Rapidly Building Visual Management Systems for Context-Aware Services

Ichiro SATOH, Member

SUMMARY A component framework for building and operating visual interfaces for context-aware services in ubiquitous computing environments is presented. By using a compound-document technology, it provides physical entities, places, stationary or mobile computing devices, and services with visual components as multimedia representations to enable them to be annotated and controlled. It can automatically assemble visual components into a visual interface for monitoring and managing context-aware services according to the spatial-containment relationships between their targets in the physical world by using underlying location-sensing systems. End-users can manually deploy and customize context-aware services through user-friendly GUI-based manipulations for editing documents. This paper presents the design for this framework and describes its implementation and practical applications in user/location-aware assistants in two museums.

key words: ubiquitous computing, graphical user interface, system management, context-aware computing

1. Introduction

Context-aware computing is one of the most typical applications of ubiquitous computing. It provides users with appropriate services according to contextual information in the physical world, e.g., people, locations, and time. The focus of current research, however, has been on the design of network and system infrastructures and applications in context-aware services, including location-based services. As a result, the task of management in context-aware services has attracted scant attention so far. This is a serious obstacle to the growth of ubiquitous computing.

The selection and customization of context-aware services tends to depend on the office/houses, businesses, and lifestyles of individual users. Therefore, the end-users themselves must customize countless computing devices and services to their individual requirements and applications. Most of their computing systems lack professional administrators. Nevertheless, they tend to consist of many heterogeneous computing devices, e.g., desktop PCs, portable PCs, PDAs, and sensing systems connected through networks as large-scale distributed systems. It is almost impossible for end-users who have no professional knowledge about their systems to manage these computing devices and services.

This paper presents a user-friendly management framework to solve these problems. It was inspired by our experiences with practical applications in the real world, e.g., museums, i.e., the National Science Museum in Tokyo and the Museum of Nature and Human Activities in Hyogo, Japan. The use of user/location-aware assistance services in museums has received a great deal of attentions from researchers over past few years, because it enables visitors to view or listen to specific annotations on exhibits at the right time and the right place through their own mobile terminals or ambient displays that are located at that place. For example, when such services detect the presence of visitors in front of an exhibit, they automatically play visual or audio annotations on the area being viewed by using a display or loudspeaker close to it. There have been several academic or commercial attempts to provide these services in museums. However, most of them have been constructed in an ad-hoc manner and lack their management systems. Nevertheless, they have to be managed and customized by curators or administrators who have no professional knowledge or skill. Therefore, it is difficult to customize them, even when exhibits are frequently replaced and relocated.

The framework provides visual interfaces for deploying, customizing, and controlling computing devices and context-aware services. Since ubiquitous computing environments are changed dynamically, such management framework, including visual interfaces, for these environments must be able to autonomously adapt themselves to these 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 that space. The framework was constructed as a combination of a location model, called M-Spaces [17], and an active document framework, called MobiDoc [15], [16], developed by the author. The former is a symbolic-location model that maintains the locations of computing devices and software to define context-aware services as well as the locations of physical spaces and entities in the real world. The latter was constructed as a Java-based compound-document framework. It enables one document to be composed of various visible parts, such as texts, images, and videos created by different applications, like other compound-document frameworks, e.g., COM/OLE [5], OpenDoc [1], Common-Point [14], and Bonobo [9]. 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 has been designed independently of the location model as much as possible, it can be used for other location models in ubiquitous computing.
2. Basic Approach

This paper addresses the management of context-aware services for ubiquitous computing environments. The framework is constructed as a visual interface for tree-based symbolic-location models. Symbolic-location models refer to locations and objects located by abstract symbols, i.e., names, such as “Room 1910”, “19th floor”, and “National Center of Sciences Building” that are virtual counterpart objects corresponding to the locations and located objects. They provide an ordering relation between locations that reflects spatial inclusion, where the relation is asymmetry, irreflexivity, and transitivity to keep the models simple. For example, a building containing a room cannot simultaneously be contained by that room (asymmetry), i.e., a location cannot be contained by itself (irreflexivity), and a room on a floor of a building must be in the building (transitivity). As a result, the symbols for locations and located objects are organized in a tree structure.

