Event-Based Blackboard Architecture for Multi-Agent Systems

Jing Dong and Shanguo Chen
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083, USA
jdong@utdallas.edu

Jun-Jang Jeng
IBM T.J. Watson Research Center
1101 Kitchawan Road, Route 134
Yorktown Heights, New York 10598
jjjeng@us.ibm.com

Abstract

Developing large multi-agent systems is a complex task involving the processes of the requirement, architecture, design and implementation of these systems. In particular, the architectural design is critical to cope with the increasing size and complexity of these systems. The multi-agent system that accesses a central repository is typically based on the Blackboard architectural pattern. However, the control strategies of the agents are inherently complex. In this paper, we present an event-based control strategy to manage the access of central repository by multi-agents. In this approach, the Blackboard pattern is composed with the Implicit Invocation pattern to separate the control policies from the control component so that the architecture is ease of evolution, reuses and changes.

Keywords
Multi-agent systems, architectural pattern, implicit invocation, blackboard, software architecture.

1. Introduction

Software systems built as society of autonomous and intelligent agents promise providing computation power to deal with difficult problems. However, large scale multi-agent systems are complex and difficult to develop. In some applications, agents may be devoted to solve computationally intensive problems where all agents are known a priori. In other open systems, agents may have to make use of service and knowledge found in other agents or environment. In this case, the agent may dynamically register/enter and withdraw/leave from the distributed and open systems. An agent may assume a role as control to manage other agents based on some control policy. The development of large multi-agent systems also needs to address system-level properties, such as performance, reusability, adaptability, security, dependability, and mobility. These properties, which require strategies to control the application logic and data, need to be considered at early architecture and design stages in the software development life cycle. In general, the use of software architectural patterns may greatly help to cope with the complexity in multi-agent systems. For example, the multi-agent system that accesses a central repository is typically based on the Blackboard architectural pattern [4]. In this case, the control strategies of the agents are inherently complex and difficult to maintain.

Implicit Invocation [4][17] is a common architectural pattern that offers engineering benefits of loose coupling between system components that may be agents, objects, processes, and servers. It provides powerful ability to implement the interactions in multi-agent systems. Each agent can announce (or broadcast) one or more events. Other agents in the system can register the interest of an event by associating a procedure with the event. When an event is announced, the broadcasting system (connector) itself invokes all of the procedures that have been registered for the event. This architectural pattern is suitable for applications that involve loosely-coupled collection of components, each of which carries out some operations and may be in the process of enabling other operations. It is particularly useful for applications that must be reconfigured on-the-fly by, for example, changing a service provider, enabling or disabling capabilities. The use of implicit invocation pattern in the blackboard architecture may help to cope with the complexity in the control strategies and ease the evolutions and changes in the multi-agent systems. In this paper, we discuss the combination of the blackboard architectural pattern and the implicit invocation pattern to manage and control the access of central repository by multi-agents.

Object [3] and agent share many common characteristics and are also distinguished from each other. From traditional object-oriented perspective, an object is not autonomous. Its internal behavior can be solicited only by external requests. An object, therefore, is not capable of deciding what to do in a specific situation. For example, it may not be able to decline a request. In traditional object applications, environment is not explicitly specified and may be encapsulated as resources in terms of internal attributes. In current practice, however, objects in distributed and concurrent systems become closing to our view of agents. An object may be active and may integrate internal threads of control enabling them to perform specific computational tasks. It can process concurrent requests and prevent unsafe or unauthorized requests. Modern objects systems exhibit characteristics resembling to agents, whereas agents themselves are often implemented in terms of active objects, e.g., event handling capabilities.

In the next section, we discuss the incorporation of implicit invocation pattern in the blackboard architecture to manage multi-agents with an example of an online auction system. In the last two sections, we present related work and conclude the paper.

2. Event-Based Blackboard Architecture

In this section, we propose the definition of event-based blackboard architecture which is constructed from the integration of two popular architectural patterns: the Blackboard pattern and the Implicit Invocation pattern. In this approach, the control policies are separated from the control agents so that they can be changed independently.

2.1 Blackboard

The Blackboard architectural pattern has been widely used to tackle the problems with the characteristics of uncertainty, non-deterministic. There is often no direct algorithmic solution to these problems and best effort of approximation is good enough. Applications of this architectural pattern can be found in different kinds of software systems, such as communication, mobility, coordination and real-time [11]. There are three main components in this architectural pattern: knowledge sources, blackboard and control.

