Cooperator-Base ⊎ Task-Base for Agent Modeling: the Virtual Secretary Approach

Wen Cao, Cheng-Gang Bian, Gunnar Hartvigsen
Department of Computer Science
Institute of Mathematical and Physical Sciences
University of Tromsø, N-9037 Tromsø, Norway
cao@cs.uit.no, bian@stud.cs.uit.no, gunnar@cs.uit.no

Abstract

The Virtual Secretary (*ViSe*) is a kind of intelligent agent that could assist its user in major secretarial work. Users give the orders of *what to do*, while agents figure out *how to do* and carry out the tasks. This paper presents the *ViSe* agents in two aspects: (1) to construct individual agent on three intelligence levels: knowledge-base level, learning level and cooperation level; and (2) to model other agents' *activities* by *cooperator-base* ⊎ *task-base* for the purpose of efficient cooperation. We argue that this agent modeling could achieve intelligent cooperation, high performance and easy maintenance in multi-agent systems.

Introduction

For most people, a common problem is to retrieve the wanted data from many different information sources. Imagine that you are visiting a city for the first time and you want to buy a suitable dress for a welcome-party. Since you are a stranger in this city, you may have to spend a lot of time walking the streets in order to become familiar with the local shopping system, and roaming from store to store to get an idea of where to buy the right clothes at the right price. Alternatively, you might ask advice from people who know the city well. Following their advice, you could select several interesting stores and go there to seek the right clothes, but it still wastes a lot of time. The third approach is to hire an agent to finish these jobs. You may just give the agent your personal profile and an order *"I need some clothes for a welcome-party"*, and leave your agent to *roam* the streets and shops to find suitable clothes for you. The agent could give you such an answer: *"You could buy an evening dress at* Linda Fashion *on* 15th avenue no.15 *with price about* $150*"*. No doubt, many similar scenarios happen all the time in our daily life, i.e. shopping, kids' teaching, job applying, or even vacation planning, people are so tired of going through these boring, iterative and time-consuming tasks. Ideas of software agents' delegating in human's tedious tasks have motivated us to build one type of Virtual Secretary (*ViSe*) agent, that can assist its user in major secretarial tasks, i.e., information filter and retrieval, personal calendar scheduling, daily life administration etc (Hartvigsen *et al.* 1996). In this user-agent interactive environment, the user just gives a specification of a task — a high level goal that the agent may achieve. Based on the goal, the agent will figure out *how to do*, and carry out the task. As such, the problem is: how to build the *ViSe* agents?

By imitating human activities, we will build the *ViSe* agents to be intelligent enough to take over most of the secretarial work from their employers. Then the question becomes: how much intelligence should our agents have? Let us generalize the shopping story to illustrate this point. When a person hires an agent from a *ViSe* agent bureau, the agent has basic knowledge of how to deal with shopping problems, and we name it as the first level of intelligence agent. Agents on this level could fulfill any amount of tasks with the help of their knowledge-based systems, but their knowledge is static. If agents only stay on this level, they will soon be eliminated. As people fetch new knowledge all the time, the key issue for agents on the second level of intelligence is to keep their knowledge fresh and updated by learning. Agents could learn new knowledge from their users or other agents. Smart people always know how to make use of other people's intelligence and experience, so do our agents. Agents on the third level of intelligence should know how to cooperate with each other to solve problems efficiently. An important characteristic for agents with such cooperative behavior is to find out *who can help me* — to predict the other agents' *activities*, and contact them for assistance. In this respect, we propose a *twin-base* system (*cooperator-base* ⊎ *task-base*) to monitor the other agents' *activities*. Every agent maintains a *twin-base* for the purpose of cooperation: (1) while receiving a task which is beyond its capabilities, the agent could directly find out *who can help me* from the *task-base*;

(2) while detecting changes of other agents' *activities*, the agent could reason out *who can do* the tasks and update the *task-base*. In other words, agents carry out users' tasks on-line, and perform reasoning and maintenance work off-line. Our work is based on the assumption that agents are truly connected to a reliable network and they are willing to cooperate with each other.

This paper describes the *ViSe* agents in two aspects: (1) to construct individual agent on three intelligence levels, and (2) to model other agents' *activities* by a *twin-base* system for the purpose of efficient cooperation. In the end, we give an overview of the *ViSe* multi-agent system architecture.

