Software Agent Evolution in Adaptive Agent Oriented Software Architecture

Guoqiang Zhong, Babak Hodjat, Tarek Helmy and Makoto Amamiya
Department of Intelligent Systems
Kyushu University
Kasuga-shi, Fukuoka (816-8580), Japan
{zhong, bobby, helmy, amamiya}@al.is.kyushu-u.ac.jp

Abstract

Adaptive Agent Oriented Software Architecture (AAOSA) is a new dynamic approach to software design based on multi-agent oriented architecture. Since the optimal agent organization is different from one environment to another, we proposed a distributed learning policy that is used in AAOSA for the purpose of agent organizational evolution. Knowing when and how to communicate and coordinate with other agents is an important efficiency and reliability question. In this paper, we propose the use of on-line feedback from users to motivate the learning whenever necessary, and show that distributing the evolution process over individual agent in AAOSA is efficient and reasonable.

Keywords: multi-agent, distributed learning, software agent evolution

Introduction

In this paper, we are interested in learning techniques that allow agents in Adaptive Agent Oriented Software Architecture (AAOSA) to improve their performance by learning from past run-time experience and users' on-line feedback. AAOSA is a multi-agent oriented software methodology that proposes the break-up of complex software into a community of simpler, independent, collaborating, adaptive, message-driven agents (Hodjat, B. 1998). According to the dynamic nature of environment, effective coordination strategies will be very important for AAOSA agents which work together to solve the problem that any individual agent is unable to do it alone. Due to their "open" operating environment where the configuration and capabilities of other agents and computational resources could change dynamically (Lesser, V. R. 1998), learning within a multi-agent based system is, in general, very hard (Parkes, D.C. 1997). Learning from reinforcement is a promising approach for creating intelligent agents (Tan, M. 1993) because it allows agent to learn which of its actions is desirable based on the exploration experience of its environment. It may take a long sequence of actions before finally arrive at a state with high reward (Kaelbing, L. P. 1996).

Note that intelligent behavior is emergent from the cooperation of agents because the agents community can accomplish more and better than any one of the agents. We also show that distributing the learning over a hyper-structure of more simple sub-domains is less complex and more feasible than centralized learning. Learning can involve a lot of things, such as, acquiring strategies for communication or coordination of agents, detecting relationship of agents, constructing new models or agents for new task, and so on. Currently, we propose that learning in an AAOSA agent system can be made on two separate layers concurrently:

- Learning of agent organization and
- Learning of local agent capabilities.

Each layer makes decisions respectively and involves different processes and goals. For example, the organizational layer makes decisions that are of long-term reward, while the local layer try to choose actions that maximize immediate reward.

In the following sections, we will first briefly re-introduce our framework for AAOSA methodology, and then describe the proposed learning policies in a domain-independent way. After giving out some examples, we conclude with a look at our future work direction.

Overview of AAOSA

AAOSA is a new approach to software design based on multi-agent oriented architecture. In this approach, agents can be considered as adaptively communicating concurrent modules divided into a white box module responsible for communications and learning, and a black box module responsible for the independent specialized processes. To reap the benefits of distribution and enhance its learning abilities, each

agent is kept simple in its responsibilities and limited in the decisions it needs to make.

AAOSA agents are not centralized, the overall task is identified and suitably decomposed by the whole AAOSA agents network. The break up of problem into sub-domains is the responsibility of the designer who should also define the interpretation policies for each agent. By doing so, AAOSA agents in the hyper-structure can be responsible for a certain input, and provide the necessary coordination between them to achieve desired output. Figure 1 shows the typical structure of an AAOSA agent.

