Ubiquitous Application Development using a Mobile Agent-based System

Kazutaka Matsuzaki[1], Nobukazu Yoshioka[2], and Shinichi Honiden[1,2]

[1] University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
[2] National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Abstract. This study proposes a methodology that allows the flexible and maintainable development of application based on mobile agent to a ubiquitous environment. Ubiquitous environment is to support people's movement in an inconspicuous and unobtrusive way while they are executing applications. This requires the various kinds of concerns to be written in application code, which makes the application code monolithic. A monolithic code reduces its flexibility and maintainance faculty. Coding techniques to improve the efficiency and testing makes the matter worse. This paper introduce a Worklfow-awareness model based on agent pairing which makes it possible to tune up a performance of the application without disorganizing the application logic. AspectJ is used to combine the non-application logics specific to a deployed environment.

1 Introduction

With the recent developments in wireless networks and diffusion of mobile devices, it is likely that mobile users would use Internet or Intranet services from their mobile terminals. [11] proposes to make hardware devices (printers) to serve as Web Services from a mobile terminal. [2] supports such service discovery for mobile users. We focus on a usage style in which mobile user uses services which stand physically near and logically available via network. For example, we think of situations where mobile user uses some interactive services through his mobile terminal with walking to other floor. After he finishes using the services, he might print out the result of the service with available printer in his reach. Application for such usage style will be needed in many situations. Its whole execution flow will be much more complicated when realizing application logic by using multiple services. We aim for aiding such application development.

Generally, there are many network groups even in a single building because of the social contexts. There may be some companies and some departments in a building. It is feasible that they belong to different network groups. As a result, mobile user is restricted using services; he can access the services only when he connects to the same network groups of the services. To overcome such restrictions, [22, 18] proposes to use mobile agent technologies. Mobile agents are deployed in a wired network and work as proxies in using services for mobile user who requests to uses the services.

Matsuzaki K., Yoshioka N. and Honiden S. (2004).
Ubiquitous Application Development using a Mobile Agent-based System.
In *Proceedings of the 1st International Workshop on Ubiquitous Computing*, pages 204-212
DOI: 10.5220/0002681802040212

Our motivation is realizing an efficient application development which is based on good maintenance ability from separation of concerns. Target application (mobile agent) contains application logic, service adjustment (discovery, negotiation, check alive), performance tuning technique (e.g. interleaving access), testing and other supportive tasks. This makes application code monolithic and complex, which troubles the application programmers. To tackle this problem, we propose Workflow-awareness model. The model consists of some notions: Pairing, Workflow-awareness (WFA) and practical use of AspectJ. Pairing means that the application is composed of two agents. The one is for executing application logic and the other is for subsidiary tasks. WFA means that execution states of the application logic agent are watched by the pairing agent to process its subsidiary tasks in assigned timings. AspectJ is used to weave concerns into an agent for application logics to realize WFA and to alter the behavior of method used in WFA.

Our methodology goal is to keep the agent for application logic being able to run anywhere without any modification to itself. Pairing agents are changed to glue application logic and deployment environment and intention depending on the situation.

In Sect. 2 we discuss the assumptions of our target environments and the actual problems for developing in target application. Sect. 3 presents our methodology feature. Sect. 4 surveys related work and Sect. 5 provides a summary and discusses future issues.

2 Assumptions and problems of the target domain

The focus of this study is on the development of mobile agent application which uses (multiple) Web Services in ad hoc way and tracks user's mobility in a building. In order to gain a narrower scope, we describe some assumptions about the application deployment environment and the application style. After reviewing the assumptions, we discuss the target problems to solve.

2.1 Assumptions for deployment environment

- User tracking systems: In order to track the physical mobility of the user (application owner), mobile agents are able to look up the user's position in a building [9, 3].
- Service advertisement/discovery mechanisms: Services are properly advertised with respect to its service description and their belonging network groups so that application (mobile agent) can judge which services are preferable for the user [2, 1].
- Mobile agent run-time: Each host in a network group has an agent hosting environment with proper access controls mechanisms. This enables mobile agents to use target services by migrating and messaging among these hosts and services.

With these three assumptions, we imagine the situation illustrated in Fig. 1.

