A Management Framework for Context-Aware Multimedia Services

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

Abstract

This paper presents a framework for building user interfaces for context-aware service management. It enables us to easily deploy context-aware services at computers through user-friendly manipulations to edit documents and monitor changes and services in the real world. It is constructed as a compound document framework whose the structural composition of visual components can be adapted to changes in the real world. Since components in the framework are programmable entities, they can directly monitor and control their target devices. This paper presents the design for the framework and describes its implementation and several practical applications with it in the real world.

1 Introduction

Ambient intelligence provides highly proactive or interactive environments that use embedded computation to observe and participate in activities that have never previously involved computation. Computers in such environments remain invisible to users and system services are obtained by means of context-awareness. Therefore, rooms, offices, classroom and homes should be provided with their own entities and improve the quality of life of their inhabitants, helping them in their daily tasks. Moreover, services must be adapted to tasks, the environment, its occupants, and available resources.

Many researchers have explored context-aware services as one of the most typical applications of ubiquitous/pervasive computing. However, They have paid scant attention to the management tasks of context-aware services. For example, a context-aware system consists of many heterogeneous computers and sensing devices connected through wired or wireless networks in a house or office. The requirements of applications in such environments tend to depend on their targets, e.g., users, houses, or offices. To support context-aware services, their management systems need to know the context and process this in the real world, e.g., in terms of people, locations, and time. Furthermore, some existing context-aware services contain multimedia content, such as text, images, and video. However, existing context-aware services systems lack professional administrators unlike other network systems. Therefore, end-users themselves are required to customize their own context-aware services environments to their individual requirements and applications.

We propose a component framework to rapidly and easily build graphical user interfaces (GUIs) for management systems in context-aware services. Our framework not only provides GUIs that enable (non-professional) administrators to manage context-aware services and sensing or computing devices in ubiquitous/pervasive computing systems but also provides GUIs for enabling end-users to use the services. The framework itself is constructed as a distributed multimedia system. For example, some context-aware services support video cameras for computer vision or security purposes.

2 Example Scenario

Suppose a context-aware visitor-guide system is installed in a museum. Most visitors to museums lack sufficient knowledge about the exhibits there and they need annotations on these. However, as their knowledge and experiences are varied, they may become puzzled (or bored) if the annotations provided to them are beyond (or beneath) their knowledge or interest. User-aware multimedia annotation services, including text, images, and video, about exhibits are required. For example, when a user stands in front of an exhibit, a multimedia annotation service about the exhibit is provided in his/her personalized form on a stationary terminal close to the exhibit. Museum curators, who have no professional knowledge about context-aware services, deploy and customize context-aware multimedia annotation services and test the services. As exhibitions are often changed in museums, even curators should be able to easily and naturally configure the systems, e.g., the topology of sensor network.
and filter the data measured by these sensors.

3 Design Principles

There have been a few attempts to enable end-users to easily manage their networks. Nevertheless, networks for context-aware services environments in houses and offices are usually administered by end-users, who may have no knowledge or experience with networks. User-friendly interfaces are therefore needed so that end-users can easily manage networks in their context-aware services environments. Several commercial or academic systems for network management have provided visual interfaces for professional administrators rather than end-users. Furthermore, they have explicitly or implicitly assumed that they were being used without any other network management systems. That is, their visual interfaces have not been able to coexist with those of other systems. We need to be able to seamlessly unify visual interfaces for different network management systems.

Although commercial or academic Web-based tools for network management have recently been used, they cannot always monitor and control their target network systems in a real-time manner. This is because they periodically query the target systems and update their visual interfaces displayed within Web browsers executed on client-side computing devices through http-based protocols. Ajax technology may be able to reduce latency between the target systems and the visual interfaces, but most network devices or sub-systems do not support the technology, because they only support simple or particular protocols.

Administrators or users must support heterogeneous network devices, sub-systems, or services that are different. Our framework needs to be independent of any network management systems and open to various network management protocols. Networks in context-aware service environments are evolving in the sense that network devices and services are dynamically being added to and removed from them. Therefore, a user-friendly management system supports the dynamic evolution of network systems or context-aware systems.

4 Compound Document Framework

This section presents a component framework for building and operating the visual interfaces for network management systems by using compound document technology.