ViSe Agents at Three Intelligence Levels

We have mentioned that *ViSe* agents were structured at three intelligence levels. An agent on the first level could perform user's pre-defined tasks with the help of its knowledge-based system. Taking the user's task as an input goal, the agent searches its knowledge-base to recognize the goal and carry out the related actions. The knowledge-based system provides not only general information, but also expert knowledge of the problem solving, so an agent on the first level could play the role of an expert in some specific domains. In the shopping example, the agent's knowledge-base consists of:
(1) general information:

shop(Name, ID, Type, Location, Clothing_Type).
clothing(Shop_ID, Clothing_Type, Clothing_Name, Size, Price, Color, Material, Brand).

(2) search rules:

search(Clothing_Name, Price, Shop_Name, Shop_Location):-
clothing(Shop_ID, _, Clothing_Name, _, Price, _, _, _),
shop(Shop_Name, Shop_ID, _, Shop_Location, _).

(3) expert knowledge:

expert_shopping(Society_Role, Activity, Clothing_Name).
e.g.,
expert_shopping(businessman, work, suit).
expert_shopping(factory-worker, work, uniform-for-factory).
expert_shopping(middle-class-youth, party, suit).

and (4) high level goals:

agent_shopping(Society_Role, Activity, Clothing_Name, Price, Shop_Name, Shop_Location):-
expert_shopping(Society_Role, Activity, Clothing_Name),
search(Clothing_Name, Price, Shop_Name, Shop_Location).

Given this knowledge-based system, the agent could help its user select the right clothing information by following search:

agent_shopping(middle-class-youth, party, Clothing_Name, Price, Shop_Name, Shop_Location)

Although agents on the first level could supply some expert services to their users, their knowledge is fixed once and for all. They cannot be customized to individual user habits and preferences (Maes 1994), and they cannot be adaptive to information change and extension which exist in our daily life. To overcome these problems, we raise our agents' intelligence into the second level by giving them learning abilities, so that they can enlarge their knowledge and become more adaptive to users' interests. We believe that agents could learn new knowledge from different sources. Firstly, agents could integrate existing information sources, i.e. database, expert system etc., through some well-defined interfaces into their knowledge-base. Secondly, agents could learn new knowledge from their users and other agents in the group. These two learning techniques are called *storage learning* since they install new facts and rules into agents' knowledge-base. Thirdly, agents could learn their users' habits and preferences adaptively. As we know, people will keep their habits and preferences for a long period, so agents may regard this information as stable data and collect it in the database for later use. We could use *memory-based learning* technology to help agents learn their users' profile. The *memory-based learning* technology was first introduced by (Stanfill & Waltz 1986), and later developed by Maes' agent group (Maes & Kozierok 1993) (Maes 1994).

Until now, either on the first or on the second intelligence level, agents could solve problems alone with a goal-oriented reasoning model. Agents compare the users' tasks against goals in their knowledge-base and invoke the related actions to perform the tasks. The process will succeed on conditions that both the inference rules and information needed are stored in the knowledge-base. If an agent can not recognize a task from its knowledge-base, it has to ask for help. In other words, when an agent faces a task that is beyond its problem-solving capability, it should cooperate with other agents to solve the problem. We believe that agents with such cooperative behavior come into the third intelligence level and this is the most complicated work. For better cooperation, it is important to know the other agents' *activities*, but it is difficult to catch this information in a distributed environment, where each agent has complete local knowledge of itself, incomplete and uncertain knowledge of the oth-

ers, and no one has total knowledge and global control over the system. As such, how does an agent know the others' *activities*? The most straightforward way is through communication. It is time-consuming and may decrease the system performance tremendously. Alternatively, agents could predict the others' *activities* through *agent modeling*, which yields two questions: (1) how to model an agent's knowledge for the purpose of intelligent cooperation? and (2) how to update and maintain this information consistently for later prediction?

