A World Model for Smart Spaces

Ichiro Satoh

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

Abstract. A world model for ubiquitous computing environments is presented. It can be dynamically organized like a tree based on geographical containment, such as in a user-room-floor-building hierarchy and each node in the tree can be constructed as an executable software component. It provides a unified view of the locations of not only physical entities and spaces, including users and objects, but also computing devices and services. A prototype implementation of this approach was constructed on a Java-based mobile agent system.

1 Introduction

Various computing and sensing devices are already present in almost every room of a modern building or house and in many of the public facilities of cities. As a result, spaces are becoming perceptual and smart. For example, location-sensing technologies, e.g., RFID, computer vision, and GPS, have been used to identify physical objects and track the positions of objects. These sensors have made it possible to detect and track the presence and location of people, computers, and practically any other object we want to monitor. There have been several attempts for narrowing gaps between the physical world and cyberspaces, but most existing approaches or infrastructures inherently depend on particular sensing systems and have inherently been designed for their initial applications.

A solution to this problem would be to provide a general world model for representing the physical world in cyberspaces. Although several researchers have explored such models, most existing models are not available for all ubiquitous computing, because these need to be maintained in centralized database systems, whereas the environments are often managed in an ad-hoc manner without any database servers. We also need often necessary to maintain computing devices and software in addition to modeling entities and spaces in the physical world. This paper foucsed on discussing the construction of such a model, called *M-Spaces*, as a programming interface between physical entities or places and application-specific services in cyberspaces in ubiquitous computing environments.

2 Background

Many researchers have explored world models for ubiquitous computing environments. Most existing models have been aimed at identifying and locating entities, e.g., people

and physical objects and computing devices in the physical world. These existing models can be classified into two types: physical-location and symbolic-location models. The former represents the position of people and objects as geometric information, e.g., NEXUS [5, 2] and Cooltown [6]. 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: place. Generically, place is the human-readable labeling of positions. The latter represent the postion of entities as labels for potentially overlapping geometric volumes, e.g., names of rooms, and buildings, e.g., Sentient Computing [4], and RAUM [3]. Existing approaches assume that their models are 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 manner. Virtual Counterpart [7] supports RFID-based tracking 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 should not distinguish between physical entities, places, and 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.

The framework presented in this paper was inspired by our previous work, called SpatialAgents [10], 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 of physical entities according to just the coverage areas of the RFID readers so that it could not represent any containment relationship of physical spaces, e.g., rooms and buildings. Moreover, we presented another location model, called *M-Space* [11] and the previous model aimed at integrating between software-based services running on introducing computing devices and service-provider computing devices whereas the model presented in the paper aims at modeling containment relationship between physical and logical entities, including computing devices and software for defining services.

3 World Model

This section describes the world model presented in this paper. The model manages the locations of physical entities and spaces through symbolic names.

Hierarchical World Model Our model consists of elements, called components, which are just computing devices or software, or which are implemented 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 associated with these entities or places.

– **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.

– **Component structure:** Each component can be contained within at most one component according to containment relationships in the physical world and cyberspace.
– **Inter-component movement:** Each component can move between components as a whole with all its inner components.

When a component contains other components, we call the former component is called a *parent* and the latter *children*, like the MobileSpaces model [8]. 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 of components corresponding to moving entities, their source and destination. Each component is a virtual counterpart of its target in the world model and maintains the target's attributes. Fig. 1 shows the correlation between spaces and entities in the physical world and their counterpart components. 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. Since each component in the model is treated as an autonomous programmable entity, it can some defines behaviors with some intelligence.

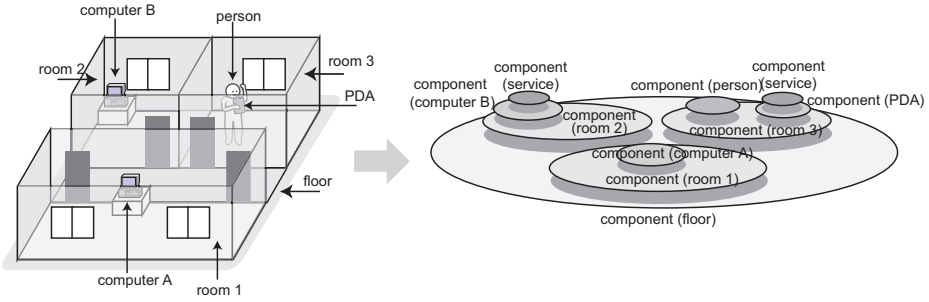

Fig. 1. Rooms on floor in physical world and counterpart components in location model.