The knowledge source is a specialist at solving certain aspects of the overall problem. Once it finds the information it needs on the blackboard, it can proceed without any assistance from other knowledge sources. A knowledge source can be represented in a number of ways, such as simple function or collection of complex and changing logical relationships. Knowledge sources have two major sub-components: condition and action. Condition specifies when a knowledge source has something to contribute. When the condition is met, it invokes appropriate action. Action includes the modification or placement of new facts on blackboard.

The blackboard is the source of all data on which a knowledge source will operate and is the destination for all conclusions from a knowledge source. There are two types of knowledge: static and dynamic. Static knowledge is unchanging long lived information, whereas, dynamic knowledge is the knowledge that is generated during execution.

The control is a manager that considers each knowledge source's request to approach the blackboard in terms of what the knowledge source can contribute and the effect that the contribution might have on the developing solution. The manager attempts to keep problem solving on track, to insure that all crucial aspects of the problem receive attention, and to balance the stated importance of different specialist's contributions. Control components are also knowledge sources but focus on the problem solving process rather than (or in addition to) specific domain expertise. The control can be implemented by directly invoking the appropriate knowledge sources or through remote procedure/method invocation. In this case, the control policies are hard-coded with the particular control making it difficult to change the control and the control policies independently. In multi-agent systems, the control role can be assumed by different agents at different time. An agent may enter the system by registering as a control and leave the system with a new successor to join in. The dynamic changing of control requires all agents that may take the control role to carry the control policies with them. Otherwise, there may be unexpected results while changing control. On the other hand, the control policies are not fixed and may be changed over time. As a consequence, each agent that is possible to be as a control has to be notified for the changes. More specifically, there are several effects on the system-level properties of the software architecture.

The coupling of control and the control policies does not accord with separation of concerns [6] which is one of the main principles of software engineering. It is one of the main causes of complexity in the blackboard architecture. It is hard to switch the control from one agent to another on-the-fly. The control policies also become difficult to modify.

Large multi-agent systems are required to be dependable and survivable. The control component is a critical part of the systems. The failure of one control agent should not result in the failure of the whole system. New agent should be able to take the control role and make the transition smoothly and transparently.

After the systems have been defined, the maintainability of the systems is also crucial. The evolution and change of system agents should be supported by the architecture. The architecture of the multi-agent systems should be able to cope with evolution so that the impact on the system parts is isolated and minimized. New agents are able to enter the systems as knowledge source or control, whereas old agents may leave the systems. Agents are also adaptable to different environment. In this situation, agents are reusable in different systems and environments.

2.2 Implicit Invocation

Implicit Invocation architectural pattern is suitable for applications that involve loosely-coupled collection of components, each of which carries out some operations and may be in the process of enabling other operations. In this pattern, each agent registers interest in particular events that may be announced by other agents. When the events are announced, all agents that are interested in the events are notified. They are implicitly invoked.

In implicit invocation architecture, the announcers of events do not know which agents will be affected by those events. An agent cannot make assumptions about the order of processing. Thus, the serving order of events may be different from the requesting order. An agent cannot make assumptions about what processing will occur as a result of their events (perhaps no agent will respond).

There are several advantages of the implicit invocation architectural pattern. First, the agents are more independent from each other than those with explicit invocation. Second, the interaction policy can be separated from interacting agents. Third, static name dependencies are not wired in. Thus, dynamic reconfiguration is easy. An agent does not need to know the identity of other agents that can provide requested services. Fourth, any component can be introduced into a system simply by registering it for the events of that system. Fifth, it eases system evolution since agents may be replaced by other agents without affecting the interfaces of other agents in the system. On the other hand, there are some disadvantages of this pattern. First, there is no control over the sequence of implicit invocations. Second, a central management is needed to keep track of events, registration and dispatch policies. Third, event handling may interact badly with other run time mechanisms.

2.3 The Composition

The software architecture of large multi-agent systems is not constructed from scratch. Architectural patterns are used frequently to solve some recurring problems. In the system architectures, it is likely to have many instances of different patterns working in concert to form the solution to the system requirements. The composition of some of these patterns may become a new pattern [18]. In this context, we discuss the composition of the Backboard architectural pattern and the Implicit Invocation architectural pattern.

