Patterns of Intelligent and Mobile Agents

Elizabeth A.Kendall, P.V. Murali Krishna, Chirag V. Pathak, C.B. Suresh
Computer Systems Engineering, Royal Melbourne Institute Of Technology
City Campus, GPO Box 2476V, Melbourne, VIC 3001 AUSTRALIA
email : kendall@rmit.edu.au

1. ABSTRACT

Agent systems must have a strong foundation; one approach that has been successfully applied to other kinds of software is patterns. This paper presents a collection of patterns for agents.

2. MOTIVATION

Almost all agent development to date has been "home grown" [4] and done from scratch, independently, by each development team. This has led to the following problems:

- **Lack of an agreed definition:** Agents built by different teams have different capabilities.
- **Duplication of effort:** There has been little reuse of agent architectures, designs, or components.
- **Inability to satisfy industrial strength requirements:** Agents must integrate with existing software and computer infrastructure. They must also address security and scaling concerns.

Agents are complex and ambitious software systems that will be entrusted with critical applications. As such, agent based systems must be engineered with valid software engineering principles and not constructed in an ad hoc fashion.

Agent systems must have a strong foundation based on masterful software patterns. Software patterns arose out of Alexander's [2] work in architecture and urban planning. Many urban plans and architectures are grandiose and ill-fated. Overly ambitious agent based systems built in an ad hoc fashion risk the same fate. They may never be built, or, due to their fragile nature, they may be built and either never used or used once and then abandoned. A software pattern is a recurring problem and solution; it may address conceptual, architectural or design problems.

A pattern is described in a set format to ease its dissemination. The format states the problem addressed by the pattern and the forces acting on it. There is also a context that must be present for the pattern to be valid, a statement of the solution, and any known uses. The following sections summarize some key patterns of agent based systems; for brevity, many of the patterns are presented in an abbreviated "patlet" form. When known uses are not listed for an individual pattern, it means that the pattern has arisen from the JAFIMA activity. The patterns presented in this paper represent progress toward a pattern language or living methodology for intelligent and mobile agents.

3. CONTEXT

An agent [23] is i) autonomous - acts without human intervention, ii) social - collaborates with other agents via structured messages, iii) reactive - responds to environmental changes, and iv) pro- active - acts to achieve goals. It is the combination of these behaviors that distinguishes an agent from objects, actors, and robots. Agents review models of the world and themselves to select a capability or plan to address the present situation. Once invoked, each plan executes in its own thread, and several of these may execute concurrently. Agents negotiate with each other; agent collaboration across disciplines may require that semantics can be exchanged.

4. THE LAYERED AGENT PATTERN

4.1 Problem:

How can agent behavior be best organized and structured into software? What software architecture best supports the behavior of agents ?

4.2 Forces:

- An agent system is complex and spans several levels of abstraction.
- There are dependencies between neighboring levels, with two way information flow.
- The software architecture must encompass all aspects of agency.
- The architecture must be able to address simple and sophisticated agent behavior.

4.3 Solution:

Agents should be decomposed into layers [6] 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. The layers can be identified from the model of the agent's real world; Fig. 1 structures an agent into seven layers. The exact number of layers may vary.

Proceeding from the bottom up in Figure 1, an agent's beliefs are based on sensory input. When presented with a problem, an agent reasons to determine what to do. When the agent decides on an action, it can carry it out

directly, but an action that involves other agents requires collaboration. Once the approach to collaboration is determined, the actual message is formulated at translation and delivered to distant societies by mobility.

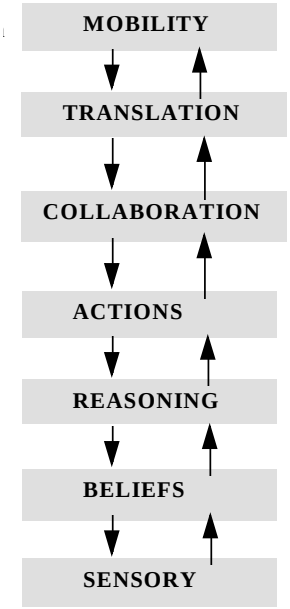

Figure 1: The Layered Agent Architectural Pattern

Top- down, distant messages arrive at mobility. An incoming message is translated into the agent's semantics. The collaboration layer determines whether or not the agent should process a message. If the message should be processed, it is passed on to actions. When an action is selected for processing, it is passed to the reasoning layer, if necessary. Once a plan placed in the actions layer, it does not require the services of any lower layers, but it utilize higher ones.

