A Formalism for Hierarchical Mobile Agents

Ichiro Satoh

Department of Information Sciences, Ochanomizu University*/
Japan Science and Technology Corporation
E-mail: ichiro@is.ocha.ac.jp

Abstract

This paper presents a theoretical and practical framework for constructing and reasoning about mobile agents. The framework is formulated as a process calculus and has two contributions. One of the contributions can model not only individual mobile agents but also a group of mobile agents because the calculus allows more than one mobile agent to be dynamically organized into a single mobile agent. The other contribution can exactly model many features of actual mobile agents, such as mobility and marshaling, which are often ignored in other existing frameworks but may seriously affect the correctness of mobile agents. To demonstrate the utility of the calculus, we constructed a practical mobile agent system whose agents can be naturally and strictly specified and verified in the calculus. The system also offers a security mechanism for mobile agents by using well-defined properties of the calculus.

1 Introduction

Mobile agents can travel from computer to computer under their own control. They can provide a convenient, efficient, and robust framework for implementing distributed applications. Over the last few years, a dozen mobile agent systems have been explored (for example [8, 9, 12, 13, 18, 22]).

The correctness of each mobile agent depends not only on the results of computation within a particular computer, but also on the migration of the agents. Therefore, the construction and debugging of mobile agent programs are far more complex and difficult than those of ordinary parallel and distributed programs. We need the support of formal methods for reasoning about mobile agent programs. In the past few years, several researchers have proposed methods such as the Ambient calculus [5] and the Join-calculus [6]. However, these methods are not always fit for the development of practical mobile applications. This is because they are just theoretical frameworks. Often they cannot model several features of actual mobile agents which may seriously affect the correctness of mobile agents, for example, the process of marshaling agents and computational resources.

The goal of this paper is to investigate not only a formal model to reason about mobile agents but also a practical framework to construct mobile agent-based applications, which are available in real distributed computing settings. We construct a process calculus which permits expressing and analyzing mobile agents. Like mobile ambients, our mobile agent can be a container of other agents and migrate itself and its inner agents into another mobile agent as a whole. Furthermore, we implement a new mobile agent system for constructing practical and large-scale mobile applications. The system is based on the process calculus presented in this paper. We try to keep the implementation of the system within the calculus as much as possible so that many well-defined and useful properties of the calculus are inherited. We believe that the calculus presented in this paper can provide a theoretical framework for the mobile agent system, in particular a security mechanism for checking whether a moving agent is valid or not in order to protect the system against invalid or malicious agents.

In the next section, we discuss some of the related work. In Section 3, we define a process calculus for hierarchical mobile agents. Section 4 presents a mobile agent system based on the calculus and how to write agent programs in the system. In Section 5 we describe the current status of an implementation of the system. Section 6 gives some concluding remarks.

2 Background

Currently, a lot of mobile agent systems have been released (for example see Aglets [9], Mole [18], Odyssey [7], Telescript [22], and Voyager [13]). To our knowl-

*2-1-1 Otsuka Bunkyo-ku Tokyo 112-8610, Japan, Tel: +81-3-5978-5388, Fax: +81-3-5978-5390 edge, all existing mobile agent systems, including mobile object systems, lack any theoretical basis. Thus, it is difficult to verify and guarantee the correctness of mobile agents.

There is another problem in the development of mobile agent-based applications. A large-scale application software is often constructed as a collection of subcomponents. Consequently, a mobile application needs to be migrated as a whole with all its subcomponents. This is because when such an application is moved, there is a possibility that if some subcomponents were left behind; the moving application could no longer rely on the functions offered by the remaining subcomponents. Therefore, we need a mechanism for combining a group of mobile agents into a single mobile agent, like component-based software development technology [20].

However, existing mobile agent systems cannot structurally assemble more than one mobile agent into a mobile agent since each mobile agent is basically designed as an isolated entity which always acts and migrates independently. Although several systems provide the notion of inter-agent communication, they can couple mobile agents *loosely*. Mole introduces the notion of agent groups in order to encourage coordination among mobile agents [3]. Mole's agent groups can consist of agents working together on a common task but are not mobile. Also, Telescript and Odyssey introduce the concept of places in addition to mobile agents. Places are agents which can contain mobile agents and places inside them, but are not mobile. This is due to a lack of practical applications of existing mobile agent systems.

