

A Collective View and Methodologies for Software Agents' Interaction

Behrouz Homayoun Far

Department of Electrical and Computer Engineering, University of Calgary
2500 University Drive N.W., Calgary, Alberta T2N 1N4, Canada
email: far@ucalgary.ca

Abstract

Software agents interactions are of special importance when a group of agents interact with each other to solve a problem that is beyond the capability and knowledge of each individual. Efficiency, performance and overall quality of the multi-agent applications depend mainly on how the agents interact with each other. We present an agent model by which we can distinguish different agent's interaction scenarios. The model has five attributes: goal, control, interface, identity and knowledge base. Using the model, we analyze and describe possible scenarios. Then for each scenario appropriate reasoning and decision-making techniques will be devised. The model will be used readily in the design and implementation of multiagent systems.

Keywords: Multiagent system, Interaction, Decision making process.

1. INTRODUCTION

Interaction is one of the main properties of multi-agent systems in which software agents interact with each other to solve problems; to share expertise; to work in parallel or sequence on common problems; to be fault tolerant through redundancy; to represent multiple viewpoints and the knowledge of multiple experts; and to be reusable [3]. Interactions include the ability to directly communicate or collect data on the other agents. The efficiency and performance of each individual agent and the overall quality of the MAS application depends mainly on how the agents operate and interact effectively with each other.

Interacting software agents can have complete knowledge about the other agents' goals, strategies (i.e., actions to select from) and utilities (i.e., the pay-offs of actions) only if the goals, strategies and utilities are accessible through direct or indirect communication. Interactions based on the complete knowledge assumption are usually classified as cooperation and coordination problems [4]. Many techniques and

methods to handle cooperation and coordination have already been proposed [1, 3].

In this paper, we focus on software agent interaction in competitive and uncertain environments in which the knowledge completeness is not a valid assumption and software agents may compete to achieve their goals. In this case the agent must predict the other agents goals, strategies and utilities and this introduces uncertainty to its decision making model. There are many sources of uncertainty including:

- Uncertainty inherent in the problem domain, usually modeled by subjective and/or objective probabilities.
- Uncertainty in the outcome of decisions, usually modeled as utility or risk.

In this paper we suggest an agent model by which we can clearly distinguish different agent's interaction scenarios and analyze and describe possible scenarios. Later for each scenario appropriate reasoning and decision-making techniques will be devised and a library of reasoning and decision making modules will be built. The model and the library will be used readily in the design and implementation of multiagent systems.

Interaction is the main driver for agent decision making. Decision making is different from reasoning in the sense that it may involve multiple threads of control. An agent's decision making process starts with tasks such as gathering information from the other agents, organizing and interpreting them. The model is presented in Section 2 and its properties are derived in Section 3 followed by conclusions in Section 4.

2. THE AGENT MODEL

There are various ways of modeling and viewing software agents. Most of the already proposed models are based on the object-agent analogy (e.g., Jennings' works [6]) which we think can potentially be misinterpreted and can lead to overselling the agent-based approaches. Object is a component level concept and agent is an application level one. By "object oriented software engineering" we usually mean engineering a computer

program based on objects which are its building block and we use object-oriented methodologies during software analysis and design. The meaning of “agent-oriented software engineering” is quite different. Here agent-oriented means development of a computer program that has properties of an agent and it involves concepts like knowledgeable, situatedness, autonomy, proactiveness and interactivity. Similarity of agent-oriented and object-oriented terms should not imply that agent and object are of the same category.

Another popular model for software agents is to model them as software components [5], i.e., a software package with complete encapsulation of its behavior that has only one attribute, called interface (i.e., what they can do). Then the component can only be accessed through its interface (Fig 1a). In this case, the other agents requiring services of a certain software agent may consult directory and naming services (i.e., agent yellow pages) and use the services that the agent offers by adhering to the strict rules specified in the interface document for that agent. Experience shows that this limits the scope and applicability of the software agents, in the sense that autonomy, proactiveness and interactivity may be compromised.

