Network environments give computer users the option of employing distributed information and services to complete a task. However, gathering information and using services distributed in networks requires knowing exactly what kinds of information and services are required for a task, where they are, and how they can be obtained or utilized. Tracking down the answers to these questions can be difficult, time-consuming tasks for users.

Mobile agent technology is expected to release them from having to do so. Instead, “intelligent” mobile agents will comprehend the user’s requirements, search network nodes autonomously for appropriate information and services, and return with the answers. But several problems must be solved before we can expect agents to perform such actions effectively. These include security problems relating both to the network nodes that the agents visit and to the information that the agents carry. There are also problems regarding how to distribute and place agent platforms on a network node to support agent behavior.

These security and platform problems will remain incidental, however, until practical and useful agents emerge from the current research. We therefore focus here on the question of intelligence as a prerequisite for agent functions.

What sort of intelligence is expected of agents? We have adopted a model based on the ability to make flexible plans. Specifically, we think mobile agents must be able to make a plan based on user input and adapt it according to information gathered from the network.

To be practical, mobile agents must be “intelligent.” The Plangent system enables mobile agents to make a plan based on user input and adapt it according to information gathered from the network.

**Plangent: An Approach to Making Mobile Agents Intelligent**

_Akihiko Ohsuga, Yasuo Nagai, Yutaka Irie, Masanori Hattori, and Shinichi Honiden_

_Toshiba Corporation_
understand user requirements,
plan actions that will satisfy the requirements
act according to the plan,
modify the plan according to actual conditions when they differ from those initially expected, and
execute the modified plan.

Mobile agent and planning technologies are not quite new. However, it is far from easy to implement mobile agents with planning functions that can support migrations and planning appropriately for each visited network node and also satisfy the user’s requirements as a whole.

We have implemented these functions in the Plangent system and validated their effectiveness in several example applications. In this article, we describe how we combined these planning functions with mobile agent facilities, and show how the agents behave intelligently in an example application of personal travel assistance. For a comparison of our system with three other implementations of intelligent mobile agents, see the sidebar “Related Work”, page 52.

PLAGENT SYSTEM ARCHITECTURE
The Plangent system is written in Java and composed of agents with these features:

- **A process for each activity.** Processes handle commands written in script languages.
- **Mobility.** Agents can move from one network node to another using a “go” command. This mobility is achieved through network management facilities.
- **Information gathering facilities.** Agents obtain information from the nodes they visit and use it for planning.
- **Planning capabilities.** Agents make plans by adapting to the new environments of each node they visit. In the Plangent system, this planning capability makes the agents “intelligent.”

Because agent mobility and planning are important features for the Plangent system, we describe them here in detail.

**Plangent Mobility**
In the Plangent system, an agent is spawned as a process in one node, which can then cease in that node and be activated in a remote node. Consequently, the processes work as if the agent continues its activity across several nodes in the network.

Figure 1 illustrates the Plangent architecture, including agent platforms and systems. Users define their requirements through the user interface (where they also get the result of agent activities). The agent generator sets up a process for an agent if it receives the user requirements.

After the agent is spawned as a process, it starts its activities. It uses data stored in the node as system information and domain knowledge to develop and execute its plan. When the agent executes the “go” action, it invokes a migrat-
RELATED WORK

Plangent has some similarities with two other mobile agent systems written in Java, namely, Cyberagent and Aglets Workbench, and with Softbot.

FTP Software’s Cyberagent technology1 has a function to authenticate mobile agents and send them to specified addresses and a security function to prohibit illegal access by agents. It also supports agent generation, registration and activation, migration route definition, execution result display, and agent cessation functions. Network administration tasks can be automated and dramatically changed by generating agents, which move from desktop to desktop, and letting them execute predefined tasks.

Aglets Workbench2 is a visual application development environment that uses mobile agents to access, retrieve, and manage data shared in networks. It has a development support function for platform-independent mobile agents based on Java, a visual agent builder, access and retrieval functions for shared databases, and an interaction function with standardized security managers. It also supports a function for gathering information necessary for a decision as to what agents should execute, and when and where they should move.