On the other hand, several researches have explored theoretical models for mobile agents (for example Distributed π-calculus [14], the Nomadic π-calculus [19], the Join calculus [6], the Ambient calculus [5], and the Seal calculus [21]).

Distributed π-calculus and the Nomadic π-calculus are extensions of π-calculus. They have the notion of locality but cannot express any structure of mobile agents themselves as opposed to the tree structure of the Join calculus and the nested structure of mobile ambients.

The join-calculus [6] introduces a notion of named locations which forms a tree and the mobility of an agent is modeled as a transformation of subtrees from one part of the tree to another. The calculus assumes a kind of a global name server and it allows an agent to directly migrate to its unique destination. Several distributed implementations of the calculus are available (for example [10]). However, the calculus lacks any mechanism to assemble more than one agent.

The ambient calculus [5] allows mobile agents (called ambients in the calculus) to contain other agents and to move as a whole with all its subcomponents. The calculus models the mobility of agents as a navigation along a hierarchy of agents, whereas in real mobile agent systems agents are directly transmitted to other computers over a communication channel. The calculus has been implemented in Java [4]. However, the implementation is not inherently designed to be executed in any distributed system settings. The Seal calculus [21] is similar to the mobile ambients and ours in its expressiveness of hierarchical structure of mobile agents, but its main purpose is to reason about the security mechanism of mobile agents.

Unlike other existing theoretical frameworks, our calculus can represent marshalled agents to transmit over network, because the process of marshalling of mobile agents is crucial in the computation of mobile agents. Moreover, it allows mobile agents to be freely moved into any agents in the same agent hierarchy, unlike the ambient calculus and the Seal calculus.

In an earlier paper [16], the author presented a practical mobile agent system, called *MobileSpaces*, which is build on the Java virtual machine and can dynamically aggregate a group of mobile agents, like the system presented in this paper. However, the main purpose of the MobileSpaces system is to construct an extensible and adaptive mobile agent system. The MobileSpaces system is characterized in that it can dynamically adapt its functions to its execution environments by migrating agents that offer the functions. Also, the previous paper [16] lacks any theoretical foundation on its mobile agent system.

On the other paper, in this paper we try to extend the MobileSpaces system in order to construct mobile applications which are large in scale and complicated. The implemented system presented in this paper can provide a practical framework for mobile agent-based applications and has paid as much attention to keep obeying our calculus for hierarchical mobile agents as possible.

3 Framework

Before formally defining our framework, we summarize its basic idea. Like other mobile agents, our mobile agents are computational entities. When each agent migrates, not only the code of the agent but also its state can be transferred to the destination. Furthermore, our mobile agent model has the following unique concepts.

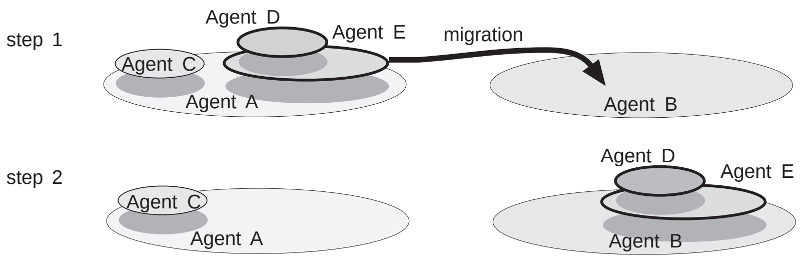

Figure 1: Agent Hierarchy and Inter-agent Migration.

- **Agent Hierarchy:** Each mobile agent can be contained within one mobile agent.
- **Inter-agent Migration:** Each mobile agent can migrate between mobile agents as a whole with all its inner agents.

Mobile agents are organized in a tree structure. Figure 1 shows an example of an inter-agent migration in an agent hierarchy. When an agent contains other agents, we call the former agent a *parent* and the latter agents *children*. We call the agents which are nested by an agent, the *descendent* agents of the agent. We call the agents which are nesting an agent, the *ancestral* agents of the agent, unlike the ambient calculus. Each agent can freely move into any agents in the same agent hierarchy except into itself or its descendants.

