Mobile Agent Based Evacuation System When The Battery Runs Out : EASTER

Heisuke KANEKO and Yoshiaki FUKAZAWA
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo, 169-8555, JAPAN
{heisuke,fukazawa}@fuka.info.waseda.ac.jp

Fumihiro KUMENO
Mitsubishi Research Institute, Inc.
2-3-6 Otemachi, Chiyoda-ku,
Tokyo, 100–8141, JAPAN
kumeno@mri.co.jp

Nobukazu YOSHIOKA and Shinichi HONIDEN
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo, 101-8430, JAPAN
{nobukazu,honiden}@nii.ac.jp

Abstract

As mobile computing becomes common, the battery issue of mobile computing devices has become increasingly notable. To this end, research and development of various power-conservation devices and methods are actively taking place. However, the conventional method of extending the battery life through power-conservation can never prevent the unintentional shutdowns of applications due to the dead battery. This research aims to realize the evacuation of applications on a mobile computing device to another device before the battery runs out by creating the application as a mobile agent. Particularly, by introducing the concept of the Crisis Management Center, dynamic and smooth evacuation of multiple application agents will become possible. This paper explains and verifies the effectiveness of the EASTER (Escape Agent System from dying batTERy), a system developed for the purpose of recovering the applications when a battery is running out through the use of mobile agent system.

1 Introduction

The use of mobile computing devices such as PDAs and cellular phones is expanding rapidly, their performance is fast improving and they start to incorporate Java virtual machine (JVM). In addition, wireless networks such as wireless LAN and Bluetooth can be used for the above devices, enabling the use of various applications at any time and any place. In other words, we can execute applications that have been downloaded via wireless networks on our mobile computing devices.

Thus, while mobile computing devices have become indispensable in our life, issues that did not exist in conventional PCs and servers are surfacing at the same time, such as batteries with limited or small capacity, unstable network that are prone to get cut off any time, etc. Particularly, the battery issue is serious. When the battery runs out, we cannot use the applications running on the device and important data can be lost.

In order to resolve this battery issue, researches on extending battery life through increasing the battery capacity, developing a power saving device or applying power management technology such as an automatic sleeping mode, have been active. Undoubtedly, power-conservation can extend the operation time of a mobile computing device, but unless the user recharges the battery, it will eventually die out; thus it is impossible to completely eliminate the risk of " battery shortage when an application is running". In facing this issue, we have changed our point of view with the following thoughts: "Can we make an application protect itself before the battery runs out?" In other words, our proposal suggests that an application carry an adaptive capacity against the dead battery.

We consider such an adaptive ability to be one of the following three types.

(1) Saving important data to files

(2) Transferring important data to the network

Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)
0-7695-1893-1/03 $17.00 © 2003 IEEE

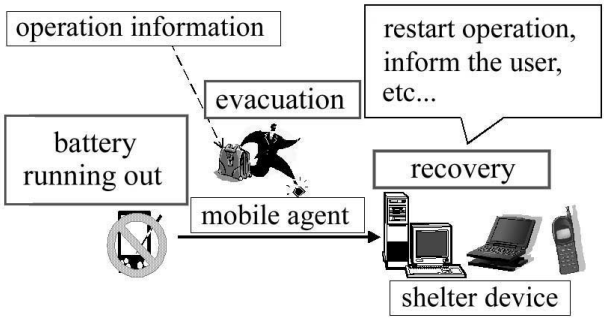

Figure 1. Evacuation action of an application agent.

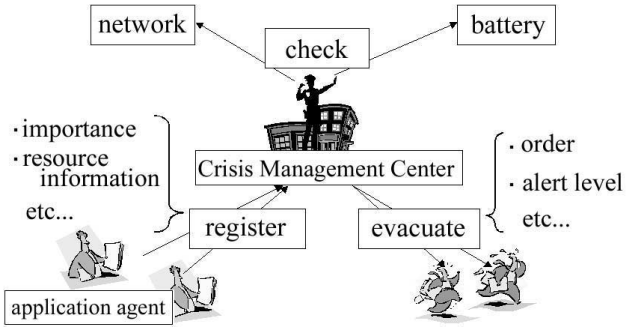

Figure 2. Crisis Management Center.

(3) Transferring both important data and the application itself to the network

In case (1), once a user recharges the mobile computing device, he can restart the operation. However, he cannot restart it unless he recharges the device whose battery has run out. Thus, restarting the operation on another device is impossible.

A user in case (2) can restart the operation on another device as long as he can download the evacuated data that was uploaded to the network. However, there is no assurance that an application running on the original device can be used on another device. Further, when the continual operation of applications on a mobile computing device is crucial, a sudden application termination and the loss of data lead to the serious damage.

