Using *Jason* and $\mathcal{M}$OISE$^+$ to Develop a Team of Cowboys

Jomi F. Hübner[1], Rafael H. Bordini[2], and Gauthier Picard[1]

[1] École des Mines de Saint-Étienne, France
{hubner,picard}@emse.fr
[2] University of Durham, UK
r.bordini@durham.ac.uk

1 Introduction

This paper gives an overview of a multi-agent system forming a team of "cowboys" to compete in the Multi-Agent Programming Contest 2008 (the 'Cows and Herders' scenario). In the two previous contests, we tested and improved *Jason* [2], an agent platform based on an extension of an agent-oriented programming language called AgentSpeak(L) [5]. The language is inspired by the BDI architecture, thus based on notions such as goals, plans, beliefs, intentions, etc. The participation in previous contests also increased our experience both in using BDI concepts as well as in programming agents with *Jason* specifically. In the 2006 contest, the focus was on the definition of agent's plans [1], leading to rather reactive agents. In the 2007 contest, the focus was on (declarative) goals [3], leading to more pro-active, goal-directed agents.

For the 2008 contest, we were motivated to continue improving the multi-agent programming abstractions, now towards social or organisational agents, using the concepts such as *roles* and *groups*. The system is therefore developed in two dimensions: agents (using declarative goals) and organisation (using groups, roles, and shared goals). Among several organisational models available, we will use the $\mathcal{M}$OISE$^+$ model because it is well integrated with *Jason* [4]. Our objective in participating in this contest was thus twofold: (i) to continue to test and improve *Jason* and its integration with $\mathcal{M}$OISE$^+$; (ii) evaluate the use of organisational constructs in the development of the team.

2 System Analysis and Design

It is clear, from the description of the scenario, the importance of cowboys working as a coordinated team. It would be very difficult for a cowboy alone to herd a group of cows. We therefore adopted a strategy strongly tied to the notion of group of agents where issues such as spatial formation, membership, and coordination would be emphasised.

The organisational structure of the team is specified in Fig. 1 using the $\mathcal{M}$OISE$^+$ notation. Our team has two types of subgroups: one to explore the environment searching for cows (the *exploration group*) and another one that leads the herd towards the corral (the *herding group*). The team always has three instances of the exploration group, each one responsible for some part of the scenario. The agents enter and leave these groups as the result of their decision to start or stop searching cows. The herding groups are dynamically created as the agents decide to herd a cluster of cows. The number of those
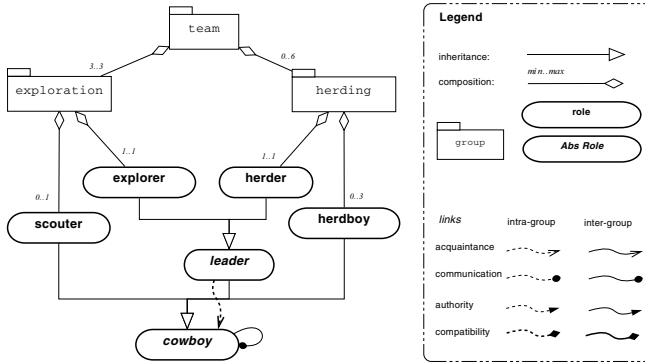
Fig. 1. The Structural Specification of the Organisation

groups and the agents that belong to them depend on the size and location of found clusters of cows. The following roles can be played by agents in the respective groups:

- *explorer*: explores the environment until it detects a cow;
- *scouter*: follows the explorer;
- *herder*: herds the cows detected by explorers until they reach the corral;
- *herdboy*: helps the herder to lead cows to the corral.

The roles *leader* and *cowboy* are abstract and used to specify common properties of their sub-roles. For example, leaders have authority over others cowboys.

The general dynamics of the agents playing the above roles is described with the help of the following scenario. (1-start) At the beginning of the simulation, three exploration groups are created with two agents in each group, on playing the explorer and the other the scouter role. Agents split themselves up so as to cover as wide a range as possible, without necessarily keeping each other in sight. (2-herd) As soon as an agent perceives cows, it informs the members of its exploration group. The explorer of the group creates a new herding group and then changes its role to herder. The scouter also changes its role to herdboy in the new group. After the new group is created, a cluster of cows is assigned to it based on the cows already seen by the agents. The leader then defines the group formation so that the cows are led to the corral. (3-merge) If two herding groups are too near, they are 'merged': one group remains and the other is removed from the organisation. All agents of the removed group change their roles to herdboy in the remaining group. (4-dissolve) Once the corral is reached and the cluster is empty, the herding group is dissolved and the agents create exploration groups returning to the first step (1-start). Table 1 briefly presents the goals that agents are obligated to achieve when playing each of the roles. An agent that adopts the role *scouter*, for instance, is obligated to achieve the goals share_seen_cows and follow_leader.

