THE LAYERED AGENT PATTERN LANGUAGE

Elizabeth A. Kendall, Margaret T. Malkoun, C. Harvey Jiang
Computer Systems Engineering, Royal Melbourne Institute of Technology
City Campus, GPO Box 2476V, Melbourne, VIC 3001 AUSTRALIA
email address: kendall@rmit.edu.au

1.0 BACKGROUND AND INTRODUCTION

A weak agent [31] is i) autonomous ii) social iii) reactive and iv) pro- active, and a strong agent, in addition to the above, has one or more of the following: v) mentalistic notions (beliefs, goals, plans, and intentions) vi) rationality vii) veracity and viii) adaptability. An agent can migrate[29] , and a multiagent system is open[1]. Agents appear in a wide range of applications, including personalized user interfaces[14] ; enterprise integration[18] ; manufacturing[12]; network management [26]; and business process support[8]. It is the combination of autonomous, social, reactive, and pro- active behavior that distinguishes an agent. Weak agents are assistants or drones[31, 14].

The following examples [27, 31] illustrate the kinds of problems that agents address.

- *The air- traffic control systems in the country of ABC suddenly fail, due to weather conditions. Fortunately, agent- based air- traffic control systems in neighboring countries negotiate between themselves to deal with affected flights, and the potentially disastrous situation passes without incident.* [31]
- *Upon logging into your computer, you are presented with a list of news group items, sorted into order of importance by your agent- based personal digital assistant (PDA). The assistant draws your attention to one article, which describes new work in your area. After discussion with other PDAs, yours obtains a relevant technical report for you from an FTP site. When a paper you have submitted to an important conference is accepted, your PDA makes travel arrangements by consulting a number of networked information sources.* [31]
- *The new home robot developed by Company XYZ is engineered by a team of agent designers. Five disciplines --- marketing, mechanics, electronics, computers, and manufacturing --- are represented on the team, and the agents work together, negotiating with each other to develop a sound, concurrently engineered product.*

An agent is an object that is able to reason and to collaborate via structured messages[9, 10]. Agents are active objects, in that they have their own thread of control. An agent system aims to be flexible and adaptive by carrying out opportunistic problem solving through reasoning, plans, and negotiation.

2.0 THE LAYERED AGENT PATTERN

2. 1 The Problem and Its Context

Agent- based systems arise out of the following needs:

- *Adaptable, fault tolerant distributed systems that solve complex problems*
- *Open systems where components come and go and new components are continually added.*
- *Migration and load balancing across platforms, throughout a network.*
- *Software modules that can easily adapt to a user.*
- *New metaphors, such as negotiation, for solving distributed, multi- disciplinary problems.*
- *Legacy systems and non- computer hardware that must be integrated into a unified system*

Strong agents reason about their models of the world and themselves to select a capability or plan that could achieve their stated goal(s). A plan from the capabilities library is instantiated by the interpreter when a triggering event occurs, and an instantiated plan is an intention. An intention executes in its own thread, and an agent may have several intentions executing concurrently. Agents negotiate with each other in conversations or protocols; languages (COOL[2] and AgenTalk[11]) support this. Agents coordinate their activities through market based economies[20], competitive bidding[11], and coalition formation[21]; the coalitions may have centralized or decentralized management.

These aspects of agent behavior are summarized in Figure 1[5, 8, 10]. The agent's reasoning is within its declarative knowledge or models, interpreter, and capabilities library. The agent's knowledge is affected by sensors that monitor external objects. A capability is chosen by the interpreter and manifested as an intention; three sample intentions are shown. One intention involves an effector object that impacts external objects. The other two intentions feature collaboration with other agents, either within the original society or external to it. If the other agents is in another society, the agents must migrate, either virtually or in reality, in order to collaborate.

