AF-ABLE: System Description

Howell R. Jordan, Jennifer Treanor, David Lillis, Mauro Dragone,
Rem W. Collier, and G. M. P. O'Hare

School of Computer Science and Informatics
University College Dublin
howell.jordan@lero.ie, {jennifer.treanor, david.lillis, mauro.dragone,
rem.collier, gregory.ohare}@ucd.ie

Abstract. This paper describes our entry to the Multi-Agent Programming Contest 2009. Based on last year's entry, we incorporated new features of the employed agent programming language and adopted a simplified hierarchical organisation metaphor. This approach, together with a re-design of the task allocation algorithm, should result in increased efficiency and effectiveness.

1 Introduction

Based on our entry in the ProMAS Agent Programming Contest 2008 [5], this paper discusses our submission for 2009 and the alterations to the previous entry. Last year's entry was specified and designed with the SADAAM methodology [1], with a hybrid architecture based on the SoSAA [4] and using the Agent Factory Programming Language (AFAPL)[2]. This year's entry uses new features of AFAPL [3] to better organise the herding agents. With this methodology we use a modified form of the Agent Factory framework [2]. We once again use a hybrid architecture based on the SoSAA architecture [4].

2 Software Architecture

Our two-tiered architecture is based on the SoSAA robotic framework [4]. The upper layer is an intentional multi-agent system. The second layer is a low-level component based infrastructure. This combination of layers allows for intentional reasoning in the upper layer along with the support for multi-agent organization separate from lower-level actuation-based functionality.

Our system uses AFAPL, taking advantage of new features introduced since last year's contest. The basic features of AFAPL still include a model of beliefs, the notion of commitment to a course of action, a set of commitment rules, a simple language for specifying plans and support for specifying ontologies. Additionally, the language now also includes additional practical plan operators, a role programming construct and the notion of a goal.

Agents are modeled as mental entities. They have a mental state that consists of a number of primary mental attitudes along with a set of supporting meta attitudes, as shown in Figure 1.

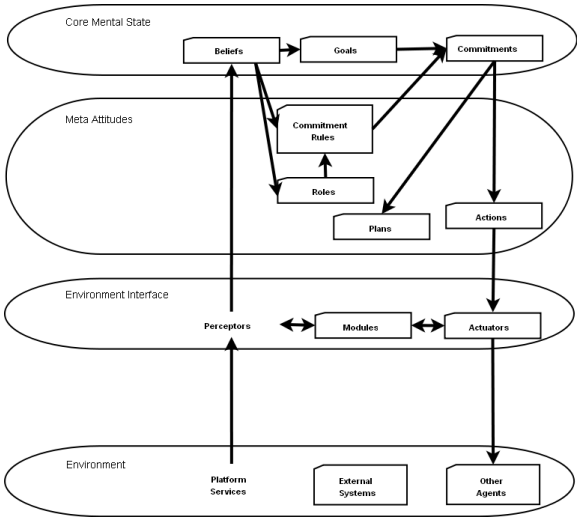

Fig. 1. AFAPL framework structure

The new approach is based around two types of agent: one commander agent and several herder agents. The commander agent analyses information from the herder agents and assigns appropriate roles to each of the herder agents, be it to 'herd', 'explore', 'push the button', etc. In this setup each agent can be told to perform a variety of roles at various times.

The new 'goal' language construct of AFAPL is key to our hierarchical command-and-control structure, as it is be used to assign roles to herder agents. The commander agent is request herder agents to adopt new goals and thereby force them to change roles. However, the herder agent may postpone or reject the new goal if the current goal is both still viable and nearly complete. Thus, oscillating between behaviours should be kept to a minimum.

3 Agent Team Strategy

Last year's strategy suffered mostly from difficulties with the adopted auction approach. Agents tended to prefer exploring and single-agent herding to the formation of groups to herd larger numbers of cows. Figure 2 illustrates this with a case where a herd of 15 cows were driven through the bottleneck of the 'RazorEdge' scenario (where the average score was a mere 5 cows). However, having pushed this herd through the gap, the agents decided to explore for more cows, rather than continue herding. This demonstrates that the weightings attached to the various scenarios for the purposes of the auctions were suboptimal.

For this year's entry, we therefore opted to eschew the auctions and, decided on a commander agent in charge of several herder agents. The commander agent

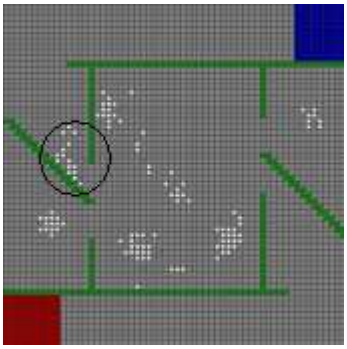

Fig. 2. Example of RazorEdge scenario

is informed of herder agents' map-related beliefs and their current commitments. The commander agent builds up an overall view of the system and the environment. Based on the current commitments, it decides on a course of action and provides the herder agents with new or amended beliefs and commitments.

Our task allocation algorithm, will be based upon several factors. Dependent on the various roles, different considerations are taken into account for a cost/benefit calculation. Herding requires considering such factors as: the number of cows in a herd, (i.e. the reward); the distance from the corral, (i.e. the time cost); the number of herders used and available; the distance of herders to the herds; the proximity of known opponents. Exploring also takes distance to the corral into account. The location, path to and extent of unexplored spaces on the map are of interest. General weighting factors to be considered regardless of the role are the time left to the end of the game and the number of known cows left on the field.

No offensive or defensive moves in relation to our opponents were made last year. After observation of some successful competitive tactics, we intend to explore the possibility of behaviours to protect the herded cattle and to provide more of an adversarial environment to our opponents.