We propose an enhanced software agent model in which besides the interface (I), four more attributes are also specified: a goal list (G) (i.e., what the agent wants to do?); knowledge (K) (i.e., how to do?) that an agent can utilize to perform tasks autonomously; the thread of control (C) (when to do?) and identity (Id) (who to contact?) (Fig. 1b).

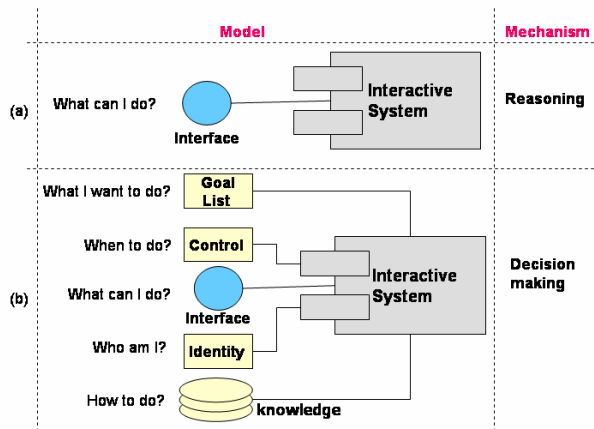


Fig 1. A situated software agent model

When interacting with the other agents, the I-G-K-C-Id attributes can be declared public, private or protected. Public means that the attribute is accessible and readable by all the other agents. Protected means that the attributes are visible only to a certain group of agents and private indicates that the attributes are not visible externally. Combination of the attributes and their states lead to

various interaction scenarios. Obviously, each interaction scenario has certain properties and satisfying those properties requires implementation of certain reasoning and decision making mechanisms.

Using this model the agent autonomy and proactiveness can be preserved if a proper decision making mechanism for the agents is devised and implemented. That is, the agent can decide upon the next task to accomplish using the current list of goals, interfaces, knowledge, identity and thread of control of self and the other agents with whom interacting.

When **Agent_1** interacts with the other agents, it is assumed that it has full knowledge of the state of its own I-G-K-C-Id attributes. However, the status for the other **Agent_2** may be:

- Case 1: fully known; (certainty).
- Case 2: unknown but the probability distribution over the triple states is known; (risk).
- Case 3: unknown and the probability distribution over the triple states is unknown; (uncertainty).

In the Case 1, **Agent_1** knows exactly what the state of the attributes of the **Agent_2** is. For example, **Agent_1** may know the identity (Id) and interface (I) for **Agent_2** or it can inquire directly the goal (G) of **Agent_2**. In this case, decision making for **Agent_1** is straightforward. **Agent_1** can usually select its strategy based on maximum expected utility [2].

For the Case 2, it is assumed that **Agent_1** does not readily know the exact states of the attributes of **Agent_2** but it has a probability distribution over the states of **Agent_2**. For example, **Agent_1** may guess that although it does not know the interface of **Agent_2**, there is a 60% chance of obtaining it from a naming service. As another example, **Agent_1** may guess that the goal of **Agent_2** is public with probability of 20%, protected with the probability of 50% or private with the probability of 30%. We consider both strategic scenario in which the agents should decide simultaneously; and extensive scenario in which the agents should decide sequentially. For strategic scenario, the agent will choose an interaction class, given the signal it gets and its belief about the state of attributes of its opponents based on the profile defined as *(Bayesian) Nash Equilibrium*. For extensive scenario, every agent observes the signals emitted by the other agent to update its belief and consequently the interaction scenario. Here an agent’s best decision is within the strategy profile defined as *Perfect (Bayesian) Nash Equilibrium*.

For the Case 3, it is assumed that the **Agent_1** doesn’t know anything about the state of the attributes of **Agent_2** except for that it is public or private or protected. For instance, **Agent_1** participating in an auction may not know the goal (i.e., exact bid) for the opponents. In this case, the **Agent_1** may assign its belief without using a

probability distribution. According to cognitive psychology, when probability distribution is not known, people evaluate belief based on degree of comfort (i.e., selecting the alternative that needs the least effort) or degree of optimism (i.e., selecting the alternative that we think is the most fit). In [4] we devised a method to assign the weight vector automatically reflecting *degree of optimism* of agents. Once the pseudo-probability values are known, we can view decision making process in uncertainty case the same as in risk case.