The framework provides each physical entity and place with a visual component, where each component can be contained by at least one other component. Visual components are manually or automatically organized according to the structure of their target virtual counterpart objects and they enable the spatial relationships between the object targets to be visualized, e.g., physical entities and places in the model (Fig. 1). The framework manages location-sensing systems. When it can detect changes in the locations of physical entities and people, it reconfigures its model according to the changes. That is, the framework reflects the containment relationships between physical entities and places in the hierarchical structure of the visual components. Since the first and second parts are constructed based on the MobiDoc framework, we will discuss how they differ from the original framework.

3. Component Framework for Managing Context-Aware Services

The framework consists of three parts: component run-time systems, visual components, and location-information servers. The first can communicate with the model and organize visual components. The second is a counterpart component of a physical entity or place and maintains its visual content and program code that enable content inside it be viewed or edited. The third maintains location-sensing systems and enables changes in the containment relationships of physical entities and places in the hierarchical structure of the virtual components corresponding to the entities and places. Since the first and second parts are constructed based on the MobiDoc framework, we will discuss how they differ from the original framework.

3.1 Component Runtime System

Each component runtime system supports the hierarchical structures of components in addition to their execution (Fig. 2). Components are organized within an acyclic-tree structure, like Unix’s file-directory.

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

When a component contains other components, we call the former a parent and the latter children. When physical entities, spaces, and computing devices move from location to location in the physical world, the model detects their movements through location-sensing systems and changes the containment relationships between components corresponding to moving entities, their sources, and destinations. Each runtime system governs all the components within it and provides them with APIs 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. The framework enables components to migrate between runtime systems through a network by using mobile agent technology. 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. Therefore, the framework intuitively enables a location model to be maintained by distributed computers. Each hierarchy is maintained in the form of a tree structure of component tree nodes of components (Fig. 3). Each node is defined as a subclass of ContextContainer or ContextComponent, where the first supports components, which can contain more than one component inside it 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. Components are migrated as a transformation of the subtree structure of the hierarchy.

3.2 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. We implemented an adapter to reuse existing software, 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 Applets and JavaBeans, because some of these existing components manage their threads and input and output devices deprecatively. Nevertheless, the framework provides adapters for several canonical Applets and JavaBeans to be used as visual components.

This framework provides an editing environment for manipulating the components for network processing, as well as for visual components. It also provides in-place editing services similar to those provided by OpenDoc and OLE. It offers several value-added mechanisms for effec-
resources, such as keyboards, mice, and windows. Each component is tied to its built-in protocols and for coordinating their use of shared resources. Each component tree node can dispatch certain events to its components to notify them when certain actions have occurred within their surroundings. ContextContainer and ContextComponent 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 the boundary of the visible area of a component is clicked, the component is selected and selected control points are designed for moving it around and resizing it (Fig. 4 (a)). Users can customize the selected component, move it to another, save it, and terminate it by dragging its handles (Fig. 4 (b)). We can customize the locations that the services should be available at and the users that the services have been 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 component to visualize and customize the status and attributes of the component’s target, e.g., physical entity, place, computing device, and service via the component, through its built-in protocols or the component’s favorite protocols.

3.3 Management for Location-Sensing Systems

The model can dynamically configure containment relationships among components according to the physical world by using location-sensing systems. Each location information server (LIS) manages location sensors and exchanges information between other LISs in a peer-to-peer manner. The current implementation can support physical entities with RFID, infrared, or ultra-sonic tags attached. After this, we will assume that entities are equipped with RFID-tags. When an LIS detects changes in the position or presence of a tag in the coverage area of the sensor that it manages, it discovers components bound to the tag in the subtree maintained by it using a breadth-first-search (BFS) approach, because such new tags often emanate from one of its neighboring spaces or its surrounding space. If the LIS discovers no components in the first step, it multicasts a query message with the identifier of the tag to other LISs and computing devices that are maintaining their subtrees and it receives reply messages from LISs that know where these components are located. The LIS requests the runtime system to deploy a component corresponding to a person or an object with the tag at a component corresponding to the area. The latter component lays out and displays the visual content of the former within its own estate. The position and size of the former’s visual content is defined by the latter’s layout manager, where the manager can be redefined as an implementation of the java.awt.LayoutManager interface. If LISs manage sensors that can measure geometric locations, they can define one or more virtual spaces within the coverage areas of their sensors and transform the geometric positions of entities within these areas into qualitative information concerning the presence or absence of entities in the spaces.

