A Spatial Communication Model for Ubiquitous Computing Services

Ichiro Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: ichiro@nii.ac.jp

Abstract

The paper presents an approach for location-aware communications in ubiquitous computing environments. It is constructed on a symbolic location model, which links everyday objects, including non-smart objects and places, with counterpart objects, which are executable software components that are dynamically organized like a tree based on geographical containment, such as in a user-room-floor-building hierarchy. The approach enables physical entities and places to communicate with one another through their counterpart objects, even when the entities and places are not smart. It supports a mechanism to detect objects and communicate with them according to their locations. This paper presents the design for the approach and describes an implementation of it.

1 Introduction

Recent technological advances have enabled parts of the real world to be turned into so-called *smart spaces*. In fact, computers are embedded into everyday objects, including appliances, chairs, and walls and sensing devices are already present in almost every room of modern buildings or houses and in many of the public facilities in cities. Such spaces perceive their environments with the help of sensors, interpret this information, and hence derive context based on their perceptions. This information is then communicated in many cases to provide additional information to other objects and context-aware services.

However, not all objects need to become smart to communicate with other objects. Moreover, smart objects, which have processors and network interfaces, cannot always communicate with other objects, for reasons of excess power consumption. Therefore, everyday objects and smart objects should delegate communications between other objects to additional computers whose capabilities are more widespread and efficient. They also tend to communicate with other objects, which are in the same room or close to them, rather than those in different rooms or on different floors. Therefore, they need to detect their communication partners according to where they and their partners are located.

This paper addresses a location-aware communication approach to ubiquitous computing environments. The key idea behind the approach is to provide physical objects, including smart and everyday objects, with counterpart objects, which are constructed as executable software components. The counterpart objects for smart objects communicate with one another on behalf of direct communication between their targets through wired or wireless communication, as much as possible. These counterpart objects were organized in a general location model we presented in a previous paper [13], where the model represented geographical containment relationships between physical entities and places, e.g., a user-room-floor-building hierarchy, as a tree structure of their counterpart entities. That approach provided a mechanism enabling communication between counterpart objects. This paper addresses location-based and personalized services in indoors, e.g., in buildings and houses, rather than in outdoor settings.

The remainder of this paper presents an approach to building and managing location-based and personalized information services in pervasive computing environments (Section 2), and presents the design of our framework (Section 3) and its implementation (Section 4). We describe some experience we have had with several applications, which we used the framework to develop (Section 5). We briefly review related work (Section 5), and concluse with a summary (Section 6).

2 Approach

Many researchers have explored location models for the physical world and existing ones can be classified into two: physical-location and symbolic-location [2, 6]. The former represents the position of people and objects as geometric information. A few outdoor-applications like moving-map navigation can easily be constructed on such physical-location models. Most emerging applications, on the other hand, require a more symbolic notion, i.e., place. This is generically the human-readable labeling of positions. A more rigorous definition is an evolving set of both communal and personal labels for potentially overlapping geometric volumes, e.g., the names of rooms and buildings. An object contained in a volume is reported to be in that place.

The approach presented in this paper addresses symbolic location as a programming model that directly maps to event-driven application programming. For example, when people enter a place, services should be provided from their portable terminal or stationary terminals should provide personalized services to assist them. Our model also introduces the containment relationship between spaces. This is because physical spaces are often organized in a containment relationship, where each space is often composed of more than one sub-space. For example, each floor is contained within at most one building and each room is contained within at most one floor. Therefore, our location model is constructed as a tree, based on geographical containment.

There have been a variety of location-sensing systems. They can be classified into two types: tracking and positioning. The former, including RFID tags, measures the location of other objects. The latter, including GPS, measures its own location. Since it is almost impossible to support all kinds of sensors, the model aims at supporting various kinds of tracking sensors, e.g., RFID-, infrared-, or ultrasonic tags as well as computer vision, as much as possible. The model can have a mechanism for managing location-sensors outside itself so that it is designed independently of these. It transforms geometric information about the positions of objects into corresponding containment relations, but it allows application-specific services to explicitly know where the geometric information locations measured by the sensors are.

2.1 Design principles

Our model has the following features that existing models do not.

Virtual counterparts: No physical objects or spaces may specify their attributes or interact with one another, because of limited resources. The model introduces the notion of counterparts. These are digital representations of physical entities or spaces. An application does not directly interact with physical objects or places, but with their virtual counterparts. The model spatially binds the positions of entities and spaces with the locations of their virtual counterparts and, when they move in the physical world, it deploys their counterparts at proper locations within it.