The first concept enables us to construct a mobile application by organizing more than one mobile agent, instead of constructing a large and monolithic mobile agent. The second concept allows a group of mobile agents to be treated as a single mobile agent. This is needed for the development of a mobile application, because a large-scale mobile application is often composed of a collection of subcomponents. Consequently, our mobile agents can be viewed as the mobile software components that have been studied in component-based software development technology [20].

3.1 The calculus

Our framework is defined as a process calculus like the join calculus [6] and the ambient calculus [5]. It is characterized in that it can model several features of mobile agents; for example marshaling of agents, which are often ignored in other existing frameworks, but may seriously affect the correctness of mobile agents.

Definition 3.1 Let $\mathcal{N}$ be an infinite set of agent names, ranged over by n,n_1,n_2,. . .. □

The framework describes mobile agents by means of the expressions defined below.

Definition 3.2 The set $\mathcal{M}$ of mobile agent expressions, ranged over by $M, M_1, M_2, \ldots$ is the smallest set containing the following expressions:

$$\begin{array}{rcl} S & ::= & \mathbf{0} \quad | \quad A \quad | \quad \mathbf{go}(n)\ .\ S \\ & | & \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\ M & ::= & n : \{\![S \,|\, M]\!\} \quad | \quad M_1\ ,\ M_2 \end{array}$$

where A is an element of a constant agent expression and is always used in the form $A \stackrel{\text{def}}{=} S$ and b is an element of the boolean set $\{true, false\}$. □

The intuitive meaning of constructors in $\mathcal{M}$ and $\mathcal{S}$ is as follows:

- $\mathbf{0}$ represents a terminated agent program.
- **if** b **then** S_1 **else** S_2 behaves as S_1 if b is true, otherwise S_2. We assume that b is given either *true* or *false* before evaluating this expression.
- $n : \{\![S \,|\, M]\!\}$ represents a mobile agent named n and behaves as sequential program S with its children denoted M. The program is executed sequentially and can instruct the agent to move. The descendent agents can be executed in parallel.
- $\mathbf{go}(n)\ .\ S$ instructs the agent and its inner agents to migrate to its destination agent named n, and then behaves as S. If there is not any agent named n, the agent is suspended until a time when such an agent exists.
- $M_1\ ,\ M_2$ represents two mobile agents M_1 and M_2 which may execute in parallel.

The operational semantics of MobileSpaces is given as a labeled transition system, written $M_1 \xrightarrow{n:\langle M_2\rangle} M_1'$. For example, $M_1 \xrightarrow{n:\langle M_2\rangle} M_1'$ means that mobile agent M_1 sends/receives a marshaled mobile agent M_2 and then behaves as M_1'. Throughout this paper we will use a structural congruence ($\equiv$) over expressions. This is the method followed by Milner in [11] to deal with the π-calculus. The use of structural congruence allows us to abstract away the static structure of networks.

Definition 3.3 $\equiv$ is the least syntactic congruence defined by the following equations:

(i) **if** *true* **then** S_1 **else** $S_2 \equiv S_1$

if *false* **then** S_1 **else** $S_2 \equiv S_2$

(ii) $A \equiv S$ if $A \overset{\text{def}}{=} S$

(iii) $M_1 , M_2 \equiv M_2 , M_1 \qquad M , \mathbf{0} \equiv M$

$M_1 , (M_2 , M_3) \equiv (M_1 , M_2) , M_3$

where we assume α-equivalence and syntactic equivalence are included in $\equiv$. We abbreviate transitive closure of $\equiv$ to $\equiv$. □

Definition 3.4 The set $\mathcal{L}$ of transition labels ranged over by $\alpha, \alpha_1, \alpha_2, \ldots$ is the smallest set containing the following expressions:

$$\alpha \quad ::= \quad n \downarrow \langle M \rangle \quad | \quad n \uparrow \langle M \rangle \quad | \quad \tau$$ □

$\langle M \rangle$ represents an agent which is marshaled into a bitstream that can be easily transferred over a network. $n \downarrow \langle M \rangle$ represents the reception of a marshaled agent M whose destination is n, $n \uparrow \langle M \rangle$ represents the dispatch of M to an agent named n, and τ is an internal execution. We now present the operational semantics of our calculus.

