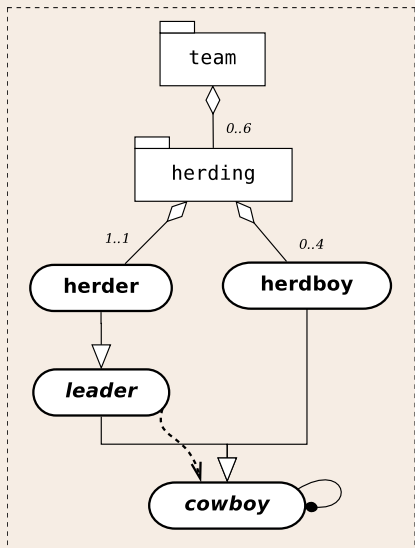
# Herding group

- Created when a member of the team sees some cow
- As many instances as the number of **clusters of cows**
- One herder:  
defines the team formation for the cluster
- scouters:  
be in formation
- **spatial coordination**



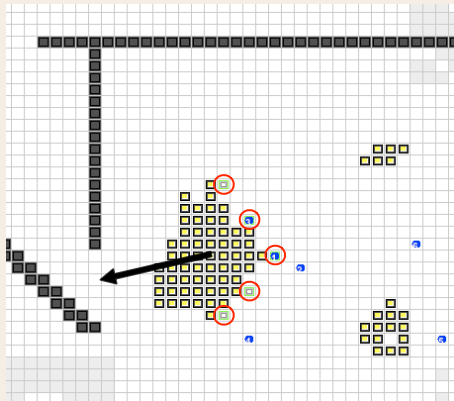
# Herding group

- Created when a member of the team sees some cow
- As many instances as the number of clusters of cows
- One **herder**:  
defines the team formation for the cluster
- scouters**:  
be in formation
- spatial coordination**



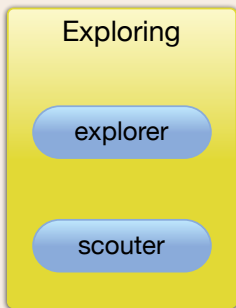
# Herding group

- Created when a member of the team sees some cow
- As many instances as the number of clusters of cows
- One herder:  
defines the **team formation** for the cluster
- scouters:  
be in formation
- **spatial coordination**



