An analysis of the contributions of the agent paradigm for the development of complex systems[1]

Rosario Girardi
Department of Computer Science, Federal University of Maranhão
São Luís - MA - Brazil

Abstract

This article analyzes main concepts of the agent-based development paradigm and its basic mechanisms to support the development of complex systems: decomposition, abstraction and flexible interactions.

Keywords: Agent-based Software Engineering; Agent-based Software Development; Agent-Based Programming; Agent paradigm; Software complexity; Intelligent systems; Rational agents

1. Introduction

Building software of high quality is difficult because of the natural complexity of software, usually characterized by many interacting components. Looking for appropriate techniques to confront complexity, software development paradigms have evolved from structured to object-oriented approaches [6, 7] to agent-oriented ones [11, 13, 22].

In spite of recent controversies around agent versus distributed object-oriented development [17], many authors strongly recommend the agent computational model for the development of complex systems, mainly because an agent is a very useful software abstraction to the understanding, engineering and use of this kind of systems [10, 21, 25].

This article analyzes main concepts of the agent-based development paradigm and its basic mechanisms to support the development of complex systems. Section 2 characterizes the agent paradigm and introduces the rational agent approach of Artificial Intelligence. Autonomy, the main feature of intelligent agents, is also analyzed. Section 3 discusses how agent attributes allow approach complexity in software development through appropriate mechanisms for software decomposition, abstraction and flexible interactions between components.

2. The agent paradigm

Main contributions to the agent development paradigm come from the Artificial Intelligence area. This discipline has proposed four main approaches for the construction of intelligent computer systems [15]:

[1] R. GIRARDI, "An analysis of the contributions of the agent paradigm for the development of complex systems", In: JOINT MEETING OF THE 5TH WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS (SCI 2001) AND THE 7TH INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS ANALYSIS AND SYNTHESIS (ISAS 2001), Orlando, Florida. 2001.

- the Turing test approach (systems that think like humans);
- the cognitive approach (systems that act like humans);
- the approach of reasoning laws (systems that think rationally); and
- the rational agent approach (systems that act rationally).

Systems that exhibit rational behavior characterize the agent approach for the construction of intelligent systems. Acting rationally means acting to achieve goals considering own beliefs.

Actions taken by a rational agent are supposed to be the right ones, those that cause the agent to be most successful. A performance measure establishes the criterion that determines how successful an agent is. Russel and Norvig [15] characterize the rationality of an agent, at a given time, through the following aspects:

- The performance measure that defines the degree of success.
- Everything that the agent has perceived so far (the complete perceptual history)
- What the agent knows about the environment
- The actions that the agent can perform.

An ideal rational agent should select and execute the action that is expected to maximize its performance measure, considering the perceptual history and the built-in knowledge the agent has.

The agent approach is more general than the one based on reasoning laws, where emphasis is on correct inferences. Multi-agent systems are composed of a community of reactive and deliberative interacting agents.

Making correct inferences often characterizes the behavior of an agent because one of the ways to behave rationally is logically reason to the conclusion that a particular action will allow achieve the agent goals. However, making correct inferences is not sufficient enough for rational behavior. There are situations where a simple reflexive action can be more effective than an action taken more slowly through some inference mechanism.

Deliberative agents have a symbolic reasoning model. They have a plan to be pursuit and negotiate with other agents to achieve their goals.

Reactive or reflexive agents do not have a reasoning model. They act as a response of an environment stimulus. Even though reactive agents do not use traditional artificial intelligent representations, they can exhibit some kind of intelligent behavior through the interaction with other agents in a multi-agent system.

Software methodologies for agent-based development are still in research phase [5, 12, 16, 20, 23, 24]. However, many agent-based applications have already been developed in a variety of industrial (Air Traffic Control, Manufacturing, Process Control, Transportation Systems), commercial (Information Management, Electronic Commerce, Business Process Management) and medical (Patient Monitoring, Health Care) application domains [2, 3, 18].

