Agent System Development Method Based on Agent Patterns

Yasuyuki Tahara, Akihiko Ohsuga and Shinichi Honiden
Systems and Software Research Laboratories,
Research and Development Center, Toshiba Corp.
70 Yanagi-cho, Saiwai-ku, Kawasaki, 210-8501, Japan
Tel: +81-44-548-5474 Fax: +81-44-520-5855
{ytahara,ohsuga,honiden}@ssel.toshiba.co.jp

ABSTRACT
As wide-area open networks such as the Internet and intranets grow larger, agent technology is attracting more attention. Agents are units of software that can deal with environmental changes and the various requirements of open networks through features such as autonomy, mobility, intelligence, cooperation, and reactivity. However, since the usual development method of the agent systems is not sufficiently investigated, the technology is not yet widespread. This paper proposes a method of agent system development based on agent patterns that represent typical and recurring structures and behaviors of agents. The agent patterns are classified according to their appropriate architectural levels and the degree to which they depend on specific agent platforms. Our method enables developers to design agent systems efficiently since they can construct complicated system architectures and behaviors by dividing the design process into two architectural levels and applying the appropriate agent patterns. In addition, the higher level designs are independent of specific agent platforms and can be therefore easily reused.

Keywords
Agent, mobile agent, design pattern, system development

1 Introduction
As wide-spread open networks such as the Internet and intranets grow larger, demand for distributed systems with rapid changes and diverse requirements is getting higher. Agent technology [2] is expected to be a basis for the development of such systems. Agents are software units that can deal with environmental changes and diverse requirements with their features such as autonomy, mobility, intelligence, cooperation and reactivity. Especially, mobile agents such as Aglets [7] and Flagent [11, 3] are expected to be the most practical. Since practical applications of agents are investigated recently, techniques for agent system development support has appeared. For example, a unified agent architecture [10] and development method using object-oriented methodology [9] has been proposed. However, these existing methods have many problems in development efficiency, applicability, extensibility and understandability that are considered necessary for development methods. Therefore these methods are hardly practical.

Also recently, design patterns [5] and software architectures [6] are attracting attention in usual software development represented by object-oriented software. Design patterns are explicit formulations of good past software development experiences consisting of proven solutions to recurring problems that arise within some contexts and systems of forces. The patterns realize easy reuse of good software design. Software architectures represent structures of software design concerned with whole systems beyond data structures and algorithms. These concepts are becoming indispensable in developing practical large-scale software in a low cost by promoting reuse.

Against this background, developers have begun to apply design patterns to agent system development [1, 8, 12]. However, these researches only describe various patterns as pattern catalogs and lack the viewpoint of how to locate each pattern in the actual system development process.

In this paper, we propose an efficient development method of agent systems using patterns. Because most of the patterns used in our method are involved with agent behaviors, our patterns are called behavior patterns. In our method, the system architecture is divided into three layers considering the relationship between the system and the agent platform used there. Then the behavior patterns are used in each layer in order to make it easy to design each layer. The layers are designed from the higher one to the lower using the behavior patterns. An extra development phase should take place before the design in order to extract the application conditions of the patterns. As a result, our method consists of the phases for extracting the conditions and designing of the three layers. Our method classifies the patterns into three categories according to the design phases in which the patterns are applied. Since the design results of the highest level do not depend on specific agent platforms, they can be easily reused. Therefore our method can solve the problems specific to agent system development.
This paper is organized as follows. Section 2 describes problems of agent system development. Section 3 explains the development method we propose in this paper in detail. We also present the agent patterns in the section. In Section 4, our method is evaluated using some examples. In Section 5, our method is compared with other related work. Section 6 summarizes this paper and refers to some future work.

2 Problems of Agent System Development
In this section, problems of agent system development are described. First, features of agent systems are explained in comparison with usual systems. Second, problems of agent system development are clarified.

Features of Agent System
In general, agents have a very wide meaning. Mobile agents, cooperative agents and interface agents are widely believed to be representative. Among them, the former two kinds are considered to be effective in open networks. Therefore the agent systems investigated in this paper should have mobility and cooperativity. In addition, copying ability is a common functionality of most practical agent systems. Mobility is the functionality for efficient use of information resources presented in wide-area open networks by moving around the networks and accessing resources. Cooperativity is the functionality by which multiple agents communicate information and perform a task or agents with different tasks perform them, solving any conflicts. By the copying functionality, agents can create other agents with the same programs of the original agents.

