Regular Paper

Bridging Physical and Virtual Worlds with Mobile Agents

Ichiro Satoh[†,††]

This paper presents a general-purpose framework for building and managing location-aware applications in ubiquitous computing settings. The goal of the framework is to provide people, places, and things with computational functionalities to support and annotate them. Using location-tracking systems such as RF (radio-frequency) or IR (infrared)-based sensors, the framework can navigate Java-based mobile agents to stationary or mobile computers near the locations of the entities and places to which the agents are attached, even when these locations change. It allows a mobile user to access personalized services available from computing devices in the environment, and provides location-dependent services to portable computing devices. The paper presents basic concepts of the framework and describes its prototype implementation and several practical applications, including follow-me applications and a navigational assistance system.

1. Introduction

As envisioned by Weiser[21)], the goal of ubiquitous computing is to enhance computer use by making multiple computers available throughout the physical environment, but, in effect, making them invisible to the user. When articulated, this might seem exotic and unrealistic, but after a decade of progress in hardware we now have many clues as to how to make this vision a reality. Computers interconnected through wired or wireless networks are present in almost every room of a modern office and are rapidly invading our homes. Another important characteristic of ubiquitous computing is that it integrates the physical world with cyberspace. Perceptual technologies have made it possible to measure and track the locations of people, computers, and practically any other object we care about. For example, indoor location systems, such as radio frequency (RF) or infrared (IR) identification tags, detect the locations of physical entities in a building and allow applications to respond to these locations. Positioning and tracking systems are likely to become even more ubiquitous in the future. Location-awareness is becoming an essential feature of software applications, especially for applications targeted at ubiquitous and mobile computing settings.

Several researchers have explored such location-aware services. Existing services can be classified into two approaches. The first is to make the computing devices move with the user. It often assumes that such devices are attached to positioning systems, such as GPS receivers that enable them to determine their own locations. For example, in HP's Cooltown project[6)], mobile computing devices such as PDAs and smart phones are attached to GPSs to provide location-awareness to web-based applications running on the devices. The second approach assumes that a space is equipped with tracking systems which establish the location of physical entities, including people and objects, within it so that application-specific services can be provided at appropriate computers. A typical example of this approach is the so-called follow-me application, which was studied by Cambridge University's Sentient Computing project[3)], to support ubiquitous and personalized services at computers located near users.

The two approaches are posed as polar opposites, although their final goals seem to coincide. This paper presents a framework for integrating the two approaches, to mitigate the disadvantages of one by juxtaposing them with the advantages of the other. That is, the framework does not distinguish between mobile and stationary computing devices. It also permits tracking sensors to be moved with the user and dynamically added to and removed from a space, because it dynamically creates a world model when detecting the appearance and movement of sensors, as well as physical entities including people, objects, and computing devices. Moreover, it is unique among existing location-aware systems in using mobile agent technology. It enables mobile agents to be spatially bound to people, places, and things, which the agents support and annotate. By

† National Institute of Informatics
†† Japan Science and Technology Corporation

using tracking systems, the framework dynamically deploys, such agents at stationary and mobile computing devices near the locations of the entities and places to which the agents are attached, even when the locations of the entities change.

In the remainder of this paper, we describe our design goals (Section 2), the design of our framework, called SpatialAgent, and a prototype implementation of the framework (Section 3). We also discuss our experience with several applications, which we used the framework to develop (Section 4), and briefly review related work (Section 5). We also briefly discuss some future issues (Section 6) and provide a summary (Section 7).

2. Approach

The framework presented in this paper aims to enhance the capabilities of users, particularly mobile users; things, including computing devices and non-electronic objects; and places, such as rooms, buildings and cities, by providing computational functionalities to support and annotate them.

2.1 Application-specific Services

Various kinds of infrastructures have been used to construct and manage location-aware services. However, such infrastructures have mostly focused either on a particular application or on a specific sensor technology. By separating application-specific services from infrastructures, our framework provides a general infrastructure for location-aware services and enables application-specific services to be implemented within mobile agents. Since mobile agents can travel between computers, the framework can naturally map the movements of physical entities such as people and objects to the movements of mobile agents in mobile and ubiquitous computing systems. Mobile agent technology also has the following advantages in ubiquitous computing settings.

- After arriving at its destination, a mobile agent can continue working without losing the results of work, such as the content of instance variables in the agent's program, at the source computers. Thus, the technology enables follow-me applications as proposed by Sentient Computing[3] to be operated locally at the destination computer without communication latency.
- Mobile and ubiquitous computers often have only limited resources, such as restricted CPU power and memory. Mobile agents can help to conserve these limited resources, since each agent only needs to be present at the computer during the time that the computer requires the services provided by that agent.
- Each mobile agent is locally executed on the computing device it is visiting and is able to directly access various items of equipment belonging to that device, as long as the security mechanisms of the device permit this.

2.2 Location-sensing Systems

Our goal is to realize a location-aware system in which spatial regions can be determined within a few square feet, that distinguishes between one or more portions of a room or building. Existing location-aware applications are typically tailored to a particular type of tracking or positioning technology. The framework itself is designed to be as independent as possible of any particular tracking or positioning technology, and is accompanied by one or more locating systems. It determines the positions of objects by identifying the spatial regions that contain the objects. In general, such locating systems consist of RF or IR sensors, which detect the presence of small RF or IR transmitters, often called *tags*, each of which periodically transmits a unique identifier. The framework assumes that physical entities and places are equipped with their own unique tags, and are thus automatically locatable. Each mobile agent can be tied to an radio-ID or infrared-ID tag attached to a person, place, or thing in the physical world.

