A Mobile Agent-Based Active Network Architecture

Chih-Lin Hu
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
e-mail: clhu@arbor.ee.ntu.edu.tw

Wen-Shyen E. Chen
Institute of Computer Science
National Chung-Hsing University
Taichung, Taiwan, R.O.C.
e-mail: echen@cs.nchu.edu.tw

Abstract

Active networks enable customization of network functionality without the lengthy standard-mediated committee processes. Most of the works in the literature utilize the capsules or active packets as the means to transfer code information across active networks. In this paper, we propose an active network infrastructure based on mobile agents technologies. In our prototype implementation, mobile agents are the building blocks of carrying functional customizations, and the active nodes offer software application layers, the Agent Servers, to process mobile agent-specific customizations to facilitate network functionality. Both integrated and discrete operational models of network customizations are supported. In addition, for the application-specific protocol development and deployment, an abstract protocol structure and a protocol loading mechanism are presented. Furthermore, we provide an agent management/control mechanism and devise a protocol management/control mechanism. As a result, improved network functionality can be achieved.

1 Introduction

With the rapid Internet proliferation and prevalence, the use of networks and the behaviors of network users are changing and diverse. A dominant TCP/IP network architecture can not keep up with the quick emergence of new network services and technologies which improve network performance and functionality. Without architectural support, current network environments impede the revolution of network communication technologies and prevalent extension of the Internet services. Several crucial problems within today's Internet infrastructure can be identified. For example, difficulty of integrating new technologies and standards into the shared network infrastructure, difficulty in accommodating new services with the existing architectural model, poor performance due to redundant operations at several protocol layers, etc. However, due to being a general architecture and the existing hung investment, the networks can not be replaced rather preserved. A "matter-of-fact" premised way is upgrading and backward-compatible. Despite the strong demands to facilitate the network evolution, the employment of new network technologies and services is still confined by concerns of the lengthy standardization implementation, and widespread incremental deployment.

Active networks [19] have been shown to address these issues through a fundamental change in the nature of the interoperability layer. Compared with the traditional network protocol, an active network aims to offer a software programmable paradigm where allows intermediate network nodes to perform dynamic network customization. Hence, active networks can fulfill rapid evolution of network technologies and introduction of service applications without the lengthy process on standardizing the packet format and protocols. In an active network, network messages can carry specific programs to customize services or configure network elements, and network elements can perform specific computation on the passing messages based on the agreement of program encoding and execution environment. For the network elements that do not support this agreement, they take these messages as legacy network packets and process them in ordinary way. In this way, new services can be brought in rapidly without the requirement for incremental deployment or the need to construct an overlay. As a result, active networks can ameliorate network computation capability and functionality.

Two main different approaches [19], discrete, and integrated approaches, have been introduced to create active networks. With the discrete approach, e.g., ANTS [20], CANES [4], DAN [10], the programs are injected into the nodes from the actual data packets traversing through network nodes. With integrated approach, e.g., IP Option [22], ANEP [1], Smart Packet [17], every message can be treated as a "program." A message (so-called capsule) contains a program fragment that may include embedded data to traverse network nodes. In addition, a mixed approach of these two,

0-7695-0568-6/00 $10.00 © 2000 IEEE

e.g., SwitchWare [2], NetScript [9], has also been proposed. However, capsules or active packets are merely suitable to transferring miniature programs and the discrete pre-loading approach is not the ultimate goal of active networks to establish a flexible architecture. To exploit programmable architecture in "service introduction [21]", in this paper, we propose to use mobile agents [15] to establish an active network which elegantly support both two approaches.

Mobile agents are characterized by their ability of roaming through wide area networks, autonomy of operating asynchronously with foreign hosts, and performing tasks on behalf of end-users. The methodology of mobile agent technology supports encapsulation, program interposition, and execution, which makes a mobile agent as a suitable building block for the construction of active networks [11]. Compared to capsules or active packets which are mainly for code transfer between network elements, mobile agents not only support code transfer mechanism but also inherit the crucial features of the mobile agent technology. Summarily, the benefits [14] to use mobile agents includes reducing network traffic, protocol encapsulation, asynchronous and autonomous execution, dynamical adaptation, integrating heterogeneous systems, and achieve robustness and fault-tolerance.