Both of these mobile agent systems can gather information necessary for a decision as to when and where the agents should move. Both systems also perform the tasks necessary for mobility by combining the mobility function and the data access and information retrieval functions.

However, it is very difficult to handle information gathering and retrieval using procedural programs because all movement options of the agent’s behavior must be described before execution. As a result, the information can easily be incomplete.

Plangent, on the other hand, uses metalevel control to work with incomplete information in open networks. In particular, using information from the “maybe” argument, Plangent can replan when the knowledge is incorrect (including when information is incomplete), and can then execute the new plan.

As for unexpected changes, Plangent can make plans for user requirements since the requirements are described as abstract specifications (sets of goals). Therefore, agents can gather the necessary information, make action sequences satisfying the requirements, and thus manage unexpected changes on nodes in open networks.

Softbot3 implements agents (or software robots) that automatically generate and execute sequences of Unix commands to satisfy user requirements described declaratively. Softbot uses communication commands such as ftp and telnet to handle network environments. However, such commands limit remote node access. For example, if the route used by an agent is disconnected while it is accessing a network node, the agent cannot continue the access during the disconnection. Plangent, on the other hand, allows agents to migrate to remote nodes. Thus, if they access a node and the route is disconnected, they can still complete their access.

REFERENCES


Plangent has an advanced planning feature—specifically, a technique that uses backward reasoning from declarative statements of user requirements to generate sequences of actions that will satisfy the requirements.

The technique interleaves planning and plan execution. In the open networks where we expect Plangent to be useful, agents often cannot obtain enough information to complete plans. In Plangent, if the agent encounters this situation, it suspends planning activities and executes sequences of actions that are included in the plan and executable at that time. Then it gathers relevant information and continues planning. This technique gives Plangent a flexible planning function, especially when mobile agents interleave the “go” action and generate plans as they gather information distributed over networks.

The planning mechanism is reflective,3 that is, agents can execute metalevel planning, or metaplaning. The metaplaning procedure is the same as the usual planning procedure (see Weld4), except that it can affect the base-level planning mechanism. Therefore the planning procedure acquires more flexibility. In fact, metaplaning strengthens the flexible control of interleaving.

Basically the planning mechanism uses the following knowledge representation:

- Conditions are denoted by relational sentences—that is, by lists whose first element is a relational constant or a functional constant followed by some arguments.
- Initial states are denoted by sets of conditions that represent the states of the whole world of the network the agents know.
- Action information consists of (a) the action descriptor composed of a name and
arguments if any, (b) the precondition, denoted by a set of conditions and representing the necessary condition to execute the action; and (c) the postcondition, denoted by a set of conditions, representing the effect of the action and consisting of elements classified according to whether they add or delete some conditions.

Figure 2 shows the structure of Plangent’s planning mechanism. It uses least commitment planning. With this technique, even if the agents do not have enough information to make complete plans—usually because they have not yet traveled to the nodes that have the necessary information—they can nevertheless construct incomplete plans containing subplans that are executable at that time.

The planning proceeds as follows:

1. The agents receive the user’s requirements. The requirements are represented using terms composed of constants, variables, and functional symbols that include predicates.

2. The agents initialize their planning procedure. They make the plan sets (sets of actions comprising the plans), define the safety conditions, and establish the causal links to initial values. The users may give these initial values to the agents.

3. The agents list unsatisfied subgoals and try to satisfy one of them either by choosing an action from the already generated plan and trying to satisfy the selected subgoal by the action (in which case, the agents do not have to update any other information) or by choosing an action not included in the already generated plan and trying to satisfy the selected subgoal by the action (in which case, the threat must be checked as described in Step 4). If there is no appropriate action, the agents backtrack and redo the choice of the subgoal to be satisfied. If there are no more subgoals to satisfy, the agents proceed to Step 6.

