Inf Syst Front (2009) 11:501–512
DOI 10.1007/s10796-008-9120-5

Location-aware communications in smart environments

Ichiro Satoh

Published online: 27 June 2008

Abstract We present a location-aware communication approach for smart home environments that is based on a symbolic location model that represents the containment relationships between physical objects, computing devices, and places. The approach enables people and devices to discover and connect with communication partners based on their co-locations. It also provides non-smart objects, including home appliances, everyday objects, and places, with virtual counterparts, which both represent them and work as proxies for their targets. We present the design for this approach and describe its implementation and applications.

Keywords Location-awareness · Symbolic location model · Smart space

1 Introduction

Communication between smart spaces, including smart home environments, is different from telecommunications. The goal of the latter is to enable people or computing devices to communicate with one another regardless of location distance of devices or people who are communicating. Communication partners in telecommunication reveal their identities, e.g., their IP addresses and MAC addresses. Communication between smart spaces, on the other hand, supports heterogeneous computing devices whose computational resources may be limited and tends to be within the same space, i.e., the same room and nearby fields, rather than between different rooms or different floors. It should also be possible to select communication partners in smart spaces based on their co-locations and capabilities.

Unlike smart offices, smart homes have both smart appliances and legacy appliances. This is because most people cannot afford to replace their legacy appliances with smart ones. Legacy appliances do not have computing capability. They may have functions that can be monitored or controlled from their own controllers or control panels through infrared or non-standard radio signals, but they cannot communicate with other appliances. Therefore, smart home environments that can enable legacy appliances to communicate and can link them with smart appliances must be developed. Furthermore, not all smart appliances or objects can communicate with other smart objects because of either their limited computational resources or because such communication consumes too much power. Therefore, legacy appliances and smart objects whose capabilities are limited should delegate their capabilities for computing and communications with other objects to other smart objects, including home servers, that have good computing capability.

To support context-aware services in an intelligent environment, we constructed a location model that maintains the capabilities of smart objects and tracks contextual information detected by the underlying sensing systems discussed in our previous papers (Satoh 2005, 2007b). The previous model is the basis for the model presented here but does not support location-aware communication. We propose a framework for enabling smart objects and legacy home appliances to discover other smart objects and communicate with

I. Satoh (✉)
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
e-mail: ichiro@nii.ac.jp

them in intelligent environments according to their spatial co-locations and capabilities. We presented an early version of this framework in previous papers (Satoh 2007a, c). However, the previous version cannot incorporate non-smart objects, including legacy appliances, into a smart home environment. The framework presented here can support both non-smart and smart objects. It enables non-smart objects to delegate control over their services and communications with other objects to smart objects or home servers.

We present an extended framework that incorporates both smart and non-smart objects, including legacy appliances, into smart environments. We also describe several experimental trials of the framework in real places with real users. The framework maintains a symbolic location model inside it and provides smart or non-smart objects with virtual counterparts, which represent them in the model and are constructed as executable software components. That is, non-smart objects have their own proxies in the model and can indrectly communicate with their physical targets via their proxies.

2 Background

We address location-aware human-machine and machine-machine indoor communication, e.g., communication within houses and other buildings rather than in outdoor settings. The framework presented here has the following features.

2.1 Location awareness

Location awareness is important for providing communications in smart spaces, including smart houses. People often want to communicate with or have services provided by computing devices and smart objects, e.g., by PCs and televisions in the same room, and users are only be able to access location-bound services, e.g., the lights and air-conditioners in a certain room. The framework needs to know which services are available in the space the users are in and provide the services. However, some services in one room, such as air-conditioning, are often available to other rooms on the same floor. Some services are provided based on the requirements of surrounding areas. For example, the window and door locks in a room are often controlled by the security policy for the floor or the entire building. Therefore, knowledge of the relationships between spaces is essential in providing services in buildings and houses. To understand these relationships, the framework needs to use a location model that represents the spatial structure of a building instead of treating each space in an ad-hoc manner. However, not all computing devices have the capability to provide services or communicate with people and other devices. As a result, even if a device is in a location where a service is desired, it may not be able to provide the service because it lacks software or capability, such as input or output facilities. The framework needs to maintain the capabilities of computing devices and deploy software to provide services using appropriate devices, but only when the services are required.

2.2 Linking legacy home appliances

The framework also needs to support non-smart objects, including legacy home appliances, which have no computing resources or network interfaces to communicate with other objects and which can be classified into two types. The first type has interfaces to communicate with other specified devices. For example, televisions and air-conditioners can be controlled through infrared-based communications, and power outlets for home appliances can be controlled through a commercial protocol called X10. The framework should be able to control such home appliances. The second type of non-smart objects has no interfaces to communicate with other devices. Everyday objects, e.g., cups, chairs, desks, and windows cannot provide services, but they may be required to annotate themselves. For example, there is information about cups that may be relevant to a user, e.g., the materials they are made of and the date they were manufactured. The framework enables everyday objects to annotate themselves and assist people in using them by tracking their proxies running on home servers, which have significant computing capability.

