LOCATION-AWARE DESPLOYMENT OF SERVICES FOR INTELLIGENT ENVIRONMENTS

Ichiro Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 JAPAN

ABSTRACT

A framework for building and managing context-aware services for intelligent environments is presented. The goal of the framework is to provide physical entities, e.g., people and things, and places with location-aware or personalized services to support and annotate them. The framework can implement context-aware services within mobile agents or codes instead of the framework itself. It enables such service to be executed at computing devices near the positions of the entities and places that the services are bound to so that users can directly access location-based services from their portable computing devices or their personalized services from stationary computing devices in the environment. This paper presents the rationale, design, implementation, and applications.

1. INTRODUCTION

Location-awareness is becoming an essential feature of services in pervasive computing devices. Recent advances in sensing technologies enable pervasive computing devices to detect their surroundings. In fact, computing devices Global Positioning System (GPS) receivers, Radio Frequency IDentification (RFID) readers, and computer-vision cameras. These sensors have made it possible to detect and track the presence and location of people, computers, and practically any other object we want to monitor. A variety of pervasive services based on such sensors have been explored.

However, there are still several problems in existing services. Most of the services are inherently designed for their initial applications. In fact, existing services can be classified into two approaches. The first is to make the computing devices that move with the user. For example, in HP's Cooltown project [8], mobile computing devices such as PDAs and smart phones are attached to GPSs to provide location-awareness for web-based applications running on the devices. The second is to offer intelligent environments where can monitor the positions of physical entities, including people and objects, to support application-specific services at appropriate computers. A typical example of this is the so-called follow-me application, which was a study by Cambridge University's Sentient Computing project [5], to offer ubiquitous and personalized services on computers located near users.

The two approaches are developed independently, although their final goals coincide. That is, services are designed for running on mobile computing devices equipped with positioning systems cannot be reused on stationary computers with tracking sensors, vice versa. Moreover, Most existing services are designed dependently on particular sensors in the sense that they explicitly or implicitly assume low-level information measured by sensing systems, for example, geometric information measured from GPSs. As a result, software for defining services with particular sensing systems cannot be reused with other sensing systems.

This paper presents a framework for developing and operating services for mobile or ubiquitous computing independently of particular sensing systems and computing devices. In other words, the framework does not distinguish between mobile and stationary computing devices and between positioning sensors and tracking sensors. Instead, it dynamically deploys software that defining services at suitable computing devices according to changes in the physical world, e.g., the positions of people and computing devices. Moreover, it provides the deployment policies of services to support their various requirements.

In the remainder of this paper, we describe our design goals (Section 2), the design of our approach, called SpatialAgent, and a prototype framework (Section 3). We present how to bridge the gap between the physical world and cyberspace (Section 4) and discuss our experience with several applications, which we developed with the framework (Section 5). We briefly review related work (Section 6). We also provide a summary and some future issues (Section 7). Lastly, we describe programming models (Appendix).

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. All these elements must have suitable computational functionalities that enable their support and annotation. The goal of the framework is to enable people to access suitable services that they want in their associate context, e.g., from suitable computers that are located at suitable places to support the services and can satisfy the requirements of the services.

2. 1. Service-oriented Location model

There have been a variety of location-sensing systems. They can be classified into two types: tracking and positioning systems. The former, including RFID, measures the location of other located objects. The latter, including GPS, measures its own location. Tracking sensors can be embedded in the environment and positioning sensors can be carried with portable computing devices. There are two different ways to locate objects: geometric location and symbolic location.

The former represents the locations of objects as geometric information. A few outdoor-applications like moving-map navigation can easily be constructed on the former. Most emerging applications, on the other hand, require a more symbolic notion: place. Generically, place is the human-readable labeling of positions, e.g., the names of rooms and buildings. An object contained in a volume is reported to be in that place. This paper addresses symbolic location as an event-driven programming model for pervasive computing environments. For example, when people enter a place, services should be provided from their own portable terminal or their own stationary terminals should provide personalized services to assist them.

