A Multi-Agent Framework for Ubiquitous Information Systems: Mobeet Framework

Nobukazu Yoshioka$^1$, Akihiko Ohsuga$^2$, Shinichi Honiden$^3$

$^1$ National Institute of Informatics, nobukazu@nii.ac.jp.
$^2$ Corporate Research & Development Center, Toshiba Corporation, akihiko.ohsuga@toshiba.co.jp.
$^3$ National Institute of Informatics and the University of Tokyo, honiden@nii.ac.jp.

Abstract. In recent years, the rapid development of network infrastructure and the spread of terminals capable of network access have made it possible to access networks at any place and anytime. Ubiquitous information systems, in which necessary information can be accessed easily and safely at any place, are becoming a more important issue. It is, however, hard to design such distributed systems when the user uses many kinds of terminals and migrates with these. That is, traditional approaches to development of distributed systems have problems when the systems are used in ubiquitous environment. This paper proposes a new framework for ubiquitous information systems. The framework includes three kinds of agents: User Interface Agents, Programmable Agents and Service Mediation Agents. We can easily design ubiquitous information systems by collaborating these agents. In addition, in cases where distributed systems must be implemented on various networks and terminals, it gives a high degree of flexibility to the systems. We also evaluate the framework from flexibility point of view.

1 Introduction

In recent years, the rapid development of network infrastructure and the spread of terminals capable of network access have made it possible to access networks not only when you are home or at your office, but also when you are away. In addition, network connections by cell phones or smart appliances have made it possible for users who have not used PC’s to use services or share information through networks. The environment in which necessary information can be accessed easily and safely at any place is called “ubiquitous information system” (UIS). The foundation for the UIS is being formed in society. Under such background conditions, various services have appeared on networks and a massive amount of information continues to be transmitted. Because of the standardization of web services and the spread of broad-band networks by which we can always connect our PC to the Internet, general users who before had difficulty providing services themselves are now beginning to be able to provide those services.
However, such web services include a mixture of services with varying degrees of reliability and quality, and using them well has become difficult. Furthermore, conventional web services are constructed based on the assumption that they will be accessed through one PC, and are not prepared for the type of usage that takes into consideration the above described network environments. Specifically, users who traverse broad-band connections at home, portable terminals away from the home or office, hot spot wireless services, and LANs at the office, have problems with today’s services, such as the presence of restrictions of available places or terminals, or no availability of continuous services when they change the place from one to another.

In this paper, we propose a new framework for development of flexible distributed systems in order to solve these problems using agent technologies. This framework achieves UIS by asynchronous communications between three kinds of agents, i.e. the user interface agent (UIA), the programmable agent (PA), and the service mediation agent (SMA).

The organization of this paper is as follows. Section 2 introduces a web information search system as an application example of the framework proposed in this paper. Section 3 summarizes the problems and challenges relating to UIS as mentioned above. Section 4 describes the details of this framework and section 5 describes the implementation of the framework. Section 6 explains how to assemble a web information search system using this framework. Section 7 evaluates this framework and then discusses the relationship with conventional systems. Section 8, the last section, summarizes this paper.

2 Application example: Web information search system

For the purpose of describing and evaluating our framework, this paper discusses a web information search system as a typical application example of UIS.

Here, a web information search system is a system that uses keywords specified by the user to search for related information on the web. Specifically, this system uses a natural language analysis service, a synonym search service, some search engines, web servers, HTML converter, content compression service, etc., on the internet. A natural language analysis service is a service that takes natural language text as an input and returns the parts of the speech and the relationship between the words; the HTML converter is a service that takes an HTML file as an input and converts the content into plain text, PDF, CHTML format, etc. We assume that some search engines and HTML converter services exist on the Internet. The server instability and usage situations of the user are the determining factors as to which ones will be used.

Typical operations for this system are as follows. When the user requests a search in the natural language form, a natural language analysis service is used to extract keywords from this natural language expression and then a synonym search is done to find related keywords. The search engine is used to find pages related to these keywords and the links from this page are followed with filtering to acquire those that meet the conditions. Finally, the HTML converter is used.
to convert the acquired contents to an appropriate format for a user’s terminal and search options, and they are put together, compressed, and then returned to the terminal.