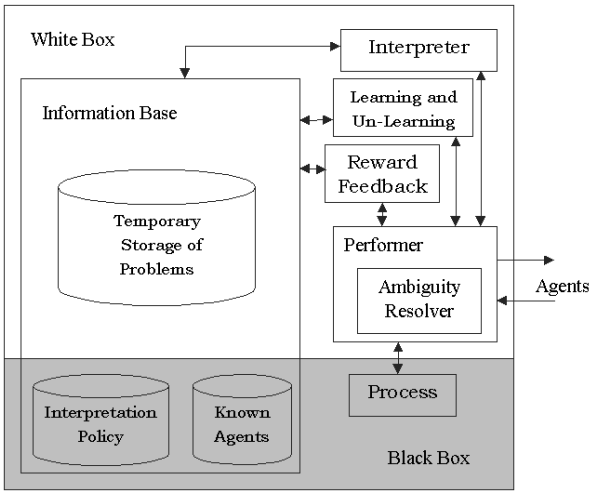

Figure 1: AAOSA Agent Structure

In an AAOSA agent, as Figure 1 shown, white box contains standard data structures and models for communications, interpretations, and learning provided by AAOSA, while black box contains agent-specified communications, interpretations, and processes defined by the designer. Note that agent's knowledge about its environment is stored in an information base consisted of "temporary storage of problem," "interpretation policy" and "known agents". "Temporary storage of problems" is used to keep a record of agent problem-solving process. "Interpretation policy" is used whenever agent tries to interpret input from other agents or users. The interpretation criteria may be the message content but is not limited to it. Process history, probabilities and outside information (e.g., interaction with other agents) are examples that can be used by agent as interpretation policy. If an agent is unable to interpret a particular input by itself, it may consult other agents. These agents are called down-chain agents relative to the requesting agents, which are called up-chain agents. "Known agents" is used to store the addresses of down-chain and up-chain agents.

The performer module actuates other modules in the agent based on the message performatives received from other agents. Each message is comprised of a message content, a performative that specifies what should be done with that content. The designer can add agent-specific performatives in sub-domains to facilitate special communications between agents. AAOSA framework provides a set of predefined general performatives by which the coordination of agents can be managed. (See Table 1)

Register	Agents make each other aware of their existence.
Advertise	An agent declares it can handle certain input.
Un-Advertise	An agent requests not to receive input from another agent.
This-Is-Yours	An agent announces another agent as responsible for handling certain input.
Is-This-Yours?	An agent that can not interpret a particular input requests interpretation from down-chain agents.
Restore	Agent requests another agent to backtrack to a state before processing took place on certain input.
Not-Mine	Down-chain agent has failed to interpret input sent down with an *Is-This-Yours?* Performative.
Maybe-Mine	Down-chain agent has encountered an ambiguity in interpreting input sent down with an *Is-This-Yours?* Performative.
It-Is-Mine	Down-chain agent has been successful in interpreting input sent down with an *Is-This-Yours?* Performative.
Commit	Agent requests immediate response, be it partial, to input sent down with an *Is-This-Yours?* Performative.
Learn	A new interpretation policy is suggested to an agent that will result in the sender agent being interpreted as responsible for certain input.
Un-Learn	An interpretation policy that results in the sender agent being interpreted as responsible for certain input is revoked.
Dissatisfied	Alternative process or interpretation is requested for input that has already been processed.
Forget-Problem	A previous request is canceled and Remove any temporary storage of interpretation results.

Table 1: AAOSA Predefined Performatives

Ambiguities occur when an agent has not been able to interpret the message content belonging to it (*This-Is-Yours*) based on its interpretation policies, and

- Either more than one agent consulted with claim it,
- Or, none of the agents consulted with claim it.

The resolution of ambiguity is particularly important in AAOSA because it provides a way by which agents can change their behavior and react to unexpected input. In the next section, we demonstrate that trying to resolve ambiguity (or conflict) is one of the main tasks of learning.

Within an AAOSA agent system, we know that generating a solution to input problem is not a straightforward combination of agents' local sub-problem solving processes. Rather, this process can be an incremental process involving coordination among agents. Coordination among agents can be explicit or implicit. As an explicit way, the agents interact and exchange information or perform actions to benefit other agents, while implicitly cooperative agents perform actions that are a part of their own goal-seeking process and will also affect the other agents in

beneficial ways (Lesser, V. R. 1998). Appropriate communication is necessary to efficiently manage the interdependencies among agent activities. Otherwise, inappropriate communication can prevent the system from generating optimal solution (or even no solution at all), and waste considerable resources on useless activities. In other words, an agent must take reasonable decisions about when to choose a best-response strategy given its current knowledge, and when to deviate and gather more information about the preferences of the other agents. We will discuss how this task can be done through learning in the following section.

