Frameworks and Environments for Mobile Agents

Haeng Kon Kim
Department of Computer Information & Communication Engineering,
Catholic University of Daegu, Korea
hangkon@cu.ac.kr

Youn-Ky Chung
Department of Computer Engineering, Kyung Il University, Republic of Korea
ykchung@kiu.ac.kr

Abstract

The Mobile agent-based distributed systems become obtaining significant popularity as a potential vehicle to allow software components to be executed on heterogeneous environments despite mobility of users and computations. However, as these systems generally force mobile agents to use only common functionalities provided in every execution environment, the agents may not access environment-specific resources. In this paper, we propose a new framework using Aspect Oriented Programming technique to accommodate a variety of static resources as well as dynamic ones whose amount is continually changed at runtime even in the same execution environment. Unlike previous works, this framework divides roles of software developers into three groups to relieve application programmers from the complex and error-prone parts of implementing dynamic adaptation and allowing each developer to only concentrate on his own part. Also, the framework enables policy decision makers to apply various adaptation policies to dynamically changing environments for adjusting mobile agents to the change of their resources.

1. Introduction

As wireless devices such as PDAs and cellular phones and new Internet based technologies, for example, grid, ubiquitous computing and active networks, has been rapidly emerged, modern computing environments are becoming very complex [3, 7, 16]. Also, mobility of users and devices leads to their softwares being executed on dynamically changing environments that support different capabilities and types of available local resources respectively. In terms of software development, these issues make it difficult to design the application programs because it is impossible to obtain completely accurate information about their dynamically changing runtime environments in advance. Thus, it is essential to develop a new middleware platform allowing software components to adapt to their local execution environments at runtime.

Mobile Agent technology is gaining significant popularity as a potential vehicle for considering the complexity and variety [6]. Mobile agent is an autonomously running program, including both code and state, that travels from one node to another over a network carrying out a task on user's behalf [8, 13, 14, 17]. However, this beneficial technology cannot become the practical alternative until it copes with the discrepancies among its changing environments. To solve the problem, many previous works primarily use resource abstractions or virtualizations, such as virtual machines or runtime systems for mobile agents [8]. If mobile agents use only common functionalities supported in all execution environments, they work well. But, as they cannot access uncommon functionalities specific only to a few environments, the environment-dependent resources may be unavailable to the agents. Therefore, the mobile agent-enabled infrastructure should be designed to accommodate a variety of static resources like heterogeneous hardwares, operation systems or system configurations, but also dynamic resources like CPU, memory, network bandwidth whose amount is continually changed even in the same environment.

To satisfy the goal, a dynamic adaptation scheme [4] was proposed to enhance efficiency of mobile code in terms of bandwidth usage and scalability. In this scheme, the code which is moved over the network is limited to the parts that are environment independent and needed everywhere. Environment dependent parts are only transferred and assembled when needed. However, this scheme requires that mobile agent application programmers should recognize every environment specific pars in advance, which is not realistic. Moreover, the scheme only considers the static resources whose states are initially set in a particular environment according to its single policy, but changed no longer.

In this paper, we present a transparently adaptive framework using AOP(Aspect Oriented Programming) technique [10, 11] to handle the two problems of the previous scheme. For the first problem, this framework divides roles of software developers into three groups, i.e., mobile agent application programmer, policy decision maker

and component implementer. The first developer has only to write the functional, application-specific code with no knowledge about adaptable software parts. The second developer recognizes environment dependent parts from the functional code and then implements the non-functional code monitoring the execution of the functional code(security, capacity and mobility etc.) using AOP. The final developer makes environment-specific codes. This development flow has the advantage of relieving application programmers from the complex and error-prone parts of implementing dynamic adaptation and allowing each developer to only concentrate on his own part. For the second problem, the framework enables the policy decision maker to apply various execution policies to dynamically changing execution environments for adapting mobile agents to the change of static resources as well as dynamic ones. To prove the power of this framework, we implemented a mobile agent system based on our proposed concepts.

The rest of the paper is organized as follows. Section 2 reviews related works that have been performed in advance and section 3 introduces our framework. In sections 4 and 5, we describe our implemented prototype and conclude this paper with our future works.

2. Related Work

There have been several studies applicable for mobile agent systems' adaptation to changing environments.

Static adaptation is used in the area of component-based software engineering [5, 9]. It accepts a component with a description of the requested modifications and produces the transformed component suitable for the particular application. However, this technique cannot solve the problem of dynamic adaptation because it is generally applied at compile time, not runtime.