Agent Modeling in a ViSe Multi-agent System

As yet, there is no consensus on how to model a general agent. (Jennings 1994) classify an agent's knowledge as *state knowledge*, *intentional knowledge*, *evaluative knowledge* and *domain knowledge*, while (Dunin-Keplicz & Treur 1994) define an agent's information as *material world*, *mental world of the agent itself*, *mental world of other agents*, *interaction with the material world*, and *communication with other agents*. (Genesereth & Ketchpel 1994) declare that an agent supports the other agents with information about their *capabilities* and *needs*. In general, what kind of information that is appropriate to represent an agent depends on the agent's application domain. Since there is no agreement about the terminology of an agent, which means, there are many diverse agent types co-existed (Riecken 1994) (Wooldridge & Jennings 1994), we believe that many theories on agent modeling exist. In this section, we first define the type of *ViSe* agent, and then explore an agent modeling mechanism: *cooperation-base*. By considering the efficiency of cooperation and easiness of information maintenance, we further improve the *cooperation-base* as *cooperator-base* $\uplus$ *task-base*.

Step I: Cooperation-Base

In a *ViSe* multi-agent environment, there are tens (or hundreds) of agents organized in a group. Each agent is related to one user and takes the role of a virtual secretary. In a user-agent interaction, a user gives the specification of the task — *what to do*, and leaves the details of *how to do* to the agent. Agents in the group are connected by a reliable network and are willing to help each other, meaning that they are *benevolent agents* (Rosenschein & Genesereth 1985) (Durfee, Lesser, & Corkill 1987).

To define the type of *ViSe* agent and to model such an agent for cooperation yield three questions:

1. What are the tasks of the *ViSe* agent?
2. When do agents need cooperation to perform a task?
3. What kind of information do agents need for the purpose of cooperation?

Agents carry out most of the secretarial work that is tedious and boring for their users. They could cover general diary functions, i.e. personal information searching, calendar scheduling, traveling arrangements etc., furthermore, they could play the roles of specialists in different areas, such as shopping guides, kids' teaching experts and so on. When an agent receives a goal that is beyond its problem-solving capabilities, it has to cooperate with the other agents in order to finish the task. Generally speaking, an agent needs cooperation in the sense that:

1. The agent has no idea of how to handle a task. It could ask for assistance from those who are experts in the field, or who have experienced the same kind of problem before. Cooperation in this area considers that agents share the problem-solving capabilities and results, in this sense, it falls into *result sharing*.
2. The agent lacks some information to complete a task. It should ask those who own the information for help. Cooperation in this area could be called *information sharing*.
3. The agent only has the ability to perform one fraction of a large task so that many agents have to work together to fulfill the job. Cooperation in this environment is called *task sharing* and agents cooperate with each other to play a joint action.
4. We have assumed that the agents in the group might not only ask for help, but also be willing to help the others. Helping the others could be applied in the above three cases, as well.

By analyzing the above cooperation scenarios, we believe that cooperation is necessary when one agent is facing a task that is beyond its capabilities and knowledge, or the agent achieves some capabilities that the other agents are waiting for. In both situations, it is important for an agent to know the others' capabilities and needs so that it could talk to the right one instead of a blind inquiry broadcasting. In other words, to have cooperative behavior, agents must know the other agents' capabilities and interests in advance. To satisfy this specification, we could let each agent maintain a *cooperation-base* that includes the *activities* of the other agents in the group. Each tuple in the *cooperation-base* contains one agent's meta-level information that consists of the following domains:

state It contains the agent's *name*, *address*, *status* in the group. Status information is important in the

sense that it could help the agent decide the manner in which it cooperates with the others, e.g., agents with a flat relationship (with the same level status) may be willing to help each other and share information, while agents in a master-slave relationship (with a hierarchical organization status) may need some access control for information sharing for the sense of security. The status information could also help the agent find out *who can help me.* Like human beings, agents would like to have contact with those who have similar status in the group.

capability It deals with an agent's problem-solving abilities. The agent's capabilities are actually those goals in its knowledge-base with meta-language descriptions.

need It describes what kind of help an agent currently is interested in.

statistics It indicates the agent's problem-solving power. The *statistics* information could also help the agent figure out *who can help me.* The Agent prefer to ask for help from those that have more problem-solving power.