Based on this view, a subset of useful and popular agent interaction problems are identified [4]. In Section 3, we will interpret the problems in terms of the interactions scenarios and identify whether collecting information on the five attributes is necessary or not.

3. AGENTS' INTERACTION REVISITED

There are 4 types of interaction problems that must be addressed by agent-based methodologies: Cooperation, coordination, loose and strict competition [4].

3.1 Cooperation and coordination

Agent cooperation and coordination are both defined as revealing an agent's goal and knowledge behind it to the other party. The difference is that in cooperation both agents have a common goal but in coordination, each individual agent has its own goal. For cooperation and coordination, it is necessary that the goal (G) and knowledge (K) attributes be declared public for both agents. In both cases knowledge sharing is possible. In [1] a model and method for knowledge sharing for cooperation and coordination was proposed.

If Interface (I) is public for both agents: Either of the agents can subscribe to other agent's services using the published interface. For example, if **Agent_1** is a dealer agent and **Agent_2** is a tax consultant agent, and if **Agent_1** wants to calculate the total tax for the sold goods it can use the published interface of **Agent_2** for calculating tax and provide the data to **Agent_2** in order to have the total tax calculated. Similarly **Agent_2** can use the published interface of **Agent_1** to obtain the sale data autonomously.

If Interface (I) is public for **Agent_2** and protected or private for **Agent_1**: Only **Agent_1** can subscribe to **Agent_2**'s services using its published interface. For the same example, **Agent_1** can use the published interface for calculating tax of **Agent_2** by providing the data to **Agent_2**, manually. **Agent_2** cannot acquire the sale data of **Agent_1** autonomously.

If Control (C) is public for both agents: Then both agents can start their tasks under separate threads of control and extra performance requirements on synchronization and concurrency can also be addressed.

For example, if **Agent_1** and **Agent_2** are two agents controlling traffic light on two consequent interchanges, besides sharing the traffic data they can synchronize their timing in order to maximize the through pass traffic. Additional performance optimization models such as Petri Nets may be used for such purpose.

If Control (C) is public for **Agent_2** and protected or private for **Agent_1**: Then both agents can start their tasks under separate threads of control but extra performance requirement on synchronization and concurrency may not be addressed.

If Identity (Id) is public for both agents: They can directly communicate using the address (e.g., unified resource identifier URI, static stubs, etc.) of the other agent.

If Identity (Id) is public for **Agent_2** and protected or private for **Agent_1**: Only **Agent_1** can communicate with **Agent_2** directly. **Agent_2**'s answers may be delivered to a proxy or alias of **Agent_1** and used by **Agent_1** later. A typical scenario is that **Agent_1** is a software metrics data collector agent for company X and **Agent_2** is a public domain metrics calculator agent, **Agent_1** may not want to reveal its identity because of the private nature of the data but still want to receive services. Table 1 summarizes the cooperation and coordination scenarios.

Table 1. Summary: Cooperation & Coordination

<i>Agent Cooperation & Coordination Scenarios</i>		
<i>Attribute</i>	<i>Visibility</i>	<i>Requirements</i>
Goal (G)	Public	Necessary requirement
Knowledge (K)	Public	
Interface (I)	Public	Full automation and Bilateral service
	Protected or Private	Unilateral service only
Control (C)	Public	Synchronization and/or concurrency
	Protected or Private	No synchronization and/or concurrency
Identity (Id)	Public	Full communication and Bilateral service
	Protected or Private	Unilateral service only
Requirements:		
<ul style="list-style-type: none"> • Knowledge sharing and semantics level message passing methods are needed. • Additional decision making and/or reasoning methods are not needed. • Uncertainty management techniques are not needed. 		

3.2 Loose and strict competition

Loose competition is defined as revealing only an agent's goal but masking the knowledge behind it to the other party. For example, two dealer agents may compete to sell an item to a user. In this case, they have the same goal and it revealed to each other. But how they are

going to achieve the goal is not public. That is, knowledge sharing is impossible.

For loose competition it is necessary to have the Goals (G) for both agents declared public and the Knowledge (K) declared private.