2.3 Related work

The remainder of this section describes previous work on location-aware communications in indoor settings. Several projects have been proposed and developed to enable communications between devices in ubiquitous computing environments, including smart spaces, using service discovery systems, e.g., UPnP, SDS, and Jini. However, most of this work has enabled devices to discover other devices and services based on their identity instead of their co-location. The notion of location-aware communications in smart spaces has attracted scant attention thus far. There have been few attempts to develop location-aware communications indoors, although many academic or commercial location-aware communications have been explored and used. Most of these indoor systems have been designed to keep track of the locations of devices in an ad-hoc manner. One of these, the RAUM system (Beigl et al. 2002), supports location-aware communications in smart spaces. Although the system is designed for location-aware discovery and routing between network-enabled computing devices, it assumes that physical objects in smart spaces have the capability to interact with one another. Although EasyLiving (Brumitt et al. 2000) was developed to build smart spaces and support computer-vision-based tracking sensors, it was designed to coordinate particular functions provided by PCs with significant computing capability. Neither system can support non-smart objects, although there are still many non-intelligent objects in smart spaces. The Sentient Computing Project (Harter et al. 1999) provides an infrastructure for location-aware services and can track locational information about people and physical objects with ultra-sonic tags attached to them. However, it does not support location-aware communications and assumes that each of its applications explicitly defines the spatial range where they were available in a room or building. Virtual Counterpart (Romer et al. 2004) supports radio frequency identification (RFID)-based tracking systems and provides objects with RFID-tags attached to them with Jini-based services. In that it represents RFID tag-equipped objects with virtual counterparts, it is similar to our model.

3 Approach

A framework that models physical objects, people, and places to identity communication partners according to their co-locations. Many researchers have explored location models of the physical world, and current location models can be classified into two types: physical and symbolic location (Becker 2004). The former type represents the position of people and objects as geometric information, which can be measured by global positioning system and ultra-sonic location sensing systems. A few outdoor applications like moving map navigation can easily be constructed on such physical location models. The former is not suitable in indoor settings, because, although the locations of two objects may be geometrically close to one another, the objects themselves may be in different places. For example, when two objects are in different rooms, on opposite sides of the same wall, outdoor applications may treat the two objects as near each other. In fact, most emerging applications for indoor settings require a more symbolic notion, i.e., place. This is human-readable labeling of positions. A more rigorous mapping is the evolving set of communal and personal labels for potentially overlapping areas, e.g., the names of rooms, floors, and buildings. An object in a such an area is reported to be there.

Our framework addresses symbolic location using a programming model that maps directly to event-driven application programming. For example, when people enter a place, services should be provided from portable or stationary terminals. The framework also introduces a containment relationship between spaces. This is because physical spaces are often organized in containment relationships, where each space is 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. The framework has the following design principles:

Virtual counterpart: No physical objects, including legacy appliances, or places may specify their attributes or interact with one another, because of limited resources. The model introduces the notion of virtual counterparts. These are both digital representations of physical entities or spaces and programmable proxies of the entities or spaces in the model. They can define application level services, e.g., annotation about their targets and interactions between physical entities, on behalf of the targets themselves. Applications do not interact directly with such physical objects or places, only with their virtual counterparts.

Support to legacy devices: Non-smart physical entities, including legacy home appliances and spaces, may not have sufficiently powerful computing capability or may lack communication interfaces, e.g., Ethernet and WiFi, to exchange contextual information with other entities and spaces. Some of them, however, may have their own built-in communication interfaces, i.e., infrared-based devices, so that they can be remotely or indirectly controlled through such communications. For example, most legacy appliances have no network interfaces but their power supplies can be controlled through power line communications systems, e.g., X10-based power outlet control communication (X10 communication 2007). Virtual counterparts communicate with other counterparts via proxies, so that the physical targets do not need to directly communicate with each other. That is, virtual counterpart objects can define protocols to communicate with their targets through their normal mode of communication and communicate with other objects within the model. As a result, an application or user can indirectly control or monitor physical entities or places by monitoring their counterparts.

Containment relationship model: Virtual counterparts are structurally organized based on geographical containment relationships between physical objects and places, e.g., a tree structure-like user-room-floor-building hierarchy. Each counterpart is contained by at most one counterpart, a parent counterpart, and can contain more than zero counterparts, called a child. For example, each floor is contained within at most one building and each room is contained within at most one floor. Each of the counterparts corresponding to these rooms is contained in the counterpart corresponding to the floor. The model spatially binds the positions of objects and places with the locations of their virtual counterparts. When a physical entity moves to another location in the physical world, the model deploys its counterpart at the counterpart of the destinations.