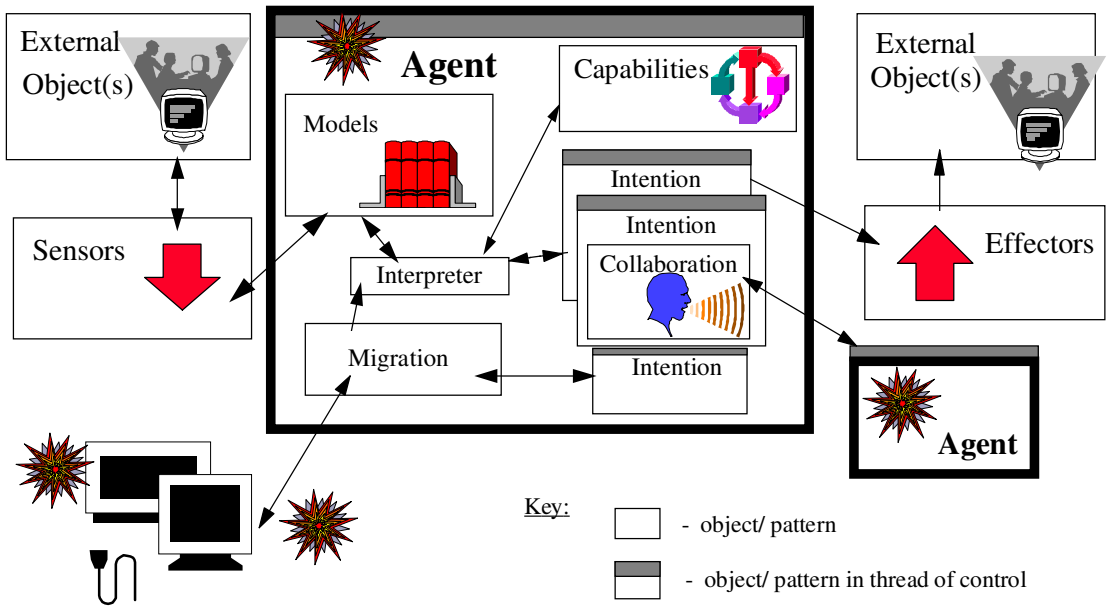

Figure 1: Aspects of Agent Behavior

2.2 The Solution

An agent is a large and complex system that spans several levels of abstraction. Agents should be decomposed into layers[25] because i) higher level or more sophisticated behavior depends on lower level capabilities, ii) layers only depend on their neighbors, and iii) there is two way information flow between neighboring layers. For example, an agent can only collaborate if it knows what its own goals and capabilities are and if it knows how to communicate with other agents. Additionally, collaboration, with or without migration, exhibits two way information flow, with an agent functioning as either a client or a server.

The layers can be identified from the model of the agent's real world that is shown in Fig. 1. Each layer must be a reflection of one of the real world domains, and the domains must be ordered on the basis of their purpose in the system[19]. Figure 2 structures Figure 1 into seven layers as follows:

1. *Sensory:* the agent gathers sensory input from its environment.
2. *Beliefs:* the agent's models of itself and its environment.
3. *Reasoning:* the agent determines what to do next, based on its goals, plans, capabilities, and models. The plans may generically specify cooperation with other agents in the form of requests for services or responses to requests.
4. *Action:* the agent's intentions (instantiated plans from the reasoning layer) are carried out. Pending actions for the agent are queued here, and these are derived from incoming requests that have been accepted by the collaboration layer.
5. *Collaboration:* the agent determines how to collaborate with another agent. This includes accepting or rejecting an incoming request for services. This layer addresses coordination protocols, including competitive bidding and coalition formation.
6. *Communication:* the agent formulates a message for another agent, translating it into another language or semantics (ontology), and performing any routing that is necessary
7. *Mobility:* a message is delivered to this agent or to another agent. The agents may be in the same society or in different ones.

In Figure 2, top down progress and information flow through the layers is shown on the left hand side of the figure, while bottom- up is on the right side. To summarize the bottom- up transactions, an agent's beliefs or models of itself and its environment are based on sensory input. When presented with a problem or a situation, an agent reasons about its beliefs to determine what to do. When the agent decides on an action, it can carry it out directly if it is within its own capabilities, such as sorting news items, but an action that involves other agents, such as concurrent engineering, requires the collaboration layer. Within this layer, the agent decides how to negotiate with another agent. Once the approach to collaboration is determined, the actual message is formulated at the communication layer (for example, the agents may speak different languages or come from different disciplines). The messages are delivered by the mobility layer.

Top- down, any messages the agent receives come in via layer 7, mobility. The incoming message is translated into the agent's dialect and semantics via the communication layer. The collaboration layer stores contacts and conversations, and determines whether or not the agent should acknowledge and process a message. If the received message should not be processed, this reply is sent out via the communication and mobility layers. If the message should be processed, it is passed on to the actions layer, where all pending actions are stored. When the action is selected for processing, it is passed to the reasoning layer. For example, the action may be a request from another agent for some information in the resident agent's beliefs. In this case, the action is processed within the reasoning layer, on the basis of a response from the beliefs in layer 2. The response is passed back to the actions layer. The lowest layer is layer 1, sensory, where the agent accumulates sensory information that updates its beliefs; the agent may also call on its sensors in response to an incoming request.