To summarize, each agent maintains its *cooperation-base* that includes meta-level descriptions of all agents in the group. The meta-level description could be defined by a *descriptor* in the following form:

agent-descriptor(
state [name, address, status, availability],
capability [description, time, location],
need [description, time, location],
statistics [number-of-rules, number-of-information]
)

Step II: Cooperator-Base ⊎ Task-Base

An important step in the cooperation process is to find out *who can help me* as quickly as possible. To satisfy this demand, we need a database to store the most updated *activities* of the other agents in the group. As explained in Step I, each agent in the group could be represented as a *descriptor* and every agent maintains such *descriptors* in its *cooperation-base* for the purpose of cooperation. Now let us figure out how the *cooperation-base* works for a cooperation process. When an agent receives a goal that is beyond its capabilities, it will check its *cooperation-base* to find out *who can help me.* What the agent does is to compare the goal received with the *capability* domain of *descriptor* in the *cooperation-base.* If more than one agents' capabilities match the goal, then the agent will select the most suitable cooperator based on the *state* and *statistics* information, and send messages to the selected one for assistance. Let us go further to analyze the procedure:

- The main purpose of exploring a *cooperation-base* is to help agents reason out *who can help me.* Facing a specific task, the agent searches domain *capability* to find out the matched *descriptor(s)*, and uses domains *state* and *statistics* to select the most suitable cooperator. In fact, for a particular task, the *cooperation-base* will give a relatively *stable* result of *who can help me* over a period of time. Thus, if we let agents do these reasoning jobs before they accept tasks from users, then the system on-line performance will be tremendously increased and the repeated reasoning process could be avoided. The trick is to improve the *cooperation-base* by directly mapping each task with a set of agents that could carry out the task.

- For an agent, to record the other agents' *activities* into *cooperation-base* is one step, while to maintain this information for later use is another step that is quite costly. In a distributed environment, it is most difficult to maintain the *activities* of others since this information is being changed. With such changes, we hope that we could minimize an agent's gathering information of the others so that less maintenance work is required, but the cooperation requires that each agent could keep as much of the other agents' information as possible. Considering the balance between maintenance and cooperation, our agent modeling should include minimal information for covering the requirement of cooperation.

- Our agents serve their users for most of the tedious and boring work. It is not easy to distinguish what kind of capabilities the agents need, and which are not needed. We hope our agents could *capture* as many skills as possible so that they could serve their users in the best way they can. With this specification, it is meaningless to set the *need* domain in the *descriptor*, especially when the scale of cooperation group is not very large, so we allow every agent broadcast its newly learned skills to the others no matter when they need them.

In response to the above analysis, we have built a *twin-base* system: *cooperator-base* ⊎ *task-base.* The *cooperator-base* is derived from the *cooperation-base* with two domains: *state* and *statistics*, while the *task-base* is improved from the *cooperation-base* by enhancing the *capability* domain. Employing such a *twin-base* modeling will greatly improve the system performance. As discussed before, single *cooperation-base* modeling

costs an agent's on-line time for reasoning out *who can help me*, the *cooperator-base* ⊎ *task-base* architecture will leave this time-consuming work to the agent in its free time. In other words, agents perform reasoning and maintenance work off-line, and carry out user-defined tasks on-line.

Unlike the *cooperation-base* where each *descriptor* stands for one agent in the group, in the *task-base*, each tuple is related to one task and recorded in the form of {*task-description, agent-set(subtask)*}. The *task-description* states goals that agents in the *agent-set* could perform jointly or separately. If agents perform the task together, then the domain *subtask* is needed to specify individual operation. If more than one agent performs the same task separately, we need the following rules to set up their priorities in the *agent-set*:

Rule 1 Self-solving has the highest priority. The host agent always holds the first place in the *agent-set*. There are two reasons for us to employ this rule. Firstly, to ask assistance from the others will increase the communication overheads, and thus reduce the system performance. Secondly, for security reasons, the agent should trust itself more than anyone else.

Rule 2 Non-host agents are sorted by their *status* and *statistics* information described in Step I. It is better to choose the agents that belong to similar *status* levels and have good *statistics* records.

The *task-base* is quite simple and easy for maintenance. Could it fit the purpose of our *ViSe* agent cooperation? To answer this question, we should review the situations where cooperation is necessary which is discussed in Step I:

- When an agent faces a task that it does not know *how to do*, the agent could search its *task-base* and retrieve the *agent-set* whose *task-description* matches the goal. If there are several agents that could do the tasks, choose the one with the highest priority.
- When an agent lacks information for solving one task, the agent could search the *agent-set* in its *task-base* and retrieve the second one who could perform the task (the first one is the agent itself).
- When an agent receives a large task, the agent could search its *task-base* to find a set of agents which take different sub-tasks, since we have such tuples in the *task-base*: { *task-description, [agent-1 (sub-task.1), agent-2 (sub-task.2), ...]*}. Ideas from *contract net protocol* (Smith & Davis 1989) (Davis & Smith 1989) could be used to distribute workload among agents and set up such tuples, and the adaptive scheduling strategies (Sen & Durfee 1994) could be employed to allow agents take the joint action.

