

Knowledge-Based Software Engineering

Proceedings of the Third Joint Conference on
Knowledge-Based Software Engineering in Smolenice,
Slovakia, 1998

Edited by

Pavol Návrat

Slovak University of Technology, Bratislava, Slovakia

and

Haruki Ueno

Tokyo Denki University, Japan

IOS
Press



Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

Dealing with Cooperative and Hostile Agents on the WWW

Sidi O. SOUEINA, Behrouz H. FAR, Z. KOONO
Department of Information and Computer Sciences, Saitama University
255 Shimo-okubo, Urawa 338-8570, Saitama, JAPAN
sidi@cit.ics.saitama-u.ac.jp

Abstract. Most of the works related to software agents is based on the assumption that the other agents are trustworthy. In the growing Internet environment this may be a false assumption. We are developing software agents for Electronic Commerce (EC) that operates in various security zones. They can exchange information with trusted agents and try to predict actions of the agents that cannot be trusted. In this paper we introduce a Multi-Agent Learning Language (MALL), aimed to make agents learn from their environment about the strategies of their opponents and later on make their own plans depending on the other players moves. Details of the language in EC agents and an implementation of such agents in JAVA are discussed.

1. Introduction

Intelligent behavior of software agents should be tested and verified in uncertain and ambiguous domains. In a multi-agent environment, in order to make decisions, compete, negotiate, etc., the agents have to reason about the environment, predict its changes and other agents' future moves. However, the amount and specifications of the information available are very crucial. Any lack of information and/or noise, affect the quality of the decisions or the actions. Therefore the agents should have the learning capability, based on the primary information about targets or things being modeled, and the signals that they emanate. A variety of methods and techniques have been suggested to deal with multi-agent environments [3], such as cooperation [8], coordination [1], and Agent Oriented Programming (AOP) is defined for agents having mental qualities such as beliefs, commitments and desires [10]. In learning other agents' strategies, however, different considerations such as avoidance of exponential time, is crucial. In this work we introduce a language, aimed to make agents learn from their environment about the strategies of their opponents and later on make their own plans depending on the other players moves. An implementation of the language and its application in a multi-agent environment for Electronic Commerce (EC) is demonstrated.

2. Decision Making Under Uncertainty

In a multi-agent environment, agents are to encounter uncertainty, engendered by the lack of knowledge when devising optimal explanations to observable actions of other parties. In this project agents are categorized by *friendly*, *competitive* and *hostile* zones, in which certainty is divided into three levels based on the amount of information about the environment and about other parties, given the signal function observed before choosing among several choices [7]. These levels are:

- *Decision Making under Certainty*: In case where the agent is certain and knows exactly what state of nature to choose, and therefore will select the alternatives that give the highest payoffs.
- *Decision Making under Risk*: In case the agent is not quite sure about the actions and it only knows with the probability $0 < P < 1$ that S is/was/will be the state of nature. Thus, after calculating the expected payoffs of each alternative (A), it selects the alternative with the largest payoffs $C^* = \text{Max } C_A$ (where C_A is the payoff of alternative A).
- *Decision Making under Uncertainty*: The third situation is when agent does not know anything about the state of nature except that it is in some set $S = \{S_1, S_2, \dots, S_n\}$.

Nevertheless, in order to deal with this defined uncertainty, a four step decision strategy was suggested [11]:

1. Pessimistic: is selecting the worst possible outcome.
2. Optimistic: is selecting the best possible outcome.
3. Hurwitz: selecting the value $\alpha = \{0,1\}$ for each alternative a weighted average of the best payoffs

$$C^{*A} \text{ and the worst } C_{*A} \text{ are taken } H = \alpha C^{*A} + (1 - \alpha)C_{*A}$$

4. Normative: averaging the payoffs across n outcomes for each alternative and then selecting the best one.

The work presented here suggests to implement an uncertainty (i.e., incomplete game theoretical) based learning language for competitive and friendly agents. The main iteration in the language is based on the use of the uncertainty level in order to make the most out of the available information as the time goes by, given the whole basic knowledge available so far and dynamic knowledge-base updating through a process of learning (*signaling explanations*).

