Network Processing of Documents, for Documents, by Documents

Ichiro Satoh

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

Abstract. This paper presents a content-dependent and configurable framework for the network processing of documents. Like existing compound document frameworks, it enables an enriched document to be dynamically and nestedly composed of software components corresponding to various content, e.g., text, images, and windows. It also enables each component or document to migrate over a network under its own control utilizing mobile agent technology and uses components as carriers or forwarders because it enables them to carry or transmit other components as first class objects to other locations. Since these operations are still document components, they can be dynamically deployed and customized at local or remote computers through GUI manipulations. It therefore allows an end-user to easily and rapidly configure network processing in the same way as if he/she had edited the documents.

1 Introduction

Document manipulation, such as editing, viewing, and distributing documents, is still a crucial role in modern information processing. In distributed computing systems, documents are always transmitted passively over a network by external systems, such as electronic mail systems and http servers. As a result, they cannot determine where, when, or how they should go next. However, there have been several applications whose network processing depends on the content of the documents that are transmitted over the network. For example, tasks in workflow management systems are required to be passed among multiple destinations with specified itineraries. End-users often want to define network processing for documents for them to accomplish their application-specific tasks. However, the customization and management of networking processing is too complex and difficult for end-users.

This paper addresses such a methodology and proposes a new compound document framework, called MobiDoc. Like other existing compound document frameworks, the framework enables an enriched document to be composed of visual components, e.g., text and images. It enables network protocols for documents to be implemented by a set of active documents. By using mobile agent technology, documents or components can define their own itineraries and migrate under their own control, like the programmable-packet approach in active network technology [12]. Furthermore, documents can transmit other documents and multimedia content as first-class objects to their destinations such as with the programmable-node approach in active network technology. The

framework introduces components for network processing such as document-centric components, so that it allows an end-user to easily and rapidly configure network processing in the same way as if he/she had edited the documents.

This paper is organized as follows. We first describe the background and related work (Section 2) and then outline our compound document framework (Section 3). After this, we present component runtime systems for executing and migrating document components (Section 4) and present our component model (Section 5). We also describes its prototype implementation (Section 6) and illustrates several applications of the framework (Section 7). We conclude by providing a summary and discussing future issues (Section 8).

2 Background

Several frameworks for compound document components have been developed, such as COM/OLE [2], OpenDoc [1], and CommonPoint [6]. They enable one document to be composed of various visible parts, such as text, image, and video, created by different applications. However, existing compound documents are inherently designed as passive entities in the sense that they can be transmitted over a network by external network systems such as electronic-mail and workflow-management systems, which cannot determine where they should go next. There have been component-based technologies for distributed computing, such as Enterprise JavaBeans (EJB) [11] and Distributed COM (DCOM). However, our framework has been designed independently of these existing component frameworks, because it requires to treat each component as autonomous, mobile, and document-centric, in the sense that each component can migrate or distribute itself and other components over a network.

Several attempts have been made to support active documents, e.g., Active Mail [4] and HyperNews [5], but these haves aimed at particular application-specific documents, such as electronic mail and newspapers, so that they have not supported editing or exchanging documents with varied and complex content. The fuseONE system [13] composes GUI-based control panels for controlling appliances from active documents, i.e., GUI-based buttons and toggle switches. Like other compound document frameworks, they cannot transmit codes for viewing and editing documents. Placeless Document [3] provides a document management system for active documents. It enables a document to delegate the properties of other components like our component hierarchy, but it is not aimed at customizing the network processing of documents.

We constructed a mobile agent system, called MobileSpaces, which we discussed in a previous paper [7]. We constructed a compound document framework based on the MobileSpaces system [8, 10]. Since the previous framework was inherently designed based on a mobile agent system, there were serious mismatches between mobile agent-based components and the requirements of document components. Moreover, the previous framework could not define or customize any network processing, because it was proposed only as an application of the MobileSpaces system.

3 Approach

The key idea behind the framework was to enable network protocols for documents to be implemented by a set of documents. That is, documents could define their own itineraries, like the programmable packet approach in active networks. Furthermore, documents can transmit other documents as first-class objects to their destinations such as with the programmable node approach.

3.1 Component Model

Like other existing compound-document frameworks, this framework provides document-centric components but enables them to define and manage network processing. It also introduces two notions of components. The first is the notion of a *self-contained* component, where the content of each component and its codes are inseparable even when it is migrated to another computer. Therefore, when a user receives a document, he/she can view or edit it by using its code instead of any applications deployed at its current computer. To our knowledge, no existing software component frameworks, including compound document frameworks, make the code and state of each component indivisible. The second is the notion of *hierarchical* components. Each component can be contained by at most one component and it can dynamically migrate to other components along with all its inner components. It can instruct its inner components to move to other components, marshal, and destroy them, whereas it cannot control its container component. Nevertheless, the former is still a self-contained component so that it can be removed from the latter.

