Distributed Agent Platform for Advanced Logistics

Tamas Mahr
Almende BV
Westerstraat 50
3016DJ Rotterdam, The Netherlands
tamas@almende.com

Mathijs de Weerdt
Delft University of Technology
Mekelweg 4
2628CD Delft, The Netherlands
M.M.deWeerdt@ewi.tudelft.nl

Categories and Subject Descriptors

I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*

General Terms

Design, Experimentation, Performance, Reliability

Keywords

Agent platform, Logistics

1. INTRODUCTION

The system that is demonstrated by live demo software through the internet is an agent platform developed for the DEAL (*Distributed Engine for Advanced Logistics*) project. It could be interesting for anyone developing agent platforms themselves, researching coordination techniques, or involved in logistical research. The demo appears in a single browser window as a Java applet that connects to the agent engine through the internet.

The primary goal of the DEAL project is to enhance the utilisation of trucks for transportation companies. This is to be realised by a distributed system that connects all trucks, orders, planners, and customers. Each of these entities is modelled by an agent. In a logistics setting, the main problem is to find the best truck for a container while finding appropriate routes (plans) for all trucks. In this project a feasible solution for this problem is constructed by distributed, communicating agents. For example, a container agent may request its transportation from a truck agent, and a truck agent can refuse such a request if it may be able to construct a more efficient route without this container. Similar systems are introduced in [1][2], with the difference that they did not model the orders by agents, instead the company and truck agents took care of all aspects. Modelling the system by agents has the benefit that the load can be distributed among several computers, and by putting the emphasis on coordination of agents exceptional situations like truck breakdowns, or traffic jams, accidents can be modelled and handled more efficiently.

The agent platform that supports the system described above has two layers. In the lower layer, called *Abbey*, a thread pooling mechanism is employed to provide processor resource to the agents. This is described in Section 2. Agents in the upper layer communicate through asynchronous messages that are coordinated by a special method (see Section 3). Section 4 discusses the logistical application and Section 5 draws the conclusions.

2. ABBEY

The lower layer is designed to transparently support the execution of a large number of agents (in parallel) by 'blind workers'. In our vocabulary a *monk* is such a 'blind worker', who runs in one thread all the time. One can dispatch an OPEN task to a monk and it will execute it without having any idea what it is about. When it has finished the execution of the task, it sets the task state to READY and looks for another OPEN task.

An *abbey* consists of a number of monks and a task pool (see Figure 1). It creates the monks, manages them and handles the task dispatching. On the one hand, the abbey ensures that there are always enough monks to execute the tasks, and on the other hand it prevents the system to be overloaded by numerous monks (threads). If it is short of system resources it can decide to spawn a new abbey on a different machine and transfer some of the agents to ease the load on the current system.

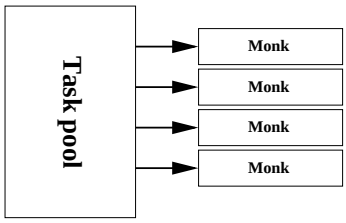

Figure 1: Monks in an Abbey

Tasks that are carried out by the monks are function pointers and arguments for the functions. When a monk receives an open task it calls the function with the specified arguments. The result is stored in the task structure and

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

then it is declared to be ready. Dispatching a task means to put it into the task pool as an open task. The agent platform dispatches agents' methods to the monks when they need to be executed.

The services of an abbey implement a non-preemptive virtual-thread scheduling of the methods of the agents. Methods are dispatched to monks when they are to be executed and there is no way to interrupt their execution (non-preemptive). A monk executes different methods of different agents subsequently, but from a perspective of an agent it looks like it has its own (virtual) execution thread.

3. AGENTS

Agents are defined by attributes and methods (like objects). They communicate through asynchronous messaging. The messages are delivered by the agents themselves through the use of a queue per agent. If an agent is busy answering a message, the next message is queued by the sender.