2.3 Architecture

The framework consists of three parts: (1) location information servers, called LISs, (2) mobile agents, and (3) agent hosts. The first part provides a layer of indirection between the underlying locating sensing systems and mobile agents. Each LIS manages more than one sensor and provides the agents with up-to-date information on the identifiers of tags, which are present in the specific places its sensors cover instead of on tags in the whole space. The second offers application-specific services, which are attached to physical entities and places, as collections of mobile agents. The third consists of computing devices that can execute mobile agent-based applications and issue specific events to the agents running in them when location sensors detect the movement of the physical entities and places that the agents are bound to. When an LIS detects the movement of a tag, it tries to notify mobile agents attached to tags about the network addresses and capabilities of the candidate hosts that the agents should visit. Each of the agents selects one host from the list of candidates recommended by the LIS and migrates to that host. When the capabilities of a candidate host do not satisfy all the requirements of an agent, the agent itself should decide, on the basis of its own configuration policy, whether or not it will migrate itself to the destination and adapt itself to the destination's capabilities.

Our final goal is widespread building-wide and city-wide deployment. It is almost impossible to deploy and administer a system in a scalable way when all of the control and management functions are centralized. LISs are individually connected to other servers in a peer-to-peer manner and exchange information with each other. LISs and agent hosts may be mobile and frequently shut down. The framework permits each LIS to run independently of the other LISs, offers an automatic mechanism for the registration of agent hosts and sensors. This mechanism requires agent hosts and sensors to be equipped with tags so that they are locatable and can advertise their capabilities. In the framework, not only portable components but also system components, such as sensors and agent hosts, are movable. As a result, it is almost impossible to maintain a geographical model of the whole system. To solve this problem, the framework provides a demand-driven mechanism to discover the agents and agent hosts that are required.

2.4 Narrowing the Gap between Physical and Logical Mobilities

This framework can inform mobile agents attached to tags about their appropriate destinations according to the current positions of the tags. It supports four types of linkages between a physical entity such as a person, thing, or place, and one or more mobile agents as shown in **Fig. 1**.

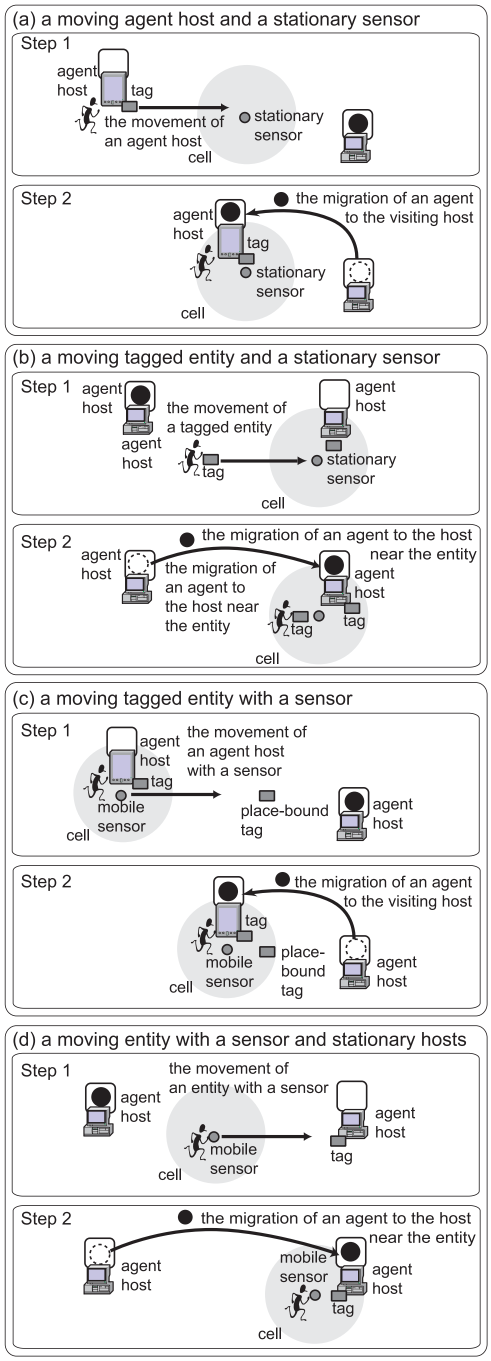
(a) a moving agent host and a stationary sensor
Step 1
agent host
tag
the movement of an agent host
stationary sensor
cell
Step 2
the migration of an agent to the visiting host
agent host
tag
stationary sensor
cell
(b) a moving tagged entity and a stationary sensor
Step 1
agent host
the movement of a tagged entity
agent host
agent host
tag
stationary sensor
cell
Step 2
the migration of an agent to the host near the entity
agent host
the migration of an agent to the host near the entity
agent host
tag
tag
cell
(c) a moving tagged entity with a sensor
Step 1
agent host
tag
the movement of an agent host with a sensor
mobile sensor
place-bound tag
agent host
cell
Step 2
the migration of an agent to the visiting host
tag
mobile sensor
place-bound tag
agent host
cell
(d) a moving entity with a sensor and stationary hosts
Step 1
the movement of an entity with a sensor
agent host
agent host
mobile sensor
tag
cell
Step 2
the migration of an agent to the host near the entity
agent host
mobile sensor
agent host
tag
cell

Fig. 1 Four linkages between physical and logical entities.