4.1 Basic approach

The framework is constructed based on a compound-document framework, called MobiDoc, developed by the author [7]. It enables one document to be composed of various visible parts, such as text, images, and video created by different applications, like other compound-document frameworks, e.g., COM/OLE [1] and Bonobo [4].Compound-document technology is useful for constructing visual interfaces for network management, because it enables end-users to easily and dynamically assemble visual components into a seamless interface.

Like other compound document frameworks, the framework presented in this paper enables components to maintain their own content within them and to be dynamically assembled into one document or component. It also supports GUI-based manipulations enabling it to edit individual components and to layout components on GUI windows or control panels so that end-users can create GUIs for their networks. Unlike other existing frameworks, e.g., COM/OLE, OpenDoc, CommonPoint, and Bonobo, it provides each component with its own program code enabling it to view and edit its content within the component. Therefore, such a component itself can implement network management protocols to communicate with its target network device, sub-system, and service. For example, when a user wants to manage a new network device in his/her networks, he/she drags and drops the visual component that can define the network protocol to monitor and control the device as well as the visual interfaces on his/her control panel.

Although the framework inherits many features of our previous compound document framework, MobiDoc, it has been extended to support network management. It consists of two parts: a visual component and a component runtime system. The former is defined as a collection of Java objects and the latter is executed on the Java virtual machine (VM). Since the Java VM and libraries abstract away differences between underlying systems, e.g., operating systems and hardware, components and runtime systems can be executed on different computers, whose underlying systems may be different.

4.2 Component runtime system

Each runtime system governs all the components within it and provides them with APIs for the components in addition to Java’s classes. It assigns one or more threads to each component and interrupts them before the component terminates, or is saved. Each component can request its current runtime system to terminate and save itself and its inner components in secondary storage. 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 (Fig. 1). When a component is created in a runtime system, it creates a component tree node for the newly created com-
ponent. 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.

A hierarchy is maintained in the form of a tree structure of component tree nodes of the components (Fig. 2). 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 and the second supports components, which cannot contain any components. For example, when a component has two other components inside it, the nodes that contains these two inner components are attached to the node that wraps the container component. Component migration in a tree only occurs as a transformation of the subtree structure of the hierarchy. When a component is moved over a network, on the other hand, the runtime system marshals the node of the component, including the nodes of its children, into a bit-stream and transmits the component and its children and the marshalled component to the destination.

Most Java Swing and AWT GUI Widgets can be used as our components in the framework without having to make any modifications, because they have been derived from the two classes.

The runtime system can invoke specified callback methods defined in components when they are created, relocated, and terminated and it can assign more than one active thread to the components. We can define program codes to communicate with network systems or devices within these callback methods. In fact, we constructed several components for network management through basic protocols. For example, an HTTP server (or client) component plays the role of an HTTP server (or client) to monitor and control network devices as HTTP clients (or servers) and a Telnet component can connect to its target device through a telnet protocol. Since these components are defined as abstract classes, we can define visual interfaces for network management by using Java Swing or AWT Widgets. We also developed various components, e.g., a text viewer/editor component and a JPEG, GIF, and MPEG viewer component and an audio-player component. Note that visual components allow their content to be in arbitrary as well as standard formats, because they have codes for viewing and modifying content. Components can support further application-specific protocols. For example, the Video Stream Player component in Fig. 2 supported a Real-Time Protocol (RTP) to receive a video stream and displayed the stream on its visual rectangle with a GUI control panel to stop, play, forward, back and pause the stream.

4.4 Component manipulation

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 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.

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 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 happen within their surroundings. MDContainer and MDComponent classes support built-in GUIs for manipulating components. For ex-

---

1This is not compatible with all kinds of Applets and JavaBeans, because some of these existing applications manage their threads and input and output devices depreciatively.
ample, when we want to place a component on another component, including a document, we move the former component to the latter through GUI manipulations, e.g., drag-and-drop or cut-and-paste.

