Agent Behavior Patterns in a Wireless Internet Environment

Eugene Hung, Joseph Pasquale
University of California, San Diego

Abstract— **Internet applications with wireless clients, which are rapidly increasing in popularity, face obstacles in terms of performance, reliability, and security. While much can be done to address these problems at the lower layers of the protocol stack, we focus on what can be done at the application level, which is complementary to the lower layer work but also has the advantage that it can be easily be deployed on top of existing or newer wireless Internet infrastructures. We propose that wireless Internet applications be enhanced through the use of a stylized form of mobile agent technology which we call *behavior patterns*. These are software templates that aid in the construction of these applications, and help in categorizing and constraining their behavior so that agent-hosts (the intermediary machines capable of executing mobile agents), are more willing to accept them for execution and provide them with the appropriate processing and communication resources. This paper provides an overview of the goals and design of this agent behavior pattern architecture, especially as it relates to supporting wireless Internet applications.**

Keywords—**Wireless, mobile agents, behavior patterns**

I. Introduction

A. Mobile Agents in Wireless Internet Environments

WITH the growth in use of cellular phones, PDAs (personal digital assistants), and other types of handheld devices, there has been a corresponding growth in demand for wireless Internet applications. Existing Internet applications, most of which are structured according to the client-server model, can exhibit problems when run in a *wireless Internet environment*. These problems are often due to the typically large changes in performance, reliability, and security, that can occur at wireless/wired boundaries. The most common scenario is where the client has wireless access to the rest of the Internet, most of which is composed of wired links (in other words, where the last hop to the client is wireless). For simplicity, we will assume this scenario when we refer to a wireless Internet environment in this paper (although our work is not limited to this special case).

One approach to addressing these problems is to use

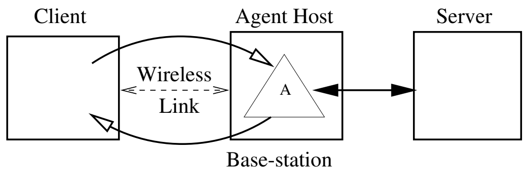

Fig. 1. Agents in a wireless Internet environment

mobile agents[1] to extend and enhance the client-server model. By a mobile agent, we simply mean a program that acts on behalf of a client, typically originating at the client machine and migrating to another machine (which we call the *agent-host*) that acts as an intermediary between client and server. The agent-host runs middleware to support the execution of mobile agents, allowing them to communicate with the client or server and possibly migrate again. In a wireless Internet environment, the agent-host would be either near or at the base-station, as shown in Fig. 1. (The base-station may or may not be able to execute downloaded software itself; if not, a general purpose computer is added alongside it for this purpose.)

Applying agent-based computing to the wireless Internet scenario promotes the following advantages:

Performance Despite the limitations of current wireless devices, a mobile agent can operate in an agent-host's resource-rich environment by migrating there. In addition, with the ability to move closer to the object of computation, a mobile agent may be able to reduce the latency in a portion of the application's network communications.

Reliability Given the ability to move the computation to an intermediary, a mobile agent allows a user to minimize the use of the less-reliable wireless link. A separate protocol (structured to work within the Internet protocol framework) between the client and the agent can be established for this purpose.

Security A mobile agent can be sent to a secure and trusted agent-host, carrying out more sensitive communications from there (e.g., bypassing the wireless link which may be more susceptible to eavesdropping). The agent-host then has the ability to control the amount, type, and format of communication sent by the agent back to the client (over the wireless link). For example, it may be en-

E-mail: {eyhung, pasquale}@cs.ucsd.edu

crypted if not already done so.

Flexibility Mobile agents allow servers to distribute content without worrying about a client's special requirements (e.g., for display), by putting the onus of customization on the client. The mobile agent is constructed to intercept a server's content and customize it specifically for the client.