The authors developed Plangent [11, 3] as an agent platform with mobility, cooperativity, and also planning functionality that enables agents to behave flexibly. In Plangent, users give their requirements to the agents as goals. Then the agents generate plans that satisfy the goals by the planning functionality. The generated plans generally include moving actions by planning with uncertain knowledge about remote nodes. It is probable that the agents fail to execute the plans because the uncertain knowledge is false or old and invalid. In this situation, the agents generate new plans (replanning) by using new knowledge at the current nodes and therefore they can deal with the situation flexibly.

Problems of Agent System Development
Agent technology is highly expected as a solution to the problems of wide-area open network environments because of the advanced features of the technology. However, it is currently difficult to develop systems efficiently because of the following reasons.

- One point is that it is more difficult to grasp behaviors of mobile agents than the usual software. When the agents start to act in remote hosts, the behaviors of the agents correspond to the environments of the hosts. Since it is not easy to grasp the environments, the difficulty results also for the agent behaviors. Therefore the understandability of the systems tends to decrease, development of the systems tends to be ad hoc, and the reliability and the extensibility tend to be low.

- Because it is not a long time since the mobile agent technology has appeared, platforms are rapidly being improved currently. Therefore it is difficult to maintain the systems according to version-up of the platforms where the systems are built on some specific platforms of some specific version. It is also difficult to coordinate systems built on different platforms.

3 Agent System Development Method Based on Patterns
In this section, we propose an agent system development method aimed to solve the aforementioned problems characteristic of agent system development and enable practical system construction. The features of the method are summarized as follows.

- The system architecture consists of the following three layers in order to establish a systematic development method. The first layer is the macroarchitecture that represents the outline of the system configuration and the agent behaviors and has such generality that the layer is not dependent on any specific agent platforms. The second layer is the microarchitecture that represents the detail of the system configuration and the agent behaviors specialized in each agent platform. The third is the object level that represents the program codes implemented according to the design of the above two layers.

- The following behavior patterns corresponding to the individual layers are presented in order to make the development of the layers easy.

  - Macroarchitecture Pattern
    - This kind of pattern is applied in the development of the macroarchitecture. Therefore patterns should be selected as they are not dependent on any specific agent platforms. The patterns are classified into the Mobility Patterns and the Cooperation Protocol Patterns.

  - Microarchitecture Pattern
    - This kind of pattern is applied in the development of the microarchitecture. Therefore patterns should be selected as they are specialized in the individual agent platforms and make good use of the advantage of the platforms.

  - Object-Oriented (OO) Design Pattern
    - This kind of pattern is the usual ones for implementation using OO languages such as Java and C++. One of the representative groups of such patterns was investigated by the Gang of Four (GOF) [5].

- The layers are designed from the upper one to the lower one using the behavior patterns. In addition, a development phase for eliciting the condition of the behavior pattern application takes place before the design. Therefore our method consists of the
four phases including the pattern requirements extraction phase and the design phases for the three layers.

By our method, the problems of mobile agent system development can be solved as shown in the following.

• You can easily develop the system by comprehending the system behaviors because the behaviors are visualized by the patterns.

• The essence of the application software and the details of the system involved in the platforms are separated by the distinction of the macroarchitecture and the microarchitecture. This point results in the following advantages. First, the essence of the application software, that is, the logical configuration of the systems is explicitly recognized. This part of the design can be reused on, for example, the system version-up, and therefore it is easy to cope with such changes.

• Because the descriptions of the essence of the applications have such generality that they are independent of agent platforms, they can be expected as bases of coordination of heterogeneous agent platforms.

**Layered Agent System Architecture**
In our method, the layered agent system architecture is defined in order to investigate a systematic development process. The details of the layers are as follows.

• **Macroarchitecture**
  The macroarchitecture represents the design specification of the outline of the system configuration and the agent behaviors described using the mobility and the cooperation that are the basic functionalities of the agents. This layer has the feature of such generality that the layer can be used for any agent platforms with the two functionalities. This layer is designed by applying the macroarchitecture patterns.

• **Microarchitecture**
  The microarchitecture is the design specification that represents the detailed system configuration and the agent behaviors specialized in each agent platform. As for the granularity, this layer is described partly by program codes and partly by specification formats such as natural languages and diagrams. In this layer, the part of the macroarchitecture supported by the agent platform is refined and the other part such as the interface to the external applications that should be individually refined is implemented in the following object level. This layer is designed by applying the microarchitecture patterns prepared for the individual agent platform.