Over the past two years, we have developed an open mobile agent infrastructure, Pathfinder [7][6], to demonstrate agent distributed computation capacity. In this paper, we further exploit the capability of mobile agents to construct a mobile agent based active network architecture. Our prototype basically supports *weak mobility*, which is the ability of allowing code transfer across different active network element, rather that the proactive or reactive migration, and remote cloning of strong mobility. Initially, we apply mobile agents common with capsule approach that provides a code transfer across network elements. However, essentially the scheme of a mobile agent is more adaptable than other approaches. Our experiment has shown mobile agents being effective in deploying specific customization. In addition, for the demand of monitoring mechanisms at mobile agents, we have devised an agent management mechanism, either absent or incomplete in the other architectures, which makes the mobile agent approach more viable in an active network construction.

Our active network architecture is flexible enough to provide rapid service introduction. For the application-specific protocol development and deployment, we integrate the descrete (out-of-band) and integrated (in-band) transfer models by which a mobile agent can utilize to introduce network applications and services. A mobile agent can be a "all-in-one" mobile agent which carries the main specific code objects along with the dependencies across an active network, or be a compact mobile agent which contacts with some *service agent* to indirectly perform customization. Basically, the installation and configuration of a service agent is similar to other discrete approaches in an out-of-band way. Note that, furthermore, we employ a "third party", which serves a service provider with trusty authentication to promote the service deployment. Although some degree of the effectiveness and performance will be compromised, this mechanism of service propagation can be simplified.

The rest of the paper is organized as follows. Section 2 presents some background and related work. Section 3 first describes a high-level abstraction of our proposed active network architecture and then the design of individual architecture components, network communication protocols, and service deployment. Section 4 discusses the implementation of the architecture prototype. Finally, Section 5 gives some concluding remarks and future research topics.

2 Background and Related Work

This section first describes the active network computation model and the relationship between mobile agent technology and active networks. The current active network research works are then introduced.

2.1 Active Network Computation Model

An active network is a programmable network that allows intermediate nodes to perform computations up to the application layer; in addition, "end-users" can program the network by injecting their programs into it [16]. By that reason, the nature of the network service is defined by the behavior of the individual nodes of the network, and how users can control that behavior through coded information placed in their packet [5]. Within an active network the network elements can have the functionality of dynamical control and network behavior. Furthermore, a message can consist of both programs and data. The embedded programs can provide the control functions, and the data can replace the payload of the traditional packet, but in a form of a customizable structure used by the programs. Messages can inject the carried programs into the network elements to customize the specific services or even to configure the network elements themselves. Therefore, the network elements can perform customized computation upon subsequent messages following the path according to a prior specification by messages. In this way, more and more specific functions can be deployed into the networks in an effort to better service the users.

2.2 Relationship between Mobile Agent Technology and Active Network

Many active network architectures currently use code mobility paradigm [11] that is very close to mobile software agent technology. However, the idea of active network is much more general in terms of protocol encapsulation, and service customization, deployment and maintenance. A fundamental difference is that active networks use the concept of network layer processing, whereas mobile agent systems run as application programs. General mobile agent systems are designed to support "agent migratory computation" to construct a ubiquitous Internet computing environment. In contrast, the primary purpose within active networks is to enable communication facilitation, and eventually to offer a flexible, dynamically customization, and programmable network infrastructure. The mobile agent paradigm proposes to treat the network as multiple agent-friendly environments, and mobile agents as programmatic entities migrating from one location to another to perform user-specific tasks. However, active network conceptualizes the network as a collection of active nodes that can perform any computations, and a collection of active packets that carry programs. From this point of view, a mobile agent may be regarded as a specific type of an active packet, and a mobile-agent-compatible node of traditional networks could be regarded as a specific type of an active node [16].

