Theme 2: Agent based software engineering

Abstract

In software engineering community there is an increasing effort of design and development of multi-agent systems (MAS). However, agent system development is currently dominated by informal guidelines, heuristics and inspirations rather than formal principles and well defined engineering techniques. In this project we devise a framework for agent-based software development called Agent-SE. We present methods to generate organizational information for cooperative and coordinative agents.

1. Introduction

Nowadays, an increasing number of software projects are being revised and restructured in terms of multi-agent systems (MAS). Software agents are considered as a new experimental embodiment of computer programs and are being advocated as a next generation model for engineering complex, heterogeneous, scalable, open, distributed and networked systems. However, agent system development is currently dominated by informal guidelines, heuristics and inspirations rather than formal principles and well-defined engineering techniques. There are some ongoing initiatives by the Foundation for Intelligent Physical Agents (FIPA) (http://www.fipa.org) and some other institutions to produce guidelines and standards for heterogeneous, interacting agents and agent-based systems. However, such initiatives fall short to address the design, development, issues in MAS. Here we present a framework for agent-based software development called Agent-SE. We present methods to generate organizational information for cooperative and coordinative agents.

2. Knowledgeability in MAS

Traditional software engineering can handle data and information. Data is defined as a sequence of quantified or quantifiable symbols. Information is about taking data and putting it into a meaningful pattern. Knowledge is the ability to use that information. Knowledgeability is an integral part of MAS paradigm. Agents in order to interact and work proactively must be knowledgeable in their area of expertise. Knowledgeability can be defined in terms of cognitive capabilities and interactions. The following subsections elaborate this.

2.1 Cognitive Capabilities

Three main capabilities of agents in MAS are representing, using and sharing the knowledge. Symbol structure (SS) is used to model individual agent's knowledge structure (see Section 4.2). Ability to use the knowledge can be realized by having a knowledge-base in the form of SS and mechanisms for problem solving using the knowledge base. Finally, ability to share

the knowledge depends on ontologies for the domain and task. Mechanisms for using and sharing knowledge are presented in [ONJO01].

2.2   Interaction in MAS

Basic agents' interactions are cooperation, coordination and competition.

1. *Cooperation:* Cooperation is revealing the agent's goal and the knowledge behind it to the other party. In cooperation both agents have a common goal.
2. *Coordination:* Coordination is revealing the agent's goals and the knowledge behind it to the other party. In coordination, the agents have separate goals.
3. *Loose Competition:* Loose competition is revealing only the agent's goal but masking the knowledge behind it to the other party.
4. *Strict Competition:* Strict competition is neither revealing the agent's goal nor the knowledge behind it to the other party.

Figure 1 shows decision making mechanism based on agents' interaction. Conventionally, it is believed that for a pair of agents to interact, each should maintain a model of the other agent, as well as a probabilistic model of future interactions [HUHNS99]. This is unnecessary when using SS representation in cooperation and coordination cases [ONJO01].
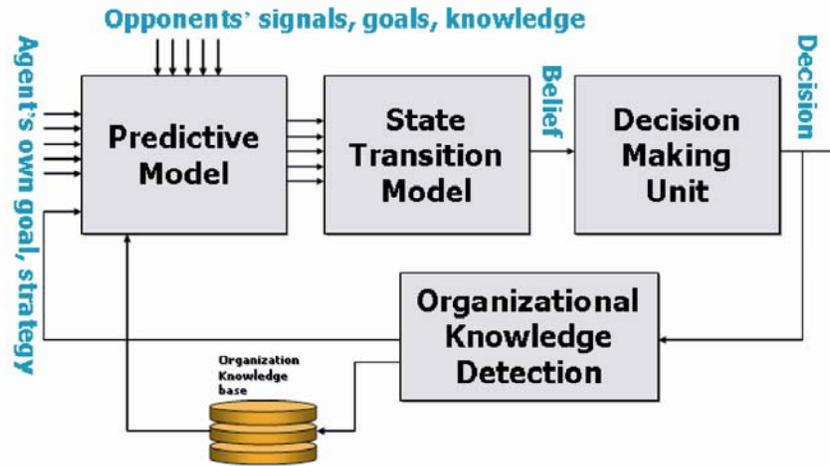

Fig. 1. Decision making mechanism

3. The Agent-SE Approach

In this section we propose a method for multiagent system design, called Agent-SE. The Agent-SE design steps are as follows:
1. Design use-cases that show the interactions of the MAS with the external world.

