ISSN (Online): 2230-7893
www.IJCEM.org

Agent-Based Software Engineering

P. Santosh Kumar Patra
Associate Prof.
Department of CSE, ACT, Chennai, TN, India

Abstract

Agent-based computing represents an exciting new synthesis both for Artificial Intelligence (and, more generally, Computer Science. It has the potential to significantly improve the theory and the practice of modeling, designing, and implementing computer systems. Yet, to date, there has been little systematic analysis of what makes the agent-based approach such an appealing and powerful computational model. Moreover, even less effort has been devoted to discussing the inherent disadvantages that stem from adopting an agent-oriented view. Here both sets of issues are explored. The standpoint of this analysis is the role of agent-based software in solving complex, real-world problems. In particular, it will be argued that the development of robust and scalable software systems requires autonomous agents that can complete their objectives while situated in a dynamic and uncertain environment, that can engage in rich, high-level social interactions, and that can operate within flexible organizational structures.

Keywords: Agent, multi-agent, Environment-Rules-agent, decomposition, abstraction, organization.

1. Introduction

1.1 Production systems

One of the simplest but nevertheless effective designs for an agent is to use a production system. A production system has three components: a set of rules, a working memory and a rule interpreter. The rules each consist of two parts: a condition, which specifies when the rule is to executed ('fire'), and an action part, which determines what is to be the consequence of the rule firing. For example, an agent might be designed to roam over a simulated landscape, collecting any food that it encounters on its travels. Such an agent might include a rule that states (in an appropriate symbolic programming language): IF I am next to some food, THEN pick it up. This would be one of many rules, each with a different condition. Some of the rules would include actions that inserted facts into the working memory (e.g. I am holding some food) and other rules would have condition parts that tested the state of the working memory (e.g. IF I am holding food THEN at it). The rule interpreter considers each rule in turn, fires those for which the condition part is true, performs the indicated actions for the rules that have fired, and repeats this cycle indefinitely. Different rules may fire on each cycle either because the immediate environment has changed or because one rule has modified the working memory in such a way that a new rule begins to fire. Using a production system, it is relatively easy to build reactive agents that respond to each stimulus from the environment with some action. It is also possible, but more complicated, to build agents with some capacity to reflect on their decisions and thus begin to model cognition. Another possibility is to enable the agents to change their own rules using an adaptive algorithm which rewards rules that generate relatively effective actions and penalizes others.

1.2 Learning

Production system based agents have the potential to learn about their environment and about other agents through adding to the knowledge held in their working memories. The agents' rules themselves, however, always remain unchanged. For some problems, it is desirable to create agents that are capable of more fundamental learning: where the internal structure and processing of the agents adapt to changing circumstances. There are two techniques commonly used for this: neural networks and evolutionary algorithms such as the genetic algorithm. It is possible for either an individual agent to be modeled using a neural network, or a whole society to be represented by a network, with each neuron given an interpretation as an agent (although in the latter case, it is hard to build in all the attributes of agents usually required for multi-agent modeling).

1.3 The Agent's environment

Agents are almost always modeled as operating within some environment consisting of a network of interactions with other agents. Sometimes, however, it is also useful to model them within a physical environment that imposes constraints on the agents' location. The usual assumption is then that nearby agents are more likely to interact or are better able to influence each other than those farther apart. Models of this kind may be built using techniques drawn from work on cellular automata. A cellular automata consists of a regular array of cells, each of which is in one of a small number of states (e.g. 'on' or 'off'). A cell's state is determined by the operation of a few simple rules and depends only on the state of its neighboring cells and its own previous state. Cellular automata have been studied intensively by mathematicians and physicists because they are useful models of some physical materials and have some interesting computational properties. The classic cellular automata have only very simple mechanisms for determining the state of their cells, but the same principles

can be used with more complex machinery so that cells represent agents and the array as a whole can be used to model aspects of a society.

Guidelines in building environment, rules and agents

The problems arising when we go from simple models to complex results suggest that a crucial role for the usefulness and the acceptability of the experiments is played by the structure of the underlying models. For this reason, we introduce here a general scheme that can be employed in building agent-based simulations.