Location-aware communications: The framework treats location-aware communications as interactions between virtual counterparts. Each counterpart can communicate with its parent, its children, and its sibling counterparts. For example, when a television and a person are in the same room, the framework maintains a sub-tree whose root is the counterpart corresponding to the room and whose two nodes are counterparts corresponding to the television and the person. Counterparts can communicate with other counterpart objects according to their structures.

Location-based service discovery: 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 surroundings 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 other devices.

4 Design

This section outlines our location model to support location-aware communication.

The model consists of software elements, called virtual counterparts (VCs), which are both digital representations of entities, e.g., people and objects, or spaces in the physical world and proxies for the computing devices and services themselves. The model represents facts about entities or places in terms of the semantic or spatial containment relationships between the VCs that are associated with them.

- **VC object:** Each physical entity, e.g., a person, legacy appliance, or physical place, e.g., a building or room, has more than one VC. Each VC is constructed as an executable software component.
- **Hierarchical structure:** Each VC can be contained within at most one VC according to the containment relationships in the physical world and cyberspace. It can also be dynamically deployed between VCs as a whole with all its inner VCs when its target object moves.

VCs are organized within an acyclic-tree structure, like Unix's file directory. When a VC contains other VCs, we call the former a *parent* and the latter *children*. When physical entities, places, 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 of the VCs corresponding to the moving entities, their sources, and destinations. Figure 1 shows the correlation between places and entities in the physical world and their VCs. The model also offers at least two basic events, entering and leaving, which enable

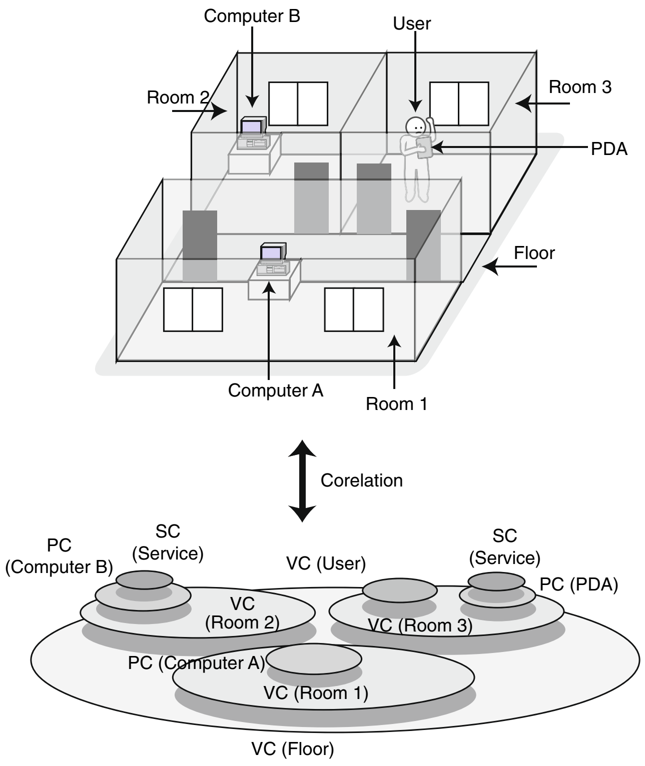

Fig. 1 Rooms on floor in physical world and VCs in location model

application-specific services to react to actions in the physical world in addition to communications between VCs and between VCs and physical objects or places.

The model has substitutes or representations of VCs similar to Unix's symbolic link. Substitutes are still VCs but have no attributes. When they receive other VCs or control messages, they automatically forward the visiting VC and arriving messages to their original VCs. When one or more places, e.g., the coverage areas of sensors, geographically collapse or overlap, our model simply treats them as coexisting VCs. As a result, two coexisting places may contain the same entity. The model allows one of the two VCs corresponding to the places to have a VC bound to the entity and the other to have a substitute for the VC. The containment relationship model itself is based on our previous modelfs M-Spaces (Satoh 2005), but the previous model cannot support location-aware communications and incorporate non-smart objects into a smart environment.

4.1 Support for home appliances

The framework provides non-smart entities, which support their own built-in communication interfaces, with their counterparts, called proxies for legacy devices (PLs), their proxies in the models. If such an object is in a certain place, its proxy is contained in the VC corresponding to that place. When a PL receives request messages, it automatically sends commands corresponding the messages to its target object through its favorite protocols.

4.2 Support for smart objects

The model classifies smart objects into two types. The first can download software that defines services from external systems and executes the software. That is, it is a platform for providing general services. The second only provides its own initial service and has built-in communication interfaces to control itself from the external systems through networks. The first counterpart, i.e., proxy for computing devices (PC), is a proxy for a smart object that can download and execute software. While such a smart object is located in a place, the model uses the PC of the object at the VC corresponding to the place. When a PC receives software to define services, it forwards it to the object to which it refers. The PC can explicitly retrieve or remove the software from its object. The model provides a smart object in the second type with a PL-like non-smart object.

PCs and PLs are unique to other existing location models and are useful in maintaining and using computing devices.

