Dublin Bogtrotters: Agent Herders

M. Dragone, D. Lillis, C. Muldoon, R. Tynan
R. W. Collier, and G. M. P. O'Hare

School of Computer Science and Informatics
University College Dublin
{mauro.dragone, david.lillis, conor.muldoon,
richard.tynan, rem.collier, gregory.ohare}@ucd.ie

Abstract. This paper describes an entry to the Multi-Agent Programming Contest 2008. The approach employs the pre-existing Agent Factory framework and extends this framework in line with experience gained from its use within the robotics domain.

1 Introduction

This paper outlines the approach adopted for the Dublin Bogtrotters entry in the PROMAS Agent Programming Contest. For the purposes of the competition, we adapted the pre-existing Agent Factory (AF) framework [1, 2], making use of our previous experience in the area of robotics [3]. As is described in Section 3, we have developed a 2-tier hybrid agent architecture that is loosely based on the SoSAA architecture [3]. This system is implemented using an agile methodology [4], outlined in Section 2, that supports agile modelling and test driven development. Some details on the strategies that were employed in the competition are discussed in Section 4.

2 System Analysis and Design

The system was specified and designed with the SADAAM methodology [4], which supports agile modelling and test-driven development. In this methodology, agile modelling is realised using a combination of Agent UML Protocol Diagrams and customised UML Case Diagrams and Activity Diagrams. As is usual in such methodologies, rather than deliver a comprehensive system design, we used our design notation only as a mechanism to clarify how certain core system features were implemented. SADAAM was chosen because it has previously been used in conjunction with AF to develop agent-based systems.

3 Software Architecture

The overall system architecture (figure 1 is oriented around a core set of herder agents, that are supported by a number of ancillary agents, including: a herd

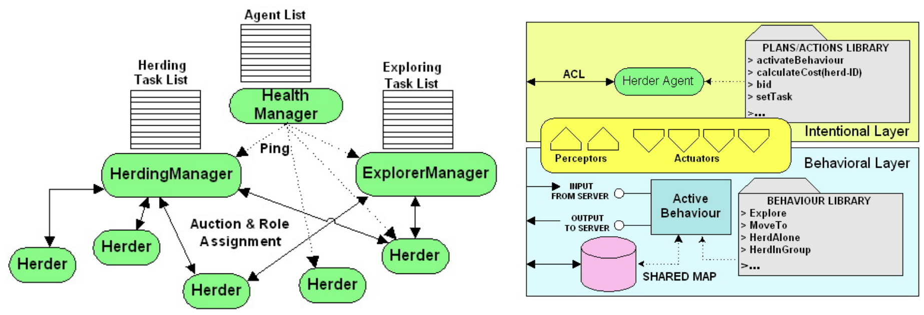

Fig. 1. Bogtrotter System (Left) and Hybrid Agent Architecture (Right)

manager agent that was responsible for creating herders and allocating usernames to them; a health agent, that monitored the health of other agents (see Section 4); and a strategy agent that oversaw potential strategies.

While all of the ancillary agents were implemented using only the Agent Factory Agent Programming Language (AFAPL) [5], the core herder agents were implemented using a hybrid agent architecture that is inspired by the SoSAA robot software framework [3]. This framework advocates the adoption of a two-tier architecture for robotic systems that combines an intentional multi-agent system with a low-level component-based infrastructure. The idea behind this approach is that the upper agent layer enhances the lower-level mechanisms by way of their intentional reasoning abilities and support for multi-agent organisation. For instance, agents can negotiate the use of system resources, and also supervise the low-level communication mechanisms that are used to exchange non-ACL messages amongst low-level components.

For this competition, the framework was realised through a combination AFAPL for the agent level and a simple Java-based architectural framework that provided basic mechanisms that attend to the run-time and data-distribution requirements of lower-level components. Interaction between these layers was facilitated by a clear and standardised interface which was realised through AF platform services. An overview of this architecture can be seen in figure 1.

The AFAPL language models agents as mental entities whose internal state consists of beliefs and commitments. Informally, beliefs represent the agent's current state of its environment, while commitments represent the outcome of an underlying reasoning process through which the agent selects what activities it should perform. In AFAPL, an agent has both primitive abilities, in the form of directly executable actions, and composite abilities, in the form of plans built from plan operators such as `SEQ` (sequential execution) and `FOREACH` (plan expansion). Execution of an AFAPL program involves the update of the agent's mental state by repeatedly applying an internal reasoning process that combines: update of the agents beliefs via perception of the environment through a set of auxiliary Java components, known as perceptors; the adoption of new commitments though the evaluation of a set of commitment rules, which map belief states onto commitments that should be adopted should that state arise; and the realisation of commitments through the performing of actions that are implemented through a set of auxiliary Java components, known as actuators.

The rationale for the hybrid architecture was to delegate many of the details of the herders' control to a reactive behavioural system, whose operation was linked to the underlying sensory-motor apparatus. This system consisted of various components that formed the agents primary skill-set, including: simple action patterns (`stop`, `turn`, `move_backward`) and more complex patterns that attempt to maintain or achieve simple conditions between the agent and the environment (`follow_border_obstacle`, `follow_border_herd`, `move_toward_target`, `explore`). Additional components embodying sensor data processing routines were also included, that were used to recognise features in the agent's world model or to signal events generated by the currently active behaviour (`path_obstructed`, `close(target)`). These features and events were passed to the agents belief set via perceptors, where they were used as perceptual triggers for the activation of other behaviours specified by AFAPL plans.