4.4 Variations:

The Layered Agent is a general architecture that addresses simple and sophisticated agents. All agents do not have the seven layers. A static agent isn't mobile, so layer 7 is not present in these agents. A translation layer is only needed for multidisciplinary agent societies (which are presently rare), and many agent societies have one collaboration layer for its agents to share. If an agent's beliefs are simple and do not change, the agent doesn't need sensors; mobility and collaboration may be combined if an agent society is dispersed over several platforms. Further, aspects of the Layered Agent pattern may be conceptual rather than architectural if, due to performance issues, direct connections between non-adjacent layers are required.

4.5 Known Uses:

There are many layered agent architectures; early ones did not require mobility or translation. GRATE [23] features domain, cooperation, and control layers, equivalent to sensory, beliefs, reasoning, action and collaboration. TouringMachines [9] consist of perception, action and control. InterRRaP [18] has four layers: cooperation, plan-based, behaviour-based and world interface.

5. SENSORY, BELIEFS, REASONING

5.1 Context and Overview

The Sensory and Beliefs layers maintain the agent's models of its environment and itself. Based on these models, the agent determines what to do next in Reasoning. There are four conceptual patterns that have evolved for these three lowest levels of agent capabilities; these patterns differ according to the presence or absence of three factors: a symbolic model, a knowledge based, prescriptive solution, and interaction with a human user. Table 1 summarizes these four patterns:

Problem	Symbolic	Prescribed	Human Int.	Solution	Pattern
How can an agent simply react to a stimulus or a request ?	-	-	-	Utilize a stimulus/response type of behavior.	Reactive Agent
How can an agent select a plan to achieve a goal ?	X	X	-	The agent reasons about a symbolic model to select a capability.	Deliberative Agent
How can an agent address problems when no solution is known beforehand ?	X	-	-	Represent the problem's constraints, and let the agent opportunistically solve the problem.	Opportunistic Agent
How can an agent adapt to the needs of a human user ?	-	-	X	Provide the agent with parametric user models and sensors to monitor the user.	Interface Agent

Table 1: Sensory, Beliefs, and Reasoning Patterns

5.2 The Reactive Agent

5.2.1 Problem

How can an agent react to an environmental stimulus or a request from another agent when there is no symbolic representation and no known solution ?

5.2.2 Forces

- An agent needs to be able to respond to a stimulus or a request.
- There may not be a symbolic representation for an application.

- An application may not have a knowledge based, prescriptive solution.

5.2.3 Solution

A Reactive Agent does not have any internal symbolic models of their environment; it acts using a stimulus/ response type of behavior. It gathers sensory input, but its Belief and Reasoning layers are reduced to a set of situated action rules. A single Reactive Agent is not proactive, but a society of these agents can exhibit such behavior. A Reactive Agent is known as a weak agent.

5.2.4 Known Uses

Reactive theory was originated by Brooks [5] and Agre and Chapman [1]; reactive agents have been widely used [21]. They have been used to simulate the behavior of ant societies and to utilize such societies for search and optimization [8].

5.3 The Deliberative Agent

5.3.1 Problem

How can an agent select a capability to proactively achieve a goal within a given problem context ?

5.3.2 Forces

- An agent should be capable of intelligent behavior, selecting a plan to achieve a goal.
- For some applications, a symbolic representation or model of the environment can be specified.
- Some problems have a knowledge based solution that can be identified by experts.

5.3.3 Solution

A Deliberative Agent possesses an internal symbolic reasoning model of their environment and themselves within their Beliefs and Reasoning layers. They select a plan or capability that can achieve their goal in the context of the present situation. A Deliberative Agent is a strong agent, and a sample use involves a society of agents with knowledge of particular business processes.

5.3.4 Known Uses

Deliberative Agents were originated by Cohen [7] and Georgeff [11], and they have been widely used by Jennings [14] and others [20].

5.4 The Opportunistic Agent

5.4.1 Problem

How can an agent opportunistically address problems, identifying an approach that is not known appriori ?

5.4.2 Forces

- A problem can have a symbolic representation but not have a knowledge based, prescriptive solution.
- For these applications, only constraints may be known; these indicate what can not be done.
- An agent needs to be able to avoid known constraints but still move toward a solution.