In the continuous adaptation techniques [1, 12], resources are monitored and the adaptation process is initiated as the resources are monitored and the adaptation process is initiated as the resource conditions change. The input for this approach is running application relying on frequently and strongly changing resources and classes of resource or quality of service parameters. Its output is generally the update of environment-dependent parameters. Thus, this approach primarily focuses on the parameter adjustment whereas we are looking at adaptable methods to change environment-specific codes.

Sumatra [2] implemented a scenario which reduces the response item of each chatting client by migrating chatting servers, if needed, based on network latency between hosts measured through resource monitoring. Thus, it allows distributed applications to monitor the network state and dynamically place computation and data in response to changes in the network state. However, this system cannot provide dynamic adaptation when an environment changes because a mobile agent programmer had to write applications in SWITCH-CASE style, which is assuming that it can obtain accurate information of execution environments and their states in advance and cope with the changes in all situations.

Brandt and Resiser [4] introduced a dynamic framework to adapt a mobile agent to its currently running environment by using adaptors for identifying, loading and integrating environment specific parts into the mobile agent. The adapters include the context awareness module and the reconfiguration component. It improves efficiency of mobile code in terms of bandwidth through reducing the size of the movable code called the core part. However, the framework requires mobile agent application programmers to recognize dynamically changing parts. Moreover, it can apply only a single policy to the corresponding resource at runtime. So, if the state of each dynamic resource such as available memory size is changing at runtime and then its current implementation becomes non-executable in the stat, the mobile agent may not continuously perform its task any longer because the implementation of the resource is determined only once when the mobile agent arrives at the current execution environment.

3. The Framework for Transparently Dynamic Adaptation

For dynamic adaptation to heterogeneous environments, the previous framework [4] divides a mobile agent program into two parts, core part and adaptable part. The core part consists of functional code and non-functional code generally includes capabilities of context awareness and reconfiguration. The adaptable part is composed of various environment-specific codes. As mentioned above, this framework has two practical problems as follow: First it doesn't achieve complete transparency to application programmers because they must understand beforehand what parts of the mobile agent are environment dependent. As the second problem, the scheme only considers the static resources whose states are initially set in particular environment according to its single policy, but changed no longer.

Our proposed framework addresses the two problems as follow: like in figure 1, initially, application programmers develop the functional code of the core part at the base level. Afterwards, policy-decision makers recognize a part of the functional code that may be affected by underlying heterogeneity and needs dynamic adaptation, determine which execution policies can be applied for the part and then write the corresponding non-functional code, including the applicable policies, interface names, environment awareness module, dynamic implementation loading module and so on, at the meta level. In here, our framework uses aspect-oriented programming technique to achieve the seamless

role separation because of the inherent conceptual separation it makes between the base level and the meta level [10]. If a monolithic mobile agent with the whole code for all different environments written in SWITCH-CASE style is intended to use for dynamic adaptation, the size of its migrated code becomes extremely larger and it is very difficult to transparently modify and add new adaptable parts to the agent program.

Figure 2 shows our dynamic adaptation procedure with multiple policies applied. When the core part of a mobile agent arrives at a new execution environment, it senses the environment and obtains environment-dependent variables from the local repository (in case of static resources) and the infrastructure monitor (in case of dynamic resources). Then, the dynamic loading module at the meta level is performed with interface names, environments variables and applicable policies to determine the appropriate adaptable pars for the local environment, find and load the corresponding implementation procedures from the code repository, and link then to the core part of the agent.

For this dynamic adaptation, component developers have to implement not only the corresponding interfaces, but also their associated condition functions, which are invoked with values of the environment variables related to the interfaces and return true or false according to whether the values are suitable for the interfaces. After checking and comparing all return values of the condition functions depending on their applicable policies, it is determined which implementation class is suitable for this environment.

3.1. Implementation

A simple prototype for realizing our framework is implemented using *u*Code [15], which is a lightweight and flexible toolkit for code mobility written in pure Java programming language. Also, in order to apply the aspect-oriented programming technique to the framework, we use AspectJ[11] that is a simple and practical aspect-oriented extension to Java.

The prototype library consists of a Java package named **Adptation.** This package is composed of three core classes and one interface, i.e., **Place, Context, AdaptationPolicy** and **Aspect.** The functionality of these classes is as follows:

• Place class

This class instantiates a mobile agent execution environment extending *u*Server in *u*Code toolkit. It also plays a role of a code repository maintaining classes for adaptable parts of mobile agents and delivering them to the dynamic loading module. For this purpose, each shared class space associated with a *u*Server is modified to implement the code repository.