Inter-counterpart communications: Smart objects and spaces may not have sufficiently powerful computing and communicating capabilities to exchange contextual information with other objects. Counterpart objects communicate with other counterpart objects on behalf of their target objects or places. Instead, when these target objects or places communicate through X-10 and infrared, their counterparts can communicate with them through their protocols.

Location-based service discovery: People often want to communicate with those who are in front of them rather than those in other rooms. Services running on a device should be valid within the bounding region surrounding the device, limiting their presence in space. The model uses location as its primary attribute for discovering and selecting services. Inter-counterpart communications select potential communication partners according to where these are located. An entity, including a person, physical object, or a computing device can specify the kind of surrounding in which it is willing to interact with its communication partners, e.g., visual, audio, or manual-manipulation, which will enable it to interact with devices.

Capability-awareness: Location-based and personalized services must be executed at computing devices whose capabilities can satisfy their requirements and that are at locations where the services should be provided. The model can maintain the locations and capabilities of computing devices as well as those of physical entities and services. It also manages the deployment of application-specific services according to changes in the locations of physical entities, spaces, and computing devices. That is, the model does not distinguish between physical entities, spaces, computing devices, including the computers that maintains them, or application-specific services.

2.2 Location-aware communications

Location awareness is important to enable computing devices and services to communicate. People often want to communicate with others in front of them in the same room rather than with those in other rooms. Such near-field communications are required not only between humans but also between human-machine interfaces and between machines in ubiquitous computing environments (Figure 1). Context-aware services for a user should be provided at computers within the bounding region surrounding his or her presence in space. People should only be able to access location-bound services, e.g., the printers and lights provided in a space, when they enter the space by carrying their terminals or using public terminals located within it. Some devices are controlled according to the requirements of their surrounding environments. For example, an electronic fan in a room should turn on when the current temperature is hotter than the recommended temperature specified for the room.

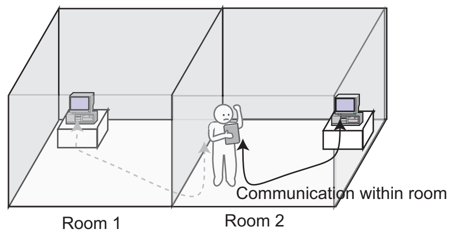

Figure 1. Component migration between computers

The approach presented in this paper was inspired by our previous work, called SpatialAgents [11], which is an infrastructure that enables services to be dynamically deployed at computing devices according to the positions of people, objects, and places that are attached to RFID tags. The previous framework lacked any general-purpose world model and specified the positions for physical entities according to just the coverage areas of the RFID readers so that it could not represent any containment relationship between physical spaces, e.g., rooms and buildings. Moreover, we presented another location model, called *M-Space* [13]. The previous model was aimed at integrating between software-based services running on computing devices and service-provider computing devices, whereas the model presented in this paper is aimed at modeling the containment relationship between physical and logical entities, including the computing devices and software for defining services. This paper also presents a hierarchical structure for components and intercomponent interactions.

3 M-Space: Location Model for Smart Spaces

This section outline the *M-Space* model.

3.1 Containment relationship model

This model is unique to other existing location models, because it not only consists of data elements but also programmable elements, called components, as virtual counterpart objects of physical entities or places. The model represents facts about entities or places in terms of the semantic or spatial containment relationships between components that are associated with these entities or places.

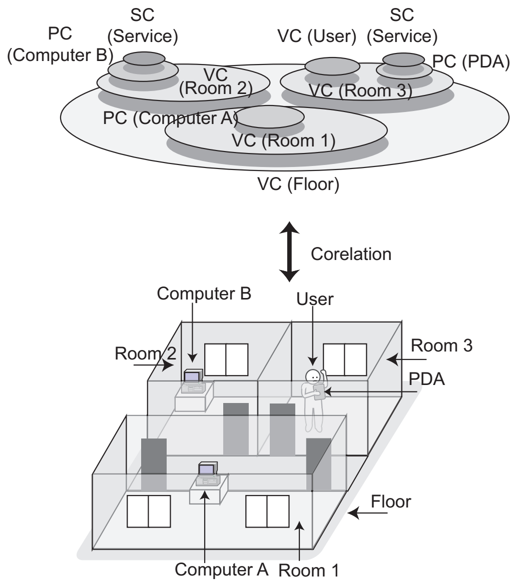

Figure 2. Rooms on floor in physical world and counterpart components in location model.

- **Virtual counterpart:** Each component is a virtual counterpart of a physical entity or place, including the coverage area of the sensor, computing device, or service-provider software.
- **Hierarchical structure:** Each component can be contained within at most one component according to containment relationships in the physical world and cyberspace. It can move between components as a whole with all its inner components.