2.3 Active Network Research

Currently, some researchers are defining a general architecture [18] for active network elements that contains multiple execution environments to achieve component interoperability. The Active Network Backbone (ABONE) is being used to test execution environments and implemented applications. The major challenge in implementing this general architecture is to determine trade-off among usability, flexibility, security, and performance concerns.

Due to the space limitation, we list just some related work in the literature as follows. MIT's ANTS [20] features the *capsule*, a packet of Java byte-coded program, to carry specific customization along with data payload. An ANTS's node with Java Virtual Machine (JVM) exports API for a capsule interpretation and execution. In ANTS, capsule can install states and invoke preloaded classes. The granularity of control is at the flow/packet level [5]. SwitchWare [2] at the University of Pennsylvania is based on the layered SANE and uses the scripting language called PLAN [12] to enable strong resource management and security guarantees. In SwitchWare, the packet is not allowed to install states in a network node. NetScript [9] at the Columbia University establishes an agent-based middleware for programming functions of intermediate network nodes. It explores alternative models of active networks in which new services are introduced for control, rather than data transfer purposes, or by network management agents, rather than all users. BBN's SmartPacket [17] applies active networks to assist the problems of managing networks through the use of language design techniques, with the Sprocket and Spanner languages. The Sprocket program is complied into Spanner code to yield very small (smaller than 1 Kbyte) encoded program. CANES [4] and Liane [3] at the Georgia Institute of Technology attempts to construct dynamic, trustworthy services by utilizing a reduced programming model that gives a set of reliable, pre-defined base services through control information in packet header. The advantage is that designers can limit the required security analysis. DAN [10] at Washington University, similar to the approach of CANES, devises a DAN network packet, which contains a finite sequence of identifiers for functions and parameters. DAN aims to set up a topology of code servers in a large scale to simplify code distribution.

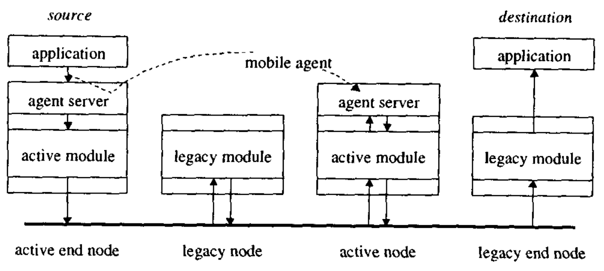

Figure 1. A Mobile Agent Based Active Network

Our work is complementary to several other active network efforts. We utilize the Java-based mobile agents as the building blocks of our active network architecture, to explore the potential of mobile agent technologies. Although the standard of JVM does not allow access to low-level transmission resources and the implementations are limited to the basic network capabilities of Java, similar to ANTS and NetScript, the use of general-purpose Java bytecode and JVM can speed up the development process of our architecture.

3 A Mobile Agent Based Architecture

This section describes the architecture design and considerations of our active network environment. Both agent migratory computation models and enhanced network functionality can be simultaneously provided in this active network architecture

3.1 Architecture Overview

As illustrated in Figure 1, our active network architecture consists of a group of active nodes which are interconnected across the local or wide area and by point-to-point or shared medium channels. An Agent Server sits at the top layer of each active node. Based on the agreement of program encoding and execution environment, an Agent Server will handle mobile agents which carry specific customizations and provide them with restricted, transient execution environments. In this way, specific services can be deployed into networks

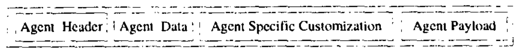

Figure 2. The Logical Format of a Mobile Agent

from active nodes to node by terms of the transport of mobile agents. In addition, dynamical code loading mechanism is utilized to remotely load referred protocol code objects as need. If a network element does not support the agreement, it processes these network messages (mobile agents) using default protocols. Therefore, active network nodes are compatible with legacy networks.