2.2. Location-aware deployment of services

Where to execute services is a major design decision. However, most services for intelligent environments should be executed at typical locations. For example, follow-me services should be provided at computers near the current positions of the users or computers inside the range of their visions rather than remote computers. Tourist-navigation services should provide location-information about the current positions in most cases. However, it is difficult to manage the location that services should be operated at according to the requirements of the services. Therefore, the framework gives a clear solution to the relocation of services. Its goal is to provide services on computing devices near the locations of people that want the services or within the locations of spaces that the services annotate. That is, suitable services should be operated on suitable computing devices in the sense that the services are wanted according to the locations of users and their associate contexts and the locations and capabilities of the devices can satisfy the requirements of the services. Although this solution may seem to be limited, it can cover most existing services for intelligent environments and makes it greatly easy to manage the locations of services.

2.3. Deployable services for intelligent environments

Most ubiquitous or mobile computers often have only limited resources, such as restricted levels of CPU power and amounts of memory. As a result, even if a computer is at suitable location for a wanted service to be provided, the computer may not be available because of a lack of software or capabilities, such as input or output facilities, for executing the software. Various kinds of infrastructure have been used to construct and manage location-aware services. However, such infrastructures have mostly focused on a particular application, such as user navigation. To solve this limitation, our framework provides three approaches. The first is to software for defining pervasive services to be composed from software components, which may run on different computers. The second is to enable software for defining pervasive services to be dynamically deployed at stationary or mobile computers by using mobile code technology and mobile agent technology. The third is to manage the location of services as well as the location of physical entities and computing devices.

3. DESIGN AND IMPLEMENTATION

This framework enables a physical entity and place to spatially bind with one or more mobile code- or agent-based services. These services annotate and support the entities or places in the sense that the services can be dynamically deployed at stationary and mobile computing devices that are near or within the locations of the entities and places. Therefore, the services can easily be customized to be person- and location-dependent. They can directly interact with their users, whereas other existing approaches, e.g. remote procedure calls and web-based interaction can be seriously affected by network latency between the client-side and service side computers.

The framework provides the middleware infrastructure for managing location-sensing systems and deploying software for defining services for intelligent environments according to the locations of users and objects, including computers. As shown in Figure 1, it consists of three parts: (1) service-provider components, which are implemented as mobile agents or codes, (2) component hosts, which are runtime systems for executing and migrating components. (3) location information servers, called LISs.

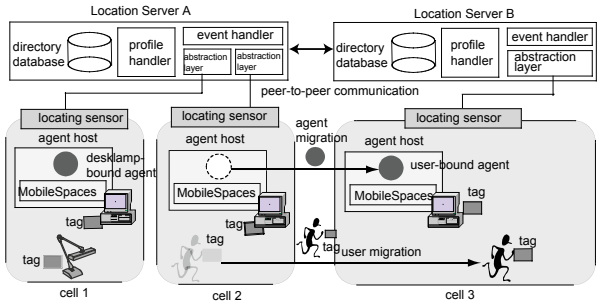

Figure 1: Architecture

3.1. Service Provider Component

The framework uses mobile agent or code technology because the technology has several advantages for ubiquitous and mobile computing settings. Each mobile code or agent can be deployable at a computing device only the time when the device is required to offer the services provided by that code or agent. Therefore, mobile code or agent technology can help to conserve the limited resources of computing devices. Moreover, When a mobile agent moves to another computer, both the code and the state of the agent is transferred to the destination. After arriving at its destination, a mobile agent can continue working without losing the results,

e.g. the content of instance variables in the agent's program, at the source computers. Since each mobile code or agent is a programmable entity, the framework enables an application-specific service, including the user interface and application logic, to be implemented within a mobile code or agent. It then separates application-specific services from itself. Therefore, it can be a general infrastructure for a variety of location-aware services. It also can directly access various equipment belonging to that device as long as the security mechanisms of the device permit this.