Definition 3.5 $\longrightarrow \subseteq \mathcal{M} \times \mathcal{L} \times \mathcal{M}$ (written $M \xrightarrow{n:\langle M \rangle} M'$) is the least relation that is given by the following axioms and rules.

$$\textbf{OUT}: \quad n_1 : \{\![\mathbf{go}(n_2)\ .\ S \mid M]\!\} \xrightarrow{n_2 \uparrow \langle n_1 : \{\![S \mid M]\!\} \rangle} \mathbf{0}$$

$$\textbf{IN}: \quad n : \{\![S \mid M_1]\!\} \xrightarrow{n \downarrow \langle M_2 \rangle} n : \{\![S \mid M_1 , M_2]\!\}$$

$$\textbf{MOVE}: \quad \frac{M_1 \xrightarrow{n \downarrow \langle M_3 \rangle} M_1' \quad M_2 \xrightarrow{n \uparrow \langle M_3 \rangle} M_2'}{M_1 , M_2 \xrightarrow{\tau} M_1' , M_2'}$$

$$\textbf{LOCAL}: \quad \frac{M \xrightarrow{\alpha} M'}{n : \{\![S \mid M]\!\} \xrightarrow{\alpha} n : \{\![S \mid M']\!\}}$$

$$\textbf{PARALLEL}: \quad \frac{M_1 \xrightarrow{\alpha} M_1'}{M_1 , M_2 \xrightarrow{\alpha} M_1' , M_2}$$

$$\textbf{STRUCT}: \quad \frac{M_1 \equiv M_2 \quad M_1 \xrightarrow{\alpha} M_1' \quad M_1' \equiv M_2'}{M_2 \xrightarrow{\alpha} M_2'}$$

where we often abbreviate transitive closure of $\xrightarrow{\tau}$ to $\longrightarrow$. □

The above labeled transitions have the following intuitive meaning:

- the **OUT** axiom transforms agent S named n_1 and its inner agents M into $\langle n : \{\![S' \mid M]\!\} \rangle$, corresponding to a marshaled agent whose destination is an agent named n_2.
- the **IN** axiom transforms a marshaled agent and its inner agents whose destination is n into an agent named n.
- the **MOVE** rule means that a marshaled agent moving to n is received if there is an agent named n.
- the **LOCAL** rule enables agent migration to occur locally.

Example 3.6 We show partial transitions of some basic expressions.

- A simple migration:
  $n_1 : \{\![\mathbf{go}(n_2)\ .\ S_1 \mid M_1]\!\} , n_2 : \{\![S_2 \mid M_2]\!\}$
  $\xrightarrow{\tau} \quad n_2 : \{\![S_2 \mid M_2 , n_1 : \{\![S_1 \mid M_1]\!\}]\!\}$
- Marshaling and Dispatching an agent:
  $n_1 : \{\![S_1 \mid M_1 , n_2 : \{\![\mathbf{go}(n_4)\ .\ S_2 \mid M_2]\!\}]\!\}$
  $\xrightarrow{n_4 \uparrow \langle n_2 : \{\![S_2 \mid M_2]\!\} \rangle} \quad n_1 : \{\![S_1 \mid M_1]\!\}$
- Receiving a serialized agent:
  $n_3 : \{\![S_3 \mid M_3 , n_4 : \{\![S_4 \mid M_4]\!\}]\!\} \xrightarrow{n_4 \downarrow \langle n_2 : \{\![S_2 \mid M_2]\!\} \rangle}$
  $n_3 : \{\![S_3 \mid M_3 , n_4 : \{\![S_4 \mid M_4 , n_2 : \{\![S_2 \mid M_2]\!\}]\!\}]\!\}$

Our framework intends to introduce mobile agents as mobile software components and it intends to construct a large-scale mobile application as a compound mobile agent, which contains mobile agents corresponding to its subcomponents. Therefore, it is very convenient to formulate a congruence relation to guarantee substitutability between two mobile agents.