When doing this, the user can continue the information search by using multiple terminals, such as portable terminals and PCs. In order to make this possible, it is necessary that searching and filtering can be performed on the server side. Furthermore, the results need to be retained after the completion of the search until the user requests retrieval. Also, the user’s usage situation and request can change while the information search is still being performed and the system needs to be able to cope with such changes. For example, after searches have finished, the system needs to respond to further filtering requests or changes of search conditions. Therefore, we cannot use only notification mechanisms, such as e-mail but also need communication mechanisms with user’s terminal and the system.

3 Challenges pertaining to UIS

This section analyses the problems arising when a conventional method is used to development a distributed system that meets UIS, and clarifies the challenges that should be overcome by the framework to resolve such problems.

Generally speaking, the following problems arise when using a C/S system to develop software for UIS as mentioned in section 1.

**Problem 1:** It is difficult to accommodate and port to various terminals.

**Problem 2:** It is difficult to cope with changes in the location and data structure of the service.

**Problem 3:** It is difficult to select an appropriate service each time.

**Problem 4:** It is not easy to add new type of services.

**Problem 5:** It is difficult to cope with situations in which the user environment or the service is unstable. In addition, in such situations, the usability of the software becomes poor.

**Problem 6:** Since there is no mechanism for automating the processes according to the situation, usability becomes poor for a terminal with a limited interface.

These problems are caused by the fact that the software on the client side includes the details of access to services on the server in addition to the user interface.

Problem 3 and the changes in the location of services mentioned in problem 2 can be resolved by separating the service selection portion from the client by using a Broker pattern[3] or a facilitator that mediates an appropriate service. However, the other problems cannot be coped with by just separating the service selection component as an independent structure.

In order to use different services and terminals depending on locations and situations, the following problems need to be resolved.

**Problem 7:** It is difficult to create functions to cope with various situations in advance.
Problem 8: It is difficult to cope with changes in situations on the user’s side.
Problem 9: It is impossible to cope with changes in the terminal that the user is using. Therefore the process cannot be continued over multiple terminals.

For problem 9, a directory service can be used to determine which terminal the user is using. However, even in such a case the process cannot be continued on multiple terminals.

Furthermore, the following problems need to be resolved in order to use services without worry.

Problem 10: When the user wants to reuse existing local services, such as intranet services, in ubiquitous open environments, it is difficult to add security functions to the services.

Problem 11: It is difficult to protect user information.

The Three-tier Model[1][2] is a software structuring method that separates the logic portion that implements the software functions from the client portion for the purpose of overcoming various problems arising in the C/S system. In this model, the structure of the program in the distributed system is divided into three layers, i.e. the client layer, the domain layer, and the database layer (Fig. 1 (a)). The client layer is a program that directly receives instructions from the user, and the domain layer is a program that processes and manages applications and transactions. This layer also executes the processing requested by the user and mediates between the user and the data to be processed. The database layer manages various data, such as customer information and employee information, and directly accesses the data when requested by an application.

Some of the aforementioned problems can be resolved by using this Three-tier Model architecture. Having the logic portion be independent increases the portability of the client portion, which makes it possible to solve problem 1. Also, the introduction of the intermediate layer increases the degree of independence between the client and the service, which, compared with conventional ways, makes it easier to do things such as coping with the changes in the service structure in problem 2, adding new services in problem 4, and adding security
4 Multi-agent framework: Mobeet Framework

4.1 Overview

The framework proposed in this paper is configured with three kinds of agents (Fig. 1 (b) and Fig. 2). These software agents are expanded versions of the components in the three layers of the Three-tier Model described in section 3. In this paper, agents corresponding to these layers are called the User Interface Agent (UIA), the Programmable Agent (PA), and the Service Mediation Agent (SMA). An overview of each agent is given below.

UIA: The UIA provides an interface for the user. In addition, it detects the user’s situation and copes with his/her various situations. Upon receiving the user’s request, it considers the situation at the moment and outputs a script that carries out the user’s processing. For the purpose of achieving these functions, the UIA contains a logic language processing system (hereafter referred to as “inference engine”). In order to cope with various situations, the system designer describes behavior of the agent corresponding to such situations by using logic language rules (hereafter referred to as “reasoning rules”).