3.2 Mobile Agent Logical Format

In our design , a mobile agent contains the specific programs together with data payload over active network environment. The data within mobile agents is in a customizable structure and the computational states of mobile agents can be portable. As depicted in Figure 2, a mobile agent must contain an Agent Header to identify themselves from other network messages. Agent Data filed includes the mobile agent attributes and related arguments or information used for this agent execution at active nodes in active networks. Agent Specific Customization field involves the specific code objects to be executed at active nodes. Agent Payload field contains delivered data or agent computational results for migratory computing purpose.

3.3 Agent Communication Protocol

An agreement of mobile agent communication protocol is a prerequisite for mobile agents to traverse an active network. In the beginning, we implemented this protocol based on the Agent Transfer Protocol (ATP) specification [13]. The original ATP aims to demonstrate mobile agents endued with ubiquitous computing capacity. At the same time, management and control mechanisms are either absent or not complete in most mobile agent architectures as well as active network architectures. With the demand of monitoring mechanisms at mobile agents or active messages; as a result, in this paper our architecture has enhanced ATP by adding management and control actions[8]. Figure 3 shows the life cycle of a mobile agent in our proposed architecture. As a mobile agent arrives at an active node, it can be in one of the five states - running, suspended, stopped, aborted, and completed.

3.4 Agent Server Design

As shown in Figure 4, basically an active node runtime environment is logically divided into two layers, the lower layer is the physical network node and the higher layer is the Agent Server. The Agent Server provides the agent-to-

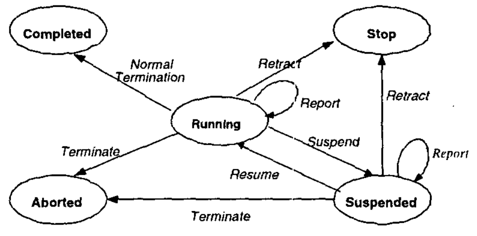

Figure 3. The Life Cycle of a Mobile Agent

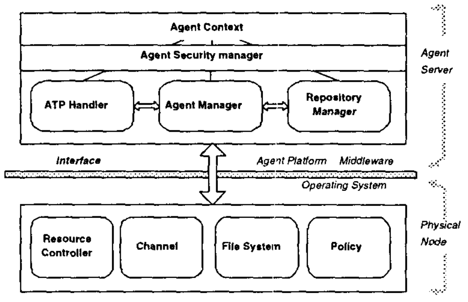

Figure 4. Active Node Runtime Environment

node interface is the real entity which mobile agents contact with for network customization. Through the consistent interfaces, mobile agents can indirectly access the local resources to facilitate agent computation and communication.

3.4.1 Components in an Agent Server

Agent Context is a transient agent execution environment. A mobile agent is constrained to achieve local resource only through this interface to use the primitives of Agent Server.
Agent Security Manager is to check any operations outside the Agent Context, filter out malicious agent behaviors and unwanted interactions. Currently we practice uniform control of system resource upon all mobile agents.
ATP Handler distinguishes active messages from incoming channel and passes it to the evaluation procedure. Similarly, ATP Handler transfers the wrapped mobile agent in an active message in ATP format.
Agent Manager records all the mobile agents and instantiated service agents, and manages the deployed protocols at this node.
Repository Manager stores the protocol-specific "soft-states" and information for agent coordination and collaboration.

3.4.2 Primitives in Agent Server

The initial set of primitives in the Agent Server decides what processing routines mobile agents can do on active nodes, then affects what applications or services mobile agents can deploy in active nodes. For the efficiency aspect in application or protocol development, such a set of primitives is applicable for the compactness and effectiveness of mobile agents. In our scheme we suggest that this initial set should include at least the following primitives.

Local storage access: the primitives to access the local node resources, e.g., the local files.

Local environment access: the primitive to obtain the active node environment information, e.g., port status.

Agent repository access: to store/access shared "soft-states," arguments or objects of application-defined to perform cooperative applications.