2.2 Assumptions for application style

- Coding agent in a state based programming style [20, 10]: Application is written in a state based agent code. The left part of Fig. 3 shows that of Bee-gent

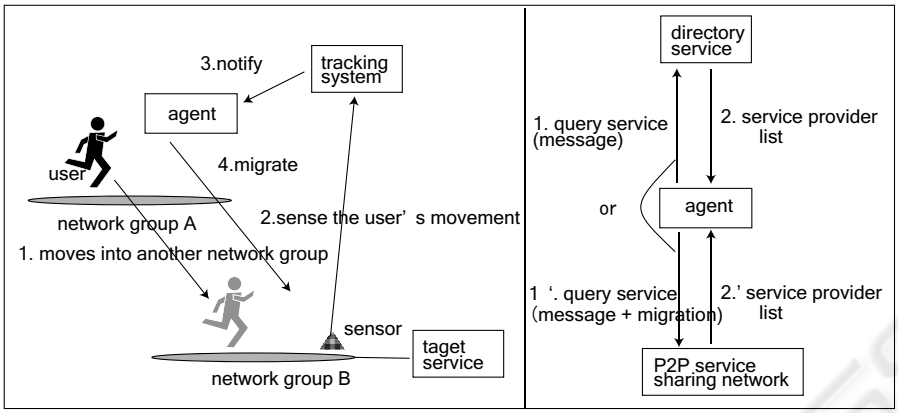

Fig. 1. Concise description of the target situation is shown. (a) mobile agents can follow the user's movement between network groups. (b) mobile agents can know the service provider's both logical and physical location from either a directory service or P2P service sharing protocols.

[10]. It is based on the concept of "states" and "state transitions". Following the "Transaction-Rule", a transition occurs from one state to a state where its "Pre-Condition" matches the Rule.

2.3 Problems

Our target application has many aspects of distributed computing as described above. Requirements for programmers to realize the target application includes

- application logic: a series of process definitions for the application
- service arrangements: arrange services to keep available, i.e. service matching, service preparation, linkage checking between user and services.
- pre-processing: a series of processes needed prior to using services, e.g. authentications, encryption, file format conversion and so on.
- adaptation: control for adapting to dynamic environmental changes due to user migration.

Furthermore, performance tuning code and testing code are also necessary for real deployment. Coding these items with predicting run-time situations at the design stage is quite challenging to application programmers and presents totally a new sphere of problems.

3 Approach

3.1 Pairing of agents

To solve the above-mentioned problems, we propose a new methodology that introduces two types of mobile agents: the Master Agent (MA) and the Shadow Agent (SA). The

fundamental objective is the separation of concerns which are entangled in one agent's code and keeps the required functions satisfied. A monolithic agent that is made with existing method is to be divided into MA and SA depending on the kind of tasks.

- **Master Agent (MA)** has an application logic fulfilled by the communication with both the user and the services. MA's state transition reflects the application logic flows.
- **Shadow Agent (SA)** carries out supportive tasks well-timed for MA execution states. SA's tasks are dependent upon the environment or situation where the MA is deployed, e.g. service matching, service preparation, linkage checking between user and services, pre-processing needed for using services, adaptation to the changes of the environment at the run-time, performance tuning, and testing (See Sect.2.3).

As for performance tuning, SA works as a proxy of MA for some tasks (interleaving execution, preemptive execution etc.). For example, preemptively searching services (service matching) and Future communication [13] against the output from the service execution. These behaviors are only enabled when MA and SA work at the remote places each other. Thus, task separation is done in the form of paired agents, which have remote communication ability between the components. This separation is a kind of AOP in agent development [12]. SA is supposed to be created or customized by the application programmer or code generator adapting the deployment environment and situations. SA is coded using APIs fetching information of the MA execution states, and weaving well-timed tasks. We describe more details in the following sections.

3.2 Workflow-awareness model

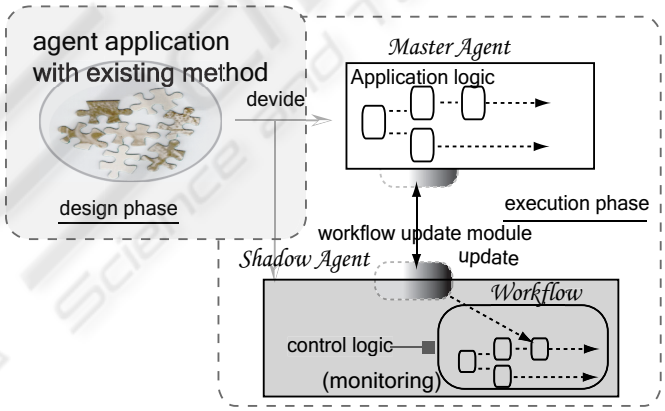

Fig. 2. Concise descriptions of WFA. At the design phase, an old-style agent application is divided into MA and SA. At the execution phase, workflow instance is kept fresh.