4.3 Location-aware communications

Our model provides location-aware communications between VCs corresponding to physical objects, places, and computing devices. The approach presented here uses the spatial co-location of physical entities as a primary attribute for selecting communication partners. The model supports two types of location-aware communications based on the locations of the communication partners.[1]

- **Vertical communication** supports interactions between parent VCs and their child VCs in the model. Therefore, the parentsf target entities or places are spatially contained in the childrenfs target places in the physical world. The parent can continue to communicate with the child, as long as it moves to another place.
- **Horizontal communication** supports interactions between VCs contained in the same VCs. The parentfs and childfs targets are contained in the same place. That is, communication partners are relocated in the same physical place. Partners are selected based on their spatial co-location rather than their identity.

Our model supports three primitives, i.e., event passing, method invocation, and stream communication, based on the spatial relations between communication partners. Since the model can be maintained on different computers, it provides programs for VCs with syntactic and (partial) semantic transparency for remote interactions by using proxy elements that have the same interfaces as the remote VCs themselves. The underlying location-sensing systems can dynamically relocate VCs based on changes in the locations of their target entities in the physical world.

4.4 Location model management

Our model is maintained as a tree structure where each node can maintain its attributes and contain one or more components as programmable items corresponding to VCs. The model can also be run both by

[1]The model supports communications between partners that may be on different sub-trees, but such communications need authentication.

centralized database servers and by multiple computing devices. The model introduces a proxy object, called proxy for model (PM), for a sub-tree maintained on a different computer. Each PM is a proxy for a sub-tree. As a result, it can attach the former sub-tree to the latter. When the PM receives VCs and control messages, it automatically forwards the visiting objects or messages to the device to which they refer (and vice versa). Therefore, even when the model consists of sub-trees that multiple computing devices maintain, it can be treated as a single tree. Note that a computing device can maintain more than one sub-tree. Since the model does not distinguish between computing devices that maintain sub-trees and computing devices that can execute services, a sub-tree computing device can execute services.

5 Implementation

We built a prototype system to evaluate the model. The model itself is independent of programming languages, but the current implementation uses Java (J2SE version 1.5 or later versions) as an implementation language that uses software components as VCs and proxies as Java objects.

5.1 Location model management system

The model is constructed on a mobile agent-based component system, where each counterpart is defined as a component and a whole or part hierarchy of counterparts is maintained as a structural composition of components. It is also a runtime system for managing and exchanging components and controls messages with runtime systems running on different computers. Figure 2 outlines the basic structure of a management system for the model.

5.1.1 Component hierarchy management

Each component can be defined as a self-contained computing entity without any description of component hierarchy. The management system provides a container object for each component corresponding to a VC. The container technology developed by Enterprise Java Beans provides interfaces for components and enables them to transparently adapt to runtime services, e.g., manages transactions. This framework provides each component with a wrapper, called a *component tree node*, as a container object for the component. Each node contains its target component, its attributes, and containment relationships between its target and its parent components and between its component and its child components. As a result, a hierarchy of components is maintained in the form of a tree structure, which has the component tree nodes of components. Suppose a user carries two physical entities to another room. In Fig. 2, component A is the VC corresponding to the source place, component H is the VC corresponding to the destination, component D is the VC corresponding to the user, and component E and F are the VCs corresponding to the entities carried by the user.

5.1.2 Component runtime management

Since each component is an autonomous computing entity, the management system provides the component with one or more active threads under its control.

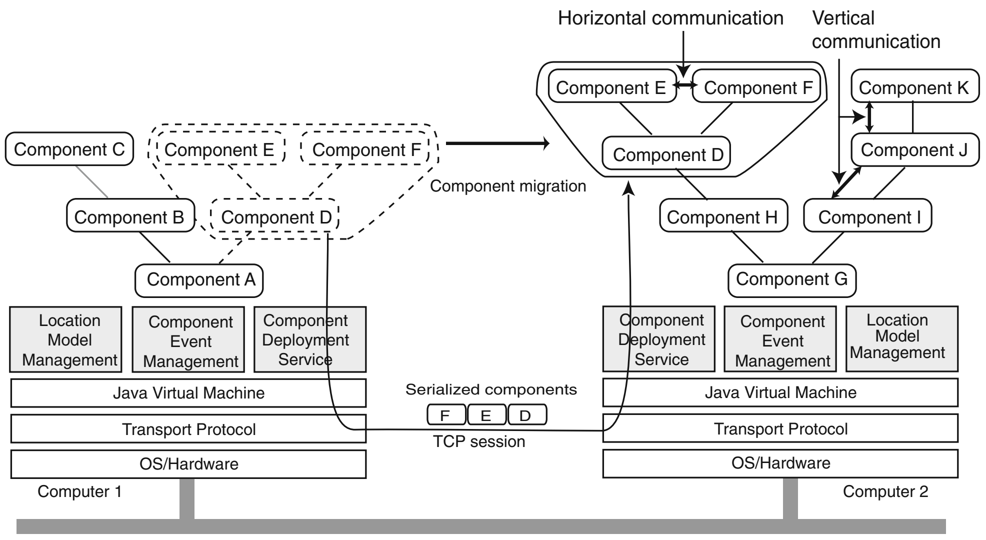

Fig. 2 Migration of VCs

Each component can have one or more activities implemented using the Java thread library. The system can control all the components in its component hierarchy, under the protection of Java's security manager. Furthermore, the system maintains the life cycle of the agents: initialization, execution, suspension, and termination. When the life cycle state of an agent changes, the system issues certain events to the component and its descendent components.