Components are organized within an acyclic-tree structure, like Unix's file-directory. When a component contains other components, we call the former a *parent* and the latter *children*. When physical entities, spaces, and computing devices move from location to location in the physical world, the model detects their movements through location-sensing systems and changes the containment relationships between components corresponding to moving entities, their sources, and destinations. Figure 2 shows the correlation between spaces and entities in the physical world and their counterpart components. Each component is a virtual counterpart of its target in the world model and maintains the target's attributes. Each component can explicitly have a substitute or representation of itself within its descendants, like Unix's symbolic link. The substitute is still a component but has no attributes. When it receives components or control messages, it automatically forwards the visiting components or control messages to its original component.

The model also offers at least two basic events, entering and leaving, which enable application-specific services to react to actions in the physical world. Readers may think that this hierarchical model is similar to the notion of hierarchical mobile agents presented in our previous paper [10]. However, that enabled large-scale mobile applications to be composed from multiple mobile agents, whereas the present model is aimed at modeling the physical world. When one or more spaces, e.g., the coverage areas of sensors, geographically collapse or overlap, our model simply treats these spaces as coexistent components. As a result, two coexistent spaces may contain an entity. The model allows one of the two VCs corresponding to the spaces to have a component bound to the entity and the other to have a substitute for the component.

3.2 Component

The model is unique to existing location models because it not only maintains the location of physical entities, such as people and objects, but also the locations of computing devices and services in a unified manner. There are VCs illustrated in Figure 2.

- **The virtual counterpart component (VC)** is a digital representation of a physical entity, such as a person or object, except for the computing device itself, or physical place, such as a building or room,
- **The proxy component (PC)** bridges the model and computing device, and maintains a subtree of the model or executes services located in a VC.
- **The service component (SC)** is software that defines application-specific services dependent on physical entities or places.

For example, a car carries two people and moves from location to location with its occupants. The car is mapped into a VC on the model and this contains two VCs that correspond to the two people. The movement of the car is mapped into the VC migration corresponding to the car, from the VC corresponding to the source to the VC corresponding to the destination. Also, when a person has a computer for executing services, his or her VC has a PC, which represents the computer and runs SCs to define the services.

3.2.1 Virtual counterpart component

A person, physical object, or place can have more than one VC, and each VC can contain other VCs and PCs according to spatial containment relationships in the physical world. However, unlike other existing location models, ours does not distinguish between entities and places in the physical world; some entities can be viewed as spaces, e.g., cars and desks, in the sense that they can contain other entities inside them. This permits places to be mobile. For example, a car carries two people and moves from location to location with its occupants. The car in the model has a VC that contains two VCs corresponding to the two people. The movement of the car is mapped from the migration of the VC corresponding to the car, to the VC corresponding to the destination location.

3.2.2 Proxy component

VCs can have software to define the context-dependent services inside them. However, they may not have the ability to execute in the software, because none of the computing devices that maintain these have unlimited computational resources. Instead, there are two facilities by which services can be provided. The first is to forward such services to computing devices embedded in or visiting a space and execute them on the devices. The second is to directly use services provided by computing devices within a space. We introduced proxy components to maintain the location of computing devices and used the devices as service providers. Our model also classifies PCs into two sub-types that handle computing devices according to their functions.

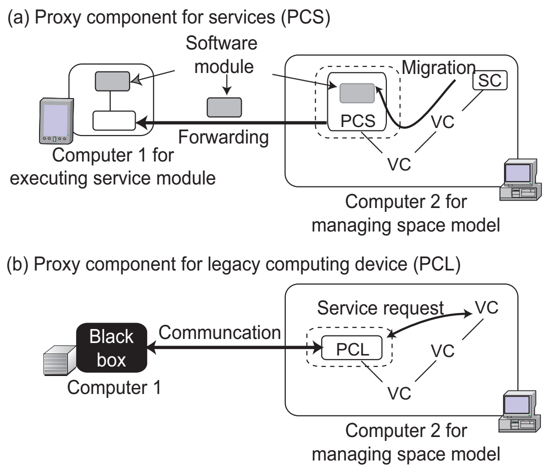
(a) Proxy component for services (PCS)
Software module
Migration
SC
PCS
VC
Forwarding
VC
Computer 1 for executing service module
Computer 2 for managing space model
(b) Proxy component for legacy computing device (PCL)
Service request
VC
Black box
Communcation
PCL
VC
VC
Computer 1
Computer 2 for managing space model

Figure 3. Two types of proxy components

- The first component, i.e., PCS (PC for Service provider), is a proxy of a computing device that can execute services (Figure 3(a)). If such a device is in a place, its proxy is contained in the VC corresponding to the space. When a PCS receives software for defining services, it forwards this to the device to which it refers. After the PCS forwards the software, it enables other components to fetch the software as if this were in it.