The language contains patterns such as, certainty level, prior and posterior beliefs, signal function, agent type, strategy detection, and planning, that help implementing learning algorithms. We divide the multi-agent environment into several *environs*. These are multi-agent, game theoretic and uncertainty assessment based concepts, all together construct a wider range of concepts that effect the learning processes. These concepts will be modeled using First Order Logic (FOL) here through. The process of defining the environs takes the forms as follows.

Game Environ

The game can be, strategic, zero sum, non-zero sum, repeated game with or without complete information, etc. Using MALL however an agent, for instance will be able to detect (recognize) games with incomplete information environ that consist of the following constraints:

- a finite set of players, I
- a finite set of states of nature, K
- a probability distribution P on K
- for each player i ($i \in I$) a k^i is i 's initial information
- a set of possible moves S^{ik} of player i in k^i

- a payoff functions for i , as $G^k : S \rightarrow R$
- a signaling function τ ; a finite set of signals to represent the information that i holds about the state of nature at a given time.
- a probability distribution on I given its type.

Agent Specifications

We use notion of agent type [6], i.e., defining an environment in which each agent belongs to a certain type. This gives an indication or a hint about the agent and its possible actions and beliefs about other agents and about the environment. (with some probability assessment over the belief or actions i.e., if the actions are already taken $P = 1$ if not $P \leq 1$) depending on the type of agent.

Signal Function

The observable signals that agent ($i \in I$) is to consider before taking any action. They can be related to the environment as well as to the other agents. The signals are the proof that the state in some state space T , exists with some probability, that is: $\tau : T \rightarrow K$

Conclusions (learning)

In the given environment, the learning agent is to gather a set of conclusions (new rules) as it learns. These rules are to be added to its background knowledge (BK). For example, based on the following premises,

agent B is from the hostile zone, and, the game is zero-sum, and, that the agent is partially revealing the information,

agent A might conclude that,

use another way to learn or effect B 's strategy by taking actions designed to change B 's belief, such as, bluffing, etc.

This concept is the most important part of MALL for it represents the goals of the programmer who is trying to make the agent learn by applying certain algorithm.

Strategy Specifications: (Strategy Detection and Making)

The agent is first to be prepared to detect the sort of environment it is going to deal with, detect other agents' strategies and construct its own plan.

3. Multi-Agent Learning Language (MALL)

A Multi-Agent Learning Language (MALL) is defined as being necessary for agents in environments where they encounter crucial situations in which they have to learn about the environment, other parties moves and strategies, and then construct an optimal plan. The language is based on two major factors, the level of *certainty* and *fully monitoring* (surveying) the agents and the environment. In this section we present the layout of the language and how it works. Table 1 depicts some of the basic names, predicate symbols and function symbols. The syntax of the language is defined in Table 2. The program mainly consists of a major processing part that is, also, consisted of declarative and inference statements. The declarations are consisted of predicate symbols and functions symbols with different arities.

Table 1 MALL's Symbols and Function Symbols

Names	Predicate Symbols	Function Symbols
Agent	action	action(agent)
Signal	decept	signal(action(agent))
Action	spy	profile(action,action,action)
CertLv	pun	strategy(agent)
PostBel	profile	degrcrt(action) decept(strategy(agent))
Stack	strategy	getsig(agent)
	abduct	spy(setsig(agent))
	push	pun(action(agent))
	pessi	pop(action,stack)
	bluf	opti(action,stack)
	plan	pessi(select(degrcrt(action)))
	getsig	bluf(action(agent))
	selec	plan(profile(agent))
	pop	type(agent(profile))
	opti	select(profile(action(agent)action(agent),action(agent)))
		Abduct(select(profile(action(agent),action(agent),action(agent))))
		Sigfunc(posbel(profile(action, action ,action)))

The role of explanation in learning is crucial. However, the amount of information in the *background knowledge (BK)* might not be sufficient to explain an explanatory target (ET). In such a case we have to come through a process of inference with an *explanatory hypothesis (EH)*. The EH is triggered by the *signal* in most cases, depending on the level of certainty about the relations between those signals and their explanations (BK or EH), whether it is *certainty, risk, or uncertainty*, the inference takes place i.e., *degrcrt(action)*.

