A Visual Framework for Deploying and Managing Context-Aware Services

Ichiro Satoh*

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Abstract. A framework for managing pervasive computing is presented. It enables end-users to easily and naturally build visual interfaces for monitoring and customizing context-aware services. It is built on an exiting a symbolic location model to represent the containment relationships between physical entities, computing devices, and places. It supports a compound document framework for visualizing and customizing the model. It provides physical entities, places, computing devices, and services in smart spaces with visual components to annotate and control them and to dynamically assemble visual components into a visual interface for managing the spaces. It can visualize and configure the spatial structure of physical entities and places and the status and attributes of computing devices and services, e.g., the location in which context-aware services are available. By using the framework, end-users can monitor and customize pervasive computing environments by viewing and editing documents.

1 Introduction

Pervasive computing tends to consist of many computing devices like grid computing. However, the former often lacks management systems, unlike the latter. In fact, the focus of current research on pervasive computing is on the design and implementation of application-specific context-aware services. As a result, the task of management in pervasive computing has attracted scant attention so far. This is a serious obstacle in the growth of pervasive computing. The purpose of pervasive computing is to bridge the gap between computing systems and the real world. In fact, one of the most typical and popular applications of pervasive computing is in context-aware services. To support such services, pervasive computing systems must be able to know the context and process this in the real world, e.g., people, location, and time. Such information tends to depend on the offices/houses, businesses and lifestyles of users. Therefore, they must customize many pervasive computing devices to their individual requirements and applications. Pervasive computing systems often lack professional administrators unlike grid computing systems.

This paper presents a user-friendly management framework to solve these problems. It was inspired by our experiences with practical applications of pervasive computing in the real world, e.g., home appliance controls and location/user-aware user-assistance

* e-mail: ichiro@nii.ac.jp

C. Cérin and K.-C. Li (Eds.): GPC 2007, LNCS 4459, pp. 397–411, 2007.

systems. The framework provides visual interfaces for deploying, customizing, and controlling computing devices and context-aware services. Since pervasive computing environments are changed dynamically, such a management framework, including visual interfaces, for these environments, must be able to autonomously adapt itself to the changes. For example, when devices and services are added to a smart space, a visual interface for managing the devices or services should be added to the interface for the space. The framework is constructed as a combination of a location model, called M-Spaces [11], and an active document framework, called MobiDoc [12,13], developed by the author. The former is a symbolic-location model to maintain the locations of computing devices and software for defining context-aware services as well as the locations of physical spaces and entities in the real world. The latter is constructed as a Java-based compound document framework. It enables one document to be composed of various visible parts, such as text, image, and video created by different applications, like other compound document frameworks, e.g., COM/OLE [3], OpenDoc [1], CommonPoint [10], and Bonobo [7]. The framework presented in this paper provides visual interfaces for a location model as a management tool for end-users to deploy, customize, and monitor context-aware services. Since the framework itself is designed independently of the location model as much as possible, it can be used for other location models for pervasive computing.

2 Background

The framework presented in this paper has two bases, i.e., symbolic-location model and compound document framework. The former is useful for providing context-aware services in smart spaces, because such a model is useful for context-aware services as discussed in the previous section. The latter enables end-users to build a visual management interface from components for compound documents and customize context-aware services through GUI-manipulations. This paper addresses location-aware communication between humans-machines or between machines-machines indoors, e.g., in buildings and houses, rather than in outdoor settings.

2.1 Symbolic Location Model

The current implementation is constructed with a symbolic-location model, called *M-Spaces* [11]. The framework can be used with other existing symbolic location models. It enables us to monitor contextual information in the models, but we cannot manage context-aware services, because the models themselves do not support services and computing devices. The M-spaces model can spatially bind the positions of entities and spaces with the locations of their virtual counterparts by using location sensing systems, and when they move in the physical world, it can automatically deploy their counterparts at proper locations within it. Physical spaces and entities 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, each room is contained within at most one floor, and a person or object may be contained in at most one room. Unlike other existing location models, it can maintain the location

and deployment of software to define context-aware services and information about the computational resources of computing devices that can execute the services, as well as represent contextual information in the real world like other existing location models.