- The second component, called PCL (PC for Legacy device), is a proxy of a computing device that cannot execute SCs (Figure 3(b)). If such a device is in a space, its proxy is contained in the VC corresponding to the space and it communicates with the device through the device's favorite protocols.

These components are unique to other existing location models and are useful in maintaining and using computing devices.

3.2.3 Service component

We should reuse existing location-based and personalized services as much as possible. The model introduces several typical software components, e.g., Java Beans and Java Applets as service provider programs. However, such existing components may not be suitable for our model. Each SC is a wrapper for software modules to define application specific services and each specifies the attributes of its services, e.g., the requirements that a device must satisfy to execute these services. The model maintains the locations of services by using SCs.

4 Location-aware Communication

This model provides location-aware communications between components corresponding to physical entities and places, computing devices, and services. It introduces each component as a service provider for its parent, descendants, or neighboring components.

The approach presented in this paper uses the spatial co-location of physical entities as primary attributes for selecting communication partners. The model supports two types of location-aware communications according to the locations of communication partners.[1]

- **Vertical communication** supports interactions between parent counterparts and their child counterparts in the model. Therefore, the former's target entities or spaces are spatially contained in the latter's target spaces in the physical world. The former can continue its communications with the latter, as long as it moves to another space.

- **Horizontal communication** supports interactions between counterparts contained by the same counterparts. The former's and the latter's targets are contained in the same space. That is, communication partners are relocated in the same physical space. Partners are selected according to their spatial co-location rather than their identity.

[1]The model supports communications whose partners may be on different subtrees, but such communications need authentication.

The current implementation supports three primitives, i.e., event passing, method invocation, and stream communication, according to the spatial relations between communication partners. Since the model can be maintained on different computers, it provides programs for counterpart objects with syntactic and (partial) semantic transparency for remote interactions by using proxy elements that have the same interfaces as the remote counterparts themselves.

Components can be dynamically deployed at computing devices according to changes in the locations of physical entities, spaces, and other components. Figure 4 shows the basic structure of the runtime system for the model and component migration between two runtime systems.

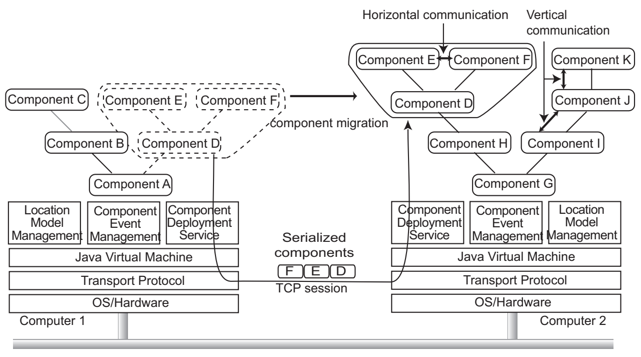

Figure 4. Component migration between computers

Our model is maintained as a tree structure, where each node contains a component and its attributes in a proxy component. It can also be maintained not only by centralized database servers but also by more than one computing device. The model introduces a proxy component, called PCM (Proxy Component for Model), for a subtree maintained on a different computer. Each PCM is a proxy for a subtree that its target computing device maintains and is located in the subtree that another computing device maintains. As a result, it can attach the former subtree to the latter. When a PCM receives other components and control messages, it automatically forwards the visiting components or messages to the device to which they refers (and vice versa) by using a component migration mechanism, like PCSs. Therefore, even when the model consists of subtrees that multiple computing devices maintain, it can be treated as a single tree. Note that a computing device can maintain more than one subtree. Since the model does not distinguish between computing devices that maintain subtrees and computing devices that can execute services, the former can be the latter. Although the model can be automatically configured according to results measured by its underlying location-sensing systems, it provides us with graphical user interfaces developed as active documents, called MobiDoc [14].

5 Implementation

To evaluate the model described in this section, we implemented a prototype system that was built on it. The model itself is independent of programming languages but the current implementation uses Java (J2SE or later versions) as an implementation language for components. Each component was implemented as a collection of Java objects.