Components The model is unique to existing world 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. As we can see from Fig. 2, components can be classified into three types.

– **Virtual Counterpart Component (VCC)** is a digital representation of a physical entity, such as a person or object, except for a computing device, or a physical place, such as a building or room,
– **Proxy Component (PC)** is a proxy component that bridges the world model and computing device, and maintains a subtree of the model or executes services located in a VCC.
– **Service Component (SC)** is software that defines application-specific services dependent on physical entities or places.

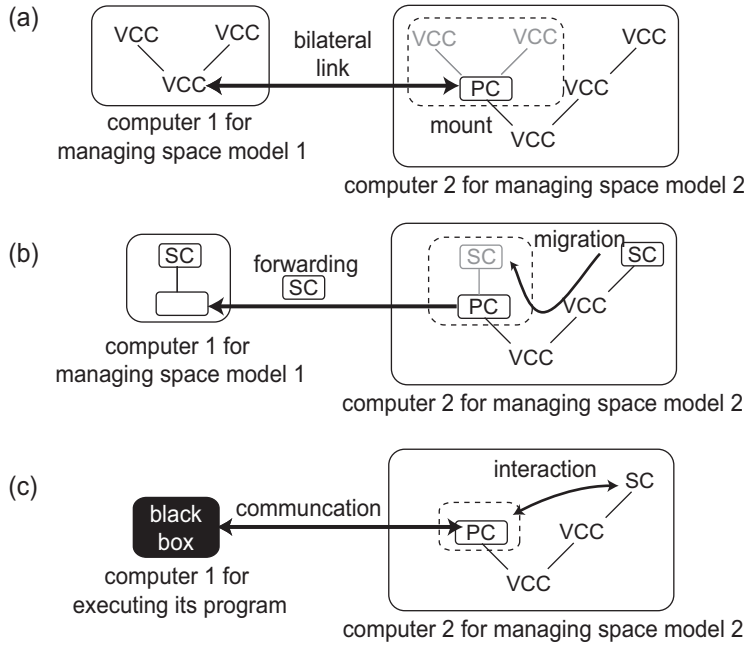

Fig. 2. Three types of proxy components

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

Furthermore, the model also classifies PCs into three subtypes, PCM (PC for Model manager), PCS (PC for Service provider), and PCL (PC for Legacy device), according to the functions of the devices. Our model can be maintained by not only the server but also multiple computing devices in ubiquitous computing environments.

- The first component, i.e., PCM, is a proxy of a computing device maintaining a subtree of the components in the world model (Fig. 2(a)). It attaches the subtree of its target device to a tree maintained by another computing device. Some computing devices can provide runtime systems to execute services defined as SCs.
- The second component, i.e., PCS, is a proxy of the computing device that can execute SCs (Fig. 2(b)). If such a device is in a space, its proxy is contained by the VCC corresponding to the space. When a PCS receives SCs, it forwards these to the device that it refers to.
- The third component, called PCL (PC for Legacy device), is a proxy of the computing device that cannot execute SCs (Fig. 2(c)). If such a device is in a space, its proxy is contained by the VCC corresponding to the space and it communicates with the device through the device's favorite protocols.

For example, a television, which does not have any computing capabilities, can have an SC in the VCC corresponding to the physical space that it is contained in and can be controlled in, and the SC can send infrared signals to it. A computing device can have different PCs whereby it can provide the capabilities to them.

4 Implementation

To evaluate the model described in Section 4, we implemented a prototype system that builds on this model. The model itself is independent of programming languages but the current implementation uses Java (J2SE or later versions) as an implementation language for components.

Component

Virtual Counterpart Component: Each VCC is defined from an abstract class, which has some built-in methods that are used to control its mobility and life-cycle. It can explicitly defines its own identifier and attributes.

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

Proxy Component: PCs can be classified into three classes, i.e., PCM, PCS, and PCL. 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 control messages to its target device from the device that it is located at, and vice versa, by using the component migration mechanism. Each PCS is a representation of the computing device that can execute SCs. It automatically forwards its visiting SCs to its target device by using the component migration mechanism. Each PCL supports a legacy computing device that cannot execute SCs due to limitations with its 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 (SC) 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 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 achieved by using mobile agent technology. The current implementation assumes SCs to be Java programs. It can be dynamically deployed at computing devices. Each SC consists of service methods and is defined as a subclass of abstract class `ServiceComponent`. Most serializable JavaBeans can be used as SCs.

```
class ServiceComponent extends Component {
  void setName(String name)
  Host getCurrentHost() { ... }
  void setComponentProfile(ComponentProfile cpf) { ... }
  ....
}
```

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 the user 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 [12]. Each SC can register such capabilities by invoking the `setComponentProfile()` method.