- The first type of linkage assumes that a moving entity carries more than one tagged agent host and that a space contains a place-bound tag and sensors. When the sensor detects the presence of the tag that is bound to one of the agent hosts, the framework instructs the agents attached to the tagged place to migrate to the visiting agent hosts to offer location-dependent services of the place as shown in Fig. 1 (a).
- The second type of linkage assumes that tagged agent hosts and sensors have been allocated. When a tagged moving entity enters the coverage area of one of the sen-

sors, the framework instructs the agents attached to the entity to migrate to the agent hosts within the same coverage area to offer the entity-dependent services of the entity as Fig. 1 (b) shows.

- The third type of linkage assumes that an entity carries a sensor and more than one agent host and that a space contains more than one place-bound tag. When the entity moves near a place-bound tag and the sensor detects the presence of the tag within its coverage area, the framework instructs the agents attached to the tagged place to migrate to the visiting agent hosts to offer the location-dependent services of the place, as shown in Fig. 1 (c).
- The fourth type of linkage assumes that an entity carries a sensor and that a space contains place-bound tags and tagged agent hosts. When the entity moves in the space and the sensor detects the presence of an agent host's tag within its coverage area, the framework instructs the agents attached to the moving entity to migrate to the agent hosts within the same coverage area to offer the entity-dependent services of the entity, as shown in Fig. 1 (d).

The framework allows physical places to have their own agents that support location-dependent services. When a user with network-enabled computing devices is in a given place, the framework instructs the agents attached to the place to migrate themselves to the visiting devices, where they provide the location-dependent services of the place.

Existing location-aware systems, such as Sentient Computing[3], EasyLiving[1], Cooltown[6], NEXUS[4], and user-tracking agents[18] can support only one of the above linkages, whereas our framework does not have to distinguish between the linkages and can synthesize them seamlessly.

3. Design and Implementation

This section presents the design of the SpatialAgent framework and describes a prototype implementation of the framework. **Figure 2** shows the basic structure of the framework.

3.1 Location Information Servers

All LISs can run on a stationary or mobile computer and provide the following functionality:

Management of Locating Sensors:

Each LIS manages more than one sensor that

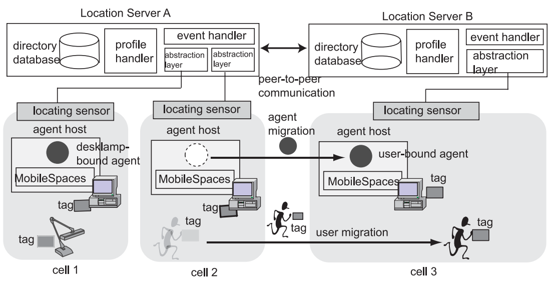

Fig. 2 Architecture of the SpatialAgent framework.

detects the presence of tags, and maintains up-to-date information on the identities of those that are within the zone of coverage by means of its sensors. This is achieved by polling the sensors or receiving events issued of the sensors themselves. An LIS does not require any knowledge of other LISs, but they need to be able to exchange their information with each other through multicast communication. To hide the differences between the underlying locating systems, each LIS maps low-level positional information from each of these to information in a symbolic model of location. An LIS represents an entity's location in symbolic terms of the unique identifier of the sensor that detects the entity's tag. We call each sensor's coverage a *cell*, as in the model of location studied by several other researchers[8]. In the framework, multiple sensors do not have to be neatly distributed in a space such as rooms or buildings to completely cover the spaces; instead, they can be placed near more than one agent host, and the coverage of sensors can overlap.

Mechanism for Agent Discovery:

Since mobile agents can travel through a network under their own control, LISs cannot always know the current locations of mobile agents. When LISs detect changes in the physical world, they inform each agent about the network address and the capabilities of more than one candidate destination that the agent should visit, but they never send the agent to the destination. Moreover, when the capabilities of such a destination do not satisfy the requirements of an agent, the agent itself should decide whether or not to migrate itself to the destination and adapt itself to the limited capabilities according to its own configuration policy.

Each LIS is responsible for discovering mobile agents bound to the tags within its cells. It maintains a database in which it stores information about each of the agent hosts and each of the mobile agents attached to a tagged entity

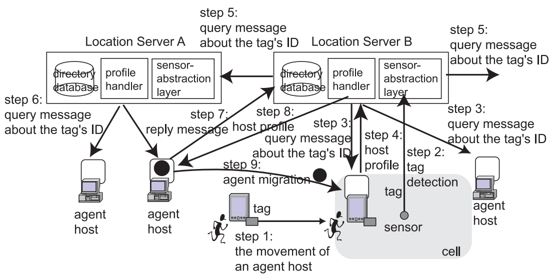

Fig. 3 Agent discovery and deployment.

or place. When an LIS detects a new tag in a cell, the LIS multicasts a query that contains the identity of the new tag and its own network address to all of the agent hosts in its current sub-network. It then waits for reply messages from the agent hosts. Here, there are two possible cases: it may be attached to an agent host or the tag may be attached to a person, place, or thing other than an agent host.

- In the first case, the newly arriving agent host will send its network address and device profile to the LIS; the profile describes the capabilities of the agent host, e.g., its input devices and screen size. After receiving this reply, the LIS stores the profile in its database and forwards the profile to all agent hosts within the cell.
- In the second case, agent hosts that have agents tied to the tag will send their network addresses and the requirements of acceptable agents to the LIS; the requirements for each agent specify the capabilities of the agent hosts that the agent can visit and perform its services at.

The LIS then stores the requirements of the agents in its database and moves the agents to appropriate agent hosts in the way discussed below. If the LIS does not have any reply messages from the agent hosts, it multicasts a query message to other LISs. When the absence of a tag is detected in a cell, each LIS multicasts a message with the identifier of the tag and the identifier of the cell to all agent hosts in its current sub-network. **Figure 3** shows a sequence for migrating an agent to a proper host when an LIS detects the presence of a new tag.