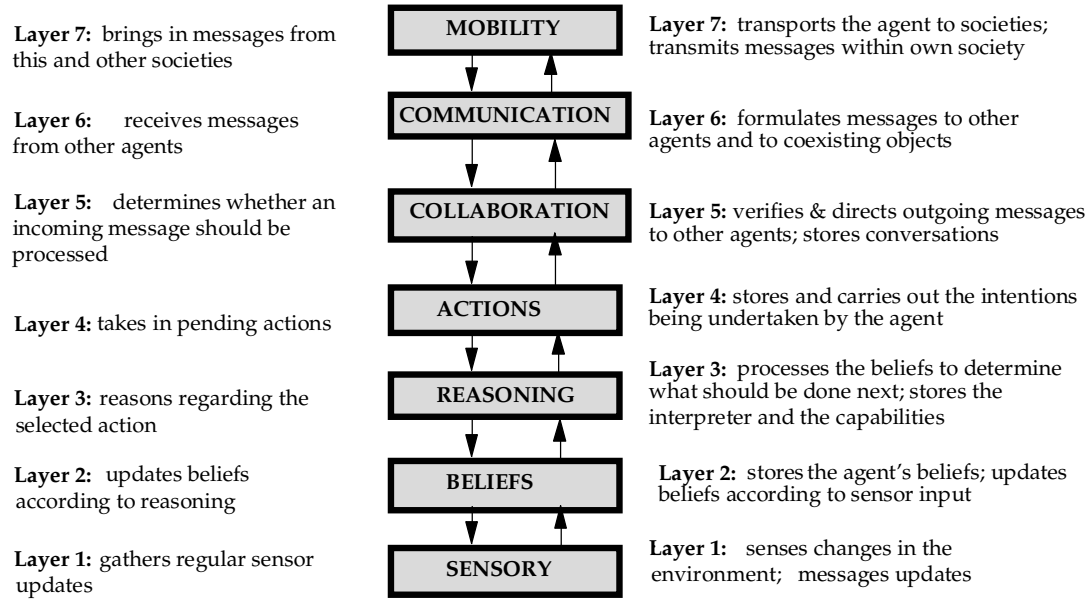

Figure 2: The Layered Agent Pattern

Once a plan is instantiated into an intention and placed in the actions layer, it does not require the services of any lower layers. (An intention executes with the beliefs or models that the agent had when it was instantiated.) An intention does not get any new tasks to perform from higher layers, but it does call on the services of collaboration, communication, and mobility when it needs to involve other agents in its tasks. Intentions request these services synchronously, awaiting the replies.

3.0 PATTERNS INTERNAL TO THE LAYERS

3.1 Synchronization and Concurrency: The Active Object Pattern

The Problem

Agents, intentions, and coexisting objects act in different threads of control and access shared resources. Agents may themselves be shared resources. All of these factors lead to problems in synchronization and concurrency.

The Solution

Figure 3 is a Booch [3] object diagram based upon the active object pattern[13] and the ACE concurrency encapsulation pattern[22] that address synchronization and concurrency. Four objects appear in the active object pattern: the Client Interface, the Scheduler, the Resource, and the Activation Queue. The Client Interface provides the interface to all clients. The invocation of a method triggers the construction of a Method Object which is then enqueued on the Activation Queue by the Scheduler. When the Resource is available, the Scheduler removes an activity from the queue and dispatches it to the Resource.

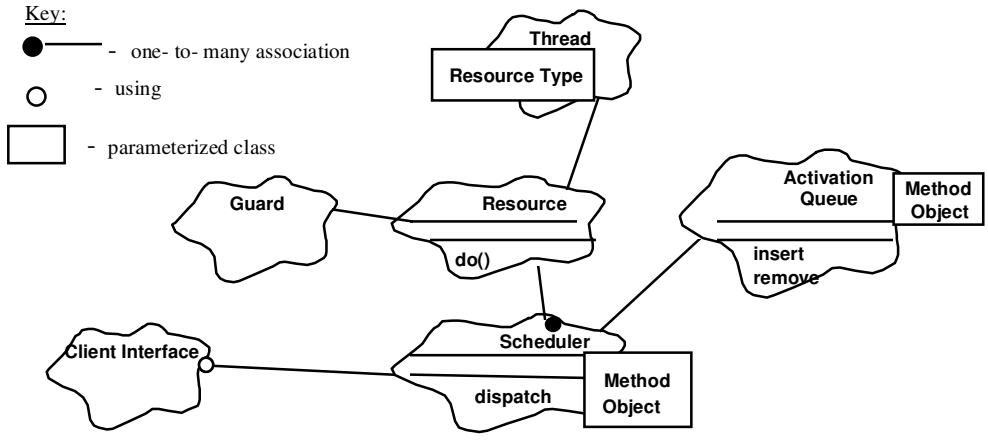

Figure 3: The Active Object Pattern with Thread and Guard Extensions

The additional two objects in Figure 3 are the Thread and the Guard. The Thread encapsulates operating system behavior for thread creation, termination, and management. Thread is a parameterized class, by Resource Type, and the Resource's do() member function is executed in the Thread. The Guard is responsible for managing exclusive access to the resource, when state must be preserved throughout an interaction. The Client Interface is responsible for non- exclusive synchronous interactions. In the active object pattern, the Client Interface and the Guard decouple the synchronization behavior from the active object (Resource) itself. This can be used to remove the inheritance anomaly [15, 22]. If continuous concurrent behavior is required, the Resource may have one or more private threads [7].

3.2 Interaction and Centralized Collaboration: The Mediator, Sensor and Adaptor Patterns

The Problem and Its Context

A sensor must recognize the interface of external objects. An effector must isolate an agent from platform or domain details. Agent collaboration is done via structured messages, such as KQML, COOL, or AgenTalk. Agent collaboration and interaction with objects give rise to similar design problems. Proliferating interconnections increase complexity, complicate maintenance, and reduce reusability.

The Solution

The Mediator[6] pattern is made up of a Colleague class category and two Mediator classes: an abstract Mediator that defines an interface for communicating with colleagues, and a concrete class that implements the behavior required to coordinate the colleagues. The Mediator pattern is a solution for centralized collaboration management.

