分散共有アクティブドキュメントの実現方式

3J－6

羽根 秀宜, 河村 元夫, 横田 実
NEC C&C研究所

1. はじめに

WWWによる情報の発信と共有が盛んになってきた。一方、複合ドキュメント技術もネットワークで利用する枠組みがいくつか発表されており、複雑なドキュメントを共有できるようになってきている。しかしこれらの技術は、ドキュメントのリアルタイムな更新ができなかったり個々のユーザ間の共同作業を考慮していない。

これらの問題を解決する、分散共有アクティブドキュメントの実現方式について提案する。本稿では、まずアクティブドキュメントの分散共有形態とその実現方式について述べる。次にJavaを用いた実現方法について述べ、最後に既存技術との比較検討を行う。

2.提案する実現方式

2.1 目標とする分散共有形態

様々なメディアを含み、時々刻々とその内容が変化するドキュメントを、ここではアクティブドキュメントと呼ぶ。アクティブドキュメントの共有では、静的なドキュメントの共有とは異なり、その内容の変化が共有している他のユーザにも即座に反映されるような共有メカニズムが必要である。分散プレゼンテーションやドキュメントの共同執筆はアクティブドキュメントの一例と見なせる。

このアクティブドキュメントの共有形態には次に示す形態が必要であると考える。

同じViewを持つ共有: 分散プレゼンテーションのように、共有しているユーザが皆同じページやポインタを見る形態。

異なるViewを持つ共有: ドキュメントの共同執筆のように、各ユーザが独自のページやカーソル位置で編集を行う形態。

これらの形態は実行時に選択/変更できるべきである。

2.2 Shaped Object Modelによる実現

前節で述べたような、様々なメディアを含むアクティブドキュメントを実現するために、ドキュメントはひとまとまりの機能を持つコンポーネントの階層的な複合構造により構成する。そこでドキュメントの分散共有を実現するために、コンポーネント自体を分散共有できる構造にする。

筆者らが提案したShaped Object Model[2]では、コンポーネントを描画/イベント処理などGUIに関する機能を担当するShapeと、Shape以外のコンポーネントの本質的な機能を担当するBodyとに分割し、サーバ側にBodyを、クライアント(ユーザ)側にShapeを配置することで、コンポーネントの分散共有を提案した[3]。

本稿ではこのモデルを拡張し、Shapeのコンテキストとして機能するViewContextを導入する(図1)。このモデルではShapeはViewContextが示すコンテキストにおいてBodyの内容を描画する。これにより各コンポーネントの性質に応じて適切なコンテキストをViewContextとして定義することで、ViewContextを共有すると同じViewを持つ共有を、ViewContextを共有しないと異なるViewを持つ共有を実現することができる。

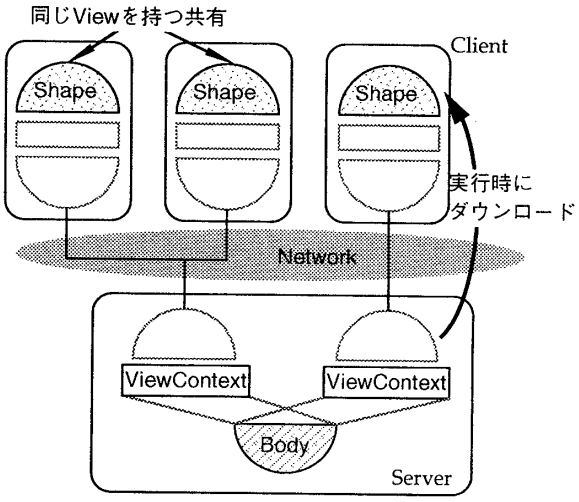

図1:ViewContextの導入による拡張Shaped Object Model

このモデルの特徴はクライアントからコンポーネントをアクセスすると、そのShapeのみがクライアントにダウンロードされ、サーバのBody、ViewContextと通信しながら動作することである。これにより、予めクライアントに各コンポーネントに応じたShapeを用意しておく必要がなく、ドキュメントのポータビリティが向上する。

また、コンポーネントの組み合わせによって構成されるアクティブドキュメントは、Body、ViewContext、Shapeを階層的に組み合わせることにより実現される(図2)。

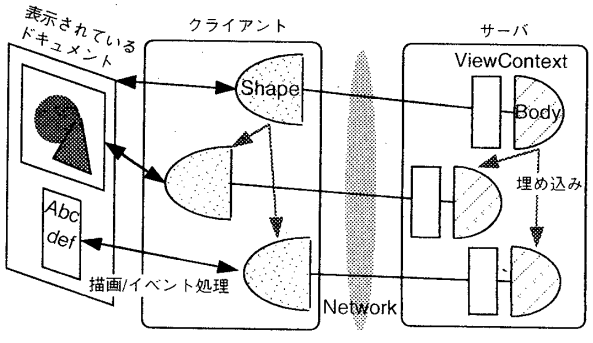

図2: 複合コンポーネントによるアクティブドキュメント

An Implementation of Distributed and Shared Active Documents
Hidetaka Hane, Motoo Kawamura, Minoru Yokota
C&C Research Laboratories, NEC Corporation

3. Javaを用いた分散共有アクティブドキュメントの実現

前節で述べた拡張Shaped Object Modelによるアクティブドキュメントを Javaを用いて実現した。Javaを用いた理由は、Shapeの実行時ダウンロードの実現が容易であり、分散オブジェクトを実現するための枠組みも存在するからである。分散オブジェクトを実現するためにHORB[4]を用いた。実現方式の概要は次の通りである。