5.4.3 Solution

An Opportunistic Agent does not attempt to have prescriptive plans to address a problem. Rather, their Beliefs consist of constraints found in the problem, and their Reasoning or capabilities accomplish constraint propagation and satisfaction. Problems with a symbolic representation but with no known appriori, prescriptive solution can be solved this way.

5.4.4 Known Uses

Fox [19, 21] has pioneered this approach and used it successfully in distributed scheduling and resource allocation; these problems typically have no knowledge based approach.

5.5 The Interface Agent

5.5.1 Problem

How can an agent adapt to the needs of a human user ?

5.5.2 Forces

- Some agents work directly with a human user, assisting them in using an application or in finding information or services.
- The needs of human users are variable, but there are certain categories of users and established patterns of user behavior.

5.5.3 Solution

An Interface Agent collaborates with a human computer user. Typically, only one agent is found, although a full agent society may be used. This kind of agent observes the user and adapts to their needs by identifying what kind of user they are and their patterns of computer useage. An Interface Agent's beliefs are typically parametric user models, and their sensors monitor the user's actions.

5.5.4 Known Uses

Maes [17] has led the development of Interface Agents, also called Personal Assistants [20].

6. THE ACTION LAYER

6.1 Context and Overview

The Action layer carries out the selected plan. There is a need for the layer to be able to schedule and prioritize. It makes use of the patterns summarized in Table 2.

6.2 The Intention

6.2.1 Problem

How can an agent commit to performing reactive and proactive behavior ?

6.2.2 Forces

- Behavior executes with the beliefs that the agent had when it (the behavior) was initiated.
- An agent may have many activities or plans executing concurrently.
- An agent's plan impacts the environment through the effectors; it calls on collaboration when it needs to involve other agents.

Problem:	Solution:	Pattern:
How can an agent commit to behavior ?	An instantiated plan is an Intention that executes in its own thread of control.	Intention
How can a plan or request be encapsulated as an object ?	Implement a plan interface for a high level operation with different subclasses.	Plan as Command
How can different plans, intentions and	Provide an interface for creating families of related objects	Plan and

requests be instantiated at runtime ?	without specifying their concrete classes. Or, let subclasses determine which class to instantiate.	Intention Factory
How do you manage different threads of control for agent actions and migrations?	Decouple method execution from method invocation to simplify synchronized access to a shared resource by methods invoked in different threads of control.	Adaptable Active Object
How can priority handling and other forms of behavior be added to an intention dynamically ?	Decorate or add behavior to the run () method of the intention thread, where a plan executes.	Prioritizer
How can messages from other agents be passed to the agent's reasoning capability ?	Decouple reasoning execution from invocation to simplify synchronized access to the agent's shared reasoning resource.	Message Forwarder

Table 2: Patterns of the Action Layer

6.2.3 Solution

An Intention represents the commitment of an agent to being in a state where it believes it is about to actually perform a set of actions [7]. An instantiated plan is an Intention that executes in its own thread of control; it executes until completion, unless it is suspended awaiting a reply. A plan's goals are stated in invocation conditions; additional criteria, such as environmental situations or stimuli, are in context conditions. Conditions and plans reside in the Reasoning layer (Figure 2). If the conditions are satisfied, the plan is instantiated and executed by an Intention in the Actions layer. All variables and expressions in the plan are evaluated, based on the agent's beliefs, at the time of instantiation, when the agent commits to performing the plan. An Intention can be specialized to a CollaborationIntention and a ReactionIntention (Figures 4 and 5). Once an Intention is created, it does not require the services of any of the lower layers; collaboration can involve higher layers.

6.2.4 Known Uses

Intentions were first introduced by Georgeff and Lansky [11], as part of their Belief- Desires- Intentions agent architecture. Intentions provide the proactive and reactive behavior of many strong and weak agent systems, including [7] and [14].

6.3 The Prioritizer

6.3.1 Problem

How can priority handling and other forms of behavior be added to an intention dynamically ?

6.3.2 Forces

- There are two main Intention subclasses: Reaction and Collaboration. Additional refinement is needed, especially for priority handling.
- Further subclassification will result in duplication, as both Reaction and Collaboration Intentions can feature the same priority handling.
- Priority handling should be attached to an object, and not a class, because the type of IntentionThread is not known before run time.

6.3.3 Solution

Additional responsibilities can be attached to an Intention dynamically using Decorators [10]. The Prioritizer pattern can be used to decorate the run() method of the IntentionThread, where action plans are executed. Additional priority handling can be added dynamically to it by using the decorator object.