• **Object Level**
  This layer is the program code implemented according to the specifications of the macro- and the microarchitecture. This layer is designed and implemented using, as occasion demands, OO design patterns such as those of GOF and OOA/OOD methods.

Figure 1 summarizes the layered architecture.

![Layered Agent System Architecture](image)

Figure 1: Layered Agent System Architecture

The macroarchitecture consists of the following elements.

• Representation of the typical scenarios of agent behaviors in networks as agent behavior diagrams: The agent behavior diagrams describe the physical configurations of the distributed systems such as the host and network configurations and how the agents behave in the configurations.

• Representation of the whole general agent behaviors as state transition diagrams: The state transition diagrams describe information about the hosts which the agents are located and the agent's task status state boxes and information of agent migration and local behaviors as transition arrows.

• Representation of temporal relations of each action as event sequence chart: The event sequence chart consists of the following components in detail.
  - Host lifeboxes that are denoted by thin rectangles. In each box, the temporal behaviors of the applications and the agents that are local in one host are described.
  - Local lifelines that are denoted by vertical lines in the host lifeboxes. They represent the temporal behaviors of the applications and the stationary agents that work in the hosts locally and the mobile agents that are working temporarily in the hosts during the migrations.
  - Event arrows that are denoted by horizontal lines between the local lifelines. They represent the interactions between the applications, the agents, and both of them that are local in the hosts or occur between the hosts via networks.
Agent migration arrows that are denoted by horizontal lines between the local lifecycles of mobile agents across the host lifeboxes. They represent the agent migrations between the hosts.

Figure 2 illustrates these components.

![Diagram](image)

Figure 2: Components of A Macroarchitecture

The structure of the microarchitecture for each agent platform differs. As for Aglets, for example, because the agents are described only in Java codes and the platform is provided only in the form of a class library, the microarchitecture is described in specification format of usual OOA/OOD such as the UML [4] or directly in Java codes. As for Plangent, agent behaviors are described in Plangent Script, the script language specific for the platform. Interactions between the agents and the external worlds are described in Java in the similar way as Aglets. Figures 3 show the layered architecture of Aglets and Plangent from the viewpoint of the behavior patterns.

![Table](image)

Figure 3: Layer Architecture of Aglets and Plangent

**Detail of the Behavior Patterns**

This paper proposes that agent system development can be made effective by applying behavior patterns corresponding to each design phase in that phase. The patterns are explicit formulations of good past software development experiences consisting of proven solutions to recurring problems that arise within some contexts and systems of forces. The effect of the behavior patterns to agent system development is similar to that of the OO design patterns to OO system development.

In our methods, the patterns are classified and applied according to the layered architecture. Such structure is explained as follows (see Figure 4).

![Diagram](image)

Figure 4: Relationship between the Patterns and the Architecture

- Each layer is designed by combining the components that are instances of the behavior patterns corresponding to the layer.
- The patterns for the lower layer refine those for the higher layers.
- The lower layer is designed by refining the design specifications of the higher layer.

In general, patterns are described using pattern templates with more than 10 features. In this paper, owing to limited space, only the summaries of the patterns are described. In addition, an example of the Mediation Pattern application is illustrated.

**Macroarchitecture Patterns**

Macroarchitecture patterns are general ones independent of agent platforms and are classified into two categories: mobility patterns and cooperation protocol patterns.

- Mobility Pattern: The patterns applied to the situations where mobile agents perform tasks while moving around networks. By using these patterns, the efficiency is improved because the networks are used only for necessary data and agent movement and the computation resources are used locally. Therefore these patterns are effective if the amount
of the data that should be communicated through the networks is small.

The mobility patterns are further classified into the basic ones and others. The basic mobility patterns represent the patterns of one migration between two hosts and the others represent the ones of multiple migrations among more than two hosts. The others include Itinerary, Star-Shaped and Branching. Branching has two variations explained later. In addition, combined mobility patterns composed by several of these patterns are useful.

The conditions of use of these patterns can be summarized as follows. The detail of each condition is explained later in the description of each pattern.

- The basic mobility patterns are appropriate for only two hosts and the others for more than two hosts.
- Then the stability of the hosts should be considered. Itinerary and Star-Shaped are appropriate for stable hosts and Branching for unstable hosts.
- Selection between Itinerary and Star-Shaped is judged from the amount of the carried data. Itinerary is appropriate for a small amount of data and Star-Shaped for a large amount.