1. BodyとShapeの基本となるクラスを実現した。このクラスは後述するBody-ViewContext-Shape間の処理を実現するために必要なメソッドを持つ。
2. ViewContextはShapeのコンテキスト管理とBodyの共有管理を行うクラスとして実現した。後者によりBody側で共有管理を行う必要が無くなり、Bodyの実装が容易になる。
3. サーバ側にはBodyやViewContextを管理するActive-DocumentServerクラス、クライアント側にはそれらをアクセスするためのShellクラスを実現した。

次に、Shapeのダウンロードと Body-ViewContext-Shapeの処理の流れを概説する。

3.1 Shapeのダウンロード

図3にダウンロード処理の流れを示す。

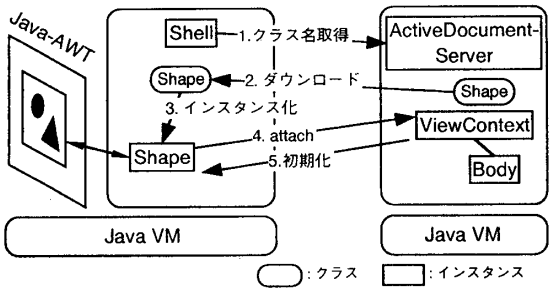

図3: Shapeのダウンロード

ShellはActiveDocumentServerを通じてViewContextにアクセスしShapeのクラス名を取得する。次にそのクラスのバイトコードをJavaのクラスダウンロード機構を用いてダウンロードしShapeのインスタンスを生成する。更にShapeをViewContextにattachすることでViewContextは新たなShapeが付加されたことを知り、そのShapeに初期化を指示する。以降、Bodyやコンテキストの変更はView-Contextを介してShapeに通知される。次節にその処理の流れを概説する。

3.2 Body-ViewContext-Shape間の処理の流れ

図4にBody-ViewContext-Shape間の処理の流れを示す。ShapeはJavaのAWT(Abstract Window Toolkit)からのイベントを解釈し、Bodyの必要な処理をリモートに呼び出す。次に、Bodyは変更があるとそれをViewContextに通知する。ViewContextではその通知に従い、Shapeの更新メソッドを呼び出す。更にShapeはその呼び出しに従い描画を更新する。

なお、HORBはリモートメソッドを同期/非同期のどちらでも呼び出せるが、Shapeの更新メソッドは同期呼び出しを用いた。これはプログラミングの容易性とShapeの確実な更新を行うためである。ただし、更新メソッドのブロッキングの影響を避けるためにViewContextではattachしているShapeそれぞれについて更新用のスレッドが必要となる。この方法はスレッドを消費するので非同期呼び出しによる実現方式も検討している。

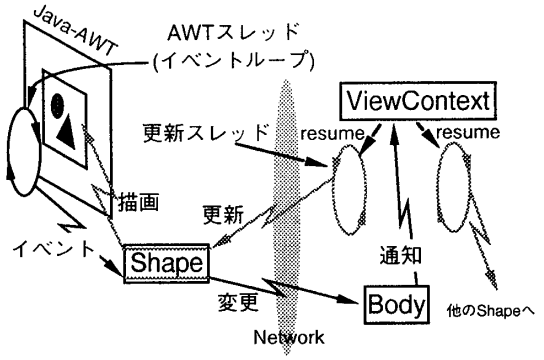

図4: Body-ViewContext-Shape間の処理の流れ

なおShapeの描画は更新スレッドからの呼び出しのみに限定しておらず、例えば周期的に描画を行うShapeの実装では周期描画用の独自のスレッドをShapeの内部に用意する方法が考えられる。

4 既存技術との比較

WWWは基本的に閲覧しか行えないが、例えばHTMLのフォーム機能とCGIを用いて簡単な共有掲示板を作成することができる。このような方法は既存のWWWドキュメントと相互運用が可能であるが、ドキュメントの変更を通知する手段が無いなど、ここで目標とするようなアクティブなドキュメントを実現することができない。

複合ドキュメント技術は現在様々な分散化が試みられている。基本的には分散オブジェクト技術を用いてコンポーネントのリモートアクセスを可能にすることで、ドキュメントの分散化を行う。しかし共有を考慮していない既存のコンポーネントを分散化するだけでは、柔軟な共有を実現することができないと考えらえれる。

本方式は、GUIとそのコンテキストをコンポーネント本体から分離して、分散共有を基本的な枠組みとして持つことで、柔軟なドキュメントの共有を実現している。

5 まとめ

ViewContextを導入した拡張Shaped Object Modelを用いて柔軟な共有が可能なアクティブドキュメントを実現する方式と、それをJavaで実現する手法について述べた。今後は様々なコンポーネントを実装し本方式の評価を行うと共に、文書校正支援やスケジュール管理などよりCSCW的なコンポーネントの実現を検討する。

参考文献

[1] Sun Microsystems, Inc., "Java: Programing for the Internet", http://www.javasoft.com
[2] 横田, "Shaped Objectによる情報の分散共有", システムソフトウェアとオペレーティングシステム, 71-13, 1995
[3] 羽根,河村,横田, "情報端末のための複合ドキュメント分散共有方式", 第52回情報処理学会全国大会, 2F-5
[4] 平野, "HORB: The Magic Carpet for Network Computing", http://ring.etl.go.jp/openlab.horb/