Distributed Agent Platform for advanced logistics

Tamas Mahr [a] Mathijs de Weerdt [b]

[a]Almende BV, Rotterdam, tamas@almende.com
[b]Delft University of Technology, M.M.deWeerdt@ewi.tudelft.nl

Introduction

The system that is demonstrated is an agent platform developed for the DEAL (*Distributed Engine for Advanced Logistics*) project. The primary goal of this project is to enhance the utilization of trucks for transportation companies. This is to be realized by a distributed computer system that connects all trucks, orders, planners, and customers. Each of these entities is modeled by an agent. In a logistics setting, the main problem is to find the best truck for a container while finding appropriate routes 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. A similar system is introduced in [FMPS95], with the difference that they have not modeled the orders by agents, instead the company and truck agents take care of all aspects.

The purpose of the *agent platform* is to study the efficiency of communicating agents. Coordination logic and processing logic in a component (agent) should be separated [PA98]. In other words, the main logic of the agent should not contain explicit references to whom it speaks to or from whom it receives messages. For this we constructed a general purpose agent system *(i)* that can run in a *distributed* manner to scale well when the number of agents grows, *(ii)* that has tight *control of system resources*, and *(iii)* in which the *coordination logic is separated* from the processing logic.

Technology

The agent platform has two layers: one to ensure scalability and another to provide support for asynchronous communication and coordination. The lower layer (*Abbey and monks*) is responsible for the system not running out of resources (it controls threads, distributes the load – fulfilling the scalability requirement), while the upper layer (*Agents)* provides an environment to ensure the separation of coordination and processing in the agent code, and provides support for asynchronous communication.

In our vocabulary a *monk* is a 'blind worker', who runs a thread all the time. Note that there is a one-to-one assignment between threads and monks. 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. The abbey 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 of the current system.

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.

Agents communicate through asynchronous message sending. Messages are called *prayers* and are delivered by the agents themselves. There is a queue for every agent. If an agent is busy answering a prayer, the next prayer is queued. There is a designated method in every agent to receive the prayer and call the corresponding method of the agent. This organization helps to separate the coordination from the processing in the agents.

Demonstration

The agent platform is demonstrated by simulating a situation within the logistics domain. For this demonstration we need a beamer and projection screen. The demonstration can be given either continuously for some time, or in a session of about 15 minutes.

Acknowledgements The research partners in the project besides Almende are the Erasmus University Rotterdam, the Vrije Universiteit Amsterdam (VU), and the National Research Institute for Mathematics and Computer Science (CWI). Our logistical partners are Vos Logistics and Post Kogeko.

References

[FMPS95] Klaus Fischer, Jörg P. Muller, Markus Pischel, and Darius 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.

[PA98] G.A. Papadopoulos and F. Arbab. Coordination models and languages. Technical report, CWI, 1998.