Agent Navigation:

We will explain how agents navigate to reach appropriate agent hosts. When an LIS detects the movement of a tag attached to a person or thing to a cell, it searches its database for agent hosts that are present in the current cell of the tag. It also selects candidate destinations from the set of agent hosts within the cell, according to the respective capabilities. The framework offers a language based on CC/PP (composite capability/preference profiles)[22]. The language is used to describe the capabilities of agent hosts and the requirements of mobile agents in an XML notation. For example, a description contains information on the following properties of a computing device: the vendor and model class of the device (i.e., PC, PDA, or phone), its screen size, number of colors, CPU, memory, input devices, secondary storage, and the presence/absence of loudspeakers. The framework also allows each agent to specify the preferable capabilities of agent hosts that it may visit, as well as the minimal capabilities in a CC/PP-based notation. Each LIS is able to determine whether or not the device profile of each agent host satisfies the requirements of an agent by symbolically matching and quantitatively comparing properties.

The LIS then unicasts a navigation message to each of the agents that are bound to the tagged entities or places, where the message specifies the profiles of those agent hosts that are present in the cell and satisfy the requirements of the agent. The agents are then able to autonomously migrate to the appropriate hosts. When there are multiple candidate destinations, each of the agents that is tied to a tag must select one destination on the basis of the profiles of the destinations. When one or more cells geographically overlap, a tag may be in multiple cells at the same time; agents tied to that tag may then receive candidate destinations from multiple LISs. However, since the message includes the network address of the LIS, the agents can explicitly ask the LIS about the ranges of the cells.[☆] Our goal is to provide physical entities and places with computational functionality from locations near them. Therefore, if there are no appropriate agent hosts in any of the cells at which a tag is present but there are some agent hosts in other cells, the current implementation of our framework is not intended to move agents tied to the tag to hosts in different cells.

[☆] The current implementation can specify the containment of tagged entities or spaces in a cell, but not the spatial distance between them. Other properties of sensors, such as precision and directionality, are not specified because they depend on the environment.

3.2 Agent Host

Each agent host must be equipped with a tag. It has two forms of functionality: one for advertising its capabilities and another for executing and migrating mobile agents. When a host receives a query message with the identifier of a newly arriving tag from an LIS, it replies with one of the following three responses: (i) if the identifier in the message is identical to the identifier of the tag to which it is attached, it returns profile information on its capabilities to the LIS; (ii) if one of the agents running on it is tied to the tag, it returns its network address and the requirements of the agent; and (iii) if neither of the above cases applies, it ignores the message.☆

This framework is currently implemented on a Java-based mobile agent system called MobileSpaces[13].☆☆ Each MobileSpaces runtime system is built on the Java virtual machine, which conceals differences between the platform architecture of source and destination hosts, such as the operating system and hardware. Each of the runtime systems moves agents to other agent hosts over a TCP/IP connection. The runtime system governs all the agents inside it and maintains the life-cycle state of each agent. When the life-cycle state of an agent changes for example, when it is created, terminates, or migrates to another host the runtime system issues specific events to the agent. This is because the agent may have to acquire or release various resources, such as files, windows, or sockets, which it had previously captured. When a notification on the presence or absence of a tag is received from a LIS, the runtime system dispatches specific events to the agents tied to that tag and run inside it.

3.3 Mobile Agent Program

Each mobile agent is a collection of Java objects and is equipped with the identifier of the tag to which it is attached. It is a self-contained program and is able to communicate with other agents. An agent that is attached to a user always internally maintains that user's personal information and carries all its internal information to other hosts. A mobile agent may also have one or more graphical user interfaces for interaction with its users. When such an agent moves to other hosts, it can easily adjust its windows to the screen of the new host by using the compound document framework for the MobileSpaces system that was presented in our previous paper[14]. We will next explain the programming interface for our mobile agents. Every agent program must be an instance of a subclass of the abstract class `TaggedAgent`, as follows:

```
class TaggedAgent extends Agent
  implements Serializable {
  void go(URL url) throws
    NoSuchHostException { ... }
  void duplicate() throws
    IllegalAccessException { ... }
  void destroy() { ... }
  void setTagIdentifier(TagIdentifier tid) { ... }
  void setAgentProfile(AgentProfile apf) { ... }
  URL getCurrentHost() { ... }
  boolean isConformableHost(HostProfile hpf) { ... }
  CellProfile getCellProfile(CellIdentifier cid)
    throws NoSuchCellException { ... }
  ....
}
```

We explain some of the methods defined in the `TaggedAgent` class. An agent executes the `go(URL url)` method to move to the destination host specified as the `url` by its runtime system. The `duplicate()` method creates a copy of the agent, including its code and instance variables. The `setTagIdentifier` method ties the agent to the identity of the tag specified as `tid`. Each agent can specify a requirement that its destination hosts must satisfy by invoking the `setAgentProfile()` method, with the requirement specified as `apf`. The class has a service method named `isConformableHost()`, which the agent uses to decide whether or not the capabilities of the agent hosts specified as an instance of the `HostProfile` class satisfy the requirements of the agent. The `getCellProfile()` method allows an agent to investigate the measurable range and types of the sensor specified as `cid`.

Each agent can have more than one listener object that implements a specific listener interface to hook certain events issued before or after changes in its life-cycle state or the movements of its tag.

```
interface TaggedAgentListener
  extends AgentEventListener {
  // invoked after creation at url
  void agentCreated(URL url);
  // invoked before termination
  void agentDestroying();
  // invoked before migrating to dst
  void agentDispatching(URL dst);
  // invoked after arrived at dst
  void agentArrived(URL dst);
```

☆ The current implementation assumes that LISs and agent hosts can be directly connected through a wireless LAN such as IEEE802.11b, and thus does not support any multiple-hop query mechanisms.