2.2 Compound Document-Based Management Interface

The framework presented in this paper uses a compound document component framework, called MobiDoc [12,13], as a visual user interface to monitor changes in the real world and deploy and customize context-aware services. It enables an enriched document to be dynamically and nestedly composed of software components corresponding to various types of content, e.g., text, images and windows. Unlike other existing compound document frameworks, it permits the content of all components and program codes to access the content that is inseparable within the components so that the components can be viewed or modified without the need for any applications. It provides an editing environment to enable the visual components to be manipulated. It also provides in-place editing services similar to those provided by OpenDoc and OLE. It offers several value-added mechanisms to allow the visual estate of a container to efficiently be shared among embedded components and to coordinate their use of shared resources, such as keyboards, mice, and windows.

2.3 Basic Approach

The framework presented in this paper provides more than one visual component for a virtual counterpart object corresponding to a physical entity, space, computing device, and service to bridge the gap between the location model and the compound document framework. Visual components are organized according to the structure of their target virtual counterpart objects and they enable the spatial relationships between the objects's targets to be visualized, e.g., physical entities, objects, and computing devices in the model (Fig. 1).

The framework supports bidirectional communications between runtime systems for virtual components and the model and communications between each visual component and virtual counterpart objects that the component represents. The framework reflects the structure of virtual counterpart objects in the structure of visual components and it permits the runtime systems to request the model to change the structure of virtual counterpart objects. We can customize the locations that the services should be available at and the users that the services should be provided for, by deploying visual components for context-aware services at other visual components corresponding to entities and places through GUI manipulations. Furthermore, since each virtual component is a programmable entity, it can directly communicate with its target counterpart object to visualize and customize the status and attributes of the object's target, e.g., physical entity, and place, computing device, and service via the object, through its built-in protocols or the object's favorite protocols.

Since compound document technology supports the dynamic composition of components, compound document-based management interfaces for pervasive computing environments can adapt themselves to changes in the physical world. For example, when computing devices and services are added, their visual components are dynamically

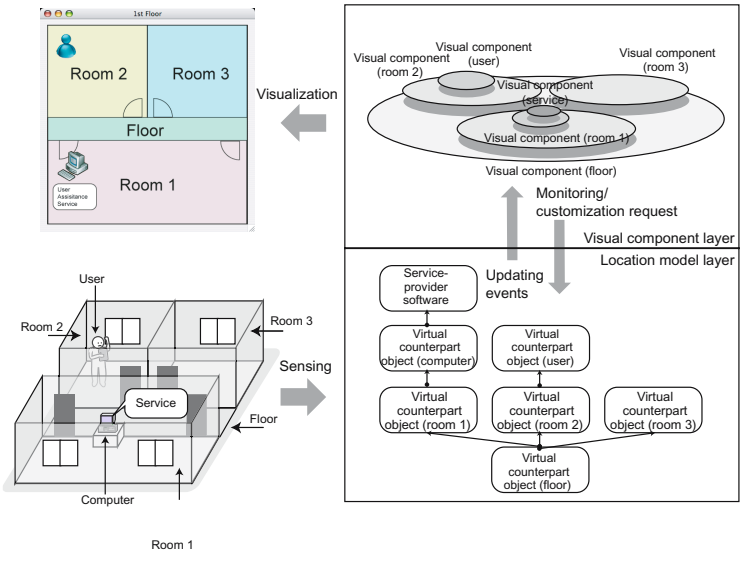

Fig. 1. Rooms on floor in physical world and virtual counterpart objects in location model

downloaded from specified servers or devices and then automatically displayed within the scope of the components corresponding to the spaces that contain them.

2.4 Remarks

We should explain the reason why our framework supports two layers. This is because the upper layer, i.e., visual interfaces, should be general so that is can be used for other models and other computing, including grid computing. In fact, it is designed independently of the lower layer, i.e., the M-Spaces model and can support non tree-based models. We should also note that the framework itself can be easily used for other locations models to monitor them but it does not support the deployment and customization of context-aware services, because they cannot maintain any services and computing devices unlike the M-Spaces model.

3 M-Space: Location Model for Smart Spaces

Existing location models can be classified into two: physical world and symbolic world. The former represents the position of people and objects as geometric information, which can be measured by GPS and ultra-sonic location sensing systems. The former is not suitable in indoor settings, because although the geometric locations of two objects may be neighboring, the objects themselves may be in different rooms. In fact, most emerging applications in indoor settings require a more symbolic notion. We use a symbolic location model, called *M-Spaces* model. This section outlines the model before explaining the framework.[1]

[1] Detail of the model was presented in our previous paper[11].