Although we have some global constraints over the agents' behaviour (based on the roles they are playing), they are *autonomous* to decide how to achieve the goals assigned to them. While *coordination* and *team work* are managed by the $\mathcal{M}$OISE$^+$ tools, the *autonomy* and *pro-activeness* are facilitated by the BDI architecture of our agents implemented in ***Jason***. Regarding *communication* (required, for example, for the share_seen_cows goal), we use speech-act based communication available in ***Jason***.

Table 1. The Organisational (Maintenance) Goals assigned to Roles

Role	Goal	Goal Description
explorer	find_scouter	find a free agent nearby to play scouter and help in the exploration
	change_to_herding	check if it is best to change to a herding group
	goto_near_unvisited	go to the nearest unvisited location within the group's area
scouter	share_seen_cows	share information about cows with other agents in the group
	follow_leader	follow the leader of the group (an explorer)
herder	recruit	recruit more herdboys depending on the size of the cluster
	release_boys	whenever the group has too many herdboys, release some
	define_formation	compute the ideal location of each member of the group
	be_in_formation	go to the place allocated to the agent in the formation
	change_to_exploring	check if it is best to change to an exploring group
herdboy	share_seen_cows	share information about cows with other agents in the group
	be_in_formation	go to the place allocated to the agent in the formation

3   Software Architecture

To implement our agent team, two features of ***Jason*** were specially useful: architecture customisation and internal actions. A customisation of the agent architecture is used to interface between the agent and its environment. The environment for the Agent Contest is implemented in a remote server that simulates the cattle field, sending perception to the agents and receiving requests for action execution. Therefore, when an agent attempts to perceive the environment, the customised architecture sends to the agent the information provided by the server, and when the agent chooses an action to be performed, the architecture sends the action execution request to the server.

Although most of the agent code was written in AgentSpeak, some parts were implemented in Java, either because we used legacy code or Java was more appropriate for the task. In particular, we already had a Java implementation of the A* search algorithm, which we used to find paths and calculate distances in the various scenarios of the competition. Also, the computation of the formation of the herding groups requires a lot of vector operations, so best done in Java. These algorithms ware made accessible to the agents by means of *internal actions*.

The organisational interaction is also made available to the agents by means of a custom architecture and internal actions. This architecture produces events when: (1) something has changed in the state of the organisation (e.g., a new group was created); and (2) when the agent has some new obligation based on the roles it is playing. These events may then lead to the creation of intentions to handle them. For example, when some agent adopts the role herder in a herding group, achievement goal events are produced for all obligatory goals of this role (Table 1). An AgentSpeak plan pattern as follows was used to program suitable reactions to those events:

```
+!define_formation[group(G),role(R)]      // plan to handle a goal addition
  <- ... <the code> ...
     .wait("+pos(X,Y,Cycle)");            // wait for the next cycle
     !define_formation[group(G),role(R)]. // achieve that same goal again
```
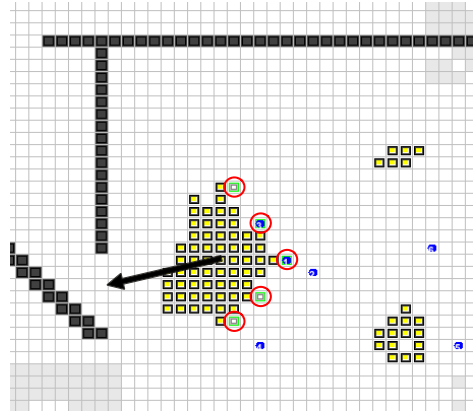
Note that organisational goals here are maintenance goals: for example, at every simulation cycle the target group formation should be (re)defined. These goals are also annotated with the group and the role that triggered the obligation. This allows us

to code interesting plans such as "`-group(Type,GroupId) <- .drop_inten tion(_[group(GroupId)]).`", i.e., whenever a group is removed (e.g., a herding group), all the intentions that originated from that group are dropped.