Component Management System

Our model can manage the computing devices that maintain it. This is because a 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 it receives other components and control messages, it automatically forwards the visiting components or messages to the device that it refers to (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.

Component migration in a component hierarchy is done merely as a transformation of the tree structure of the hierarchy. When a component is moved to another component, a subtree, whose root corresponds to the component and branches correspond to its descendent component is moved to a subtree representing the destination. When a component is transferred over a network, the runtime system stores the state and the code of the component, including the components embedded within it, into a bit-stream formed in Java's JAR file format that can support digital signatures for authentication. The system has a built-in mechanism for transmitting the bit-stream over the network

through an extension of the HTTP protocol. The current system basically uses the Java object serialization package for marshaling 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 agent to stop its active threads.

People should only be able to access location-bound services, e.g., printers and lights, that are installed in a space, when they enter it carrying their own terminals or using public terminals located in the space. Therefore, this model introduces a component as a service provider for its inner components. That is, each VC can access its neighboring components, e.g., SCs and PCs located in the parent (or an ancestor) of the VC. For example, when a person is in the room of a building, the VC corresponding to the person can access SCs (or SCs on PCs) in the VC corresponding to the room or the VC corresponding to the building. In contrast, it has no direct access over other components, which do not contain it, for reasons of security. Furthermore, like Unix's file-directory, the model enables each VC to specify its owner and group. For example, a component can explicitly permit descendent components that belong to a specified group or are owned by its user to access its services, e.g., PCs or SCs.

Location-sensor Management The model offers an automatic configuration mechanism 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. Each LCM manages location sensors and maintains a database where it stores bindings between references of physical entities in sensors, e.g., the identifiers of RFID tags attached to the entities and the identifiers of VCCs corresponding to the entities. Each LCM is responsible for discovering VCCs 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 VCCs (or PCs) bound to the entity (or device) and informs computing devices that maintain the VCCs (or PCs) about the VCC corresponding to the reader. Then the VCCs (or PCs) migrate to the reader's VCC. 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 VCC as a new entity. When the tag is attached to an unknown device that can maintain a subtree or execute SCs, the LCM instructs the VCC that contains the device to create a default PCM or PCS for the device.

5 Applications

This section briefly discusses how the model represents and implements typical applications and what advantages the model has.

5.1 Follow-Me Applications

Follow-me services are a typical application in ubiquitous computing environments. For example, Cambridge University's Sentient Computing project [4] enabled applications

to provide a location-aware platform using infrared-based or ultrasonic-based locating systems in a building.[1] While a user is moving around, the platform can track his or her movement so that the graphical user interfaces of the user's applications follow the user. 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 the applications, it can also migrate such applications themselves to computers near the moving users. That is, the model provides each user with more than one VCC and can migrate this VCC to a VCC corresponding to the destination. For example, we developed a mobile window manager, which is a mobile agent and could carry its desktop applications as a whole to another computer and control the size, position, and overlap in the windows of the applications. Using the model presented in this paper, the window manager could be easily and naturally implemented as a VCC bound to the user and desktop applications as SCs. They could be automatically moved to a VCC corresponding to the computer that was in the current location of the user by an LCM and could then continue processing at the computer, as outlined in Fig. 3.

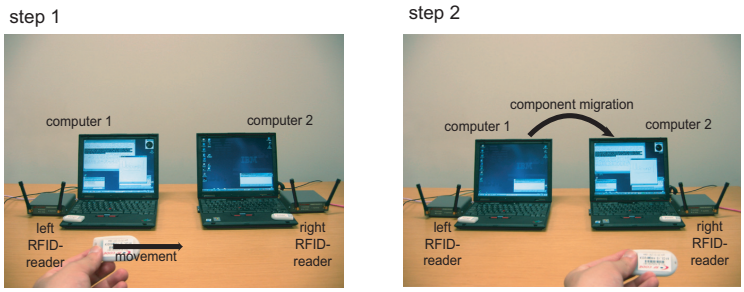

Fig. 3. Follow-me desktop applications between two computers.