Definition 3.7 A binary relation $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{M}$ is a *bisimulation* if $(M_1, M_2) \in \mathcal{R}$ implies, for all $\alpha \in \mathcal{L}$;

(i) $\forall M_1'$: $M_1 \overset{\alpha}{\Longrightarrow} M_1'$ then
$\exists M_2'$: $M_2 \overset{\alpha}{\Longrightarrow} M_2'$ and $(M_1', M_2') \in \mathcal{R}$.

(ii) $\forall M_2'$: $M_2 \overset{\alpha}{\Longrightarrow} M_2'$ then
$\exists M_1'$: $M_1 \overset{\alpha}{\Longrightarrow} M_1'$ and $(M_1', M_2') \in \mathcal{R}$.

where $M \overset{\alpha}{\Longrightarrow} M'$ is given as $M\,(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^*\, M'$. We let "$\approx$" denote the largest bisimulation, and we call M_1 and M_2 *equivalent* if $M_1 \approx M_2$. □

From the above definition, we easily see that $M \approx M$, $M_1 \approx M_2$ if $M_2 \approx M_1$, and $M_1 \approx M_3$ if $M_1 \approx M_2$ and $M_2 \approx M_3$.

The following property can be proved by the action induction on the structure of expression M.

Theorem 3.8 equivalence $\approx$ is preserved by all operators. □

By this theorem we guarantee that if two mobile agents are equivalent, the agents can substitute for one in any context.

4 Implementation

This section presents a mobile agent system which can accomplish mobile agents based on the calculus presented in the previous section. Hereafter, we describe some features of the system.

4.1 The Runtime System

The runtime system is a platform for executing and migrating mobile agents. It is built on the Java virtual machine.

Agent Hierarchy Management: Each runtime system maintains an agent hierarchy that is implemented as a tree structure where each node contains a name, an agent program, and the references of the nodes of its child agents. Each node corresponds to an expression formed in $n : \{\![S \mid M]\!\}$. The system can transform the tree structure to migrate agents in the agent hierarchy according to the rules presented in Definition 3.5. It can also be abstracted as a stationary agent at the root node of the tree structure.

Agent Migration Management: When an agent is transferred over network, the runtime system marshals the agent and its descendent agents into a bit-stream. The runtime system can transfer the bit-stream to the destination computer by using an application-layered protocol for agent transmission whose mechanism is extended of the HTTP protocol over TCP/IP communication. On the receiver side, the runtime receives the bit-stream and then reconstructs an agent and its descendent agents. The process of migrating agents over network corresponds exactly to the **OUT** and **IN** axioms.

In the current implementation of our system, each agent is transformed into a bit stream formed in Java's JAR file format that can support digital signatures, allowing for authentication. It uses the Java object serialization package for marshaling agents. The package does not support the capturing of stack frames of threads. Consequently, our system cannot serialize the execution states of any thread objects.[1] Instead, when an agent is serialized, the core system propagates certain events to its descendent agents in order to instruct the agent to stop its active threads, and then automatically stops and serializes them after a given time period.

[1]This limitation is not serious in the development of real mobile agent-based applications, as discussed in [18].

Agent Execution Management: According to the **PARALLEL** rule, mobile agents can be executed in parallel. The system controls the execution of agents and activities of agents are implemented by the Java thread library.

Security Mechanism: Security is essential in mobile agent computing. The current implementation of the system relies on the JDK 1.1 security manager and it provides a simple mechanism for authentication of agents. Moreover, since our system allows mobile agents to be software components, a mobile agent embedded in a system can be dynamically replaced by another agent whose behaviors are equivalent to those of the original agent. We intend to offer a security mechanism based on the bisimulation presented in Definition 3.7 into a mobile agent system in order to analyze whether an old mobile agent can be replaced by a new one.

However, the process of completely verifying whether two agents are bisimilar or not is too heavyweight to be used for runtime checking of receiving agents. Therefore, in the current implementation of our system, each program for agents is statically identified by a 64-bit hash of its methods and fields. The runtime system checks whether a new agent can offer the same methods and fields of the old one according to the 64-bit identifier. An agent can be dynamically replaced by a new agent which is equipped with all the public methods supported by the original one, but the current implementation of our system does not allow the new agent to inherit any internal state of the old agent.