5.1.3 Location-sensing systems

The model itself is independent of any location-sensing system. The management system has interfaces for monitoring its underlying location-sensing systems. The current implementation provides lower-level monitoring systems for four commercial locating systems: Elpas's infrared tag sensing system, RF Code's Spider (active RF-tag), Alien Technology's UHF-RFID tag (passive RF-tag), and Aeroscout's positioning systems. The first three are proximity-based tracking systems that can detect the presence of their tags in the coverage areas of their sensing units. The monitoring systems provide VC objects corresponding to the coverage area of each of their sensing units. Aeroscout can map the position of its tags as geometric information. The monitoring system for the locating system maps the geometric information for the specified areas with symbolic names and returns the names to the management system.

5.1.4 Distributed management

When one component is moved to another, a sub-tree, whose root corresponds to the component and branches and corresponds to its descendent component, is moved to a sub-tree representing the destination. Component migration within a component hierarchy occurs merely as a transformation of the tree structure of the hierarchy. Components in a sub-tree, on the other hand, can be deployed in another sub-tree maintained on another computer through PMs. When a component is transferred over a network, the management system marshals the state of the component, e.g., instance variables, and the components embedded within the component, into a bit-stream and transmits the serialized state using the code of the component, including the components embedded within it, to the destination through TCP sessions. The model enables each component to confine its movement within its ancestor components to ensure security and to protect privacy. When a component carries another component beyond the latter's range, the latter remains in the range or terminates.

5.2 VC component

Each component is defined as a set of Java objects and is classified as one of the following two types.