The agents' code is essentially a set of plans to achieve such organisational goals. In many cases, these plans have to decide whether to change the organisation. For example, the goal recruit may trigger a merging of two herding groups; the actions of this plan are roughly: destroy one group and ask their members to change their roles (Algorithm 1). By changing the roles, new goals are automatically defined for the agents. To sum up, decisions are taken at the organisational level (groups/roles), the goals and intentions are a consequence.

Fig. 2. Team formation in a contest scenario. Cows are yellow and obstacles are black. Green squares inside red circles are target locations for the agents (blue). The arrow indicates the direction of the corral.

The overall performance of the team is, however, also dependent on lower level algorithms. The most important are: (i) A* to find good paths; (ii) the definition of the cluster of cows for a herding group — the cluster should be the largest the agents can herd (see Algorithm 2); and (iii) the definition of the agent formation so that the cows are led to the corral (Fig. 2 illustrated the result[1] of our algorithm).

4   Discussion

The AgentSpeak code for the team is, in our opinion, quite an elegant solution, being declarative, goal-based (or BDI-based), and adequately integrated with an organisational mode. In this paper, we have emphasised the modelling and programming of the team by means of organisational concepts, specially groups and roles. Agents' goals originate from the obligations attached to their roles. This allows us to maintain high abstraction level and good coding style. In some cases, to change the team behaviour we simply changed the organisational specification that was followed by our cowboys. The *Jason* interpreter provided good support for high-level communication, transparent integration with the contest server, use of existing Java code, and integration of organisational programming through $\mathcal{M}$OISE$^+$. As in previous contests, the experience helped us to improve several issues of *Jason*, $\mathcal{M}$OISE$^+$, and their integration.

We had three main difficulties in developing of our team. The first was the lack of an analytical tool to model the organisational dynamics regarding both the changes of agent's roles and the life-cycle of groups. Although the $\mathcal{M}$OISE$^+$ specification language

---

[1] Note that cows stuck to clusters which were difficult to move in the competition simulation (cows behaved differently from the initial scenario description). Thus, even though the formation seems efficient, the best strategy to herd large clusters of cows was to herd them separately.

plan merge(g_i)   // g_i is the herding group of the agent using this plan

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 *scouter* in g_i

Algorithm 1: Group merging. The leaders of herding groups check a possible merging with all other herding groups that have a smaller ID number.

function cluster(V, m) ;   // V is the set of all seen cows in the group
                                        // m is the maximum number of cows in the cluster

$C \leftarrow \{$ the cow in V nearest to the corral $\}$;   // C is the resulting cluster
repeat
    $add \leftarrow false$
    forall $v \in V$ **do**
        if *some cow in C sees v* **then**
            move v from V to C
            $add \leftarrow true$
until $\neg add \vee |C| \geq m$

Algorithm 2: Cluster function. The leaders of herding groups use this function to compute the current cluster of the group.

is used at runtime to *constrain* the dynamics of the organisation (e.g., by the cardinality of roles), it does not help the agents to make decisions about when and what exactly to change. The second problem was the lack of suitable tools to debug the team. Even with the ***Jason*** mind inspector, communication sniffers, and organisational GUIs, finding bugs take most of the development time. Due to its high abstraction level, BDI and organisational programming require new kinds of debugging tools. These two issue will be the subject of our future work. The third difficulty was due to the various problem-dependent parameters (e.g., perception range, repulsion force, cluster size, herding group size) that influenced the collective behaviour, differing from one scenario to another. This led us to long tuning activities to obtain adequate behaviors, without any automatic learning phase. Such an exploration of the parameter space may be an interesting challenge, but hardly generalisable and outside of our interests.

References

1. Bordini, R.H., Hübner, J.F., Tralamazza, D.M.: Using Jason to implement a team of gold miners. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA 2006. LNCS, vol. 4371, pp. 304–313. Springer, Heidelberg (2007)
2. Bordini, R.H., Hübner, J.F., Wooldrige, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. John Wiley & Sons, Chichester (2007)
3. Hübner, J.F., Bordini, R.H.: Developing a team of gold miners using Jason. In: Dastani, M., Fallah Seghrouchni, A.E., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS, vol. 4908, pp. 241–245. Springer, Heidelberg (2008)
4. Hübner, J.F., Sichman, J.S., Boissier, O.: Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels. Int. J.Agent-Oriented Software Engineering 1(3/4), 370–395 (2007)
5. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS (LNAI), vol. 1038, pp. 42–55. Springer, Heidelberg (1996)