電気学会研究会資料

ケミカルセンサ研究会

2000年6月16日

社団法人電気学会

東京都千代田区五番町6-2
Integrated Reasoning and Learning Using Multiagent Systems

マルチエージェント技術による推論と学習

Behrouz H. Far  (Faculty of Engineering, Saitama University)

ベルーズ H. ファー（埼玉大学 工学部）

Abstract In this paper we present the agent-oriented approach to software engineering (Agent-SE) as a logical evolution of contemporary approaches to software engineering. We also present a method to generate organizational information using formal descriptions of the participant agents. We argue that Agent-SE can be used in designing, and implementing complex, heterogeneous, distributed and networked software systems.

1. Introduction

Software development is a very high risk task. Only about 1/6 of software projects are successful and almost all of the software projects costs exceed initial estimation. In order to manage such a crisis, software engineering paradigms have been evolved from device oriented assembly languages to procedural and structured programming, to Object Oriented programming, to distributed objects and component-ware and to design patterns. Due to the increasing popularity of the Internet, heterogeneous, scalable and networked software systems are highly needed. However, neither of the software engineering paradigms could make software technology keep up with the current business needs.

Nowadays, an increasing number of software projects are being revised and restructured in terms of software agents. 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.

Agent research encompasses many disciplines including but not limited to Software Engineering (SE), Artificial Intelligence (AI), and Operation Research (OR).

Agent system development is currently dominated by informal guidelines, heuristics and inspirations rather than formal principles and well-defined engineering techniques. Unfortunately, there has been comparatively little work on viewing agent system design and development as a software engineering paradigm that has the potentiality to enhance software developments in a wide range of applications. There is no standard way of incorporating agent-oriented viewpoint into design and development of agent-based software systems. Specifically, there is no standard technique for system requirement (specification) analysis, design, implementation, testing, debugging and verification of agent based software systems.

In this paper we argue that the development of heterogeneous, robust and scalable software systems requires software agents that can complete their objectives while situated in a dynamic and uncertain environment, engage in interactions with other agents or humans and operate within flexible organizational hierarchies.

We also argue that agents can be used as a framework for bringing together the components of Artificial Intelligence (AI) and Software Engineering (SE) and Operation Research (OR) that are necessary to design and build intelligent artifacts.

Finally, we present agent oriented software engineering (Agent-SE) as a new method for designing multiagent systems by blending ideas borrowed from AI, SE and OR.

The structure of this paper is as follows. In Section 2 we discuss how scalability and complexity can be handled in agent systems. In Section 3 the Agent-SE approach is introduced. In Section 4 a method to derive organizational information is presented. Section 5 gives a brief
discussion on the future of agent based software engineering techniques followed by a conclusion.

2. Software System Complexity

Business software has a large number of parts that have many interactions (i.e., complexity). The role of software engineering is to provide techniques that make it easier to handle such complexity [JENNINGS2000].

2.1 Complexity in SE

Complexity in software systems is structural in nature. As a software system evolves its structure deteriorates and major efforts is needed to maintain it. Therefore, hierarchical design is a major way of handling complexity in software engineering. A complex software system is usually composed of many interrelated subsystems, each of which is in turn hierarchic in structure. And the relationships among the components are supposed to be dynamic.

Two kinds of relationships can be devised: interactions among subsystems and intra-actions within subsystems. Interactions are between an artifact (or a component) and its outer environment. Intra-actions are the characteristics of the artifact's (or a component's) inner environment.

Complexity in software engineering is handled via problem decomposition and abstraction. Decomposition is dividing a large problem into smaller, more manageable units each of which can then be dealt with in relative isolation. Abstraction is to define a simplified model of the system that emphasizes some of the details or properties, while intentionally neglecting the others. Attention can be focused on some aspects of the problem, at the expense of the other less relevant details. All of the contemporary software engineering paradigm, such as: object-orientation, component-ware, design patterns and software architectures provide techniques for decomposition and abstraction.

2.2 Complexity in AI

Complexity in AI is handled via using ontologies and applying synthesis techniques. Ontology in AI refers to a set of concepts or terms that can be used to describe some area of knowledge or build a representation of it. Interest in ontologies has grown due to interests in reusing or sharing knowledge across systems. Developing reusable ontologies that facilitates sharing and reuse is a goal of ontology research [CHANDRASEKAREN99].

Synthesis works exactly opposite to decomposition. In synthesis, one first defines a subclass of problem to be solved and builds a simplified model or prototype system that will be later incrementally updated to account for additional properties.

2.3 Complexity in OR

Complexity in OR is handled via studying organizations and deriving organizational properties, i.e., identifying and managing the interrelationships between the various system components. This helps designer tackle complexity in two ways: By enabling a number of basic components to be grouped together and treated as a higher-level unit of analysis (e.g., the individual components of a subsystem can be treated as a single coherent unit by the parent system) and by providing a means of describing high-level relationships between various units (e.g., a number of components may cooperate to provide a particular functionality).

3 Agent Oriented SE Approach

In this section we propose a method for multiagent system design, called Agent-SE, based on the abstraction and decomposition (Section 2.1), ontology and synthesis (Section 2.2) and organizational properties (Section 2.3). Agent-SE offers:

1. An effective way of decomposition (partitioning the problem space) and synthesis.
2. A means of introducing abstraction to the model.
3. An appropriate way of modeling and viewing organizational relationships of complex systems.