3.2 Configurable Network Processing

This framework provides two approaches for enabling components to customize their own network processing. The first is to make components *mobile* in the sense that they can define their itinerary and travel among multiple computers along the itinerary by using mobile agent technology. The second enables components to define network processing for themselves. The framework also introduces a container component, called *forwarder*, that can treat its inner components as first-class objects and migrate them to other components. Components can also carry or forward other components over a network and visual components can not only contain visual components but also carrier and forwarder components. They can be customized and assembled through GUI manipulations and embedded into a document as visual components. Therefore, end-users can define and customize their application-specific network processing by combing components through GUI manipulations in the same way as if they were editing visual components in documents.

4 Design

This framework consists of two parts: runtime systems and components. It can execute components and migrate them to/from other runtime systems, even when underlying systems, i.e., operating systems and hardware, are heterogeneous, since runtime systems and components are constructed on Java language and executed on Java VM.

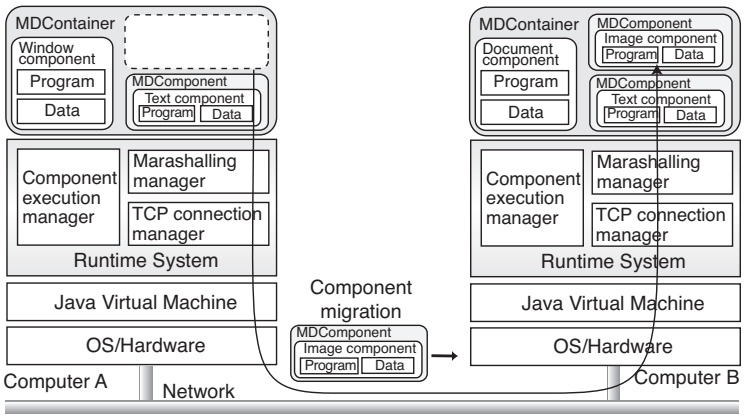

Fig. 1. Component migration between two computers.

4.1 Component Runtime System

Each runtime system governs all the components inside it. It maintains the life-cycle state of each component, e.g., creation, execution, migration, persistence, and termination. It establishes TCP connections with other systems and exchanges control messages and components through the connection. Fig. 1 outlines the basic structure of a runtime system. When a runtime system saves or migrates a component over a network, it marshals the component, the component's inner components, and information about their containment relationships and visual layouts, called component nodes, into a bit-stream and transmits the marshaled component to its destination through an extension of the HTTP protocol. When a runtime system receives components, it unmarshals the components and information from the bit-stream later. The current implementation uses the Java object serialization package for marshaling the states of components. 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 propagates certain events to components before and after marshaling and unmarshaling them. To reduce the size of the bit-stream, the current implementation compresses the bit-stream. If inner components embedded in a component share the same codes, the runtime system can detect and remove such redundant codes from the bit-stream corresponding to the marshaled component, including its inner components.

4.2 Component Hierarchy

As we can see Fig. 3, a hierarchy of components is maintained as a tree structure in which each component can contained by at most one component node. Fig. 2 shows the structure of hierarchical components. Each node is defined as a subclass of two component layout manager classes, `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. The runtime

system basically provides a node derived from the `MDContainer` class for components, except for the visual components that is designed to have no inner components, e.g., text-viewer and sound-player components. For example, when a component has two other components inside it, the nodes that contains the two inner components are attached to the node that wraps the first component. Component migration in a tree is only performed 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 marshaled component to the destination.

4.3 Visual Layout Management

When a component contain components inside it, its `MDContainer` object is responsible for assigning its inner components and their rectangular estates within its estate, and controlling the sizes, positions, offsets, and order of their estates. This framework provides an editing environment for manipulating the components for network processing as well as for compound documents. The environment supports GUIs for manipulating components. It also deals with in-place editing services similar to those provided by OpenDoc [1] and OLE/COM [2].