To summarize, we argue that our *cooperator-base* ⊎ *task-base* modeling covers necessary requirements of the *ViSe* agent cooperation with advantages of high performance and easy maintenance.

Task-Base Maintenance

The *task-base* provides the direct mapping between tasks and the related agent sets that could perform such tasks. As we know, the capabilities of agents in the group may be changed. Agents could adapt new abilities from users and other agents, some agents may leave the group or do not carry out some tasks any more. So even if an agent has great information snapshot over a group at one time, it does not mean that the agent knows the others' information at all times. In order to correctly predict the other agents' capabilities, it is necessary to set up one *capability revision* process to update the *task-base* consistently. The revision process could proceed in the following way:

- Detecting a unavailable agent in the group (the agent left the group or does not carry out tasks any more), our *ViSe* agent should check the domain *agent-set* of its *task-base* to see which tasks are dependent on this agent. The matched tasks should be deleted if they are solely dependent on the unavailable agent, or the agent should be removed from the *agent-set* under the matched tasks if there are other agents that could also carry out the tasks.
- Detecting some unavailable capabilities of an agent, our *ViSe* agent should check the domain *task-description* of its *task-base* to delete the related tasks if only the agent could perform them, or remove the agent from the *agent-set* if other agents could also perform these tasks.
- Receiving a message of new capability announcement from an agent, our *ViSe* agent should make a new tuple in the *task-base* if the capability is new to the group, or put the agent into the *agent-set* if the related capability already exists in the *task-base*.

Let us make a simple example to illustrate the above revision procedure. Assume we have five agents in the group with related capabilities:

agent-1: shopping clothes in the Tromsøya
agent-2: shopping clothes in the Tromsdalen
agent-3: shopping clothes in the Kvaløya
agent-4: shopping food in the Tromsøya
agent-5: inviting people for party

Assume there are the following tasks in the group:

task-1: to buy clothes
task-2: to buy food
task-3: to arrange a party including food shopping, clothes shopping and people inviting
task-4: to select the cheapest clothes in the Tromsø[1]

So one agent, e.g., *agent-2*, has the following information in its *task-base*:

{*task-1, ([agent-2], [agent-1], [agent-3])*}
{*task-2, ([agent-4])*}
{*task-3, ([agent-2, agent-4, agent-5], [agent-1, agent-4, agent-5], [agent-3, agent-4, agent-5])*}
{*task-4, ([agent-1, agent-2, agent-3])*}

For example, if *agent-1* leaves the group and broadcasts the message to all the others in the group, *agent-2* will update its *task-base* as following:

{*task-1, ([agent-2], [agent-3])*}
{*task-2, ([agent-4])*}
{*task-3, ([agent-2, agent-4, agent-5], [agent-3, agent-4, agent-5])*}

Obviously, with the *task-base*, it is very convenient to manipulate the changing of agent's capabilities. There is no backtracking, no sophisticated reasoning process, just simple set operation.

System Architecture

Based on the agent modeling and problem definitions declared above, we have constructed a *ViSe* agent that consists of the following modules (Figure 1):

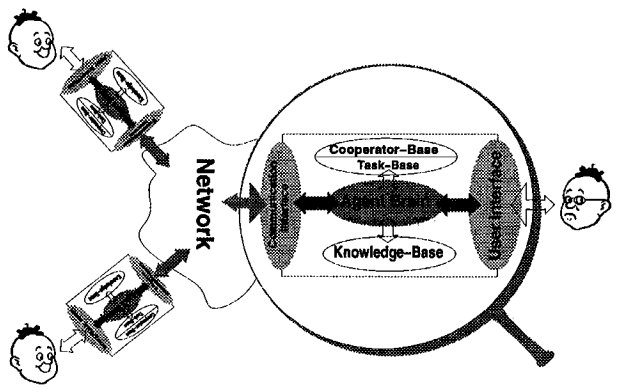

Figure 1: The ViSe Multi-agent System Architecture.