Some novel points in this research are:

1. Decomposition is based on the function and input/output rather than conventional data/object.
2. Participating agents are described by:
   - Specifying their input/output and/or functions (interfaces).
   - Specifying their inner environment (classes).
3. Organizational properties are derived dynamically.

The agent-oriented approach advocates decomposing problems in terms of autonomous agents and then synthesizing the system using those agents. Thus the whole system is simply the collection of independently developed abstract software agents that are interacting.
with one another dynamically.

Note that this notion of decomposition is different from conventional reductionist approach to decomposition. Decompositions are based upon function/input/output rather than those based upon data/objects in conventional object-oriented SE. This matches the natural multiple loci of control in real world complex systems [JENNINGS2000]. This decentralization, reduces the amount of interaction between components. The system's total control know-how is taken from a centralized repository and localized inside each individual agent.

Autonomy, means that the agents have control over their own behavior and that they decide for themselves which actions they should perform and when.

The abstraction model of each agent means describing them in terms of high-level terms from an ontology of the domain and encapsulating their actual implementation and only specifying the participating agents in terms of their input/output, interactions and organizational relationships.

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.

3.4 Design Methodology
The Agent-SE design steps are as follows:
1. Decompose the problem based on function/ input/ output into subsystems: an organization of agents
2. Build the abstraction model by defining interactions and organizational relationships
3. Design each agent and its internal intra-actions using a predefined agent model
4. Derive the organizational properties based on interactions of pairs of agents

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.

Organizations, of various forms, physical, cognitive, temporal and institutional have been studied in operation research, management and computer sciences. The game theoretic approach to study organization focuses on modeling and suggesting computational algorithms for certain aspects of the coalitions, such as social welfare [SANDHOLM99], individual rationality, voting consensus, etc. The computational approach focuses on identifying general principles of organization and their exceptions.

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
a) Intelligence of Pair (IoP)
All of the proposed theories and formalisms have implicitly assumed that Organizational Intelligence (OI) exists and implemented using a meta-agent (e.g., directory and ontology service agents) (such as [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.

b) 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. Flexibility, extendibility and interoperability are three main advantages of knowledge representation and reasoning with SS.

4.3 Reasoning Rules

a) Join rule:
Join rule merges identical concepts. If a concept c in u is identical to a concept d in 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.

b) 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. If the relations r and s in the symbol structure u are duplicates, then one of them may be deleted from u together with all its arcs.

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

4.4 Interaction Among Agents

Now we have a framework for representing and reasoning with the knowledge on an individual agent basis. Knowledge sharing by moving from one agent to another and on an organizational basis requires defining the basic agent interactions, i.e., cooperation, coordination and competition. For a pair of agents to interact, each should maintain a model of the other agent, as well as a model of future interactions [HUHNS99].

a) Cooperation:
Cooperation is revealing an agent's goal and the knowledge behind it, i.e., its symbol structure to the other party. In cooperation both agents have a common goals.

b) Coordination:
Coordination is revealing an agent's goals and the knowledge behind it, i.e., its symbol structure to the other party. In coordination, agents have separate goals.

c) Loose Competition:
Loose competition is revealing only an agent's goals but encapsulating the knowledge behind it to the other party.

d) Strict Competition:
Strict competition is neither revealing an agent's goals nor the knowledge behind it to the other party.

Therefore, knowledge sharing is equivalent to merging two or more symbol structures using join, simplification, generalization and specialization rules.

4.5 Computational OI

We propose a computation method for generating OI concepts based on the IoP and HoP assumptions (see Section 4.1). Figure 1 depicts the idea. In this method, first, a pair of agents are selected and by using reasoning rules (see Section 4.3) their pairwise profile is produced. Then by using a simple pattern detection algorithm, possible repetition and persistence patterns are derived and added to the knowledge base of the organization.

Applications using the framework and techniques
described here, such as a multiagent system for electronic commerce are under investigation and development [FAR98], [FAR2000].

5. Discussion and Conclusion

The history of computer sciences is full of conceptual paradigms that could not survive to their expectations. Will agent-SE have the same faith? The main reason for believing survival of the agent-based software engineering is that Agent-SE is a natural next step in the evolution of a whole range of software engineering paradigms, such as object-orientation, design patterns and component-ware.

There may be certain drawbacks when adopting software agents, such as performance and security, real-time system control, global perspective and optimality issues. Agents may not be appropriate for systems and domains with global constraints, real-time response requirements, and deadlocks. As in multiagent systems all decisions are made based on local knowledge and reaching globally optimal performance is almost impossible.

There are still many research issues and unsolved problems, ontology sharing and uncertainty are two main issues. Shared ontologies can form the basis for domain specific knowledge representation languages, thereby eliminating the need for replicating the knowledge analysis process. Uncertainty may be further broken down to uncertainty about the parameters of the environment, actions of the other agents, reasoning of other agents, imperfect information of the events happened, etc.

References

[FAR98]

[FAR2000]

[CHANDRASEKAREN99]

[CARLEY99]

[JENNINGS2000]

[HUHNS99]

[SANDHOLM99]