3.2. Component Host

Each component host offers two functionalities: advertisement of its capabilities and a runtime system for executing and migrating components. When a host receives a query message with the identifier of a newly arriving entity (or a tag attached to an entity) from an LIS, it can respond in one of the following three responses: (i) if the identifier in the message is equal to the identifier of the entity (or its tag) to which it is attached, it returns profile information about the component's capabilities to the LIS; (ii) if one of components running on its runtime system is tied to the entity (or its tag), it returns its network address and the requirements of the component; (iii) if neither case is true, it ignores the message. The current implementation of this framework is based on a Java-based mobile agent system called MobileSpaces [19]. Each component host provides a MobileSpaces runtime system built on the Java virtual machine and can move components to other component hosts over a TCP/IP connection. The runtime system governs all of the components inside it and maintains the life-cycle state of each of the components. When the life-cycle state of a component changes, for example, when it is created, terminates, or migrates to another host, the runtime system issues specific events to the component. Each component host can have its counterpart component, called a proxy component. The component is a representation of the device located in the model. When it receives service-provider components or request messages, it forwards the components or messages to the device that it refers to.

3.3. Location Information Management

The framework provides digital representations, called counterpart of components, spaces, e.g., rooms, floors, and streets, in the physical world. The components are programming entities so that they can be explicitly defined as the ranges within which services should be operated. The framework maintains containment relationships between spaces as an acyclic-tree structure of these counterpart components, like Unix's file-directory. The location management of this framework seems to be similar to that of our previous infrastructure [22,23,26], but the framework presented in this paper introduces the notion of proxy components so that it can manage the locations of computing devices and service provider components in a unified approach. Since the proxy components control the deployment of components, the framework itself is independent of any component migration mechanisms.

Sensor Management:

Each LIS manages more than one sensor and agent hosts, and maintains up-to-date information on the identities of those that are within the zone of coverage by means of its sensors. To hide the differences between the underlying location-sensors, each LIS provides an abstract three-layer stack. This can be mapped to a number of architectures to provide the acquisition function as in the acquisition stack [8]

- The *Reception* layer is responsible for extracting the data from the sensors, as sensors generally tend to be proprietary or vendor-specific. For example, some sensors can be retrieved at any time through synchronous queries and other sensors can issue results continuously or periodically. The layer polls sensors or receives events issued from sensors.
- The *Abstraction* layer receives low-level data about the locations of entities from sensors and then transforms the data in a symbolic model. For example, the current implementation maps geometric locations measured by sensors, e.g., GPS and wireless and cellular network, into specified regions, e.g., one or more portions of a room or building. When an RFID reader detects the presence of a tagged entity, the location of the entity is represented as the identifier of the reader. We call each sensor's coverage and each region a *cell*, as location models studied by several other researchers [8].
- The *Fusion* layer correlates the sightings belonging to the same located-object from different sensors. This infrastructure allows sensors to be mobile and throughout a space. When one or more cells overlap geographically, an entity may be in multiple cells at the same time and each of the LISs that manage the cells sends update information to agents bound to the entity.

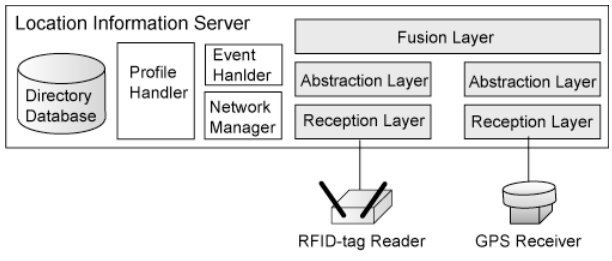

Figure 1: Location information server.

Component Discovery Mechanism:

The framework provides demand-driven mechanisms for discovering the components and component host

required. Each LIS discovers components bound to the entities (or their tags) present in its sensing spaces and maintains a database by storing information about each component host and each component attached to an entity or place. When a LIS detects the presence of a new entity (or tag) in a space, it multicasts a query message with the identify of the new entity (or tag) and its own network address to all the component hosts in its current sub-network and then waits for reply messages from the hosts. We anticipate one of two possible cases: the entity is a component host, i.e., the tag is attached to the host, or the entity is a person, place, or thing, i.e., the tag is not attached to an component host. (1) The newly arriving component host sends its proxy component to the LIS; the proxy component describes the capabilities of the host, e.g., input devices and screen size. The LIS locates the component at the counterpart component corresponding to the space that spatially contains the host. (2) Component hosts that have components tied to the entity send their network addresses and requirements of the components to the LIS; the requirements for each component specify the required capabilities of the component hosts that the component can visit and at which it performs its services at. Then, the LIS stores the requirements of the components in its database and moves the components to appropriate component hosts via the proxy components of the hosts. When the absence of an entity is detected, each LIS multicasts a message with the identifier of the entity and the identifier of the cell to all component hosts in its current sub-network. Since LISs can be individually connected to other servers, which may be in other sub-networks and with which they exchange information in a peer-to-peer manner, they can discover component hosts and components that may be in other sub-networks.