- Cooperation Protocol Patterns: In these patterns, multiple stationary agents cooperate by exchanging messages. This type of patterns are classified according to the protocols. Representative protocols includes the contract net protocol, the (a)synchronous backtracking and the multi-stage negotiation. These protocols are selected according to the requirements of the applications. For example, the (a)synchronous backtracking is applied to constraint satisfaction problems and the selection between synchronism and asynchronism is judged considering the requirements for performance, and the contract net protocol is applied to the problems such as load balancing. The cooperation protocol patterns are also classified to Direct Interaction, Mediation and Dispatching according to the ways of message exchange and agent arrangement.

Next, the detail of each pattern is explained.

- Basic Mobility Pattern: This is the pattern of one migration between two hosts. This pattern is classified further into three types according to the ways of execution continuation of migrating agents. The types are No Continuation, Shallow Continuation, and Deep Continuation. In addition, since it is important for mobile agent systems to be given security and safety mechanisms, their treatment is included in the Pattern. As a result, as Figure 6 shows, the Basic Mobility Pattern has $3 \times 2 \times 2 = 12$ variations.

While these patterns have the same agent behavior diagrams, they are distinguished by their state transition diagrams and event sequence charts.
Purpose and Applicability: This pattern is applied for development of distributed systems when the hosts are connected to networks with relatively narrow bandwidth. This pattern is also used as a component of the other mobility patterns.

- Itinerary Pattern: Agents itinerate around the destination hosts and perform tasks in each host.
  
  Purpose: This pattern is applied when the user wants to use the resources on the network with relatively narrow bandwidth.
  
  Applicability and Forces: This pattern is effective if the hosts that have the necessary resources are specified in advance and all hosts are stable. However, if the amount of data the agents carry during the itinerary is large, the pattern is inappropriate since the network traffic increases.

- Star-shaped Movement Pattern: Agents get there and back between necessary hosts and base hosts and perform their jobs in the destination hosts.
  
  Purpose: To meet the requirements using resources around networks in a situation where the bandwidth of the networks is relatively broad.
  
  Applicability and Forces: This pattern is applied if the hosts where the necessary resources exist are fixed in advance and each host is relatively stable, but the user wants to use as confirming the status of each resource.

- Branching Pattern: Agents generate their copies as many as the number of hosts that it wants to use. After that, each copy moves to the hosts, accesses the resources, and then goes back to the original host, vanishes, or stays there. If the copy goes back, the merger pattern described later is usually combined. This pattern has according to the way of merging the results of the multiple agents that have returned. In the synchronous merger variation, the results are unified on the merger. In the selection pattern, the result of only one agent is reserved and the results of the other agents are abandoned.
  
  Purpose: To meet the requirements using resources around networks in a situation where the bandwidth of the networks is relatively broad.
  
  Applicability and Forces: This pattern is applied if the hosts where necessary resources exist are not fixed in advance or if they are fixed but are not stable and the requirements can be satisfied if only a part of the resources are available. In addition, if the amount of data that should be communicated through the network is relatively large, this pattern can alleviate the network traffic.

  These mobility patterns other than the Basic Mobility Pattern are illustrated in Figure 8.

- Cooperation Protocol Pattern: Cooperative behaviors by interaction of multiple agents are investigated as cooperation protocols in the area of distributed AI for a long time. In this paper, these cooperation protocols are considered as macroarchitectures and applied in the macroarchitecture design phase. The cooperation protocols include the contract net protocol, the multistage negotiation and the asynchronous backtracking.

  Contract net protocol: The protocol for an agent that has been given a task and does not have capability of performing the task to request the execution of the task to other agents. First the original agent requests the other agents to bid for the task. The original agent selects the bidding whose cost is the lowest among the bidding the agent has received and accepts the bidding. The original agent requests the agent with the successful bidding to perform the task.

  Purpose: To request task execution to the agent that can perform the task the most effectively.

  Applicability and Forces: In general, cooperation protocol patterns are applied if the bandwidth of the network is very broad.

- Direct Interaction Pattern: This pattern realizes cooperation protocols by message exchange generally via networks.

  Purpose: To implement cooperation protocols easily.

  Applicability and Forces: This pattern is appropriate if the network bandwidth is very wide.

- Mediation Pattern: A mobile agent called the mediation agent is prepared separately from the stationary agents. The mediation agent mediates the cooperation of the stationary agents by moving around the hosts. The cooperation protocol of the whole system can be altered by changing the cooperation protocol of the mediation agent.