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 entities, called agents, as virtual counterpart objects of physical entities or places. Agents have the following notions: (1) Each agent is a virtual counterpart of a physical entity or place, including the coverage area of the sensor, computing device, or service-provider software. (2) Each agent can be contained within at most one agent according to containment relationships in the physical world and cyberspace. It can move between agents as a whole with all its inner agents. Agents When an agent contains other agents, we call the former a *parent* and the latter *children*. The model permit agents to interact with each others. The model represents facts about entities or places in terms of the semantic or spatial-containment relationships between agents that are associated with these entities or places. 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 agents corresponding to moving entities, their sources, and destinations. The below figures of Fig. 1 shows the correlation between spaces and entities in the physical world and their counterpart agents. Each agent is a virtual counterpart object of its target in the world model and maintains the target's attributes.

3.2 Agent

The model cannot 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.

- **The virtual counterpart agent (VCA)** is a digital representation of a physical entity, such as a person or object, except for the computing device itself, or physical surroundings such as a building or room,
- **The proxy agent (PA)** bridges the model and computing device, and maintains a subtree of the model or executes services located in a VCA.
- **The service agent (SA)** 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 VCA on the model and this contains two VCAs that correspond to the two people. The movement of the car is mapped into the VCA migration corresponding to the car, from the VCA corresponding to the source to the VCA corresponding to the destination. Also, when a person has a computer for executing services, his or her VCA has a PA, which represents the computer and runs SAs to define the services.

Virtual counterpart agent. A person, physical object, or place can have more than one VCA, and each VCA can contain other VCAs and PAs 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.

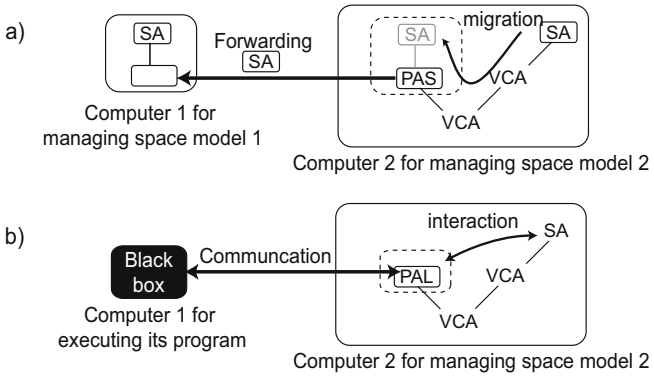

Fig. 2. Two types of proxy agents

Proxy agent. VCAs can have software to define the context-dependent services inside them. However, they may not be able to be executed in the software, because none of the computing devices that maintain these have unlimited computational resources. Instead, there are two facilities through 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 agents to maintain the location of computing devices and used the devices as service providers.[2] Our model also allows PAs to be classified into two sub-types that handle computing devices according to their functions.

- The first agent, i.e., PAS (PA for Service provider), is a proxy of a computing device that can execute services (Fig. 2(a)). If such a device is in a place, its proxy is contained in the VCA corresponding to the space. When a PAS receives software for defining services, it forwards this to the device to which the software refers. After the PAS forwards the software, it enables other agents to fetch the software as if this were in it.
- The second agent, called PAL (PAC for Legacy device), is a proxy of a computing device that cannot execute SAs (Fig. 2(b)). If such a device is in a space, its proxy is contained in the VCA corresponding to the space and it communicates with the device through the device's favorite protocols.

Service agent. We should reuse existing location-based and personalized services as much as possible. The model introduces several typical software agents, e.g., Java Beans and Java Applets as service-provider programs. However, such existing agents may not be suitable for our model. Each SA 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 SAs.

[2] Proxy agents are unique to other existing location models and are useful for maintaining and using computing devices.

