Herding Agents - JIAC TNG in Multi-Agent Programming Contest 2008

Tuguldur Erdene-Ochir, Axel Hessler, Jan Keiser, Tobias Küster, Marcel Patzlaff, and Alexander Thiele

DAI-Labor, Technische Universität Berlin, Germany

Abstract. A competition always shows the performance of the participants. We have developed the JIAC TNG agent framework for two years and take this as a chance to see where we stand. This paper describes our approach to the contest scenario from a software engineering point of view. We also show different strategies which we are considering to use.

1 Introduction

The JIAC TNG agent team has been prepared by members of the Competence Center Agent Core Technologies of DAI-Labor at Technische Universität Berlin. We use the new JIAC TNG agent framework with accompanying toolkit, which have been created in the course of several projects at DAI-Labor.

2 System Analysis and Design

2.1 JIAC TNG Agent Framework

Java-based Intelligent Agent Componentware - The Next Generation (JIAC TNG) is the successor of the JIAC IV agent framework. It contains a new architecture, which integrates different standards and technologies (e.g. ActiveMQ, OWL, JMX and web services) and provides a high-performance communication infrastructure. JIAC TNG also comes with a customised methodology and a number of tools integrated in the Eclipse IDE. The new framework incorporates concepts of the service-oriented architectures such as an explicit notion of services as well as the integration of service interpreters in agents. Interpreters can provide different degrees of intelligence and autonomy, allowing technologies like semantic service matching or service composition. JIAC TNG supports the user with a built-in administration and management interface, which allows hot deployment and configuration of agents at runtime.

The JIAC TNG methodology is based on the JIAC TNG meta-model and derived from the JIAC IV methodology. JIAC TNG has explicit notions of rules, actions and services. Composed services are modeled in BPMN and transformed to DFL, which is the language of the above mentioned service interpreters. We distinguish between composed services and infrastructure services. The former can be considered a service orchestration with some enhancements (e.g. service matching) whereas the latter ascribes special services that agents, composed services or basic actions can use, e.g. user management, communication or directory services. Rules may trigger actions, services or updates of factbase entries. Besides it is possible to access rules from within the DFL making it possible to implement complex service orchestrations. Basic actions are exposed by AgentBeans and constitute agentroles, which are plugged into standard JIAC TNG agents. The standard JIAC TNG agent is already capable of finding other JIAC TNG agents and their services, using infrastructure services and it provides a number of security and management features.

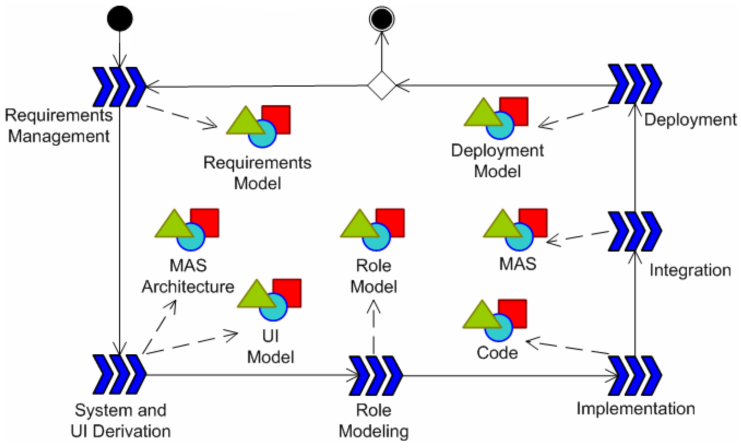

Fig. 1. JIAC methodology - iterative and incremental process model in SPEM [1] notation

As shown in Figure 1, the development process starts with collecting domain vocabulary and requirements, which then are structured and prioritised. Second, we take the requirements with the highest priority and derive a MAS architecture by listing the agents and create a user interface prototype. The MAS architecture then is detailed by creating a role model, showing the design concerning functionalities and interactions. We then implement basic actions and composed services, which are plugged into agents during integration. Agents are deployed to (one or more) agent nodes and the application is ready to be evaluated. Depending on the evaluation we align and amend requirements and start the cycle again with eliminating bugs and enhancing and adding features until we reach the desired quality of the agent-based application.

For any part of the JIAC TNG meta-model we provide an editor (source code as well as visual editor) in the JIAC TNG IDE for easy agent and application development. Reuse is supported by a plugin that allows search and retrieval of components and solutions. A context sensitive help and a number of interactive tutorials complete the JIAC TNG toolbox.

2.2 Single Agent Behaviour

Following our methodology we started collecting the simulation domain vocabulary and created the appropriate ontologies containing concepts as cows, cells, obstacles and so on. Next we came up with some necessary basic requirements such as the ability to communicate with the server as well as with other agents, to sense the world and to walk.