Figure 1 illustrates the architecture diagram of the event-based blackboard system including knowledge source agent, control agent and blackboard. The data on the blackboard may consist of two parts: knowledge data and control policies. The control policies can be accessed only by control agents, while knowledge data (i.e. application data) can be accessed by both kinds of agents. Alternatively, the control policies may reside in a separate place. In this case, the control strategies of the system depend on the control agent. The solid lines with arrow-head represent the access of knowledge data and control policies, whereas the dotted lines with arrow-head denote implicit invocation. In this architecture, knowledge source agents and control agents are invoked implicitly. Consequently, they are independent and can be changed dynamically.

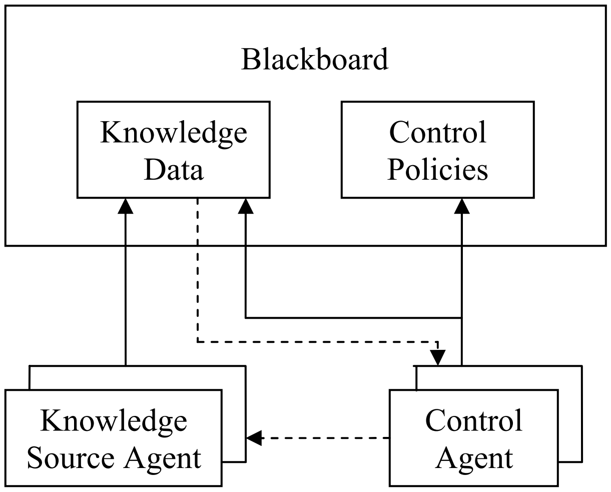

Figure 1: Event-based Blackboard System

Consider the example of an online auction system with multi-agents registered as bidders and auctioneers in a dynamic environment. The bidders can join or leave the system at anytime. An auctioneer may also be replaced by another one. The auction rules may vary according to the items auctioned. In this example, the auction board can be considered as the Blackboard that holds the related information of the articles to be auctioned, and the auction rules (i.e. control policies). The auctioneer is registered as a control agent who control the auction processes based on the auction rules. Since the auction rules are separated from the control agent (auctioneer), the control agent can be replaced by another one in case of failure or other accidents with little impact on the system. A bidder may register itself as a knowledge agent and get the list of items to be auctioned. The bidder may register the interests of some of the items, thus is notified each time when the status of the items is changed, e.g. a higher price is bid.

As presented previously, the Implicit Invocation architectural pattern is suitable for the applications that involve loosely-coupled collection of agents. By incorporating this pattern with the Blackboard system, an agent is not directly communicating with the blackboard and other agents. They communicate with each other through sending and receiving events. For example, bidder agent and auctioneer agent do not know each other. Nevertheless, the auctioneer agent can control the bidder agents according to the auction rules. They can exchange information in an implicit way to achieve the ultimate goal (the sale of an item to whom gives the highest prices). The blackboard may register the interest of the write event from all knowledge source agents. Each knowledge source may send a write event when they have something to contribute. When the blackboard changes its states, it announces a state-change event. The control agent may register the interest for the state-change event generated by the blackboard. When it catches the state-change event, the control agent may select the knowledge source(s) based on the current control policy. The control policy contains rules at meta-level or at system level. For example, knowledge sources can contribute to the blackboard only in alternative order. Knowledge source A can contribute only after knowledge source B. Each time a new price is bid on the blackboard, all bidders (knowledge sources) should be notified.

As the result of this solution, the system is no longer managed directly by the control agent. Instead, a level of indirection is added into the system architecture such that the control agent is independent from the control policies. As a result, the control agent can be replaced on-the-fly. For example, an auctioneer agent can leave during the auction and the auction still can proceed smoothly if a new agent registers as auctioneer since both of them follow the same auction rules. In addition, multiple control agents may work in parallel as long as they share the same control policy and adopt some synchronization mechanism. In this case, the system is more dependable and secure in face of the failure of control agents.

The control policy can be modified without any effects to the control agents since it has been separated from the control agents. Each control agent has to be informed for any modification of the control policies if the control policies are carried within the control agent. As a consequence, inconsistencies may appear when some control agents are not notified. Therefore, the separation allows modifying the control policy consistently. For instance, different auction rules can be defined for different items to be auctioned. These rules are not dependent on the knowledge and experience of a particular auctioneer. Therefore, the change of the auction rules can be carried out by all auctioneers.