4.2 Mobile Agent

Each agent has to be an instance of a subclass of abstract class `Agent` as shown below. The `Agent` class consists of certain methods invoked in the life cycle of a mobile agent. Each agent has a globally unique identifier and one or more activities.

```
public class Agent extends MobileObject {
  // (un)registering a context for its children
  void addChildrenContext(Context context){ ... }
  void removeChildrenContext(Context context){ ... }
  // registering listeners to hook certain events
  void addDefaultListener(
    DefaultEventListener listener){ ... }
  void removeDefaultListener(
```

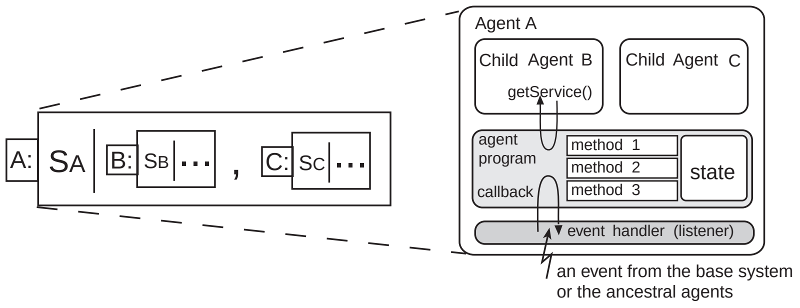

Figure 2: the expression of a hierarchical mobile agent and its implementation.

```
    DefaultEventListener listener){ ... }
  // registering a name in an environment variable
  void register(String name, String value)
    throws ... { ... }
  // migrating the agent specified as url1 to
  // the target agent specified as url2
  void go(AgentURL url)
    throws NoSuchAgentException ... { ... }
  void go(AgentURL url1, AgentURL url2)
    throws NoSuchAgentException ... { ... }
  // asking its parent agent a message
  void getService(Message msg)
    throws NoSuchMethodException ... { ... }
  // issuing an event to an agent specified as url
  void dispatchEvent(AgentURL url, AgentEvent evt)
    throws ... { ... }
  ....
}
```

go(n) . S in our calculus is accomplished with the `go()` method in the `Agent` class. When an agent invokes the `go()` command, the agent is moved to the destination agent.

```
go(new AgentURL("matp://
         some.where.com/agent1/agent2"));
```

where `matp` denotes a protocol for agent migration. When an agent performs the above command, the agent migrates itself and its descendants into the `/agent1/agent2` mobile agent located at the host (addressed as `some.where.com`). The other primitives of the calculus, for example **if** b **then** S_1 **else** S_2 and A can be accomplished through control flows of the Java language.

The runtime system maintains the life-cycle of agents: initialization, execution, suspension, and termination. When the life-cycle state of an agent is changed, the system can invoke certain methods of agents registered as listener objects. For example, the `DefaultEventListener` interface specifies a listener object whose methods are invoked by the system when the agent is created, destroyed, serialized, and migrated to another agent and when visiting agents enter and leave from it.

```
interface DefaultEventListener
  extends AgentEventListener {
  // invoked after creation at url
  void create(AgentURL url);
  // invoked before termination
  void destroy();
  // invoked after accepting a child
  void add(AgentURL child);
  // invoked before removing a child
  void remove(AgentURL child);
  // invoked after arriving at the destination
  void arrive(AgentURL dst);
  // invoked before moving to the destination
  void leave(AgentURL dst);
  ....
}
```

The above interface specifies fundamental methods invoked by the core system, when agents are created, destroyed, persisted, and migrated to another agent.

Several existing mobile agent systems introduce the concept of places. Though places are not mobile, they can provide their own services and resources for visiting agents. Similarly, our parent agents can be responsible for providing their own services and resources for its children. A child agent can call methods given in its parent agent by calling the `getService()` method defined in the `Agent` class.

On the other hand, each agent has direct control of its descendent agents. That is, an agent can instruct its descendent agents to move to other agents, serialize them and destroy them. Each agent can delegate certain events to its descendent agents so that they do something.[2] In contrast, each agent has no direct control of its ancestral agents.

5 The Current Status

The MobileSpaces mobile agent system has been implemented in the Java language (JDK1.1 or later version).

[2]The current implementation of MobileSpaces permits a parent agent to obtain references to the Java objects corresponding to its descendants.