4 Compound Document Framework for Managing Pervasive Computing

This section presents a compound document framework for building and operating visual interfaces for context-aware services. The framework inherits many features of our compound document framework, `MobiDoc`, but is extended to manage pervasive computing. The framework provides each agent in the model with more than one visual component to view and customize the status and attributes of the agent by using the program code defined in the agent. It organizes these components in a tree structure according to their target agents. It consists of two parts: component runtime systems and visual components. The former can communicate with the model and organize visual components. The latter maintains its visual content and program code to enable content inside it be viewed or edited.

4.1 Visual Component

Each visual component is a collection of Java objects wrapped in a component and it has its own unique identifier and image data displayed as its icon. All the objects that each component consists of need to implement the `java.io.Serializable` interface, because they must be marshaled using Java's serialization mechanism. Each visual component needs to be defined as a subclass of either the `java.awt.Component` or `java.awt.Container` from which most of Java's visual or GUI objects are derived. To reuse existing software, we implemented an adapter to use typical Java components, e.g., Java Applets and JavaBeans, that are defined as subclasses of the `java.awt.Component` or `java.awt.Container` class within our components. This is not compatible with all kinds of Applets and JavaBeans, because some of those existing components manage their threads and input and output devices depreciatively. Nevertheless, the framework provide adapters for several canonical Applets and JavaBeans to be used as visual components.

4.2 Component Runtime System

Each runtime system governs all the components within it and provides them with APIs for components in addition to Java's classes. It assigns one or more threads to each component and interrupts them before the component migrates, terminates, or is saved. Each component can request its current runtime system to terminate, save, and migrate itself and its inner components to the destination that it wants to migrate to. This framework provides each component with a wrapper, called a *component tree node*. Each node contains its target component, its attributes, and its containment relationship and provides interfaces between its component and the runtime system. When a component is created in a runtime system, it creates a component tree node for the newly created component. When a component migrates to another location or duplicates itself, the runtime system migrates its node with the component and makes a replica of the whole node.

Each VCA, PA, and SA, has more than one visual component and the structure of VCAs, PAs, and SAs in the model is reflected in the hierarchical structure of visual components. Each hierarchy is maintained in the form of a tree structure of component tree

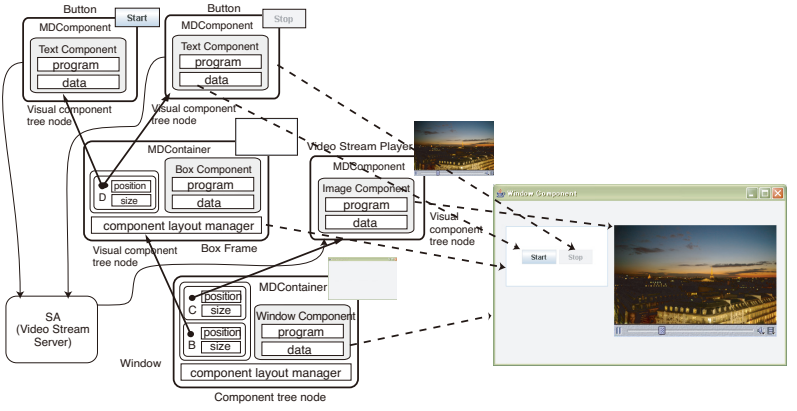

Fig. 3. Visual component hierarchy

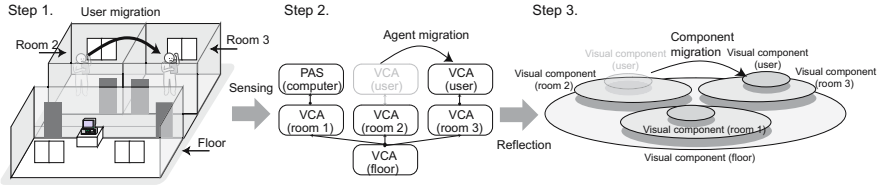

Fig. 4. The movement of agents and components when changes in the real world

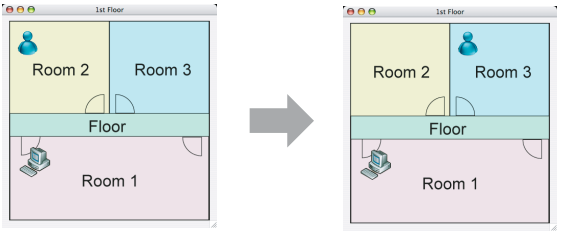

Fig. 5. Relocation of visual components

nodes of components (Fig. 3). Each node is defined as a subclass of `MDContainer` or `MDComponent`, where the first supports components, which can contain more than one component inside them while the second supports components, which cannot contain any components. For example, when a component has two other components inside it, the nodes that contain these two inner components are attached to the node that wraps the container component. Component migration is only performed as a transformation of the subtree structure of the hierarchy. The framework does not support direct-interactions between visual components. Instead, it permit each VCA, PA, or SA, to have more than one visual component.