4 Agent Team Strategy

Central to our approach were the herder agents. These agents were responsible for controlling the behaviour of the herders and were organised into teams that were formed to achieve a particular task (e.g. exploration, herding a particular group of cows). Each team was controlled by a team manager agent. Resource allocation was carried out by team managers holding auctions in which the herders would bid using a greedy bidding strategy to join particular teams or cover certain roles. These auctions were intended to have three notable benefits. Firstly, because of the greedy nature of the agents' bidding strategy, the time needed to carry out these auctions was minimal. Secondly, an agent that is most suited to a task was most likely to win an auction (e.g. for a task to explore a particular part of the map, the bidding agent that is closest will win the auction). Finally, it enabled dynamic reallocation of agents' priorities. For example, as more cows were discovered, agents could switch from exploring to teams that engaged in herding, returning to exploring once the cows were gathered.

A key strategy underlying our approach was the use of hybrid communication by combining Agent Communication Language (ACL)-based communication and blackboard-based communication. The ACL-based communication was realised the AF implementation of the FIPA-ACL standards, and the blackboard-based communication was realised through a shared map that was accessible via Java RMI. The shared map exported a distributed update interface to all the agents in the system. Through this interface, each agent could update the server with its own observations and receive in return an update of all the observations collected by the rest of the team.

In each simulation step, each herding agent had a limited time slice to send an action message. The message sent was determined by the agent's current low-level behaviour. For instance if the agent was engaged in a `move_toward_target` behaviour, the next message would be a movement in a direction that aids in the fulfillment of this goal. At all times, however, the higher-level management agents reasoned about their perceived current state of the world in order to optimise the overall strategies of the participating agents. This meant that an agent's active behaviour could change because of an instruction from a team leader, or due to a change in team membership brought about by an auction.

Robustness was a high priority in participating in the contest. Herding agents who were still active but had become disconnected from the competition server needed to be capable of re-establishing that connection. Additionally, a health management system monitored both agents and the agent platforms to detect any failures that may occur. A failed agent was replaced with a new agent of the same type and a failed platform resulted in all the agents formerly residing on it being recreated on other platforms.

5 Discussion

Much of our effort was in designing and implementing the infrastructure framework as this was our first entry in the contest. Unfortunately by the beginning of the contest we had not completed testing and tuning of our functional components to the extent we would have wished. As a result, our behavioural functions occasionally encountered unexpected exceptions that had not arisen during development, and the resource allocation auctions were not optimised in terms of evaluating the costs and the benefits of engaging in the various tasks.

The difficulties with the auction were the greatest limiting factor in our performance. Agents tended to prefer exploration and single-agent herding to the formation of groups to herd larger numbers of cows. Figure 2 illustrates this through a case where a herd of 15 cows were driven through the bottleneck of the "RazorEdge" scenario (a map on which 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 pushing this particular herd to the corral. This shows that while the simple agent behaviours were effective, the weighting attached to various scenarios for the purposes of the auctions were sub-optimal.

We believe that this year's scenario was a very useful and well organised attempt to promote multi-agent programming. Since we have now a working infrastructure framework, we are in a position to be more competitive in the next contest, as we will be able to focus our work on adapting our system to the new application scenario and focus on our real interest: multi-agent coordination.

We feel that the slowness of the simulator was a big obstacle to our development plan in this year's first entry to the contest, as it was difficult to run a sufficient number of simulations to test different task and environment configurations. In the future, this would be also an obstacle to the adoption of machine learning techniques that may require substantial amounts of training data.

We believe that real systems need MAS self-organisation techniques that are shaped by ACL-based coordination but that still manage to produce reliable

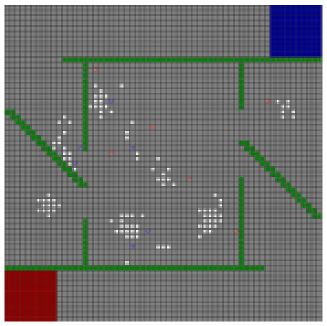

Fig. 2. RazorEdge Scenario: A herd is pushed through the bottleneck

and efficient control. Because of the large time slices allotted between moves, teams are currently able to make use of computational expensive deliberation phases before transmitting an instruction to the central server. While our system architecture allowed us to produce very fast response in the behavioural layer, we felt that the present organisation of the contest does not place a high value such a feature. We believe that reducing the size of this time slice will force participants to develop solutions that are closer to their real world counterparts.

6 Conclusion

This paper presents an overview of our submission to the ProMAS Multi-Agent Programming Contest. The solution developed employed a hybrid agent architecture, whose upper deliberative layer was realised using AFAPL, and whose lower layer consisted of a reactive architecture. High-level system behaviours were designed using an agile modeling process, and were implemented in AFAPL. These high level behaviours drove the adoption of various lower level reactive behaviours, including obstacle avoidance, herding, and exploring.

References

1. O'Hare, G., Jennings, N.: Foundations of Distributed Artificial Intelligence. Wiley-IEEE (1996)
2. Collier, R., O'Hare, G., Lowen, T., Rooney, C.: Beyond Prototyping in the Factory of Agents. Multi-Agent Systems and Application III, Lecture Notes in Computer Science (LNCS 2691), (2003)
3. Dragone, M.: Sosaa: An agent-based robot software framework. `http://csserver.ucd.ie/%7emdragone/pubs/MauroDragonePhdThesis.pdf` (2008)
4. Clynch, N., Collier, R.: Sadaam: Software agent development - an agile methodology. Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems (LADS'007), Durham, U.K. (2007)
5. Collier, R.: Agent Factory: A Framework for the Engineering of Agent- Oriented Applications. PhD thesis, University College Dublin (2002)