2. Devise a candidate set of agents that can realize the use-cases based on functions, inputs and outputs.
3. Design the task ontology of the problem.
4. Build an abstraction model and add interactions and signal level organizational relationships using the task ontology.
5. Design each agent and its internal intra-actions using conventional SE techniques (preferably, object-oriented design with UML, etc.) and a predefined agent model, if necessary.
6. Based on the domain ontology, design each agent's knowledge-base using Symbol Structure. (See Section 4).
7. Derive and record symbol level organizational properties based on interactions of pairs of cooperative or coordinative agents. (See Section 4).

Organizational formation, maintenance and updating are typical of the dynamic nature of groupings in complex systems. Agent-based systems require computational mechanisms for dynamic formation, maintenance and disbanding of organizations. One such mechanism is presented in Section 4.

4. Organization

Organization is a goal directed coalition of software agents in which the agents are engaged in one or more tasks. Control, knowledge and capabilities are distributed among the agents. The already proposed organizational models for multiagent systems have certain drawbacks. First, they cannot explain the organizational knowledge in terms of its comprising agents without reference to any other intermediary concepts. Second, they cannot provide frameworks for comparing and evaluating different organizations. Third, the organizational knowledge base cannot be updated dynamically, accounting for different configuration of the participant agents. Finally, they cannot explain the need for services of a certain agent in an organization. All of these factors are necessary in organization design and are addressed in our research.

4.1 Assumptions

Intelligence of Pair (IoP): The already proposed theories have implicitly assumed that Organizational Intelligence (OI) exists and implemented using a meta-agent (e.g., directory and ontology service agents) [CARLEY99]. However there are certain difficulties in both logical formulation and actual implementation of such theories. This is mainly due to ignoring the dynamic interactions among the agents when devising the components of OI.

A point in our research is that in a purposeful (i.e., not random) organization, OI is a property of interaction among agents and can only be ascribed to at least a pair of agents. We call this Intelligence of Pair (IoP) assumption.

History of Patterns (HoP): In biological coalitions, participants may have a kind of role or function (during interaction with the other participants), if they show some persistence in their profile of actions over time. The same could be devised for artificial coalitions. As a matter of fact, it is not difficult to find organizations that display non-random persistent and repeated patterns of actions [CARLEY99].

Agents act and perform in a physical world. Their past experiences can be recorded and explained in terms of their histories, that is, their profile of actions and states that they go through. Intuitively, histories can display certain patterns. A basic feature of state representation is that it assigns a certain characteristic to its reference agent. Therefore it is possible to define OI patterns with reference to agents' history.

Another point is that OI patterns emerge from discovering a persisted state or an ordered pattern in the agent's profile of actions. We call this History of Patterns (HoP) assumption. IoP and HoP assumptions account for dynamic interactions and a computation method based on this assumption is proposed below.

4.2 Modeling

Symbol structure (SS) is used to model individual agent's knowledge structure. SS is a finite connected multi-layer bipartite graph. There are two kinds of nodes in each layer of SS: concepts (*c*) and relations (*r*). One source of difficulty when processing concepts, is distinguishing a concept at various levels of abstraction, as well as differentiating between generic concepts and their instances. Function type is defined to ease such differentiation. The function type maps concepts and relations onto a set *T*. The elements of *T* are called type labels. Type hierarchy provides a means of evaluating a concept at various levels. The type hierarchy is a partial ordering defined over the set of type labels, *T*.

We can mention that SS is semantically richer than semantic networks because both the concepts and relations are augmented with types. Furthermore, it explicitly defines mechanisms for combining and using the private knowledge together with the agents' profile of actions in the MAS environment.

4.3   Reasoning Rules

Join rule: Join rule merges identical concepts. If a concept *c* in symbol structure *u* is identical to a concept *d* in symbol structure *v*, then let *w* be the symbol structure obtained by deleting *d* and linking to *c* all arcs of relations that had been linked to *d*.

Simplification rule: Redundant relations of the same type linked to the same concept in the same order can be reduced by deletion all but one.

B.H. Far's Research
© B.H. Far
far@ucalgary.ca
http://www.enel.ucalgary.ca/People/far

Generalization/ Specialization rule: For two arbitrary levels *p* and *q* of any SS, if *p* is identical to *q* except that some type labels of the nodes of *q* are restricted to subtypes of the same nodes in *p*, then *p* is a specialization of *q*, and *q* is called a generalization of *p*.

4.4   Computational OI

Here we propose a method for generating organizational concepts based on the IoP and HoP assumptions and definitions of interaction. In case of cooperation and coordination the agent's private knowledge is exposed to the other party. In this method, first, a pair of agents is selected and by using join, simplification, generalization and specialization rules their SS are merged (see Figure 2). There are finite sets of actions and states associated with each agent. Each agent's state transition is represented by a non-deterministic finite state automaton (NFSA).