ET[] WHEN infer[] SO learn[]

The **ET[]** is the signal that triggers the inference which should be performed depending on whether the background knowledge is sufficient to explain or not. The structure of the command expresses the way the signaling function $\tau^{-1} : K \rightarrow T$ works. What comes after the **SO** indicates what should be learned, and sometimes, what should be done. The conditional statement does not differ from other language statements: *IF Game_Condition THEN action ()*.

4. Implementation and Applications

A major application area is WWW based Electronic Commerce (EC). In earlier papers, we have proposed a general model of EC composed of intelligent software agents, that operate on the WWW and each agent is an expert in its own field (i.e., Ex-W-Pert agent)[4]. In our project, EC is viewed as a society of software agents that interact and negotiate with each other. We have devised 7 types of EC agents, namely: customer, search, catalog, manufacturer, vendor, bank and delivery agent [5] (see Fig. 1).

Table 2 MALL Syntax in BNF

```

< program > ::= < processing >
< processing > ::= < statement > *
< statement > ::= < enviState > *
< enviState > ::= < declaration > * | < inferStat > *
< declaration > ::= < prediSymb > * | < FuncSymb > *
< prediSymb > ::= < action(agent) > | < strategy(agent) > | < dgrcrt(action) > | < getsig(aget) >
< FuncSymb > ::= < signal(action(agent)) > | < profile(action,action,action) > |
    < type(agent(profile)) > | < spy(getsig(agent)) > | < decept(strategy(agent)) > |
    < pun(action(agent)) > | < push(action,stack) > | < pop(action,stack) > |
    < opti(action,stack) > | < pessi(select(degrcrt(action))) > |
    < bluf(action(agent)) > | < plan(profile(agent)) > |
    < select(profile(action(agent),action(agent),action(agent))) > |
    < abduct(select(profile(action(agent),action(agent),action(agent)))) >
< inferStat > ::= IF < declaration > * THEN < declaration > * |
< ExplTar > WHEN < declaration > * SO < learn > * < declaration > * |
    < ExplTar > WHEN < inferSate > * SO
    < learn > * < declaration > *
    < ExplTar > ::= < signal(action(agent)) >
< learn > ::= < push > * | < pop > *
    
```