An example of a wireless mobile-agent application that achieves these advantages is an e-commerce transaction broker. By inserting an agent at the base-station to handle the transaction for the client, the wireless handheld client gains flexibility by having the output customized for its limited display, performance by operating closer to the data server and limiting communications over the problematic wireless link (including the use of compression), reliability by the agent's ability to cache and resend a transaction result, and security by the agent's ability to use an encryption protocol with the client, in a flexible, client-specific manner.

B. Agent Behavior Patterns

Despite the advantages offered above, mobile-agent applications are not as common as might be expected. The reason is twofold: using mobile agents involves learning an unfamiliar programming paradigm, and potential agent-hosts are reluctant to execute foreign code without guarantees on how this code will behave, such as what types of resources will be requested and how they will be used, and what are its security implications. While efforts have been undertaken to improve resource control and security in specific agent systems, more work needs to be done on defining the movement and communication behaviors of an agent application. More generally, there is a need for frameworks that facilitate the building of agent applications that take these issues into account.

As a step towards such frameworks, we introduce the concept of an *agent behavior pattern*. An agent behavior pattern is a codification of general distributed program behavior that would benefit from the use of mobile agents. Specifically, a behavior pattern assumes the client/server model where the client sends a mobile agent to interact with the server on its behalf, and is characterized by (1) how the agent moves, and (2) how it communicates. Applications that are similar in these characteristics can be created from the same pattern, yet are distinguished by their *client-specific functionality* (CSF), which is essentially their application-specific program logic. By combining a pattern's generic code with an application's specific functionality, a programmer can build a mobile-agent application more easily.

Agent behavior patterns also improve control over deployment. By constraining the CSF to use specialized methods for movement and communication, the pattern restricts the application's behavior to act according to its characteristics. This gives agent-hosts the ability to understand the mobile agent's behavior (with respect to movement and communication) before execution, which it may then use to decide whether in fact to even run the agent.

In this paper, we will demonstrate how agent behavior patterns help build deployable mobile-agent applications for a wireless Internet environment. After reviewing the related work in this area in Section II, we describe the patterns and their characteristics in Section III. We then discuss the design and use of these patterns in Section IV, before presenting our conclusions.

II. Related Work

With the growth of the wireless market, many programmers have attempted to port popular network applications to wireless Internet environments. One class of problems is due to the fact that TCP/IP (and related Internet protocols) were not originally designed to take into account some of the unique problems (or opportunities, such as mobility) presented by the integration of wireless links with an Internet comprised of mainly wired networks. Perhaps the best known example is that of TCP's congestion avoidance mechanisms, which may be incorrectly triggered due to wireless link failures that have nothing to do with congestion, leading to a further loss in performance. Some solutions use various forms of agents to mitigate these problems. For example, in [2], the SNOOP Protocol uses a *Snoop agent* (not necessarily mobile) at the base-station to re-send packets that appear to have been dropped by the network due to wireless losses. Extending this concept to constructing an architecture to support a similar mechanism (the *active proxy*) for controlling communications between a wireless client and its base-station was well-demonstrated in [3]. In [4], a more mobile solution is used by sub-dividing the communication link into wired and wireless and migrating the communication handler if the mobile client moves far away from the original base-station.

Mobile agents also provide a means for an application to adapt to the wide variation of capabilities provided by wireless clients. There has been extensive work done on adaptive applications in the past, the majority of which has been under the topic of handling multimedia data streams. Reference [5] introduced the Odyssey system as a set of extensions to an operating system to allow applications to adapt to environment changes in a flexible and agile manner. Reference [6] describes a proxy-based mechanism implemented within the network infrastructure called *dynamic distillation* to provide appropriate levels of data compression for a wide variety of clients. Reference [7] then extends the concept of adaptability to mobile code by describing a mobile proxy server that hosts proxies that transform incoming data into a suitable format for the client.

III. Agent Behavior Patterns

An agent behavior pattern describes and codifies the general movement and communication behavior of a mobile agent. Through use of a pattern, a programmer can create a mobile agent application that embodies the pattern's behavior, while the server receives useful information to decide whether or not to execute the agent.