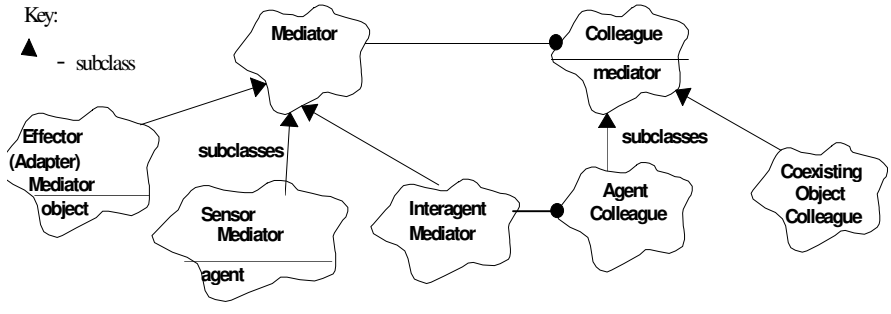

Figure 4: A Metalevel View of the Mediator, Sensor, and Adapter Patterns in Agent Based Systems

The Adapter pattern[6] is applicable to the Effectors in Figure 1 that pass the agent's actions on to coexisting objects. The Adapter pattern translates incoming requests into a form that is understood by a domain object. The agent becomes the client who sends messages to an Adapter. The Adapter translates the messages and passes them on to the Adaptee. A Sensor can be thought of as another form of an Adapter, addressing the fact that the agents follow structured messaging while the objects do not. The roles of the Mediator, Adapter, and Sensor patterns in agent systems are summarized in Figure 4. (Concrete mediators have direct links with all the concrete colleagues served, but not all are shown.)

3.3 A Decentralized Collaboration Pattern

The Problem and Its Context

If communication bottlenecks occur or if an agent does not have a mediator to represent it, decentralized collaboration can be utilized. For this, each agent must support different interfaces and maintain knowledge of every agent it wants to interact with.

The Solution

A Proxy[6] controls access to the Real Subject; it can also provide a distinct interface. The KQML Proxy would support the KQML interface, while the COOL Proxy and the AgenTalk Proxy subscribe to the other interfaces. In addition to intercepting messages, the Proxy provides a layer of indirection and is responsible for determining if the message should be passed on to the agent (the Real Subject).

For collaboration via coordination protocols, the agent must be able to determine its behavior based upon the state of the protocol[11] that it is engaged in. One agent may be engaged in several conversations simultaneously, requiring that it be able to accomplish context switching. The Memento pattern [6] externalizes an object's internal state so that the object can be restored to this state later. Agent proxies that support languages which involve conversations (such as the contract net) must store and recover their state. With the Memento pattern, they can delegate this to a companion Memento object. The Memento class is parameterized by Protocol, which gives the possible states and transitions. The resulting pattern for decentralized agent collaboration is shown in Figure 5.

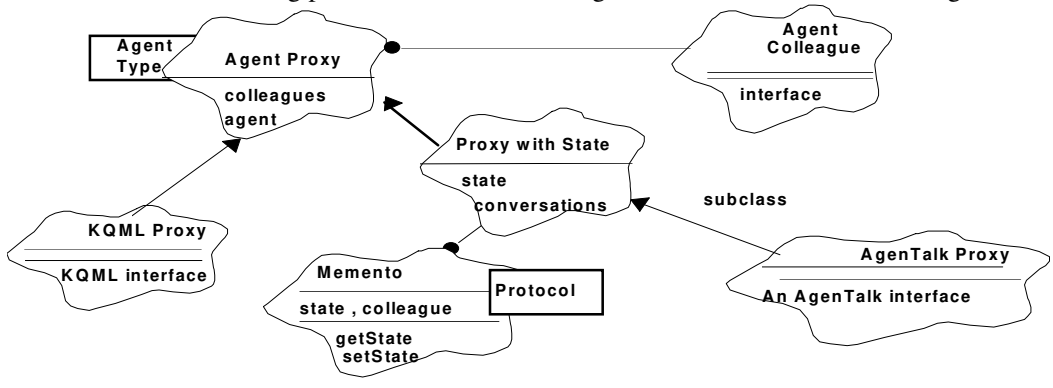

Figure 5: A Pattern for Decentralized Agent Collaboration Based on Proxies and Mementos

Companion memento objects are required for the coordination protocols that follow a conversation or a speech act finite state machine [11], and AgenTalk is shown in Figure 5 The Proxy is the originator of the Memento, and a Memento is instantiated when a conversation is undertaken with another agent in the given language. The class Proxy with State therefore stores all conversations (instances of Memento) in a data member. When a collaboration is suspended, the Memento instance's state is stored so that the conversation can be subsequently resumed.

3.4 Migration: The Broker Pattern

The Problem and Its Context

Agents must access each other and other resources across platforms. Making every agent responsible for access, security and interactions for a society, leads to N- to- N connections and redundancy.

The Solution

The Broker pattern[24] (Figure 6), which is an abstraction of Object Request Brokers[28], provides for communication and location transparency for objects that wish to be clients and servers of one another. With this pattern, an agent can be a client or a server to any registered agent. The agent (or its proxy) can become a virtual member of open societies managed by the brokers. Bridges between societies are also supported. Agents who wish to be clients and servers for one another must employ a broker who is responsible for locating a server once a client has requested its services. Both the client and the server must register with the broker. Also, the client and server employ proxies who respond to the interface known by the broker; the proxies may also be located in a different (the broker's) address space. The proxies also shield an agent from a request that should not be answered.

