MobiDoc：ネットワークを考慮した複合ドキュメントフレームワーク

佐藤一郎†

複合ドキュメントを実現するコンポーネントフレームワークを設計・実装する．これはテキストや画像などの多様なコンテンツを実現するコンポーネントとその合成によって高次なドキュメントを実現するものである．従来の複合ドキュメントと同様にコンポーネントの合成やシームレスなコンテンツ表示・編集能力を提供するとともに，このフレームワークでは各コンポーネントがコンテンツに相当するデータ部分に加えて，そのコンテンツを表示・編集するプログラム部分も内蔵できるという自己完備性を導入する．さらにモバイルエージェント技術を利用することにより，コンポーネントにコンピュータ間移動性を与える．これによりネットワークを自律的に移動しながら情報配信を行うドキュメントが実現できるようになる．本論文ではこのフレームワークの概要を述べるとともに，Java 言語を用いた設計・実装について概説し，さらに応用事例を示す．

MobiDoc: A Framework for Network-enabled Compound Documents

Ichiro Satoh†

This paper presents a new framework for building mobile compound documents in distributed system, where a compound document to be dynamically and nestedly composed of software components corresponding to various contents, e.g., text and image. The framework enables each component to migrate over a network under its own control by using mobile agent technology and be are self-contained in the sense that they include not only their contents but also their programs for viewing and editing the contents. It also provides several value-added mechanisms for visually manipulating components embedded in a compound document and for seamlessly combining multiple visible components into a single one. Therefore, we can easily create and operate autonomous documents, which can change their contents and distribute themselves over a distributed system. This paper describes this framework and its implementation, currently using Java as the implementation language as well as a component development language, and then illustrates several interesting applications to demonstrate the utility and flexibility of this framework.

1. はじめに

分散システムを含む現在の情報システムにおいてドキュメントの編集，配布，保存は依然として中心的な処理となっている．そこで本論文ではドキュメントに特化した分散システムの実現方法を，複合ドキュメントと呼ばれるドキュメントを対象としたソフトウェアコンポーネント技術を発展させることにより実現していく．複合ドキュメントはテキストやイメージなどの個々のコンテンツに対応したコンポーネントを合成することにより，高次のドキュメントを実現する技術であり，たとえば COM/OLE [3] や，OpenDoc [1]，CommonPoint [9] などがその代表例として知られている．しかし，これらの既存システムはネットワークを前提にしたものではなく，後述するようドキュメントのネットワーク配信では配信先コンピュータにドキュメントを構成するすべてのコンポーネントの表示（または編集）プログラムが必要となり，配信先およびコンテンツ種類を限定することが多かった．また，その配信過程においてドキュメントは受動的な転送対象にすぎず，そのコンテンツに応じて配信先を変えるなどのドキュメント中心の分散処理からはほど遠い．

本論文ではドキュメントを中心とした分散処理を目標に，複合ドキュメントの有用性を保持しながら，ネットワーク処理を考慮した新しい複合ドキュメントフレームワークを提案していく．これの特徴は，(1) ドキュメントを自己完備なデータおよびプログラム群として構成・配信することにより，アプリケーションを不要にすることと，(2) ドキュメントがそれのコン

† 国立情報学研究所ソフトウェア研究系
Software Research Division, National Institute of Informatics

テンツやプログラムにより配信先を制御できるようにし，ドキュメント主体の分散処理を実現可能にすることである．

ここで本論文の構成を示す．続く 2 章では研究背景と複合ドキュメントフレームワークへの要求を述べる．そして，3 章ではフレームワーク（Mobidoc と呼ぶ）の設計と実装を説明し，4 章では応用事例を与える．そして，5 章では有用性を議論し，6 章では関連研究をまとめる．7 章では将来の研究課題を，最後の 8 章では結論を述べる．

2. 研究背景と目的

ここでは既存の複合ドキュメントとその問題点を議論し，その後で研究方針を述べる．

2.1 複合ドキュメントフレームワーク

前述のように COM/OLE [3]，OpenDoc [1]，CommonPoint [9] などの複合ドキュメント技術はテキスト，イメージなどの多様なコンテンツに対応したコンポーネントを動的に合成することにより高次なドキュメントを構成するものであり，具体的には複合ドキュメントフレームワークと呼ばれるソフトウェア群から構成され，次のような機能を提供する．

コンポーネント構造化： テキストやイメージ，スプレッドシートなどのコンテンツの種類に応じたコンポーネントを合成・埋め込むことにより，それらのコンテンツからなる 1 つのドキュメントを構成する．また，2 次記憶上に保持される際もそのコンポーネント間構造を保持する．

インプレース表示・編集： 各コンポーネントはそれを埋め込むドキュメント上でコンテンツを配置・表示し，さらにマウスクリックなどの操作により，コンテンツを編集するプログラムを呼び出すことにより編集できるようにする．

コンピュータ内のコンポーネント転送： ドラッグアンドドロップなどの GUI 操作により，ドキュメント内にコンポーネントを生成･削除することや，あるドキュメントから別のドキュメントにコンポーネントまたはその複製を埋め込めむことができる．

これらの機能を組み合わせることにより，多様なコンテンツを含むドキュメントが容易に実現できることから注目を集めいていたが，必ずしも広範に普及しているとはいえない．特に，複合ドキュメントのネットワーク処理には多くの問題が残されている．

2.2 既存の複合ドキュメントフレームワークの問題

既存の複合ドキュメントにおけるコンポーネントは，そのコンテンツ部分とそれを表示（および編集）するためのプログラム部分に分けられ，ドキュメントの保存や配信の対象になるのはコンテンツ部分に限られる．この結果，複合ドキュメントを受け取っても，そのドキュメントを構成するすべてのコンポーネントの表示（および編集）プログラムを保持していない限り，そのコンテンツの表示（および編集）ができないことになり，ドキュメント配布先が制限されたり，配布先コンピュータ上のプログラムによりコンテンツ種類が制限されたりしていた☆．また，既存の複合ドキュメントは，そのネットワーク配信処理では受動的に転送される．つまり，複合ドキュメントをネットワーク上で交換するには，電子メール送受信ソフトウェアや Web サーバに代表される共有ファイルシステムなどの外部通信システムを通じて実現する必要があった．この結果，配布先や配布タイミングは外部システムにより制御がされ，例えばドキュメントのコンテンツに応じて配布先や配布経路を変更することはできない．

2.3 目　　的

本論文の目的は，既存の複合ドキュメントの有用性を保持しながら，ネットワーク処理に対応できる新しい複合ドキュメントフレームワークを設計・実装することであり，そのフレームワークには次のような機能が要求される．

移動性： ドキュメントおよびそれを構成するコンポーネントはコンピュータ間で移動可能とする．この際，それらは移動先コンピュータにおいて移動前のコンテンツ内容から表示・編集が継続できるようにする．

自律性： ドキュメントおよびコンポーネントはユーザ指示や外部通信システムだけでなく，それ自身のコンテンツおよびプログラムにより移動先や移動経路を決定できる．

自己完備性： コンポーネント実行システムが提供されていれば，移動先に個々のドキュメントまたはコンポーネント固有の情報やプログラムコードをあらかじめ用意する必要がないようにする．

保存性： ドキュメントおよびコンポーネントはその構造やコンテンツを保持したまま保存できるとともに，ネットワークの接続有無にかかわらず再び表示・編集できるようにする．