6.4 Adaptable Active Object

6.4.1 Problem

How do you manage different threads of control for agent actions ? How can the agent's actions conform to different environments ?

6.4.2 Forces

- Agent intentions act concurrently in different threads of control.

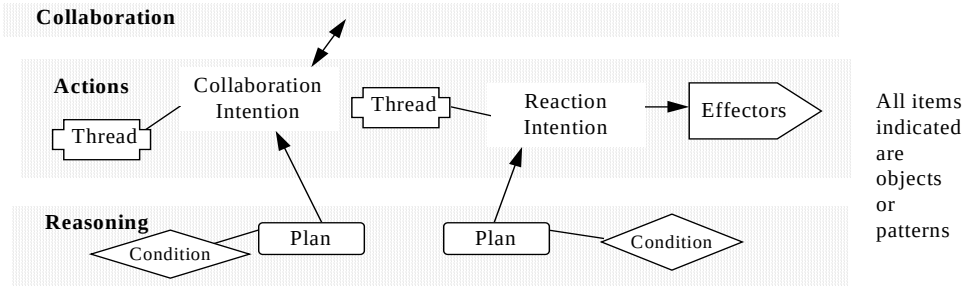

Figure 2: Intentions in the Action Layer, Plans and Conditions in the Reasoning Layer

- An object in the environment may need to be affected or impacted by the agent in a sequential manner.
- Agents may act in various environments, with different effectors.
- The Active Object pattern uses MethodObjects, but it is not practical to represent each method as a separate class and instantiate it at runtime because of the variability in effectors.

6.4.3 Solution

The Active Object pattern [17] decouples method exectuion from method invocation in order to simplify synchronised access to a shared resource. ClientInterface,

Scheduler and ActivationQueue form the Active Object pattern, along with Method Object; however, new Method Object classes would be necessary for each method in each environment. The solution to this problem is provided by the Adapter pattern [10] and the class ConcreteAdapter. The user has to provide the ConcreteAdapter which marshalls the method call when a method is invoked and later on demarshalls the method object when it is dispatched.

7. THE COLLABORATION LAYER

7.1 Context and Overview

In the Collaboration layer, the agent determines its approach to cooperating or working with other agents. Conceptual, architectural, and design patterns are utilized for messaging (Conversation), centralization (Facilitator), decentralization (Agent Proxy), and social policies (Protocol, Emergent Society); they are summarized in Table 3:

7.2 The Conversation

7.2.1 Problem

How can structured messaging between agents occur in sequences rather than in isolated acts ?

7.2.2 Forces

- Successive messages between agents may be related.
- Endless loops of messages need to be avoided.

7.2.3 Solution

A Conversation [4] is a sequence of messages between two agents, taking place over a period of time. There are termination conditions for any given occurrence, and Conversations may give rise to other Conversations. In some agent societies, messages between agents may occur only within the context of conversations; isolated messages are not supported.

7.2.4 Known Uses

AgenTalk [20] supports Conversations between agents, as does KAoS [4].

7.3 Centralized Collaboration: Facilitator

7.3.1 Problem

How is an agent able to freely collaborate with other agents without direct knowledge of their existence?

7.3.2 Forces

- Each agent may not have knowledge of every other agent
- Proliferating interconnections and dependencies increase complexity, complicate maintenance, and reduce reusability

7.3.3 Solution

Each Mediator [10] is associated with a multitude of Colleagues, objects that rely on it for all communication. The Facilitator is based on the Mediator, and it provides a gateway or clearinghouse for agent collaboration [5]. With a Facilitator, agents do not have to have direct knowledge of one another, and agents within the same society share a single Collaboration layer.

7.3.4 Known Uses

ARCHON [14], PACT [22], and other agent applications have utilized Facilitators, refering to this approach as a federated agent architecture [20].

7.4 Decentralized Collaboration: Proxy

7.4.1 Problem:

How can agents collaborate directly with one another?

7.4.2 Forces:

- An agent may not have a Facilitator to represent it. Then, each agent must communicate directly with other agents, support different interfaces, and maintain collaboration knowledge.
- Agents collaborate with each other via structured messages; there are many agent dialects.
- Bottlenecks encountered in a centralized architecture need to be avoided.
- An agent must be able to recover Conversations that it is involved in.