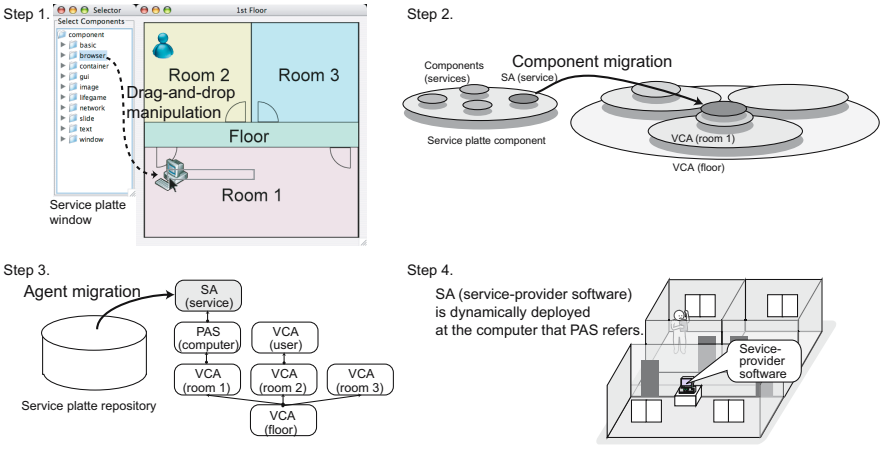

Fig. 6. Dynamic service-deployment according to migration of visual component

5 Binding Between Visual Components and Virtual Counterparts

The framework permits each agent to have more than one visual component. When it detects changes in the attributes of an agent, it sends events to the visual components that refer the agent.

5.1 Updating the Structure and Attributes of Visual Components

Component runtime systems support WebDAV servers. When the framework detects changes in the structure of agents in the model (Fig. 4), it transforms the structure of visual components that refer the agents by sending WebDAV-based commands to the runtime systems (Fig. 5). When new physical entities and people arrive at spaces, visual components that refer the counterpart objects for the visiting entities or people may not be available in these runtime systems. When entities or people leave from spaces, visual components for the missing entities or people may be unnecessary. To solve these problems, the framework provides a mechanism for fetching/dispatching components from/to specified servers, called repository servers. When a component is fetched from or dispatched to servers, the runtime system marshals the node of the component, including its state and codes, and the nodes of its descendants, into a bit-stream by using Java's object serialization mechanism and then transmits the bit-stream to/from the servers. Therefore, the attributes and structure of visual components become persistent, even while they are stored in these servers.

5.2 Updating the Structure and Attributes of Agents

Each component can display its content within the rectangular estate maintained by its container component. The node of the component, which is defined as a subclass of the `MDContainer` or `MDComponent` class, specifies attributes, e.g., its minimum size and preferable size, and the maximum size of the visible estate of its component