補　　足

このフレームワークの有用性は複合ドキュメントの枠組みにとどまるものではない．たとえばオフィスア

☆ プログラム部分のオンデマンドで配信することにより対応できるが，既存の複合ドキュメントフレームワーク自体はネットワーク処理に対応していないことが多い．

プリケーション（例：ワープロや表計算の作成ソフトウェア）市場の寡占化を解消するためにも重要となる．プレインテキストやJPEGイメージのようにデータ形式が規格・標準化されているものを除くと，そのデータを作成したアプリケーション以外では正しく表示・編集できないことが多い．一方で，インターネットをはじめとしてネットワークの普及とともに，特定のアプリケーションで作成・編集されたデータを電子メールやWebを介してコンピュータ間で交換する機会が多くなり，そうしたデータの表示および編集を円滑化する目的から，特定オフィスアプリケーションに市場が集約化する結果を生んだ．一方，このフレームワークではコンポーネント内にそれのコンテンツを表示・編集するプログラムを保持するため，ドキュメント配信において受け手側はドキュメントに対応したアプリケーション（プログラム）をインストールしておく必要がなく，また特定のアプリケーションの有無により配布範囲が制限されることもない．このほか，各コンポーネントはそれ自身でコンテンツを表示・編集できるため，新しい種類・フォーマットのコンテンツであっても自由に扱えることとなり，アプリケーションの表示・編集能力によってコンテンツの多様化が阻害されることもない．

3. 設計・実装

ここでは前章において議論した要求を満足する新しい複合ドキュメントフレームワークを設計・実装する．このフレームワークは各コンピュータで稼働するコンポーネント実行システムとコンポーネントに対する補助クラスから構成される．現実装はJava言語を利用している現在の実装では，Java言語（J2SE ver.1.2以上）によりその実行システムとコンポーネントが記述され，同言語の仮想機械上で実行される☆．なお，コンポーネントの移動・実行先となるコンピュータのOSやハードウェアの違いはその仮想機械によって吸収されることから，異なるOSやハードウェアを持つコンピュータ間であってもコンポーネント移動・実行が可能となっている．

3.1 方　　針

このフレームワークは既存の複合ドキュメントフレームワークが提供する機能に加えて，ネットワークを考慮した次のような機能を提供する．

コンポーネント

各コンポーネントはテキストやイメージなどのコンテンツや，ウィンドウやタブなどのGUI単位に対応する計算実体であり，データとプログラムの組から構成される．ここでプログラムは配置編集操作や実行状態の変化に応じて受け取るイベントに対するコールバックメソッドの集合体として定義される．また，コンポーネントはアイコンイメージを持つことができる．

コンポーネント階層

OpenDocやTaligentなどと同様に，ドキュメントもコンポーネントの1つとし，各コンポーネントは0個以上のコンポーネントを含有できるとする．このとき含有されるコンポーネントはそれを含有するコンポーネント上でシームレスに表示・編集ができるようにする．なお，各コンポーネントはそれを含有するコンポーネントによって割り当てられた矩形の表示領域を持ち，その中でコンテンツの表示・編集を行うとする．

コンポーネントの自律的移動

コンポーネントはそれの含有先を変更できる．つまり，コンポーネントから他のコンポーネントに移動することができる．その際に，GUI操作などを用いたユーザによる明示的なコンポーネント移動のほかに，コンポーネントはそれのデータおよびプログラムに従って移動先を決めることができる．これによりたとえばコンテンツに応じて移動先を変更したり，複数の移動先をまわったりすることもできる．

コンピュータ間移動

コンポーネントの移動先は同一コンピュータ内のコンポーネントだけでなく，遠隔コンピュータ上のコンポーネントでもよいとする．なお，コンポーネントのコンピュータ間移動では，モバイルエージェントの実装技術を利用することにより，コンテンツに相当する実行状態（たとえばヒープ領域上のインスタンス変数）とプログラムコード（Javaクラス）とともに移動先コンピュータに転送される．

3.2 アーキテクチャ

次に複合ドキュメントフレームワークの設計方針を述べる．コンポーネント移動性の導入は複合ドキュメントフレームワークのアーキテクチャに大きな影響を与える．既存のフレームワーク[1),3),9)]では**図1** (a)に示されるように，コンポーネントのコンテンツに相当するデータ部分，そのコンテンツを表示・編集するプログラム部分（アプリケーションプログラム），そしてコンポーネントの配置・実行を管理するコンポーネント実行システムの3つから構成される．一方，このフレームワークは図1 (b)のように，コンポーネント実行システムはコンポーネントのロードや転送など

☆ フレームワークそのものはJava言語を前提としていない．

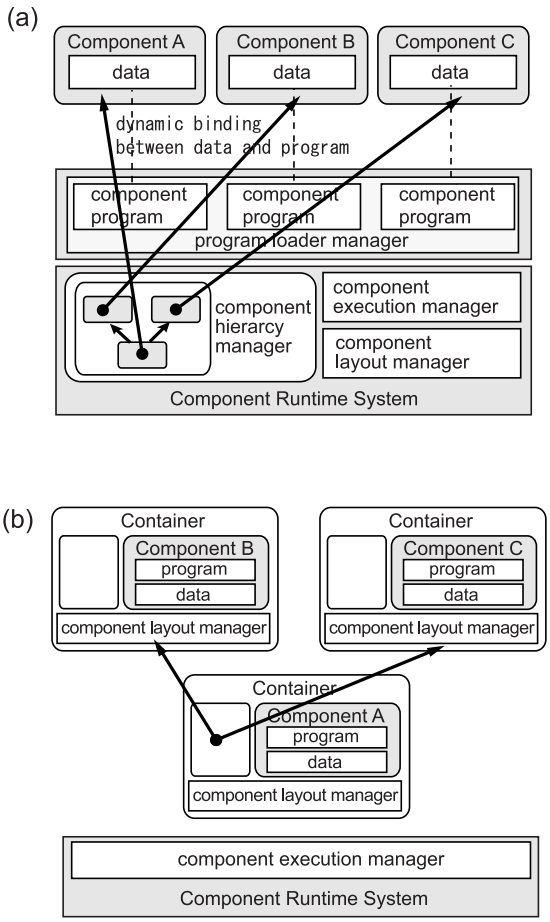

図 1 複合ドキュメントフレームワークのアーキテクチャの比較
Fig. 1 Comparison between compound document framework architectures.

の実行・移動制御を行うが，配置管理は行わない．そして，前述のようにコンポーネントのコンテンツに相当するデータ部分とコンテンツ表示プログラムを不可分とするとともに，コンポーネント配置管理プログラム（コンポーネントコンテナと呼ぶ）もコンポーネントと一体化する．これは下記の理由から，コンポーネントの移動時にそれのコンテンツの表示・編集プログラムおよびそれが含有するコンポーネントの配置プログラムもコンポーネントとともにデータ列に整列化（marshalling）して，移動先コンピュータに転送する必要があるためである．

- 既存の複合ドキュメントシステムはコンポーネントおよびドキュメントのネットワーク配信には対応していない．このため，ドキュメントまたはドキュメント中のコンポーネントをネットワーク配信する場合には，そのドキュメントまたはコンポーネントをあらかじめファイルとして保存した後に，メール送受信システムなどの外部通信システムを利用して送信することになる．この結果，ネットワーク配信時にはドキュメントまたはコンポーネントを表示・配置するプログラムは終了していることになる．一方，このフレームワークではコンポーネントおよびドキュメントはそれらのプログラムにより配信先を決定し，実行システムに送受信を依頼することになるため，その配信時にドキュメントまたはコンポーネントの表示や編集するプログラムが終了しているとは限らない．したがって，それらのプログラムも整列化対象に含める必要がある．
- 既存の複合ドキュメントフレームワークでは実行システムによりコンポーネントの配置を管理していることから，ドキュメントの種類によらず配置ポリシは固定されるという問題があった．一方，このフレームワークではコンポーネント配置プログラムを実行システム側ではなく，コンポーネント側で管理することにより，それが含有するコンポーネントの配置ポリシを定義・変更できることになり，複合ドキュメントの多様化に貢献する．