Component Deployment:
When an LIS knows the movement of an entity, e.g., a person or thing, to a cell, it tries to deploy components attached to the entity at computing devices in the cell. It searches its database for component hosts that are present in the current cell of the entity. It then selects candidate destinations from a set of component hosts within the cell based on their device capabilities. This framework offers a description language based on CC/PP [30], for specifying the properties of computing devices (vendor, model class of device, e.g., pc, pda, phone, etc., screen size, display colors, CPU, memory, input device, secondary storage, loudspeaker, etc) in XML notation. Each LIS informs each component of the profiles of the component hosts that are present in the cell and that satisfy the requirements of the component, and then the component can migrate autonomously to the appropriate host.

4. COMPONENT PROGRRAMMING

This section explains the programming interface for mobile codes or agents. The former is implemented as Java Beans and can support specified callback methods invoked by runtime systems.

```
interface ServiceListener extends
SystemEventListener {
  // invoked after the entity
  // arrives at the space
  void entityArrived(URL dst_splace);
  // invoked after the entity leaves from
  // the space
  void entityLeft(URL des_space);
  // invoked after the component host
  // arrives at the space
  void entityArrived(URL dst_space);
  // invoked after the component host
  // leaves from the space
  void entityLeft(URL des_space);
  ....
}
```

The above interface specifies the fundamental methods that are invoked by the runtime system when entities or spaces that components are bound to enter and exit from spaces or component hosts enter and exist from the spaces. On the other hand, each mobile agent-based component can also 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 the entity or tag that the component is bound to.

```
interface AgentListener extends ServiceListener {
  // invoked after creation at url
  void agentCreated(URL url);
  // invoked before termination
  void agentDestroying();
  // invoked before migrating to dst
  void agentDispatching(URL dst);
  // invoked before moving to dst
  ....
  void agentArrived(URL dst);
  // invoked after arrived at dst
  ....
}
```

This interface specifies the fundamental methods that are invoked by the runtime system when agents are created, destroyed, or migrated to another agent host.

The current implementation provides counterpart components corresponding to spaces. Each counterpart component is defined as a subclass of abstract class `CounterpartComponent`, which has some built-in methods that are used to control its mobility and life-cycle like service provider components. It is bound to at least one entity or space in the physical world.

```
class CounterpartComponent {
  void setIdentity(String name) { ... }
  void setAttribute(String attribute,
    String value){ ... }
  void add(Component comp) throws
    NoSuchComponent { ... }
```

```
  void remove(Component comp) throws
    NoSuchComponent { ... }
  ComponentInfo getParentComponent() {..}
  ComponentInfo[] getChildren() { ... }
  ....
}
```

By invoking `setIdentity`, a counterpart component can assign the symbolic name of the physical entity or space that it represents. For example, a counterpart component refers to the coverage area of an RFID reader and it has the identify of the reader. By invoking `setAttribute`, a counterpart component can explicitly record attributes about its entity or space inside it, e.g., owner, position, shape, and size. When a counterpart component invokes the `add` (or `remove`) method, it contains the component specified as `comp` inside it (or take out the component specified as `comp` from itself), where `comp` is an instance of `CounterpartComponent` (or its subclass) or a service-provider component. For example, when a user enters a room, the framework deploys a counterpart component corresponding to the user in a counterpart component corresponding to the room. People should only be able to access location-bound services, e.g., printers and lights, that are installed in a space, when they enter it carrying their own terminals or using public terminals located in the space. Therefore, the framework allows service-provider components to access attributes and services provided in their parent or ancestor components. In contrast, it has no direct access over other components, which do not contain it, for reasons of security.