The agent paradigm appears as a natural evolution of the object-oriented one. However, an agent software abstraction is more than a passive object with memory and behavior and can be seen as a kind of active object, autonomous, social and able to learn [1, 5, 8, 19].

Having control over its own behavior is the main issue distinguishing agents over objects. In spite that an object encapsulates both state and behavior, it cannot activate its own behavior. An object can invoke public accessible methods of any object. Once the method is invoked, corresponding actions are performed. In this sense, objects are not autonomous because they are totally dependent on each other for the execution of their actions.

We are considering a strong definition for autonomous behavior. Autonomy is supported both by the agent own experience and by the built-in knowledge used when constructing the agent for the particular environment in which it operates. Therefore, if agent actions are based completely on built-in knowledge, such that it need pay no attention to its percepts, then we say that the agent lacks autonomy.

Agents are assumed to be continually active, and are typically engaged in an infinite loop of observing their environment, updating their internal state, and selecting and executing an action to perform. In contrast, objects are assumed to be passive most of the time, becoming active only when another object requires their service invoking one of their methods.

Finally, a multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of control.

3. Attributes to approach complexity

The agent paradigm provides support for mechanisms approaching the complexity of software, like decomposition, abstraction and flexible interactions [5, 9, 10].

Through decomposition, a complex problem is divided into more simple sub-problems that can be approached in a independent manner. Designers controls complexity by concentrating at a given time only in one simple problem.
An abstraction mechanism allows to define a more simple model of a problem, emphasizing the more relevant aspects and hiding details that are irrelevant in that level or particular situation.

Interaction is probably the most important characteristic of complex software. For instance, in open systems it is impossible to know at design time exactly what components the system will be comprised of, and how these components will be used to interact with one another. To operate effectively, in such systems, the ability to engage in flexible autonomous decision making is critical.

Agent-based software decomposition

Agent-based software decomposition provides effective mechanisms for the decomposition of a complex system problem domain, allowing approach a complex problem through a community of autonomous agents interacting in a flexible manner. Such decomposition results in a natural representation of distributed systems.

This kind of decomposition reduces the level of coupling between system components. As agents are active and autonomous, they know when they should act and when they should update their state, differently from a passive object, which needs to be invoked by other external entity to do it. That reduces the complexity of control, which is not more

centralized but localized in each agent component.

Agent-based software abstraction

Software Engineering usually confront the complexity of the real world looking for appropriate abstraction mechanisms to map the real world to the computational one, i.e. trying to reduce the semantic distance between these two worlds.

An agent component is an effective software abstraction [5]. An agent abstraction provides a natural metaphor for modeling complex systems, mainly because there is a strong correspondence between the notion of subsystems and agent communities. Both involve a set of components acting and interacting according to their roles.

Agent interactions

The agent paradigm simplifies the development of complex systems because agents can decide about the kind and scope of their interactions at run time.

Usually, because of system complexity, it is difficult to know at design time all possible interactions that can occur. For this reason, software components should be able to initiate interactions and respond to other agents in a flexible manner.

Autonomy allows the design of agents able to attend no pre-established requirements and able to provide assistance when needed.

On the other hand, in the agent paradigm, control management between components is considerably reduced. All agents are active and any required coordination is performed bottom-up through agent interaction. So, high level interactions do not need to be prescribed rigidly in design time. They can be initiated at run time in a way context sensitive.

Therefore, multi-agents systems are suitable for open systems, because they are capable of reacting to unforeseen events, exploiting opportunities when they arise, and dynamically reaching agreements with system components whose presence could not predicted at design time. However, it is difficult to know how to specify and implement such systems.

4. Concluding remarks

The demand of complex software applications is constantly increasing and current software development practices are not allowing the construction of this kind of software appropriately.

The rational agent approach of the Artificial Intelligence discipline is contributing to the Software Engineering field in the formalization of the agent development paradigm.

There is an intense research work towards the systematization of methods and methodologies for the construction of multi-agent systems mainly because they provide several mechanisms to approach the complexity of software systems. A lot of work remains to be done in this area, both on theory and practice.