4. If the agents have chosen an action not included in the already generated plan, they check whether the newly selected action breaks the consistency of the already generated plan. If it does, the action is called a threat, and the threat is checked. The agents use the causal links and update the safety conditions if necessary. If this step fails, the action is backtracked and another action is selected. If no more actions are available, the procedure jumps to Step 6.

5. The procedure returns to Step 3.

6. Plan execution. The agents select actions that are executable—that is, whose preconditions are satisfied—and execute the actions. If possible, they execute actions other than “go” in order to minimize the agent-transfer costs.

7. Metaplanning. The agents reify their states—that is, treat their states as data, which they can manage at the metalevel. If the agents recognize one of the following situations during the planning, they generate metaplan to cope with it:
   — Failure of plan execution: Even if planning succeeds, the information used in the planning is not always true in open distributed environments. In particular, execution of actions may fail when pre- or postconditions become false because of an exceptional situation such as a change in the system configuration.
   — Plan size limit: The planning procedure may not terminate because the procedure loops. To manage such cases, a limit is set on the size of the plan tree and planning is interrupted when the size exceeds the limit.

In particular, in the case of the failure of plan execution, agents refer to incomplete knowledge using the “maybe” predicate, including the relational constant maybe, and generate new plans using this knowledge. This procedure is called replanning.
EXAMPLE SCENARIOS

To explain Plangent’s agent planning behaviors, we present three scenarios in the context of a travel scheduling and reservation application. The scenarios involve transportation between Tokyo and the Japan Advanced Institute of Science and Technology (JAIST), and they address typical problems involving information retrieval, gathering, and use in open networks. We have implemented the application on workstations (with Solaris), personal computers (with Windows 95 and/or Windows NT), and mobile computers (with Windows 95).

In these scenarios, the user inputs travel requirements to a mobile computer such as a Personal Digital Assistant. An agent schedules the travel to satisfy the requirements and reserves transportation if possible. In making the schedule, the agent gathers as much useful information as possible from network nodes and displays it for the user. In reserving the transportation, the agent makes a reservation when it finds appropriate transportation in the retrieval process and then displays the result. We assume that the agent can obtain the necessary information about transportation timetables and reservation conditions from transportation companies’ Web servers.

In this example, goals represent travel dates, arrival times and (departure and arrival) places, and user preferences such as specific transportation vehicles, transportation companies, and cost.

The scheduling and reservation process begins with the user inputting the necessary conditions—date and time, places, and so on—through the user interface, as shown in Figure 3. The agent then determines the travel route by decomposing the travel interval to local and global areas and composing the routes determined in both kinds of areas.

Next, the agent schedules the travel, first applying a default preference order of transportation (for example, airplanes > trains > coaches, meaning that a plane ticket would be reserved if a flight is available, then a train ticket, then a coach.) Next, the agent generates candidate transportation companies and decides on the travel schedule (route and transportation). The transportation is reserved by executing the appropriate transactions with the companies. The results are displayed through the Plangent user interface.

Agent Behaviors in Plangent

Figure 4 presents example representations of knowledge required by the mobile computer and the travel agent site. The representations use KIF (Knowledge Interchange
Format), a Lisp-like language for knowledge representation and interchange of knowledge.\(^5\)

To simplify the descriptions of the agent functions in these examples, we begin with simple behaviors and progress to more complex ones. The numbered steps in the text correspond to the numbers identified in the figures.

**Case 1.** Figure 5 illustrates the agent behavior of Case 1.

1. The user inputs goals to the agent through a mobile computer. The goals are stated as requirements to make a travel schedule and to reserve the necessary transportation to depart from Tokyo on a certain date and arrive at JAIST by 12:00.
2. Because the mobile computer contains no information about travel scheduling, the agent moves to a travel agent site and makes a schedule and a global plan for transportation reservation.
3. By executing the plan at the travel agent site, the agent tries to gather information about access to JAIST (the destination), decides the travel route and confirms the reservation condition of necessary transportation.
4. The plan execution succeeds and the agent makes the detailed travel schedule and reserves the transportation.
5. The agent moves back to the mobile computer as soon as the computer turns on and sends an e-mail including the travel schedule and the reservation results.