Both *actions* and *states* take concepts and relations as their attributes. For each agent, the NFSA state transition model has an initial state *idle* and final state *goal*. A sequence of actions that convert the *idle* state to *goal* is a plan of actions towards a goal. Theoretically, all such sequences form a *regular language* (RL) whose elements are generated by a *regular grammar* (RG) equivalent to the NFSA.

As both actions and states take concepts and relations of SS as their attributes, in the case of cooperation and coordination, a joint sequence of actions can be generated by matching the actions whose attributes are concepts or relations belonging to the same type hierarchy. Such sequences form the joint plans of actions handled partially by either of the agent pair. The proposed algorithm is as follows:

1. Select an agent pair, Agent (G_1) and (G_2).
2. Merge their SS using join, simplification, generalization and specialization rules.
3. Select mode of operation based on external information.
4. Select the NFSA model and generate sequences of actions for the selected mode.
5. Compare two such sequences $w_i \in L_{G1}$, and $w_j \in L_{G2}$ where L_{G1}, and L_{G2} are regular languages of agent (G_1) and (G_2), respectively.
6. For a common action $a \in w_i$ and $a \in w_j$ check the type of attributes of *a*. If the attributes belong to the same type hierarchy, merge the sequences from that point on after adjusting the types.
7. Record such joint sequences and check for possible repetition and/or persistence patterns in the future course of actions.
8. When repetition and/or persistence becomes present, add such strings to the *organizational knowledge base*.

Examples of our electronic commerce project [FAR98] are presented below. In the electronic commerce project seven types of agents, *customer*, *dealer*, *manufacturer*, *delivery*, *banker*, *search* and *catalog* agents work together and/or compete to do business on the Web [FAR98]. By default, the *dealer* agent only knows about the goods to be sold, the *delivery*

agent knows about transporting goods, *banker* agent has information on customers and their credit and/or cash accounts and *customer* agent is a personal assistant agent for a human user and has information on the user's preferences, etc.
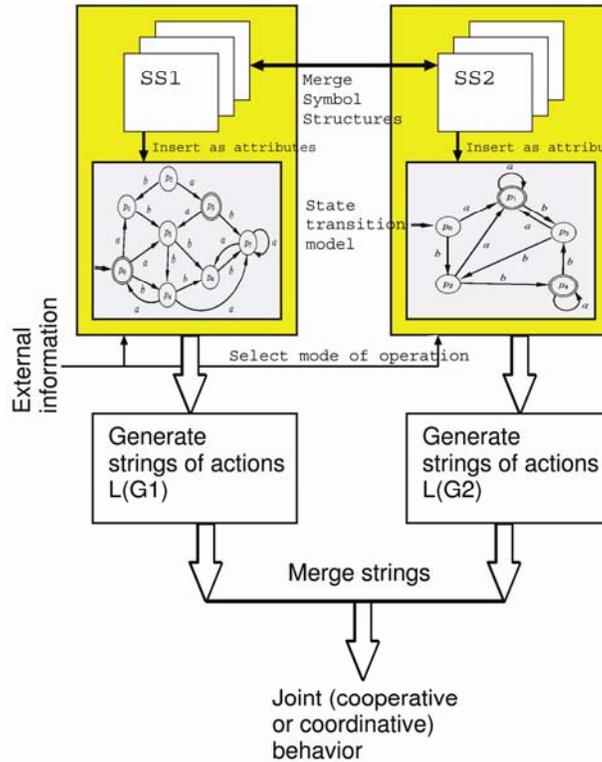

Fig. 2. Deriving organizational knowledge

4.5 Example: Cooperation

Let's consider a case of a *dealer* agent and a *delivery* agent cooperating to sell and deliver an article of commerce, such as a PC to a human user. A portion of SS for the *dealer* and a *delivery* agents is shown in the upper portion of Figure 3. It is visible that the concept Goods is a super-type for PC for both agents. However, based on the agents' roles, the concepts may have different data associated with them. For example, PC in *dealer* agent's SS may be associated with CPU, memory, etc. However, for *delivery* agent the same concept may have weight and size as its attributes.

The lower part of Figure 3 depicts an example of the sequence of actions for the *dealer* and *delivery* agents. It is shown that the action transport(Goods,...) for the *delivery* and transport(PC,..) for the *dealer* agents have attributes belonging to the same type hierarchy. Therefore, the strings can be merged by adjusting the type and changing transport(Goods,...) to transport(PC,..) for the *delivery* agent and let the plan be executed by assigning this action to the *delivery* agent.

4.6 Example: Coordination

Let's consider a case of a *dealer* agent and a *banker* agent coordinating their operations in order to exchange information related to a particular human user. Apparently the *dealer* and *banker* agents have separate goals but they coordinate their activities to help each other. Let's assume that the *dealer* agent already receives a purchase order from a human user through his/her *customer* agent. A portion of the *dealer* agent's SS is shown in the upper right part of Figure 3.