The method in case (3) offers a solution to all the problems described in case (1) and (2). Therefore, the best way for an application to protect itself in the event of the battery running out is to transfer not only the data but also the application itself to another safe device via the network. We call this safe device the "shelter device". When an application that succeeds in evacuating, it can continue operating on the shelter device. Moreover, it is possible to transfer an evacuated application from the shelter device to another device. This enables the user to restart evacuated applications on any device.

To achieve this function, we use a mobile agent that is a program that migrates freely on the network along with its code and state. That is, we configure applications for mobile computing devices as mobile agents in which the evacuation function is embedded for when the battery runs out. The application agent can thus evacuate to another device before the battery runs out (Figure 1).

However, if multiple agents evacuate on their own, without any considerations to other agents, some applications may fail to evacuate. We therefore introduce the Crisis Management Center, which manages all information on a mobile computing device and devises an effective evacuation plan (Figure 2).

This paper explains and verifies the effectiveness of the EASTER (Escape Agent System from dying batTERy), a system developed for the purpose of recovering the applications when a battery is running out through the use of mobile agent system.

The remainder of this paper is organized as follows. Section 2 describes the kinds of applications benefit from the EASTER. Section 3 describes the introduction of the Crisis Management Center. Section 4 presents a detailed overview of the EASTER. In Section 5, we report some experiment results to confirm the applicability and validity of the EASTER. Section 6 presents a review of related works, while Section 7 offers our conclusions and future work.

2 Application Examples

In this section, we describe the kinds of applications that usefully benefit from the EASTER.

The EASTER enables not only the saving of data but also the continuous execution of the application. Thus, applications, whose continued operation is essential will benefit from the evacuation accomplished by the EASTER. To explain further, we will use the scenario of an emergency service call out. When an accident occurs, the ambulance attendants hurry to the rescue of injured person with PDA because they cannot carry the heavy equipment in the ambulance vehicle. On arriving, they confirm the identity of the injured party and download his medical record from the database at the medical center. Next, an attendant examines the patient and inputs information such as heart rate or blood pressure into his PDA. Care of the patient may require multiple applications. For example, one application may communicate with the server at the medical center, another may show the real time graphic representing the heart rate. If the PDA battery were to run out, important data would be lost and essential operations cease. It is crucial that the operation continue in a crisis situation as the one

described in this scenario. If applications could evacuate to another attendant's PDA by means of the EASTER, the important data and operation would not be lost.

Other than applications for emergent situations, applications that run automatically without a user, such as web collection programs that surf the WWW, benefit from the EASTER. These applications can continue to run after evacuation, and thereafter return to the original device, which a user has recharged the battery, with the results of their operation.

Normal applications can also benefit from the EASTER. For example, when a user is writing an e-mail and the battery runs out, if the mailer can evacuate, a user can keep the content of his unfinished e-mail and continue to write on another device nearby immediately. Because of evacuation of application, a user need not install a new mailer and set up large amounts of personalized information on the new device that a user must start to use instead of the original device whose battery has runs out.

3 Crisis Management Center

In the future, more than one application will be running on a mobile computing device as same as a PC. If all application agents detect the battery shortage and attempt to evacuate simultaneously, there is a great possibility that the entire evacuation system may be obstructed. In other words, if multiple agents individually make the decision to evacuate, they may end up obstructing each other. This phenomena caused by individual agents not understanding the surrounding environment (only thinking about itself) is defined as rush crush.

One of the measures to avoid rush crush is similar to the multi-agent system where all the agents exchange information frequently and devise an evacuation plan cooperatively. However, this type of feature generally increases the burden of the system, and is difficult to operate on the low-performance machines such as PDAs and cellular phones. Furthermore, adding a sophisticated feature on each agent causes the data size of each agent increase and may result in longer migration time on the network during the evacuation. Finally, sophisticated planning in a low-resource environment takes much time and the situation may worsen during the execution.

For these reasons, we decided not to adopt the above multi-agent solution. In the EASTER, each application is implemented as a simple mobile agent, which migrates with both code and data but does not have high intelligence and cooperative features. Alternatively, we introduce the Crisis Management Center that manages all agents' information and monitors the situation on a mobile computing device to devise an efficient evacuation plan.

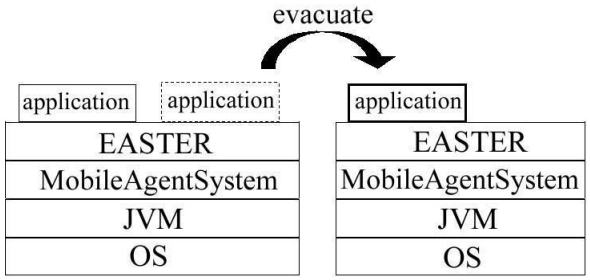

Figure 3. System hierarchy.