なお，この複合ドキュメントフレームワークのプロトタイプに相当する同名システム[11),12),17)] は既存の複合ドキュメントフレームワークと同様に実行システム側によりコンポーネントの配置管理を行っており，配置編集中に移動する場合にはその配置情報の一部が欠ける可能性があり，またコンポーネント配置ポリシを変更することはできなかった．

3.3 コンポーネント実行システム

コンポーネントの実行環境を提供するとともに，コンポーネントのコンピュータ間移動を実現するソフトウェアである．なお，その機能と実現方法の多くはモバイルエージェントの実行システムと同様になることから，以下では複合ドキュメントに関わる部分を中心に説明していく．

3.3.1 コンポーネントの実行管理

コンポーネントは生成，実行，移動，永続化，停止の 5 つのライフサイクル状態を持つが，その状態は実行システムにより制御・管理される．また，コンポーネントが所定のイベントリスナーオブジェクトを登録しているときは状態変化の直前または直後に通知を受けることができる．たとえば現在の実装ではコンポーネントの生成の直後，移動の直前と直後，永続化の直前と直後，停止の直前にイベントをコンポーネントに送る．なお，コンポーネントは 1 つ以上の実行スレッドを生成・保持することができるが，スレッドは各コン

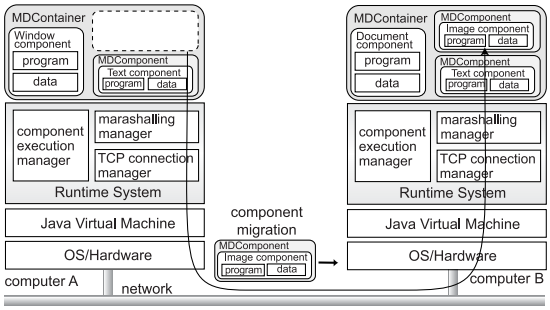

図 2 2 つのコンピュータ間のコンポーネント移動
Fig. 2 Component migration between two computers.

ポーネントの含有関係に応じて Java 言語のスレッドグループが割り当てられており，実行システムはコンポーネントの移動や永続化，停止時にはそのコンポーネントとそれが含有するコンポーネントに割り当てられたスレッドを停止するため，そのスレッドグループに停止割込みを送る．また，多くの Java 言語によるモバイルエージェントシステムと同様に Java 言語のセキュリティマネジャの利用および利用可能クラスファイルの制限を行うことで，到着したコンポーネントによるファイルシステムやネットワークへのアクセスを制限することができる．

3.3.2 コンポーネント移動の実現

図 2 のように実行システムはコンポーネントからコンピュータ間移動または永続化，複製のコマンドを受け取ると，上述のライフサイクル状態変化に対応したイベントをコンポーネントに送信し，そのコンポーネントとそのコンポーネントに含有するコンポーネントをそれぞれのコンテナとともにデータ列に整列化する．なお，この整列化では Java 言語の直列化（Serialization）機構を利用して，コンポーネントを構成するオブジェクトのインスタンス変数をデータ化し，さらにコンポーネントを定義するクラスファイル群を署名付き JAR ファイル形式に圧縮して，1 つのアーカイブデータとする．この際，コンポーネントとそれに含有されるコンポーネントで同一のクラスファイルを含む場合は，重複するクラスファイルをアーカイブ対象から外すことができる．

- そして，移動コマンドの場合は宛先となるコンピュータに TCP ソケットを通じて送信し☆，一方，そのデータを受信した実行システムは非圧縮および逆直列化処理を行って，再びコンポーネントに変換して実行を再開させる．なお，クラスファイルを含めコンポーネントの実行に必要な情報を一括して移動先に転送しているので，移動後はネットワークは切断してもよい．
- 永続化コマンドの場合は整列化したコンポーネントを 2 次記憶システム上の JAR 形式ファイルとして圧縮・保存する．また，必要に応じてそのファイルを読み込んで再びコンポーネントとして実行する．なお，このファイルはコンポーネントに関する情報を保持しているので，フロッピーディスクなどのリムーバブルメディアに保持した場合，そのメディアを他のコンピュータで読み込んでも，その内部コンポーネントを含めてコンポーネントに変換できることになる．
- 複製コマンドの場合は，Java 言語にはオブジェクトのディープコピー（deep copy）機能がないことから，コンポーネントを整列化したデータおよびクラスを複製して，それをコンポーネントに変換する方法を用いている．なお，複製におけるオリジナルと複製コンポーネントは独立したものとして扱うことから，両コンポーネント間においてコンテンツの一貫性制御は行わない．

3.3.3 コンポーネントの名前解決

セキュリティ上の理由から，各コンピュータのコンポーネント実行システムはあらかじめ認証・登録されたコンピュータからのコンポーネントだけを受信する．一方，コンポーネントで他のコンピュータからのコンポーネントを含有できるものは生成時や到着時にそれの名前とともに受入れ許可を実行システムに通知し，その実行システムは受入れ許可を行ったコンポーネントの識別子と名前，アイコンイメージを送信元コンピュータとして認証されている実行システムに定期的に通知する．一方，名前とアイコンイメージを受け取った実行システムは，その識別子とともにアイコンイメージおよび名前を表示するアイコン表示コンポーネントを生成する．このコンポーネントは受信先コンポーネントへの宛先アドレスとして GUI 操作が可能であり，たとえばこれをドキュメント上の宛先入力箇所にドラッグアンドドロップすることにより，受信先コンポーネントへの移動用アドレスを入力することができる．なお，アイコンに対応づけられた宛先アドレスは実行システム間の情報交換により定期的に更新されるが，その更新間隔中に宛先アドレスが本来参照するコンポーネントが他のコンピュータに移動している可能性がある．そこで，MobileSpaces システムが提供する転送エージェント[13] と同様に移動先への転送機能だけを持つダミーコンポーネントを導入して移動

☆ ファイアウォールなどの通信制限に対応するため，MobileSpaces[10] と同様にコンポーネント送受信は HTTP や SMTP にトンネリングする機構も提供している．

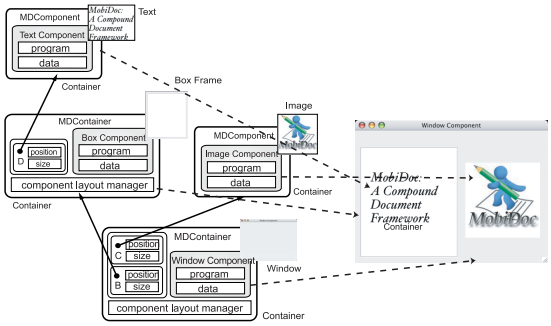

図 3 コンポーネント階層
Fig. 3 Component hierarchy.