Virtual counterpart component

Each VC in the current implementation is defined as a subclass of abstract class `VirtualComponent`, which has some built-in methods that are used to control its mobility and lifecycle. It is bound to at least one entity or space in the physical world and is located at the VC that spatially contains the entity or place.

```
class VirtualComponent extends Component {
  void setIdentity(String name) { ... }
  void setAttribute(String attribute, String value){ ... }
  String getAttribute(String attribute) {..}
  ComponentInfo getParentComponent() { ... }
  ComponentInfo[] getChildren() { ... }
  ServiceInfo[] getAncestorServices(Attribute attr) { ... }
  ServiceInfo[] getNeighboringServices(Attribute attr) { ... }
  Object execService(ServiceInfo si, Message m)
    throws NoSuchServiceException { ... }
  ....
}
```

By invoking `setIdentity`, a VC can assign the symbolic name of the physical entity or space that it represents. For example, a VC refers to the coverage area of an RFID reader and it has the identity of the reader. By invoking `setAttribute`, a VC can record attributes about its entity or space inside it. e.g., owner and size. Each VC can provide its inner components as a service provider with services defined inside it . Furthermore, it allows them to access the service methods provided by the SCs contained within it.

Proxy component

PCs are key elements in the model. According to the types of computing devices, PCs can be classified into two classes, i.e., PCS and PCL. Note that a computing device can have different PCs.

- Proxy component for service provider (PCS) is a representation of a computing device that can execute SCs. It automatically forwards its visiting SCs to its target device by using the component migration mechanism. Each SC can have one or more activities that are implemented by using the Java thread library. PCSs can control all SCs inside them under the protection of Java's security manager. Furthermore, PCSs maintain the life-cycle of SCs: i.e., initialization, execution, suspension, and termination. When the life-cycle state of an SC is changed, the runtime system issues certain events to the SC and the SC's descendent components (and the SC's parent component).

- Proxy component for legacy device (PCL) supports a legacy computing device that cannot execute SCs due to limited computational resources. It is located at a VC corresponding to the space that contains its target device. It establishes communication with its target device through its favorite approach, e.g., serial communications and infrared signals. For example, a television, which does not have any computing capabilities, can have an SC in the VC corresponding to the physical space that it is contained in and can be controlled in, and the SC can send infrared signals to it.

Service component

Many computing devices in ubiquitous computing environments only have a small amount of memory and slower processors. They cannot always support all services. Here, we introduce an approach to dynamically installing the upgraded software that is immediately required in computing devices that may be running. SCs are mobile software that can travel from computing device to computing device, which is achieved by using mobile agent technology. The current implementation assumes SCs to be Java programs. They can be dynamically deployed at computing devices. Each SC consists of service methods and is defined as a subclass of the abstract class `ServiceComponent`. Most serializable JavaBeans can be used as SCs.

```
class ServiceComponent extends Component {
  void setName(String name)
  Host getCurrentHost() { ... }
  void follow(ComponentID id) throws
    NoComponentException { ... }
  void setComponentProfile(
    ComponentProfile cpf) { ... }
  Hosts[] getNeighboringHosts() { ... }
  Host[] getCandidateHosts(Host[] hosts) {..}
  ....
}
```

When an SC migrates to another computer, not only the program code but also its state are transferred to the destination. For example, if an SC is included in a VC corresponding to a user, when he or she moves to another location, it is migrated with the VC to a VC corresponding to the location. The model allows each SC to specify the minimal (and preferable) capabilities of PCSs that it may visit, e.g., vendor and model class of the device (i.e, PC, PDA, or phone), its screen size, number of colors, CPU, memory, input devices, and secondary storage, in CC/PP (composite capability/preference profiles) form [16]. All SCs can register such capabilities by invoking `setComponentProfile()`.

Vertical communication

All components can send events to invoke callback methods provided their children can subscribe to the events that they are interested in so that they can receive these events. Each component can be viewed as a service provider for its ancestral and child components. That is, it can provide them with the service methods explicitly defined inside it. When a component invokes the `getAncestorServices()` or `getChildServices()` of the `VirtualComponent` class with the attributes that it requires, the runtime system searches for suitable services along the route of the component's tree structure from its parent or in its children. If ancestral or parent components (or child components) have service methods that match the attributes, the runtime system returns a list of suitable service methods to the component. The component can then access one of the methods by invoking `execService()` with an instance of the `Message` class, which can specify the kind of message, arbitrary objects as arguments, and deadlines for timeout exceptions.

Horizontal communication

Each component can invoke service methods provided by its neighboring components, which are contained in its parent. When a component invokes `getNeighboringServices()` with the attribute that specifies its requirements, the runtime system searches for suitable services in its neighboring components. The component can access one of the methods by invoking `execService()` as in vertical communication. The attributes can be written in XML. The model provides a keyword-based search for each of the attributes as a lightweight mechanism for discovering services because it needs to be available for less powerful computing devices.