This inference engine is capable of outputting a script and the UIA can execute this script. Using this capability, it inspects the current situation and the user’s request as a goal, carries out inference, and it can output a script as a sequence of actions to achieve the goal. For example, in the case of the web
information search system wherein the user’s request is a restaurant search, it infers that restaurants serving lunch in the vicinity should be searched for, and then outputs a script for this search. The output script is executed as a PA described below.

The reasoning rules defined at the UIA are event driven so as to be able to cope with changes in the situation. That is, in addition to when there is a direct request from the user, relating rules are reevaluated when the values of the predicate defining the situation has changed. This mechanism makes a system autonomous in coping with changes. Using this mechanism, for example, when a user in the outdoors goes back to his company office in the middle of an information search, the system may sends the results to the company’s computer.

**PA:** The script (PA) output from the UIA is executed as a mobile agent migrating between hosts. The PA copes with terminal instability and changes in the using terminal of the user. This is because the agent can execute the actions after migrating from the UIA onto a server host called an “agent pool”. The PA communicates with other agents by using the Agent Communication Language (ACL[4][5]). This allows for an increase in independence between the agents and prevents unnecessary leakage of user information by increasing the autonomy of each agent.

**SMA:** The SMA is an agent that actualizes the request from the PA. This agent is dynamically synthesized at the agent pool to cope with diverse services and requests. Components that describe logic for using web services are registered in the agent pool and the agent pool responds to requests from the PA by synthesizing an SMA that is a group of components that meet the request. For example, upon receiving a web information search request in the form of a natural language input from the PA, the agent pool synthesizes an SMA to actualize the request by combining components that generate search keywords from the natural language and components that search web information using the search keywords.

Thus synthesized, the SMA is then executed by the framework. Multiple components having the same specifications can exist; the mechanism is such that an error resulting from calling a component will lead to calling another component having the same specifications. For example, when there is no response from a search engine and an error occurs, this mechanism automatically copes with this error by calling a component that uses a different search engine.

Fig. 3 is a UML representation of the collaboration between these three agents and the agent pool. All the agents have a mechanism for sending/receiving asynchronous messages and can communicate with each other by using the ACL. Typical operations for this framework are as follows.

1. The user sends a search request to the UIA. Using this as a goal, the inference engine of the UIA is called. In the case of a web information search, the goal is a search request whose arguments are the search keywords
2. The framework of the UIA defines the current state as a predicate and then infers the goal. At this time, a script (PA) is output from the executed reasoning rules and is then executed. (Corresponding to 1 in Fig. 3)
3. The PA sends a request to the agent pool. (Corresponding to 2 in Fig. 3)
4. The agent pool generates an SMA according to the PA's request. (Corre-
sponding to 3 in Fig. 3)
5. The PA makes the actual request for the SMA generated by the agent pool.
   (Corresponding to 4 in Fig. 3)
6. The SMA coordinates web services to perform the service and return the
   result back to the PA. (Corresponding to 5 in Fig. 3)
7. The PA sends the result from the SMA to the UIA specified by the user.
   (Corresponding to 6 in Fig. 3)
8. The UIA presents the result to the user in the manner suitable for the
   situation of the user and the terminal.

By communicating with each other, these three kinds of agents can flexibly
cope with changes in the user environment.

4.2 The User Interface Agent copes with situations

The UIA has the role of accommodating users in different and various situations
and coping with changes in their situations, as well as the role of protecting user
information.

The user’s situation depends on time and location, as well as the terminal
environment, such as network connection situations. For the purpose of using
these environments to infer the user’s intention, the framework is equipped with
a mechanism that monitors the terminal environment and provides information
in the form of predicates. Specifically, the time, location, and network connection
situation can all be looked up as situation dependent predicates (environment
predicates).

For the purpose of calculating the user’s intention and situation from the
terminal environment, the framework is equipped with an inference mechanism
using a logic language. Its reasoning rules include scripts that are outputted
when the applicable conditions and rules are met.

The format of a rule is “Head:- Guard| Body{Script}.”. The grammar
of the head and the body in the rule is the same as that of the Prolog. The
guard part is the same as the GHC (Guarded Horn Clauses[6]) of the parallel
logic language, and the conditions for applying this rule are described as an
arithmetic equation.