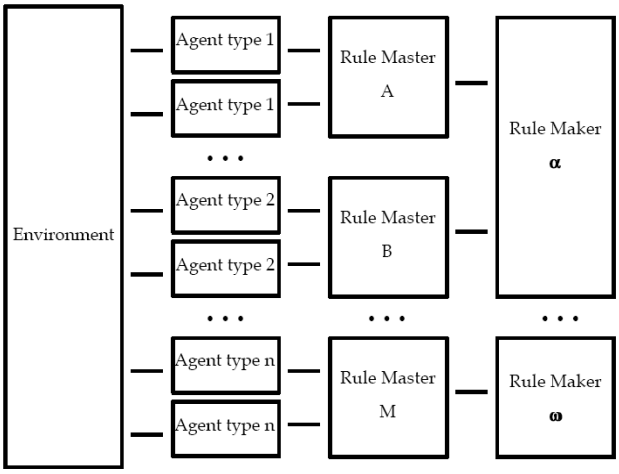

Fig.1 - The Environment-Rules-Agents framework to build agent based computational models

The main value of the Environment-Rules-Agents (ERA) scheme shown in Figure 1 is that it keeps both the environment, which models the context by means of rules and general data, and the agents, with their private data, at different conceptual levels. To simplify the code, we assume that agents should not communicate directly, but always through the environment; that means, the environment allows each agent to know the list of its neighbors. This is not mandatory, but if we admit direct communication between agents can make the code more complex. With the aim of simplifying the code design, agent behavior is determined by external objects, named Rule Masters, that can be interpreted as abstract representations of the cognition of the agent. Production systems, classifier systems, neural networks and genetic algorithms are all candidates for the implementation of Rule Masters. We may also need to employ meta-rules, i.e., rules used to modify rules. The Rule Master objects are therefore linked to Rule Maker objects, whose role is to modify the rules mastering agent behavior. Rule Masters obtain the information necessary to apply rules from the agents themselves or from special agents charged with the task of collecting and distributing data. Similarly, Rule Makers interact with Rule Masters, which are also responsible for getting data from agents to pass to Rule Makers.

2. Agent Development Characteristics

Agent Based Software Engineering effectiveness claims are based upon three strategies for addressing complex systems: *decomposition*, *abstraction*, and *organization* and that the "agent-oriented mindset" gives one an advantage in using these strategies. Agents have objectives that gives one a clear way to decompose the problem into agents. That agent systems work largely by emergent behavior and handle errors gracefully reduces the need for detailed specifications, since not all interactions need be specified in advance, and allows more abstraction to be used in system building. And finally, agent systems are naturally hierarchical organizations themselves. Indeed, it is important to note that agent identity is a fundamental component of all agent languages and methodologies for interactions.

The notions introduced in that one aspect of emergent behavior is having software modules being able to model themselves and other modules, which leads to modules that are able to attempt different methods of accomplishing a task based upon runtime data and these models, which in turn leads to flexibility and robustness without the necessity of the programmer having correctly considering every possibility. The fundamental idea here is that the programmer focuses on the types of interactions possible without specifying all possibilities in advance and one technique for doing this is modeling interaction behaviors. These notions are important and fundamental

3. CASE STUDY: IN HEALTH CARE

3.1 Introduction

Health care at all levels - local, regional, national and international - is a vast open environment characterized by shared and distributed decision making and management of care, requiring the communication of complex and diverse forms of information between a variety of clinical and other settings, as well as the coordination between groups of health care professionals with very different skills and roles. It is the aim of health care software systems to operate effectively in this environment providers & the usual properties of intelligent agents match quite precisely with our needs in this field (basically with the requirement of having autonomous intelligent proactive collaborative entities in a distributed environment).

3.2 Reliability Theory

Reliability theory is the foundation of reliability engineering. For engineering purposes, reliability is defined as: the probability that a device will perform its intended function during a specified period of time under stated conditions.

ISSN (Online): 2230-7893
www.IJCEM.org

$$R(t) = Pr\{T > t\} = \int_t^{\infty} f(x)\,dx$$

where **f(x)**is the failure probability density function and *t* is the length of the period of time (which is assumed to start from time zero).

4. RESULTS OBTAINED

4.1. Case Study 1

The case study one is done using standard data collected. Around 80% of data is used for training purpose and rest 20% of data is used for the testing so that the result obtained can be compared with the standard data and the accuracy level of neural network can be assessed. Here inputs 1& 2 are inputs to simulator and expected and obtained are shown below table1:

Table 1: Set of data sets

Input1	Input2	Expected output
0.029800	0.800000	0.977364
0.029800	0.960000	0.976650
0.629800	0.368900	0.792536
0.329800	0.698900	0.796503
0.929800	0.998900	0.408812
0.699800	0.568900	0.665736
0.589800	0.533900	0.727370
0.454500	0.960000	0.646120
0.404500	0.660000	0.767112

Result obtained

0.976441
0.971797
0.792681
0.794138
0.395036
0.671583
0.729866
0.646411
0.765696

Analysis: After training the neural network, the weights (case one) are generated which are summarized below. On the left most side integer numbers represent the layer number and followed by the weights of each neuron on that layer. The number of weights represent the number of links from one node of a layer to the node of next layer. Results obtained:

• average error per cycle =0.09927
• error last cycle =0.079994
• error last cycle per pattern=0.009
• total cycles = 85530
• total patterns = 1068870

4.2. Case Study 2

Here is a list of standard data below:

Table2: Set of data sets

Inputs1	Input2	Expected result
0.1236	0.4563	0.945162
0.8575	0.6652	0.565294
0.1236	0.2525	0.969272
0.0298	0.8000	0.976441
0.0356	0.2525	0.991051
0.1236	0.4563	0.945162
0.9298	0.9989	0.395036
0.6998	0.5689	0.671583
0.5898	0.5339	0.729866
0.4545	0.9600	0.646411
0.4045	0.6600	0.765696
0.5845	0.9613	0.570136
0.0845	0.8613	0.929805
0.0356	0.3636	0.987139
0.0356	0.0036	0.999871

Result obtained

0.945168
0.565299
0.969279
0.976448
0.991063
0.945168
0.395031
0.671589
0.729860
0.646409
0.765699
0.570149
0.929806
0.987135
0.999877

Results obtained:

• average error per cycle =0.027662
• error last cycle =0.007597
• error last cycle per pattern =0.000019
• total cycles = 98000
• total patterns = 6880000

Conclusion

In this position paper we want to argue that multi-agent systems have a set of characteristics that make them appropriate to be used to improve the provision of health care to citizens. They may be integrated with existing applications, for example agents may access a database to obtain the information about the patients of a certain hospital. The agents in a multi-agent system may be running in different locations, for example there may be an agent associated to each department of a medical centre, or

an agent associated to each person that is included in a health care program in a certain community. The autonomy of each agent in a multi-agent system permits to maintain the independent views of each modeled actor, for example each agency involved in the provision of health care to a community, such as social workers, health care professionals or emergency services may have different private policies that determine their relationship with other agents and their individual decisions. Information agents may help both citizens and health care professionals to obtain up-to-date and relevant health care information from Internet. Agents may help to address the growing demand of patient-centre management of medical data, for example it is feasible to think about the possibility of having personal agents which are able to get in touch with agents at a medical centre to receive information about their medical record or to make an appointment to be visited by a doctor.

References

1. Fox, J., Das, S. Safe and sound: Artificial Intelligence in hazardous applications.
AAAI Press/MIT Press, 1st edition, (2000)
2. Weiss, G. Multi-agent systems. A modern approach to Distributed Artificial Intelligence M.I.T. Press, (1999)
3. Wooldridge, M. An introduction to Multiagent systems. John Wiley, Chichester, (2002)
4. Shankararaman, V. (ed.) Proceedings of the Workshop on Agents in Health Care, at the 4th International Conference on Autonomous Agents - AA 2000, Barcelona, (2000)
5. Corts, U., Fox, F., Moreno, A. (eds.). Proceedings of the Workshop on Agent Applications in Health Care, at the 15th European Conference on Artificial Intelligence, ECAI 2002, Lyon, (2002). A revised and extended version of the best papers will appear in 2003 in a special issue of AI-Communications.
6. Moreno, A. and Garbay, C. (eds.), Special issue of Artificial Intelligence in Medicine devoted to Software Agents in Health Care. (2003) (in press)
7. AgentCitiesWorking Group on Health Care Applications. More information available at the web page http://wwwcms.brookes.ac.uk/hcwg
8. Petrie, C. Agent-based software engineering. In: Agent-Oriented Software Engineering.
Lecture Notes in Artificial Intelligence, Vol. 1957. Springer-Verlag, Berlin, (2001), 58-76
9. Jennings, N. On agent-based software engineering. Artificial Intelligence 117, (2000), 277-296
10. Decker, K., Li, J. Coordinated hospital patient scheduling. Proceedings of the 3rd International Conference on Multi-Agent Systems, ICMAS-98. Paris, France, (1998)
11. Kumar, A.D., Kumar, A.R., Kekre, S., Prietula, M.J., Ow, P.S. Multi-agent systems and organizational structure: the support of hospital patient scheduling. In: Proceedings of the Leading Edge in Production and Operations Management, South Carolina, USA, (1989)
12. Marinagi, C. et. al. Continual Planning and scheduling for managing patient tests in hospital laboratories, AIM 2000, (2000), 139-154.
13. John Nealon and Antonio Moreno 13. Aldea, A., Lpez, B., Moreno, A., Riao, D., Valls, A. A Multi-Agent System for Organ Transplant Co-ordination. In: Quaglini, S., Barahona, P., Andreassen, S. (eds.): Artificial Intelligence in Medicine. Lecture Notes in Computer Science, Vol. 2101, Springer-Verlag, Berlin, (2001), 413-416.
14. Silverman, B.G., Andonyadis, C., Morales, A. Web-based health care agents
.