透過性を実現する．なお，このダミーコンポーネントは他のコンポーネントと同様にプログラムを内蔵しており，更新間隔以上の時間が経過すると自動的に消去することもできる．

3.4 コンポーネントコンテナ

コンポーネントは EJB などの最近の分散オブジェクト技術と同様にコンポーネント本体部分とインタフェースを実現するコンテナ部分から構成される．そして，このフレームワークにおいてコンテナ部分に相当するコンポーネントコンテナは実行システムと本体部分のインタフェースを提供するとともに，コンポーネントの含有関係や配置，コンポーネント間通信を実現する．

3.4.1 コンポーネント階層管理

図 3 のようにコンポーネント階層は各コンポーネントに対応したコンテナどうしが木構造を形成することによって維持される．コンテナは `MDComponent` クラスまたはそのサブクラスである `MDContainer` クラスから定義され，前者は `java.awt.Component` クラスのサブクラスとして定義されたコンポーネントのコンテナとなり，他のコンポーネントを含まないコンポーネントを管理する．後者は `java.awt.Container` のクラスのサブクラスとして定義されたコンポーネントのコンテナとなり，他のコンポーネントを 0 個以上含有可能なコンポーネントを管理する．両者ともに木構造の枝になるが，後者はさらに 0 個以上のコンテナを枝として持つことができる．コンピュータ内のコンポーネント移動はコンテナが維持する木構造における部分木の移動として実現される．一方，コンピュータ間移動では前述のように移動対象となるコンポーネントだけでなくコンテナも整列化する．これによりコンポーネント間の含有関係やコンテンツの配置などはそのまま保持されることになる．

3.4.2 コンポーネント表示配置機構

コンポーネント本体の描画プログラムは矩形領域の中にそれのコンテンツを表示する．また，コンポーネントは他のコンポーネントを含有する場合は，そのコンポーネントの領域内にそれが含有する各コンポーネントに矩形表示領域を割り当てる．コンテナはコンポーネントに関する属性情報を保持し，`MDComponent` クラスによるコンテナはコンポーネントのコンテンツ表示領域の最小，最大，推奨サイズ，編集可能性，重ね合わせ順番，オーナ名などを保持・管理する．一方，`MDComponent` クラスのサブクラスとして実現される `MDContainer` クラスによるコンテナは `MDComponent` クラスの保持・管理情報に加えて，コンポーネントに含有されるコンポーネントのコンテンツ表示領域の位置やサイズを管理する．なお，コンポーネント移動では移動元と移動先コンポーネントが同じサイズの表示領域を提供するとは限らない．このため移動対象となるコンポーネントは移動先のコンテナにより与えられた領域に合わせて，それのプログラムにより再描画を行うことになる．また，移動元コンポーネントでは，これまで含有していた移動対象コンポーネントが消失することになるが，移動元コンポーネントは後述の配置管理プログラムに従ってそれ自身のコンテンツおよびそれが含有するコンポーネントの再配置を行うことになる．

3.4.3 コンポーネント移動・配置の GUI 操作

コンポーネントの表示領域上において，マウスなどの右ボタンをクリックするとこのコンポーネントを含有する `MDContainer` コンテナがコンポーネントの表示領域の四隅にコントロールボックスと外周枠を描画する（**図 4**）．そして，このボックスをドラッグすることによりコンポーネントの表示領域サイズを変更できる．また，外周枠をドラッグすることにより表示位置を変更できる．コントロールボックスを描画した状態でさらにマウスなどの右ボタンをクリックするとポップアップメニューが表示され（**図 5**），そのコンポーネントのコンピュータ間移動，永続化，停止などができる．また，OS 側のドラッグドロップ処理とも連動させていることから，ファイルとして永続化されたコンポーネントをドキュメント上にドラッグアンドドロップするとコンポーネントとしてそのドキュメント上に表示・実行させることができる．このほか，既存のドローイングソフトウェアでは描画オブジェクトのグループ化機能を提供することが多いが，このフレームワークでは複数コンポーネントを透明コンポーネントに含有させることにより，コンポーネントのグルー

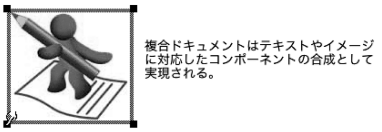

図 4 コンポーネントの配置サイズ・位置の変更
Fig. 4 Editing the layout of component.

図 5 コンポーネントの制御ポップアップメニュー
Fig. 5 Popup-menu for controlling component.

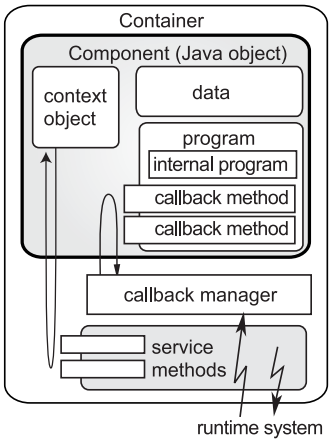

図 6 コンポーネントの構造
Fig. 6 Structure of component.

プ化もできる．

補　　足

実行システムはコンポーネント階層の根に位置するコンテナを提供し，デスクトップ上の透明なコンテンツを提供するコンポーネントとして扱われる．この結果，たとえばウィンドウに相当するコンポーネントは実行システムに対応するコンポーネントに含有されるコンポーネントとなる．また，コンポーネントの改竄を防ぐために，コンテナはそれが含有するコンポーネントの移動や配置，含有を明示的に制限・固定することができる．一方，コンポーネントのコンテンツ編集はコンポーネント本体に任されるが，その本体プログラムが `setEditable(boolean b)` メソッドを実装しているときは，その引数値により編集可否を設定できる☆．

3.5 コンポーネント

コンポーネントの本体部分は `java.awt.Component`（またはそれのサブクラスである `java.awt.Container` クラス）のサブクラスから定義されるオブジェクトであり，Java 言語 GUI クラスである Swing ライブラリ内の多くの GUI 部品，たとえばテキストエディタクラス `JTextArea` やウインドウクラス `JFrame` などはコンポーネント本体としてそのまま利用できることになる．ただし，コンポーネントを定義するクラスは `Serializable` インタフェースを実装する必要がある．**図 6** にコンポーネントの構造を示す．

3.5.1 プログラミングモデル

コンポーネントの本体部分は `java.awt.Component` クラス（または同クラスのサブクラスである `java.awt.Container`）のサブクラスとして定義されることから，それらのクラスを通じて GUI 描画に関わる基本メソッドを提供されることになる．たとえばコンポーネントの移動やそれのコンテンツ領域のサイズまたは位置が変更されたときは，そのコンポーネントを含有するコンポーネントのコンテナにより，`java.awt.Component` クラスの再描画メソッド（`repaint()`）が呼び出される．また，コンポーネントの定義において，それらのメソッドをオーバライトとすることにより，そのコンポーネントに特有の表示ができるようになる．一方，コンポーネント側から実行システムが提供する各種サービス，たとえば移動や永続化などを利用するには，そのコンポーネントを定義するクラスは，`setMDContext( MDContext c)` メソッドを定義する `MDContextListener` インタフェースを実装する必要がある．実行システムはコンポーネントの生成時や到着時などに同メソッドを呼び出し，その引数としてコンテキストと呼ぶオブジェクト（`MDContext` クラスのインスタンス）を与える．このオブジェクトはたとえば次のようなコンポーネントの移動，永続化，複製コマンドを提供する．

移動 `move(URL u, String s)` メソッドを実行したコンポーネントおよびそれが含有するコンポーネントは，`u` で示されたコンピュータ上のコンポーネント `s` に移動する．ここで `s` はコンポーネント名または識別子である．

永続化 `save(URL u)` メソッドを実行したコンポーネントおよびそれが含有するコンポーネントは，`u` で示されたファイルに保存される☆☆．

☆ ただし，編集可否に関する指示に対して，その編集制限の有無およびその方法はコンポーネントに任される．

☆☆ WebDAV プロトコルに対応していることから，遠隔サーバ上にコンポーネントを保存することもできる．

複製 duplicate() メソッドを実行したコンポーネントおよびそれが含有するコンポーネントの複製が生成される

なお，移動や永続化における整列化では Java 言語の直列化機構を利用することから，他の Java オブジェクトと同様に整列化の直前にはコンポーネントを構成する Java オブジェクトの readObject() メソッドが，再びコンポーネントに戻されるときに writeObject() メソッドが呼ばれることになる．このため，コンポーネントは必要に応じて整列化の準備およびコンポーネントに逆整列化されたときの準備を明示的に定義することもできる☆．また，前述したように実行システムはコンポーネントのライフサイクル状態が変化するごとにそのコンポーネントにイベントを発行することができるが，コンポーネントがそれらのイベントを受け取るには，コンテキストを通じてイベントに対応したリスナオブジェクトを登録する必要がある．現在の実装では，たとえばコンポーネントの含有関係やコンテンツ表示領域サイズおよび位置が変化したときに呼び出されるリスナオブジェクトを提供している．