4.5 Component persistence

By using mobile agent technology, the runtime system can save or duplicate a component, its children, and information about their containment relationships and visual layouts into a bit-stream and can then later unmarshal the components and information from the bit-stream. When a component is saved or duplicated, its code and state, e.g., instance variables, can be marshalled by using the Java object serialization package. The package does not support the capturing of stack frames of threads. Consequently, our system cannot marshal the execution states of any thread objects. Instead, the runtime system (and the Java VM) propagates certain events to components before and after marshalling and unmarshalling them. The current implementation of our system uses the standard JAR file format for passing components that can support digital signatures, allowing for authentication.

5 Early Experience

We developed various components for monitoring and controlling computing devices, appliances, and sensors, in addition to basic visual components, e.g., text viewer/editor components, JPEG or GIF viewer components, and stream-video player components. Most Java Swing and AWT GUI Widgets can be used as our components in the framework without modifications.

Figure 2. Component Hierarchy

Figure 3. Screenshot of remote control interface.

5.1 Remote controller for home appliances

The first 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 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-base 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. 4. As we can see from Fig. 3, the component corresponding to the house contains the components corresponding to the rooms in the containment relationship between these physical spaces and entities. This system is connected to a smart meter, called TED 5000C, and we provide a component for display the power consumption measured by the meter. Since it treats the amount of power consumption as
context, it can control appliances according to the amount of power they consume.

Figure 4. X10-based power-outlet controlling system

5.2 Management system for context-aware services

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. Visitors were provided with active RFID-tags to track their locations. When they came sufficiently close to various exhibits, e.g., zoological specimens and fossils, located at the spots, they could listen to multimedia content that provided annotation about the exhibits.

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 multimedia content and they selected and played content according to their visitor’s knowledge and interests when they received the identifiers of the tags from the readers. Fig. 5 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 visitors daily and the system continued to monitor and manage RFID-tag readers and location-aware services for a week without experiencing any problems.

The interface consisted of four visual components that monitored four RFID-tag readers located at spots throughout the exhibition consisting of four spots. As we can see from Fig. 5, 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 the spots and 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 such tags. Fig. 6 shows the visual interface for 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. We could dynamically add/remove location-aware services to/from spots. We deployed software to define the service at the component corresponding to a spot by dragging-and-dropping the visual component of the software on the visual components corresponding to the places in which the services should be provided. When more than one user enters a spot, the system selects and plays content according to the combination of people in the spot. The mechanism is presented in another paper [9].
change context-aware multimedia services at exhibitions.

**Figure 6. Monitoring computer for six spots at National Science Museum.**

### 6 Related Work

Several projects have proposed scripting or markup languages for building and configuring GUIs [10]. However, they were intended to define static GUls, whereas our framework supports dynamic GUls in the sense that the structural composition of visual components is modified according to changes in the real world.

Several researchers have explored toolkits for context-aware services [3, 2], but most of them have aimed at building user interfaces on mobile devices, including smartphones. Other projects have focused on building context-aware systems but not on building user interfaces. There have been several mechanisms for automatically generating GUls for controlling devices [6, 5]. Most existing approaches can provide GUls individual devices and can support them, being dynamically generated for devices that may be added. They assume the target devices, which they should provide visual interfaces for, can support their own control protocols for visual interfaces. However, most network devices do not support such protocols. Instead, they can be controlled and monitored through their own favorite protocols, e.g., SMNP, HTTP, and Telnet, in addition to management-specific protocols. Therefore, the framework presented in this paper is required to support various protocols. Network devices and ubiquitous computing devices only have limited resources, such as restricted amounts of CPU power and memory. However, client-side devices, i.e., PCs and PDA, have sufficient resources with their input/output devices, e.g., displays, keyboards, and mice. Therefore, visual interfaces should be managed and executed as much as possible in external systems, including client-side devices, rather than in network devices.

### 7 Conclusion

We presented a component framework for rapidly building and operating visual interfaces for network management. The framework can dynamically assemble visual components into a visual interface. It enables components to communicate with their target network sub-systems through the sub-systems’ 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 network management on their control panels. We designed and implemented a prototype system based on the framework and demonstrated its effectiveness in a network management system for sensor networks in public museums.

### Acknowledgments

This research was supported in part by a grant from the Promotion program for Reducing global Environmental load through ICT innovation (PREDICT) made by the Ministry of Internal Affairs and Communications in Japan.

### References