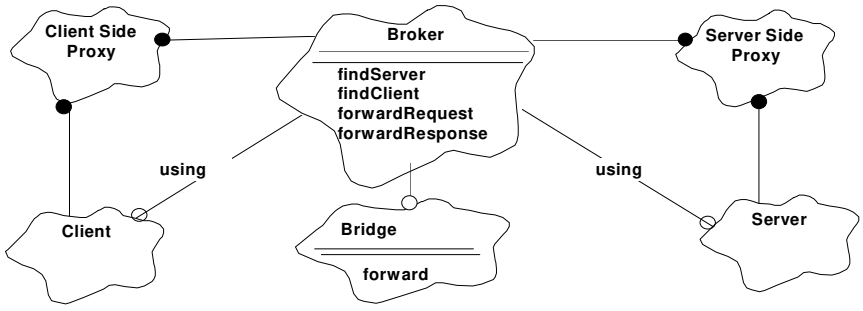

Figure 6: The Broker Pattern

3.5 Patterns for Automated Reasoning

The Problem and Its Context

Agents must formulate appropriate reactions or response to stimuli.

The Solution

Two patterns relevant to agent automated reasoning are the Interpreter and Strategy patterns[6]. The Interpreter pattern defines objects for the grammar of a language. Composite objects are used to represent the syntax and the statements to be interpreted, and an Iterator is used to traverse the structures. Interpreters relevant to agent oriented reasoning, such as one that interprets beliefs, goals, plans, context conditions and invocation conditions, could be designed via this pattern. Plans would be instantiated into intentions when the beliefs, context, and invocation conditions are satisfied. This provides the autonomous, reactive, and pro-active behavior that is central to the definition of an agent.

The Strategy pattern encapsulates or objectifies an algorithm or a strategy for solving a problem. With this approach, each plan in the plan library can be an object, and each plan has its own invocation and context conditions. The key to the pattern is that each stategy has the same interface; each plan and each intention can also have the same interface. Performance issues are crucial to agent automated reasoning, so the designs must be refined to address this.

4.0 THE LAYERED AGENT PATTERN LANGUAGE

Now the internal or component patterns can be placed in their respective layers (Figure 7, 8 and 9) to define a pattern language for agent systems.

4. 1 Agent Senses, Beliefs, Reasoning, and Actions

The sensory layer is made up of Sensors, while the beliefs layer has passive objects. As shown, the Sensors and the Effectors communicate with coexisting external objects. The reasoning layer's main object is the Interpreter, which processes the objects in the beliefs layer, along with the Strategy (Plan) objects, to determine which plans should be intended. The link between the Intention and the Strategy objects signifies that an intention is constructed from a plan. The mobility layer encapsulates the Broker pattern[24], with client and server proxies (for each agent or society), brokers (shared), and bridges (shared). Except for the agent's own proxies, the components in the mobility layer are shared with and interact with other agents.

The actions layer has multiple purposes. First, intentions reside in this layer; they are not full active objects, but they do have their own thread of control. Asynchronous tasks from the intentions are carried out by the Effectors, which translate the agent's messages into commands for coexisting objects. Synchronous requests (collaboration with other agents) are output directly from the intention to the collaboration layer. Lastly, the agent's own active object components reside in the actions layer to handle incoming requests. That is, the agent's Client Interface, Scheduler, Resource, Guard, and Activation Queue are found here. If the agent receives a synchronous request, the Guard denies any new requests until the Resource completes the service, and the reply is forwarded via the Client Interface. If the request is asynchronous, the reply can be made via an intention.

4. 2 Decentralized Collaboration for Agents

The collaboration layer routes outgoing messages to a collaborator and translates them into an appropriate coordination protocol; it also determines whether or not an incoming request should be processed. This involves the Proxies with State and Mementos (for decentralized collaboration), as shown in Figures 7 and 8. The upward links shown between Interpreter and Resource, Resource and

Client Interface, and Client Interface and Proxy with State are required for synchronous replies from the resident agent to another agent. The Resource object shown in Figure 8 would pass control to the Interpreter, which may in turn pass control to the Beliefs and Sensory layers if sensor updating was required