4 Overview of the EASTER

As mentioned above, the Crisis Management Center efficiently evacuates some applications that are implemented as mobile agents before the battery runs out, thereby preventing the loss of applications and their operation information. After an evacuation, an application can still easily use its operation information and restart its operation on the shelter device. Evacuation is impossible, however, when the battery life is very short or a wireless network cannot be used. In such a case, the Crisis Management Center commands applications to evacuate into local files. Evacuation to local files is an undesirable alternative, however, as mentioned in Section 1. Thus, the basic principle in making an evacuation plan of the Crisis Management Center is to evacuate the maximum number of agents to shelter devices.

4.1 System Hierarchy

The hierarchy of the EASTER is presented in Figure 3.

Using a Java-based mobile agent system enables any application to run on any device after evacuation. At this time, we employed AgentSpace [10] as a mobile agent system, which is a Java-based mobile agent system that supports weak mobility. However, because code/state mobility is required for achieving evacuation, other mobile agent systems such as Aglets [5] or Voyager [6] may also be used. We employed AgentSpace because it zips the necessary transfer data when an agent migrates, and we expected to shorten the evacuation time by this migration method. However, it turned out the zip process takes a long time for zip process on mobile computing devices whose performance is highly restricted. We therefore appended the non-zip migration method to AgentSpace.

The EASTER hierarchy functions to provide the Crisis Management Center, evacuation of applications, and acceptance of evacuated application agents.

Table 1. Category of callback methods.

before an evacuation	after an evacuation
escapeCompletely()	recoverCompletely()
escapeQuickly()	recoverQuickly()
escapeToFile()	recoverFromFile()

Table 2. Behavior when an agent evacuates.

Alert Level	Behavior	
Low	before	execute escapeCompletely()
	after	execute recoverCompletely()
Medium	before	execute escapeQuickly() discard low important-level resources
	after	execute recoverQuickly()
High	before	execute escapeToFile()
	after	execute recoverFromFile()

4.2 Shelter Device

A shelter device may be a PC at a user's home that is continually connected to the Internet or a PC at user's office. It may be a server accessible by the public. Or, it may be a mobile computing device of a user's friend who happens to be nearby and so the user can restart the operation instantly.

As showed in Figure 3, for a device to function as a shelter device, JVM and Mobile Agent System (AgentSpace) and the EASTER must be installed in the device.

We expect in the future that when a certain shelter device becomes usable by movements of a user or purchases of a service, the mobile computing device will automatically detect and use it. Currently, however, the user must assign shelter devices he wants to use.

4.3 Agent's Evacuation Behavior

4.3.1 Callback Methods

Each of the application agents must execute the built-in process before and after an evacuation. For example, an agent may discard unnecessary data and GUI components or stop Java threads before an evacuation. Or the agent may restart its operation or migrate to other devices after an evacuation. The functions to be executed before and after an evacuation, however, are not common to all applications but depend on each individual application. Therefore, we provide callback methods, which are executed before and after an evacuation. The callback methods are interfaces of an abstract class, which is the super-class of an application agent in the EASTER. An application developer only describes each built-in process in the callback methods. We provide three types of callback methods, as listed Table 1. This enables an agent to execute an evacuation appropriate to the situation. The evacuation flow is described in 4.3.3.

4.3.2 Evacuation Policy

In the EASTER, each agent does not evacuate by itself but follows evacuation plans provided by the Crisis Management Center. It is necessary, however, to reflect each agent's policy on an evacuation. We call this an evacuation policy whose contents are as follows.

- agent's level of importance (levels $1 \sim 3$)
- resource's level of importance (low or high)
- desirable shelter device
- desirable evacuation timing

4.3.3 Evacuation Flow

An application agent starts its operation after informing its evacuation policy to the Crisis Management Center. While an application operates carries out its operation, it need not keep track of the condition of battery and other agents; that is the Crisis Management Center's job. When an agent must evacuate, the Crisis Management Center gives an evacuation command to the agent.

The evacuation command includes an alert level that is divided into three stages. Each agent executes a callback method and discards unnecessary resources according to the alert level. Table 2 illustrates the evacuation behavior of agents.

4.4 Crisis Management Center

The Crisis Management Center provides the following functions for making efficient evacuation plans in order to prevent rush crush.

1. information management
2. estimation of evacuation time
3. making evacuation plans
4. dynamic modification of evacuation plans

4.4.1 Information Management

The information, in addition to each agent's evacuation policy, which is necessary for making evacuation plans, is as follows.

- battery life
- bandwidths between a mobile computing device and shelter devices

These pieces of information need to be checked periodically because they vary constantly.

4.4.2 Estimation of the Evacuation Time

Predicting each agent's evacuation time is necessary in order to make an appropriate evacuation plan. The flow of an individual agent's evacuation is as follows.

(1) executing a callback method depending on the alert level indicated by the Crisis Management Center

(2) transforming an agent into transfer data

(3) migration of the transfer data to a shelter device

We call (1) and (2) pre-evacuation process and (3) migration process.