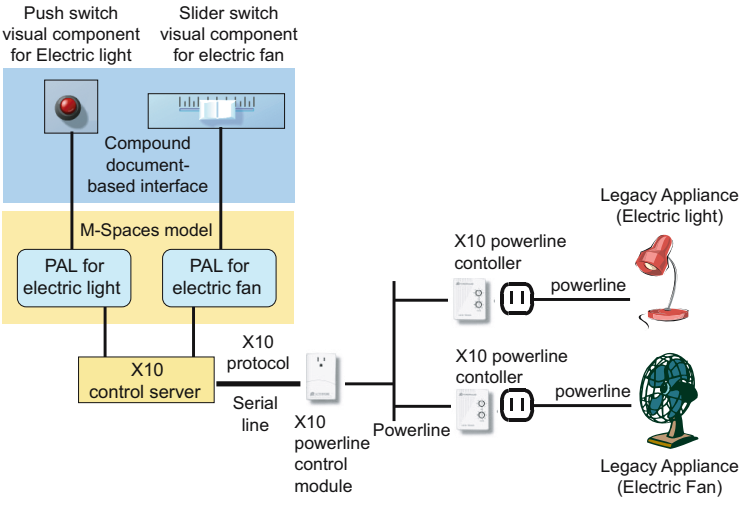

Fig. 7. X10-based power-outlet controlling system

in the estate is controlled by the node of its container component. These classes can define their new layout manager as subclasses of the `java.awt.LayoutManager` class. When a component is dynamically added to a container, the layout manager of the container's MDContainer manage the position and size of the new component. For example, if a container has an instance of Java's `java.awt.FlowLayout` as its layout manager, components that visit it automatically stand in rows in its estate.

This framework provides an editing environment for manipulating the components for network processing, as well as for visual components. It offers several value-added mechanisms for effectively sharing the visual estate of a container among embedded components and for coordinating their use of shared resources, such as keyboards, mice, and windows. Each component tree node can dispatch certain events to its components to notify them when certain actions occur within their surroundings. `MDContainer` and `MDComponent` classes support built-in GUIs for manipulating components. For example, when we want to place a component on another component, including a document, we move the former to the latter through GUI manipulations, e.g., drag-and-drop or cut-and-paste.

When users change the structure or attributes of visual components, the framework sends events to the model to update the structure or attributes of corresponding agents (Fig. 6). When the underlying sensing system detects the arrival of people and physical objects, the model fetch and load agents corresponding to these people and objects from such storage and then issue specified events to runtime systems. To duplicate agents or components, the system marshals them into a bit-stream and then duplicates the marshaled agent or component, because Java has no deep-copy mechanisms that can make replicas of all objects embedded in and referred to from these components.[3]

[3] Since the framework treats a component and its clones as independent, it does not support any consistency control mechanisms between them.

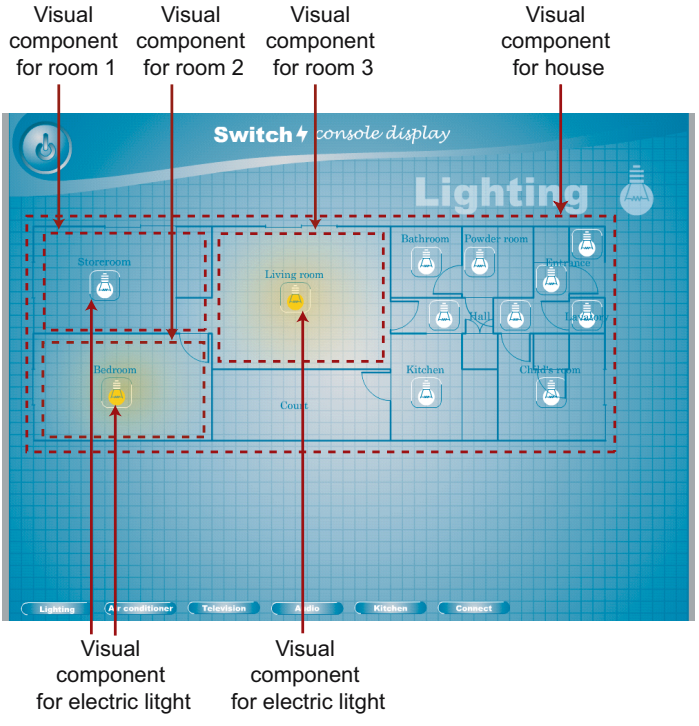

Fig. 8. Screenshot of remote control interface

6 Early Experience

We developed various components for managing VCAs, PASs, PALs, and SAs as well as basic visual components, e.g., text viewer/editor component , JPEG or GIF viewer components, and stream-video player components.[4] Most java Swing and AWT GUI Widgets can be used as our components in the framework without modifications, because they have been derived from the `java.awt.Component` class. The performance of visual components is reasonable as management interfaces.

We describe a remote controller for power-outlets of lights through a commercial protocol called X10 with this framework. The lights are controlled by switching their power sources on or off according to the X10-protocol. We provide all lights with their PALs to switch them on or off. Each PAL communicates with an X10-base server, which controls an X10-module connected to the power-outlet to switch the outlet on or off, and it has its own visual component to display the GUI of its target (Fig. 7). The current implementation of the component sends commands to its PAL through an HTTP-based protocol. When a new PAL is added to the model, it sends a specified event to the component runtime system, which downloads a visual component for the PAL.