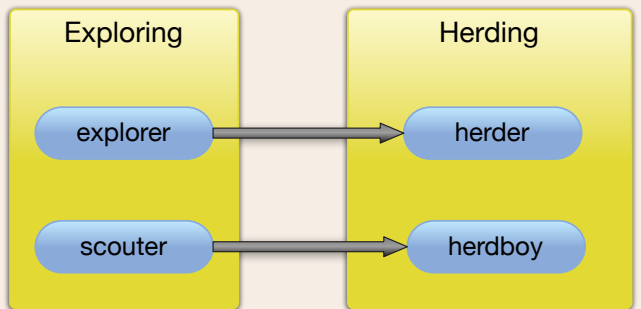
# Structural Dynamics

Start exploring



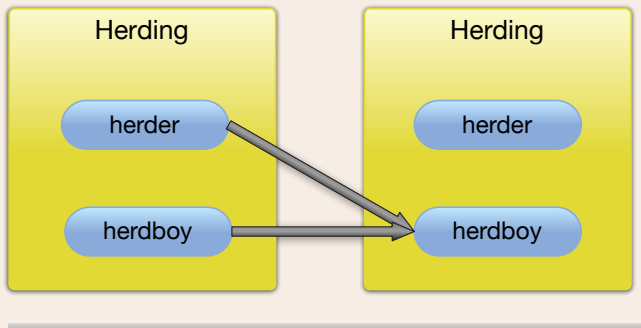
# Structural Dynamics

## Creation of herding group



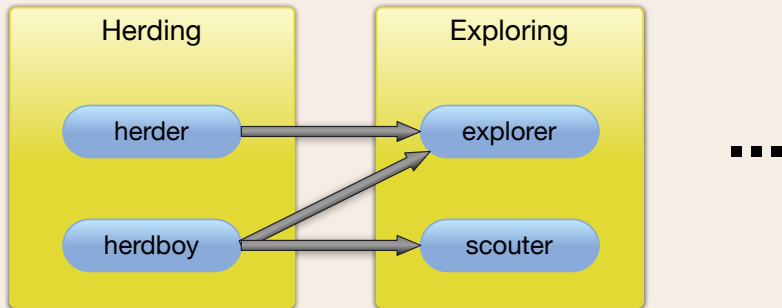
# Structural Dynamics

Merge two herding groups



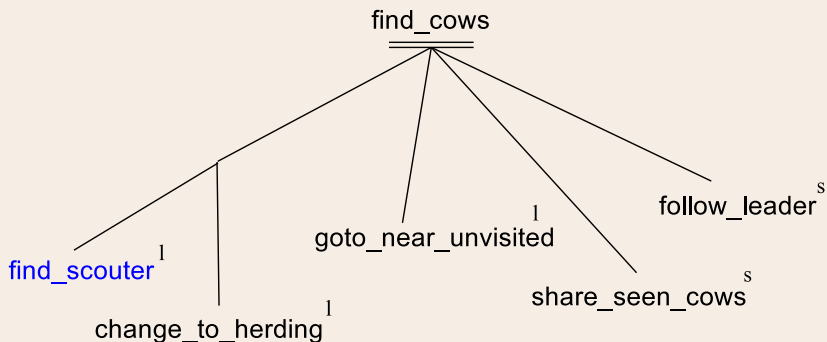
# Structural Dynamics

Dissolve herding group



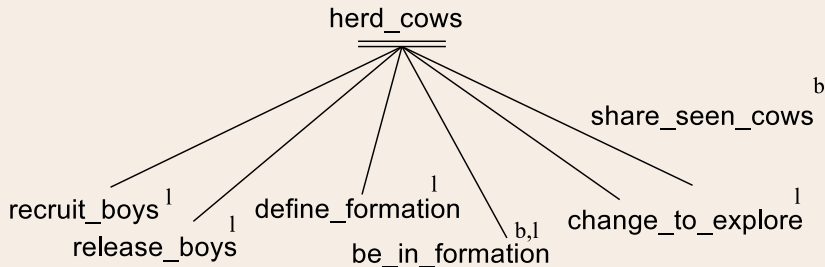


# Functional specification — Scheme to explore



Explorer (the leader) is obligated to mission /  
 Scouter is obligated to s

# Functional specification — Scheme to herd



Herder (the leader) is obligated to mission  $l$   
 Herdboy is obligated to  $b$

# Goals

Role	Goal	Goal Description
<i>explorer</i>	find_scouter change_to_herding goto_near_unvisited	find agent nearby to play scouter change to a herding group go to the nearest unvisited location
<i>scouter</i>	share_seen_cows follow_leader	share information about cows follow the leader of the group
<i>herder</i>	recruit release_boys define_formation be_in_formation merge change_to_exploring	recruit more herdboys release some herdboys compute the formation of the group go to the place allocated to the agent merge two herding groups change to an exploring group
<i>herdboy</i>	share_seen_cows be_in_formation	

- 1 Introduction
  - context
  - objectives
- 2 Design
  - specification
  - dynamics
  - goals
- 3 Implementation**
  - MOISE<sup>+</sup>
  - Jason
  - Java
  - tools
- 4 Conclusions
  - results
  - discussion

# Organisation Oriented Programming I

...

```
<group-specification id="team">  
  <sub-groups>  
    <group-specification id="exploration_grp"  
                        min="0" max="3" >  
      <roles>  
        <role id="explorer" min="1" max="1"/>  
        <role id="scouter"  min="0" max="1"/>  
      </roles>  
    </group-specification>  
  </sub-groups>  
</group-specification>
```

...

# Organisation Oriented Programming II

Tools to run the organisation:

- $\mathcal{S}\text{-}\mathcal{M}OISE^+$ : organisational infrastructure
  - manage the state of the organisation
- $\mathcal{J}\text{-}\mathcal{M}OISE^+$ : integration with **Jason**
  - library of organisational actions
  - organisational architecture

# Organisation Oriented Programming III

- Agents are informed about their obligations  
∴ new goal event

## Plan to handle a new goal — maintenance goal pattern

- the goal is annotated with the group and role that generated the obligation

```

+!define_formation[group(G),role(R)]
  <- ... <the code> ...

      // wait for the next cycle
      .wait("+pos(X,Y,Cycle)");

      // achieve that goal again
      !define_formation[group(G),role(R)].
  
