

Beliefs, Desires and Intentions Through the Environment

Tom Holvoet
AgentWise, DistriNet, Dept. Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
3001 Leuven, Belgium
Tom.Holvoet@cs.kuleuven.be

Paul Valckenaers
PMA, Dept. Mechanics
Katholieke Universiteit Leuven
Celestijnenlaan 300B
3001 Leuven, Belgium
Paul.Valckenaers@mech.kuleuven.be

ABSTRACT

BDI-based agent architectures have proven their usefulness in building MASs for complex systems - their explicit attention for coping with dynamic environments is one obvious explanation for this. One large and quite interesting family of MAS applications is characterized (1) by their large scale in terms of number of agents and physical distribution, (2) by their very dynamic nature and (3) by their complex functional and non-functional requirements. This family includes a.o. manufacturing control, traffic control and web service coordination. In this paper, we present an innovative approach to BDI agents which alleviates agent complexity through “delegate MASs”. Delegate MASs consist of light-weight agents, which are issued either by resources for building and maintaining information on the environment, or by task agents in order to explore the options on behalf of the agents and to coordinate their intentions.

1. INTRODUCTION

The term “coordination and control applications” can be coined to refer to a large family of applications which share a number of characteristics. First, in the applications, one can distinguish (1) an underlying physical or software environment, and (2) a software system that is connected to this environment. The underlying environment contains fixed entities or “resources” capable of performing particular operations, as well as mobile entities which can move in the environment. Second, the coordination and control software system is able to observe and direct the entities in the underlying environment. Third, the purpose of the application is to execute “tasks”. Executing a task requires moving through the environment and performing operations by resources. The purpose of the software system is to manage the underlying environment by controlling entities that live in the environment, and coordinating the collective behavior of these entities. Fourth, the underlying system evolves several orders of magnitude slower than the coordination and control software. This allows the software to observe the en-

vironment and to plan ahead. Fifth, the environment itself is highly dynamic. Resources may crash, new resources may be added, connections between resources may be added, lost, or their characteristics (e.g. throughput, speed) may change. Members of this family of coordination and control applications include manufacturing control, traffic control and web service coordination. As an example, in manufacturing control, the environment is the physical world equipped with resources such as machines and conveyor belts, and the tasks are the client orders for fabricating particular products. The software in manufacturing control is responsible for controlling the resources and for guiding the orders through the factory floor. Both the mobile entities (partially fabricated products, vehicles, client software) and the fixed entities or resources in the environment (machines or conveyor belts, roads and intersections, web services) are obvious candidates to be represented as agents - which we call task agents and resource agents respectively.

In this paper, we propose a particular BDI-based approach [1, 4] that aims to avoid part of the internal agent complexity by exploiting the environment [6]. In the approach, so-called “delegate MASs” are issued by the different agents. Delegate MASs consist of light-weight agents (called ant agents). These ant agents explore the environment and bring relevant information back to their responsible task agent, evaluate optional paths, and put the intentions of their task agent as information in the environment. This allows delegate MASs of different agents to coordinate by aligning or adapting the information in the environment according to their own tasks.

2. BASIC COMPONENTS

The basic architecture comprises the task agents, resource agents, and the environment. The environment of the targeted applications is a dynamic directed graph. The nodes in the graph correspond to the resources in the environment and the edges represent connections between different resources.

A resource agent represents a resource in the environment and contains an information processing part for controlling the resource. In manufacturing control systems, a resource agent is an abstraction of the production means such as machines, conveyors, tool holders, material storage or even personnel. A resource can abstractly be described as a set of capabilities, i.e. the operations that the resource is able to perform. Resource agents need to be able to make schedules based on requests from task agents, and must be able to answer “what-if questions”: a task agent may ask a re-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

source agent when and according to what quality standards a particular operation could be performed if the task would arrive at a particular moment in the future.

A task agent represents and controls a task in the coordination and control application, and resides on a (physical or virtual) mobile entity in the environment. A task agent is responsible for performing its task by guiding its mobile unit through the environment, and communicating with resource agents in order to perform operations on the unit. Every task agent is aware of the schemes or plans that can be followed in order to reach its goal. For manufacturing control, the task agents represent (unfinished) client orders and are associated with the pallets with partially fabricated products. The production schemes describe possible sequences of operations on the product in order to obtain the final product. Task agents are obvious candidates to be modeled as BDI agents. Task agents need to deal with the observations of the environment and its entities (beliefs), consider possible options on how to proceed (desires), choose a particular option (intention) and communicate and coordinate this with the other task agents.