Agent manipulation primitives: the primitives for a mobile agent to access its header, agent date, specific customization and payload and manipulate itself during agent execution.

Agent communication channels access: to access the network communication channel to achieve agent migration between active nodes.

Mobile agent control operations: the agent management/control operations, e.g. suspend and terminate operations.

3.5 Applications/Services Deployment Scheme

3.5.1 An Abstract Protocol Structure

For the rapid applications/services protocol developments and deployments, an abstract data structure of protocol is necessary. In our design, this abstract protocol structure would enclose the protocol identifier, protocol-defined information of service_agent and mobile_agent, protocol-defined arguments, protocol-defined code objects, etc. Any new protocol development has to extend this protocol abstraction so that all protocols have a uniform structure. Therefore, in our active network environment an active node will be enabled to operate, administrate and maintain (OA&M) those deployed protocols.

3.5.2 Protocol Deployment Mechanisms

Since neither integrated nor discrete approach suits every need for active networks, we have firstly devised a mobile agent in combining both of two transfer models. Furthermore, we employ a "third party" - a Service Provider, to assist the protocol deployment.

Active nodes should prohibit arbitrary injection of application-specific code. The customization right is granted only to a few authenticated users for safety and security reasons. As a result, it is reasonable and amenable that we employ a "service provider" with trusty authentication to launch trustworthy mobile agents to promote the protocol deployment mechanisms. The third parties are the repository of application-specific codes, by responding to "protocol code requests." The mechanisms of protocol propagation can be simplified although some degree of the effectiveness and performance will be compromised.

4 Prototype Implementation

Section 4.1 firstly lists the characteristics of our mobile agents based architecture. Section 4.2 introduces the experimental environment. Section 4.3 describes the implementation of encapsulating a mobile agent. Section 4.4 explains how an agent server works. Mobile agent operational model is introduced in Section 4.5 and mobile agent management mechanism is given in Section 4.6. Finally, Section 4.7 describes the protocol deployment and management mechanisms.

4.1 Characteristics of Our Architecture

- A mobile agent includes an itinerate schedule used for agent mobility. This schedule provides the information about how this agent is transferred within active network; hence we can get agent information from it.
- An Agent Server provides individual incoming mobile agent a transient execution environment. Different mobile agents perform different tasks bounded in its individual agent context at an active node. No resource conflict and overlapping will happen.
- The ATP communication protocol provides agent mobility operations and our extensions add the needed management and control operations.
- A specific service agent in an Agent Server is responsible for mobile agents of a specific applications/services. Multiple service agents can be installed in Agent Server. As a result, multiple services are feasible in active network through the service agent facility.
- The design of the abstraction of protocol structure enables the dynamical protocol deployment across active nodes.

4.2 Experimental Prototype Environment

Our prototype is implemented in the Java programming language. The programs of the Agent Server and mobile agents are compiled in JDK 1.1.x. The experimental platforms include Windows 98, Linux Slackware 2.0.35 and Solaris 2.5 workstations, which all support Java Virtual Machine. Our prototype uses the enhanced Agent Transfer Protocol (ATP) in application layer and the lower layer is the conventional TCP/IP protocol.

4.3 The Encapsulation of Mobile Agent

Figure 5 depicts the syntax and structure of agent encapsulation. Agent Header Field identifies the ATP communi-