4 Discussion

As noted in the report on last year's entry [5], our performance suffered due to runtime exceptions. Our efforts this year are centred around improving the quality of our agent-layer code by using better modelling techniques, a simplified architecture, and by taking advantage of advanced language features.

Figure 3 depicts a simplified class diagram showing all the dependencies between the most important classes implemented for this year entry. In contrast to the previous version, new agents share a direct access to a global WorldModel object that is updated with any new agents perception. The TeamLeader agent periodically examines the WorldModel class to find the best assignments for

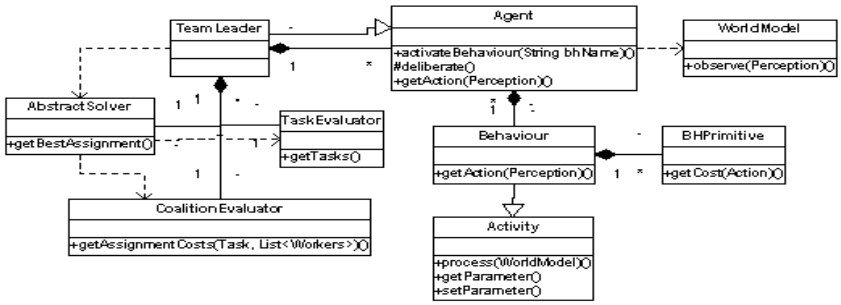

Fig. 3. Class diagram

each agent in the team. First of all, the TaskEvalutor class runs a clustering algorithm on the cows registered in the WorldModel in order to group them in herds to be herded to our corral. In addition to herding tasks, the TaskEvaluator generates exploring tasks and open-fence sub-tasks whenever a primary task requires agents to pass through a fence. For each task, a cost-value analysis is carried out by estimating the number of iterations needed to execute the task.

Secondly, the AbstractSolver examines each task, and tries to assign them to the most suitable team. It does this by removing the selected agents from the pool of available agents each time. By altering the ordering criteria we use to order the tasks, we can realise different strategies; from the greedy sequential auctioning to more sophisticated assignment schemas.

As an example of the progression of the code, we now compare two functionally similar portions of code from the two submissions. Each represents a herder agent that can be told (using a 'fipaMessage') to explore an area of the board surrounding a pair of x, y co-ordinates. The behaviours 'MoveToViaShortestPath' and 'Explore' are implemented in the underlying Java layer. On receipt of a 'fipaMessage', the first excerpt simply installs the given task as a belief. When the given task is an 'Explore' task, the agent moves to the desired location. Once the agent arrives at its destination, the Java layer installs a 'closeTarget' belief, and this, in turn, triggers the 'Explore' behaviour.

```
BELIEF(fipaMessage(request, sender(?name, ?addr), doTask(?task, ?params))) =>
BELIEF(chosenTask(?task, ?params));

BELIEF(chosenTask(Explore, params(?x, ?y))) =>
COMMIT(?self, ?now, BELIEF(true),
    activateBehaviour(MoveToViaShortestPath(x, ?x, y, ?y, tolerance, 5)));

BELIEF(closeTarget) & BELIEF(chosenTask(Explore, params(?x, ?y)))
    & BELIEF(active_behaviour(MoveToViaShortestPath)) =>
COMMIT(?self, ?now, BELIEF(true),
    activateBehaviour(Explore));
```

On receipt of a similar 'fipaMessage', the second excerpt adopts the given task as a goal. The Agent Factory interpreter selects a compatible plan with a

postcondition that results in the goal being achieved; our 'Explore' plan is the only plan in the agent's plan library and is therefore chosen. The 'Explore' plan then performs its three component actions (move to the location, explore it, and believe that the task is complete) in parallel.

```
BELIEF(fipaMessage(request, sender(?name, ?addr), doTask(?task, ?params))) =>
COMMIT(?self, ?now, BELIEF(true),
    ADOPT(GOAL(completedTask(?task, ?params))));

PLAN explore(?x, ?y) {
    PRECONDITION BELIEF(true);
    POSTCONDITION BELIEF(completedTask(Explore, params(?x, ?y)));
    BODY PAR (
        activateBehaviour(MoveToViaShortestPath(x, ?x, y, ?y, tolerance, 5)),
        DO_WHEN(BELIEF(closeTarget),
            activateBehaviour(Explore)
        ),
        DO_WHEN(BELIEF(completed(Explore)),
            ADOPT(BELIEF(completedTask(Explore, params(?x, ?y))))
        ));
}
```

It can be seen that the second excerpt is more cohesive in that all of the exploring-related agent code is contained in a single plan. We hope that agent code written in the new style will prove a lot easier to test and debug.

5 Conclusion

This paper presents an overview of our submission to the ProMAS Multi-Agent Programming Contest 2009. We aim to improve on last year's result by implementing changes to last year's system including the utilisation of new features of AFAPL2.

References

1. Neil Clynch and Rem W. Collier. SADAAM: Software Agent Development An Agile Methodology. *In Proceedings of the Workshop of Languages, methodologies and Development tools for multi-agent systems (LADS'07)*, 2007.
2. Rem W. Collier. *Agent Factory: A Framework for the Engineering of Agent-Oriented Applications.* PhD thesis, School of Computer Science and Informatics, 2002.
3. Rem W. Collier. AFAPL2: Development Kit. online, 2008. `http://www.agentfactory.com/index.php`, accessed on 28th April 2009.
4. Mauro Dragone, David Lillis, Rem W. Collier, and G.M.P O'Hare. SoSAA: A Framework for Integrating Components and Agents. *SAC '09*, 2009.
5. Mauro Dragone, David Lillis, Conor Muldoon, Richard Tynan, Rem W. Collier, and G.M.P O'Hare. Dublin Bogtrotters: Agent Herders. *In Post-Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, ProMAS*, 2008.