To measure the time for pre-evacuation process, the EASTER makes a copy of each agent and has it execute a pre-evacuation process. We call this sequence a rehearsal process. There are three types of pre-evacuation process according to the three alert levels. Moreover, as mentioned in 4.1, we prepare two methods of data transfer (zip or non-zip). Thus, the Crisis Management Center executes six types of rehearsal processes for every agent.

It is possible to estimate the time required for a migration process by using formula (1) if the transfer data size and the bandwidth are known.

$$MigrationTime(sec) = TransferDataSize(bit)/Bandwidth(bps) \cdots (1)$$

Transfer data size can be predicted by the copied agent's transfer data that results from its rehearsal process. Bandwidth can be calculated by sending data periodically to a shelter device from a mobile computing device and measuring the elapsed time.

In a real situation, some agents and the Crisis Management Center work in parallel. Thus, it is difficult to make exact predictions because their operations often come into conflict. Moreover, predictions differ significantly from actual value in the following situations.

- the amount of data that an agent holds changes dramatically after its most recent rehearsal process
- the pre-evacuation process in the most recent rehearsal process differs from that of the actual evacuation
- there is a delay in the migration process due to a change in the network condition

As the result, the Crisis Management Center needs to modify an evacuation plan when the predicted migration time differs from the actual value. Moreover, in order to have the most up-to-date information as possible, the EASTER must carry out the rehearsal process periodically.

4.4.3 Making Evacuation Plans

After the rehearsal processes of all agents have been performed, the Crisis Management Center can devise evacuation plans.

First, the Crisis Management Center selects the shelter device to which each agent evacuates. In principle, the Crisis Management Center selects the device whose bandwidth is the widest among the available shelter devices. However, if a desirable shelter device is assigned in the evacuation policy of an agent, the Crisis Management Center evacuates the policy owner agent to the assigned shelter device.

The selection of a shelter device enables the estimation of evacuation time because the rehearsal process is completed and the migration process can be predicted by formula (1). Thus, the Crisis Management Center determines a transfer method for every alert level. The Crisis Management Center compares the estimation time of zip migration method and non-zip migration method and selects the shorter migration method for every alert level.

Finally, the Crisis Management Center determines the evacuation order and the alert level for each agent by the algorithm shown in Figure 4. As a rule, the Crisis Management Center evacuates agents in the order of their importance level. For agents of the same level, the Crisis Management Center evacuates agents in short order of the predicted evacuation time in order to evacuate the maximum possible number of agents. $A = \{a_1, a_2, \cdots, a_N\}$ represents a permutation of agents. Order(A,3), order(A,2) and order(A,1) present operations which order the permutation of agents A with the alert levels low, medium, and high. $T_n^1, T_n^2 and T_n^3$ represent predicted evacuation times of an agent $a_n (1 \leq n \leq N)$ with the alert levels low, medium and high. $C_n^1, C_n^2 and C_n^3$ are evacuation commands to an agent $a_n (1 \leq n \leq N)$ with the alert levels low, medium and high. C is the permutation of evacuation commands that results in this process.

From the results of this process, we can calculate the start time of the evacuation plan from formula (2).

$$StartTime = CurrentTime + (LifeTime - AllTime) \cdots (2)$$

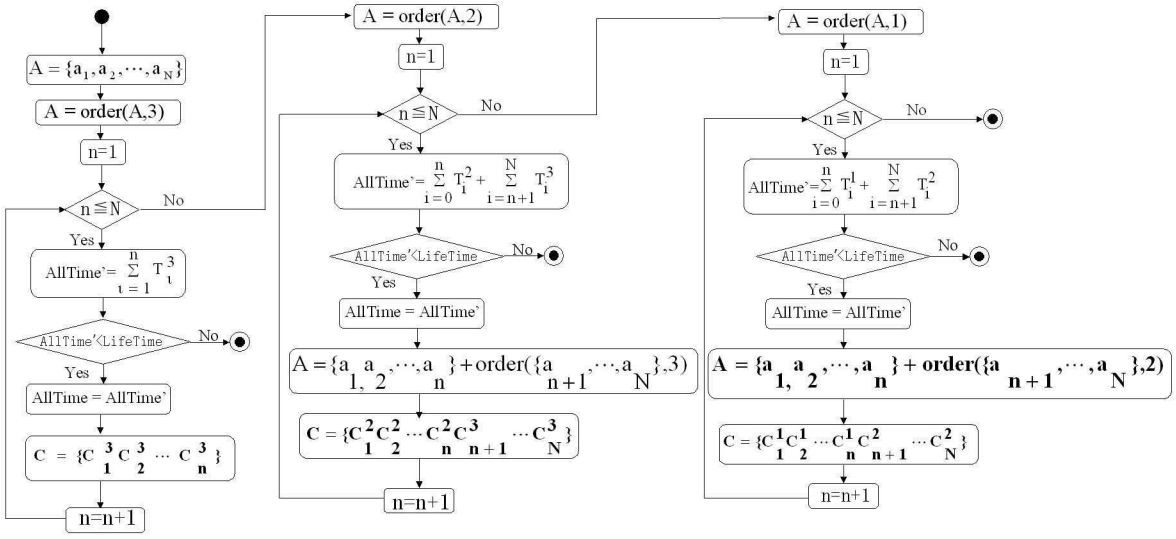

Figure 4. Algorithm for an evacuation plan.