General Learning Policies

AAOSA agents use learning to improve their overall performance. By learning, for example, agents can find ways to solve unforeseen problems, exploit faster and more robust ways for those known problems, as well as adapt to different users' preferences. To get greater flexibility, learning in AAOSA is incorporated on the architecture and distributed over the multi-agent structure, rather than introduced one particular agent responsible for learning. In fact, learning can be viewed as an integral part of AAOSA, and each agent is responsible for choosing the proper strategy for itself in different problem instances. Each agent has a role in its own agent community. The main problem here is that an agent has to deal with its limited view of the interactions between its own activities and those of other agents (Nagendra Prasad, M. V. 1997). Reinforcement learning is proposed to resolve this problem, because it enables agent to learn behavior through trial-and-error interaction with a dynamic environment (e.g., an AAOSA agent society).

As a reinforcement-learning agent, agent's diagnosis of history plays a central role in its selection of reward-maximizing actions. In AAOSA, each agent keeps a record of its problem-solving process in "temporary storage of problems". While learning, performance criteria are used, which combine both local and non-local perspective to analyze agent previous experience. Furthermore, these performance criteria will be changed due to emerging conditions so that an agent's view of the global situation will be more reasonable and adaptive. Some performance criteria are domain-independent while some of them are domain-dependent. From the system's point of view, the following criteria are usually considered, such as computational resource, efficiency, communication cost, ambiguity, user satisfaction, and so on. Designer can also add other domain-dependent criteria. The past experience will be analyzed based on performance criteria. The result of this analysis will be used to effectively detect and diagnose the cause of agent inappropriate behavior. Finally, we can formulate learning policies for AAOSA agent as following:

> *It will try to choose the best action given a past history, actions of the other agents in the system, and performance criteria.*

From each agent point of view, all decisions made from learning can be applied either inside the agent or over the system's structure. Therefore, learning can be considered on two layers separately: learning inside agent and learning over the agent structure.

Learning Inside Agent

Since no agent will be added or removed, learning in this layer is used to improve the agent's own specialized performance by choosing actions to maximize its own reinforcement. The performance criteria concerned on this layer are mostly local and short-term. Learning in this layer can modify the interpretation rules or relevant agents' addresses. By doing so, we can enhance the agent's local capabilities, improve coordination strategies, resolve ambiguities and so on. When a new interpretation rule is learned by an agent, it should sent "Un-Learn" message to other agents. By doing so, the potential ambiguities can be avoided because there is only one agent will claim "It-Is-Mine" based on the new learned interpretation policy.

Learning Over the Agent Structure

There is no centralized control enforced over the AAOSA agent network. Agents will introduce themselves and their abilities to one another at the beginning or during execution. Therefore, one of the major features of AAOSA is that agents can be added to or removed from the application at runtime. This feature makes AAOSA agent system evolutionary. That means, social structures and mechanisms can be dynamically reasoned and changed. The performance criteria and actions concerned at this level are mostly non-local and long-term. Learning in this layer can make the decision to split too complicated agent into simpler ones, or join redundant agents to form more efficient one. We call them merge and split function respectively, and this two fundamental functions can be used to achieve a balance between the degree of generalization (merge) and specialization (split) of the overall agent community. Whenever merge or split function is done, it is necessary to send "Forget-Problem" messages to relevant agents to remove the corresponding problem records stored in "temporary storage of problems". Otherwise, the reinforcement learning will still use the old records of problem, and choose an incorrect action. Both ambiguity and efficiency play important roles in making this balance. It must be noted that resolving ambiguity will improve system's efficiency by decreased the communication cost. In fact, resolving ambiguity is a useful and

primary method to decrease the communication burden of coordinating agents' activities in an AAOSA system.

Since learning in each level involves different goals and processes, it is easy for agent to decide on which level the learning should be implemented. In the next two sections, we will show that the AAOSA agent system can reach a high-speed goal with the combination of local learning and organizational learning mechanisms.