4. Early Experience

We also developed various components, e.g., a text viewer/editor component and a JPEG, GIF, MPEG viewer component and an audio-player component. 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. Although the current implementation of the framework was not built for performance, we measured the cost of migrating a counterpart component between the visual container components after migrating its visual component is within 30 ms. We believe that this is acceptable for a visual interface for deploying services.

4.1 Monitor and Controller Interfaces for Devices

Exhibition rooms in museums are equipped with various devices, e.g., surveillance cameras, electronic lights, and air-conditioners. A few of them have wired/wireless network interfaces that can be controlled remotely but the others have no interfaces. We need to support the former and the latter interfaces. Since visual components could define program code, we implemented some components for both types.

The first example is a video-stream player component. Since visual components have codes for viewing and modifying content, they allow their content to be presented in arbitrary as well as standard formats. Components can support further application-specific protocols. For example, the video stream player component in Fig. 3 supports a Real-Time Protocol (RTP) to receive a video stream and display the stream on its visual rectangle with a GUI control panel to stop, play, forward, rewind, and pause the stream.

The second example is a remote controller for the power outlets of lights through a commercial protocol called X10. The lights are controlled by switching their power

---

1When a component does not explicitly define any layout manager, it uses the simple layout management policy defined in java.awt.FlowLayout.

11Visual components corresponding to visual components, e.g., documents, image viewers, and text editors, were presented in our previous papers [15], [16].
sources on or off according to the X10-protocol. We provided all lights with their visual components to switch them on or off. Each component communicates with an X10-based server, which controls an X10-module connected to the power outlet to switch the outlet on or off, and each displays its own visual interface to turn the outlet on or off as shown in Fig. 5.

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. As we can see from Fig. 6, the component corresponding to the house contains the components corresponding to the rooms in the containment relationship between these physical spaces and entities. Each of the components corresponding to the room contains more than one component corresponding to the light. We can easily construct such an interface for the controller with the framework. These components represent a map of the arrangement of the rooms in the house. Moreover, they can be used as a controller for some appliances, including lights. If the house has a user-tracking system, when a user enters a room, an LIS for the house detects his or her presence by using the visual component corresponding to him or her, which is displayed within the estate of the component corresponding to the room.

4.2 Context-Aware Visitor Guide System Management

The framework has already been used in a management system for context-aware user-assistant services in public museums, e.g., at the National Science Museum in Tokyo and the Museum of Nature and Human Activities in Hyogo, Japan. The system was constructed as a sensor network where RFID tag readers were connected through a wireless network. These readers were located at specified spots in several exhibition spaces at these museums (Fig. 7). Visitors were provided with active RFID-tags to track their locations. When they came sufficiently close to various objects, e.g., zoological specimens and fossils located at the spots, they could listen to sound content that annotated the objects. The RFID-tag readers identified all the visitors within their range of coverage, i.e., a 2-meter diameter and sent the identifiers of their detecting RFID tags to a service-provider computer through TCP sessions. All service-provider computers had databases for storing audio content and they selected and played content according to their guest’s knowledge and interests when they receive the identifiers of the tags from the readers. Figure 8 shows a screenshot of the visual interface for the management system. The interface enables users to deploy services at areas by using drag-and-drop manipulations. For example, the exhibition had more than 200 daily visitors and the system continued to monitor and manage RFID-tag readers and location-aware services for a week without experiencing any problems (Fig. 9).

The interface consisted of four visual components that monitored four RFID-tag readers located at four spots throughout the exhibition. As we can see from Fig. 8, the window was implemented as a window component that contains six components corresponding to the frame-boxes in it. Four of the box-frame components represented spots and
Experiments at National Science Museum, Tokyo.

