A Catalog of Agent Coordination Patterns

Sandra C. Hayden, Christina Carrick, Qiang Yang

(shayden)(ccarrick)(qyang)@cs.sfu.ca
www.cs.sfu.ca/~isa
Simon Fraser University
Burnaby, BC
V5A 1S6

Abstract

This paper surveys the current state of the art in agent-oriented software engineering, focusing on the area of coordinated multi-agent systems. In multi-agent systems, the interactions between the agents are crucial in determining the effectiveness of the system. Hence the adoption of an appropriate coordination mechanism is pivotal in the design of multi-agent system architectures.

This paper does not focus on agent theory, rather on the development of an agent-oriented software engineering methodology, collaboration architectures and design patterns for collaboration. A catalog of coordination patterns inherent in multi-agent architectures is presented. Such patterns may be utilized in the architectural design stage of an agent-oriented software engineering methodology.

1 Introduction

The architect Christopher Alexander developed the notion of design patterns as a fresh approach to designing buildings. [DH] explores how patterns recognized during the design stage enable re-use of architectural frameworks in model-based software engineering (MBSE). Design patterns for coordination, coordination patterns, are a recently emerging concept. An appropriate coordination pattern must be selected to satisfy the interactive behaviors required of the system. This requires the establishment of a comprehensive catalog of agent coordination patterns, which does not exist at this time.

Currently it appears that most patterns are specified at a low level of design abstraction. [CS95] has identified a number of design patterns for concurrent, parallel, and distributed systems. The scope of this work is unfortunately limited - it is defined at too detailed a level, in terms of operating system facilities. This situation will hopefully improve as successive visitations remove irrelevant detail from the design core. Buschmann et al recognize the significance of the level of abstraction and helpfully categorize patterns into architectural patterns, design patterns, and implementation patterns or idioms [BMR+96]. Many existing patterns can be incorporated into a multi-agent system architecture, such as Gamma et al.s' State and Decorator [GHJV95]. However, some common previously developed patterns are not applicable to agent systems. For instance, the client-server pattern executes a program on the server from the client. With agent mobility now possible, this approach is no longer relevant or necessary in agent systems.

We propose a number of coordination patterns, grouped into four basic architectural styles: hierarchical, federated, peer-to-peer, and agent-pair patterns. The agent-pair patterns describe one-on-one interaction. The other architectures exhibit increasing degree of freedom of the agents, with agents in a hierarchical system having the least freedom and peer-to-peer agents the most freedom. In peer-to-peer architectures, individual agents are responsible for managing coordination and potential conflicts with others (e.g. intentional systems). In federated architectures, an umbrella system provides overall coordination that the agents submit to. In hierarchical structures, top-down control is imposed by agents in a supervisory or managerial role. The patterns discuss the roles of the agents and the manner in which they affect other agents in the pattern. Due to space restrictions we present here only a single pattern, with our other patterns following the same format.[1]

Autonomous Agents '99 Seattle WA USA

2 A Sample Pattern

Pattern Name: Broker
Intent: The broker allows decoupling of the client and service-provider by accepting requests from a client, farming out the work to a willing and available service-provider, and returning results to the client. This allows for communication and location transparency for interoperating applications.
Motivation: In multi-agent systems involving numerous agents with a range of capabilities, it is not feasible for each client to hold capability models for each service-provider. This would result in a complete graph if agents held both roles of client and service-provider, and complex graphs for lesser cases, with consequent messaging overhead in maintaining the capability models. It is more efficient for a broker or a number of brokers to serve as go-betweens or matchmakers, maintaining capability models of service-providers and connecting clients to service-providers providing their needs.
Applicability: If many clients and many service-providers exist in a particular application, this pattern is applicable. If

[1] The pattern template used here is adopted from [GHJV95].

distributed heterogeneous components are to be integrated, but their independence is to be maintained, this pattern is applicable. Distributed, heterogeneous components may not have been designed to interoperate: they may be dispersed over multiple platforms and may be implemented in different languages.

Structure: See Figure 1

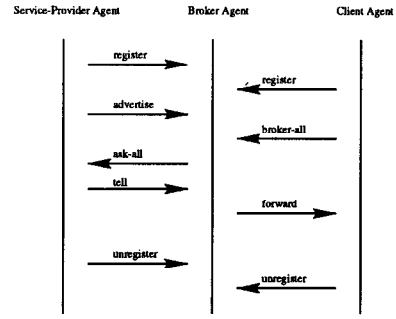

Figure 1: Broker Interaction Diagram

Participants: This coordination pattern involves at least one broker, a number of clients/requesters and providers of services. The broker's role is that of arbiter and intermediary, accessing services of one agent to satisfy the requests of another. The clients may also be service providers, and the service providers may also be clients. The roles of the agents are established in the context of a particular conversation.