Intercomponent communication mechanism

Each component hierarchy is maintained as a tree structure where each node contains a component and its attributes in a PCM. Each PCM attaches a subtree maintained by its target computing device to a tree maintained by another computing device. It forwards its visiting components or controls messages to its target device from the device that it is located at, and vice versa, by using the component migration mechanism. For example, when it receives SCs and VCs, it transmits to its target device to deploy them at appropriate nodes of the subtree maintained by the device. The containment relationship between components in this model can be explicitly configured by users deploying PCMs in another PCM. The framework supports three types of intercomponent communications.

- Remote method invocation supports vertical and horizontal communications. It offers APIs for invoking the methods of other components on local or different computers with copies of arguments. Our programming interface for method invocation is similar to CORBA's dynamic invocation interface and does not have to statically define any stub or skeleton interfaces through a precompiler approach, because ubiquitous computing environments are dynamic.

- Publish/subscribe-based event passing supports vertical and horizontal communications. Publish/subscribe approaches are useful and efficient for capturing changes in the physical world, because they provide subscribers with the ability to express their interest in an event so that they can be notified afterward of any event fired by a publisher. The approach is useful in minimizing the number of events passed to remote computers. This model provides a generic remote publish/subscribe approach using Java's dynamic proxy mechanism, which is a new feature of the Java 2 Platform since version 1.3.[2]

- Stream communication supports horizontal communications. The notion of a stream is highly abstracted representing a connection to a communication channel. When partners are different computers, the model enables two components on different hosts to establish a reliable channel through a TCP connection managed by the hosts.[3]

Component migration mechanism

Component migration in a component hierarchy is merely done as a transformation of the tree structure of the hierarchy (Figure 5). When one component is moved to another, a subtree, whose root corresponds to the component and branches correspond to its descendent component is moved to a subtree representing the destination. The current system basically uses the Java object serialization package to marshal the components. The package does not support the stack frames of threads being captured. Instead, when a component is serialized, the system propagates certain events within its embedded components to instruct the

[2]As the dynamic creation mechanism is beyond our present scope, we have left it for a future paper

[3]Since our channel relies on TCP, it can guarantee *exactly-once* communication semantics across the migration of components.

agent to stop its active threads. When a component is transferred over a network, the runtime system stores the serialized state and code of the component, including the components embedded within it, into a bit-stream formed in Java's JAR file format, which can support digital signatures for authentication. The system has a built-in mechanism for transmitting the bit-stream through TCP sessions.

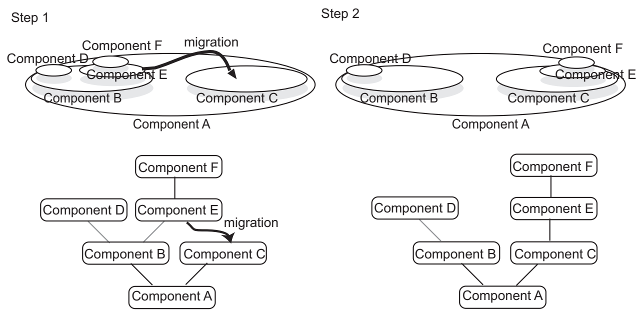

Figure 5. Component containment and migration

The model enables each component to confine its movement within an ancestor component for reasons of security and to protect privacy. When a component carries another component beyond the latter's range, the latter remains within the range or terminates.

Location-sensor management

The model offers a mechanism for automatic configuration to deploy components by using location-sensing systems. To bridge PCMs and location-sensors, the model introduces location-management systems, called LCMs, outside the PCMs. All LCMs manage location sensors and maintain a database where they store bindings between the references of physical entities in sensors, e.g., the identifiers of RFID tags attached to the entities and the identifiers of VCs corresponding to the entities. Each LCM is responsible for discovering VCs bound to entities or PCs bound to computing devices within the coverage areas of the sensors that it manages. When an entity (or device) attached to an RFID-tag and an LCM detect the presence of the entity (or device) within the coverage area of an RFID reader managed by the LCM, the LCM searches its database for VCs (or PCs) bound to the entity (or device) and informs computing devices that maintain the VCs (or PCs) about the VC corresponding to the reader. The VCs (or PCs) migrate to the reader's VC. If the LCM's database does not have any information about the the entity (or device), it multicasts query messages to other LCMs. If other LCMs have any information about the entity, the LCM creates a default VC as a new entity. When a tag is attached to an unknown device that can maintain a subtree or execute SCs, the LCM instructs the VC that contains the device to create a default PCM or PCS for the device. The current implementation enables each component to have a deployment policy for the location and migration of its target.[4]

Current status

A prototype implementation of this model was built with Sun's J2SE version 1.4 or later versions. It uses the Mobilespaces mobile agent system to provide mobile components and supports three commercial locating systems: Elpas's infrared tag sensing system, RF Code's Spider (active RF-tag system), and Alien Technology's UHF-RFID tag (passive RF-tag system). Although the current implementation was not built for performance, we measured the cost of migrating a 4-Kbyte component (zip-compressed) from the source to the destination recommended by an LSM over a network. The latency of component migration to the destination after the LSM had detected the presence of the component's tag was 390 ms and the cost of component migration between two hosts over a TCP connection was 41 ms. This experiment was done with two computing devices that maintained the component tree, and source and destination computing devices, each of which was running on one of six computers (Pentium-M 1.6 GHz with Windows XP and J2SE ver. 5) connected through a Fast Ethernet network. We believe that this latency is acceptable for location-aware systems used in rooms or buildings.