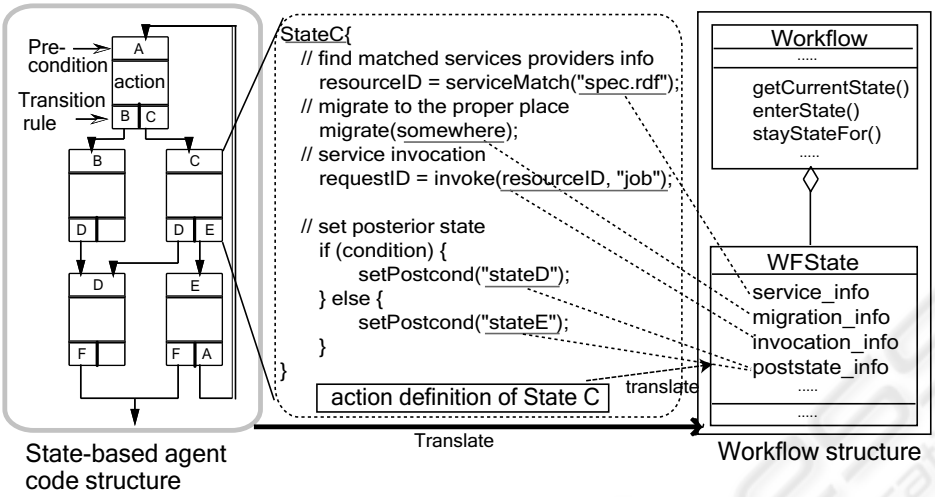

Fig. 3. Translation processes of an agent code. Each state of an agent code is translated into WFState, which becomes a workflow object's component. In this figure, state C is used as example. Potential workflow elements (method arguments) are extracted into WFState.

We propose to use Workflow-Awareness (WFA) to promote the separation of concerns. The key characteristic is that the SA can trace the MA execution states at run-time through the *workflow* object. A *workflow* object is a structure that is composed of extracted elements of the MA state transitions (WFState), i.e. the *workflow* object has miniature state machine (Fig. 3). *workflow* elements are as follows:

- state name: This is used as an identifier.
- migration sentence: An MA migration event would be transparent to the SA.
- service matching sentence: An MA service matching message transition to the service provider would be intercepted by the SA.
- service invocation sentence: An MA service invocation message transition to the service provider would be intercepted by the SA.
- state transition command: MA state transition must be cached by the SA, because the *workflow* object must be kept updated during the MA execution.

Programmer can code the SA with awareness of the timings of important MA actions through the provided WFA related APIs. Fig. 4 is a list of some examples of WFA APIs. These APIs are implemented to send/receive ACL messages [8] bi-directly, which enables SA to make order to the MA actively.

SA's action is basically *workflow* event driven, e.g. "if the MA enters state food, then starts service matching preemptively which will be used by the MA in posterior state." The important point of this is that the application logic would be preserved even when testing or performance tunings occur.

state information	gather the information related to the MA
example	boolean enterState(String state)
description	return true if the MA enters state identified by “state”
state control	control the MA internal state
	boolean suspendMA()
	suspend the MA’s execution
service matching control	control the service matching command
	ResourceID prefetch(Spec specfile)
	find the services matched with specfile beforehand
get the MA info	get the MA field values
	Object getMAFieldValue(String tag)
	ask the values of the MA field identified by “tag”
output control	work as a proxy
	RequestID proxyRecieve([signature of invoke()])
	invoke the service as a proxy and control the output data flow

Fig. 4. Some examples of APIs in WFA.

3.3 Practical use of AspectJ

AspectJ is an AOP programming language which is a seamless aspect-oriented extension to the Java programming language. AspectJ is typically used to generally insert mostly nonfunctional concerns into functional code.

Since WFA intends to support development, some smart measures to realize WFA are needed so as not to make it either difficult to develop or taking much time at runtime.