The inference engine is activated when a new goal is thrown with a request
from the user or when a new goal from the PA is thrown. In addition, whenever
this framework changes the environment predicates, inference is performed so
that the behavior can be determined in response to changes in the situation.

4.3 Execution of the request by the programmable agent

The PA executes the request from the user while coping with instability in the
user’s terminal. It also copes with switching the terminal that the user is using.
In order to do this, a script contains instructions for autonomously migrating between hosts.

Potential causes of terminal instability include power loss and network disconnection due to changes in the location of users. For example, in the case of a web information search, the user may turn off the portable terminal or the network may be disconnected because the terminal has changed its location. In order to cope with such instability, the PA transfers the necessary information to the agent pool after receiving it from the UIA. The PA then executes the given request.

The script includes abstract service request to realize user requests. What actualizes the abstract service is an SMA, which is explained in section 4.4. For this purpose, the script is equipped with functions to search for necessary services and obtain their addresses. The actual search for services and the generation of the agent (SMA) that executes the service is performed by the agent pool. The PA searches for necessary services and asks the SMA thus found to perform the service by sending a request message in the ACL. For this purpose, the script is equipped with commands for communicating with any agent using the ACL.

The PA uses the ACL to send user information to the SMA. By using different performatives, the user information presentation request from the SMA can be expressed not only as an unconditional request but also as a conditional request specifying added benefits obtained by presentation. In response to the user information presentation request, the PA can communicate an intention of rejection by using “refuse” as well as an intention of presentation by using “inform”.

4.4 Actualization of the service by the service mediation agent

The SMA is synthesized by the agent pool and executed automatically. In the case of a web information search, various requests are possible depending on, for example, whether the related information is specified by a natural language or by a string of keywords. If all services are fixed in advance to cope with such diversity, it is not possible to flexibly cope with request changes and additions. In view of this, the present framework copes with this problem by defining a necessary service as a group of components (service components), synthesizing it by automatically combining these components, and then executing them one after another. To this end, for each service, the input specifications and the output specifications are defined as types with a meaning, in addition to the service name to be implemented. When a service search request comes from the PA, the service that is suitable for the request is synthesized from a group of components having the specified service name.

For example, in the case of a web information search service, the service component for generating keywords from a natural language is defined by the following specifications:

<table>
<thead>
<tr>
<th>Service name: Information search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: String:Natural language text</td>
</tr>
<tr>
<td>Output: list of String:Keywords</td>
</tr>
</tbody>
</table>
This means: the input/output specifications are defined in the “type:meaning” format, the input of the aforementioned component is String type natural language text, and the output is keyword expressed as a String type list.

When components with such specifications are found, the agent pool collects a group of service components that can provide output from the input specifications of the service requested by the PA and designates this as the SMA. These service components are assembled by using one or more web services.

The SMA is executed by the framework in the following manner. First, the framework receives data from the PA that meets the input specifications of the service and executes a component that uses the data as input specifications. When the output of the component is calculated, another component that has the necessary inputs is executed. Finally, when all the outputs requested by the PA are obtained, the obtained values are returned to the PA by using the ACL. If the service components to be executed run out before all the outputs are obtained, then an ACL failure message is sent.

Each service component can use the API for migration in order to cope with instability in the services used. The migration request in a service component implies migration of the group of components that constitutes the SMA, and the execution of the components is continued at the migration destination. Instability of services can be due to instability in the server or instability in the network; in the former case, the service is called remotely. In the latter case, after the migration of components, the service is used locally.

4.5 Coping with changes by means of inter-agent communications

When the situation of the user changes, the UIA detects the change and alters the request that is already being executed. The PA responds to this change by altering its own behavior using the re-inference (re-planning) function. The change is propagated by the ACL.

An example of a change in the user situation is a change due to a location change of the terminal when the user who has been away comes back to the office. This change can be inferred by the change in the environment of the terminal that the user is using.

The UIA has the function to infer the user’s situation when the environment, such as the time and location, has changed. Specifically, a change in the environment results in redefinition of the environment predicates and the reasoning rules are reevaluated. Based on this inference, a new script corresponding to the change is generated to cope with the change.