A. Pattern Characteristics

Patterns define the movement and communication behavior of a client/server-based application extended with mobile agents. While these characteristics are by no means the only possible categorizations of behavior, we chose them because they directly addressed the problems faced by agent-hosts in determining whether or not to run an agent. Controlling movement allows the server to limit potential security leaks, and controlling communication rate allows the agent-host to determine whether the agent's demands regarding communication with the client (e.g., over the wireless link) are acceptable.

While each pattern is unique in its synthesis of movement and communication, patterns do share some similarities. All patterns are based on the assumptions that the agent will migrate to a remote agent-host and execute there, and then communicate with a specified server on the client's behalf, the communication being restricted by the agent to a pre-determined (by the programmer) maximum bandwidth rate. Furthermore, upon completion, the agent never migrates off the agent-host, a model that we call the *one-shot model* of agent computing. A one-shot model of mobile agents provides similar functionality to an itinerant (can move to multiple sites in a pre-determined order) or autonomous model (can choose its destination(s) at runtime) by allowing the agent to spawn child agents that can migrate to other sites, while eliminating the requirement that the CSF actually use migration commands to move to another site and thus violate behavior. The child agents must also use patterns as the basis for their creation, so that their behavior does not violate the behavior of their parents.

A.1 Differences in Characteristics

The differences between patterns lie in whether the agent can spawn other agents from the agent-host, and whether it can communicate with the client (as opposed to simply moving back with the results).

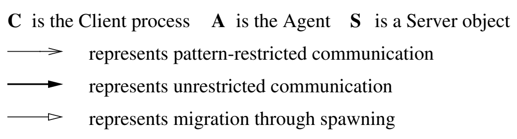
C is the Client process **A** is the Agent **S** is a Server object
represents pattern-restricted communication
represents unrestricted communication
represents migration through spawning

Fig. 2. Structure graph key

A.1.a Movement.

- **Cannot spawn children from the agent-host**. When a one-shot agent is unable to spawn from the agent-host, the agent-host knows it has full control over the agent's future behavior and the information it sends, while a programmer knows that all the necessary operations must be done from that agent-host.
- **Can spawn children from the agent-host to specific destinations**. The agent-host is given a list of potential destinations, which it can use to decide whether the agent is a potential security leak. In return, the capability to spawn children gives the programmer the ability to emulate further migration without circumventing behavior constraints.

A.1.b Communication.

- **Cannot communicate to the client from the agent-host**. A programmer will use patterns with this behavior to minimize the amount of communications over the wireless link between the client and the agent-host. An agent-host can run an agent based on this pattern knowing that it will not communicate with the client during execution.
- **Can communicate from the agent-host at a maximum rate of *N* Kbps**. The pattern will enforce a maximum rate of communication back to the client, allowing the agent-host to better allocate bandwidth when hosting multiple agents. While the programmer gets to specify the maximum rate of communication, an agent-host is given the opportunity to reject the agent if its communication demands are deemed too high.

B. Fundamental Behavior Patterns

For mnemonic purposes, the patterns have been given names that reflect their characteristics. Each pattern is described in terms of its movement and communication characteristics, and its utility towards wireless Internet applications. There will also be a graphical structure depicting the pattern, using symbols as defined in Fig. 2.

B.1 Standard

- Movement: Can spawn children but not move off host.
- Communication: Can communicate with client.

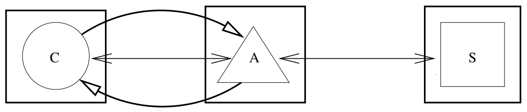

Fig. 3. Structure of the Standard pattern

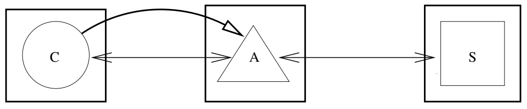

Fig. 4. Structure of the Deadend pattern

The Standard pattern does not bar any specific behavior, but it is still restricted in that it can only create children at the destinations within the list provided, and the communication rate is bounded as specified. While the Standard pattern could be used for any application from the programmer's perspective, agent-hosts may be reluctant to execute a Standard pattern-based application over a more restricted one.