There are two typical scenarios in intelligent environments as shown in Figure 2.

- Figure 2 (a) shows that a moving user carries a portable component host and sensors are located in a place. When a sensor detects the presence of the host or measures the position of the host within the place, the LIS deploys components attached to the place at the visiting host. The components can provide the moving user with location-dependent services of the place from his/her portable host.
- Figure 2 (b) shows that sensors and component hosts are located in places. When a sensor detects the movement of a user from place to place, the LIS deploys components attached to the user at the component host in the destination place so that the components provide the moving user with his/her peronalized-dependent services from the stationary host.

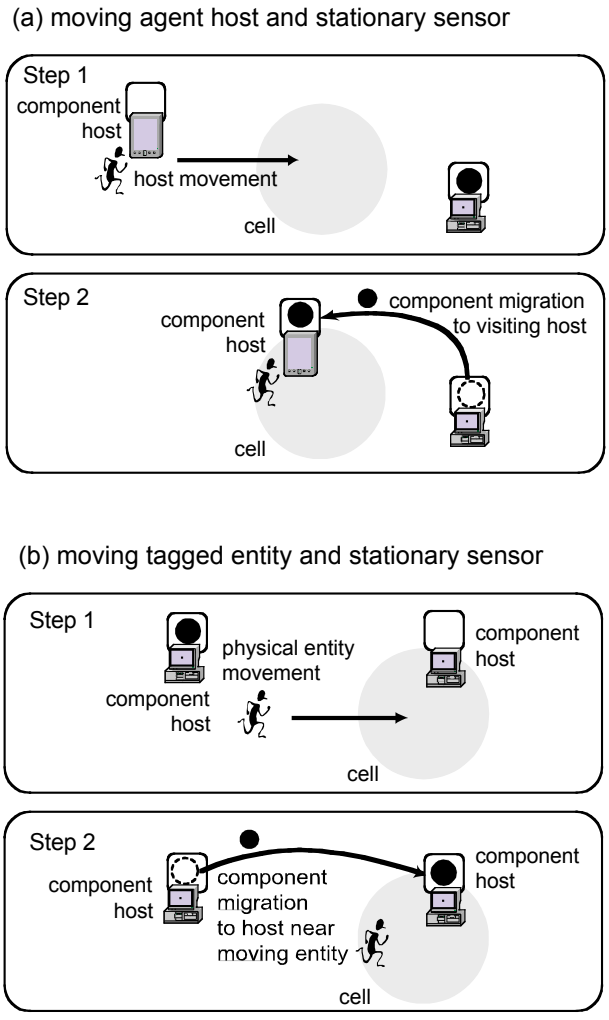

Figure 2: Two scenarios for intelligent environments.

Existing location-aware systems can only support each of the scenarios. For example, the Cooltown [8] and NEXUS[6] projects support the first and the person tracking display approach in the EasyLiving project [1] and the Follow-me applications approach in the Sentient Computing project [5] support the second. On the other hand, this framework supports the both scenarios. Components can be executed on mobile and stationary computing devices, only when the devices can satisfy the requirements of components, since this framework does not distinguish between the both computers. Moreover, the framework hides differences in sensing

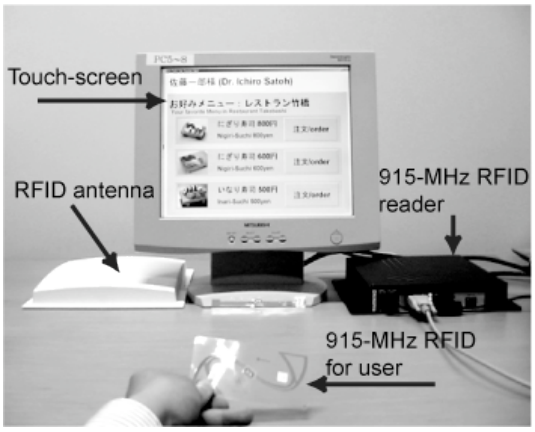

Figure 4: Screenshot of follow-me user assistant agent that selects user's favorite sushi from menu database of the restaurant in front of the user.

systems, since it maps geometric location information into symbolic location information.