5.2.1 Component for VC object

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

```
class VirtualComponent extends Counterpart
  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 is identified as belonging to the reader. By invoking `setAttribute`, a VC can record attributes about its entity or the space within it. e.g., owner and size. Each VC can provide its inner components as a service provider with services defined inside it.

5.3 Proxy component

Proxies are classified into two classes, PC and PL, based on the type of computing device. A single computing device can have different proxies.

- The proxy for a computing device (PC) is a representation of a computing device that can download and run software. It is located at a VC corresponding to a space that contains its target device. It automatically forwards its visiting software to its

target device by using the component-migration mechanism.
- The proxy for a legacy device (PL) supports a legacy computing device that cannot download or execute software. It is located in a VC corresponding to a 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 has no computing capabilities, has a PL that can send infrared signals to it.

PC and PL are defined as subclasses of `Virtual Component`.

5.4 Inter-component communication

Each management system controls communications between components as location-aware communication.

5.4.1 Structural events

The system can issue certain events to components when changes in its component tree. When a component, including its inner components, migrates from the source component to the destination component, the visiting component and its inner components receive `leaving` events and the source component receives a `removing` event. After the component arrives at the destination, it and its inner components receive `arrival` events and the destination component receives `arrive` event. These events can include geometric information about the position of moving entities measured by the underlying location-sensing systems.

5.4.2 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 ancestor and child components. That is, it can provide them with the service methods explicitly defined in it. When a component invokes `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 ancestor 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.

5.4.3 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()` 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.

5.4.4 Inter-component communication management

Our model supports three types of inter-component communication primitives for vertical and horizontal communications, which can be located at the same or different computers.

- Remote method invocation supports vertical and horizontal communications and 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 mentioned by a publisher. This approach is useful because it minimizes 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[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]

5.5 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 although the current implementation was not built for performance, we measured the cost of migrating a 4-Byte 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 version 5) connected through a fast Ethernet network. We believe that this latency is acceptable for location-aware systems used in rooms or buildings.

6 Experiments

This section describes how the model implements typical applications and what its advantages are. To evaluate the expressiveness and effectiveness of the model, we applied it to different kinds of applications.

6.1 Universal controller for nearby appliances

The first application provides a location-aware universal controller for appliances to show communications between physical entities through their counterpart objects according to their co-locations. The application

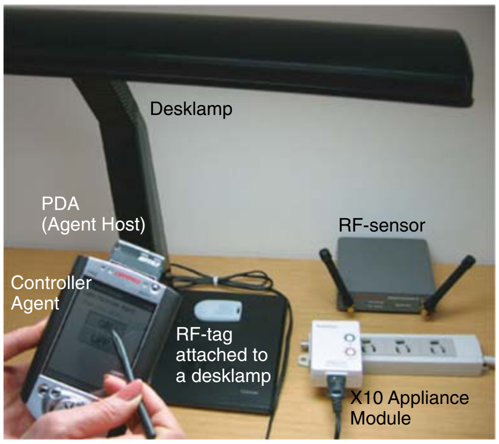

Fig. 3 Desk lamp controlled by PDA

enables people to use a PDA to remotely control electric lights in a nearby room. Active RFID tags are attached to home appliances, e.g., lights, and client-side devices, e.g., PDAs, and RFID readers detect the presence of RFID tags. Suppose that an RFID-tagged legacy light, whose power outlet can be controlled through the X10 protocol, is located at a place.