  Purpose, Applicability and Forces: To use a cooperation protocol in a network with narrow bandwidth or in the case that agents do not have remote communication functionalities. In addition, this pattern is especially effective if changes of cooperation protocols are probable.
- Dispatching Pattern: Agents move from the user's host to the hosts where the resources are located. The agents are mobile at this moment but after the migration they realize cooperative activities as cooperative agents by communication among the agents (therefore among the hosts).

Purpose, Applicability and Forces: To effectively integrate cooperative procedures that are not yet integrated in the system.

**Example of the Mediation Pattern Application**
The Mediation Pattern can be effectively applied to the problem of heterogeneous application interconnection in multiple enterprises described as follows.

Recently, requirements for system development that needs interconnection of heterogeneous applications in multiple enterprises such as EC (Electronic Commerce), CALS (Commerce At Light Speed), and SCM (Supply Chain Management). According to such requirements, the distributed object technologies are being widely spread as one of the newest. However, these technologies have problems in the insufficient mutual coordination of object platforms provided by different vendors and the lack of flexibility against changes. On the other hand, by applying our method, especially the Mediation Pattern, these problems can be solved by the above mentioned reasons.

The problem of this example is as follows. In this example, each of some enterprises has one host and the hosts are connected via an extranet. More than one application for each host is operated. Under this situation, the user wants to build a system where the application a makes requests to b and c and then receives the results.

Then the Mediation Pattern can be appropriately applied by the following strategy. Stationary agents are prepared as each of them realizes the interface of each application. Mobile mediation agents are also prepared that mediates the interaction of the stationary agents. Figure 9 is the description of the macroarchitecture of this example.

**Microarchitecture Patterns**
As patterns applied in the development of the microarchitecture, the microarchitecture patterns are specialized in the individual agent platforms and make good use of the advantage of the platforms. In this paper, Aglets and Plangent are used as platform examples and the microarchitecture patterns of them are introduced. In addition, the Security/Safety Pattern is also presented as a pattern that can be used commonly for several platforms.

**Aglets Patterns**
The followings are explanations of summary of the patterns introduced in Aridor & Lange [1]. A feature of Aglets is that the agents are described only in Java codes and the platform is provided only as a class library. Therefore the patterns of Agents make good use of the characteristics. The Agents patterns are classified to the following three types.
• Traveling Patterns
  These patterns are involved in agent migration (traveling). They include the Itinerary that can be used in the implementation of the Itinerary Macroarchitecture Pattern, the Forwarding and the Ticket that can be used in the implementation of the Basic Mobility Pattern.

• Task Patterns
  These patterns are used to make agents perform various tasks. They include the Master-Slave and the Plan.

• Interaction Patterns
  These patterns are applied for interaction of agents. They can be used in realization of cooperation protocols. They include the Meeting, the Locker, the Messenger, the Facilitator, and the Organized Group.

Plangent Patterns
As mentioned before, one of the main features of Plangent is the intelligent behavior based on planning. Therefore the appropriate microarchitecture patterns of Plangent should make the best use of the feature. The following patterns are extracted from this viewpoint.

• Basic Action Patterns: Examples of basic actions of agents includes Migration, Knowledge Updating, Copying, Operation and Recognition. The knowledge means the one used in planning. The Copying is classified into Cloning where agents give their copy the same data and Spawning where the agents give their copy the parents' subgoals.

• Plan Generation/Execution Pattern: The functionalities of Plangent with AI features such as planning and replanning can be put together as the plan generation/execution pattern.

Security/Safety Patterns
Security and safety are regarded as important in agent technology, especially mobile agents. Most agent platforms support some security and safety mechanisms. However, some systems may need more strict security and safety levels. Therefore, security/safety patterns are needed to improve the security and safety levels in usual agent platforms (Figure 10).

An example of security patterns is the Dummy Data Verification Pattern [13]. In this pattern, dummy data is attached to the agent in advance, a copy of the agent remains in the original host when the agent moves to the network and after the agent has come back the dummy data is verified.

An example of safety patterns is the Procedure Completion Confirmation Pattern. In this pattern, if an agent moves to an unstable host, a copy of the agent remains in the original host and the copy is prepared for receiving the message representing that the original agent has completed its procedure in the destination host. In addition, so as to deal with the situation when an accident occurs in the destination host and the agent cannot continue the procedure, if the copied agent does not receive the message after a specified time, the copied agent repeats the migration process.