There are several implementation issues in the composition of the two patterns. Several programming languages and environments support event handling. In this case, the interface of each agent defines a set of incoming procedure calls and a set of outgoing events. Event-procedure binding connects agents together. In this context, the procedures are registered with events. Agents communicate by announcing events for particular service. When an event is announced, the associated procedures are implicitly invoked. There are several implementation issues:

- Where are the control policies kept? The control policies can be stored as part of blackboard where the control agents are able to access. On the other hand, the control policies can be saved in a separate place.
- How are the control agents authorized and authenticated? Since control agents can register into the system and leave the system on-the-fly, it is important to make sure they are who they are and they can do what have been authorized.
- How can multiple control agents work in parallel? In this case, the synchronization of the control agents becomes a critical implementation issue. Each available control agents may process the first event in the event queue. Alternatively, all events can be classified into several categories so that each control agent may handle one kind of events.
- Who can change the control policies and how to do it? The control policies describe the rules that a control agent needs to follow while managing the knowledge sources. The policies can be modified by some special system agent or by a particular control agent. For consistency, the modification is more likely to be done by a single designated agent. For the online auction example, an auctioneer normally cannot modify the auction rules. Typically, the administrator of the website can do it.
- Can control policies be modified on-the-fly? Many agent-based systems cannot be stopped for maintenance and modification. Changes of these systems have to be conducted at run-time. In particular, how to conduct the change of control policies is a critical issue. For example, what would happen if the auction rules are changed in the middle of an auction?
- How do the control agents handle the handoff process? One control agent may want to delegate its right of accessing some data sources to another agent before it leaves the system. This handoff should be smooth and seamless. When an auctioneer is changed by another, the auction should process without interruption and the bidders (users) should have no knowledge and effects of this change.

2.4 Consequences

There are several benefits of event-based blackboard architecture:

Implementing multi-agent systems using event-based blackboard architecture promotes the separation-of-concerns principle. Control policies are not only separated from control agents, but also separated from system data and logic. This separation is especially important since it allows the control policies to be modified independently from the control agents. Tangling them together makes the system control complex and difficult to implement.

Each agent needs to register into the system as a control or knowledge source. This registration allows the multi-agent systems to verify the identity of the agent and restrict it to a specific right in the systems. Thus, the systems are able to prevent intruders from accessing sensitive system data. In addition, each agent may be allowed to execute particular level of operations. Therefore, security mechanisms can be incorporated into the system.

Since multiple control agents may work in parallel, the bottleneck of control processing may be eased. Concurrency processing allows different threads of executions of the systems. The multi-agent systems are more dependable since the failure of a single control agent is not likely to cause the failure of the whole systems.

Event-based blackboard architecture also allows dynamic reconfiguration of system agents. Both knowledge source agents and control agents are able to join and leave the systems on-the-fly. They are not directly tied with each other, making it more reusable and maintainable.

Nevertheless, there are some liabilities of using the event-based blackboard architecture. Event-based implementation may cause some performance overhead due to the cost of event handling, which may slow down the system. In traditional blackboard systems, all communications are direct access. In addition, event-based systems are hard to implement and debug. Due to non-deterministic, it is hard to control the order of processing, especially in concurrent environment.

2.5 Known Uses

Event-based blackboard architecture has been applied in several applications. For example, the HASP system for interpreting sonar signals was an early system based on the event-based blackboard architecture [14]. In this system, HASP makes predefined events to be the primary basis of control. Several event types, such as clock and expectation events, are defined and a sequence of knowledge sources is executed for each event type. In effect, the predefined blackboard event types serve as the preconditions of knowledge source in HASP.

Corkill [5] also described a system based on event-based blackboard architecture. An event is generated when the state of the blackboard is changed. Some knowledge sources may respond to external events. Instead of having the knowledge sources scan the blackboard, each knowledge source informs the blackboard system about the kind of events in which it is interested. Whenever the kind of events occurs, the blackboard system notifies the corresponding knowledge source.

T Spaces [19] is an application of event-based blackboard architecture. It is a middleware component which provides group communication and event notification services. Its tuple spaces (blackboard) hold data to be exchanged. A set of APIs implements the function of control policies by means of implicit invocation. The data producer and consumer (knowledge sources) do not know each other. Thus, they can be attached to (detached from) the system easily by registering (removing) interests in events.

JavaSpaces [20] is designed to provide a simple unified mechanism for dynamic communication, coordination, and data- and object-sharing. It also implements the event-based blackboard architecture. Data or objects are stored in the Spaces, shared by providers and requesters in the way of implicit invocation.

3. Related Work

A reflective blackboard architectural pattern is presented in [16]. In this pattern, system architecture is divided into base level and meta-level. The control component is implemented in the system's meta-level by a separate component, called Meta-Object Protocol. Control strategies and rules can be specified at the meta-level, which is separated from the data and logic of multi-agent systems.