This coping action includes altering the request as well as altering parameters such as search options. For example, when the user comes back to the office, changes are made so the user can view the search results at the terminal in the office. Such changes in the request are managed by using the PA’s re-planning function as described bellow.

That is, the new request is sent by means of the ACL to the already executing PA; upon receipt of this, the PA activates the inference engine, using this request as a new goal (re-planning). If this inference needs user information, migration
to the terminal occurs. The inference for the new goal is conducted here and a new script is output. This script becomes the definition of the new behavior of the PA.

5 Implementation of the framework

This framework is implemented by using Plangent[7] for the UIA and PA, and using Bee-gent[8] for the SMA. Although Plangent has a unique feature which is that it generates scripts being agents behavior from Prolog programs, the scheme of our framework cannot be implemented in its original form by Plangent. Specifically, Plangent has limitations in terms of the activation of the inference engine and the description of the reasoning rules. The following is the format for rules (action definitions) in Plangent.

\[\text{action}(_, _, _, \{\text{script}\}, \{\text{Precond}\}, \{\text{Postcond}\}, \{\text{Manifestation}\}) \leftarrow \text{Body}.\]

Here, the list of postconditions (Postcond) corresponds to the goals and the inference engine is done in such a way that the preconditions (Precond) are met. The body part is the same as common Prolog. The user information called “info file” can be accessed at the manifestation string to view/update the information.

The inference engine of Plangent is not event driven; it is driven when a goal is put in. Furthermore, since the guard cannot be described, the rules to be inferred cannot be determined without a goal. In order to cope with this, a dummy goal “response” is given when the situation changes\(^4\) so as to activate the inference engine. Also, conditions corresponding to the guard are described at the beginning of the body of this predicate so that an appropriate action definition can be selected.

In addition, Plangent does not support ACL. Therefore, a mechanism for sending/receiving an ACL used by Bee-gent is prepared, so that the ACL can be read/written from inside of a script and also from the UIA to communicate any other agents.

A Plangent script has a function to rewrite itself (newgoal), which can be used to cope with changes in the user’s situation. When a notification of changes in the user’s situation is received while the script is executed, the situation is written on the agent’s own information (agent info) and the newgoal is executed to regenerate itself.

The service components of this framework are made to correspond to state classes in Bee-gent. Bee-gent has a mechanism that executes a specific state (mental state) when necessary conditions are satisfied. Using this mechanism, all the service components are defined as mental state classes, and the input/output specifications of the framework are made to correspond to preconditions/postconditions of Bee-gent. The planner of Bee-gent is used to simulate the SMA’s operations. For each mental state class, this planner mechanism can set goals for which the class is activated and for the goals to be executed next. The number of state classes that can be executed simultaneously is set to 1, so that state

\(^4\) Changes in the environment are checked at a time interval in the UIA.
classes having the same precondition are selected one after another by changing
the priority (utility value) for goal selection.

For the purpose of picking up a group of SMA service components, the input/output specifications of each state class are managed by a definition file that is different from Bee-gent. In the external command of Plangent, for a service search, this file is looked up and necessary service components are collected to generate Bee-gent mediation agents.

6 Implementation of a web information search system

Using the framework described in section 4, a prototype for a web information search system was implemented. An overview of the implementation is described below.

The action definition of Plangent can be used to define rules for inferring the user situation and rules for generating scripts. Fig. 4 is a portion of the action definition. The user’s situation in terms of whether or not he is away from the office is inferred by “place” in the body of this definition: if the user is away, “outdoorOption” sets optimum options. When this action definition is applied, a PA calling “goAgentPool”, “webSearch”, “convert”, and “sendResults” is output. This PA migrates to the agent pool (goAgentPool), searches web information
A web search engine service, a content conversion service that converts the HTML to various formats, and data compression service are newly assembled for this system. Web services used for the web information search service are of four kinds, i.e. a natural language analysis service, a synonym search service, a search engine, and a web server. Specifically, these web services are used to prepare the SMA’s service components. Table 1 shows definition examples of the input/output specifications of the SMA service components. Among these, the service component that generates keywords from a natural language (KeywordGenerator) is implemented by combining two web services, i.e. a natural language analysis service and a synonym search service.