```

# Organisation Oriented Programming IV

- Agents are also informed by changes in the organisation
  - ∴ change the belief base
  - ∴ produce events

## Examples

```
+play(Me,herder,G)
  : .my_name(Me)
  <- +group_leader(G,Me);
     .broadcast(tell, group_leader(G,Me)).

-group(Type,GroupId)
  <- .drop_intention(_[group(GroupId)]).
```



# Agent Oriented Programming I

The achievement of organisational goals is implemented in **Jason**

Goal: merge herding group

**plan** merge

let  $g_i$  be my herding group

**forall** herding group  $g_j$  such that  $g_i > g_j$  **do**

let  $S_i$  be the set of cows of  $g_i$ 's cluster

let  $S_j$  be the set of cows of  $g_j$ 's cluster

**if**  $S_i \cap S_j \neq \emptyset$  **then**

remove group  $g_j$  from the organisation

ask all agents of  $g_j$  to adopt the role *herdboy* in  $g_i$

// new role  $\rightarrow$  new goals

# Agent Oriented Programming II

## Code in Jason

```
+!merge
: .my_name(Me) &
  play(Me, herder, Gi) & // I play role herder
  current_cluster(MyC) // MyC is the list with my cows

<- // for all other groups
  for (group_leader(Gj, L) & Me < L) {
    .send(L,askOne,current_cluster(_),current_cluster(TC));
    .intersection(MyC,TC,I);
    if (I \== []) {
      .send(L, achieve, change_role(herdboy,Gi))
    }
  }.
}
```

When the leader of other group change the role, he will ask his herdboys to also change the group

# Object Oriented Programming

The following components were implemented in **Java**

- Integration with the contest simulator  
Agent perception and action
- Find paths: A\*
- Compute the formation (a lot of vector calculations)
- ...

# 130 lines of code in MOISE<sup>+</sup>

# 696 lines of code in **Jason**

# 4218 lines of code in Java

# Object Oriented Programming

The following components were implemented in **Java**

- Integration with the contest simulator  
Agent perception and action
- Find paths: A\*
- Compute the formation (a lot of vector calculations)
- ...

# 130 lines of code in  $\mathcal{M}OISE^+$

# 696 lines of code in **Jason**

# 4218 lines of code in Java

# Useful tools — Mind inspector

```

play(gaucha1,herder,gr_herding_grp_13){source(orgManager)}.
play(gaucha4,herdboy,gr_herding_grp_13){source(orgManager)}.
play(gaucha5,herdboy,gr_herding_grp_13){source(orgManager)}.
pos(45,44,128){source(percept)}.
scheme(herd_sch,sch_herd_sch_18){owner(gaucha3),source(orgManager)}.
scheme(herd_sch,sch_herd_sch_12){owner(gaucha1),source(orgManager)}.
scheme_group(sch_herd_sch_12,gr_herding_grp_13){source(orgManager)}.
steps(700){source(self)}.
target(6,44){source(gaucha1)}.

```

## - Rules

```

random_pos(X,Y):-
    (pos(AgX,AgY,_418) & (jia.random(RX,40) & ((RX > 5) & ((X = ((RX-20)+AgX)) & ((X >

```

## - Intentions

Sel Id	Pen	Intended Means Stack (hide details)
16927	<b>suspended-self</b>	+lbe_in_formation[scheme(sch_herd_sch_12),mission(hel <b>+lbe_in_formation</b> [scheme(Sch),mission(Mission)]

# Useful tools — MOISE<sup>+</sup> GUI

## jason-cowboys (Organisational Entity)

---

### Agents

gaucho3 ; gaucho4 ; gaucho5 ; gaucho6 ; gaucho1 ; gaucho2 ;

---

### Groups

- [gr\\_team\\_01](#)
  - [gr\\_exploration\\_grp\\_02](#) **players (2):** [gaucho3](#) ([explorer](#)) ; [gaucho6](#) ([scouter](#)) ;
  - [gr\\_herding\\_grp\\_05](#) **players (4):** [gaucho1](#) ([herder](#)) ; [gaucho2](#) ([herdboy](#)) ; [gaucho4](#) ([herdboy](#)) ; [gaucho5](#) ([herdboy](#)) ;

# Typical screen

The screenshot displays the MOISE+ simulation environment. The main window shows a grid-based simulation titled "Herding (view of cowboy 1) -- against SHABan". The simulation area contains a grid with various colored squares representing agents and groups. A vertical black bar represents a corral. Below the grid, a log window shows the simulation progress:

Cycle: 23/1800 Cows in corral (blue x red): 0 x 0

Step	Time	Agent	Position	Direction	Distance	Speed	Angle	Energy	Health	Weight						
Step 14	50	9/2	e	16,27/0	n	33	9/0	w	49,15/5	e	35,9/0	w				
Step 15	51	9/0	e	16,27/1	n	33	9/1	w	50,15/0	--	34,9/0	mw	13,7/0	w	8671	ms
Step 16	52	9/0	nw	16,26/0	n	33	9/2	w	50,15/1	--	34,9/1	mw	13,7/1	w	8671	ms
Step 17	52	9/1	nw	18,24/0	ne	32	9/0	w	50,15/2	sw	33,9/0	mw	14,6/0	e	10000	ms
Step 18	51	8/0	nw	18,24/1	ne	32	9/1	w	50,15/3	sw	33,9/1	mw	14,6/1	e	217	ms
Step 19	50	7/0	sw	18,24/2	ne	31,9/0	nw	50,15/4	sw	33,9/2	nw	14,6/2	e	9582	ms	
Step 20	49	8/0	nw	18,22/0	e	31,9/1	nw	49,16/0	a	32,8/0	nw	15,6/0	ne	9652	ms	
Step 21	49	8/1	nw	19,22/0	nw	30,9/0	w	49,17/0	s	30,6/0	nw	15,8/0	w	10005	ms	
Step 22	48	8/0	nw	19,22/1	nw	30,9/1	w	50,18/0	se	30,6/1	nw	15,8/1	w	59	ms	
Step 23	48	8/1	nw	19,22/2	nw	30,9/2	w	50,18/1	se	30,6/2	nw	14,7/0	w	10000	ms	
Step 24	48	8/2	w	18,21/0	nw	30,9/3	w	51,19/0	se	29,5/0	sw	14,7/1	w	708	ms	
Step 25	47	8/0	nw	18,21/1	nw	28,9/0	w	52,20/0	se	29,5/1	sw	13,7/0	sw	8524	ms	

The right-hand side of the screen shows the "component description" for "Jason-cowboys (Organisational Entity)". It lists the following components:

- Agents:** gaucho3; gaucho4; gaucho5; gaucho6; gaucho1; gaucho2;
- Groups:**
  - gr\_team\_10
    - gr\_herding\_grp\_14 players (2): gaucho2 (herdboy); gaucho2 (help\_herder); gaucho1 (herder); gaucho3 (herd); gaucho4 (help\_herder);
    - gr\_herding\_grp\_15 players (2): gaucho3 (herder); gaucho4 (help\_herder);
    - gr\_herding\_grp\_16 players (2): gaucho3 (herder); gaucho4 (help\_herder);
- Schemes:**
  - sch\_herd\_sch\_12 responsible groups: gr\_herding\_grp\_14 gaucho2 (help\_herder); gaucho5 (herd);
  - sch\_herd\_sch\_14 responsible groups: gr\_herding\_grp\_16 gaucho3 (herd); gaucho6 (help\_herder);
  - sch\_herd\_sch\_13 responsible groups: gr\_herding\_grp\_15 gaucho1 (herd); gaucho4 (help\_herder);

The bottom right corner shows a "CPU Monitor" window with a graph of CPU usage over time.

# Typical development cycle

- have a brilliant idea
- code it
- basic test (JUnit, ASUnit, ...)
- watch the result in the contest scenarios
- read and analyse long logs, minds dumps, traces, performance...  
note that we need to analyse the execution of 6 concurrent agents
- find bugs (in the team, in  $\mathcal{S}$ -MOISE<sup>+</sup>, ...), start again
- tuning of parameters (the cluster size?), start again
- give up the idea, start again



# Typical development cycle

- have a brilliant idea
- code it
- basic test (JUnit, ASUnit, ...)
- watch the result in the contest scenarios
- read and analyse long logs, minds dumps, traces, performance...  
note that we need to analyse the execution of 6 concurrent agents
- find bugs (in the team, in  $\mathcal{S}$ -MOISE<sup>+</sup>, ...), start again
- tuning of parameters (the cluster size?), start again
- give up the idea, start again

# Typical development cycle

- have a brilliant idea
- code it
- basic test (JUnit, ASUnit, ...)
- watch the result in the contest scenarios
- read and analyse long logs, minds dumps, traces, performance...  
note that we need to analyse the execution of 6 concurrent agents
- find bugs (in the team, in  $\mathcal{S}$ -MOISE<sup>+</sup>, ...), start again
- tuning of parameters (the cluster size?), start again
- give up the idea, start again