4.4.4 Modification of an Evacuation Plan

The appropriate evacuation plan can evacuate all agents successfully in ideal situations. In practice, however, there are several external factors that come into play, such as the user's operations or changes of the network environment with the movement of mobile computing devices. Such external factors may affect the prediction. Moreover, as mentioned in 4.4.2, it is impossible to eliminate accidental error from the prediction. Thus, the Crisis Management Center must observe the surrounding situation after having made an evacuation plan and modify the evacuation plan when the prediction is found to be incorrect.

For example, as soon as a network disconnection is detected between the shelter device and the mobile computing device, the Crisis Management Center cancels the most recent plan and replans. Or, when the accidental error exceeds the threshold that is assigned in advance, the Crisis Management Center cancels and replans.

4.5 Use of the EASTER

An application developer implements his application as a subclass of the abstract class EscapeAgent as shown in Figure 5. Then the application has an ability to adapt in the event the battery runs out. When an evacuation is necessary, callback methods are executed automatically by the EASTER.

A user only executes an application that is implemented as above on the EASTER. There are, however, operational risks if the evacuation plan is set to end just before the battery runs due to accidental error in the prediction.

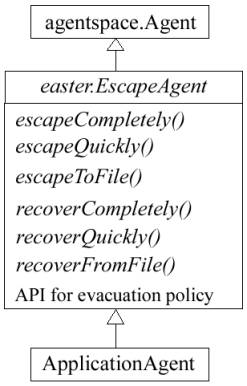

Figure 5. Class hierarchy of application in the EASTER.

Therefore, we introduced the concept of SafetyTime, which shows the time required to accomplish an evacuation plan before the battery runs out. A user can assign the value of SafetyTime, and thereby, the value of LifeTime in Figure 4 is shown in formula (3).

$$LifeTime = BatteryLife - SafetyTime \quad (3)$$

Also, a user can set the evacuation policies or the period of examination of the network conditions.

When a user wants to set a device as a new shelter device, he installs and starts the EASTER on the device. When a user wants to register a new shelter device, he can assign it as discussed in 4.2. A user can register a number of shelter devices, and the Crisis Management Center selects the one to be used for an evacuation, as described in 4.4.3.

Table 3. Implementation environment.

mobile computing device	COMPAQ iPAQ 3600
OS on mobile computing device	pocketPC (WindowsCE ver3.0)
OS on shelter device	Windows2000
wireless network	IEEE802.11b
mobile agent system	AgentSpace
JVM on the mobile computing device	jeode(Personal Java 1.2)

4.6 Implementation Environment

Table 3 shows the current implementation environment for the EASTER.

4.6.1 AgentSpace

AgentSpace is a Java-based mobile agent system. It provides the load and save function of a mobile agent, agent migration between AgentSpace systems, and communication between agents. Figure 6 depicts the structure of AgentSpace. AgentSpace transforms an agent into bit data that consists of code and state by using Java serialization function in order to migrate an agent. Moreover, AgentSpace zips the bit data to shorten migration time.

We rehandled AgentSpace in two points. First, as described in 4.1, we appended the non-zip migration method to AgentSpace because the zip migration method takes a long time on a PDA . Next, we appended the escape and recover mechanism, which is the same as the dispatch and arrive mechanism prepared in AgentSpace except for providing the handling of information peculiar to EscapeAgent such as escape policy and alert level. Currently, the boundary between the EASTER and the Mobile Agent System, which is described in Figure 3, is unclear in the implementation. In future, we want to clearly delineate the EASTER and AgentSpace (Mobile Agent System) to enable the various Mobile Agent Systems to be freely replaceable.

4.6.2 Access to the Battery

The EASTER employs a Java-based mobile agent system to enable any application to run on any device after evacuation. The battery information cannot be accessed from JVM because it is a native resource. Therefore, we attempted to access the battery data by using Win32API GetSystemPowerStatus() through Java Native Interface (JNI). However, we were only able to get the capacity of the battery only in 10% units and could not get the exact battery life. Thus, in this study, we have decided to use virtual values for battery life.

However, performing an evacuation consumes some energy. Particularly, the wireless network interface is a significant consumer of power. As a result, the battery life may decrease more rapidly than predicted. Although we expect this problem can be resolved by the modification function of the EASTER, we must prepare the environment whose battery life can be obtained in detail.