Both resource and task agents control the entities in the environment, and obviously need to interact to achieve the goal of the application. A typical interaction amongst these agents goes as follows. When a new “task” enters the system, a task agent is created and connected to the appropriate unit. The agent is aware of the initial state of the task and investigates possible next operations that can be performed. The agent searches and selects a combination of a next processing step and a suitable resource that has the capability to execute the step. When the selected processing step is executed by the resource, the resource reports on the new state of the task. This may or may not be the expected outcome of the operation. Based on this state, the task agent investigates possible next steps. The process repeats until the task is finished.

3. DELEGATE MULTI-AGENT SYSTEMS

A typical approach would be to use direct communication protocols, knowledge engineering and means-ends reasoning to achieve the required coordination. Here, we use delegate MASs for (1) obtaining feasibility information, (2) exploration and (3) propagation of intentions toward the resource agents. The following interesting principles are recognized in “ant-like multi-agent systems”: (1) make the environment part of the solution to handle a complex environment without being exposed to its complexity - ants are quite simple agents; (2) place relevant information as signs in the environment ensuring that locally available data informs about remote system properties; (3) limit the lifetime of this information (evaporation) and refresh the information as long as it remains valid - this allows the system to cope with changes and disturbances. We exploit these principles in our approach and define three types of light-weight agents, which each represent a different delegate MAS and which share a common environment for indirect communication. To distinguish them from task and resource agents, we call the light-weight agents “ant agents” or ants further on. Delegate MASs consist of ant agents that reside in a virtual software environment which reflects the application environment, and in which ant agents can navigate.

Feasibility Ants. Feasibility ants form a delegate MAS that is issued by resource agents. Their purpose is to roam

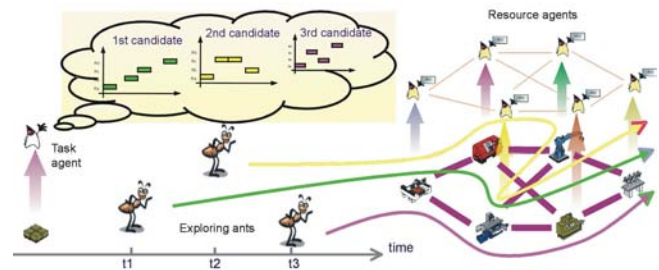


Figure 1: *Exploration ants scout feasible paths*

the environment and, at each node they pass, drop information on feasible paths that start from this node. Resource agents which have not seen a feasibility ant passing by for a particular period will create new feasibility ants themselves at a certain frequency. The behavior of a feasibility ant is as follows. A feasibility ant communicates with the resource agent at its current node, and asks for its capabilities. The ant observes the environment locally and finds out from which other nodes the current node can be reached. The ants can be seen as moving upstream. When arrived at the new node, the ant asks the local resource agent for its capabilities, and merges this information with information of previous nodes. Now this ant knows that from this node, a sequence of operation that requires a capability from the current node followed by an operation that can be performed at its previous node is currently feasible. This information is dropped at the local information space of the current node, as a kind of road sign. Then the ant clones itself for every node upstream, and the process repeats itself. A feasibility ant dies if there are no nodes upstream. The information that is stored in local information spaces is time-stepped, and if not refreshed by another feasibility agent in time, that information disappears.

Exploration Ants. Using the feasibility information available locally, a task agent is able to find out which paths are physically or virtually feasible for achieving the goal of its task. A task agent generates exploration ants at a certain frequency which explore feasible paths. These exploration ants are scouts that each explore a feasible route through the underlying system and evaluate this route. To make the evaluation, an exploration ant follows a path through the environment, and interacts with the resource agents at the different nodes by asking the resource agent *what* the timing, quality or costs would be *if* a task in a particular state would arrive at a particular moment in time. The exploration ant collects this information, and then proceeds to the next node in the path, in which this behavior is repeated. When arrived at the end of the path, the exploration ant returns and reports back to its base, i.e. the task agent that created the exploration ant. Figure 1 illustrates this process for a simple factory (see right-hand side of figure, cf. www.mech.kuleuven.be/benchmarking). The task agent on the left hand side creates three exploration ants for scouting feasible paths.

The list of candidates get refreshed regularly as exploration ants are sent out regularly. Candidates that are not refreshed are removed over time, assuming that these candidates have become invalid or infeasible because of changes in the environment.