Problem:	**Solution:**	**Pattern:**
How can messaging between agents occur in sequences ?	Agent messaging can occur within a context established by previous messages.	Conversation
How can agents collaborate without direct knowledge of each other ?	Encapsulate agent interaction in a Facilitator that coordinates agents within a given society.	Facilitator
How can agents collaborate directly with one another ?	Provide a Proxy to control access to the agent and to provide distinct interfaces. Store and retrieve conversations.	Agent Proxy
How can agent collaboration be prescribed ?	Establish conversation policies that explicitly characterize communication sequences.	Protocol
How can agents cooperate to achieve goals when there is no established protocol ?	Though stimulus/response behavior, each agent can stimulate its neighbors. Complex patterns of behavior emerge when viewed globally.	Emergent Society

Table 3: Patterns of the Collaboration Layer

7.4.3 Solution:

A Proxy [10] controls access to the Real Subject; it can also provide a distinct interface. Each Agent Proxy class would subscribe to a certain interface. An agent must be able to determine its behavior based upon the state of the conversation it is involved in. One agent may be engaged in several conversations simultaneously, requiring context switching. The Memento pattern [10] externalizes an object's state so that the state can be restored later. Agent Proxies that support conversations must store and recover their state, delegating this to a Memento.

7.5 Protocol

7.5.1 Problem

How can agent collaborative behavior be prescribed to follow certain policies ?

7.5.2 Forces

- Agents need to be able to follow certain conventions or policies for collaboration.

7.5.3 Solution

Conversation policies [4] or Protocols prescriptively encode regularities that characterize communication sequences between users of a language. Agent Protocols explicitly define what sequences of which messages are permissible between a given set of participating agents.

7.5.4 Known Uses

KAoS [4] and AgenTalk [20] stipulate several protocols or conversation policies, including contract net, inform, offer, and request.

7.6 The Emergent Society

7.6.1 Problem

How can agents collaborate without known protocols ? How can Reactive Agents collaborate ?

7.6.2 Forces

- There may not be known agent protocols for a given application.
- Reactive Agents need to be able to collaborate and carry out proactive behavior together.
- Reactive Agents simply react to stimuli and are not capable of any knowledge based behavior.

7.6.3 Solution

Each individual agent, even a Reactive Agent, can, through their own actions, provide a stimulus to a neighboring agent. As each individual agent reacts to stimuli provided by their neighbors, the net result is the Emergent Society. Complex patterns of behavior can emerge from these interactions when the agent society is viewed globally [20]. No model exists for this behavior, although economic and game theory have been applied successfully. Reactive Agents and agents from Emergent Societies have reduced Collaboration layers; they merely provide stimuli to neighboring agents.

7.6.4 Known Uses

All Reactive Agent systems [8] rely on the Emergent Society for collaboration [20].

8. THE MOBILITY LAYER

8.1 Context and Overview

The Mobility layer must support real and virtual migration. It consists of a region shared across several agents and agent societies, and a region that belongs to an individual agent. It is made up of the architectural, conceptual, and design patterns in Table 4.

8.2 The Clone

8.2.1 Problem

How can an agent relocate itself and become resident in distant societies ?

8.2.2 Forces

- An agent must be able to bring its capabilities, facilities, and state with it to a new society.
- The agent must be able to travel to a remote location and interact, negotiate, and exchange information in the new society.

8.2.3 Solution

Make a copy or clone of the original agent, and place the new agent in the distant society. The clone must have all of the capabilities and facilities of the original agent, along with any state information.

8.2.4 Known Uses

The original use of agent self replication was cooperating mobile WAVE agents [3]. More recent approaches that utilize cloning include IBM Aglets [12], and the Agent Transfer Protocol (ATP) [13]. An aglet is a Java object that can move from one host on the Internet to another.

Problem:	**Solution:**	**Pattern:**
How can an agent become resident in a distant society ?	Replicate the agent, providing sensors and effectors for the new environment.	Clone
How can an agent be cloned in a distant society ?	Define operations for cloning in the destination society without changing the agent class. Separate the construction of the agent from its representation.	Remote Configurator
How is an agent able to gain access to resources and other agents outside its society transparently ?	Location transparency is provided by a Broker. Proxies must be employed for the client and server to be able to respond to the interface of the Broker.	Broker
How can an agent migrate virtually or in reality, dynamically ?	Provide a Thread Manager and a Handler Creator, and allow the subclasses for virtual and actual migration to address thread instantiation.	Migration Thread Factory

Table 4: Patterns of the Mobility Layer

When the aglet moves, it takes along its program code as well as its state (data). Bradshaw [4] refers to agent cloning as teleportation.