7 Discussion

7.1 Evaluation of the framework

This section describes how the framework presented in this paper meets the challenges mentioned in section 3.

Meeting challenge 1: This framework can cope with instability on the terminal side by transferring the PA to an appropriate server. This transfer control needs to be described in the PA generation rules that the UIA possesses, and the programmer has to specify it explicitly. Instability on the service side is handled by transferring the SMA, or replacing the service component to be executed. Specifically, it is possible to cope with network instability by transferring the SMA to the host providing the service or to a host experiencing stable communications. Upon error termination of the service component calling the service remotely, the framework can execute an alternative component and thus cope with a service down in a semi-automated manner.

Meeting challenge 2: This framework copes with diverse requests by dynamically generating the request details and the usage order of the services. Since the response to the user’s requests are defined and added by the reasoning rules, the existing definitions do not have to be changed for a new request.

There can be multiple combinations of services for the same request; this can be handled by SMA’s service component selection mechanism by means of automatically calling different service components having the same input type. The addition of a new service can be handled by simply registering its components without altering existing components.

Meeting challenge 3: This framework contains an inference engine in the UIA as a mechanism for automating various processes according to the situation. Also, predicates for checking the terminal environment are made available for evaluating the situation.
In order to carry out processing in the place of the user, a script is output as the result of the inference and then made executable. Using this mechanism, batch processing, such as format conversion of all the collected texts at once, can be performed. If this processing fails, another processing can be selected by means of re-planning. However, when the reasoning rules for the selection are on the terminal side, returning to the terminal is required. If the terminal is not connected at that time in this case, the processing cannot be continued. That is, the managing method for the reasoning rules of the UIA is limited.

**Meeting challenge 4:** In order to cope with changes in the situation on the user side, situations and instructions corresponding to them are described as reasoning rules and asynchronous communications between agents are made possible. For such reasoning rules, the event driven logic language is adopted to allow inference responsive to the situations. Also, the ACL is made available for sending/receiving messages for smooth cooperation between the agents. Furthermore, re-planning is introduced to the PA as a mechanism to alter its behavior in response to changes.

When re-planning involves the necessity to cancel existing processing, scripts have to be carefully custom-made. For example, when there is an interdependent relationship between a series of PA requests, such as in transaction processing, it is difficult to define the precise scripts.

**Meeting challenge 5:** In this framework, the PA is designed as a mobile agent and continuous processing over multiple terminals is achieved through the PA carrying around the processing states. For example, the PA that makes a search request is generated at the portable terminal and placed on standby on the server side; another PC can call back this PA and the results can be viewed at another terminal. This way, the user side can provide processing, such as filtering of the search results, in the form of a script.

**Meeting challenge 6:** This framework supports the use of the UIA’s inference engine to infer the user situation and supports flexible exchanges of user information by means of the ACL. Because of these, the system can be designed in such a way that user information is treated appropriately.

When user information is needed to use a service, an information query message from the SMA to the PA can be used. Although this is not used in the example, for nonessential information, the message “user information will be requested if you need to know this information” can be expressed by using a query-if message. In response to a query, the example is implemented in such a way that the PA returns messages such as “inform” or “refuse”. However, a scheme, such as content language, for this purpose is not defined, which increases the programmer’s burden if complex information needs to be exchanged.

In addition, the framework uses user information defined at the UIA to derive appropriate actions for the user at the terminal side, thus maintaining higher security of user information compared with the case in which the same processing is done at the server side.
7.2 Functional comparison with conventional methods

In this section, we compare our framework with a PAC agent framework\cite{9} from the viewpoint of the UIS.

The PAC agent framework is a framework for a distributed system that configures the system with three kinds of agents, i.e. the Top-level, Intermediate-level, and Bottom-level. These agents cooperate with each other by using three interfaces called “Presentation”, “Abstraction”, and “Control” to constitute a flexible distributed system. These three interfaces increase independence between the agents and secure flexibility in terms of portability and configuration. Fig. 1 (c) shows the structure of this framework. The top-level agent accesses the repository and is in charge of data exchanges necessary for the system. The bottom-level agent allows users to display and manipulate the results, and the intermediate-level agent mediates between the bottom-level agent and the top-level agent and sends necessary data from the top-level to the bottom-level.