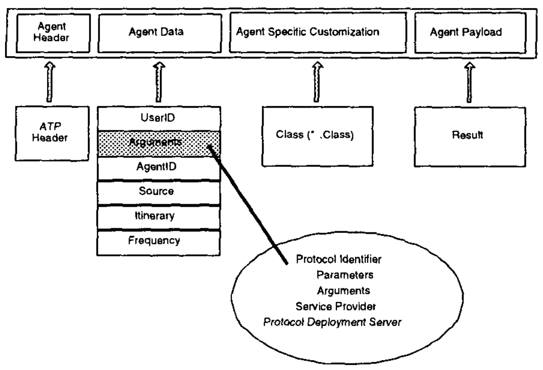

Figure 5. The Mobile Agent Encapsulation

cation protocol. The prefix of the header identifies itself as one kind of ATP message according to our ATP specification. The contents of Agent Data Field[1] are open for the network designer to define. For instance, User ID identifies the service provider with trusty authentication; Arguments contains all the related information referred or modified by customization process; Agent ID is a unique ID for management and control purposes. Agent Specific Customization Field stores lead class and all necessary other classes. When Agent Server receives a mobile agent, it will extract this field, and execute the customization. Agent Payload Field stores the intermediate/final computational results of String type.

4.4 Agent Server Implementation

Our prototype provides an Agent Server, which is a software application layer, to run on an active node to support the specified customizations on behalf of mobile agents.

Functions of Agent Server

First, an Agent Server can parse the incoming active messages, and accordingly generate this agent and then export it a transient execution environment. Second, it can encloses this mobile agent code objects, arguments, dependencies and intermediate results as an active message, and then dispatch it to the next active node. Third, it can pass a mobile agent to its corresponding service agents. Finally, an Agent Server can apply agent control operations accordingly by ATP control messages from the Management Server.

Agent Server Instantiation

When an active network is instantiated, the Agent Server in an active node first initializes the prototype properties profiles stored in the local file system and loads in the local registered protocols to service protocol-specific mobile agents. Finally, the Agent Server triggers the ATPHandler to process the incoming ATP messages.

The Evaluation of a Mobile Agent

In our active network architecture, all active nodes implement the same agent evalution procedure model. The model has four entities with mutual relationship, described in turn.

1. ATP Handler distinguishes the ATP messages from incoming network messages, and likewise, transforms the wrapped mobile agent to next active node accordingly.
2. ATP Parser parses the ATP message from the ATP Handler. After parsing, the code information are extracted to instantiate this mobile agent.
3. ATP Loader initializes an agent execution context for the instantiated mobile agent, and then loads in this mobile agent. Agent Server then starts the specific customization within this agent context.
4. ATP Creator re-generates (and encapsulate) a mobile agent after mobile agent finishes its assignment in this active node. ATP Creator will encapsulate ATP header, agent contents, agent arguments, and another related information into a mobile agent according to the ATP specification.

Mobile Agent Execution in Agent Server

For the safety and security considerations, we implement that mobile agents are only permitted to indirectly use the primitives provided by the Agent Server through the agent context interface. In this way, the agent-specific customization is confined in the transient agent context. Direct operations beyond this scope are forbidden. The operations on the "soft-states" in Repository and Agent Manager and the local resource access are required through the assistance of agent context interface and agent-to-node interface.

4.5 Mobile Agent Operational Model

From the interaction between mobile agent and agent server, we have two mobile agent operational models.

- **All-in-one mobile agent** is responsible for the deployment of application-specific protocol and the customization of active nodes. The agent should carry the main application-specific code objects along with the dependencies. This model is simple, but mobile agent may be heavy and the agent evaluation and execution will degrade make the performance of active node. Moreover, the design of agent server will be more complicated for the reason that agent server needs to interact with various mobile agents.

[1]We utilizes Itinerary for agent migration, because active node currently has no "Agent Forwarding" mechanism like "Capsule Forwarding [3]."

- **Service agent** is a stationary agent in active node, and can provide assistance for mobile agents to customize the active node. Basically, service agents are separate entities from the agent server; therefore, the functions of service agent can be flexibly designed and developed to meet specific application demands. In this case, the agent server is just an intermediator that passes the mobile agents to respective service agents. Multiple service agents can reside in one active node for different services without complicating the agent server's design. Our prototype utilizes the multiple service agent facilities to enable application/protocol deployments and developments.