3.5.2 コンテンツの編集・配置機構

コンポーネントの配置やサイズは，ユーザが図 4 に示されているコントロールボックスなどをドラッグアンドドロップすることにより変更されるが，一方でこのフレームワークではコンポーネント移動にともない，ユーザの編集とは独立にコンポーネントの追加や削除が発生することがある．そこで自動配置に対応した配置管理プログラムを定義・導入することができる．これは Java 言語の AWT 配置管理クラス（java.awy.LayoutManager）のサブクラスとして定義されるものであり，たとえば java.awt.FlowLayout クラスのインスタンスを配置管理プログラムと登録すれば含有するコンポーネントをその到着した順番に並べることができ，また java.awt.Grid によりコンポーネントを所定サイズにして並べることもできる．

3.5.3 コンポーネント間通信

複合ドキュメントフレームワーク以外のコンポーネント技術[18] ではコンポーネントを組み合わせることにより，情報システムの動作を実現することから，コンポーネント間の同期や情報交換が重要であったが，このフレームワークは複合ドキュメントという表現を実現するためにコンポーネントを利用することから，コンポーネント間の同期や情報交換に対する要求は少ない．ただし，分散処理の実現を考慮して，現在の実装ではコンポーネントは他のコンポーネントへのサービスとして提供することが可能である．具体的にはコンポーネントはそれのコンテナにオブジェクトを外部サービスとして登録すると，階層関係または隣接関係にあるコンポーネントがそのオブジェクト内のパブリックメソッドを同期またはフューチャオブジェクトを介して呼び出せるようにしている．

補　　足

このフレームワークにおけるコンポーネントのプログラミング方法は Java 言語の GUI プログラミングと同様であり，Java 言語の Applet や JavaBean による GUI 部品の開発経験があれば，コンポーネントの開発も容易にできると予測される．

4. 応 用 事 例

ここでは複合ドキュメントフレームワークの応用例を示す．

4.1 フレームワークの実行画面

図 7 はフレームワークの実行画面例であり，同図の左ウィンドウはコンポーネントのパレットであり，複合ドキュメントを作成するうえで基本となるコンポーネントが登録されている．そして，パレットからコンポーネントに対応したアイコンをドラッグし，複合ドキュメント上でドロップすることにより，コンポーネントの複製がドキュメント上に配置される．なお，パレット自体もコンポーネントを含有するコンポーネントとして実装されているため，編集・変更済みコンポーネントをパレットに登録することもできる．また，このパレット自体もそれが含有しているコンポーネントとともに移動や永続化することもできる．そして同図の中央および右ウィンドウは編集中の複合ドキュメントコンポーネントである．

4.2 コンテンツ対応コンポーネント

図 7 に示された複合ドキュメントのように，たとえばテキストコンポーネント，JPEG または GIF イメージビューアコンポーネント，Web ブラウザコンポーネント，ドキュメントコンポーネント，ウィンドウコンポーネント，ドキュメント内フレームに対応したボッスコンポーネントなど数多くのコンポーネントが実装されている．なお，右のウィンドウにある時計コンポーネントは 1 秒おきに現在時刻を表示するものであるが，このフレームワークではコンポーネントは実行スレッドを保持することができるため，能動的なコンポーネントも扱える．

☆ Java オブジェクトのインスタンス変数は整列化の対象となることから，コンテンツをインスタンス変数として保持している限りは，これらのメソッドを再定義する必要はない．

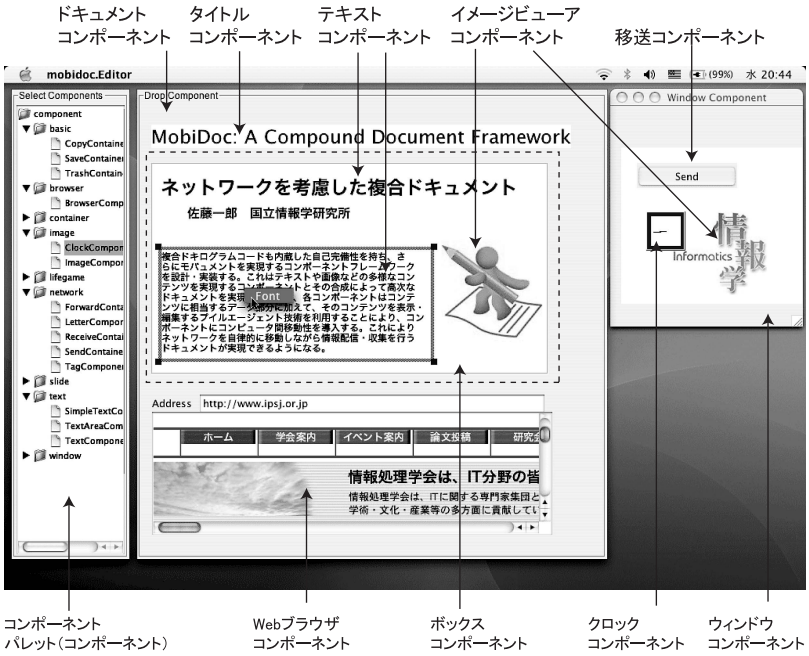

図 7 複合ドキュメントの例
Fig. 7 Example of compound document.

4.3 コンポーネント合成例

次の複数コンポーネントの合成事例を示す.

4.3.1 電子メール

既存の電子メールシステムでは，各種コンテンツを含むドキュメントをメールの添付ファイルとして送信することはできるが，ドキュメントそのものが電子メールになるわけではない．また，稟議書などの処理では複数の宛先を巡回するメールが必要となることがあるが，既存の電子メールシステムでは宛先はたかだか 1 個に制限されることが多い．一方，このフレームワークではドキュメントやコンポーネントに対して，移動制御を行うコンポーネントをドラッグアンドドロップ操作などにより貼り付けるだけで自律的に移動可能なドキュメントやコンポーネントを実現できる．たとえば**図 8** に示されているように，メール本文に相当する複合ドキュメントにドラッグアンドドロップなどの操作を通じて，コンポーネントパレットからメールヘッダ部分に相当する移動制御コンポーネントを選択し，そのドキュメントコンポーネントに含有させることができる．そして，移動制御コンポーネントに宛先アドレスなどを指定して，送信ボタンを押すとその宛先にドキュメントコンポーネントが移動することになる．

このほか，複数宛先を持つ移動制御コンポーネントを含有させることもできる．たとえば**図 9** の稟議書用の移動制御コンポーネントである．あらかじめ決裁

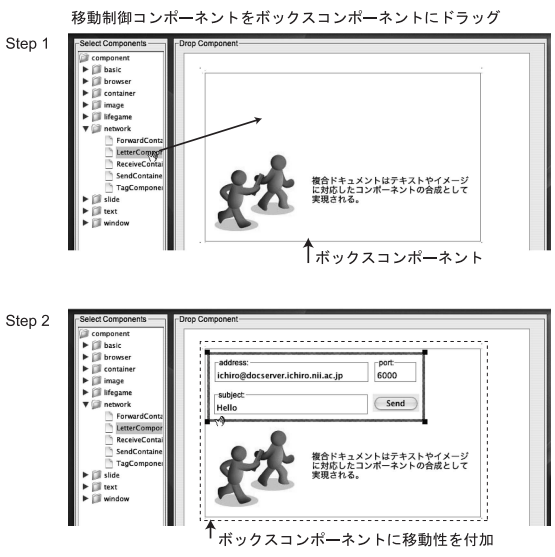

図 8 複合ドキュメントによる電子メール
Fig. 8 Letter document.