☆☆ The framework itself is independent of the MobileSpaces mobile agent system and can thus work with other Java-based mobile agent systems.

```
    // invoked after the tag arrived at another cell
    void tagArrived(HostProfile[] apfs,
                    CellIdentifier cid);
    // invoked after the tag left from the current cell
    void tagLeft(CellIdentifier cid);
    // invoked after an agent host arrived
    // at the current cell
    void hostArrived(AgentProfile apfs,
                     CellIdentifier cid);
    ....
}
```

The above interface specifies the fundamental methods that are invoked by the runtime system when agents are created, destroyed, or migrate to another agent host. In addition, the `tagArrived()` callback method is invoked after the tag to which the agent is bound has entered another cell, to obtain the device profiles of agent hosts that are present in the new cell. The `tagLeft()` method is invoked after the tag is no longer in a cell.

3.4 Security and Privacy

Security is essential in mobile agent computing. The framework can be built on many Java-based mobile agent systems with the Java virtual machine. Therefore, it can directly use the security mechanism of the underlying mobile agent system. The Java virtual machine can explicitly restrict agents so that they can only access specified resources to protect hosts from malicious agents. To protect against the arrival of malicious agents from agent hosts, the MobileSpaces system supports a Kerberos-based authentication mechanism for agent migration[16]. It authenticates users without exposing their passwords on the network, and generates secret encryption keys that can selectively be shared by mutually suspicious parties.

The framework only maintains per-user profile information within those agents that are bound to the user. It promotes the movement of such agents to appropriate hosts near the user in response to the user's movement. Since agents carry the profile information of their users within them, they must protect such private information while they are moving over a network.[☆] The MobileSpaces system can transform agents into an encrypted form before migrating them over a network and decrypt them after they arrive at the destination. Moreover, since each mobile agent is just a programmable entity, it can explicitly encrypt its inner particular fields and migrate itself with the fields and its own cryptographic procedure, except for its secret keys.

[☆] The framework itself cannot protect agents from malicious hosts, because this problem is beyond the scope of this paper.

3.5 Current Status

The framework presented in this paper was implemented in Sun's Java Developer Kit version 1.1 or later versions, including Personal Java. The remainder of this section discusses some features of the current implementation.

Locating Systems:

The current implementation of this framework supports two commercial locating systems: RF Code's Spider and Elpas's EIRIS. The former provides active RFID tags. Each tag has a unique identifier that periodically emits an RF-beacon conveying an identifier (every second). The system allows us to explicitly control the omnidirectional range of each of the RF readers to read tags within a range of 1 to 20 meters. The other system provides active IR-tags, which periodically broadcast their identifiers through an IR interface (every four seconds), like the Active Badge system[20]. Each IR reader has omnidirectional infrared coverage, which can be adjusted to cover distances from 0.5 to 10 meters. Although there are many differences between the two locating systems, the framework abstracts these differences away.

Performance Evaluation:

Although the current implementation of the framework was not built for performance, we measured the cost of migrating a 3 KB agent (zip-compressed) from a source host to the destination host recommended by the LIS. This experiment was conducted with two LISs and two agent hosts, each of which was running on one of four computers (Pentium III-1GHz with Windows 2000 and JDK 1.4), which were directly connected via an IEEE802.11b wireless network. The latency of an agent's migration to the destination after the LIS had detected the presence of the agent's tag was 380 msec, and the cost of agent migration between two hosts over a TCP connection was 48 msec. The latency includes the cost of the following processes: UDP-multicasting of the tags' identifiers from the LIS to the source host, TCP-transmission of the agent's requirements from the source host to the LIS, TCP-transmission of a candidate destination from the LIS to the source host, marshaling of the agent, migration of an agent from the source host to the destination host, unmarshaling of the agent, and security verification. We believe that this latency is acceptable for a location-aware system

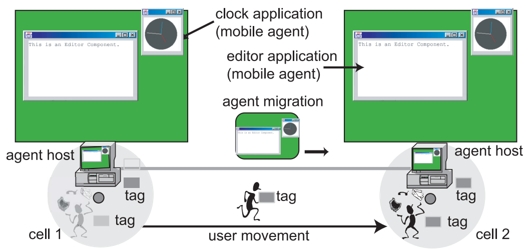

Fig. 4 Follow-Me desktop applications.

used in a room or building.

4. Initial Experience

To demonstrate the utility of the SpatialAgent framework, we developed several typical location-aware applications for mobile or ubiquitous computing settings.

4.1 Follow-Me Desktop Applications

A simple application of the framework is a desktop *teleporting* system, like a *follow-me* application[3], within a richly equipped, networked environment such as a modern office. The system tracks the current location of a user and allows him/her to access the user's applications at the nearest computer as the user moves around within the building. Unlike previous studies of such applications, our framework can migrate not only the user interfaces of applications but also the applications themselves to appropriate computers in the cell that contains the tag of the user. In a previous paper[14], we also developed a mobile window manager, which is a mobile agent that can carry its desktop applications as a whole to another computer and control the size, position, and overlap of the windows of the applications. Using the framework presented in this paper, the window manager and desktop applications can be automatically moved to and then executed at the computer that is in the current cell of the user and has the resources required by the applications in the manner described in **Fig. 4**.