Collaborations: See the interaction diagram.

Consequences: The major benefit of this pattern is a reduction in messaging overhead for systems with many clients and service-providers, since it is more efficient to maintain their information at a central location for matching purposes, the broker, rather than each client having to maintain a list of potential service providers.

A limitation which may arise is that the broker may become a bottleneck if too many clients and service-providers need to access it. Also, a lone broker provides a single point of failure. These concerns may be mitigated by using a number of brokers.

Implementation: In order to support its role, the broker requires a request handler, a registration mechanism for service-providers, a matching facility to find providers to satisfy a request, and a mechanism to map results back to requesters. Data encoding and decoding mechanisms are required to bridge heterogeneous application domains with different languages or platforms.

The KQML performative *broker-all* can be issued by a client to the broker. The broker will use all possible service providers to meet the request. Service providers should register their capability with the broker via the *advertise* performative. If only one service provider needs to be accessed in order to obtain acceptable service, the *broker-one* performative may be used. If the client is interested in the name(s) of all relevant service providers, the *recommend-one* or *recommend-all* performatives may be used.

Known Uses: OAA [CCWB94], KAoS [BDBW97], InfoSleuth [NU97]. The Object Management Group's (OMG) Object Request Broker (ORB) and remote procedure call (RPC) which provides location transparency are examples of a broker architecture, although they are not agent systems.

Related Patterns: Coplien and Schmidt's Broker pattern [CS95].

3 Conclusion and Future Work

Currently, the Broker pattern (along with the related patterns for Facilitators, Mediators and Matchmakers) appears to be the most frequently-used coordination mechanism. More patterns remain to be defined, in particular the many possible variations on peer-to-peer coordination such as the Adverserial and Altruistic patterns. Another pattern which remains to be specified is the Failure Recovery Pattern. A future improvement to the patterns presented here would be the inclusion of Java source code as sample implementations.

There is much work still to be done in the area of software agent engineering, particularly in the area of defining coordination models and methodologies for their implementation. The efficient utilization of established patterns of cooperation in the design of a multi-agent system has the potential for significant benefit to the architectural process and for successful implementation of multi-agent systems.

References

[BDBW97] J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley. KAoS: Toward an industrial-strength generic agent architecture. In Jeffrey Bradshaw, editor, *Software Agents*. AAAI Press / The MIT Press, Menlo Park, CA, 1997.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons, 1996.

[CCWB94] Philip R. Cohen, Adam Cheyer, Michelle Wang, and Soon Cheol Baeg. An open agent architecture. In *Proceedings of the AAAI Spring Symposium Series on Software Agents*, pages 1–8, Menlo Park, California, March 1994. American Association for Artificial Intelligence.

[CS95] J. Coplien and D. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[DH] Jorge L. Daz-Herrera. Integrating architectures, frameworks and patterns: a model-based approach. URL: www.sei.cmu.edu/activities/plp/oopsla/jorge.htm.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *Proceedings of the ACM SIGMOD Conference*, May 1997.

[KAH94] Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating agents for information retrieval. In *Proceedings of the 2nd International Conference on Cooperative Information Systems*, Toronto, Canada, 1994. University of Toronto Press.

[MOMC97] David Martin, Hiroki Oohama, Douglas Moran, and Adam Cheyer. Information brokering in an agent architecture. In *Proceedings of the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*, London, England, April 1997.

[NU97] M. Nodine and A. Unruh. Facilitating open communication in agent systems: the InfoSleuth infrastructure. In *Agent Theories, Architectures and Languages: Proceedings of the 4th International Workshop*, pages 281–295, Providence, Rhode Island, July 1997.

[PMF92] Jon A. Pastor, Donal P. McKay, and Timothy W. Finin. View-concepts: Knowledge-based access to databases. In *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, Maryland, 1992.

[SZ96] Katia Sycara and Dajun Zeng. Multi-agent integration of information gathering and decision support. In *Proceedings of the 12th European Conference on Artificial Intelligence*. John Wiley & Sons, Ltd., 1996.