B.2 Deadend

- Movement : Cannot spawn children or move off host.
- Communication : Can communicate to client.

A mobile agent built from the Deadend pattern is a program that is sent to execute on an agent-host in order to capitalize on that machine's resources or network location. Since the agent is not allowed to move or create further agents from this host, either the agent-host is highly desirable from the agent's viewpoint for operations, or the agent plans to terminate at the end of execution.

B.3 Silent

- Movement : Can spawn children but not move off host.
- Communication : Can not communicate with client.

Many applications require reliable or private communications between client and server. An unstable, insecure wireless network can reduce the utility of such applications. An application that uses the Silent pattern creates a mobile agent that attempts to circumvent the wireless link nearest the client. The agent is sent to an agent-host, possibly the base-station, to transact on behalf of the client, and does not communicate with the client over the problematic link while executing. Upon completion, the agent spawns a child to transmit the results back to the client, invoking any agent recovery methods built into the system in case of failure.

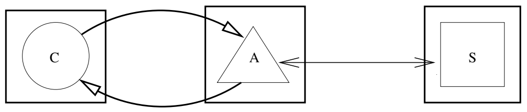

Fig. 5. Structure of the Silent pattern

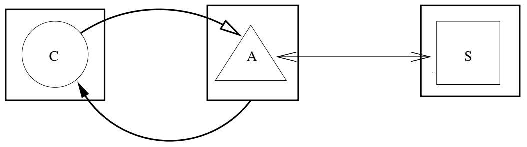

Fig. 6. Structure of the Isolated pattern

B.4 Isolated

- Movement: Cannot spawn children or move off host.
- Communication: Cannot communicate to client or remote server.

The Isolated pattern is the most restrictive pattern in that it forces the mobile agent to act in complete isolation from the client. When the agent completes, it is destroyed by the agent-host and no children are allowed. However, when the agent is destroyed, the agent may provide the agent-host with results to be communicated back to the client, at the agent-host's discretion. This pattern is most useful for running on agent-hosts that distrust the security of the client that the agent comes from: these will accept Isolated agents that give them complete control over the data that gets sent back.

IV. Software Interface

A. Programming Language

We have chosen to use Java[8] as our implementation language for patterns, due to several reasons. First, Java's modularity allows the delineation of the client-specific functionality from the core pattern code as separate *classes*, or method packages. Second, Java's dynamic extensibility allows the pattern code to instantiate and load classes on demand, a necessity for providing a pattern's flexibility in restriction and invocation. Java programs are also portable, only requiring that the destination server be able to run a Java Virtual Machine (JVM). Portability is a requirement in any heterogeneous environment, which is the common case for the pattern's targeted wireless Internet scenario. Finally, JVMs are currently the *de facto* standard for running foreign code on the Web, making Java-based applications highly deployable.

Consequently, we expect that any programmer wishing to employ patterns in their applications will use agent

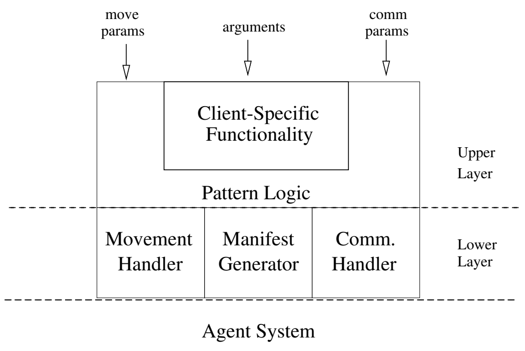

Fig. 7. Pattern architecture

systems that are both able to invoke Java programs and have their movement and communication procedures be invoked in return by a Java program. We also require that the agent system be able to obtain a copy of the code for the client-specific functionality, whether it be carried by the agent or downloaded from a software repository. These requirements are all met if the agent system is written to support Java-based agents. We feel this to be a reasonable requirement, as leading agent systems such as Aglets[9], Concordia[10], D'Agents[11], Mole[12], and NOMADS[13], all support Java-based agents to some degree.