User Interface This module offers an interface between a normal user and a *ViSe* agent. The user gives the specifications of tasks through the interface.

[1]Tromsø consists of Tromsøya, Tromsdalen and Kvaløya.

Communication Interface This module offers an interface among different agents. It is responsible for sending and receiving messages over the network.

Agent Brain This is a central control module. It is responsible for scheduling agent work on learning and performing tasks. An agent could learn *how to do* from its users, or other experienced agents. It could perform tasks specified by its user, or asked by other agents in the group.

Knowledge-Base Each agent is a specialist in some areas, and the *knowledge-base* consists of this domain-dependent information.

Cooperator-Base ⊎ Task-Base This *twin-base* includes all the necessary knowledge for agent cooperation.

Generally, a *ViSe* agent works in two different states: *performing tasks* or *learning new knowledge*. These two action-scenarios are described as following:

Performing tasks A *ViSe* agent will perform tasks ordered by its user, or asked by other agents. In both cases, it should activate its *knowledge-base* to perform the related operations and retrieve the wanted information for the user or other agents.

Learning new knowledge A *ViSe* agent is intelligent enough to learn new knowledge so that it will become more and more clever. The agent could learn new knowledge from its user or the other more experienced agents in the group. Users could teach the agents new information, rules, or their preferences and habits. The agents could also learn new knowledge from the others by importing information and rules.

Concluding Remarks

This paper presents two aspects of the *ViSe* agents: (1) to construct individual agent on three intelligence levels; (2) to model other *ViSe* agents' *activities* by using a *twin-base* system for the purpose of efficient cooperation. Results received so far indicate that the *twin-base* system achieves intelligent cooperation, high performance and easy maintenance in the *ViSe* multi-agent system. Considering the cooperation in our system, there are some other issues that we are going to working on, i.e. establishment and management of a cooperation group, organization structure, communication paradigm and policy, security problems etc.

References

Davis, R., and Smith, R. G. 1989. Negotiation as a Metaphor for Distributed Problem Solving. In

H.Bond, A., and Gasser, L., eds., *Distributed Artificial Intelligence*, 333–356. Morgan Kaufmann.

Dunin-Keplicz, B., and Treur, J. 1994. Compositional Formal Specification of Multi-Agent Systems. In Wooldridge, M. J., and Jennings, N. R., eds., *Intelligent Agents*, 102–118. Springer-Verlag.

Durfee, E. H.; Lesser, V. R.; and Corkill, D. 1987. Cooperation through Communication in a Distributed Problem Solving Network. In Huhns, P., ed., *Distributed Artificial Intelligence*, 29–58. Pitman.

Genesereth, M. R., and Ketchpel, S. P. 1994. Software Agents. *Communication of the ACM* 37(7):48–53.

Hartvigsen, G.; Johansen, S.; Helme, A.; Widding, R.; Bellika, G.; and Cao, W. 1996. The Virtual Secretary Architecture for Secure Software Agents. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM 96*, 843–851. The Practical Application Company Ltd.

Jennings, N. 1994. *Cooperation in Industrial Multi-agent Systems.* World Scientific.

Maes, P., and Kozierok, R. 1993. Learning Interface Agents. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.* Cambridge, MA: MIT Press.

Maes, P. 1994. Agents That Reduce Work and Information Overload. *Communication of the ACM* 37(7):31–41.

Riecken, D. 1994. Intelligent Agents. *Communication of the ACM* 37(7):18–21.

Rosenschein, J. S., and Genesereth, M. R. 1985. Deals among Rational Agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 91–99.

Sen, S., and Durfee, E. H. 1994. On the Design of an Adaptive Meeting Scheduler. In *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, 40–46.

Smith, R. G., and Davis, R. 1989. Frameworks for Cooperation in Distributed Problem Solving. In H.Bond, A., and Gasser, L., eds., *Distributed Artificial Intelligence*, 61–70. Morgan Kaufmann.

Stanfill, C., and Waltz, D. 1986. Toward Memory-based Reasoning. *Communication of the ACM* 29(12):1213–1228.

Wooldridge, M. J., and Jennings, N. R. 1994. *Intelligent Agents.* Springer-Verlag.