Agent design patterns have been presented in [1], which include three classes: Itinerary, Forwarding, and Ticket. Each class consists of several specific design patterns suitable for particular contexts. Basically, their framework and representations are an extension of design patterns [10] into the agent systems. There is no consideration on the compositions of patterns.

In addition to discovering agent design patterns, a classification of agent patterns has also been proposed based on different views of multi-agent systems in [13]. A pattern description template is presented to facilitate the documentation and discovery of agent design patterns.

The composition of design patterns [10] has also been investigated in [12][2]. These patterns can be applied in object-oriented framework [9]. Riehle [15] proposed an analysis method for the composition of design patterns. Role diagrams were introduced to describe the patterns, and a role relation matrix was used to visually depict the composition constraints. Formal engineering approaches have been presented to reason about the consistencies of the composition in [7][8].

4. Conclusion

In this paper, we introduce the event-based blackboard architectural pattern in multi-agent systems. This pattern separates control policies from the control agents so that they can vary independently. It is a composition of the Blackboard architectural pattern and the Implicit Invocation pattern. In addition to the description of the pattern, the implementation, consequences and known uses are discussed.

With the increasing scale and complexity of agent-based system architecture, single pattern is more likely insufficient for large system development. More and more architectural patterns and their compositions become important and critical solutions to recurring problems in complex agent-based systems. These architectural patterns and their compositions are important engineering techniques and methodologies to cope with the complexity of agent-based systems.

References

[1] Yariv Aridor and Danny B. Lange. Agent Design Patterns: Elements of Agent Application Design. *Proceedings of the International Conference on Autonomous Agents*, ACM Press, 1998.

[2] Paulo Alencar, Donald Cowan, Jing Dong, and Carlos Lucena, A Pattern-Based Approach to Structural Design Composition. *Proceedings of the IEEE 23rd Annual International Computer Software & Applications Conference (COMPSAC), pp160-165, Phoenix, USA*, October 1999.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal. Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.

[5] Daniel D. Corkill. Blackboard Systems. *AI Expert*. 6(9):40-47. September 1991.

[6] Donald D. Cowan and Carlos J.P. Lucena. Abstract Data Views: An Interface Specification Concept to Enhance Design. *IEEE Transactions on Software Engineering*, 21(3):229–243, March 1995.

[7] Jing Dong, Paulo Alencar, and Donald Cowan, A Behavioral Analysis and Verification Approach to Pattern-Based Design Composition. *International Journal of Software and Systems Modeling, Springer-Verlag*, Volume 3, Number 4, December 2004, Pages 262-272.

[8] Jing Dong, Paulo Alencar, and Donald Cowan. A Formal Framework for Design Component Contracts. *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI)*, pp53-60, Las Vegas, US, October 2003.

[9] M. Fontoura, W. Pree, and B. Rumpe. UML-F: A Modeling Language for Object-Oriented Frameworks. *Proceedings of the European Conference on Object-Oriented Programming*, pages 63–82, July 2000.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, 1995.

[11] Alan Garvey and Victor Lesser. Design-to-time Real-Time Scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*. 1993.

[12] Rudolf K. Keller and Reinhard Schauer. Design Components: Towards Software Composition at the Design Level. *Proceedings of the 20th International Conference on Software Engineering*, pages 302–311, 1998.

[13] Jurgen Lind. Patterns in Agent-Oriented Software Engineering. *Proceedings of the International Workshop on Agent-Oriented Software Engineering*, 2002.

[14] H. Penny Nii, Blackboard Model of Problem Solving. *The AI Magazine*, Vol. 7, No. 2, 38-53, 1986.

[15] D. Riehle. Composite Design Patterns. *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA)*, USA, pages 218–228, October 1997.

[16] Otavio Silva, Alessandro Garcia, and Carlos Lucena. The Reflective Blackboard Architectural Pattern for Developing Large Scale Multi-Agent Systems. *Proceedings of the First International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*. pages 73-93, May 2002.

[17] Mary Shaw and David Garlan. Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall. 1995.

[18] John Vlissides. Composite Design Pattern (They Aren't What You Think). C++ Report, June 1998.

[19] Peter Wyckoff, Stephen McLaughry, Tobin Lehamn, and Daniel Ford. T Space. *IBM System Journal*, Vol. 37, No. 3, pp 454-474, August 1998.

[20] JavaSpace. http://www.javaspaces.homestead.com/