In a further iteration, some higher level services have been designed, embodied into special roles such as *Explorer* or *Herder*. In particular, we created behaviours for exploring the grid world, moving to a certain position, and predicting a cows next move. The explorer role has capabilities to systematically search the terrain for cows, the herder role is able to direct the cows to the corral.

Each Agent has its own perceptions, thus developing its own world model over time and also choosing its actions based on this world model. However, we believe that team communication and coordination will play a vital role in solving the contest scenario. Therefore agents will need to communicate with each other, sharing not only knowledge but also intentions. Based on the ontologies, agents will communicate facts which are used by other agents to update their own world model. Thus, what each agent knows is the sum of all the agents perceptions. As mentioned, agents also share their intentions, e.g. ”I am going to cell x,x” enabling team coordination. Coordination strategies are specified in more detail later.

3 Software Architecture

JIAC TNG combines a scalable component framework, a knowledge memory, an action invocation architecture, a service interpreter and a communication infrastructure. Additional features are a runtime environment, tools, and libraries. JIAC TNG has been implemented in Java.

JIAC TNG provides the DFL (Declarative Formal Language) programming language. It incorporates OWL ontologies, messages, and a (procedural) scripting part for composed actions.

4 Agent team strategy

When dealing with MAS we always assume that the MAS must be worth more than the sum of its parts. We addressed this in a number of iterations dealing with communication, coordination, and cooperation.

Agents cooperate on a number of levels. First, they share their perception. Next, we enabled our agents to share their intentions (such as ”I plan to direct cow at cell x,y”). This prevents agents from going to the same unknown field or even exploring the same region of the world. Now every agent can appraise from what it knows if it will be better to leave the team member alone or to take the intention as its own when its more promising. In addition, agents can explicitly call for help, e.g. when trying to direct a cow to the corral.

We are currently investigating different coordination strategies. However, we will see how they perform once a test environment is available. It needs to be tested whether cows can be directed usefully by one agent or if two or more agents are necessary. If so, agents would need to coordinate their moves in order to direct a cow. We are also thinking about directing a small herd of cows with 3 or 4 agents by making use of swarm-like behaviour models, such as keeping a minimum and a maximum distance to each other and at the same time covering a preferably wide area so that cows are forced to move to one direction. If it turns out that cows cannot be directed usefully when staying in a herd, we will try to separate a single cow from the herd and direct the cows one by one to the corral.

Following the discussion about nonmoral agents, we have decided not to implement such a strategy, although we did discuss such strategies, e.g. disturbing the opponents corral if at the end of a game the opponent had more cows than us.

5 Discussion

There are still issues left. Concerning the so called nonmoral behaviour of agents, it is easy to foresee that - depending on the anxiousness of cows of agents - if such a strategy is used no team will eventually score (maybe by chance) and thus counteracting the idea of the contest. However, putting such fears aside, the scenario poses an interesting challenge regarding team coordination strategies, especially if more than one agent is needed to usefully direct a cow. In addition, this also depends on the terrain, making things a bit more complex, but not necessary more difficult since obstacles decrease a cows possibilities to move.

Our approach that every agent shares the complete perception with all other agent might not be useful in really large worlds. Also it produces a lot of communication overhead when perceptions are large too. It might be better not to share the complete perception but only an important part of, although it needs to be defined what exactly important means. Another way might be to equip agents with an perception filter that deleting incoming unimportant information before its own perception gets updated.

We have not talked about timestamps when dealing with perceptions. Since each cow moves every 3 steps, information about a cows position is probably not correct after some time. Still it might be possible that the cow didn't move. Such probabilistic information can be turned certain if the agents could predict a cow next move w.r.t a certain environment. For a coordinated team effort it will be vital to precalculate the cows movement.

Observation of opponents would be a possible extension as well as to guess what they plan and then to crisscross it or adapt our own strategies. A simple strategy could be to let one agent always interfere the herding efforts of the opponent. Yet we are not sure if this behaviour will be regarded as nonmoral and will leed to disqualification.

6 Conclusion

We have shown that it is easy and effective to solve the simulation problem using the JIAC TNG agent framework. We first collect the domain vocabulary and basic requirements from the scenario description, derive a monitor GUI and the basic agent architecture. Then we gather basic capabilities and interactions in roles and implement them. Implemented roles are plugged into the agents and deployed on the agent node. After evaluating the performance of our agents we will collect new requirements, bugs and feature enhancements and start a new iteration.

References

1. Object Management Group: Software Process Engineering Metamodel (SPEM) Specification. Version 1.1. Object Management Group, Inc. (2005)