[4] Visual components corresponding to visual components, e.g., documents, image viewers, and text editors, were presented in our previous papers [12,13].

We developed an improved version of the remote controller for electronic lights in several rooms of a house and each room had more than one light. The VCA corresponding to the house contained the VCA corresponding to the rooms in the containment relationship between these physical spaces and entities, We constructed an interface for the controller with the framework (Fig. 8). The visual component for the VCA corresponding to the house had several visual components displayed for the VCAs corresponding to the rooms in its area. The visual component for the house drew a map of arrangement of the rooms in the house. It contained VCAs corresponding to the rooms in spaces corresponding to the rooms on the map. A VCA corresponding to a room could contain PALs and PASs, e.g., PALs for controlling X10 modules connected to power outlets in the room through the X10 protocol. The interface was used to control home appliances, including lights.

6.1 Management System for Context-Aware Services

The second application is a management system for context-aware assistant services. The system was constructed with the framework and actually used at an exhibition in a public museum. This was in the Museum of Nature and Human Activities at Hyogo, Japan, which mainly has information and objects that concerned the natural environment. The exhibition space had RFID-tag readers installed and visitors were provided with active RFID-tags to track their locations. When they came sufficiently close to some objects, e.g., zoological specimens and fossils, located at several spots in the exhibition, they could listen to sound content that annotated the objects. The RFID-tag readers identified all the visitors within their coverage range, i.e., a 2-meters diameter and selected sound content according to their knowledge and interests. Fig. 9 shows

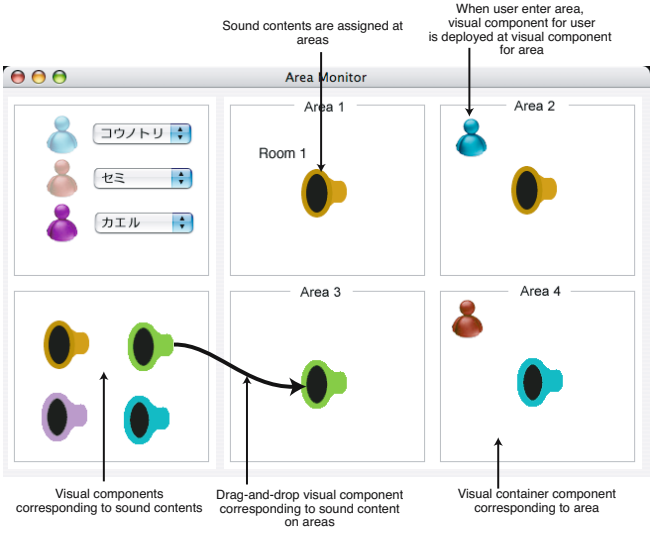

Fig. 9. Screenshot of monitor system windows for location/user-aware audio guiding system

a screenshot of the visual interface for the management system. The interface enables users to deploy services at areas by using drag-and-drop manipulation. Each day the exhibition has more than 200 visitors and the system continued to monitor and manage RFID-tag readers and location-aware services for one week without any experiencing problems.

The interface consisted of four visual components that monitored four RFID-tag readers located at spots throughout the exhibition. When a visitor with an RFID-tag entered a spot, the VCA corresponding to him or her is deployed at the VCA corresponding to the spot. We could dynamically add/remove location-aware services to/from spots. To add a service to a spot, we deployed SA to define the service at the VCA corresponding to the spot by a drag-and-drop operation of the visual component of the SA on the visual component of the VCA. Curators who have no knowledge about pervasive computing systems, can easily and naturally change audio-based assistance services at the exhibition.

7 Related Work

This paper addresses a user-friendly management system of context-aware services in indoors settings, e.g., in buildings and houses, rather than in outdoor settings.