Relationships of the Patterns
As 3 and Figure 4 show, there are relationships among the behavior patterns of each layer according to the development process. In brief, a lower layer pattern is a component of the refined higher layer patterns and therefore the lower layer pattern is applied in the refinement of the outline design specifications where the higher layer pattern is applied. An example of the relationships of Plangent microarchitecture patterns are shown in Figure 11.

Development Process Based on the Layered Architecture
The development process is defined on the basis of the layered architecture. Its summary is as follows (Figure 12).

- The development phases are assigned to the layers from the higher one to the lower. The result is the definition of the three phases in total.
- In each of the phases, the architecture is designed by applying the behavior patterns.
- In order to specify the conditions of pattern application, the pattern requirements extraction phase is located before the design phases.
The followings are the detailed description of the phases.

**Pattern Requirements Extraction Phase**
In this phase, the application conditions of the patterns used in the following architecture design phases. The conditions includes the followings.

- Available infrastructures of the environments where the agent system is operated and their properties.
  
  The infrastructures includes both hardware and software. As for the hardware, the types and the performance of the computers that are the components of the network and the network connections are examined. As for the software, the conditions are investigated from the foundational levels such as the OS's to the higher levels such as reused applications.

- Conditions in system requirements.
  
  Conditions of the performance and the amount of data are extracted from the system requirements.

**Macroarchitecture Design Phase**
In this phase, the macroarchitecture design specifications are composed by instantiating and combining the macroarchitecture patterns matching with the application conditions extracted in the previous phase.

The instantiating procedure is as follows.

1. All the hosts in the system are classified from the viewpoint of the pattern application conditions.

2. Agent behaviors are described using agent behavior diagrams by applying the patterns matching the application conditions for each group of the hosts. Then, by combining the diagrams, typical agent behaviors in the whole system are described.

3. State transition diagrams representing the agent behaviors described in the agent behaviors diagrams are drawn. Then various alternatives are added if necessary and the state transition diagrams representing general agent behaviors are completed.

4. The event sequence charts are generated by describing the temporal system behaviors from the state transition diagrams.

In order to enable this procedure, the macroarchitecture defines the following guidelines explicitly.

- The guideline for macroarchitecture design by sequential generation of the three types of diagrams that comprise the macroarchitecture by extension and ordering of the events.

- The guideline for pattern application, that is, instantiation of the patterns into the solutions of the given problems, such as how to substitute which parameters.

**Microarchitecture Design Phase**
In this phase, the microarchitecture design specifications are designed by refining the macroarchitecture designed specifications generated in the previous phase. In detail, as Figure 4 shows, the following two methods are used.

- The instances of the macroarchitecture patterns that comprise the macroarchitecture design specification are refined by combination of the instances of the microarchitecture patterns.

- The part of the system out of the instances of the microarchitecture patterns are designed by also applying the microarchitecture patterns.

The pattern application conditions are important also to this phase. Another point is how to highlight the features of the agent platforms. Therefore, these points are explicitly stated in the microarchitecture patterns applied in this phase.

**Object Level Design and Implementation Phase**
In this phase, the object level is designed and finally implemented by refining the microarchitecture designed in the previous phase. In detail, the instances of the microarchitecture patterns that are the components of the microarchitecture design specification are implemented by applying OO design patterns and combined as a system.

4 **Evaluation of Our Method**
In this section, the method proposed in this paper is evaluated by analyzing the effect to the examples described below on the basis of the viewpoint of the performance of the developed system. The method of this paper aims to realize architectures optimized in performance by applying patterns in the macroarchitecture decision phase considering the structure and characteristics of the system environments extracted in the system environment analyzing phase. From this viewpoint, how much the macroarchitectures optimize performance is evaluated quantitatively. Therefore performance in such network environments, especially network traffic is evaluated.
The evaluation is based on actual data of the following examples.

- The remote monitoring example to which the Itinerary Pattern is applied.
- The keyword search example to which the Branching Pattern is applied.
- The example of heterogeneous application interconnection in multiple enterprises to which the Mediation Pattern is applied.

Heterogeneous Application Interconnection in Multiple Enterprises

The performance of the system in this example is evaluated in the case where the Mediation Pattern is applied and mobile agents are not used. The network traffics are calculated in the both case using the following data.