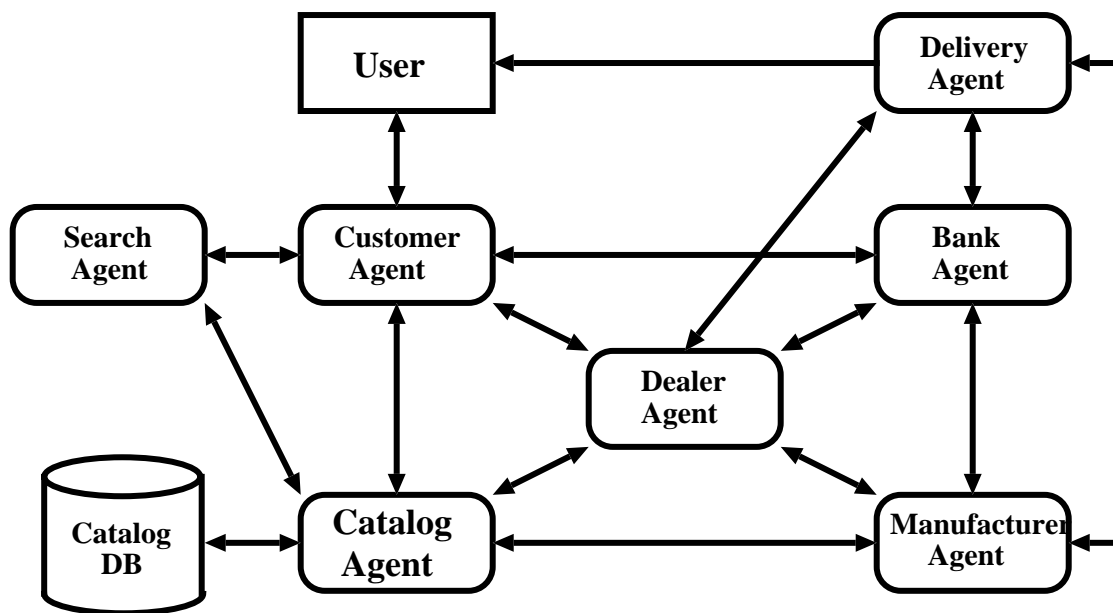


Figure 1 7 types of EC agent

4.1 A General Model for EC Agents

Fig. 2 shows an overview of the EC agents. Similar to conventional expert systems, each agent, no matter local or remote, has its own *local knowledge base* and *reasoning engine*. Compared to the conventional expert systems, a main difference is that all agents have additional *communication engine* and *documentation engine*.

The communication engine is mainly responsible for maintaining connection to the other agents and managing messages. The documentation engine perform three main tasks: acquiring data from the other agents, as requested by the reasoning engine; performing

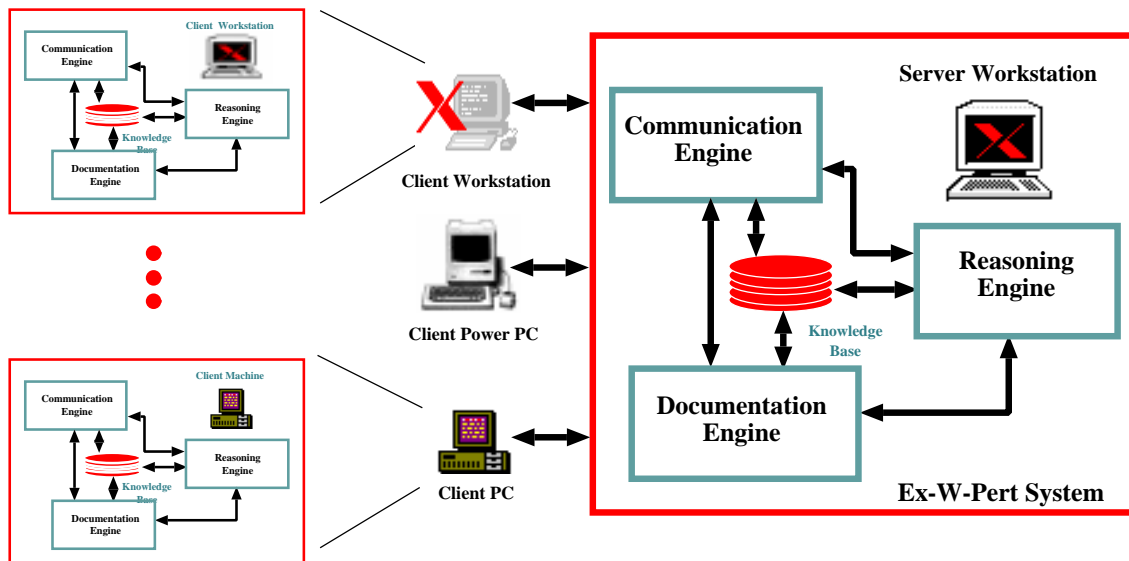


Figure 2 Internal Structure of EC agents

index and relational keyword search; and preparing and reformatting data items to be appropriate for transferring over the internet. From implementation point of view, each agent is actually composed of:

- A data translator for converting the input data to an internal frame data structure and conversely, convert the internal frame structure to output data.
- A reasoning engine together with customization and learning modules.
- A local multi-layer knowledge base, as described above, including domain rules, case data base etc.
- An interface module that can be plugged into a window-based user interface for interacting with the user and reporting thus fulfilled jobs.
- A documentation engine for converting data to and from the hypertext and other formats suitable for transferring over the WWW.
- A communication engine, for launching http, ftp, etc., applications, managing I/O messages and communicating to the other agents.