Then, the VC corresponding to the place contains the PL corresponding to the light. When a user with an RFID-tagged PDA visits a place where there is an RFID-tagged light, the model deploys the PC of the PDA at the VC as shown in Fig. 3. The VC has software that opens a control panel and runs the software on the PC. The PC uploads the software to its target device, i.e., the PDA. The software connects the PDA to a home server to control power outlets using the X10 protocol. When the PDA leaves the place, its PC retrieves the software from the PDA. Smart and non-smart objects can automatically communicate while they are both within a specified area.

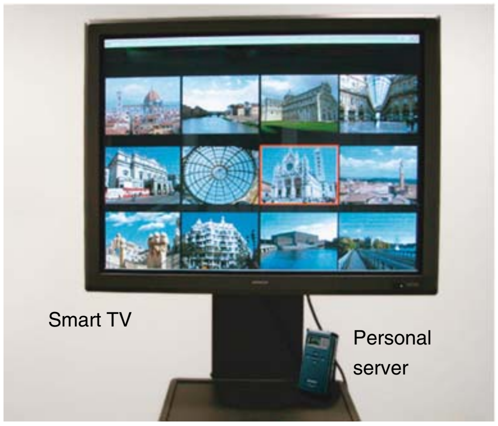

Fig. 4 Smart TV executes PCL to access image data from personal server to display GUI on screen

[2]Since 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.

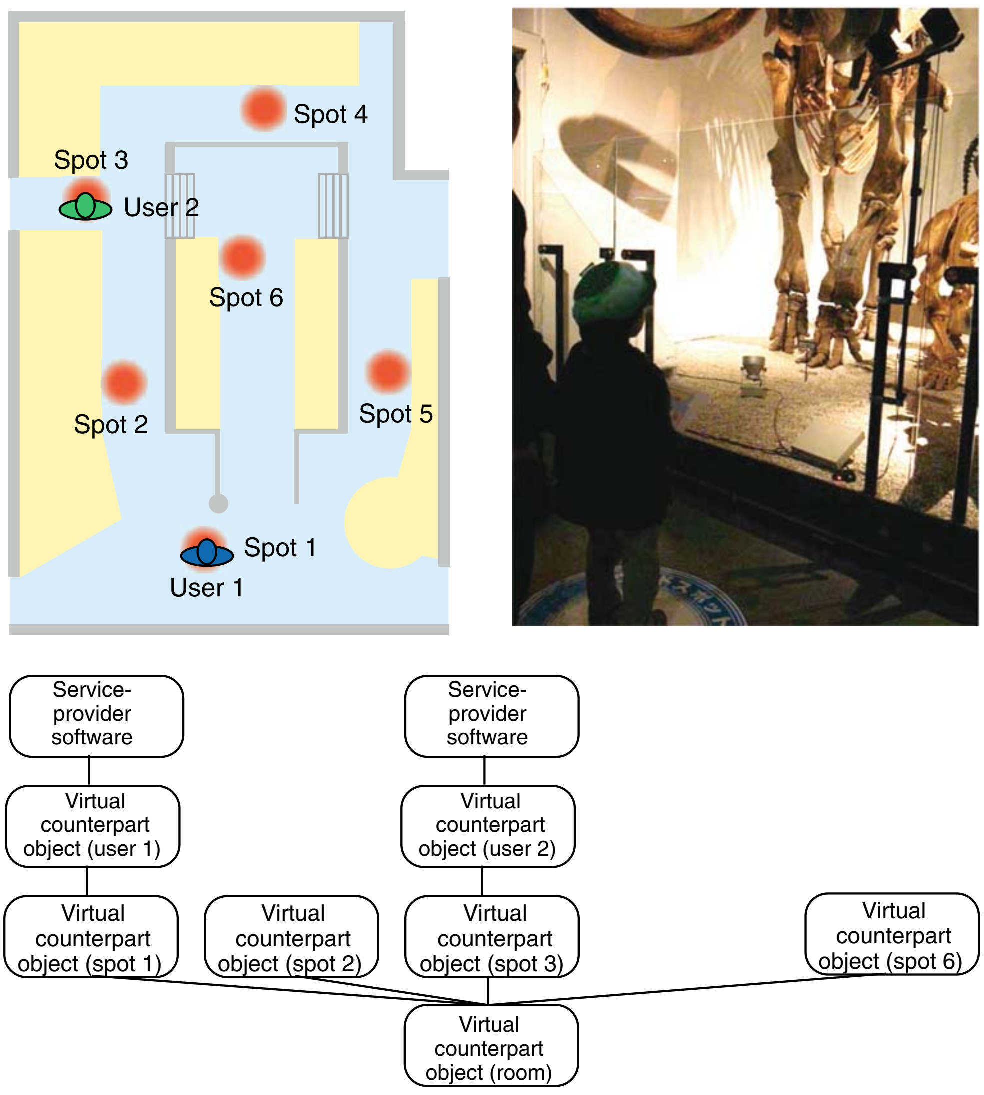
Spot 4
Spot 3
User 2
Spot 6
Spot 2
Spot 5
Spot 1
User 1
Service-provider software
Service-provider software
Virtual counterpart object (user 1)
Virtual counterpart object (user 2)
Virtual counterpart object (spot 1)
Virtual counterpart object (spot 2)
Virtual counterpart object (spot 3)
Virtual counterpart object (spot 6)
Virtual counterpart object (room)

Fig. 5 Location model of museum exhibition room

6.2 Personal server

The second application was inspired by the personal server proposed by Want (Want 2002). In our experiment we organized a handheld file-sharing personal server, a stationary computer, wall-mounted TVs, and a public terminal in a unified manner using this model. A user carries a 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 the user comes close to the stationary computer, his or her personalized services are dynamically deployed at smart TVs. When the user carries the server to a room with a TV, the model accesses the PC bound to the server in the VC corresponding to the room. Each of the TVs can be controlled through infrared control signals and has a PL as its own proxy to control it via specified infrared signals. Although the PL can forward deployable software to the server, it can still communicate with other neighboring components, including the PL, so that the user can view images stored in the server on the TV screen by using the horizontal communication mechanism shown in Fig. 4. The PL can gather image data from the server and then display it through the mechanism. This experiment shows that PCs for smart objects can dynamically customize target software and that PLs can control target objects using current communication devices, e.g., infrared signals.

6.3 Location-aware visitor assistance system for museums

We also applied our model to a framework that has already been used at the National Science Museum of Japan, which is the largest such museum in Asia. The museum has a user/location-aware audio system that

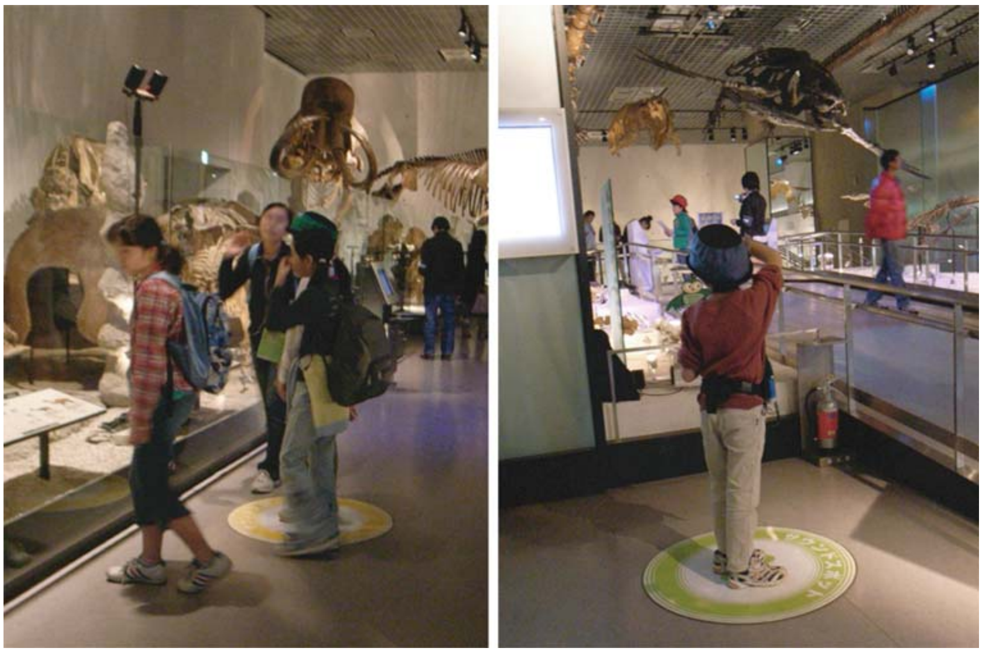

Fig. 6 Experiments in museum

describes the displays in a room where dinosaurs are exhibited.[4] We set up and operated the system's location management system using the model presented above. The exhibition room had six displays, each of which had a computing device, an RFID reader, and speakers. These areas almost overlapped their neighbors. We had participants wear hats with active RFID tags. When they entered a spot, the reader detected their presence and played the audio content for that display.

Figure 5 shows the location model of the room. Each display has its own VC, which corresponds to its readerfs coverage area. The VCs contained a PC corresponding to a computer located at the display and all the VCs were contained in a VC corresponding to the room. When a participant approached a display, the VC corresponding to it contains the userfs VC and the VCs were able to interact with each other through vertical communication. The visitor's VC contained audio player software with the userfs annotated audio data, which sent a message with the program and the data to the computing device's PC about the userfs current location through horizontal communication. These computers located at displays have extended HTTP servers that can download and automatically play sound files. This means visitors can listen to audio-based descriptions of the display they are viewing. When the VCs of one or more visitors are in a single spot, the PC can play descriptions of the displays in the order they are viewed, since the PC has a message queue.

During the experiment, there were more than 200 visitors per day (Fig. 6). The system continued to monitor and manage RFID-tag readers and location-aware services for one week without any problems. Although similar existing systems tend to depend on application-specific services and underlying systems, e.g., location-sensing systems, ours is independent of any services or underlying systems. In fact, the first such system used mat switches, instead of RFID readers, at the displays to detect the presence of visitors. When the system shifted from the first sensing system to the second, we did not have to change the model or application-specific services.