4.2 User Navigation System

We also developed a user navigation system that assists visitors to a building. Several researchers have reported on similar systems[2),4]. In our system, tags are distributed through several places within a building, such as its ceilings, floors, and wall. Each visitor carries a wireless-LAN-enabled tablet PC that includes an LIS and an agent host, which is equipped with a locating sensor to detect tags. The system initially deploys place-bound agents to invisible computers within the building. When a

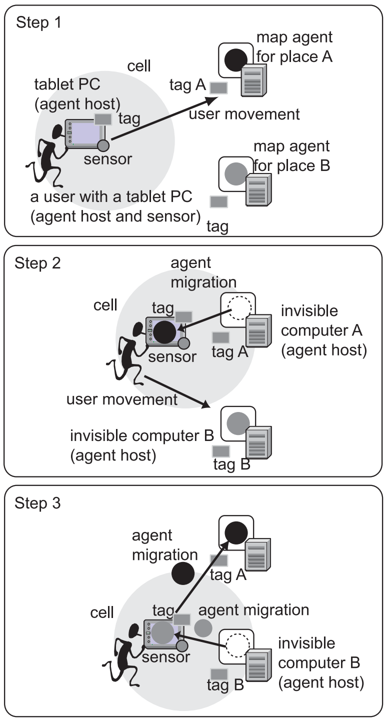

Fig. 5 The migration of an agent, which is attached to a place, to a visiting computer in the place.

tagged position is detected within the cell of the moving sensor, the LIS running on the visitor's tablet PC detects the presence of the tag. The LIS detects the place-bound agent that is tied to the tag. It next instructs the agent to migrate to its agent host and performs the agent's location-dependent services at the host. **Figure 5** shows a situation where a visitor with a sensor-equipped tablet PC and sensor is roaming, first approaching place A and then place B. The system enables one or more agents tied to place A to move to the tablet PC. The agents then return to their home computer and other agents, which are tied to place B move to the tablet PC. **Figure 6** shows a place-bound agent displays a map of the surrounding area on the screen of a tablet PC.

4.3 Proactive Control of Home Appliances

We also used this framework to implement two prototype systems to for controlling electric lights in a room. Each light was equipped with a tag and was within the range at covered by a sensor in the room. We controlled power outlets for the lights through a commercial protocol called X10. In both of the approaches we describe here, the lights were controlled by

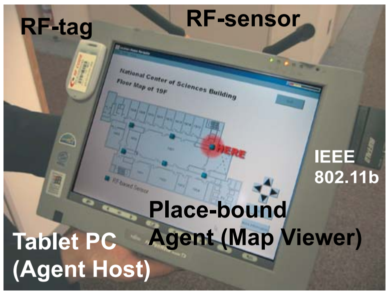

Fig. 6 A screen-shot of a map-viewer agent running on a tablet PC.

switching their power sources on or off according to the X10 protocol.

User-Aware Automatic Controller:

The first system provides proactive control of room lighting, which is a similar approach to that used by the EasyLiving project [1]. Our approach enables turn the lights in a room to be autonomously turned whenever a tagged user is sufficiently close to them. Suppose that each light is attached to a tag and is within the 3-meter range of the RF Code's Spider sensor. The tag attached to each of the lights correlates with the mobile agent, which is our X10-based server's client and runs on the stationary agent host in the room. When a tagged user approaches a light, an LIS in the room detects the presence of the user's tag in the cell that contains the light. The LIS then moves the agent bound to the tag to the agent host on which the light's agent is running. The user's agent then requests the lights' agent to turn the light on through inter-agent communication.

Location-Aware Remote Controller:

The second system allows us to use a PDA to remotely control nearby lights. In this system, place-bound controller agents, which can communicate with X10-base servers to switch lights on or off, are attached to places with room lights. Each user has a tagged PDA, which supports the agent host with WindowsCE and a wireless LAN interface.☆ When a user with a PDA visits the cell that contains a light, the framework moves a controller agent to the agent host of the visiting PDA. As shown in **Fig. 7**, the agent, now running on the PDA, displays a graphical user interface for controlling the light.

☆ Since existing Java VMs for WindowsCE-based PDAs are lacking in terms of function and performance, the current implementation of this example uses a lightweight version of the MobileSpaces system.

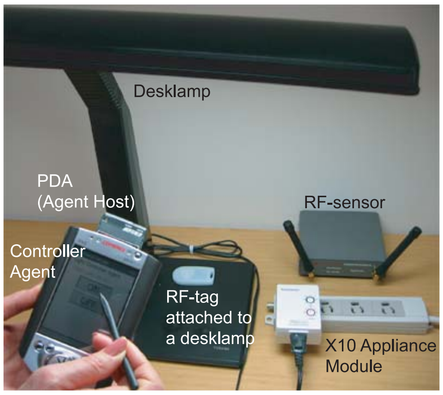

Fig. 7 Controlling a desk lamp from a PDA.

When the user leaves the location, the agent automatically closes its user interface and returns to its home host.

5. Related Work

This section discusses several systems that have influenced various aspects of this framework, which seamlessly integrates two different approaches, i.e., ubiquitous and mobile computing.

We compared several projects that support mobile users in a ubiquitous computing environment with our framework. Research on smart spaces and intelligent environments has become popular at many universities and corporate research facilities. Cambridge University's Sentient Computing project [3] provides a platform for location-aware applications using infrared-based or ultrasonic-based locating systems in a building. Using the VNC system [12], the platform can track the movement of tagged entities, such as individuals and things, so that the graphical user interfaces of the user's applications follow them while they are moving around. Although the platform provides similar functionality to that of our framework, its management is centralized and it is difficult to dynamically reconfigure the platform when sensors are added to or removed from the environment. Since the applications must be executed in remote servers, the platform may have non-negligible interactive latency between the servers and the hosts that the user accesses locally. On the other hand, our framework enables various applications, including user interfaces, to be dynamically deployed and directly run on computers close to him/her so that it can minimize temporal and spatial distances in interactions between him/her and the applications. Recently, the project provided a CORBA-based middleware system called LocARE[10]. The middleware can move CORBA objects to hosts according to the location of tagged objects, but CORBA objects are not always suitable for implementation on user interface components.