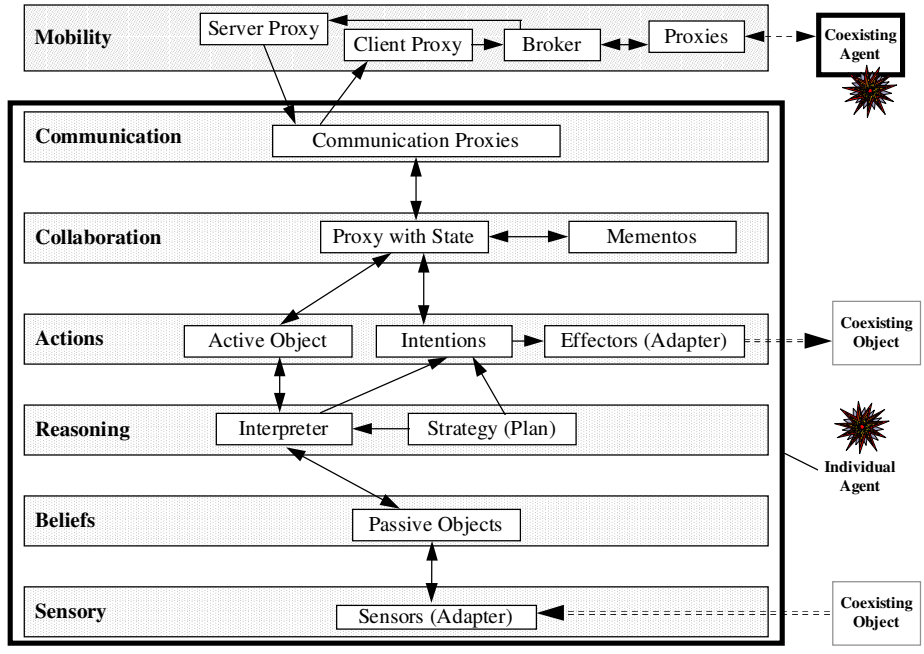

Figure 7: The Internal Layer Patterns and Their Interactions

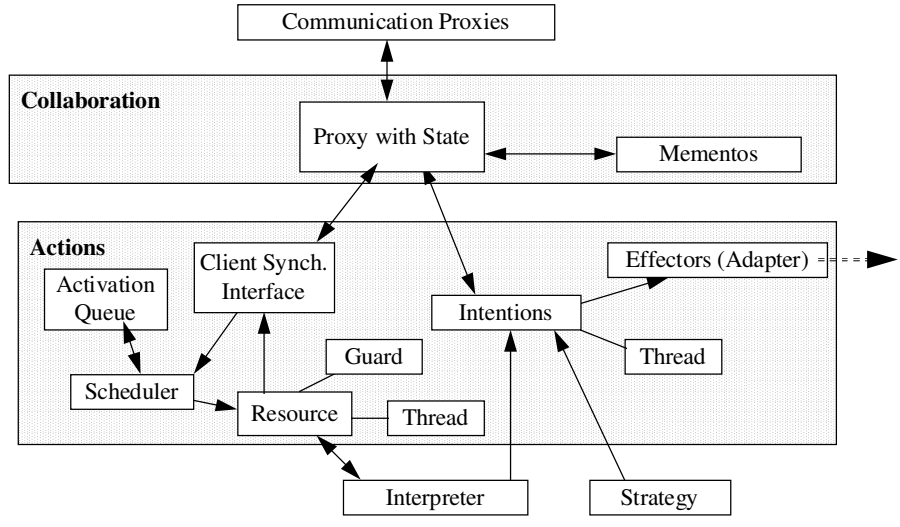

Figure 8: The Actions Layer and a Decentralized Collaboration Layer

4. 3 Centralized Collaboration for Agent Societies

The agent shares aspects of the mobility layer with other agents. The same is true for the collaboration and communication layers when centralized collaboration is appropriate. In this case, the collaboration layer is comprised of the Mediator(s) for a given agent society. The Mediator stores all contacts and conversations that are relevant to the aspect of the society that is under its guidance; it also has Communication Proxies which address translation and semantic information if it needs to communicate with another Mediator. The Mediator can deliver messages to agents within its own society. For messages that need to go outside of the society, the mobility layer is used. Centralized collaboration and communication are depicted in Figure 9. Two agent societies are shown, and they both have three agents (each with actions, reasoning, beliefs, and sensory layers) and centralized collaboration and communication layers. For cooperation across more than one society, the two societies share a mobility layer, with Client and Server proxies for the societies, along with the responsible Broker. (Any number of agent societies can share this mobility layer.)

The overall form or pattern language of the layered agent patterns can be seen in Figure 9. Agent societies are made up of any number of individuals, each with senses, beliefs, reasoning, and actions. With centralized organization, each society has mediators or managers who delegate tasks and handle collaboration between agents who are under their supervision. If necessary, the mediator calls on a communication layer for translation so that it can work with other mediators or managers within its society. Societies then join mobility or network layers so that they can exchange services with other societies.

The difference between centralized and decentralized collaboration is that an agent is totally responsible for representing itself and carrying out its own negotiation in the decentralized case. In actual applications, the agents may work on their own for some considerations and rely on a Mediator for others, and this would lead to some shared collaboration layers being mixed in with some decentralized ones. The pattern in Figure 9 also addresses agent coalition formation[20, 21] in that one or more agents may, during the course of their negotiation, decide to work together. They would then instantiate a Mediator and allow their collaboration layers to coalesce.