5. APPLICATIONS

This section presents two typical location-based services developed using this framework. The first example tracks the current location of a user by using a 915-MHz RFID system and provides a user assistant agent that follows the user and maintains profile information about the user, so that the user can assist the agent in his/her personalized manner anywhere. Suppose that a user has a tag and is moving by a restaurant that offers a RFID reader and an agent host with a touch-screen. When the tagged user enters the coverage area of the reader, the framework enables his/her assistant agent to automatically move to the agent host near his/her current location. After arriving at the host, the agent accesses a database provided by the restaurant to obtain the restaurant menu. It then selects appropriate candidates meals from the menu based on the user's profile information, such as favorite foods and recent during experiences, stored inside the agent. Next, it displays the list of the candidate meals on the screen of the current agent host in a personalized manner. Figure 4 shows how the user's assistant agent runs on the agent host of the restaurant and seamlessly embeds the pictures, names, and prices of the candidates meals with buttons for ordering them into its graphical user interface. Since a mobile agent is a program entity, we can easily define a more intelligent assistant agent.

The second example is a user navigation system that assists visitors to a building. Active RFID tags are positioned at several places in the building in the ceilings, floors, and walls. Each visitor carries a wireless-LAN- enabled tablet PC equipped with an RFID reader to detect tags. The PC includes an LIS and an agent host. The system initially deploys place-bound agents to hidden computers within the building. When a tagged position is located in the coverage area of the moving sensor, the LIS running on the visitor's tablet PC detects the presence of the tag and detects the place-bound agent tied to the tag. It then instructs the agent to migrate to its agent host and provide the agent's location-dependent services to the host. The system enables more than one agent tied to a place to move to a tablet PC; the agent then returns to its home computer and other agents, which are tied to another place, move to the tablet PC. Figure 2 shows a place-bound agent being used to display a map of its surrounding area on the screen of a tablet PC.

6. 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 our approach with several projects that support mobile users in a ubiquitous computing

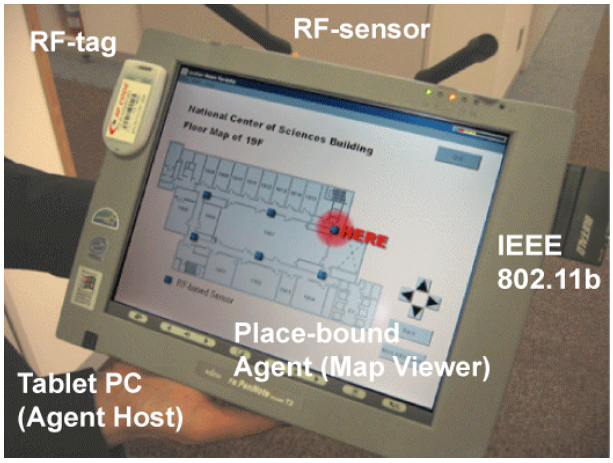

Figure 2: Screenshot of map-viewer agent running on tablet PC with positioning sensor.

environment. Research on smart spaces and intelligent environments has become popular at many universities and corporate research facilities. Cambridge University's Sentient Computing project [5] provides a platform for location-aware applications using infrared-based or ultrasonic-based locating systems in a building. Using the VNC system [16] 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 of our approach, its management is centralized and thus 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. Our approach, however, enables a user's application, including user interfaces, to be dynamically deployed and directly run on computers close to the user 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 [14]. The middleware can move CORBA objects to hosts according to the location of tagged objects. However 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 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. Our approach, however, permits sensors to be mobile and scattered throughout the space.

MIT's Project Oxygen Alliance has tried to introduce intelligent spaces that are as abundant and accessible to use as oxygen 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 [13]. The goal of these infrastructures has been to offer suitable services at suitable locations within the space based on contextual information within the environment and information emanating from users. However, 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 [8] 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 with 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 approach, however, is not limited by a 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 [6], 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.