In principle, this framework is designed to provide various displays and manipulations to one resource, and the three kinds of agents constitute a tree-like structure with the top-level being the root. However, this structure does not allow flexible responses in application domains where various services exist. Even if the Broker pattern is used to introduce a mechanism for selecting appropriate services, flexibility such as altering the procedures for cooperation between services cannot be implemented. Our framework dynamically derives the behavior of the SMA by synthesizing components and selectively executes the components. This makes it possible to cope with a diversity of services and instability in the servers.

The PAC agent does not specifically prescribe where each agent is executed, how cooperation is done, and a mechanism for autonomy inside an agent. Therefore, when assembling a flexible system by making the agents cooperate with each other, it is hard to design it. In addition, the instability of servers or clients and dynamical changes of user’s situation are not taken into consideration; i.e. there is no support for these in the framework. In comparison, our framework focuses on a domain of UIS; it prescribes not only the roles of the three kinds of agents but also the mechanisms to perform these roles. Specifically, a combination of an asynchronous communication mechanism, ACL, inference engine, agent transfer mechanism, PA’s re-planning mechanism, etc. allows this framework to easily cope with cooperation between the agents, instability in the servers, and changes in the situations.

7.3 Related work

Our framework provides more flexible than conventional C/S system, Three-tier Model and PAC Model. There are other architecture styles and design patterns of these models. Some patterns are proposed for flexibility. For example, \cite{3} describes Broker pattern, Reflection pattern, Client-Dispatcher-Server pattern and so on. In addition, \cite{9} describes Strategy pattern. In contrast, we can regard our framework as a pattern for ubiquitous environment. Our framework is more
flexible than only use of conventional patterns. In addition, the conventional patterns, such as Broker pattern, are used as complementary to our framework.

BDI agents and distributed multi-agent systems have been proposed to cope with changes of environments flexibly [10–12]. In the case of BDI agents, we can consider that an environment model is constructed as Belief and appropriate behavior is derived using inference engine. In the case of almost all distributed multi-agent research, agents negotiate each other with coordination relation to make the whole system flexible. However, it is difficult to design a system responding to various changes, because we need to make each agent with high intelligence individually. On the other hand, our framework consists of three kinds of agents having each role and the agents cooperate with each other keeping each role to realize flexibility. Consequently, it is easier to make a design plan and to design the system using our framework than the case of other multi-agent system.

Some mobile agent platforms[13,14] have been proposed for mobile computing, especially for resource-limited devices. LEAP is a FIPA-compliant agent platform sufficiently lightweight and powerful to execute both on mobile devices and on enterprise servers. The extentions also have been proposed for nomadic users[15], for ad hoc networks[16] and for handheld devices[17]. Although these platforms and extensions are for ubiquitous computing infrastructures, no support for application level design using such agents is provided. However, we can adapt the platforms for UIA’s mobility to support nomadic applications.

Automated or semi-automated composition mechanisms of Web Services have been proposed[18–20]. [18] uses DAML+OIL, which is a description logic-based language, for describing the formal semantics of Web services. The mechanism described in [18] can compose the sequence of Web services satisfying a user’s goal using each service semantics. We can use such mechanisms to synthesize SMAs. In other words, our framework is consistent with Web Services as service parts of SMA.

8 Conclusions

In this paper, we proposed a new multi-agent framework to make ubiquitous information service available easily. This framework is for distributed system consisting of three kinds of agents: UIA, PA and SMA. The UIA monitors the client environment and copes with changes in the situation on the user's side. The PA is a mobile agent that is created from reasoning rules and executes a series of requests from the user. The SMA is an agent that is composed from service components based on specification of a requirement to provide a service for PA. PA responds to change of user terminals and instability of user environment, and SMA responds to instability of server side.

Further work includes construction of various systems for UIS to elaborate our framework and CASE tools to develop the system easily.
References

9. E.Gamma, R.Helm, Rohnson and J.Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley Longman, Inc. (1997)
16. J. Lawrence, “LEAP into Ad-Hoc Networks”, Workshop on Ubiquitous Agents held in AAMAS’02 (2002)