There is a designated method in every agent to handle received messages (called *Coordination* in Figure 2). This method is responsible for dispatching the right method for the first message in the queue to a monk. This organisation helps to separate the coordination from the processing in the agents. The coordinating method chooses the appropriate method based on the agents state, which is modified by the processing methods. For example let us assume that after an agent has sent a message it has to wait for an answer. In this case it sets which method should be called when the answer arrives and terminates. When the answer is handed to the coordinating method, it will dispatch the method set by the processing method.

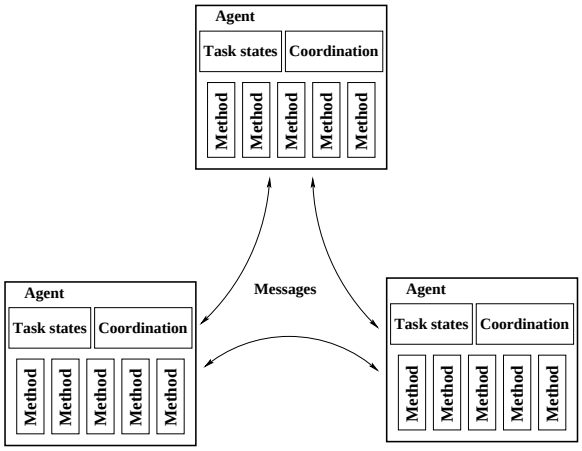

Figure 2: Agents communicating by asynchronous messages

4. APPLICATION

In our logistical setting in the simplest situation there are two types of agents: orders and trucks. Order agents describe customer orders (size, origin, destination, etc.) and try to find the cheapest truck. Truck agents describe real world trucks (size, location, etc.) and try to collect a valuable set of orders that could be efficiently transported.

One advantage of agent modelling is that the constraints and preferences of different roles in the system (trucks, orders) can be taken care of separate agents. Orders concentrate on getting transported by a cheap truck, and trucks concentrate on collecting orders that give an efficient route.

The way orders search for trucks is by auctioning. Every order is auctioned separately when it is introduced into the system. Trucks place bids, and in the end the order agent chooses the cheapest offer.

To place their bid, truck agents have to calculate their costs regarding the transportation of the order in question. For this they have to plan as if they would have accepted the offer (solve a *Travelling Salesmen Problem*). Having solved that, the bid they place can be based on their additional costs regarding the transportation of the order. To keep the execution time below polynomial bounds the truck agents use the insertion heuristic for planning. This results in suboptimal plans. To fix this we are considering to apply decommitment techniques. Since trucks and orders are modelled by agents, they are capable of monitoring their environment. They can recognise situations where it would be better either for the truck or for the order to decommit. By moving around orders among trucks the sub-optimality of the truck-plans can be decreased.

5. CONCLUSION

The contribution of the system demonstrated is twofold. It introduces an agent platform and an application of agents in a logistical setting.

One goal of our platform is to manage resources efficiently and to ensure balanced system performance for all agents. A pool of threads is maintained to execute agent methods on demand and the architecture enables the use of several hosts. However, this functionality of distributing the system on hosts is not implemented yet and is one of our plans for the future.

The other design issue is to enable research on coordination. The platform supports the separation of processing and coordination in the agents' code. Agents communicate by asynchronous messages and the selection of a method in response to a message is driven by state variables. Receiving a message and react on it is only half of the story, though. Messages sent are still explicitly addressed. Future work involves the extraction of the knowledge about other agents from the processing code.

As an application of the system a logistical setting is used. Trucks and orders are modelled by agents to enable proper distribution of responsibilities and quick and good quality reactions on unexpected events. Modelling the system in such a way causes that the solution of the problem is not directly programmed in the system but emerges as the interaction of all the players. Such an emergence, however, can depend on various parameters and can be hard to achieve.

6. REFERENCES

[1] K. Dorer and M. Calisti. Agent-based dynamic transport optimization. Technical Report WT-2004-05, Whitestein Technologies, 2004.

[2] K. Fischer, J. P. Muller, M. Pischel, and D. Schier. A model for cooperative transportation scheduling. In *Proceedings of the First International Conference on Multiagent Systems.*, pages 109–116, Menlo park, California, June 1995. AAAI Press / MIT Press.