5.2 Location-based Navigation Systems

The next example is a user navigation system application running on portable computing devices, e.g., PDAs, tablet-PCs, and notebook PCs. The initial result on the system was presented in a previous paper [10]. There has been a lot of research on commercial systems for similar navigation, e.g., CyberGuide [1] and NEXUS [5]. Most of those have assumed that portable computing devices are equipped with GPSs and are used outdoors. Our system is aimed at use in a building. As a PDA enters rooms, it displays a map on its current position. We has assumed that each room in a building has a coverage of more than one RFID reader managed by an LSM, the room is bound to a VC that has a service module for location-based navigation, and each PDA can execute service modules and is attached to an RFID tag. When a PDA enters a room, the RFID reader for the room detects the presence of the tag and the LSM tries to discovery the component bound to the PDA through the procedure presented in the previous section. After

[1] The project does not report their world model but their systems seem to model the position of people and things through lower-level results from underlying location-sensing systems.

it has information about the component, i.e., a PCS bound to a PDA, it informs to the VC corresponding to the room about the capabilities of the visiting PDA . Next, the VC deploys a copy of its service module at the PCS and then the PCS forwards the module to the PDA to which it refers to display a map of the room. When the PDA leaves from the room, the model issues events to the PCS and VC and instructs the PCS to returns to the VC. Fig. 4 (right) outlines the architecture for the system. Fig. 4 (left) shows a service module running on a visiting PDA displaying a map on the PDA's screen.

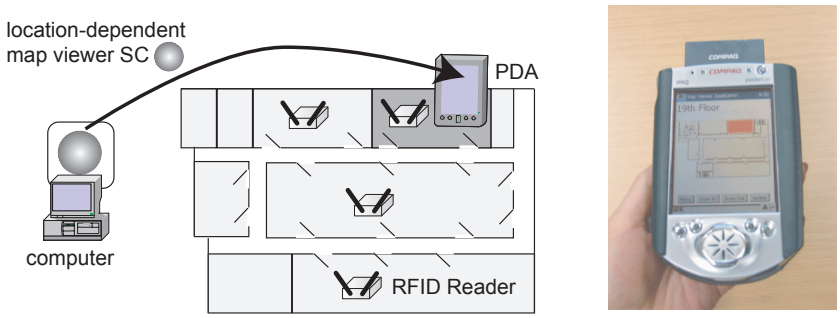

Fig. 4. RFID-based location-aware map-viewer service and location-aware map-viewer service running on PDA

5.3 Software Testing for Location-based Services

To test software for location-based services running on a portable device, the developer often has to carry the device to locations that a user's device may move to and test whether software can connect to appropriate services provided in the locations. We developed a novel approach to test location-aware software running on portable computing devices [9]. The approach involves a mobile emulator for portable computing devices that can travel between computers, and emulates the physical mobility and reconnection of a device to sub-networks by the logical mobility of the emulator between sub-networks. In this model, such an emulator can be naturally implemented as a PC, which provides application-level software, with the internal execution environments of its target portable computing devices and target software as SCs. The emulator carries the software from a VCC that is running on a computer on the source-side sub-network to another VCC that is running on another computer on the destination-side sub-network. After migrating to the destination VCC, it enables its inner SCs to access network resources provided within the destination-side sub-network. Furthermore, SCs, which were tested successfully in the emulator, can run on target computing devices without modifying or recompiling the SCs. This is because this model provides a unified view of computing devices and software and enables SCs to be executed in both VCCs and PCs.

6 Conclusion

We presented a world model for context-aware services, e.g., location-aware and personalized information services, in ubiquitous computing environments. Like existing related models, it can be dynamically organized like a tree based on geographical containment, such as a user-room-floor-building hierarchy and each node in the tree can be constructed as an executable software component. It also has several advantages in that it can be used to model not only stationary but also moving spaces, e.g., cars. It enables context-aware services to be managed without databases and can be managed by multiple computers. It can provide a unified view of the locations of not only physical entities and spaces, including users and objects, but also computing devices and services. We also designed and implemented a prototype system based on the model and demonstrated its effectiveness in several practical applications.

References

1. G.D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, Cyberguide: A Mobile Context-Aware Tour Guide, ACM Wireless Networks Vol. 3, pp.421–433. 1997.
2. M. Bauer, C. Becker, and K. Rothermel Location Models 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.
3. 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
4. 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.
5. 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).
6. 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.
7. 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.
8. 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.
9. I. Satoh, A Testing Framework for Mobile Computing Software, IEEE Transactions on Software Engineering, vol. 29, no. 12, pp.1112-1121, December 2003.
10. I. Satoh, Linking Physical Worlds to Logical Worlds with Mobile Agents, Proceedings of International Conference on Mobile Data Management (MDM'2004), IEEE Computer Society, January 2004.
11. 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.
12. World Wide Web Consortium (W3C), Composite Capability/Preference Profiles (CC/PP), http://www.w3.org/ TR/NOTE-CCPP, 1999.