4.6 Mobile Agent Management and Control

We use four tables - Agent Table, Launch Table, Service Table and Management Table to record the management and control information in our prototype. Each Agent Server employs an Agent Table to record the information of the arrived mobile agent as well as service agents. When starting a service agent, Agent Server assigns a port to this service agent, and then records it. Once a mobile agent requests a service agent to help the assigned customization, it can look up the Service Table by the "service agent name" to get the corresponding port. The Management Table located in the Management Server is used to record the information of launched mobile agent in active network. When the Mobile Supporting Server launches a mobile agent, it also sends this information to the Management Server. Hence, the Management Server can manage/control a launched mobile agent according to its record. Note that because the Management Server does not actively trace the launched mobile agent in our implementation, the information in the Management Table may be out-of-date. However, the information is still useful for us to locate mobile agents.

4.7 Protocol Deployment and Development

The Abstract Data Structure of Protocol

Our prototype implements a data structure to maintain the abstract information and dependencies of the protocol. The Figure 6 shows the ProtocolInfo constructor. Vector Class_Group records all names of the protocol-related classes for the purpose of the dynamical protocol deployment. Vector Agent_Group records all of the protocol-specific mobile agent names. Because in our prototype, multiple mobile agents can be grouped in a protocol domain, they can perform collaboration or coordination in an Agent Server. String Service_Agent_Name records the service agent name of this protocol. Vector Arguments serve as a temporary repository which stores all the protocol-related arguments or parameters.

Protocol Deployment

```
Public class ProtocolInfo {
   public ProtocolInfo( String uid, String pid, Vector class_group,
            Vector agent_group, String sa_name, Vector args ) {
     User_id = uid;
     Protocol_ID = pid;
     Protocol_Status = RUNNING;
     Class_Group = class_group;
     Service_Agent_Name = sa_name;
     Arguments = args;
   }
   ......
}
```

Figure 6. The ProtocolInfo Constructor

Initially, an Agent Server only permits the agent's customization if and only if the protocol status is "RUNNING." In order to meet the ultimate goal of dynamical protocol deployment, our prototype employs the trustworthy Service provider to facilitate the protocol deployment. The location of Service Provider is default specified, and also we can flexibly assign its location in the Arguments Vector of the Agent Data Field. Note that, it is reasonable that we create a new default channel for the utilization of protocol deployment to communicate among a Service Provider and Agent Servers. Compared that ANTS [20] implemented their protocol propagation mechanisms by referred to the last active node where this mobile agent comes from. Intuitively, our prototype also can perform this mechanism. However, in order to reduce the complexity of implementing the prototype, we employ the central Service Provider. In addition, through the facility we can efficiently handle the protocol propagation.

Protocol Management and Control

Our implementation can support multiple protocols simultaneously in an Agent Server. In addition, mobile agents with different specific customizations can deploy different protocols in an active node. In order to maintain efficiently the deployed protocols, the Agent Server employs a Protocol Table, as shows in Figure 7, which records the information of the deployed protocols for protocol management and control in an active node.

When an Agent Server is instantiated, it first stores the information of deployed protocols into the Protocol Table. When a new protocol is deployed in this active node, its information is recorded in this table. Moreover, trustworthy third parties are granted the authority to delegate a mobile agent to adapt the configurations of their deployed protocols on the active nodes.

User ID	Protocol ID	Status	Service Agent	Deploy Time	Update Time

Figure 7. Protocol Table

5 Conclusion

In this paper, we have presented an active network architecture that is based on the mobile agent technology to speed up the deployment of new network innovations. It is appealing that our architecture supports both integrated and discrete approaches simultaneously with the operational interaction between mobile agents and specific service agents, and in addition, a dynamical protocol deployment mechanism is introduced to facilitate the network evolution. Compared with those conventional active networks' dedicated code transfer mechanisms, furthermore, we have developed a management mechanism that provides controls to network operators while increasing the network functionality. With the design and implementation of the agent encapsulation format, the active node architecture, and the agent-contained customization process in the Agent Server, coupled with the prototype implementation, we have demonstrated that mobile agent is a suitable alternative for constructing active networks.