Intention Ants. Exploration as described above requires the resource agents to possess an adequate estimate of their future workload. To serve this purpose, task agents generate intention ants, which propagate the intention of task agents through the environment. When a task agent has constructed a set of valid paths to follow, the task agent selects one candidate path to become its intention. The criteria used for this selection depends on the requirements of the task, and is application-specific. Then, the task agent creates intention ants, at a certain frequency, to inform the resource agents that are involved in this intended path. The intention ants follow the selected path, and virtually execute the routing and processing of their selected candidate solution. On their virtual journey, the intention ants acquire travel, queuing, and processing times from the resource agents on their path. Similar as exploration ants, intention ants report back to their task agent to inform the agent about the schedule and performance of the bookings.

The intention information at the resource agent - the booking - evaporates. Task agents must create intention agents to refresh their intention at a frequency that is sufficiently high to maintain their bookings at the resources. While refreshing, a task agent observes the evolution of the expected performance of its current intentions through the reports on the estimated performance that intention ants bring back. This performance estimate is compared to the estimates of the candidate solutions that are found and refreshed by the exploration agents. When the estimated performance of the current intention drops significantly below the estimated performance of other candidate solutions, the task agent may revise its intention. When the task of a task agent reaches the point where a decision needs to be executed, the task agent triggers the action in the underlying system in accordance with the intention.

Task and Resource Agents. Let us now list the responsibilities of task and resources agents. These lists should guide a developer of a coordination and control application to define a concrete software architecture for these agents.

A resource agent is responsible for (1) answering what-if questions from exploration ants, (2) making schedules based on requests for bookings by intention ants - the schedules must obviously respect resource constraints as well as follow a predefined policy for re-scheduling, e.g. when high-priority tasks make a booking, this may reject earlier reservations, (3) keep an up-to-date view on the resource that the resource agent is managing (e.g. observe operation quality and status), and (4) send out feasibility ants if no feasibility ants have contacted this resource agent for a while. These guidelines should suffice to produce a concrete architecture according to the requirements of a concrete application. The resource agent could execute each of these responsibilities in sequence, or one may opt to define concurrent execution of some of these responsibilities.

The responsibilities of the task agents are the following. First, a task agent must have knowledge about its task, about the initial state of the task, and about task schemes. Task schemes describe one or several plans to achieve the goal of the task, and given any intermediate state of the task, the task scheme should provide one or more sub-plans to fulfill the task. Second, a task agent must manage its beliefs. Beliefs on feasibility are readily available from the information space in the environment. Beliefs about explored paths are gathered by exploration ants. Third, this informa-

tion needs to be filtered out, yielding valid paths - the agents desires. Fourth, based on the beliefs about the task and the options, a task agent then chooses or revises its intention. Fifth, at appropriate times and frequency, exploration ants as well as intention ants are sent out. Finally, a task agent is responsible for interacting with resource agents in order to perform operations on its task. Again, these responsibilities are either executed as a sequence, or one may choose for concurrent execution of some of these responsibilities.

4. EVALUATION AND CONCLUSION

In the approach we propose, we stick to the basic philosophy of belief-desires-intention agents, but exploit the environment and delegate MASs, inspired by ant behavior, to realize beliefs, desires and intentions. Based on simulation experiments for a real-world case of manufacturing control, we can state that this innovative approach yields very interesting results for the targeted application domains, both in terms of the reduction of the agent software complexity and in overall performance in a highly dynamic environment [5].

One important difference between an approach using delegate MAS and traditional BDI approach for coordination and control systems is that a delegate MAS design extends Brooks concept of having the world as its own best model, while traditional BDI approaches rely on maintaining world models. The basic idea of having the “world as its own best model” only discusses the present state of the world. In contrast, the delegate MAS approach in this paper extends this idea toward the future state, using exploration and intention ants, while keeping modeling efforts acceptable. Indeed, resource agents only need to be knowledgeable and intelligent about their own small section of the world. Known scalable MAS coordination and control developments are myopic [2, 3], while our approach allows to take future state into account.

5. REFERENCES

- [1] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard, Cambridge, MA, USA, 1987.
- [2] S. Bussmann, N. Jennings, and M. Wooldridge. *Multiagent systems for manufacturing control: A design methodology*, volume XIV of *Springer Series on Agent Technology*. Springer-Verlag, 2004.
- [3] H. Parunak, A. D. Baker, and S. J. Clark. The aaria agent architecture: From manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering*, 8(1):45–58, 2001.
- [4] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
- [5] P. Valckenaers and H. V. Brussel. Holonic manufacturing execution systems. *CIRP Annals-Manufacturing Technology*, 54(1):427–432, 2005.
- [6] D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems state-of-the-art and research challenges. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *E4MAS*, volume 3374 of *Lecture Notes in Computer Science*, pages 1–47. Springer, 2004.