- There are one client host and ten server hosts.
- The average amounts of data that the client and each of the servers exchanges are as follows.
  - The request from the client to a server: 1
  - The result the client requires finally: 10
  - The data the client and a server exchange during the interaction: 100
- The amount of the mobile agent code is 5.

The traffics are calculated as follows.

- If mobile agents are not used, since the all data described above are exchanged via the network, the total traffic is \( (1 + 10 + 100) \times 10 = 1110 \).
- If the mobile mediation agents are used, only the final results at each host are accumulated in the agents. The amount of data that the mobile mediation agent carries when it depart from the client is \( 5 + 1 \times 10 = 15 \) including the request description. Therefore the total traffic is \( 15 + (15 + 10) + \ldots + (15 + 10 \times 10) = 715 \).

The result of comparison is shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>Network Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Mobile Agents</td>
<td>1110</td>
</tr>
<tr>
<td>With Mobile Mediation Agents</td>
<td>715</td>
</tr>
</tbody>
</table>

Table 1: Evaluation of the Example of Heterogeneous Application Interconnection in Multiple Enterprises

be seen that the traffic will be lower if the Mediation Pattern is adopted than if there are no mobile agents for the proportion of the amounts of data shown in this example.

Remote Monitoring

In this example, the system monitors machines connected to a wide-area network. If a disorder in the monitored data is detected, the system deals with the situation. It is supposed that multiple machines, usually many, that should be monitored, are connected to the network. The monitoring person manages the central monitoring host connected to the same network and performs remote monitoring. When a disorder in the monitored data is detected, the disorder data handling procedures corresponding to the degree of emergency of the disorder are executed. In addition, let us consider the case when such disorder rarely happens and it is desirable to keep the amount of the computation resources used in the monitoring process small. In this case, because the monitored machines are specified in advance and the requirement that the network traffic should be low is given, the Itinerary Pattern is applied. The agents gather the monitored data and execute the disordered data handling procedures if a disorder is detected as they move around the monitored hosts.

The performance of the remote monitoring system is evaluated on the basis of the following data.

- The numbers of those kinds of the connected hosts are 10 and 2 respectively.
- The network traffic of an agent migration, a monitoring request message and a monitoring result data are 100, 1, and 2 respectively.

The evaluation is as follows.

- In this example, both of the itinerary pattern and the dispatching pattern are applied according to the characteristics of the monitored machines. The case is compared with the one where only the itinerary pattern is applied. Since the agents cannot stay and continue to be active in the monitoring hosts with a small amount of the resources, application of only the dispatching pattern is not considered.
- In the evaluation of the network traffic, the traffic per unit time is calculated in the similar way as the travel scheduling example.

  - The combination of the itinerary pattern and the dispatching pattern
    - In the itinerary part, the traffic is calculated similarly as the travel scheduling example and is \( 100 + 102 + \ldots + 120 = 1210 \).
    - In the dispatching part, because the agent codes flow only once in the beginning, the traffic is considered as 0 per unit time. Therefore the monitoring requesting messages and the monitoring result data are considered to be communicated and the traffic is \( 1 \times 2 + 2 \times 2 = 6 \).
Therefore the total is 1216.

- Only the itinerary pattern: Similarly as the above, $100 + 102 + \ldots + 124 = 1466$.

The evaluation is shown in Table 2. Table 2 shows that

<table>
<thead>
<tr>
<th>Applied patterns</th>
<th>Network Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Itinerary and</td>
<td>1216</td>
</tr>
<tr>
<td>Dispatching</td>
<td></td>
</tr>
<tr>
<td>Only Itinerary</td>
<td>1466</td>
</tr>
</tbody>
</table>

Table 2: Performance Evaluation of the Remote Monitoring System

A system with low network traffic can be developed by combining the itinerary and the dispatching patterns.

Keyword Search

In this example, information retrieval from search engines on the network using keyword search is performed fully automatically. For example, a retrieval is executed on each search engine by keyword "a" first and if the number of the hits is too large (more than fifty, for example), the narrowing process is done using keyword "b." In this case, because the amount of data that should be communicated is relatively large, the branching pattern is applied. Each agent moves to each search engine one by one, retrieves according to the above procedure, goes back to the user’s host and display the results.

As for this example, we compare the network traffics of the cases where the branching pattern and the itinerary pattern are applied and agents are not used. The basic data are the following.

- The number of the search engine hosts is 10.
- Among them, the keyword “a” hits 30 in seven hosts and the keyword hits 1000 and the narrowing by the keyword “b” results in 20 hits in the other three hosts.
- The amount of data of the agent code, the search request message, and the search results per one hit are 10, 1 and 1 respectively.