Several research projects have introduced software mobility as a technology for enabling ubiquitous computers to support various services, which they may have not been initially designed for. The Aura project [3] of CMU and the Gaia project [17] of the University of Illinois at Urbana-Champaign provide infrastructures for binding tasks associated with users, and migrating applications from computer to computer as users move about, like our approach does. Although they share several common design goals with our framework, they focus on the development of contextual services for users rather than the location-aware deployment of services. Kangas [7] 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. The one.world project [4] by the University of Washington provides a middleware infrastructure for ubiquitous computing, but does not provide any location-aware mechanisms for deploying services at computing devices. It assumes a distributed shared memory and builds applications based on the principle of separating data and functionality in applications, where our approach always treats applications as a set of data and functionality to be deployed at various devices that is not initially designed for executing the application. Hive [15] 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 mechanism 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 concrete geographical coordinates of objects and spatial models based on 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 [11] 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 both be mobile in and to be dynamically added to or removed from a space. In previous papers [22,23,26], we presented an early prototype of the present framework. This approach does not support the mobility of sensors and agent hosts so that the four linkages described in the second section of this paper were not available in the previous framework unlike the framework presented in this paper. We will present a location model based on containment relationship between spaces and entities in the physical world in our previous paper [27], but the model aims at constructing a general-purpose model for managing location-aware services, whereas the

framework presented in this paper provides a location-aware deployment of software.

7. CONCLUSION

We presented a middleware infrastructure for managing location-sensing systems and dynamically deploying services at suitable computing devices. Using location-tracking systems the infrastructure provides entities, e.g. people and objects, and places, with mobile agents to support and annotate them and migrate agents to stationary or mobile computers near the locations of the entities and places to which the agents are attached. It is a general framework in the sense that it is independent of any higher-level applications and location-sensing systems and supports a variety of spatial linkages between the physical mobility of people and things and the logical mobility of services. Furthermore, we designed and implemented a prototype system of the infrastructure and demonstrated its effectiveness in several practical applications.

Finally, we would like to point out further issues to be resolved. 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 location model of the framework was designed for operating real location-sensing systems in ubiquitous computing environments. We plan to design a more elegant and flexible world model for representing the locations of people, things, and places in the real world by incorporating existing spatial database technologies. We have developed an approach to testing context-aware applications on mobile computers [21,24]. We are interested in developing a methodology that would test applications based on the framework.

References

1. B. L. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer, "EasyLiving: Technologies for Intelligent Environments", Proceedings of International Symposium on Handheld and Ubiquitous Computing, pp. 12-27, 2000.
2. K. Cheverst, N. Davis, K. Mitchell, and A. Friday: "Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project", Proceedings of Conference on Mobile Computing and Networking (MOBICOM'2000), pp. 20-31, ACM Press, 2000.
3. D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: Towards Distraction-Free Pervasive Computing", IEEE Pervasive Computing, vol. 1, pp. 22-31, 2002.
4. R. Grimm, el. al., "Systems Directions for Pervasive Computing", Proceedings of 8th Workshop on Hot Topics in Operating Systems, pp.147-151, May 2001.
5. Harter, A. Hopper, P. Steggeles, A. Ward, and P. Webster, "The Anatomy of a Context-Aware Application", Proceedings of Conference on Mobile Computing and Networking (MOBICOM'99), pp. 59-68, ACM Press, 1999.
6. F. Hohl, U. Kubach, A. Leonhardi, K. Rothermel, and M. Schwehm, "Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications", Proceedings of Conference on Mobile Computing and Networking (MOBICOM'99), pp. 249-255, ACM Press, 1999).
7. K. Kangas and J. Roning, "Using Code Mobility to Create Ubiquitous and Active Augmented Reality in Mobile Computing", Proceedings of Conference on Mobile Computing and Networking (MOBICOM'99), pp. 48-58, ACM Press, 1999.
8. T. Kindberg, et al, "People, Places, Things: Web Presence for the Real World",
9. Technical Report HPL-2000-16, Internet and Mobile Systems Laboratory, HP Laboratories, 2000.
10. B. D. Lange and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets", Addison-Wesley, 1998.
11. U. Leonhardt and J. Magee, "Towards a General Location Service for Mobile Environments", Proceedings of IEEE Workshop on Services in Distributed and Networked Environments, pp. 43-50, IEEE Computer Society, 1996.
12. U. Leonhardt and J. Magee: "Multi-Sensor Location Tracking", Proceedings of Conference on Mobile Computing and Networking (MOBICOM'98), pp.203-214, ACM Press, 1998.
13. J. Lin, R. Laddaga, and H. Naito, "Personal Location Agent for Communicating Entities (PLACE)", Proceedings of Mobile HCI'02, LNCS, Vol. 2411, pp. 45-59, Springer, 2002.
14. D. Lopez de Ipina and S. Lo, "LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing", Proceedings of Conference on Information Networking (ICOIN-15), IEEE Computer Society, 2001.
15. N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes, "Hive: Distributed agents for networking things", Proceedings of Symposium on Agent Systems and Applications/Symposium on Mobile Agents (ASA/MA'99), IEEE Computer Society, 2000.
16. T. Richardson, Q, Stafford-Fraser, K. Wood, A. Hopper, "Virtual Network Computing", IEEE Internet Computing, Vol. 2, No. 1, 1998.
17. M. Román, C. K. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell, K. Nahrstedt K, "Gaia: A Middleware Infrastructure to Enable Active Spaces", IEEE Pervasive Computing, vol. 1, pp.74-82, 2002.
18. K. Romer and T. Schoch, "Infrastructure Concepts for Tag-Based Ubiquitous Computing