The core system is constructed independently of the underlying system and can run on any computer with a 1.1-compatible Java runtime. We have tried to keep the implementation within the framework as much as possible.[3]

Even though our implementation was not built for performance, we have performed a basic experiment of agent migration for two cases: agent migration in an agent hierarchy and agent migration between two computers (Pentium II-300 MHz with WindowsNT 4.0 and JDK 1.1.8) connected by 10BASE-T Ethernet.

5.1 Performance

To evaluate the cost of agent migration, we examined a basic experiment of agent migration between two computers.

Table 1: The time of agent migrations (msec)

	time
agent migration between two computers	30
agent migration in an agent hierarchy	5

The first result is the time of an agent migration in an agent hierarchy, and includes the cost to check whether the visiting agent is permitted to enter the destination agent or not. In the second experiment, agent migration is supported by the runtime system allocated on two computers. Each runtime system can communicate with the other by using an application-level protocol between TCP/IP sockets. On the sender side, the runtime system serializes and transfers the codes and state of an agent (including its inner agents) to the transmitter on the receiver side and waits for an acknowledgment message. The marshaled agent consists of its serialized state, its codes, and its attributes such as name and capability, and is packed and compressed into a bit-stream which amounts to 1.5Kbytes. The second result is the sum of the marshaling, zip-based compression, opening TCP connection, transmission, security verifications, decompression, unmarshaling, and closing TCP connection.

5.2 Examples

Various mobile application running on our mobile agent system have been developed so far, for example workflow management, CSCW, distributed information retrieval, active networks, and so on (for example see our other paper [16]).

[3]An implementation of the mobile agent system, including its examples is available from `http://islab.is.ocha.ac.jp/`.

One of the most important examples is applications based on the concept of compound documents like OpenDoc developed by Apple Computer and IBM [1]. Our agent hierarchy allows compound documents given as mobile agents to be dynamically composed into a compound document, while traditional mobile agents are isolated programs and thus cannot support any compound documents.

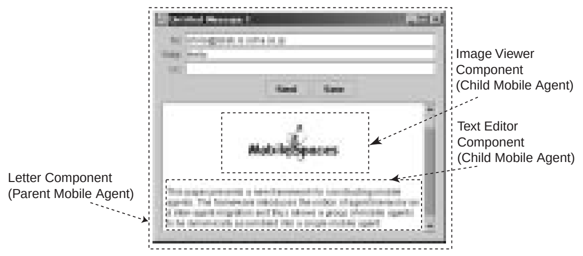
Image Viewer Component (Child Mobile Agent)
Text Editor Component (Child Mobile Agent)
Letter Component (Parent Mobile Agent)
Send
Save
MobileSpaces

Figure 3: Window of the Compound Letter Agent

We have constructed an electronic mail system where each letter is a mobile agent incorporated with the framework presented in this paper. Therefore, each letter can contain more than one mobile agent-based component: some text, graphics, and animations on the document of the letter as shown in Figure 3 and 4. Users can edit these inner components written in arbitrary data formats, because they are mobile agents and thus can include programs to edit their own contents. For example, to edit the text, simply click on it, and its editor program is invoked. The letter agent can autonomously deliver itself and its inner components to the destination. The receiver can read all the contents of the arriving letter, because the letter is a mobile agent that contains its components to view the contents.

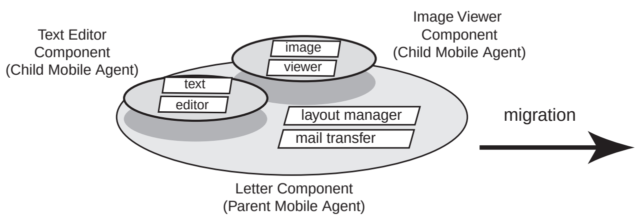
Text Editor Component (Child Mobile Agent)
Image Viewer Component (Child Mobile Agent)
image
viewer
text
editor
layout manager
mail transfer
migration
Letter Component (Parent Mobile Agent)

Figure 4: Structure of the Compound Letter Agent

6 Conclusion

We have reported a project to construct a formal and practical framework for mobile agents. While other work in the area is focusing on a unification between a theoretical foundation and its practical implementation. The framework introduces two new notions: agent hierarchy and inter-agent migration. These notions allow a group of mobile agents to be dynamically assembled into a single mobile agent. The framework consists of a process calculus for reasoning mobile agents and a mobile agent system. The calculus can model many features of mobile agents that are incorporated with these notions. The mobile agent system is based on the Java language and can migrate hierarchical mobile agents over network. The implementation of the system has tried to follow the calculus as much as possible. This framework provides not only a formal model to reason about mobile agents but also a powerful framework to construct a mobile application that is large in scale and complicated.