From these data, the network traffics are calculated as follows.

- As for the branching pattern, the total of the amount of the agent codes for the round trips and the average 30 hits and 20 hits is $20 \cdot 10 + 30 \cdot 7 + 20 \cdot 3 = 470$.
- As for the itinerary pattern, similarly as the previous examples, the maximum is $10 + 40 + \ldots + 220 + 240 + 260 + 280 = 1700$ and the minimum is $10 + 30 + \ldots + 150 + 180 + 210 + 240 = 1270$.

- If agents are not used, the calculation is as follows. First, as for the sessions with only one retrieval, the total of the one retrieval requesting message and the returned retrieval results is $31 \cdot 7 = 217$. Next, as for the sessions with two retrievals, the total of two retrieval requesting messages and the all retrieval retrieval results is $1001 \cdot 3 + 21 \cdot 3 = 3066$. Their total is 3283.

The result of the comparison is as the Table 3 shows. This shows that the branching pattern can reduce the network traffic the most effectively. In addition, because the first search where the hit numbers are very large can be performed locally by using mobile agents, the traffic is reduced.

5 Comparison with Related Work

In this section, the effect of our method is examined by comparing with related work.

Design using design patterns [5] is widespread recently and there are some researches in development of agent systems.

Aridor and Lange [1] classifies some design patterns into three kinds, that is, the Traveling Pattern, the Task Pattern and the Interaction Patterns and presents some patterns. In addition, it applies the patterns to some examples. However, those design patterns are not distinguished as for the two levels in this paper and there are no viewpoints of in which order the application of the patterns should be examined in the examples. Therefore, as the number of patterns increases, examination of pattern application becomes difficult. On the other hand, this paper not only presents patterns but also classifies them into the macroarchitecture patterns and the microarchitecture patterns and provides a guideline and an order of examination of pattern application. For example, in travel schedule information retrieval by mobile agents, the branching pattern is applied in the macroarchitecture decision phase and then the plan generation-execution pattern is applied in determining detailed destinations of the agents. Thus a clear design guideline is given in this example. Therefore, if the number of the patterns increases, they can be effectively applied in the design process by classifying them appropriately.

Kendall et al [8] examine design patterns for agents with a layered architecture. They illustrate patterns applicable to each layer constructing the agents. However, these patterns can only be applied to the implementation of each agent. This means that they examine only those patterns in the microarchitectures and more...
detailed levels and do not consider the macroarchitectures level. For example, they introduce some patterns to realize agent mobility as the coarsest ones. On the other hand in this paper, we think that it is necessary to support design process from the macroarchitectures level by applying patterns for practical agent system development, and its advantage by application to the examples can be seen.

Silva et al [12] proposes patterns that are centered on one agent and composed by collaboration diagrams and class diagrams. However, these patterns lack the viewpoints of multi-agent interaction and whole structure and behaviors of agent systems that are necessary to practical system development. On the other hand, we present a development method and associated patterns that unify these viewpoints and practical system development, and especially the whole design step can be effectively implemented.

Finally, the usual agent system development methods such as [10] and [9] cannot give guidelines that are easy to understand. On the other hand, our method give such guidelines by dividing the design process into two phases and providing exhaustive patterns that are easy to understand and correspond to the phases.

6 Conclusions
This paper proposed a method of agent system development based on agent patterns. The patterns are classified according to their appropriate architectural levels and the degree of their dependence on specific agent platforms. Our method enables developers to design agent systems efficiently since they can construct complicated system architectures and behaviors by dividing the design process into two architectural levels and applying the appropriate agent patterns. In addition, the higher level designs are independent of specific agent platforms and can be therefore easily reused.

The following are some future work possibilities:

• Extraction and examination of patterns
  This paper, necessary patterns are extracted in some examples including travel scheduling. We will extract and examine additional useful patterns through practical application of Plangent in the future.

• Expansion to a systematic development methodology
  This paper proposed a design process with three phases and two level patterns. We aim to expand the method to a systematic development methodology of agent systems including requirement analysis, implementation, maintenance and management processes.

Acknowledgments
We would like to thank Dr. Matsushiro Arinobu, General Manager of Systems and Software Research Laboratories, who gave us the opportunity of this research, and Dr. Samuel R. H. Joseph, who gave us a special support.

REFERENCES