者のコンピュータアドレスが与えられており，押印に相当するスタンプコンポーネントが貼られていない担当者のコンピュータに移動する．この際，他稿[14),15)]で示した移動経路記述言語により複雑な移動経路も可能である．また，ドキュメントコンテンツの表示・編集機能はコンポーネントに任されることから，現在の実装では，たとえば配信後は編集機能を制限して改竄

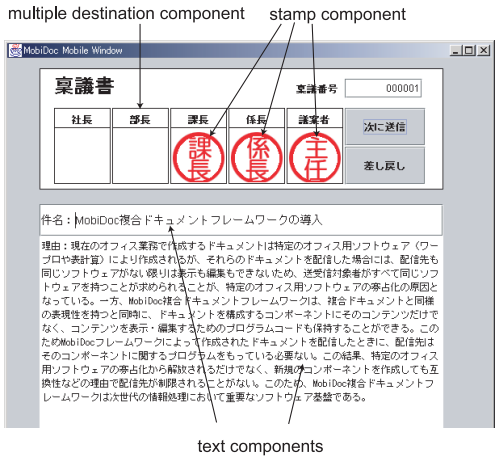

図 9 複数宛先を持つ電子メール
Fig. 9 Letter document with multiple destinations.

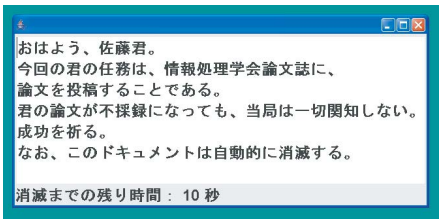

図 10 所定時間後に消滅するウィンドウコンポーネント
Fig. 10 Time-limited window.

を防ぐメールや，所定時間が経過すると含有するコンポーネントとともに自らを消滅させるメール（**図 10**）なども提供されている．

4.3.2 分散プレゼンテーションシステム

複合ドキュメントとして作成されたプレゼンテーション用スライドを表示するシステムである．その特徴はスライドの全体または部分をコンピュータ間で移動できることであり，たとえばスライドを別のコンピュータに移動させて再表示することや，スライド中の図やテキストを別のコンピュータに移動させることができる．**図 11** はスライドコンポーネントとそれを含有するスライド表示コンポーネントである．前者は 1 枚のプレゼンテーション用スライドに対応するものであり，ここではテキストやイメージに相当するコンポーネントを含有している．後者はそのスライドコンポーネントの表示・移動を管理するコンポーネントである．これは 0 枚以上のスライドコンポーネントを含有し，その配置管理プログラムとして `java.awt.CardLayout` クラスを拡張したコンポーネントの積層配置用クラスを利用することにより，スライドコンポーネントを重ねて配置する．そして図 11 にあるボタンを操作することで最上層のコンポーネントを変更して，順送りなどの複数スライドの切替えを実現する．また，これはボタン操作により，スライドコンポーネントを他のコンピュータ上のスライド表示コンポーネントに移動さ

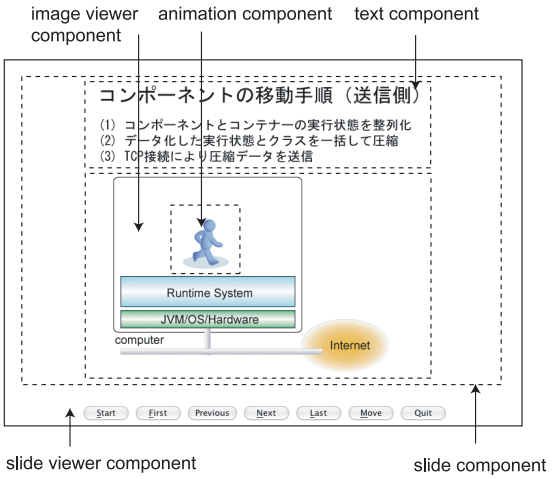

図 11 スライドコンポーネント
Fig. 11 Slide component.

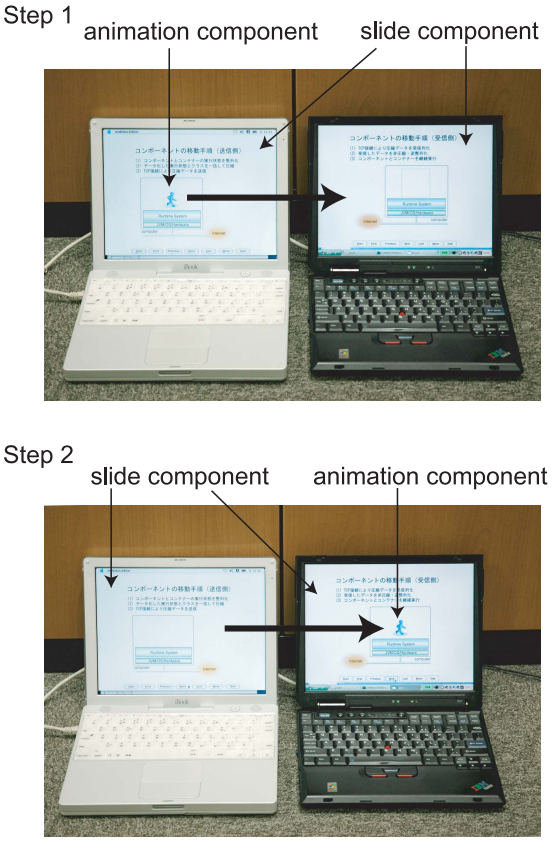

図 12 相違なコンピュータ上のスライドコンポーネント間を移動するコンポーネント
Fig. 12 Migration of component between slide components running on different computers.

せることもできるため，単なるスライドの切替えだけでなく，別のコンピュータにスライドを移動させながらプレゼンテーションすることも可能になる．図 11 の中央にあるのはアニメーションコンポーネントであり，これは GIF アニメーションを表示するコンポーネントであるが，コンピュータ間移動性を持ち，それの領域内をクリックすることで，他のコンピュータ上のスライドコンポーネントに移動して，表示を継続する．**図 12** は分散プレゼンテーションシステムを 2 台のコンピュータで利用したときの様子である．左のコンピュータ（Macintosh）において，スライドコンポーネント上のアニメーションコンポーネントをクリックすると，右のコンピュータ（Windows 稼働の PC）に移動してアニメーション表示を継続する．

5. 議　　論

本論文で示したフレームワークの効用について議論する．

コンポーネントサイズ

コンポーネントにプログラムを内蔵することより，ドキュメント配信における通信トラフィックの増大や永続化における 2 次記憶領域の消費する可能性がある．そこで，現在の実装は必ずしもコンポーネントサイズおよび移動速度に関して最適化した実装ではないが，既存の代表的なドキュメント作成アプリケーション，たとえば MS-Word との比較を示しておく☆．プレインテキストからなるドキュメントを例にとると，このフレームワークでは空のテキストコンポーネントのサイズは 5.6 KB，1 千文字を格納したときは 6.3 KB，1 万文字のときは 9.9 KB，10 万文字のときは 38.3 KB となるのに対し，MS-Word による格納したファイルはそれぞれの文字数におけるサイズが 19.5 KB，21.0 KB，47.1 KB，306.2 KB となり，このフレームワークによるコンポーネントの方が格納または転送サイズが小さい☆☆．このほか，複数のフォント種類やサイズを含むリッチテキストも同様の傾向を持つ．また，さらに複雑なコンポーネント，たとえば図 7 における点線枠で囲まれたコンポーネント，つまりボックスコンポーネントとそれに含有された 2 個のテキストコンポーネントおよびイメージビューアコンポーネントのサイズは 68 KB となる．