This work is still working in progress. The original goal of our calculus is to construct a security mechanism for checking whether receiving mobile agents are valid or not in hierarchical mobile agent settings. However, the calculus and its algebraic properties are too costly to be used as a security mechanism in runtime. Therefore, we are developing a type-theoretic framework based on the calculus. Also, in our previous paper [15], we constructed an algebraic framework for algebraic framework for optimizing parallel programs and are interested in applying the framework to our calculus.

Acknowledgements

We would like to thank anonymous referees for providing many constructive and valuable suggestions.

References

[1] Apple Computer Inc., OpenDoc: White Paper, Apple Computer Inc., 1994.

[2] K. Arnold and J. Gosling, The Java Programming Language, Addison-Wesley, 1996.

[3] J. Baumann and N. Radounklis, Agent Groups in Mobile Agent Systems, Conference on Distributed Applications and Interoperable Systems, 1997.

[4] L. Cardelli, Ambient, Available from http://www.luca.demon.co.uk/Ambit/Ambit.html

[5] L. Cardelli and A. D. Gordon, Mobile Ambients, Foundations of Software Science and Computational Structures, LNCS, Vol. 1378, pp. 140–155, 1998.

[6] C. Fournet, G. Gonthier, J. Levy, L. Marnaget, and D. Remy, A Calculus of Mobile Agents, Proceedings of CONCUR'96, LNCS, Vol. 1119, pp.406-421, Springer, 1996.

[7] General Magic, Inc. Introduction to the Odyssey, http://www.genmagic.com/agents, 1997.

[8] R. S. Gray, Agent Tcl: A Transportable Agent System, CIKM Workshop on Intelligent Information Agents, 1995.

[9] B. D. Lange and M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998.

[10] C. Le Fessant, The JoCAML system prototype, from http://join.inria.fr/jocaml, 1998.

[11] Milner, R., Parrow. J., Walker, D., A Calculus of Mobile Processes, Information and Computation, Vol.100, p1-77, 1992.

[12] D. S. Milojicic, W. LaForge, and D. Chauhan, Mobile Objects and Agents (MOA), USENIX Conference on Object Oriented Technologies and Systems, April 1998.

[13] ObjectSpace Inc, ObjectSpace Voyager Technical Overview, ObjectSpace, Inc. 1997.

[14] J. Riely and M. Hennessy, Distributed Processes and Location Failures, ICALP'97, LNCS, Vol. 1256, pp.471-481, Springer, 1997.

[15] I. Satoh, An Algebraic Framework for Optimizing Parallel Programs, Proceedings of Symposium on Software Engineering for Parallel and Distributed Systems, pp.28–38, IEEE Press, April, 1998.

[16] I. Satoh, MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System, to appear in Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2000), IEEE Press, April, 2000.

[17] I. Satoh, A Hierarchical Model of Mobile Agents and Its Multimedia Applications, to appear in Proceedings of Workshop on Multimedia Network Systems, IEEE Press, July, 2000.

[18] M. Strasser and J. Baumann, and F. Hole, Mole: A Java Based Mobile Agent System, Proceedings of ECOOP Workshop on Mobile Objects, 1996.

[19] P. Swell, P. T. Wojciechowski, and B. C. Pierce, Location-Independent Communication for Mobile Agents: A Two-Level Architecture, Workshop on Internet Programming Languages, LNCS, Vol. 1686, Springer, 1998.

[20] C.Szyperski, Component Software, Addison-Wesley, 1998.

[21] J. Vitek, Seal: A Framework for Secure Mobile Computations, Workshop on Internet Programming Languages, LNCS, Vol. 1686, Springer, 1998.

[22] J. E. White, Telescript Technology: Mobile Agents, General Magic, 1995.