• Context class

This class consists of concerning environment variables, e.g., OS name, CPU type, version of Java Virtual Machine, network bandwidth, CPU utilization and so on, and functions

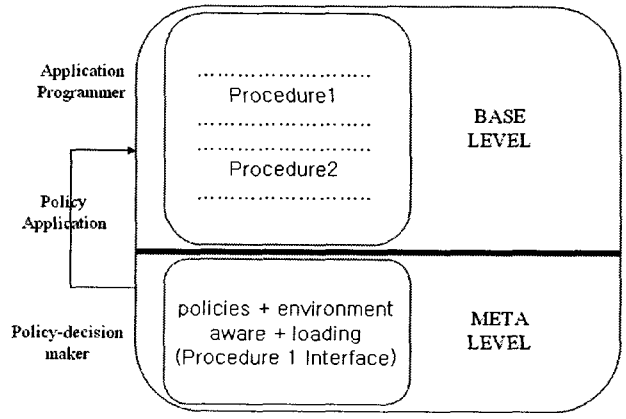


Figure 1. Role separation for transparent adaptation

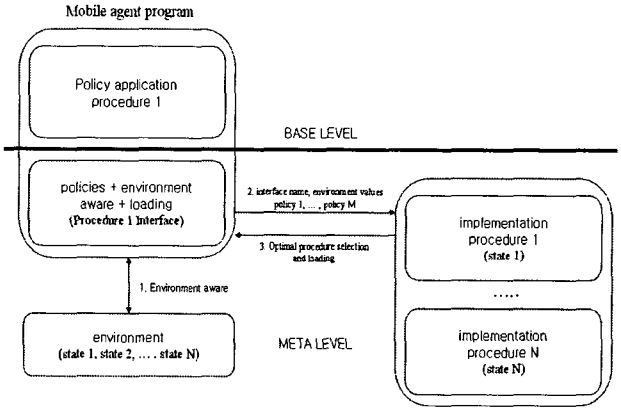


Figure 2. Multiple policies applied dynamic adaptation

for getting values of the variables. In here, each corresponding function is declared in the form of *get_CONDITION(Context.VARIABLENAME).*

For example, if a variable is defined as OS_NAME, function *get_CONDITION(Context.OS_NAME)* is invoked. Thus, when a new context is added, mobile agent programmers have only to use the corresponding variable in Context class and component developers just use the added function without implementing a new one.

• AdaptionPolicy interface
This interface defines the following constants representing various policies.

- AND: the policy suitable in case that all return values of condition functions for the environment variables applied are true.
- OR: the policy suitable in case that at least one among all return values of condition functions for the environment variables applied is true.
- NONE: the policy suitable in case that all return values of condition functions for the environment variables applied are false.

• Adaptation class
This class implements kernel modules of our framework for applying various adaptation policies transparently, sensing execution environments and dynamically loading the corresponding implementation classed from the repository. In Particular, function setAdaptation() is defined to apply policies for a method of a mobile agent to its current runtime environment variables and several policies is invoked, the adaptive implementation of the method is automatically linked to the mobile agent. Also, this class has a function enabling it to access to its local Place class.

4. Conclusions

This paper proposed a transparent adaptation framework using AOP technique to enable policy decision makers to apply multiple policies to dynamically changing environments for adjusting mobile agents to the change of their resources. Unlike previous works, this framework divides roles of software developers into three groups to relieve application programmers from the complex and error-prone parts of implementing dynamic adaptation and allowing each developer to only concentrate on his own part. Second, this framework can accommodate not only various static resources, but also dynamic ones whose states are continually changed at runtime even in the same execution environment. Finally, we implemented a prototype to realize the power of our conceptual framework. However, this prototype is not a full-fledged mobile agent system yet. Also, for performance evaluation, we should implement some previous works, e.g., traditional monolithic adaptation framework where a mobile agent is migrated with the whole code for all heterogeneous execution environments. If all of them have been completed, we will attempt to compare our framework with the previous works with respect to the following two performance indexes: One is the size of migrated code of a mobile agent, which significantly affects network bandwidth. The other is the runtime overhead resulting from the execution of context awareness and the time for loading the implementation classes.

References

[1] G.D. Abowd, A. Dey, R. Orr and J. Brotherton. Context-awareness in wearable and ubiquitous comuting. Virtual Reality, Vol. 3. pp. 200-201, 1998.