B. Interface

The software interface to these patterns can be separated into two layers: the upper layer, which is the interface to the application programmer, and the lower layer, which is the glue between the agent host and the pattern logic(Fig. 7).

B.1 Upper Layer

The programmer invokes a pattern by calling the constructor of the appropriate pattern class. The constructor takes the following as arguments: the name of the target agent-host, the name and port of the data server connection, the name of the client-specific functionality class file and its arguments, and behavior characteristic restrictions (i.e., potential destinations and maximum bandwidth rate (in Kbps) to the server and client). For example, this is the header of the Standard constructor:

```
Deadend (String hostName,
        int serverPort, String serverName,
        String csfName, String[] csfArgs,
        String[] destinationList,
        int serverBand, int clientBand)
```

Other pattern constructors may omit some or all of the final characteristic restriction parameters if those characteristics are not supported by the pattern.

The constructor then runs the pattern logic, which encapsulates the differences unique to each pattern. The pattern logic checks whether the arguments are consistent with the behavior and generates an agent that moves to the specified agent-host to execute the CSF while limiting its movement and communication behavior. In addition, depending on the pattern, the programmer has access to the communication handler's interface of send/receive methods, which consists of $\{$*sendServer*, *receiveServer*$\}$ and possibly $\{$*sendClient*,*receiveClient*$\}$. The programmer should use these methods, and only these methods, for the CSF's communications.

B.2 Lower Layer

The lower layer consists of the manifest generator, movement handler, and communication handler.

B.2.a Manifest Generator. A method that generates a *manifest*, which is a declaration (in the form of a text file) that a potential agent-host can read to understand the expected behavior. The manifest currently describes the parameters that restrict movement and communication.

B.2.b Movement Handler. A method that acts as an interface for the Standard and Silent patterns to spawn another agent on a remote agent-host. (The Deadend and Isolated patterns do not have a movement handler.) The destination list parameter is incorporated into the method's variables during agent creation, and this list is used to guarantee that a child agent will not migrate to an unexpected host. A child agent is formed by instantiating a pattern so that the behavior guarantees of the parent may still be maintained. To support flexibility, the child is not required to be the same type as the parent, but Silent agents are only allowed to spawn Silent or Isolated children to prevent violation of the parent's Silent behavior. In addition, the child's permitted destination list, if any, must be a subset of the parent's destination list so that the movement behavior is also enforced.

B.2.c Communication Handler. This consists of send and receive methods that regulate and control the agent's ability to communicate with the client and server. The CSF is only able to call these methods to communicate so that the pattern can enforce the promised communication behavior. In the case of the Silent and Isolated patterns, these methods do not exist.

The send methods are implemented as separate threads so that its routines can be performed without the parent blocking. Each thread is initialized during agent creation with the maximum bandwidth parameter (in Kbps), the appropriate socket handle (to server or client), and the maximum burst it can send (specified by the agent-host through an argument to the agent). The thread keeps a queue of messages that are added whenever the CSF calls the appropriate *send* method. When there is a message in the queue, the thread tries to send as much of the message as possible within the bandwidth rate limits. These limits are maintained by keeping track of the time and size of the last packet transmitted in order. If the size and time preclude sending any data, the thread will sleep for a pre-determined interval (chosen by the agent-host) before attempting to send again.

The receive methods are implemented as functions that listen at a socket and return data that arrives — all the work in regulating bandwidth is done in the send methods.