If Interface (I) is public for both agents: Either of the agents can potentially know what services the other agent can offer. But this is usually irrelevant to the loose competition problem unless it gives a basis for comparison of the two competitors. For example, if two tax consultant agents compete to sell their services to a customer, usually, the services they offer is a good basis for comparison. Each agent may want to know what services the other agent can offer and if possible try to match it.

If Interface (I) is public for **Agent_2** and protected or private for **Agent_1**: If the competitor encapsulates its interface, a good strategy is to match it.

If Control (C) is public for both agents: This means that both agents can start their tasks under separate threads of control, either simultaneously, as in strategic games or sequentially, as in dynamic games. For example, if **Agent_1** and **Agent_2** are competing to sell their services, they can submit their offers simultaneously or wait for the other party to submit and then come up with a matching offer.

If Control (C) is public for **Agent_2** and protected or private for **Agent_1**: Then both agents can start their tasks under separate threads of control. Here the problem can only be formulated as a dynamic game.

If Identity (Id) is public for both agents: They cannot hide their identity from each other. Therefore changing the role and acting as a customer is impossible.

If Identity (Id) is public for **Agent_2** and protected or private for **Agent_1**: **Agent_1** can pretend to be a client for **Agent_2** and through communication via a proxy can gather valuable information on **Agent_2**. An example is that dealer **Agent_1** may pretend to be a customer for **Agent_2** and ask for offer about the goods it is selling itself and then use this knowledge to produce a match up offer to its own customers.

In case of loose competition, basically, each agent should find out whether the interface, control and identity of the opponent agent is public or not and try to use the public ones to gather more information. Such information gathering may be costly and subject to justification.

Strict competition is defined as neither revealing an agent's goal nor the knowledge behind it to the other party. For strict competition it is necessary to have the Goals (G) and the Knowledge (K) for both agents declared private. The rest of the discussion would be similar to what mentioned for loose competition. Table 2 summarizes the loose competition scenarios.

Table 2. Summary: Loose competition

<i>Agent Loose Competition Scenarios</i>		
<i>Attribute</i>	<i>Visibility</i>	<i>Requirements</i>
Goal (G)	Public	Necessary requirement
Knowledge (K)	Private	
Interface (I)	Public	Comparison of services
	Protected or Private	Encapsulate own
Control (C)	Public	Strategic or dynamic game
	Protected or Private	Strategic game only
Identity (Id)	Public	Role changing is impossible
	Protected or Private	Role changing possible
Requirements:		
<ul style="list-style-type: none"> • Ability to gather and interpret signals indirectly. • Additional decision making and/or reasoning methods, based on certainty and uncertainty, such as game and utility theory are needed. 		

4. CONCLUSIONS

In this paper a unified model for software agent interactions was presented. The interaction scenarios were classified using five attributes: goal, control, interface, identity and knowledge base. Using the model, appropriate decision making techniques for each interaction scenario were identified. Developing a library of reasoning and decision making modules that can serve as a generic and customizable decision making engine for both certain (factual) and uncertain decision making is the next step.

Acknowledgments

This research was supported by Natural Sciences and Engineering Research Council of Canada, NSERC.

References

- [1] B.H. Far, et al, "Formalization of Organizational Intelligence for Multiagent System Design," Trans. of IEICE, vol. E83-D, no. 4, pp. 599-607, 2000.
- [2] S. French, "Decision Theory: An Introduction to the Mathematics of Rationality," Wiley, 1986.
- [3] M.N. Huhns and M.P. Singh, (eds.) "Readings in Agents," Morgan Kaufmann, 1998.
- [4] H. Onjo and B.H. Far, "A Unified View of Heterogeneous Agents' Interaction", Trans. of IEICE, vol. E84-D, no. 8. pp. 945-956, 2001.
- [5] V. Parunak, S. Bruekner, J. Sauter, "ERIM Approach to Fine-Grained Agents," NASA/JPL Workshop on Radical Agent Concepts, Greenbelt (MD), 2002.
- [6] N.R. Jennings, "An agent-based approach for building complex software systems," Communications of ACM, vol. 44, no. 4, pp. 35-41, 2001.