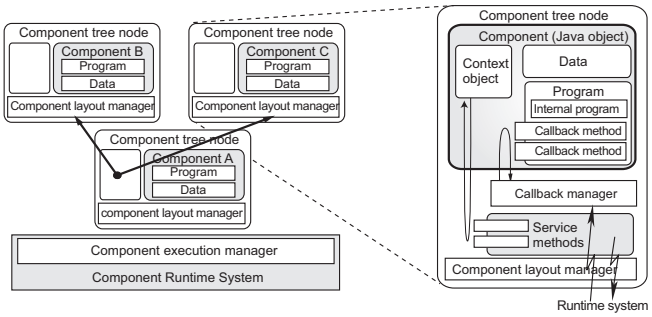

Fig. 2. Component hierarchy and structure of components.

4.4 Components for Network Processing

Each component for network processing is designed to provide its service to its inner components. A component can directly instruct its inner components to move to another location, and can transform them. When a component wants a service, it migrates into one of the components that can provide the service. We present four basic network processing components for other components as outlined in Fig. 4.

– **Forwarding:** A forwarding component can redirect its inner components to other places. When it receives a component, it automatically transfers the visiting component to its specified destination.

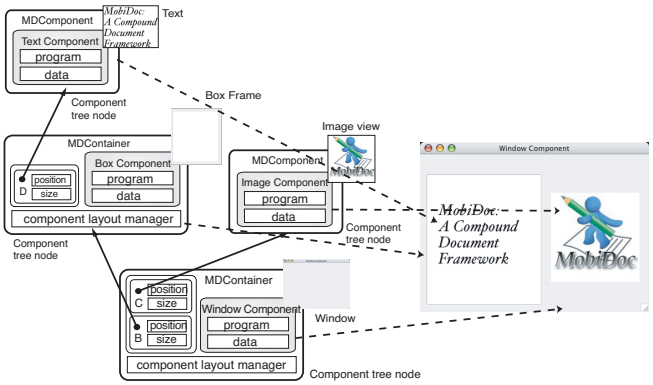

Fig. 3. Component Hierarchy

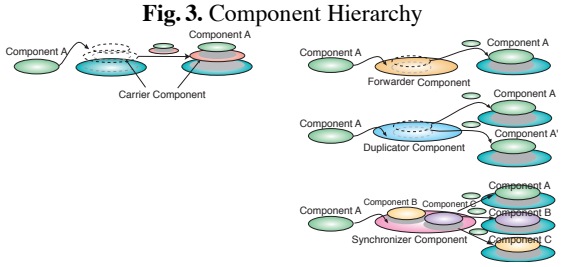

Fig. 4. Basic components for network processing.

- **Duplication:** A duplicator component can create copies of its inner components including all instance variables. When receiving the original components, the cloned components have the same content as the original components.
- **Synchronization:** A synchronizer component can strand its inner components until it can satisfy specified conditions. Within the notion of barrier synchronization, a typical synchronizer component defines a group of moving components. Until it receives all the components within the group, it strands the visiting components inside it.
- **Carrier:** A carrier component can carry its inner components to other places. When it receives a component, it encapsulates the visiting component within it and carries the component to its own destination or the visiting component's destination.
- **Linking:** A reference component is a representative of another component, which may be located at a remote computer. When it receives a component, it fetches its referring component for the visiting component.

The above components have properties that customize their processing and provide support to GUI editors.

4.5 Security

Security is essential in active documents as well as mobile agents, because such documents run their own programs and access resources within the computers they visit.

The current implementation uses the standard JAR file format for passing components that can support digital signatures, allowing for authentication. It also relies on the Java security manager like existing mobile agent systems. To protect components from malicious computers, the runtime system provides an authentication mechanism for component migration borrowed from mobile agent research, so that each runtime system host can only send components to, and only receive from, trusted runtime systems.

4.6 Current Status

We implemented the framework using Java language (JDK1.4 or later version), and we developed various components for compound documents and network processing. Fig. 5 shows a screen-shot of this framework. The left window is a palette of part components and the center and right windows are compound documents contained in the components corresponding to GUI windows. When a user wants to place a component on his/her editing compound document, he/she drags the wanted component from the palette and then drops it on the estate of the document. Since the palette itself is implemented as a container component of part components, it can migrate to another computer and be saved in secondary storage. We can register new components, which may be edited or modified, in the palette through GUI-based data-transfers, e.g., drag-and-drop or copy-and-past operation.