Applications", Workshop on Concepts and Models for Ubiquitous Computing, Ubicomp 2002, September 2002.
19. I. Satoh, "MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System", Proceedings of Conference on Distributed Computing Systems (ICDCS'2000), pp. 161-168, IEEE Computer Society, 2000.
20. I. Satoh, "MobiDoc: A Framework for Building Mobile Compound Documents from Hierarchical Mobile Agents", Proceedings of Symposium on Agent Systems and Applications/Symposium on Mobile Agents (ASA/MA'2000), LNCS, Vol. 1882, pp. 113-125, Springer, 2000.
21. I. Satoh, "Flying Emulator: Rapid Building and Testing of Networked Applications for Mobile Computers", Proceedings of Conference on Mobile Agents (MA'01), LNCS, Vol. 2240, pp. 103-118, Springer, 2001.
22. I. Satoh, "Physical Mobility and Logical Mobility in Ubiquitous Computing Environments", Proceedings of Conference on Mobile Agents (MA'02), LNCS, Vol. 2535, pp. 186-202, Springer, 2002.
23. I. Satoh, "Location-based Services in Ubiquitous Computing Environments", to appear in Proceedings of International Conference on Service Oriented Computing (ICSOC'2004), LNCS, vol. 2910, pp.527-542, Springer, December 2003.
24. I. Satoh, "A Testing Framework for Mobile Computing Software", IEEE Transactions on Software Engineering, vol. 29, no. 12, pp.1112-1121, 2003.
25. I. Satoh, "Configurable Network Processing for Mobile Agents on the Internet", Cluster Computing, (Accepted) vol. 7, no.1, Kluwer, January 2004.
26. I. Satoh, "Linking Physical Worlds to Logical Worlds with Mobile Agents", Proceedings of IEEE International Conference on Mobile Data Management (MDM'04), pp. 332-343, IEEE Computer Society, January 2004.
27. I. Satoh, "A Location Model for Pervasive Computing Environments", Proceedings of IEEE 3rd International Conference on Pervasive Computing and Communications (PerCom'05), pp.,215-224, IEEE Computer Society, March 2005.
28. R. Want, A. Hopper, A. Falcao, and J. Gibbons, "The Active Badge Location System", ACM Transactions on Information Systems, vol.10, no.1, pp. 91-102, ACM Press, 1992.
29. R. Want, "The Personal Server - Changing the Way We Think about Ubiquitous Computing", Proceedings of 4th International Conference on Ubiquitous Computing (Ubicomp 2002), LNCS 2498, pp. 194-209, Springer, September 2002.
30. World Wide Web Consortium (W3C), "Composite Capability/Preference Profiles (CC/PP)", http://www.w3.org/TR/NOTE-CCPP, 1999.