8.3 The Remote Configurator

8.3.1 Problem

How can an agent be appropriately configured for various destination societies ?

8.3.2 Forces

- For actual migration, an agent has to be cloned in the destination society. Configuration details are needed for cloning, such as the plan library and the beliefs.
- The configuration details and their format depend on the given society's requirements. Thus each agent has to support various kinds of configuration access operations.
- There is a need to represent the configuration accessing functions separately from the agent structure; otherwise each agent has to support many distinct and unrelated operations in object structure.
- Each agent has a similar object structure as all of them are created by the same framework

8.3.3 Solution

The Visitor pattern [10] has been utilized to design the RemoteConfigurator. The ActualMigration handler transfers the Visitor from a distant society to the migrating agent. The ClientProxy in the Mobility layer instantiates the Visitor; this object is passed to the corresponding layers. As the structure of the layers is fixed, the Visitor can gather the configuration information by using the public interface methods. Thus there is no need to define separate methods for transferring the configuration details to another society. Moreover, services can be added by adding new Visitor subclasses, with no change in the agent structure.

8.4 The Broker

8.4.1 Problem:

How is an agent able to gain access to resources and agents outside its society without actually migrating ?

8.4.2 Forces:

- Agents must be able to access each other and other resources across platforms and societies without having to actually migrate.
- Making every agent responsible for access, security and interactions for a society is complex.

8.4.3 Solution:

The Broker pattern [6] provides for location transparency for objects that wish to be clients and servers of one another. With this, the agent (or its Agent 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. The Broker pattern provides virtual agent migration.

8.4.4 Known Uses:

Bradshaw refers to a Broker as a Matchmaker [4].

9. CONFIGURATION, INTEGRATION

9.1 Context and Overview

The process of creating and configuring an agent consists of creating the various layers and then integrating them. The design for creating the object structure of individual layers and integrating them uses the following patterns: Agent Builder and Layer Linker.

9.2 The Agent Builder

9.2.1 Problem

How can the construction of the agent be separated from its representation so that the same construction process can create different representations?

9.2.2 Forces

- Each agent has fundamentally the same structure.
- The creation process should be isolated so that the same process can be used for different structures.

9.2.3 Solution

The AgentCreator collects the user configuration details and passes it to the Creator object. According to the Builder pattern [10], Creator is a Director. The AgentCreator instantiates the Creator with seven Configurator objects, and the Configurator objects are the builder objects which actually create the object structure of each individual layer.

9.3 Layer Linker

9.3.1 Problem :

How can the various individual layers of the agent be integrated together ?

9.3.2 Forces

It is necessary to provide an interface to each layer but also to decouple the layers as much as possible.

9.3.3 Solution

Each Configurator creates the Facade object and other objects which form the structure configured by an agent application developer. Use of the Facade pattern [10] provides both a simple interface and decoupling between the layers. The Facade object implements the unified interface for the layer, promoting layer independence and portability. The Configurator registers them in the configuration repository object BlayerConfigInfo. These repositories are used to get the Facade object reference during the integration phase executed by the Creator object's integrate().

10. CONCLUSIONS

This paper has presented various patterns of use to agent based systems. Many have been used to design, implement and document the JAFIMA system at RMIT. Additional patterns, in particular those pertaining to security in the mobility layer, are under consideration. All of this work is aimed at providing a strong software engineering foundation for agent based systems.

11. REFERENCES

[1] Agre, P. E., and D. Chapman, "Pengi: An Implementation of a Theory of Activity," Proceedings of the 6th National Conference of Artificial Intelligence, 1987.

[2] Alexander C., *The Timeless Way of Building*, New York: Oxford University Press, 1979.

[3] Al- Jabir, S., Sapaty, P. S., and Underhill, M., "Integration of Heterogeneous Databases Using WAVE Cooperative Agents," Proceedings of the First International Conference on the Practical Application of Multi Agent Systems, London, 1996.

[4] Bradshaw, J. M., S. Dutfield, P. Benoit, J. D. Woolley, "KAoS: Toward an Industrial- Strength Open Distributed Agent Architecture," J.M. Bradshaw (Ed.), *Software Agents*, AAAI/ MIT Press, 1997.

[5] Brooks, R.A., "A Robust Layered Control System for Mobile Robot", IEEE Journal of Robotics and Automation, 1986. RA-2(1).

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

[7] Cohen, P. R., and Levesque, H. J., "Intention is Choice with Commitment," Artificial Intelligence, 42 (3), 1990.