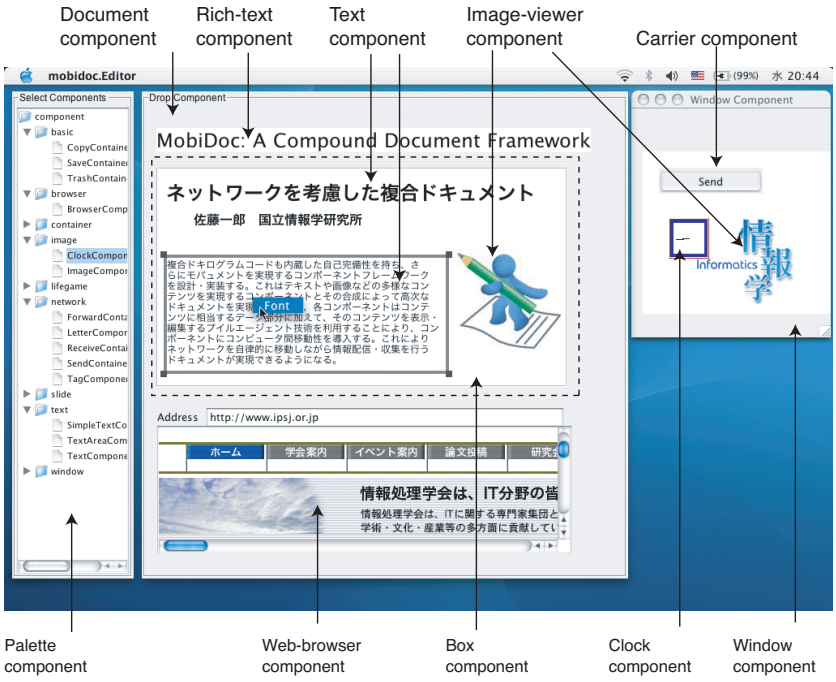

Fig. 5. Examples of compound documents

Even though our implementation was not built for performance, we conducted a basic experiment on component migration with computers (Pentium III 1.2-GHz with Windows XP and SUN JDK 1.4.2). The time for component migration measured from one container to another in the same hierarchy was 10 ms, including the cost of drawing the visible content of the moving component and checking whether the component was permitted to enter the destination component. The cost of component migration measured between two computers connected through a Fast-Ethernet was measured at 64 ms. The cost was the sum of marshaling, compression, opening a TCP connection, transmission, acknowledgment, decompression, security and consistency verification, unmarshaling, visual space layout, and drawing of content. The moving component was a simple text viewer and its size (sum of code and data) was about 9 KB (zip-compressed). The latency of component migration was reasonable for a Java-based visual environment for exchanging compound documents between computers.

5 Experiences

We developed a variety of components based on this framework. This section introduces several components and their uses.

5.1 Compound Document Letter

Most electronic mail systems disallow letters from traveling among multiple destinations along their own itinerary. We developed a legacy decision-making system, called *ringi*, for group decision-making, which has been widely used throughout Japan. When an employee proposes something to his/her company, he/she describes the proposition on a workflow document, called a *ringi-sho*. The document must be handed over to all sections involved with the proposed issue. When the managers of the sections concerned deem the proposal to be adequate, they give it their hanko, or their stamp of approval. Fig. 6 shows a ringi-sho component, which is a carrier component with multiple destinations. It has a destination table whose frames are the areas that its receivers stamp with their own hankos, where each hanko is a component and cannot be removed or modified once it is applied at the frame. The carrier can contain more than one visual component inside it and itineraries between the computers of its receivers until all the receivers stamp their hankos.

5.2 Application-specific Document Distribution

The second example is an editing system for an in-house newsletter. Each newsletter is edited by automatically compiling one or more text parts, which are written by different people as we can see from Fig. 7. A newsletter compound document has one page component, which can contain editor components for visual content, e.g., text and images. When the newsletter is being edited, it forwards the page component to a duplicator component to make as many replicas of the component as the number of writers. The duplicator component then migrates the replicas to forwarder components so that each of the page components is forwarded to a window component on its writer's computer.

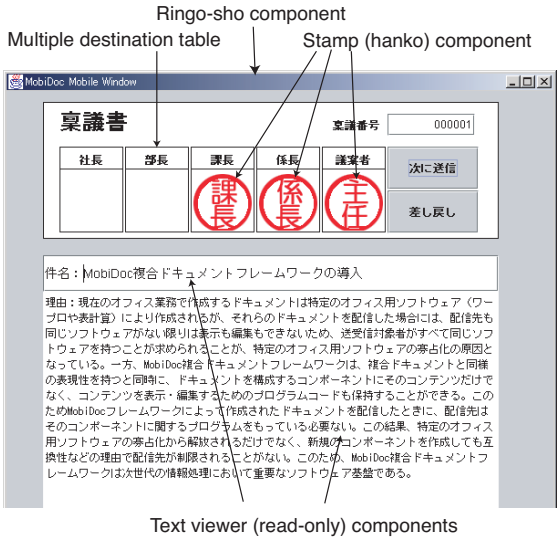

Fig. 6. Ringi-sho compound document.