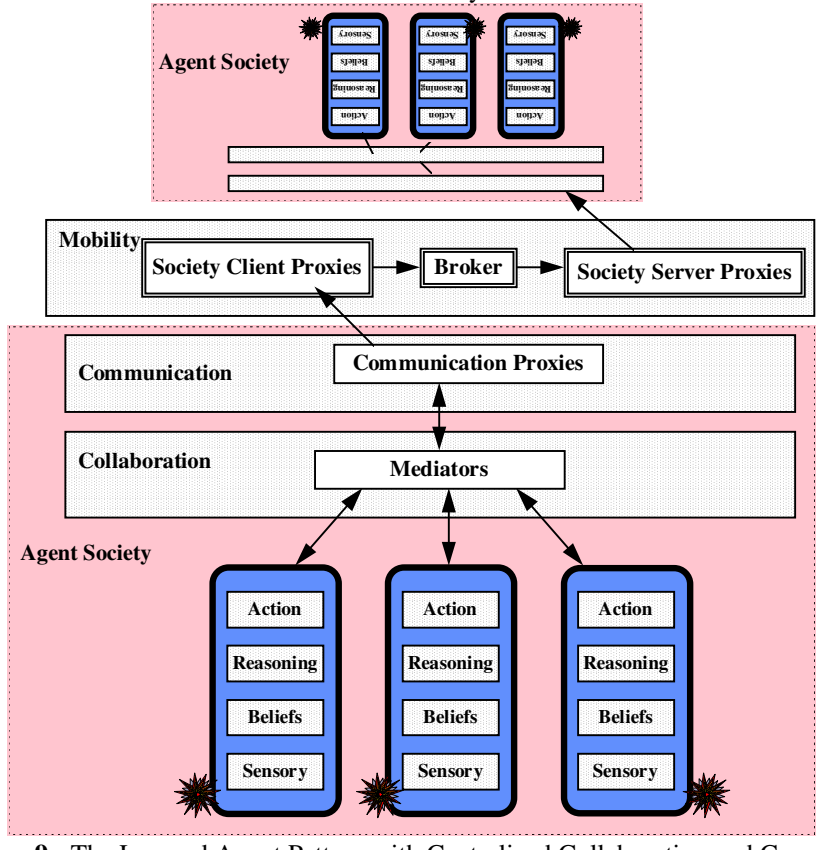

Figure 9: The Layered Agent Pattern with Centralized Collaboration and Communication

5.0 SUMMARY OF THE PATTERNS

5.1 Examples

The patterns address most existing agent- based systems. In terms of the patterns, a weak agent has a simplied reasoning layer: one that simply determines the appropriate action for a given stimulus or situation. Sensors and effectors are added when agents interact with coexisting applications. Some applications have relied on totally decentralized agent collaboration[8, 14], whereas others have utilized coalitions[20, 21] that are either static or dynamic. Coordination protocols are addressed by the proxies in decentralized collaboration or in the communication layer.

Robots have the lower level capabilities of agents. Several layered architectures for robots exist [4]; the layers in the robots comprise senses, models, reasoning, and actions.

The patterns described above provide a structure for modeling aspects of agents that reflect the behavior of humans and human societies. Individuals have senses, beliefs, reasoning, and actions. When presented with a problem or asked to accomplish a task, they reason about their models (usually not in a hierarchical fashion) to determine an action. When the humans interact with one another, they either do so directly or via mediators or managers. Once a collaborator has been identified and an approach to collaboration has been formulated (either internally or through a mediator), the

message must be conveyed or communicated in the appropriate language. Lastly, societies or organizations deal with each other via brokers, networks or other vehicles for exchange or mobility.

5.2 Benefits and Liabilities

The agent patterns described here have many of the benefits of any layered pattern[26]:

- *conceptual model* - the layers and their components address all of the key aspects of agency in a structured fashion
- *reusable building blocks* - individual layer implementations, for example sensors, can be replaced by other semantically equivalent ones
- *support for standardization* - for example, the CORBA standard can be used for the mobility layer; coordination and communication protocols can also be addressed by the layers
- *localized dependencies* - the synchronization behavior is localized to the Actions layer
- *changeability* - different collaboration layers can be brought in for centralized, decentralized, or hybrid collaboration. Each layer is only dependent on neighboring ones.

In addition, the patterns that are inside the layers have many demonstrated advantages. The active object pattern removes the inheritance anomaly. The broker pattern and the mediator pattern remove N-to-N connections and hence improve maintainability. Additionally, the pattern provided for decentralized collaboration delegates coordination and conversation storage, which enhances changeability and support for standardization within that pattern.

The major liability of these patterns would be performance, especially in the area of automated reasoning. An agent often needs to perform computationally intensive tasks that may not be best addressed by a layered architecture.

6.0 Ackowledgements

The authors wish to express their gratitude to Dr. Dirk Riehle for his expert advice as the shepherd for this paper. This project was partly funded by the Cooperative Research Centers for Intelligent Decision Systems and for Intelligent Manufacturing Systems and Technologies under the Australian Government's CRC Program.