Example 1: Agent Capabilities Learning

With the use of AAOSA, the design of adaptive user interfaces will be decentralized and become an individual affair, mostly driven by the personalized information. The long-term customization of user interfaces will be gained from numerous shared experiences between agents and user. An agent can acquire its competence from user's indirect or direct feedback. For example, indirect feedback is provided when user ignores the agent's suggested action, while direct feedback can be get from user-supplied examples. Finally, the agent also can ask advice from other agents that may have more experience with the same task (Bradshaw, J. M. 1997). Using a simple example, we show that this task can be done by incremental learning of agent capabilities in an AAOSA agent system.

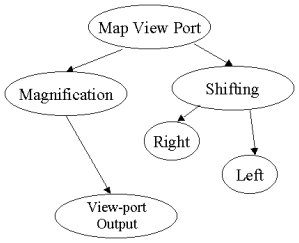

Figure 2: AAOSA agent community for Map Viewing

Figure 2 shows a small part of agent community based on AAOSA, which is designed for map viewing. A sample of a contradiction resolution process is given out as following:

1) User inputs "move it closer"

a) "Move it closer" is sent to agent *Map View Port*.

b) Agent *Map View Port* in turn sends message down to *Magnification* community and *Shifting* community

c) Both agent *Magnification* and agent *Shifting* claim "It-Is-Mine" to agent *Map View Port*.

d) Agent *Map View Port* announces a contradiction by sending out a "Maybe-Mine" message.

2) System asks user:

"Do you mean magnification or shifting?"

3) User responds with "Magnification".

a) Agent *Map View Port* learns that "move it closer" belongs to the *Magnification* community

b) Agent *Map View Port* sends "move it closer" to the agent *Magnification* with "This-Is-Yours"

c) Agent *Magnification* interprets "move it closer", sends a "This-Is-Mine", and the map is magnified.

In this example, ambiguity occurs within step 1, because both agent *Magnification* and agent *Shifting* claim "It-Is-Mine" to the input "move it closer". Then, the system asks user to give out his intention in step 2, and gets the direct feedback from user in step 3. That is, "move it closer" means "magnification". Learning is done in agent *Magnification* by two steps:

- Add new interpretation policy specified to input as "move it closer".
- Sent Un-Learn message to agent *Shifting*, so that agent *Shifting* no longer claims "It-Is-Mine" to the input "move it closer".

Example 2: Organizational Learning

AAOSA can be used as a new approach of relational database retrieving. One of the significant features is that query optimization can be done by AAOSA agent hyper-structure optimization. As shown in Figure 3, agent *Flight* and agent *Flight-Fare* represent two tables respectively. Each of them has three down-chain agents, which represent three attributes of the table respectively. Agent *Flight-ID* is shared by agent *Flight* and agent *Flight-Fare*, because *Flight-ID* is the primary key between table "Flight" and "Flight-Fare".

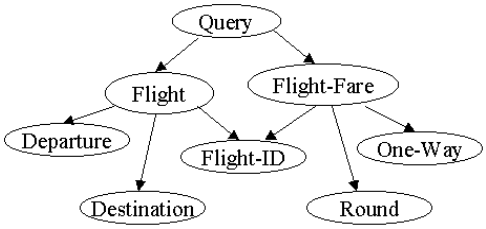

Figure 3: AAOSA agent community for Query DB

This initial structure is optimal for queries involved only one table, because messages can be sent and interpreted among agents without contradiction. Unfortunately, it is not optimal for queries involved both tables, because both agent *Flight* and agent *Flight-Fare* claim "It-Is-Mine" to the agent *Query* when queries come in. Therefore, this ambiguity information will be stored in the "temporary storage of problems" of agent *Query*, and used as the information source for reinforcement learning. Therefore, agent *Query* can indicate that high communication cost occurs between its two down-chain agents. Since this kind of contraction can not be resolved by local agent learning, agent *Query* decides to merge its relevant down-chain agent *Flight* and agent *Flight-Fare*, as figure 4 shows.