The system is implemented using JAVA programming language. The MALL language is currently bundled with the *dealer* and *manufacturer* agents when competing against each other by applying individual strategies attempting to increase their own *utilities*. For instance, a dealer applying a strategy with lower cost and better services to the customers will be more likely to increase its income.

For the purpose of this example, let us say that given the basic knowledge of the environment (the market), plus, the signals of dealer agent B is sending, the agent A tries to learn B's strategy. Obviously agent B is omniscient about his own strategy and given that his utility (benefit) is higher than A's we can say that the game is zero-sum (since the benefit of B is the loss of A) and A by analyzing the signals that B gives will get some hints. Agent B can be said to *play partially revealing* to make his opponent get the least indicating signals. On the other hand, A in this situation would construct his own plan through a process of explanation and analysis of the information revealed by B.

5. Discussion and Conclusions

Some current issues in EC are mentioned in [2] and [8]. The traditional idea of Electronic Commerce, by means of electronic data interchange (EDI) and private or local area networks (LAN) has been around for some years. What Internet based EC adds to this is changing from private networks to a global network together with interactive data interchange, with a reasonable cost, but at the expense of lower security. Moving to a global network implies less control over the kind and amount of data available. A new method for agents autonomously learning other agents strategies and construct their own plans, was introduced. A first step of a Multi-Agent Learning Language, we believe is to be the base of a new era of making agents, act, think, decide and learn, etc, in a completely possible independent way. This version of the language is based on game theoretical concepts, specifically in incomplete information games where players (agents) are non-omniscient.

The present paper stated the very first layout of the language and demonstrated the basic concepts. In coming works we intend to define more detailed components we believe are necessary for standardizing the multi-agent learning, such as belief estimation, cost estimation and optimal move calculation. MALL is used in EC's competitive dealer and manufacturer agents at a first stage.

References

- [1] M.F. Aghraham, "Modern Sociological Theory" Oxford University Press, (1982).
- [2] N. Adam and Y. Yesha (Edts.), "Electronic Commerce: Current Research Issues and Application," Springer Verlag, 155 p., (1996).
- [3] J.M. Bradshaw (Edt.), "Software Agents," MIT Press, 480 p., (1997).
- [4] B. H. Far, H. Mukai and Z. Koono, "Intelligent Agents for Electronic Commerce," *Proc. Annual Conference of Japanese Society for Artificial Intelligence JSAI' 97*, pp. 482-485, (1997).
- [5] B.H. Far, J. Nagayama and Z. Koono, "A WWW Based Intelligent Search Engine for Electronic Commerce," *Proc. 2nd Joint Conference on Knowledge Based Software Engineering, JCKBSE' 96*, pp. 174-181, (1996).
- [6] J.C. Harsanyi, "Games with Incomplete Information Played by Bayesian Players," Parts I, II, III. *Management Science*, vol. 14 (3,5,7), (1967 & 1968).
- [7] K.J. Engimann, "Decision Making With belief Structures: An Application In Risk Management," *Int. J. of Uncertainty Fuzziness and Knowledge based systems*, vol. 4, no. 1, pp. 1-25, (1996).
- [8] R. Pyle (Edt.), "Special Issue on Electronic Commerce on the Internet," *Communications of The ACM*, vol. 39, no. 6, (1996).
- [9] J.S. Rosenshien, "Rational Interaction: Cooperation Among Intelligent Agents," *PhD thesis*, Computer Sciences Dept., Stanford University, Stanford CA, (1985).
- [10] Y. Shoham, "Agent-Oriented Programming," *Artificial Intelligence*, vol. 60, pp. 51-94, (1993).
- [11] R.R. Yager, "On ordered weighted averaging aggregation operators in multi-criteria decision making," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 83-190, (1988).