一方，同等内容のコンテンツを MS-Word で作成したときの格納ファイルのサイズは 24 KB となる．各コンポーネントのサイズはそのデータ量とプログラムの大きさに依存するとともに，複数コンポーネントからなるドキュメントの場合は，それを構成するコンポーネントの種類が多くなるほどサイズが大きくなり，逆にコンポーネントの種類に対してデータ量が大きい場合は，既存のアプリケーションの格納ファイルサイズとの差は小さくなり，むしろ逆転することも多い．ただし，最近のネットワーク通信帯域の向上や 2 次記憶容量の増大を考慮すると，プログラムの内蔵によるコンポーネントサイズの増加は必ずしも運用上の制約とはならない．また，通信や永続化対象となるプログラムはそのドキュメントを構成するコンポーネントに関するものだけであり，さらにドキュメント内のコンポーネント間で重複するプログラムを省くこともできる．

コンポーネント移動

コンポーネントの移動性能は次のようになる．上述のボックスコンポーネントおよびその含有コンポーネントを 2 つのコンピュータ間における移動・再描画に要する時間は 580 msec となる．ここでコンピュータは Pentium-III（1 GHz）の PC-AT 互換機，OS は Windows 2000，接続ネットワークは Fast Ethernet であり，フレームワークおよびコンポーネントは Java 言語（J2SE ver. 1.4.2）の仮想機械上で実行された．なお，このうちでコンピュータ間転送そのものに要した時間は 20 msec 以下であり，残り時間のほとんどは整列化とその逆変換，再描画処理などの，通常のドキュメント編集アプリケーションにおいても想定される処理に関するコストである．このため，一般のドキュメント交換に要求される通信速度を考慮すると，Fast Ethernet 以上の通信性能を持つネットワークにおいて，コンテンツとともにプログラムを転送することが，実運用上のボトルネックになることは少ない．なおコンポーネントがいったん移動すればネットワークは切断してもよい．

セキュリティ

モバイルエージェントなどと同様に，このフレームワークは他のコンピュータにプログラムの移動させることになるため，アクセス制限などのセキュリティ機構が重要である．ただし，モバイルエージェントが移動先で情報収集などのファイルシステムやデータベースへのアクセスを要求するのに対し，このフレームワークのコンポーネントの場合，プログラムはコンポーネ

☆ このフレームワークのドキュメントサイズは，コンテンツ量に加えて，ドキュメント中のコンポーネントの種類，コンポーネントの実装などに影響される．また，既存アプリケーションのドキュメント格納ファイルサイズはコンテンツ種類や格納フォーマット・アルゴリズムに依存するため，体系的な比較は困難である．

☆☆ 実験に用いたテキストデータは英文．なお，このフレームワークによる格納コンポーネントは JAR 圧縮されている．

ントが保持するデータの表示・編集することが目的となり，移動先コンピュータのファイルシステムやデータベースなどの 2 次記憶装置を利用することは少なく，また他のコンポーネントを直接アクセスすることもない．このほか，コンポーネントがコンピュータ間移動する場合も実行システムを通じて転送されるため，コンポーネントが直接ネットワークを操作する必要はない☆．つまり，コンポーネントはそれの領域内の描画ができるとともに，それに与えられたキーボードおよびマウスなどからの入力イベントを受け取れれば，移動先コンピュータのその他の計算リソースにアクセスする必要はない．このため，コンポーネントのプログラムコード（Java クラス）に対する Java 言語のセキュリティマネジャにより管理されたアクセス制限項目に 2 次記憶装置やネットワークを加えることにより，Java の Applet と同様に Java セキュリティマネジャがコンポーネントが移動先コンピュータのそれらのリソースにアクセスすることを容易かつ安全に制限できるとともに，その際のコンポーネントの動作への影響もない☆☆．

6. 関 連 研 究

複合ドキュメント技術として COM/OLE [3]，OpenDoc [1]，CommonPoint [9] などの複合ドキュメントフレームワークが数多く提案されてきた．その中でも COM/OLE は幅広く利用されている枠組みであるが，コンポーネントとそれを含有するコンテナの構造が異なるため，コンポーネントの入れ子することができないなどの表現性の制限が大きく，またコンポーネントのコンピュータ間移動などは扱えない．なお，COM/OLE モデルを基礎として DCOM などの分散システム向けコンポーネントが提供されているが，遠隔手続き呼び出しを中心とした分散処理の実装を目的としており，複合ドキュメントとして編集・利用は考慮されていない．一方，Taligent により提案された CommonPoint [9] と Apple コンピュータや IBM などが開発した OpenDoc は本論文の枠組みと同様にコンポーネントの中に複数のコンポーネントを含めることができる．また，両枠組みともにコンポーネントの永続化が可能であり，スクリプティング言語およびファイル共有サーバなどを組み合わせるなどの技巧的な手法によりコンピュータ間コンポーネント移動は原理的には可能であるが，フレームワーク自体は移動性を提供しない．このほか，Java 言語のコンポーネントモデルとして JavaBean がある．当初，JavaBean の仕様[7] では各コンポーネントは対等な関係に限定されていたが，その後の仕様では階層構造を許している[5]．ただし，CommonPoint や OpenDoc などと同様に JavaBean は永続化が可能であるが，ネットワーク処理においては受動的な存在にすぎず，自律的な移動はできない．このほか，複合ドキュメントフレームワークとして Intelligent Pad [19] や HotDoc [4]，DPI [2] などがあるが，ドキュメント移動などネットワークおよび分散処理に対応した機能は提供していない．

なお，複合ドキュメントと類似した技術にアクティブドキュメントがある．アクティブドキュメントと称する既存枠組みの多くは対話処理やアニメーションなどの動的コンテンツを指向したものが多いが，その中でも Active Mail [6] や HyperNews [8] などの少数の枠組みはコンテンツに応じて配布先を変更できる機能を提供している．ただし，メールなどに経路制御用のスクリプティング言語を埋め込むものであり，多様なコンテンツを合成するなどの表示・編集機能に欠けている．

このほか，コンピュータ間移動性を持つソフトウェアとして，モバイルオブジェクトやモバイルエージェントがある．これらは分散処理における通信コストの削減を意図しているものであり，またオブジェクトまたはエージェントどうしの合成手法を欠いており，そのまま複合ドキュメントを実現できない．その中でも著者が過去に設計・実装した MobileSpaces システムはモバイルエージェントに階層構造を導入しており，さらに同構造を利用した複合ドキュメントフレームワークを提案している[11),12),17)] ☆☆☆．ただし，これらはモバイルエージェントを複合ドキュメントのコンポーネントとして利用することが目的となっているのに対し，本論文のフレームワークは複合ドキュメントに特化したコンポーネントの設計したものであり，両者は実装方法が大きく異なっている．このほか，前者は汎用的なプログラムであるモバイルエージェントと複合

☆ Web ブラウザなどネットワークアクセスを目的としたコンポーネントを除く．

☆☆ 現在の実装のセキュリティ機構は Java 言語によるセキュリティ機能を利用している．なお，他のプログラミング言語による実装の際には同様の機能がその言語またはプラットフォームにより提供されていることが前提となる．

☆☆☆ 査読者へ：前稿[17] は未刊であるが，他稿[11),12)] で提案した複合ドキュメントフレームワークの編集機能を使ってネットワーク処理の定義・変更を実現するものであり，他稿[11),12)] のフレームワークの問題をそのまま継承している．また，モバイルエージェントを利用したコンポーネントを前提にしており，そのフレームワークも本論文のものは異なっている．