C. Using Patterns

In order to build an agent application from patterns, the programmer selects the appropriate pattern and prepares the CSF by only communicating through that pattern's *send* and *receive* methods (if any). The programmer then invokes the pattern constructor with its parameters to generate an agent that behaves according to its general pattern and specific functionality, and whose movement and communication methods are instantiated with the limitations set forth by the parameters. The pattern logic then negotiates with the target agent-host to run the agent (through generating and sending a manifest describing the agent's behavior), and, upon acceptance, sends the agent to the agent-host. The agent-host can then start the agent (with parameters such as the maximum burst to further control the agent's communication abilities). The agent code is responsible for executing the CSF, spawning child agents from other patterns, and communicating with the server and client.

V. Conclusions

We have constructed a number of wireless Internet applications using the agent behavior patterns described in this paper, and our current experience is that they indeed ease (and speed up) the design process, and provide improved operation over their non-agent-based counterparts. Through the pattern-based process of construction, the application is able to describe its expected behavior to the agent-host, which can use the information to properly allocate resources and consequently is encouraged to execute pattern-based agents over other agents. We are presently investigating the design of easily-deployable middleware that specifically supports our approach to application-level pattern-based mobile agents, with an emphasis on demonstrating improved flexibility, performance, reliability, and security, relative to standard agent construction and middleware-support methods.

References

[1] D. Chess, C. Harrison, and A. Kershenbaum, *Mobile agents: are they a good idea?*, IBM Research Report, reprinted with an update in J. Vitek (Ed.) "Mobile Object Systems", Springer, 1997.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, *Improving TCP/IP performance over wireless networks*, In Proc. of the 1st ACM Int'l Conf. on Mobile Computing and Networking (Mobicom), 1995.

[3] B. Zenel, *A proxy based filtering mechanism for the mobile environment*, PhD Thesis, Columbia University, 1998.

[4] A. Fieger, A. Boger, and M. Zitterbart, *Migrating state information in mobile environments*, In 5th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS), Tunis, Tunisia, 1997.

[5] B. Noble and M. Satyanarayanan, *Agile application-aware adaptation for mobility*, In Proc. of the 16th ACM Symposium on Operating Systems Principles (SOSP), St. Malo, France, pg 276–287, 1997.

[6] A. Fox, E. Brewer, S. Gribble, and E. Amir, *Adapting to network and client variability via on-demand dynamic distillation*, In Proc. of the 7th Intl. Conf. on Arch. Support for Prog. Lang and Oper. Sys (ASPLOS-VII), Cambridge, MA, 1996.

[7] J. Vass, S. Zhuang, J. Yao, and X. Zhuang, *Efficient mobile video access in wireless environments*, IEEE Wireless Communications and Networking Conference, New Orleans, LA, 1999.

[8] K. Arnold and J. Gosling, *The Java programming language* Addison-Wesley, Reading, MA, 2nd ed., 1998.

[9] D. Lange, M. Oshima, G. Karjoth, and K. Kosaka, *Aglets: programming mobile agents in Java*, In Proc. of Worldwide Computing and its Applications (WWCA'97), Lecture Notes in Computer Science, Vol. 1274, 1997.

[10] D. Wong, N. Paciorek, et al, *Concordia: An infrastructure for collaborating mobile agents*, in Proc. of the First International Workshop on Mobile Agents 97 (MA'97), April 1997

[11] R. Gray, G. Cybenko, D. Kotz, R. Peterson, and D. Rus, *D'Agents: Security in a multiple-language, mobile-agent system.*, In Mobile Agents and Security, Lecture Notes in Computer Science, ed. Giovanni Vigna, Springer-Verlag, 1998.

[12] M. Straber, J. Baumann, and F. Hohl, *Mole — Concepts of a mobile agent system*, World Wide Web Journal, Vol 1., Nr. 3, pp. 123–137, 1998.

[13] N. Suri, J. Bradshaw, et al, *Strong Mobility and Fine-Grained Resource Control in NOMADS* In ASA/MA 2000, p. 2–15, Zurich, Springer-Verlag, 2000.