7.0 References

1. Avouris, N. M., and L. Gasser, *Distributed Artificial Intelligence: Theory and Praxis*, Kluwer Academic Publishers, 1991.
2. Barbuceanu, M. and M. S. Fox, “COOL: A Language for Describing Coordination in Multi-Agent Systems,” First International Conference on Multi- Agent Systems, pp. 17 - 24, 1995.
3. Booch, G., “Object Oriented Analysis and Design with Applications, Second Edition”, The Benjamin/ Cummings Publishing Company, 1994.
4. Brooks, R.A., “A Robust Layered Control System for a Mobile Robot”, IEEE Journal of Robotics and Automation, Volume RA-2, Number 1, March 1986.
5. Farhoodi, F., I. Graham, “A Practical Approach to Designing and Building Intelligent Software Agents”, First International Conference on the Practical Application of Intelligent Agents and Multiagent Technology, London, 1996.
6. Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison- Wesley, 1994.
7. Gilbert, J. W., “A Software Pattern for the Implementation of Autonomic Object Behavior,” Workshop on Patterns for Concurrent, Parallel, and Distributed Object- Oriented Systems at OOPSLA ‘95, Austin, Texas, 1995.
8. Jennings, N. R., P. Faratin, M. Johnson, P. O’Brien, and M. Wiegand, “Using Intelligent Agents to Manage Business Processes”, First International Conference on the Practical Application of Intelligent Agents and Multiagent Technology, London, 1996.
9. Kendall, E. A., M. Malkoun, and C.H. Jiang, “A Methodology for Developing Agent Based Systems for Enterprise Integration”, EI’95, IFIP TC5 SIG Working Conference on Models and Methodologies for Enterprise Integration, Heron Island, Australia, November, 1995a.
10. Kendall, E. A., M. Malkoun, and C.H. Jiang, “A Methodology for Developing Agent Based Systems”, First Australian Workshop on Distributed Artificial Intelligence at the Eighth Australian Joint Conference on Artificial Intelligence (AI'95), Canberra, Australia, November, 1995b.

11. Kuwabara, K., T. Ishida, and N. Osato, “AgenTalk: Describing Multiagent Coordination Protocols with Inheritance,” submitted to Tools for Artificial Intelligence Conference, 1995.
12. Kwok, A., and D. Norrie, “Intelligent Agent Systems for Manufacturing Applications”, J. of Intelligent Manufacturing, Vol 4, pp. 285- 293, 1993.
13. Lavender, R. G., and D. C. Schmidt, “Active Object: an Object Behavioral Pattern for Concurrent Programming”, Pattern Languages of Programming Conference, Illinois, 1995.
14. Maes, P., “Agents that Reduce Work and Information Overload,” Comm. of the ACM, Vol 37, No. 7, 1994.
15. Matsuoka, S., and A. Yonezawa, "Analysis on Inheritance Anomaly in Object-Oriented Concurrent Programming Languages", in *Research Directions in Concurrent Object-Oriented Programming* (ed. Gull Agha, P. Wegner, and A. Yonezawa), The MIT Press, Cambridge, MA., 1993.
16. Meunier, R., “The Pipes and Filters Architecture,” in *Pattern Languages of Program Design*, ed by J. Coplien and D. Schmidt, Addison- Wesley, 1995.
17. Mularz, D., “Pattern- Based Integration Architectures,” in *Pattern Languages of Program Design*, ed by J. Coplien and D. Schmidt, Addison- Wesley, 1995.
18. Pan, J. Y. C., and J. M. Tenenbaum, “An Intelligent Agent Framework for Enterprise Integration,” IEEE Trans. on Systems, Man, and Cybernetics, Vol. 21, Nol. 6, November/ December, 1991.
19. Rubel, B., “Patterns for Generating a Layered Architecture”, in *Pattern Languages of Program Design*, ed by J. Coplien and D. Schmidt, Addison- Wesley, 1995.6
20. Sandholm, T. W., and V. R. Lesser, “Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework,” in First International Conference on Multiagent Systems (ICMAS’95), June, 1995.
21. Sandholm, T. W., and V. R. Lesser, “Coalition Formation among Bounded Rational Agents,” CMPSCI Technical Report 95- 71, University of Massachusetts at Amherst, September, 1995.
22. Schmidt, D. C., “The ACE Object- Oriented Encapsulation of Lightweight Concurrency Mechanisms”, 1995.
23. Schmidt, D. C., and S. Vinoski, “Object Interconnections: Comparing Alternative Client- Side Distributed Programming Techniques (Columns 3)”, SIGS C++ Report Magazine, May, 1995.
24. Siemens AG, *The Broker Pattern*, ftp site pub/patterns/siemens/broker.A4.ps, November, 1995.
25. Siemens AG, *The Layered Pattern*, ftp site pub/patterns/siemens/layers.A4.ps, May, 1996.
26. Somers, F., “Intelligent Agents for High- Speed Network Management, “ PAAM- 96, First International Conference on the Practical Application of Intelligent Agents and Multi- Agent Technology, London, 1996.
27. Tenenbaum, J. M., J. C. Weber, and T. R. Gruber, “Enterprise Integration: Lessons from SHADE and PACT”, in Enterprise Integration Modeling, Proceedings of the First International Conference, Etd. by C. J. Petrie, MIT Press, 1992.
28. Vinoski, S., “Distributed Object Computing with CORBA”, C++ Report, Vol. 5, July/ August 1993.
29. Wayner, P., *Agents Unleashed: A Public Domain Look at Agent Technology*, AP Professional, 1995.
30. Wolf, K., and C. Liu, “New Clients with Old Server: A Pattern Language for Client/ Server Frameworks,” in *Pattern Languages of Program Design*, ed by J. Coplien and D. Schmidt, Addison- Wesley, 1995.
31. Wooldridge, M. J., and N. R. Jennings, “Agent Theories, Architectures, and Languages”, Tutorial, First International Conference on Multi- Agent Systems, 1995.