To meet the requirements, we use AspectJ in some points:

- Automation of *workflow* instance updating: Each MA state transition is handled by the workflow update module interwoven by aspect. This module send/receives notification messages to keep the *workflow* information up-to-date (Fig. 5(a)).
  - main concern: Application logic mapped into MA states
  - cross cutting concern: Notification message passing
- WFA method delegation: WFA supported performance tunings are done by an SA’s processing as a proxy of an MA. (See Sect. 3.2) Since we do not want to lose an MA’s ability to run stand-alone, AspectJ seems to suit well. Fig. 5(b) illustrates how the delegation occurs. Suppose that main function of *serviceMatching* (send query etc.) in *main_part*. This part is rewritten by the advice in the aspect as in Fig. 5(b), which would change the behavior of the MA to make use of the SA. This figure follows the notation of [7]. The “xor” in Fig. 5(b) means that only explicitly declared method invocation should be changed, otherwise “proceed” the normal execution (service matching activity by the MA itself).
  - main concern: MA’s WFA method execution
  - cross cutting concern: Alternate the actor of the method execution

Owing to this methodology, performance tuning can be made with the MA's logics being entirely kept.

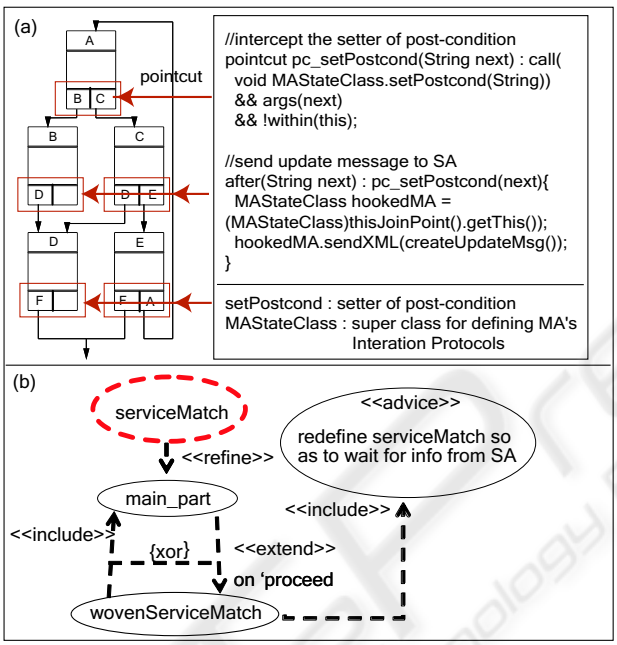

Fig. 5. (a): MA state structure and aspect which offers workflow update functions. (b): Delegation of the method execution by being rewritten with AspectJ.

4 Related Work

This section discusses several studies that have influenced various aspects of this methodology, which is in conjunction with research areas.

As for pairing of the agents, [21] introduces a secure protocol using a cooperation agent recording the path history of each other. [15] uses a pair of agents for interleaving access to services. [15] deploys a delegate-agent for using web services so that efficiency and reliability are increased. In our methodology, the MA and SA each have their own programmed tasks and sometimes act in remote places, which enables a new function in addition to interleaving effects.

Several papers have explored meta-level architecture and reflection for mobile agents [6, 14, 4]. In [6], an agent reasons about the beliefs of another agent, as well as about the actions that other agents may take. This meta-reasoning is similar to WFA : however, it does not support the agent division of labor in different places. This support is desirable under mobile settings so as to efficiently complete tasks.

Several papers discuss mobile agents from an AOP point of view. [12] proposed to introduce the Role Model into an agent system for analysis, design, and implementation. [17] took another approach for separation of concerns for mobile agents by using policy control for binding. In this approach, a mobile agent, called a shadow proxy, roams a fixed network to bind needed resources in place of the applications on mobile terminals. [17] has good mechanisms for resource binding treatment. This is done under the control of Ponder [5], a policy specification language. In our methodology, mobile agent tasks not only work as a proxy resolver for resources but also coordinate between applications and services in wired and wireless networks, i.e. a main part of the application logic. Our priority is considered in two points:

- Availability of the agent execution state enabling the programmer to adapt it to the environment characteristics
- Capability to handle complex jobs by coordination of a pair, which cannot be resolved only by forking a new process.