The *dealer* agent interacts with the *banker* agent by sharing its SS with that of the *banker* agent. By merging the two SS, the customer concept is common between the two SS and in this way the *dealer* agent can verify that the customer has an Account and from the *banker* agent's SS verify that there are either Cash or Credit accounts available. Also it can verify that the user may also use a Cash Card. Therefore, the *dealer* agent can contact the user to get data for Cash or Credit the accounts. It can further verify genuineness of the data supplied by the user by consulting the *banker* agent again. In this simple example, knowledge sharing is used to enable the *dealer* agent to successfully proceed with the selling task in spite of possessing only a limited amount of knowledge about the user without implementing a redundant customer database within the *dealer* agent.
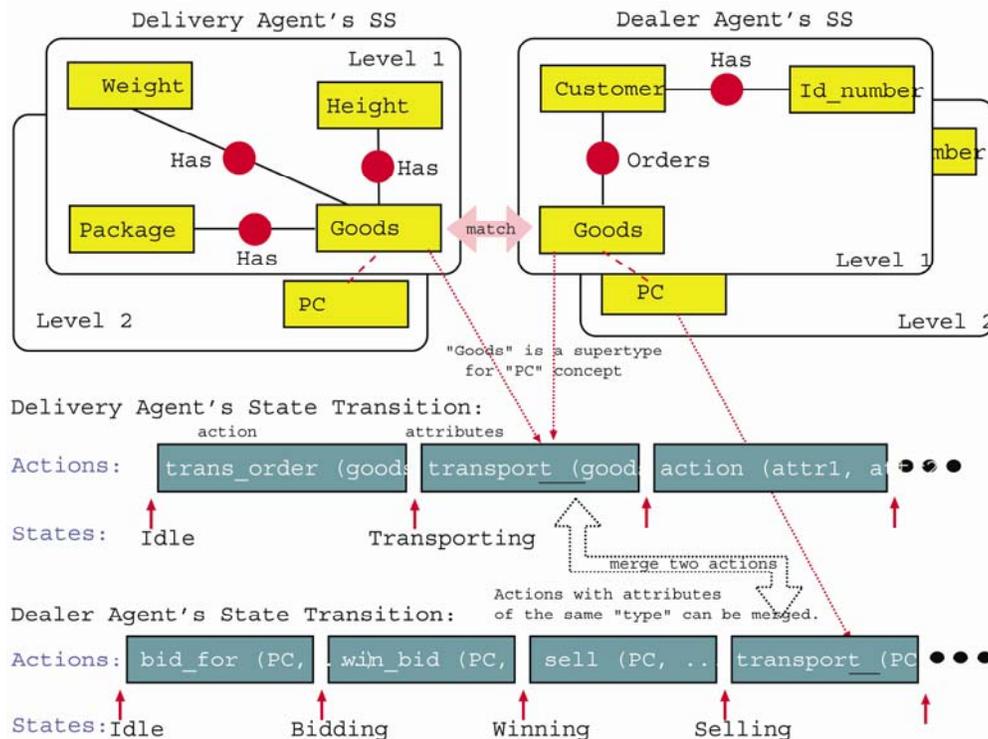

Fig. 3. A portion of SS for dealer agent

5. Conclusions

In this proposal, we proposed a method for MAS design. We show that multiagent system design can be achieved at the expense of additional design steps including, design of domain ontology and agents' internal knowledge structure. Actually, designing domain ontology is not a totally new task in software engineering practice. It has been carried out informally in almost all of the contemporary software engineering techniques in the form of data definitions, glossary or non-functional specification. Applications using the framework and techniques described above, such as a multi-agent system for electronic commerce [FAR98, FAR00] have already been developed.

References

[CARLEY99] K.M. Carley and L. Gasser, "Computational Organization Theory," in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, ed. G. Weiss, G., pp. 299-330, MIT Press, 1999.

[FAR98] Far, B.H., et al., "An Integrated Reasoning and Learning Environment for WWW Based Software Agents for Electronic Commerce," IEICE Trans. Inf. and Syst., vol. E81-D, no. 12, pp. 1374-1386, 1998.

[FAR00] Far, B.H., et al., "Formalization of Organizational Intelligence for Multiagent System Design," IEICE Trans. Inf. and Syst., vol. E83-D, no. 4, pp. 599-607, 2000.

[HUHNS99] Huhns, M.N. and Stephens, L.A., "Multiagent Systems and Societies of Agents," in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, ed. G. Weiss, pp. 79-120, MIT Press, 1999.

[ONJO01] Onjo, H., and Far, B.H., "A Unified View of Heterogeneous Agents' Interaction," Transactions of the IEICE, vol. E84-D, no. 8, pp. 945-956, 2001.