Fig. 9 Experiments at National Science Museum, Tokyo.

had programs that communicated with their readers through TCP sessions to monitor the presence of tags within their coverage areas, where RFID readers could work as TCP servers to send the identifiers of these tags. Figure 10 shows the visual interface for the user/location-aware systems used at the National Science Museum. An image-view component drew a map of the exhibition room and it contained six management components for RFID readers connected through a network.

When a visitor with an RFID-tag entered a spot, the component corresponding to him or her was deployed at the component corresponding to the spot. All the components corresponding to the spots had a queue for events issued to their inner components and services; they could explicitly activate the services on a first-come-first-served policy. Therefore, when two visitors were in a spot, their content was played in the order they arrived. We could dynamically add/remove location-aware services to/from the spots. We deployed software to define services at the component corresponding to the spot by dragging-and-dropping the visual component of the software on the visual components corresponding to the places where the services should have been provided. Curators, who may have no knowledge about ubiquitous computing systems, can therefore easily and intuitively change audio-based assistance services at exhibitions.

5. Related Work

Contextual information in the real world need to be detected with sensing systems to provide context-aware services. However, most existing services explicitly and implicitly depend on the underlying sensing systems, e.g., GPS and RFID. They cannot be used with other sensing systems that they have not initially been assumed. To solve these problems, several research projects on context-aware services have attempted to offer general-purpose world models to cancel out differences between sensing systems. Since location is one of the most typical and useful kinds of contextual information, location models will be discussed. Existing location models can be classified into two: the physical and the symbolic worlds [2], [3], [12]. The former represents the position of people and objects as geometric information, which can be measured by GPS and ultrasonic 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 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, i.e., place. An object contained in a volume is reported to be in that place. Existing location models, unfortunately, lack
user-friendly interfaces to enable end-users to easily manage and customize them. Moreover, most existing context-aware service platforms [6], [7], [10] assume the management systems designed for their underlying (non-context-aware) service-discovery mechanisms will be directly used, and they permit users to manage services according to contextual information.

We constructed a symbolic-location model, called M-Spaces [17]. It can maintain the location and deployment of software to define context-aware services and information about the computational resources of the computing devices that can execute these services. It also represents contextual information in the real world like other existing location models. We also constructed a visual interface for the model. Like the framework presented in this paper, the interface consists of visual components [18], [19].

There have been several mechanisms for automatically generating graphical user interfaces (GUIs) for pervasive computing services and devices [8], [11], [13]. Most existing approaches can provide GUIs for individual devices and can support their dynamic generation for devices that may be added later. However, they assume the use of specified protocols to communicate with their target devices. They do not support the deployment or configuration of context-aware services.

6. Conclusion

We presented a component framework for monitoring and managing context-aware services. The framework can dynamically assemble visual components into a visual interface. It enables components to communicate with their target sub-systems and services through the sub-systems' and services' favorite protocols, since these components contain their own program code in addition to the content inside them. Since it provides GUI-based manipulations for editing visual components, end-users can easily add and remove the visual interfaces for system management from their control panels. We designed and implemented a prototype system based on the framework and demonstrated its effectiveness in a management system for user/location-aware assistance systems in museums, e.g., the National Science Museum in Tokyo and the Museum of Nature and Human Activities in Hyogo, Japan.

In conclusion, we would like to identify further issues that need to be resolved. As the framework is general, it has not merely been designed for visual components for monitoring and controlling user-assistant services in museums. We plan to apply it to a variety of context-aware systems. Even though the current implementation relies on Java's security manager, we are interested in additional security mechanisms for these components. The framework maps physical entities or spaces, and services into visual components in the way one-to-one mapping is done. Therefore, when there are numerous physical entities or places, and services in a space, the visual interface for the space tends to become complicated. We need a one-to-many mapping mechanism for abstracting the interface according to users' interests and demands.

References


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. He has also been a professor in Department of Informatics, the Graduate University for Advanced Studies (Soken-dai). His current research interests include, distributed, cloud, and ubiquitous/pervasive computing. He received IPSJ paper award, IPSJ Yamashita SIG research award, and JSSST Takahashi research award. The Young Scientists’ Prize award 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.