The plan is generated in the form of the script descriptions presented in Figure 6a (for the mobile computer) and Figure 6b (for the travel agent site). The agent first tries to generate a plan in the mobile computer for the goal *(travelSchedulingAndReservation tokyo jaist (time_and_date (12 0) (9 25)))*. The agent searches the knowledge in the mobile computer, then notes that it has no useful knowledge resulting from this search and should go to the travel agent site. The agent generates the plan in Figure 6a, executes it, and moves to the travel agent site. At the travel agent site, it makes the plan in Figure 6b based on backward chaining to achieve subgoals derived with preconditions in other actions from the goal *(travelSchedulingAndReservation tokyo jaist (time_and_date (12 0) (9 25)))*, as shown in Figure 7.

**Case 2.** Figure 8 illustrates the agent behavior in Case 2, in which the following knowledge is added to the travel agent site:

\[(\text{maybe} (\text{exists} (\text{routeDB tokyo jaist}) \text{jaist}))\]

The maybe is a relational constant, which shows that the argument is a relation representing incomplete knowledge...
and should be used in any replanning.

1. The behavior is similar to that in Case 1 until the plan is executed at the travel agent site.
2. When the agent executes the plan, it fails to obtain \((routeDB \text{ tokyo jaist})\) and performs replanning.
3. The replanning goal is \((get (routeDB \text{ tokyo jaist}))\).

Accordingly, the agent generates a plan to move to JAIST, retrieve and obtain the route database, and move to the travel agent site.
4. The agent succeeds in executing the remade plan and acquires the route database.
5. At the travel agent site, the agent replans according to the goal \((travelSchedulingAndReservation \text{ tokyo jaist} (time\_and\_date \text{ (12 0) (9 25)}))\). Having executed the plan successfully, the agent makes the schedule and the reservation.
6. The agent moves to the mobile computer as soon as the computer turns on and sends an e-mail including the travel schedule and the reservation results.

In this scenario, the incomplete knowledge \((maybe (exist (routeDB \text{ tokyo jaist}) \text{ jaist}))\) means that route information from Tokyo to JAIST may exist at JAIST. The agent uses this knowledge in replanning.

Case 3. Figure 9 illustrates the agent behavior in Case 3.

1. The user inputs goals to the agent through a mobile computer as in Case 1.
2. Because there is no information about travel scheduling in the mobile computer, the agent moves to a travel agent site, and makes a schedule and a global plan for reserving transportation.
3. By executing the plan at the travel agent site, the agent tries to gather information about access to JAIST (the destination), decide the travel route, and confirm the reservation.
4. When the agent executes the plan, it fails to obtain \((routeDB \text{ tokyo jaist})\) and performs replanning.
5. As in Case 2, the agent goes to JAIST, retrieves \(routeDB\), and returns to the travel agent site.
6. In executing the plan for the goal \((travelSchedulingAndReservation \text{ tokyo jaist} (time\_and\_date \text{ (12 0) (9 25)}))\) at the travel agent site, the agent fails in retrieval and acquisition of \((ticketDB \text{ ?travelPlan})\) and again performs replanning.
7. By replanning for the goal \((get (ticketDB \text{ ?travelPlan}))\), the agent makes a plan to move to All Nippon Airways (ANA), retrieve and obtain \((ticketDB \text{ ?travelPlan})\), and return to the travel agent site.
8. In executing the plan, the agent fails in the retrieval and the acquisition of \((ticketDB \text{ ?travelPlan})\). Then it deletes the maybe knowledge by which it decided to move to ANA and returns to the travel agent site.
9. The agent replans according to the goal \((get (ticketDB \text{ ?travelPlan}))\). As a result, it generates a plan to move to JAL, obtain and retrieve \((ticketDB \text{ ?travelPlan})\), and move to the travel agent site.
10. The agent executes the plan, but fails to obtain \((ticketDB \text{ ?travelPlan})\), and returns to the travel agent site.
11. The agent replans according to the goal and makes a plan to reserve transportation for the route via Toyama on the basis of knowledge about the routes.
12. In executing the plan, the agent successfully acquires and retrieves \((routeDB \text{ tokyo jaist})\), but fails in pointer acquisition and retrieval of \((timeTableDB \text{ toyama jaist})\), and replans.
13. The replanning goal is \((get (timeTableDB \text{ toyama jaist}))\). Accordingly, the agent makes a plan to move to JAIST, retrieve and obtain the pointer of \((timeTableDB \text{ toyama jaist})\), and return to the travel agent site.
14. The agent succeeds in retrieving and obtaining timetable database pointer and returns to the travel agent site.
15. Executing the plan at the travel agent site fails in retrieving and acquiring \((ticketDB \text{ ?travelPlan})\); the agent replans.
16. In this replanning, the agent makes a plan to move to
ANA, retrieve and obtain the ticket database, and return to the travel agent site.