ドキュメントのコンポーネントの実行・管理粒度が一致しないという問題もあった．また，3 章で議論したように前稿のシステムではコンポーネント移動時のコンポーネント再描画や配置管理はコンポーネント側でなく，別の実行スレッドで動作するフレームワーク側システムに任されていた．この結果，コンポーネント移動が，フレームワーク側の配置管理プログラムがそのコンポーネントの再配置処理中であると，コンポーネント側に用意された描画・配置に関する属性領域に，画面表示上の最新状況が反映していない可能性が残されていた．また，前稿のシステムでは配置管理などを基本ポリシをフレームワーク側で定義していることから，コンポーネントやドキュメントの配置・編集は 1 つの方法は固定されていた．この結果，ドキュメント上のコンテンツもデスクトップ上のウィンドウもその配置やサイズ変更操作が同じになり，操作上の違和感が大きいなどの問題があった．一方，本論文で提案したフレームワークはコンポーネントのコンテナ内で配置管理することから，配置・編集方法をコンポーネント単位で変更可能であり，ドキュメントの多様性が向上している．

7. 課　　題

今後の課題を述べる．このフレームワークの最大の問題はビジネスモデルをどのように構築するかである．コンポーネントにプログラムを内蔵するため，従来のパッケージアプリケーションのようにプログラムの配布時に課金することは難しい．一方でこのフレームワークでは，コンポーネントのコンテンツへのアクセスはそのコンポーネントが内蔵するプログラムに限定されることから，そのプログラムを通じてコンテンツの表示や編集を明示的に制限することも可能であり，さらにネットワーク上の認証システムを組み合わせることでコンテンツ単位の従量課金にも道を開くことができる．また，コンテンツを盗聴や改竄などから保護することも重要となるが，これもコンテンツ側プログラムに暗号化などのセキュリティ機構を導入するという方法が考えられる．ただし，具体的な保護・課金方法およびそれらの安全性・可用性に関する検証は今後の課題となる．また，多様なコンポーネントの実装が求められるとともに，フレームワークの性能向上と安定化も重要である．特に現在の実装ではコンポーネントのプログラミングインタフェースや GUI 操作機構は必要最小限のものであり，その発展・整備が求められる．このフレームワークとコンポーネントの実装では Java 言語を利用したが，実装言語は Java 言語を前提とするわけではない．他の言語や実行環境でも実現可能であるが，異なるプラットフォーム間のコンピュータ移動やセキュリティ上の理由から仮想機械により実行可能であることと，コンポーネントプログラムおよびその変数格納領域（ヒープ領域など）のデータ化（直列化または整列化）機構が提供されていることが望まれる．たとえば .NET フレームワークにより提供されている言語非依存な仮想機械（CLR）は実装・実行環境として重要である．

8. ま と め

ネットワーク処理を考慮した複合ドキュメントフレームワークを設計・実装した．これはテキストやイメージなどのコンテンツに対応したコンポーネントから多様なドキュメントを生成できるとともに，各コンポーネントにコンピュータ間移動性を導入することにより，コンテンツに応じて配信先を変更したり，複数コンピュータを移動したりするドキュメントも可能になる．また，コンテンツとプログラムコードを一体化した自己完備コンポーネントとすることにより，移動先のアプリケーションプログラムの有無によらず，多様なドキュメントの表示と編集が可能となる．そのフレームワークの実現例として Java 言語による実行システムおよびコンポーネントの実装を行った．

謝辞　担当編集委員および査読者から数多くの貴重なコメントをいただいた．つつしんで感謝の意を表する．

参 考 文 献

1) Apple Computer Inc.: OpenDoc, White Paper, Apple Computer Inc. (1994).
2) Beaudoux, O. and Beaudouin-Lafon, M.: DPI: A Conceptual Model Based on Documents and Interaction Instruments, People and Computer XV—Interaction without frontier, *Joint proceedings of HCI 2001 and IHM 2001*, pp.247–263, Spring Verlag (Sep. 2001).
3) Brockschmidt, K.: *Inside OLE 2*, Microsoft Press (1995).
4) Buchner, J.: HotDoc: a framework for compound documents, *ACM Computing Surveys*, Vol.32, No.1, p.33 (2000).
5) Cable, L.: Extensible Runtime Containment and Server Protocol for JavaBeans, Sun Microsfystems (1997). http://java.sun.com/beans
6) Goldberg, Y., Safran, M. and Shapiro, E.: Active Mail: A framework for Implementing Groupware, *Proc. ACM Conference on Computer-Supported Cooperative Work*

(*CSCW'92*), pp.75–83, ACM Press (Oct.1992).
7) Hamilton, G.: The JavaBeans Specification, Sun Microsfystems (1997). http://java.sun.com/beans
8) Morin, J.: HyperNews, a Hypermedia Electronic-Newspaper Environment based on Agents, *Proc. HICSS-31*, pp.58–67 (Jan. 1998).
9) Potel, M. and Cotter, S.: *Inside Taligent Technology*, Addison-Wesley (1995).
10) Satoh, I.: MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, *Proc. International Conference on Distributed Computing Systems* (*ICDCS'2000*), pp.161–168, IEEE Press (Apr. 2000).
11) Satoh, I.: Mobile Agent-based Compound Documents, *Proc. ACM Symposium on Document Engineering* (*DocEng'2001*), pp.76–84, ACM Press (Nov. 2001).
12) Satoh, I.: MobiDoc: A Mobile Agent-based Framework for Compound Documents, Informatica, *International Journal of Computing and Informatics*, Vol.25, No.4, pp.493–500 (2001).
13) Satoh, I.: Network Protocols for Mobile Agents by Mobile Agents, 情報処理学会論文誌, Vol.44, No.3, pp.760–770 (2003).
14) 佐藤一郎：モバイルエージェントの経路記述と選択機構，情報処理学会論文誌，Vol.44，No.6, pp.1473–1482 (June 2003).
15) Satoh, I.: Building Reusable Mobile Agents for Network Management, *IEEE Trans. Systems, Man and Cybernetics*, Vol.33-C, No.3, pp.350–357 (2003).
16) Satoh, I.: A Testing Framework for Mobile Computing Software, *IEEE Trans. Softw. Eng.*, Vol.29, No.12, pp.1112–1121 (2003).
17) Satoh, I.: A Component Framework for Document-centric Networking, *IEICE Trans. Comm.*, Vol.E87-B, No.7, pp.1826–1833 (2004).
18) Szyperski, C., Gruntz, D. and Murner, S.: *Component Software*, 2nd Edition, Addison-Wesley (2002).
19) Tanaka, Y. and Imataki, T.: IntelligentPad: A Hypermedia System allowing Functional Composition of Active Media Objects through Direct Manipulations, *Proc. IFIP'89*, pp.541–546 (Aug. 1989).

(平成 16 年 5 月 24 日受付)
(平成 16 年 12 月 1 日採録)

佐藤　一郎（正会員）

1991 年慶應義塾大学理工学部電気工学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻後期博士課程修了，博士（工学）．1996 年お茶の水女子大学理学部情報科学科助手，1998 年同学科助教授．2001 年より国立情報学研究所助教授．また，2002 年より総合研究大学院大学数物科学研究科助教授（併任）．このほか，1994〜1995 年 Rank Xerox Grenoble 研究所客員研究員，1999〜2001 年科学技術振興事業団さきがけ研究 21（「情報と知」領域）研究員．1996 年度情報処理学会論文賞，1999 年度同学会山下奨励賞受賞，分散システムおよびユビキタスコンピューティングのミドルウェアやアプリケーション等の研究に従事．電子情報通信学会，IEEE，ACM 各会員．