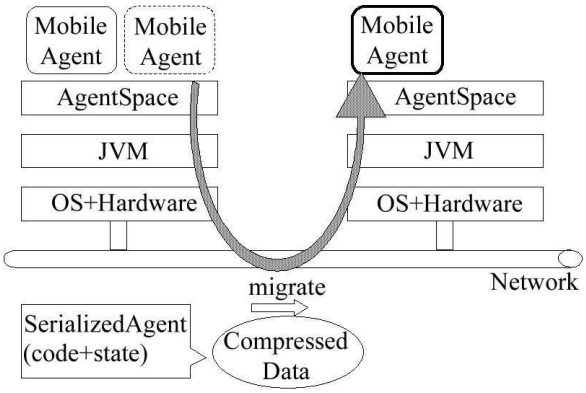

Figure 6. System architecture of AgentSpace.

5 Evaluation

We carried out several experiments to evaluate the EASTER. The agent used in the following experiments was a simple console program. The agent's operations were only to accept to the user's input and generate a simple message. The average bandwidth between iPAQ and the shelter device was 50Kbps.

5.1 Occurrence of Rush Crush

We carried out experiments with the following two scenarios to observe the agent's behavior when the EASTER evacuates two or more agents to another device.

- evacuating agents sequentially
- evacuating agents simultaneously

Figure 7 shows one of the experimental results obtained when evacuating five agents. The transferred data size of each agent was about 870KB.

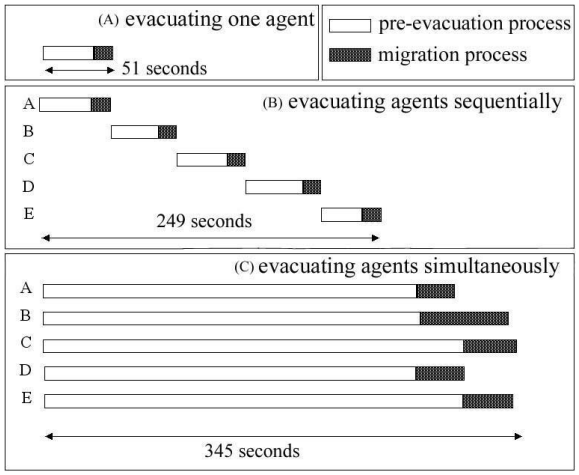

Figure 7. Evacuation of two or more agents.

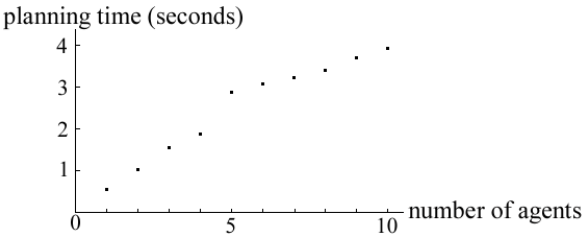

Figure 8. Time required to make an evacuation plan.

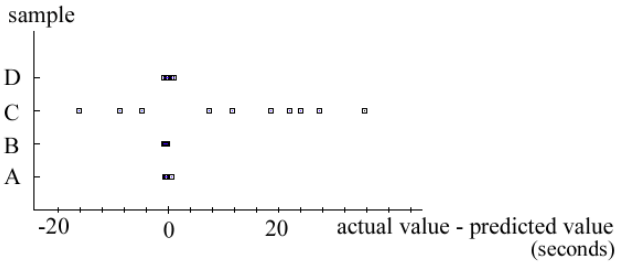

Figure 9. Pre-evacuation process.

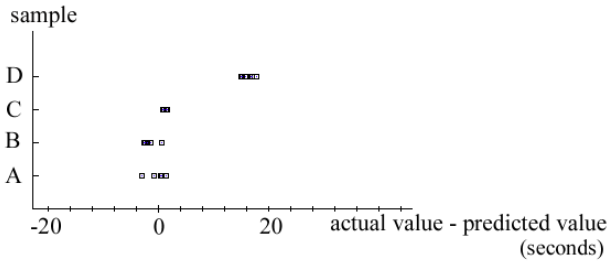

Figure 10. Migration process.

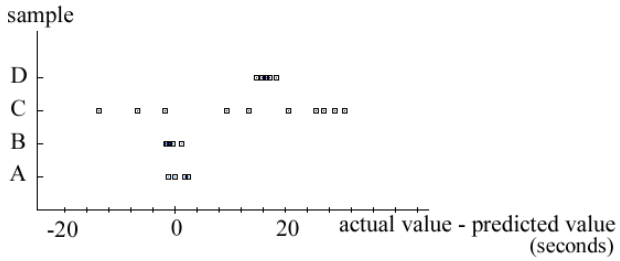

Figure 11. Entire evacuation process.