Most of the approaches intended to exploit logical mobility for supporting user's physical mobility provide middleware [19, 16, 18, 22]. These middleware supports logical mobility in infrastructure levels. [18] realizes the access control of resources from the context information of the middleware components (mobile proxies). Relevant access control rules, client location, user preferences, privacy requirements, terminal characteristics, and current state of hosting environments can be considered, which helps mobile clients obtain flexible access to the resources. [22] is characterized in *capabilities*, a unit that provides a specific functionality to the user, the middleware or to other applications and adheres to a specific interface. These capabilities provide the middleware (SATIN) with modularization and make it easy to deploy and update applications.

We would like to investigate a middleware design to fit our methodology reusing some functions of these existing approaches.

5 Conclusions and Future Work

In the present study, a mobile agent based development methodology for ubiquitous application has been proposed. The approach offers a Workflow-awareness model based on paired agent cooperation for aiding programmers in handling run-time events efficiently while separation of concerns is kept. The methodology also applies AspectJ practically in its implementation. A workflow object contains the latest information on the agent execution state and the pair agent can decide on its action accordingly. This cooperation enables the effective behavior of agents at the service exploitation time.

The direction of our future research is to provide some patterns or rules for programmers as for SA's functions. We are also presently striving to design middleware that suits well with WFA methodology.

References

1. Jxta v2.0 protocols specification, 2003.

2. Web services dynamic discovery, 2004.
3. Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *INFOCOM*, pp. 775–784, 2000.
4. Walter Cazzola, Shigeru Chiba, and Thomas Ledoux. Reflection and Meta-Level Architectures : State of the art and future trends. In *Object-Oriented Technology (ECOOP 2000 Workshop Reader)*, Vol. 1964 of *Lecture Notes in Computer Science*, pp. 1–15. Springer-Verlag, 2000.
5. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, Vol. 1995, pp. 18–38, 2001.
6. Jürgen Dix, V. S. Subrahmanian, and George Pick. Meta-Agent Programs. Technical Report 21–98, 1998.
7. Stefan Hanenberg Dominik Stein and Rainer Unland. An uml-based aspect-oriented design notation for aspectj. In *Proceedings of the 1st international conference on Aspect-oriented software development (AOSD '02)*, pp. 106–112, 2002.
8. FIPA, 2001. FIPA ACL Message Structure Specification, Foundation for Intelligent Physical Agents.
9. D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive computing*, Vol. 1, No. 2, pp. 22–31, 2002.
10. Bee gent Website, 1999. http://www2.toshiba.co.jp/beegent/.
11. The Printer Working Group, 2003. Print Service Interface Version 1.0 Working Draft.
12. Elizabeth A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, Vol. 8, No. 2, pp. 34–41, 1999.
13. Doug Lea. *Concurrent Programming in Java: Design Principles and Patterns, Second edition*. Addison-Wesley, 1999.
14. Thomas Ledoux and Noury M. Bouraqadi-Saadani. Adaptability in mobile agent systems using reflection. In *RM'2000, Workshop on Reflective Middleware, Middleware'2000*, April 2000.
15. Zakaria Maamar, Quan Z. Sheng, and Boualem Benatallah. Interleaving web services composition and execution using software agents and delegation. In *AAMAS 2003 Workshop on Web Serices and Agent-Based Engineering*.
16. C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. Xmiddle: A data-sharing middleware for mobile computing. *Wireless Personal Communications*, Vol. 21, No. 1, pp. 77–103, 2002.
17. Rebecca Montanari Paolo Bellavista, Antonio Corradi and Cesare Stefanelli. Dynamic binding in mobile applications : A middleware approach. *Internet Computing*, Vol. 7, No. 2, 2003.
18. Rebecca Montanari Paolo Bellavista and Daniela Tibaldi. Cosmos: A context-centric access control middleware for mobile environments. In *Mobile Agents for Telecommunication Applications, 5th International Workshop, MATA 2003*, Vol. 2881 of *Lecture Notes in Computer Science*, pp. 77–88, 2003.
19. Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME: Linda meets mobility. In *International Conference on Software Engineering*, pp. 368–377, 1999.
20. Martin C. Rinard and Monica S. Lam. The design, implementation, and evaluation of Jade. *ACM Transactions on Programming Languages and Systems*, Vol. 20, No. 3, pp. 483–545, 1 May 1998.
21. Volker Roth. Secure recording of itineraries through co-operating agents. In *ECOOP Workshops*, pp. 297–298, 1998.
22. Stefanos Zachariadis and Cecilia Mascolo. Adaptable mobile applications through satin: Exploiting logical mobility in mobile computing middleware. *UK-UbiNet Workshop*, September 2003.