17. The agent succeeds in executing the plan and returns to the travel agent site.

18. The agent moves to the mobile computer as soon as the computer turns on, sends an e-mail including the travel schedule and the reservation results, and reports these results through the user interface shown in Figure 10.

As this case shows, the agent can adjust to various situations, allowing it to select travel routes that satisfy the user's requirements, make travel schedules, reserve the transportation, gather information, and return to the mobile computer. In addition, if more than one node at JAIST has timetable databases, the agent can send queries to those nodes simultaneously. Then the agents at all nodes examine their respective timetables concurrently. In this way, Plangent offers a simple version of a multiagent function.

FUTURE WORK

The Plangent system has been implemented to gather distributed information over networks and uses it effectively. The system generates and modifies global and local plans by using a layered planning/plan-execution function based on metalevel control. It can easily repeat the cycle of plan generation, migration, execution, checking results, and replanning.

Because we have been able to confirm the system's effectiveness for developing practical agent applications, we plan to consider design methodologies for these applications and security problems. We are currently developing another Plangent system that includes higher multiagent functions.

ACKNOWLEDGMENTS

We are grateful to Mutsuhiro Arinobu, director of the Systems and Software Research Laboratories at Toshiba, for the opportunity to conduct this research. We also thank Yasuyuki Tahara, a researcher at the Laboratories, for his contributions to this research. Many thanks are due Kazuma Asakawa, Hideki Yamaga, and Hiroshi Sasaki, all from Toshiba Information Systems (Japan) Corporation, for their assistance in developing the system.

REFERENCES


Akihiko Ohsuga is a research scientist in the Systems and Software Research Laboratories, Toshiba Corp. His research interests include agent technologies, formal specification and verification, and automated theorem proving. He received a BS in mathematics from Sophia University, and a Dr. Eng. in electrical engineering from Waseda University.

Yasu Nagai is a research scientist in the Systems and Software Research Laboratories. His research interests include agent systems, design expert systems, constraint satisfaction, constraint programming, and object-oriented design and patterns. He received a BE and an ME in electrical engineering from Waseda University.

Yutaka Irie is a researcher in the Systems and Software Research Laboratories. His research interests include agent-based languages, object-oriented software development, and parallel processing. He received a BE and an ME in electrical engineering from Waseda University.

Masanori Hattori is a researcher in the Systems and Software Research Laboratories. His research interests include agent-oriented languages, network-computing, and human interface. He received a BE and an ME in computer science from Kyushu University.

Shinichi Hidonen leads Laboratory IV in the Systems and Software Research Laboratories, Toshiba Corporation. His research interests include software engineering and artificial intelligence. He received a BE, ME, and Dr. Eng. in electrical engineering from Waseda University.

Readers may contact Ohsuga or Nagai at Toshiba Corp., 70 Yanagi-cho Saiwai-ku, Kawasaki 210; e-mail [ohsuga, nagai]@ssel.toshiba.co.jp.