When the Crisis Management Center evacuated agents sequentially, the behavior of each agent was the same as evacuating individually. On the other hand, the evacuation time increased significantly in simultaneous evacuation. Competition among Java threads, which operate to migrate each agent, resulted in increasing the evacuation time. This result indicates the occurrence of the rush crush situation discussed in Section 3 but is effectively handled by the Crisis Management Center proposed in this paper.

5.2 Required Time to Make an Evacuation Plan

Figure 8 shows the relation between the number of agents and the time it takes to make an evacuation plan. Although the time required for making an evacuation plan is not short, we still considering it to be not impractical.

5.3 Prediction Accuracy

Exploring the relationship between the time predicted by a rehearsal process and the actual time is important. We investigated the difference between the predicted time and the actual time for the pre-evacuation process, the migration process, or the entire evacuation process, for the following four types of agents. The results are presented in Figures 9 ∼11.

A pre-evacuation process required about 10 seconds (10KB transfer data size)

B pre-evacuation process required about 10 seconds (1010KB transfer data size)

C pre-evacuation process differed each time (10KB transfer data size)

D pre-evacuation process required about 10 seconds and adds 1000KB data after the latest rehearsal process (the transfer data size changed from 10KB for the latest rehearsal process to 1010KB for the actual evacuation)

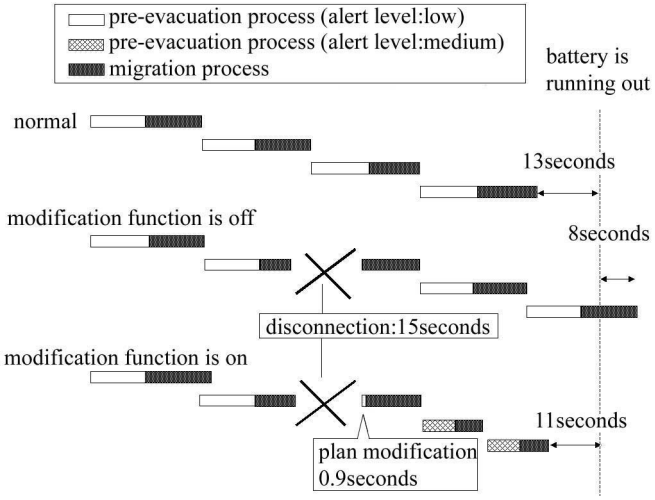

Figure 12. Modification function operation upon network disconnection.

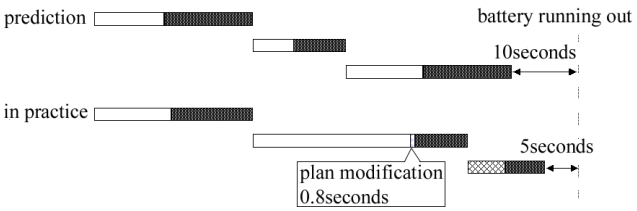

Figure 13. Error generation→modification.

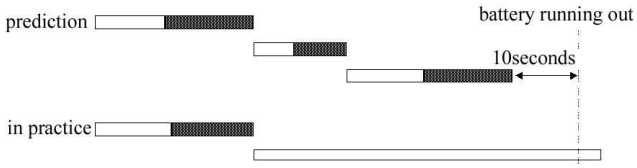

Figure 14. Error generation→failure.

The results of samples A and B show that the prediction function of the EASTER is operable in actual situations. The results of samples C and D show, however, that the actual behavior differs from that predicted in the situation described in 4.4.2.

5.4 Modification Facility of a Plan

5.4.1 When Wireless Network Is Disconnected

When using a mobile computing device, the network connection frequently changes due to the user's movement. A function for modifying an evacuation plan is useful for dealing with such changes. We explored how agents behave when the network is temporarily unavailable in order to validate the modification function.

Figure 12 shows that there existed some agents that could not be evacuated before the battery ran out when the evacuation plan was executed without modification. On the other hand, when the modification function was available, the EASTER could evacuate all agents successfully, although some agents must have been evacuated at a higher alert level. The modification function is important for evacuating maximum possible number of agents.

5.4.2 When Actual Behavior Differs from Prediction

We investigated how the modification function of an evacuation plan acts when the actual behavior differs from the prediction. Some of the results obtained are shown in Figures 13 and 14. In this case, the threshold, which is used to detect the accidental error discussed in 4.4.4, was assigned as five seconds and the second agent's actual pre-evacuation process differed the prediction.

In Figure 13, all agents could be evacuated successfully because the Crisis Management Center detected that the accidental error between the actual behavior and the prediction exceeds the threshold, cancelled the rest of plan and replaned when the second agent's pre-evacuation process was executed. In Figure 14, in contrast, two agents could not be evacuated because callback methods cannot be cancelled once executed even if the actual behavior is found to differ from the prediction, while the migration process can be cancelled any time. To reduce the damage induced by this phenomenon as far as possible, we added the function that records the history of the rehearsal process and evacuates an agent whose rehearsal results differ each time after evacuating all other agents at the same level of importance.