7 Conclusion

We presented a framework for building and managing location-aware services and communications in smart home 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 a VC that is implemented as executable software and can work as a proxy for its physical reference objects and places. VCs communicate with one another by directly communicating between their target smart objects, since

[4]This room is one of the most popular rooms in the museum.

home appliances are not equipped with computational resources for direct communications. These VCs enable counterparts to communicate with one another based on the geographic relationships between their physical target objects and places through two commutation primitives, i.e., vertical and horizontal communications.

Finally, we would like to identify further issues that need to be resolved. This paper described three experiments but we still need more experiments to evaluate the effectiveness and availability of our framework in various applications and smart spaces, because criteria to evaluate middleware systems for smart spaces tend to depend on target applications and spaces. The prototype implementation presented in this paper was constructed on Java, but the model itself is independent of programming languages. We are therefore interested in developing it with other languages and designing more elegant and flexible APIs for the model. We believe that our location-aware communication will be useful in supporting security mechanisms for smart spaces, including smart home.

References

Becker, C. (2004). Context-aware computing, tutorial text in IEEE International Conference on Mobile Data Management, (MDM'2004), Januray.

Beigl, M., Zimmer, T., & Decker, C. (2002). A location model for communicating and processing of context. In P. Thomas (Ed.), *Personal and ubiquitous computing* (Vol. 6(5–6), pp. 341–357). New York: Springer.

Brumitt, B. L., Meyers, B., Krumm, J., Kern, A., & Shafer, S. (2000). EasyLiving: Technologies for intelligent environments. In *Proceedings of international symposium on handheld and ubiquitous computing* (pp. 12–27).

Harter, A., Hopper, A., Steggeles, P., Ward, A., & Webster, P. (1999). The anatomy of a context-aware application. In *Proceedings of conference on mobile computing and networking* (*MOBICOM'99*) (pp. 59–68). ACM Press.

Romer, K., Schoch, T., Mattern, F., & Dubendorfer, T. (2004). Smart identification frameworks for ubiquitous computing applications. *Journal Wireless Networks, 10*(6), 689–700.

Satoh, I. (2003). SpatialAgents: Integrating user mobility and program obility in ubiquitous computing environments. *Wireless Communications and Mobile Computing, 3*(4), 411–423.

Satoh, I. (2004). Software testing for wireless mobile computing. *IEEE Wireless Communications, 11*(5), 58–64.

Satoh, I. (2005). A location model for pervasive computing environments. In *Proceedings of IEEE 3rd international conference on pervasive computing and communications* (*PerCom'05*) (pp. 215–224). IEEE Computer Society, March.

Satoh, I. (2007a). A spatial communication model for ubiquitous computing services. In *Proceedings of 4th European Conference on Universal Multiservice Networks* (*ECUMN'2007*) (pp. 32–44). IEEE Computer Society, February.

Satoh, I. (2007b). A location model for smart environment. *Pervasive and Mobile Computing, 3*(2), 158–179.

Satoh, I. (2007c). Location-aware communications in smart spaces. In *Proceedings of international conference on multimedia and ubiquitous engineering* (*MUE'2007*). IEEE Computer Society, April.

Want, R. (2002). The personal server—changing the way we think about ubiquitous computing. In *Proceedings of 4th international conference on ubiquitous computing* (*Ubicomp 2002*). *LNCS 2498* (pp. 194–209). Springer, September.

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

X10 communication (2007). X10 communication for home automation. http://www.x10.com.

Ichiro Satoh received his B.E., M.E, and Ph.D. degrees in Computer Science from Keio University, Japan in 1996. From 2001 to 2005, he was an associate professor in National Institute of Informatics (NII), Japan. Since 2006, he has been a professor of NII. His current research interests include distributed, cloud, and ubiquitous/pervasive computing. The Young Scientists' Prize awardee by the Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology. He is a member of six learned societies, including ACM and IEEE.