6 Early Experience

This section describes how the model implements typical applications and what advantages it has.

6.1 Follow-me applications

Since *follow-me* services are a typical application in ubiquitous computing environments, we developed a follow-me application system by using the approach presented in this paper. For example, Cambridge University's Sentient Computing project [5] enabled applications to provide a location-aware platform using infrared-based or ultrasonic-based locating systems in a building. The platform can track a user's movements while he or she is moving around so that the graphical user interfaces of the user's applications follow him or her. The model presented in this paper, on the other hand, enables moving users to be naturally represented independently of location-sensing systems. Unlike previous studies on applications, it can also migrate such

[4]Such policies are described as Java programs, but we plan to develop a policy specification langauge.

applications themselves to computers near moving users. That is, the model provides each user with more than one VC and can migrate this VC to a VC corresponding to the destination. We developed a mobile window manager as a VC that could carry its desktop applications implemented as SCs to another computer. When the VC bound to the user migrates to the destination's VC, the former detects appropriate service providers, which are implemented as PCs or SCs within the latter VC by using horizontal communication mechanisms. Moreover, the former VC can detect and coordinate PCs or SCs in the descendants of the destination's VC by using the vertical communication mechanism. For example, when a VC bound to a user arrives at a VC bound to another room, it tries to communicate with PCLs to control electric lights in the room to turn them on. Using the model presented in this paper, the window manager could be easily and naturally implemented as a VC bound to the user and desktop applications as SCs. They could automatically be moved to a VC corresponding to the computer that was in the current location of the user by an LCM and they could then continue processing at the computer, as outlined in Figure 6.

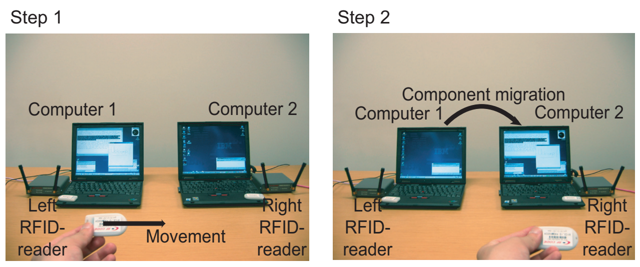

Figure 6. Follow-me desktop applications between two computers.

6.2 Personal server

The second application was inspired by the personal server proposed by Want [15]. We could easily implement interactions between personal servers and stationary computers, i.e., bwall-mounted smart displays and public terminals by using this model. A user carries a handheld file-sharing server that has no integral user interface, but has a processor, secondary storage, a wireless LAN network interface, and is tied to an active RFID-tag. When he/her comes close to a stationary computer, his/her personalized services are dynamically deployed at smart TVs. When he/she carries the server to a room with a TV, the model deploys the PCS bound to the server at the VC corresponding to the room. The TV has a PCL as its proxy. Although the PCL can forward SCs to the server, it can still communicate with other neighboring components, including the PCL, so that he/she can view images stored in the server on the TV screen by using the horizontal communication mechanism shown in Figure 7. We developed a mechanism enabling communication between personal servers and embedded computers [8]. The PCL can gather image data from the server and then display these through the mechanism.

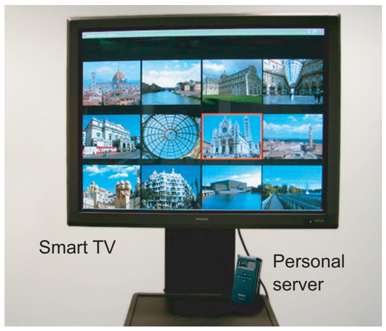

Figure 7. Smart TV executes PCL to access image data from personal server to display the GUI on its screen.