When it arrives at the destination, it displays a window for its editor program on the screen of the computer to assist the writer. Also, the writer can add his/her visual components to the page component. It goes back to the document after the writer finishes writing his/her text and then the document arranges the arriving components as a bound set. Since the newsletter document, duplicator, and forwarder components are still mobile, they can thus be easily deployed and coordinated according to the requirements of applications.

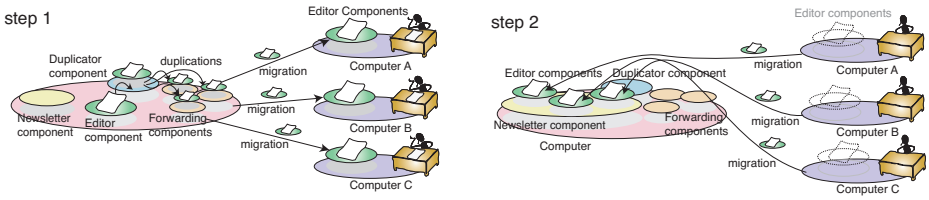

Fig. 7. Newsletter editing system.

6 Conclusion

We presented a framework for network-enabled documents, including hypermedia. It offers five basic network processing operations for documents, i.e., forwarding, duplication, synchronization, carrying, and linking. We can achieve various types of network processing by combining these operations. Since the operations are implemented as

document components, they can be dynamically deployed at remote computers. Moreover, the framework provides a GUI-based editor not only for editing documents but also for easily deploying document components for network processing at remote computers.

To conclude, we would like to point out further issues that need to be resolved. Resource management and security mechanisms in the current system were incorporated in a relatively straightforward way. These should now be designed to incorporate compound documents. When a component migrates to another component or computer, its visual resources, the size of its estate and colors, in the destination may not be the same as those in the source. Although it must adapt its visibility to the resources available in the current location, the current implementation relies on Java's layout manager. The programming interface for the current system is not yet satisfactory. We plan to design a more elegant and flexible interface for programming components. We developed an approach for the development and testing of software running on mobile computers. We are interested in applying the framework to this approach [9]. This is because the framework enables us to easily design and implement active and configurable graphical user interfaces for mobile computers as well as stationary computers.

References

1. Apple Computer Inc. OpenDoc: White Paper, Apple Computer Inc, 1994.
2. K. Brockschmidt, Inside OLE 2, Microsoft Press, 1995.
3. P. Dourish et al, A Programming Model for Active Documents, Proceedings of 13th Symposium on User Interface Software and Technology (UIST'2000), pp.41-50, ACM Press, 2000.
4. Y. Goldberg, M. Safran, and E. Shapiro, Active Mail - A Framework for Implementing Groupware, Proceedings of ACM CSCW'92, pp. 75-83, ACM Press, 1992.
5. J. Morin, HyperNews, a Hypermedia Electronic-Newspaper Environment based on Agents, Proceedings of HICSS-31, pp.58-67, 1998.
6. M. Potel and S. Cotter Inside Taligent Technology, Addison-Wesley, 1995.
7. I. Satoh, MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, Proceedings of International Conference on Distributed Computing Systems (ICDCS'2000), pp.161-168, IEEE Computer Society, April 2000.
8. I. Satoh, MobiDoc: A Mobile Agent-based Framework for Compound Documents, Informatica, vol.25, no.4, pp.493-500, December 2001.
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, A Compound Document Framework for Multimedia Networking, Proceedings of 1st International Conference on Distributed Frameworks for Multimedia Applications (DFMA'2005), pp.80-87, IEEE Computer Society, February 2004.
11. Sun Microsystems, Inc., Enterprise JavaBeans Technology (EJB) http://java.sun.com/products/ejb, 2002.
12. D. L. Tennenhouse et al., A Survey of Active Network Research, IEEE Communication Magazine, vol. 35, no. 1, 1997.
13. P. Werle, F. Kilander, M. Jonsson P. Lonqvist, C. G. Jansson, A Ubiquitous Service Environment with Active Documents for Teamwork Support, Proceedings of 3rd International Conference on Ubiquitous Computing (UBICOMP'2001), Lecture Notes in Computer Science, vol. 2201 pp.139-155, Springer, 2001.