# Typical development cycle

- have a brilliant idea
- code it
- basic test (JUnit, ASUnit, ...)
- watch the result in the contest scenarios
- read and analyse long logs, minds dumps, traces, performance...  
note that we need to analyse the execution of 6 concurrent agents
- find bugs (in the team, in  $\mathcal{S}$ -MOISE<sup>+</sup>, ...), start again
- tuning of parameters (the cluster size?), start again
- give up the idea, start again

# Typical development cycle

- have a brilliant idea
- code it
- basic test (JUnit, ASUnit, ...)
- watch the result in the contest scenarios
- read and analyse long logs, minds dumps, traces, performance...  
note that we need to analyse the execution of 6 concurrent agents
- find bugs (in the team, in  $\mathcal{S}$ -MOISE<sup>+</sup>, ...), start again
- tuning of parameters (the cluster size?), start again
- give up the idea, start again

# Summary I

- Team
  - agents are **autonomous** to
    - adopt roles
    - decide how to achieve goals
  - **coordination** is essentially spacial (follow leader and formation)
  - **communication** is used to share information (speech act based)

# Summary II

- **Jason**

- declarative and goal oriented programming
  - goal patterns (maintenance goal)
  - meta-programming (`.drop_intention(_[group(g1)])`)
  - customisations (integration with the simulator and the organisation)
  - internal actions (code in Java)
- ∴ good programming style

# Summary III

- $\mathcal{M}OISE^+$ 
    - definition of groups and roles
    - allocation of goals to agents based on their roles
    - to change the team, we (developers) 'simply' change the organisation
    - global orchestration
- ∴ team strategy defined at a high level

# Good points

- New scenario of the contest
- Use of 3 programming paradigms
- Improve several issues of **Jason**,  $\mathcal{MOISE}^+$ , and their integration
  - New type of goal in  $\mathcal{MOISE}^+$  (maintenance goal)
  - More suitable for collaborative systems (group deletion)



# Weak points

- Too much time in 'debug, test, and tuning mode'  
we rather prefer analysis and programming
- The organisation dynamics is specified inside the agents  
it is coded and mixed in the agent's plans
  - new language to define it from a global perspective
- The functional dimension of the team is quite simple  
it allows the definition of global plans useful to achieve shared goals
  - more complex team strategies
  - changes in the scenario
- It is quite difficult to map an idea into different levels of analysis  
what is organisation and what is agent planning; what is *MOISE<sup>+</sup>*, **Jason**, or Java

# Weak points

- Too much time in 'debug, test, and tuning mode'  
we rather prefer analysis and programming
- The organisation dynamics is specified inside the agents  
it is coded and mixed in the agent's plans  
→ new language to define it from a global perspective
- The functional dimension of the team is quite simple  
it allows the definition of global plans useful to achieve shared goals  
→ more complex team strategies  
→ changes in the scenario
- It is quite difficult to map an idea into different levels of analysis  
what is organisation and what is agent planning; what is  $\mathcal{MOISE}^+$ , **Jason**, or Java

# Weak points

- Too much time in 'debug, test, and tuning mode'  
we rather prefer analysis and programming
- The organisation dynamics is specified inside the agents  
it is coded and mixed in the agent's plans
  - new language to define it from a global perspective
- The functional dimension of the team is quite simple  
it allows the definition of global plans useful to achieve shared goals
  - more complex team strategies
  - changes in the scenario
- It is quite difficult to map an idea into different levels of analysis  
what is organisation and what is agent planning; what is *MOISE*<sup>+</sup>, *Jason*, or Java

# Weak points

- Too much time in 'debug, test, and tuning mode'  
we rather prefer analysis and programming
- The organisation dynamics is specified inside the agents  
it is coded and mixed in the agent's plans
  - new language to define it from a global perspective
- The functional dimension of the team is quite simple  
it allows the definition of global plans useful to achieve shared goals
  - more complex team strategies
  - changes in the scenario
- It is quite difficult to map an idea into different levels of analysis  
what is organisation and what is agent planning; what is *MOISE*<sup>+</sup>, **Jason**, or Java

# More information

- <http://moise.sf.net>
- <http://jason.sf.net>  
(the code of our agents is available there)
- J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multi-agent systems using the  $\mathcal{MOISE}^+$  model: Programming issues at the system and agent levels. *Int. J. Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.