7 Related Work

Many researchers have explored location models for ubiquitous computing environments. Most existing models have aimed at identifying and locating entities, e.g., people and physical objects and computing devices in the physical world. These existing models, including recent models proposed a few years ago, can be classified into two types: physical-location and symbolic-location models.[5] The former represents the position of people and objects as geometric information, e.g., NEXUS [4, 1] and Cooltown [7]. A few applications like moving-map navigation can easily be constructed on a physical-location model with GPS systems. However, most emerging applications require a more symbolic notion, i.e., place. This place is generically the human-readable labeling of positions. The latter represents the position of entities as labels for potentially overlapping geometric volumes, e.g., the names of rooms and buildings, e.g., Sentient Computing [5] and EasyLiving [3]. Existing approaches assume that their models will be maintained in centralized database servers, which may not always be used in ubiquitous computing environments. Therefore, our model should be managed in a decentralized manner and be dynamically organized in an ad-hoc and peer-to-peer fashion. Virtual Counterpart [9] supports RFID-based tracking

[5]Extensive works has been done for location-aware services but most of these have been constructed based on either of the two models. Therefore, we discuss typical examples of the two models.

systems and provides objects attached to RFID-tags with Jini-based services. Since it enables objects attached to RFID-tags to have their counterparts, it is similar to our model. However, it only supports physical entities except for computing devices and places. Our model does not distinguish between physical entities, places, or software-based services so that it can provide a unified view of ubiquitous computing environments, where not only physical entities are mobile but also computing devices and spaces.

Several projects have proposed and developed communications between devices in ubiquitous computing environments, e.g., UPnP, SDS, and Jini. However, most existing work has enabled devices to discover other devices and services within local networks without any notion of location. The notion of location-aware communications in ubiquitous computing environments has attracted scant attention thus far. This is a serious obstacle to the growth of ubiquitous computing, because devices are often required to coordinate with nearby devices or services running on nearby computers rather than with remote devices, even those that are connected to current local networks. Of these, the RAUM system [2] supports location-aware communications based on a containment location model in ubiquitous computing environments like our approach does. The system aims at location-aware discovery and routing between network-enabled devices, whereas the approach itself supports location-dependent communications between the virtual counterparts of physical objects in addition to network-enabled devices.

8 Conclusion

We presented an architecture for location-aware services in ubiquitous computing environments. It consists of two parts: a symbolic location model and a location-aware communication mechanism. The former can be dynamically organized like a tree based on geographical containment, such as that in a user-room-floor-building hierarchy and each node in the tree can be constructed as counterpart objects that are implemented as the executable software and proxies of their physical target objects and places. The latter enables counterpart objects to communicate with other counterpart objects according to the geographical relationships between their physical target objects and places. We designed and implemented a prototype system based on the model and demonstrated its effectiveness in several practical applications.

References

[1] M. Bauer, C. Becker, and K. Rothermel, Location Mmodels from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks, Personal and Ubiquitous Computing, vol. 6, Issue 5-6, pp. 322-328, Springer, 2002.

[2] M. Beigl, T. Zimmer, C. Decker, A Location Model for Communicating and Processing of Context, Personal and Ubiquitous Computing, vol. 6 Issue 5-6, pp. 341-357, Springer, 2002

[3] 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.

[4] 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).

[5] A. 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] 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.

[7] T. Kindberg, et al, People, Places, Things: Web Presence for the Real World, Technical Report HPL-2000-16, Internet and Mobile Systems Laboratory, HP Laboratories, 2000.

[8] T. Nakajima and I. Satoh, Personal Home Server: Enabling Personalized and Seamless Ubiquitous Computing Environments, Proceedings of 2nd International Conference on Pervasive Computing and Communications (PerCom'2004), pp.341-345, IEEE Computer Society, March 2004.

[9] K. Romer, T. Schoch, F. Mattern, and T. Dubendorfer, Smart Identification Frameworks for Ubiquitous Computing Applications, IEEE International Conference on Pervasive Computing and Communications (PerCom'03), pp.253-262, IEEE Computer Society, March 2003.

[10] I. Satoh, MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2000), pp.161-168, April 2000.

[11] I. Satoh, Linking Phyical Worlds to Logical Worlds with Mobile Agents, Proceedings of International Conference on Mobile Data Management (MDM'2004), IEEE Computer Society, January 2004.

[12] I. Satoh, Software Testing for Wireless Mobile Computing, IEEE Wireless Communications, vol. 11, no. 5, pp.58-64, IEEE Communication Society, October 2004.

[13] 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.

[14] I. Satoh, A Document-centric Component Framework for Document Distributions, A Location Model for Pervasive Computing Environments, Proceedings of 8th International Symposium on Distributed Objects and Applications (DOA'2006), Lecture Notes in Computer Science (LNCS), vol.?, Springer, October (2006).

[15] 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.

[16] World Wide Web Consortium (W3C), Composite Capability/Preference Profiles (CC/PP), http://www.w3.org/ TR/NOTE-CCPP, 1999.