[2] A.Acharya, M.Ranganathan and J.Saltz. Sumatra: A Language for Resource-Aware Mobile Programs. Lecture Notes In Computer Science, Vol. 1222. pp. 111-130, 1997

[3] P.Bellavista, a.Corradi and C.Stefanelli. The Ubiquitous Provisioning of Internet Services to Portable Devices. IEEE Pervasive Computing, Vol. 1, No. 3, pp. 81-87, 2002.

[4] R.Brandt and H.Reiser. Dynamic Adaptation of Mobile Agents in Heterogeneous Environments. In Proc. of the International Conference on Mobile Agents, LNCS 2240. pp. 70-87, 2001.

[5] A.Duncan and U.Hölzle. Load-Time Adaptation: Efficient and Non-Intrusive Language Extension for Virtual Machines. Technical Report TRCS99-09, University of California at Santa Barbara, April 1999.

[6] A.Fuggetta, G.P.Picco and G.vigna. Understanding Code Mobility. IEEE Transactions on Software Engineering, Vol. 24, No. 5, pp. 342-361, 1998.

[7] M.Fukuda, Y.Tanaka, N.Suzuki, L.F.Bic and S.Kobayashi. A Mobile-Agent-Based PC Grid. In Proc. of the Fifth Annual International Workshop on Active Middleware Services, pp. 142-150, 2003.

[8] D.Lange and M.Oshima. Programming and Deploying Mobile Agents with Aglets. Addison-Wesley, 1998.

[9] R. Keller and U.Hölzle. Binary code adaptation. In Proc. of the 12th Annual European Conference on Object-Oriented Programming, July 1998.

[10] G.Kiczales, J.Lamping, A.Mendlheka, C.Maeda, C.V.Lopes, J.M.Loingtier and J.Irwin. Aspect-orintend programming. *Lecture Notes In Computer Science*, Vol. 1241, pp. 220-242, 1997.

[11] G.Kiczales, E.Hilsdale, J.Hugunin, M.Kersten, J.Palm and W.Griswold. An overview of AspectJ .*Lecture Notes in Computer Science*, Vol. 2072, pp. 327-353,2001.

[12] B.Noble. System support for mobile, adaptive

applications. *IEEE Personal Communications*, pp. 44-49, 2000.

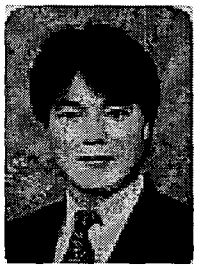
[13] ObjectSpace. Voyager. *http://www.objecspace.com/.*

[14] V.Pham and A.Karmouch. Mobile Software Agents: An Overview. *IEEE Communications Magazine*, Vol. 36, pp. 26-37, 1998.

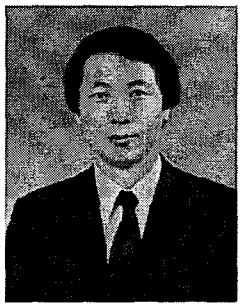
[15] G.P.Picco. μ Code: A Lightweight and Flexible Mobile Code Toolkit. In Proc. of the 2nd Int. *Workshop on Mobile Agents*, pp. 160-171, 1998.

[16] G.P.Picco, A.L.Murphy and G-C Roman. LIME: Linda meets mobility. *In Proc. of the International Conference on Software Engineering*, pp. 368-377, 1999.

[17] K.Rothermel and M.Schwehm. Mobile Agents. *Encyclopedia for Computer Science and Technology*, Vol. 40, pp. 155-176, 1999.



Dr. Haeng-Kon Kim is currently a professor in the Department of Computer Engineering, and Dean of Engineering College, Catholic University of Daegu in Korea. He received his M.S and Ph.D degree in Computer Engineering from Chung Ang University in 1987 and 1991, respectively. He has been a research staff in Bell Lab. and NASA center in U.S.A. He also has been researched at Central Michigan University in U.S.A. He is a member of IEEE on Software Engineering, KISS and KIPS. Dr. Kim is the Editor of the international Journal of Computer and Information published quarterly by Korea Information Science Society. His research interests are Component Based Development, Component Architecture, & Frameworks Design.



Dr. Youn Ky Chung is Professor in the Department of Computer Engineering in Kyungil University in Korea. He received his M.S and Ph.D degree in Computer Engineering from Yeungnam University in 1984 and 1996, respectively. He spent 6 years as an assistant professor of Catholic Sang-Ji Junior College in Korea. He was a visiting Professor at the University of Newcastle in Australia form 1998 to 1999. He was Head of computer Center in Kyungil University during the last four years. His research interests are Multimedia Communication, LAN/WAN Technology, Network management(TINA/TMN) and Next Generation Internet.