6 Related Work

Recently, the power-saving approach that migrates power-consuming process from a mobile computing device to a fixed server has been proposed [7], [8]. This approach enables a power-consuming process to be migrated over wireless networks to a server that performs the actual computation, and the results are migrated back. Furthermore, Rudenko et al. have described that the cost of process migration over wireless network is significant and developed the Remote Processing Framework that decides where a task should be run locally or remotely [9]. While this research considers only the power-saving, Flinn have developed Spectra, a remote execution system that balances the competing goals of performance, energy conservation, and application quality [4]. Although these researches studies

are similar to the EASTER in their approach that use a mobile program technique, their goal of power-saving is different from that of our study. As mentioned in Section 1, the power-saving approach cannot resolve the risk caused by the battery running out.

Acharya et al. have performed a research study that uses a mobile program in order to adapt resource changes on a mobile computing device [1]. They have developed Sumatra that is an extension of Java that supports resource-aware mobile programs. This language provides programs with mobility so that they can adapt to resource changes. This research facilitates development of resource-aware mobile programs and resembles the EASTER in terms of its basic concept and goal. The resources treated in this research are, however, network latency, network bandwidth, server load, and CPU cycles; the battery is not referred to in that paper. Moreover, in this research, applications are left to judge the adaptation timing on their own. To avoid rush crush, applications in the EASTER do not judge the evacuation timing on their own; it is the Crisis Management Center that makes an evacuation plan.

There are a few researches that investigate the energy consumption on a mobile computing device. Todd et al. have described an energy model and an execution-driven simulator incorporating this model for the PalmOS [2].Flinn and Satyanarayanan have developed PowerScope, a tool measuring application energy usage [3]. These tools obtain the data of energy consumption by using a monitor device such as oscilloscope and Digital Multimeter. We may have to use these monitoring devices to obtain the detailed battery life data.

7 Conclusions and Future Work

In this research, we have implemented the EASTER and performed some experiments to confirm its usability and validity. The saving of operation information and execution of application when the battery runs out is crucial to a system that cannot stop and must run continually. Moreover, applications that run automatically without a user or other normal applications can benefit substantially from the EASTER as described in Section 2.

However, we used virtual battery life because we could not obtain the detailed battery life data. In the real world, performing an evacuation consumes some energy, and as a result, the battery life may decrease more rapidly than predicted. Although we expect this problem can be resolved by the modification function of the EASTER, real experiments must be executed.

Currently, we do not conduct detailed surveys on the EASTER's performance. In the future, we will explore its performance in various operating systems, hardware, and networks, and implement and evaluate practical applications. Moreover, in its current implementation, the boundary between the EASTER and the Mobile Agent System is unclear. We will reimplement the EASTER so as to clearly delineate the Mobile Agent System in order to enable the Mobile Agent System to be freely changed.

References

[1] Anurag Acharya, M. Ranganathan, and Joel Salt. "Sumatra: A Language for Resource-Aware Mobile Programs." Mobile Object Systems: Towards the Programmable Internet Lecture Notes in Computer Science 1219. Springer Verlag. 1997.

[2] Todd L. Cignetti, Kirill Komarov, and Carla Schlatter Ellis. "Energy Estimation Tools for the Palm." ACM MSWiM 2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems. 2000.

[3] Jason Flinn and M.Satyanarayanan. "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications." Second IEEE Workshop on Mobile Computing Systems and Applications. 1999.

[4] Jason Flinn, SoYoung Park, and M.Satyanarayanan. "Balancing Performance, Energy, and Quality in Pervasive Computing." 22 nd International Conference on Distributed Computing Systems (ICDCS'02). 2002.

[5] Danny B. Lange and Mitsuru Oshima. "Programming and Deploying JAVA Mobile Agents with Aglets." Addison Wesley. 1998.

[6] ObjectSpace, Inc. "ObjectSpace Voyager Technical Overview." ObjectSpace, Inc. 1997.

[7] Mazliza Othman and Stephen Hailes. "Power Conservation Strategy for Mobile Computers Using Load Sharing." Mobile Computing and Communications Review. 1998.

[8] Alexey Rudenko, Peter Reiher, Gerald Popek, and Geoff Kuenning. "Saving Portable Computer Battery Power Through Remote Process Execution." ACM Mobile Computing and Communication Review (MC2R). 1998.

[9] Alexey Rudenko, Peter Reiher, Gerald J.Popek, and Geoffrey H.Kuenning. "The Remote Processing Framework for Portable Computer Power Saving." Proceedings of the 1999 ACM Symposium on Applied Computing. 1999.

[10] Ichiro Satoh. "AgentSpace." http://research.nii.ac.jp/~ichiro. 1997.