7.1 Location Models

Perceptual technologies have made it possible to sense contextual information in the real world. For example, indoor location systems, such as Radio Frequency IDentification (RFID) tag systems, measure and track the locations of physical entities attached to RFID tags. Existing context-aware services tend to be selected and operated in an ad-hoc manner. For example, most existing services explicitly and implicitly depend on the underlying sensing systems. They are not available with other sensing systems that they have not initially assumed. To solve these problems, some research projects on context-aware services have attempted to offer general-purpose world models to cancel the differences between sensing systems. Since location is one of the most typical and useful kinds of contextual information, location models will be discussed [2]. Existing location models, unfortunately, lack any user-friendly interfaces to enable end-users to easily manage and customize them.

7.2 Management Systems for Pervasive Computing

As mentioned in the previous section, there have been a few attempt to construct management systems or tools that monitor and customize context-aware services in pervasive computing environments. The EasyLiving project [4] provides context-aware spaces, with a particular focus on homes and offices. It uses mounted sensors such as stereo cameras on a room's walls and tracks the locations and identities of people in the room. The system can dynamically aggregate networked-enabled input/output devices, such as keyboards and mice, even when they belong to different computers in the space. It provides monitoring tools for visualizing the positions of users in rooms. However,

the project, including its monitoring tools, seemed only to be designed for its target rooms in an ad-hoc manner. Cambridge University's Sentient Computing project [5] provides a platform for location-aware applications using an ultrasonic-based locating system in a building. It can track the movement of tagged entities, such as individuals and objects, so that the graphical user interfaces of the users' applications follow them while they move around. It provides a visual editor to enable the ranges of location-aware services to be configured, but cannot deploy services at locations.

There have been several mechanisms for automatically generate graphical user interfaces for pervasive computing services and devices [6,9,8]. Most existing approaches can provide GUIs for individual devices and can support the dynamic generation of GUIs for devices, which may be added. However, they assume the use of specified protocols to communicate with their target devices. They do not support the deployment and configuration of context-aware services.

8 Conclusion

We presented a visual framework for monitoring and managing context-aware services in smart spaces. It supports a symbolic location model to represent the containment relationships between physical entities, spaces, computing devices, and software for defining services as virtual counterpart objects that correspond to them. It enables physical entities, places, computing devices, and services in smart spaces to have visual components to annotate and control them and to dynamically and seamlessly assemble multiple visual components into a visual interface for managing the spaces. It can monitor the spatial structure of physical entities and places and customize the status and attributes of computing devices, and services, e.g., the location in which context-aware services are available. It provides document-based interfaces to monitor and customize pervasive computing environments as viewing and editing documents by using a GUI to manipulate the compound document technology. For example, end-users can add and customize location-aware services at specified locations by deploying the visual component corresponding to the services at the visual component corresponding to the location. The framework presented in this paper can be used for the management of grid computing. Our visual components themselves are independent of the model. Since they are also programmable entities, they can communicate with computers in grid computing environments and displays various information of the computers.

References

1. Apple Computer Inc., OpenDoc: White Paper, 1995.
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. K. Brockschmidt, Inside OLE 2, Microsoft Press, 1995.
4. 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.

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. K. Gajos and D. S. Weld, SUPPLE: automatically generating user interfaces, Proceedings of the 9th International Conference on Intelligent User Interface (IUI'04) pp.93-100, ACM Press, 2004.
7. The GNOME Project, Bonobo, http://developer.gnome.org/ arch/component/ bonobo.html, 2002.
8. T. D. Hodes, R. H. Katz, E. Servan-Schreiber, L. Rowe, Composable ad-hoc mobile services for universal interaction, Proceedings of International Conference on Mobile Computing and Networking (MobiCom'97), pp.1-12, 1997.
9. J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol, Generating remote control interfaces for complex appliances, Proceedings of Symposium on User Interface Software and Technology (UIST'02), pp.161-170, ACM Press, 2002.
10. M. Potel and S. Cotter Inside Taligent Technology, Addison-Wesley, 1995.
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. I. Satoh, Network Processing of Documents, for Documents, by Documents, Proceedings of ACM/IFIP/USENIX 6th International Middleware Conference (Middleware'2005), Lecture Notes in Computer Science (LNCS), vol. 3790, pp.421-430, December 2005.
13. I. Satoh, A Document-centric Component Framework for Document Distributions, Proceedings of 8th International Symposium on Distributed Objects and Applications (DOA'2006), Lecture Notes in Computer Science (LNCS), vol.4276, pp.1555-1575, Springer, October 2006.