Microsoft's EasyLiving project[1] provides context-aware spaces, with a particular focus on the home and office. It uses mounted sensors such as stereo cameras on the room's walls and tracks the locations and identities of people in the room. The system can dynamically aggregate network-enabled input/output devices, such as keyboards and mice, even when they belong to different computers in the space. However, its management is centralized and it does not dynamically migrate software to other computers according to the position of users. Both the projects assume that locating sensors have initially been allocated in the room, and it is difficult to dynamically configure the platform when sensors are added to or removed from the environment, whereas our framework permits sensors to be mobile and scatteredly throughout the space.

MIT's Project Oxygen Alliance has tried to introduce intelligent spaces that are as abundant and accessible to use as oxygen in the air into people's lives by incorporating several perceptual devices, including location systems. It has provided agent-based infrastructures to construct and manage location-aware services in such spaces[9]. The goal of these infrastructures has been to offer suitable services at suitable locations within the space, based on the contextual information within the environment and emanating from users, but they have not been able to dynamically deploy service-provider services at suitable computers in the space, as we have done.

There have also been several studies on enhancing context-awareness in mobile computing. HP's Cooltown[6] is an infrastructure that supports context-aware services on portable computing devices. It is capable of automatically providing bridges between people, places, and things in the physical world and the web resources that are used to store information about them. The bridges that it forms allow users to access resources stored on the web via a browser, using standard HTTP communication. Although user familiarity with web browsers is an advantage in this system, all the services available in the Cooltown system are constrained by the limitations of web browsers and HTTP. Our framework, however, is not limited in its web-based approach and can dynamically change mobile agent-based applications, including viewer programs for location-sensitive information based on the locations and requirements of users.

The NEXUS system[4], developed by Stuttgart University, offers a generic platform that supports location-aware applications for mobile users. Like the Cooltown system, users require a PDA or tablet-PC, which is equipped with GPS-based positioning sensors and wireless communication. Applications that run on such devices (e.g., user-navigation) maintain a spatial model of the current vicinity of users and gather spatial data from remote servers. Unlike our approach, however, neither Cooltown nor NEXUS can support mobile users through stationary computers distributed in a smart environment.

Even though a number of mobile agent systems have been developed, few researchers have attempted to apply mobile agent technology to mobile and ubiquitous computing. Kangas[5] developed a location-aware augmented-reality system that enables the migration of virtual objects to mobile computers, but only when the computer was in a particular space, in a similar way to our framework. However, the system is not designed to move such virtual objects to ubiquitous computing devices. Hive[11] is a distributed agent middleware for building decentralized applications. It can deploy agents at devices in ubiquitous computing environments and organize the devices as groups of agents. Although it can provide contextual information for agents, it does not support any mechanisms for monitoring sensors and deploying agents according to changes in the environment, unlike ours.

Several researchers have explored location-sensitive servers like our LIS. Their location models can be classified into two types: spatial models based on the concrete geographical coordinates of objects and spatial models based on the geographical containment between objects. For example, the EasyLiving project provides a geometric model based on the former approach, so it accurately represents the physical relationships between entities in the world. Leonhardt[8] developed a location-tree model based on the latter approach and used location-aware directory servers. Our framework is based on a symbolic location model similar to the geographical containment model. However, it is unique in having the ability to dynamically manage spatial models. That is, it provides a demand-driven mechanism that discovers the locations of agent hosts and agents, because it permits all its elements, such as hosts and sensors to be mobile in and to be dynamically added to or removed from a space.

In a previous paper [18], we described an approach for developing location-aware mobile agents. The approach allows mobile agents to follow their users as they move, like the framework presented in this paper. However, the previous approach allows the positions of the users to be detected through a computer vision technique and it maintains a geographical model of the environment, including the positions of the users. On the other hand, our present framework uses RF or IR sensors to detect their presence, and maintains a symbolic location model in the sense that it can only detect the presence of tagged entities that are within the coverage of the sensors. Since the aim of the previous approach was to support mobile users from stationary computers in a ubiquitous computing environment, it could not support mobile users from portable computing devices, whereas our framework can support both ubiquitous and mobile computing environments. Another previous paper [17] presented an early prototype of the present framework, but did not provide four linkages between physical and virtual worlds as described in the second section of this paper.

6. Future Work

Since the framework presented in this paper is general-purpose, in future work we need to apply it to specific applications, as well as the three applications presented in this paper. The MobileSpaces system, which is the basis of this framework, allows application-specific services to be implemented as a collection of multiple agents rather than a single agent. We are now developing a mechanism that enables an application-specific service to be divided into multiple mobile agents. For example, a mobile agent-based service may often require various I/O devices, such as keyboards and speakers, but it cannot locate an agent host that has all of these. If there are two hosts, where one has a keyboard and the other host has speakers, the service can be provided by the two in combination. The current mechanism for the exchange of information between LISs is not satisfactory. We therefore plan to develop a publish-subscribe system for the framework. We have an approach for managing sensor networks [19]. The approach can dynamically customize our location information servers. We have also developed an approach to testing context-aware applications on mobile computers [15]. We are interested in developing a methodology that would test applications that were based on the framework.