Currently, although there are many research efforts in active network infrastructures, the proposals do not provide means for them to interwork. Mobile agent-based active networks provide a promising basis for future interoperability as some standard communication technologies, e.g., KQML, KIF, CORBA, etc., have been proposed. At the same time, several standardization bodies, such Object Management Group (OMG), Mobile Agent System Interoperability Facility (MASIF), and Foundation for Intelligent Physical Agents (FIPA), etc., are working on mobile agent standardization. We are investigating the issues related to interoperability and integrating the standards into our active network architecture.

References

[1] D. S. Alexander, et al. Active Network Encapsulation Protocol (ANEP). In *RFC Draft*, July 1997.

[2] D. S. Alexander, et al. The SwitchWare Active Network Architecture. *IEEE Network*, pages 29–36, May/June 1998.

[3] S. Bhattacharjee, K. Calvert, and E. Zegura. LIANE - Composition for Active Networks. In *Proceeding of IEEE Computer Communications Workshop*, September 1998.

[4] K. Calvert, E. Zegura, and J. Sterbenz. CANEs: A Modest Approach to Active Networking. In *Proceeding of IEEE Computer Communications Workshop*, September 1997.

[5] K. L. Calvert, et al. Directions in Active Networks. *IEEE Communications Magazine*, October 1998.

[6] W.-S. E. Chen and C.-Y. Lin. A Mobile Ray Tracing Agent. In *Proceeding of the 1999 Autonomous Agents Workshop*, 1999.

[7] W.-S. E. Chen, et al. An Open Infrastructure for Mobile Agents in Mobile Computing. In *Proceeding of the 3rd Workshop on Mobile Computing*, pages 61–67, May 1997.

[8] W.-S. E. Chen, et al. Mobility and Management for Mobile Agents. In *Proceeding of the 2nd International Conference on Autonomous Agents*, May 1998.

[9] S. da Silva, D. Florissi, and Y. Yemini. Composing Active Services in Netscript. In *Proceeding of DARPA Active Networks Workshop*, March 1998.

[10] D. Decasper and B. Plattner. DAN: Distributed Code Caching for Active Networks. In *Proceeding of IEEE INFOCOM'98*, March 1998.

[11] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transaction on Software Engineering*, 24(5), May 1998.

[12] M. Hicks, et al. PLAN: A Packet Language for Active Networks. In *Proceeding of the Intl. Conf. On Functional Programming (ICFP'98)*, 1998.

[13] D. B. Lange and Y. Aridor. Agent Transfer Protocol - ATP/0.1 draft number: 4. In *IBM Tokyo Research Laboratory, http://www.trl.ibm.co.jp/aglets/atp/atp.htm*, March 1997.

[14] D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Communication of the ACM*, 42(3), March 1999.

[15] H. S. Nwana and D. T. Ndumu. A Breif Introduction to Software Agent Technology. In *N. R. Jennings and M. J. Wooldridge (Eds.) Agent Technology: Foundations, Applications, and Markets*, pages 29–47, Spring-Verlag 1998.

[16] K. Psounis. Active Networks: Applications, Security, Safety, and Architecture. *IEEE Communications Surveys*, First Quarter 1999.

[17] B. Schwartz, et al. Smart Packets for Active Networks. In *Proceeding of IEEE OPENARCH'99*, March 1999.

[18] J. M. Smith, et al. Activating Networks: A Progress Report. *IEEE Computer*, April 1999.

[19] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. In *Proceeding of Multimedia Computing and Networking*, January 1996.

[20] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceeding of IEEE OPENARCH'98*, April 1998.

[21] D. J. Wetherall, U. Legedza, and J. Guttag. Introducing New Internet Services: Why and How. *IEEE Network*, pages 12–19, May/June 1998.

[22] D. J. Wetherall and D. L. Tennenhouse. The Active IP Option. In *Proceeding of the 7th ACM SIGOPS European Workshop*, September 1996.