We are adopting a reuse-based approach to multi-agent system development trying to find solutions not only to complexity problems of software but also to the ones

of productivity and quality [5].

5. References

1. Bradshaw, "Software Agents", The AAAI Press, 1997.
2. B. Burmeister, "Models and Methodology for Agent-Oriented Analysis and Design", In: Proceedings of KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, 1996.
3. B. Burmeister and M. Haddadi, "Application of multi-agent systems in traffic and transportation", IEEE Proceedings on Software Engineering, v. 144, p. 51-60, 1997.
4. C. Iglesias, M. Garijo and J. Gonzalez, "A Survey of Agent-Oriented Methodologies", In: Intelligent Agents V. Agents Theories, Architectures, and Languages, Lecture Notes in Computer Science, Springer-Verlag, 1998.
5. R. Girardi, " Software Abstractions in Agent-Based Application Engineering". In: Joint Meeting Of The 5th World Multiconference On Systemics, Cybernetics and Informatics (Sci 2001), 2001, Orlando, Florida.
6. R. Girardi and R. T. Price, "A Class Retrieval Tool for an Object-Oriented Environment". In: International Conference on Technology on Object-Oriented Languages And Systems (Tools'90), Sydney, 1990.
7. R. Girardi and R. T. Price, "The Object-Oriented Development Paradigm", Revista de Informática Teórica e Aplicada, Porto Alegre, v.1, n.2, 1990. p.26-36. (In Portuguese)
8. S. N. Hyacinth, "Software Agents: An Overview", Knowledge Engineering Review, v. 11, n. 3, p. 205-244, 1996.
9. N. Huhns and L. M. Stephens, "Multiagent Systems and Societies of Agents", In: "Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence", G. Weiss (ed.), The MIT Press, 1999.
10. N. R. Jennings, "On Agent-based Software Engineering", Artificial Intelligence, 117 (2000) 277-296.
11. N. R. Jennings**,** K. Sycara and M. Wooldridge**,** "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems, v.1, n.1, p.7-38**,** 1998**.**
12. J. Odell, H. Van Dyke Parunak and B. Bauer, "Extending UML for Agents", Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, pp. 3-17.
13. J. Odell, "Agent Technology", Object Management Group, Agent Working Group, OMG Document ec/2000-08-01, 2000.
14. D. Riecken (ed.), Communication of the ACM, Special Issue on Intelligent Agents, v. 37, n. 7, p. 18-21, 1994.
15. S. Russel and P. Norvig, “Artificial Intelligence: A Modern Approach”, Prentice Hall, 1995.
16. Y. Tahara, A. Ohsuga and S. Honiden, "Agent system development method based on agent patterns", In: Proceedings of International conference on Software engineering, 1999, p. 356-367.
17. M. Schroeder, "Are distributed objects agents?", Proceedings of AOIS'99, 1999.
18. H. Van Dyke Parunak, "Industrial and Practical Applications of DAI", In: "Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence", G. Weiss (ed.), The MIT Press**,** 1999.
19. G. Weiss (ed.), "Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence", The MIT Press**,** 1999.
20. M. Wood and S. A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology", In: Proceedings of the First International Workshop on Agent-Oriented

Software Engineering (AOSE-2000), June 2000.

21. M. Wooldridge, Agent-based software engineering, IEEE Proc. Software Engineering 144 (1) (1997) 26-37.

22. M. Wooldridge, Agent, "Intelligent Agents", In: "Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence", G. Weiss (ed.), The MIT Press, 1999.

23. M. Wooldridge, N. Jennings and D. Kinny , "The Gaia Methodology for Agent-Oriented Analysis and Design", Journal of Autonomous Agents and Multi-Agent Systems, v. 3, 2000.

24. M. Wooldridge and N. R. Jennings, "A Methodology for Agent-Oriented Analysis and Design", In: Proceedings of Autonomous Agents '99.

25. M. Wooldridge and P. Ciancarini, "Agent-oriented Software Engineering: The State of the Art", In: "Agent-Oriented Software Engineering", Springer-Verlag, Lecture Notes in AI Volume 1957, 2000.