7. Conclusion

We have presented a novel framework for the development and management of location-aware applications in mobile and ubiquitous computing environments. The framework provides people, places, and things with mobile agents to support and annotate them. Using location-tracking systems, the framework can migrate mobile agents to stationary or mobile computers near the locations of the people, places, and things to which the agents are attached. The framework is decentralized, and is a generic platform independent of any higher-level applications and locating systems. We also designed and implemented a prototype system based on the framework, and demonstrated its effectiveness in several practical applications.

References

1) Brumitt, B.L., Meyers, B., Krumm, J., Kern, A. and Shafer, S.: EasyLiving: Technologies for Intelligent Environments, *Proc. International Symposium on Handheld and Ubiquitous Computing*, pp.12–27 (2000).

2) Cheverst, K., Davis, N., Mitchell, K. and Friday, A.: Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project, *Proc. Conference on Mobile Computing and Networking*, pp.20–31, ACM Press (2000).

3) Harter, A., Hopper, A., Steggeles, P., Ward, A. and Webster, P.: The Anatomy of a Context-Aware Application, *Proc. Conference on Mobile Computing and Networking*, pp.59–68, ACM Press (1999).

4) Hohl, F., Kubach, U., Leonhardi, A., Rothermel, K. and Schwehm, M.: Next Century Challenges: Nexus — An Open Global Infrastructure for Spatial-Aware Applications, *Proc. Conference on Mobile Computing and Networking*, pp.249–255, ACM Press (1999).

5) Kangas, K. and Roning, J.: Using Code Mobility to Create Ubiquitous and Active Augmented Reality in Mobile Computing, *Proc. Conference on Mobile Computing and Networking*, pp.48–58, ACM Press (1999).
6) Kindberg, T., et al.: People, Places, Things: Web Presence for the Real World, Technical Report HPL-2000-16, Internet and Mobile Systems Laboratory, HP Laboratories (2000).
7) Lange, B.D. and Oshima, M.: Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley (1998).
8) Leonhardt, U. and Magee, J.: Towards a General Location Service for Mobile Environments, *Proc. IEEE Workshop on Services in Distributed and Networked Environments*, pp.43–50, IEEE Computer Society (1996).
9) Lin, J., Laddaga, R. and Naito, H.: Personal Location Agent for Communicating Entities (PLACE), *Proc. Mobile HCI'02*, LNCS, Vol.2411, pp.45–59, Springer (2002).
10) Lopez de Ipina, D. and Lo, S.: LocALE: A Location-Aware Lifecycle Environment for Ubiquitous Computing, *Proc. Conference on Information Networking*, IEEE Computer Society (2001).
11) Minar, N., Gray, M., Roup, O., Krikorian, R. and Maes, P.: Hive: Distributed agents for networking things, *Proc. Symposium on Agent Systems and Applications/Symposium on Mobile Agents* (*ASA/MA'99*), IEEE Computer Society (2000).
12) Richardson, T., Stafford-Fraser, Q., Wood, K. and Hopper, A.: Virtual Network Computing, *IEEE Internet Computing*, Vol.2, No.1 (1998).
13) Satoh, I.: MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, *Proc. Conference on Distributed Computing Systems*, pp.161–168, IEEE Computer Society (2000).
14) Satoh, I.: MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents, *Proc. Symposium on Agent Systems and Applications/Symposium on Mobile Agents*, LNCS, Vol.1882, pp.113–125, Springer (2000).
15) Satoh, I.: Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers, *Proc. Conference on Mobile Agents*, LNCS, Vol.2240, pp.103–118, Springer (Nov. 2001).
16) Satoh, I.: Dynamic Configuration of Agent Migration Protocols for the Internet, *Proc. Symposium on Applications and the Internet*, pp.119–126, IEEE Computer Society (Jan. 2002).
17) Satoh, I.: Physical Mobility and Logical Mobility in Ubiquitous Computing Environments, *Proc. Conference on Mobile Agents*, LNCS, Vol.2535, pp.186–202, Springer (Oct. 2002).
18) Tanizawa, Y., Satoh, I. and Anzai, Y.: A Mobile Agent Framework for Ubiquitous Computing Environments (in Japanese), *Journal of Information Processing Society of Japan*, Vol.43, No.12, pp.3774–3784 (Dec. 2002).
19) Umezawa, T., Satoh, I. and Anzai, Y.: A Mobile Agent-based Framework for Configurable Sensor Networks, *Journal of Information Processing Society of Japan*, Vol.44, No.3, pp.779–788 (Mar. 2003).
20) Want, R., Hopper, A., Falcao, A. and Gibbons, J.: The Active Badge Location System, *ACM Trans. Inf. Syst.*, Vol.10, No.1, pp.91–102, ACM Press (1992).
21) Weiser, M.: The Computer for the 21st Century, *Scientific American*, pp.94–104 (Sept. 1991).
22) World Wide Web Consortium (W3C): Composite Capability/Preference Profiles (CC/PP), http://www.w3.org/TR/NOTE-CCPP (1999).

(Received July 2, 2002)
(Accepted May 6, 2003)

Ichiro Satoh Ichiro Satoh received his B.E., M.E, and Ph.D. degrees in Computer Science from Keio University, Japan in 1996. From 1996 to 1997. Since 2001, he has been an associate professor in National Institute of Informatics, Japan. Also, he was a visiting researcher of Rank Xerox Laboratory from 1994 to 1995 and a PRESTO researcher of Japan Science and Technology Corporation from 1999 to 2002. His current research interests include distributed and ubiqutious computing. He received IPSJ paper award, IPSJ Yamashita SIG research award, and JSSST Takahashi research award. He is a member of six learned societies, including ACM and IEEE.