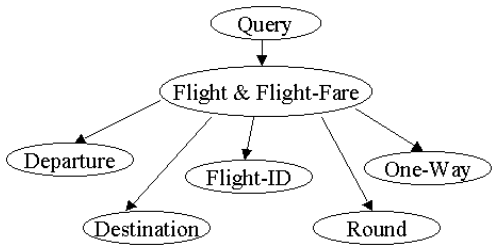

Figure 4: AAOSA agent community after merging

By merging two agents into a new agent "*Flight & Flight-Fare*," we can extend its capabilities. The new merged agent will take not only the responsibility inherent from its predecessors but the responsibility for the joint queries. So far, this agent community works better for queries involved a single table or both.

However, this is not the end of evolutionary process, but a new start point. Suppose that we have a lot of queries on table *Flight* or both tables. Then, agent "*Flight & Flight-Fare*" has a heavy task load. Having a heavy task load is a problem of inefficiency. Unlike the ambiguity, the inefficiency is a relative term that is depended on agent's performance criteria, or user's reward. In this example, we suppose that either the system has a high requirement on efficiency or the user gives out the reward that indicates this inefficient problem. Then, each agent in the system will try to analyze its stored experience information to check the execution time of input queries. It is easy to know that agent "*Flight & Flight-Fare*" becomes the bottleneck of the whole agent system. Once the problem agent has been found, it is naturally to make the decision to split itself. The result is shown in Figure 5.

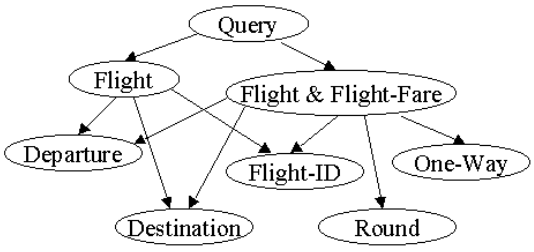

Figure 5: AAOSA agent community after splitting

Now, agent *Flight* takes a part of responsibilities broken down from its predecessor, so that agent *Flight* is responsible for those queries on table *Flight*. Thus, agent "*Flight & Flight-Fare*" is responsible for those queries on table *Flight-Fare* or both tables. We can see that query tasks are actually distributed after splitting, and system becomes more efficient.

Conclusion & Future Work

This paper discussed the mechanism of software agent evolution in AAOSA by use of reinforcement learning. In this approach, AAOSA agent can be viewed as a dynamic entity which can adapt and change its strategies by itself as the environment evolves. By distributing learning task to individual AAOSA agent, the system becomes more flexible and adaptive. We also mentioned that the separation of the organizational learning from local agent learning can make the learning mechanism simpler but more efficient. Two examples of using the proposed learning method for AAOSA agent are also introduced. The result demonstrates that the proposed learning techniques can help an AAOSA agent system to reach a desirable goal with high flexibility and efficiency. However, there are a lot of works remained for further study. We plan to implement and evaluate the proposed learning method in a practical application of AAOSA. We also want to add confidence factors with interpretation policies in an AAOSA agent, and use reinforcement learning to modify those confidence factors on-line. All of these are direction of our future work.

Reference

Bradshaw, J. M. 1997. An introduction to software agents. *Software Agents*. AAAI/MIT Press. (pp. 3-46)

Decker, K. S. and Lesser, V. R. 1995. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multiagent Systems*

Hodjat, B., Savoie C. J. and Amamiya, M. 1998. Adaptive agent oriented software architecture. *Pacific Rim International Conference on Artificial Intelligence*. (pp. 33-48)

Kaelbing, L. P., Littman, M. L. and Moore, A. W. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligent Research*. No.4 (pp. 2347-285)

Lesser, V. R. 1998. Reflections on nature of multi-agent coordination and its implications for an agent architecture. *Autonomous agents and multi-agent system*. Kluwer Academic Publishers, 1. (pp.89-111)

Parkes, D. C. and Ungar, L. H. 1997. Learning and Adaption in Multiagent systems. *AAAI-97 Workshop on Multiagent Learning*.

Nagendra Prasad, M V and Lesser, V. R. 1997. Learning problem solving control in cooperative multi-agent systems. *AAAI-97 Workshop on Multiagent Learning.*

Tan, M. 1993. Multi-agent reinforcement learning: independent vs. cooperative agents. In *Proceedings of 10th International Conference on Machine Learning*. (pp.330-337)