マイクロコンピュータと
ワークステーション 55－2
（1989. 5. 16）

「鼎（かなえ）」における複合文書データ混在方式

暦本 純一
日本電気（株）ソフトウェア生産技術開発本部

山崎 剛
日本電気マイコンテクノロジ（株）

猪狩 錦光
日本電気技術情報システム開発（株）

1989年5月16日

概要

我々は現在、ユーザインタフェース構築基盤システム「鼎」を開発中である。鼎では、ソフト開発環境でよく利用される、6種類のメディアに対する編集機能を部品として提供している。本稿では、これらのメディアをひとつの文書中に混在させ、表示・編集・蓄積する方式について述べる。まず、複数のタイプのメディアを関連づける方法について考察し、ネスト型とリンク型を提供する必要性を述べる。オブジェクト指向に基づいた方式で、それらがメディアとしての独立性をたもったまま実現できることを示す。最後に、実装経験から得た知見について述べる。

The Media-Mixed Document Processing in CANAE

Junichi REKIMOTO
Softoware Engineering Laboratory, NEC Corporation
11-5, Shibaura, 2-chome, Minato-ku, Tokyo 108, Japan

Go YAMAZAKI
NEC Microcomputer Technology, Ltd.

Kanemitsu IGARI
NEC Scientific Information System Development Corporation

ABSTRACT

Canae is a user interface development system which is currently developed in our group. Canae supports editing facilities of six medeia types used in various software-development environments. Canae has two types of media-mixing methods, Nest type and Link type. In this paper, we describes the necessity of two methods. Using object-oriented programming style, we show that these media-mixing can be implemented without affecting on each media type's implementation. Our experience in implementation is also described.

1 鼎システムの概要

最近の著しいハードウェアの機能向上と低価格化，X-Windowのような共通基盤としてのウインドウシステムの普及によって，マルチウインドウ環境化で，テキストだけでなく各種のメディアを扱えるようなアプリケーションへの要求が非常に高まってきている。ところが，これらのアプリケーションでは，ユーザとの対話部分(以下UI部と略す)の実現に高度なプログラミングを必要とし，UI部の開発がネックになってシステムの構築を困難なものにしていた。UI部は，実際にそのシステムを使いながら修正して，使い易さを改善していくべきものであるが，従来はそれも困難であった。

この問題を解決するために，アプリケーションからUI部を分離して，専門のモジュールないしはプロセスに担当させようというアプローチ (UIMS - User Interface Management System) がある。従来のUIMSではUI部の対話制御(メニューからコマンドを選択する，等)を担当し，後はアプリケーションが担当することになっていた。しかしUI部の中で最も作成にコストがかかるのは，画面上の対象物の編集操作を制御する部分である。たとえばCASEツールではモジュール階層図をユーザに編集させたり，モジュールの機能概要を日本語入力させる部分の構築に手間がかかる。逆に，多くのアプリケーションを編集という観点から見直すと，目的は異なっていても共通するものが多い。編集機能の共通的な部分を部品化し，アプリケーションごとの差異を吸収する拡張機能をUIMSが持てば，UI部の作成コストは大幅に減少すると予測される。

このような発想に基づいて，われわれはUI部の構築基盤システム「鼎(かなえ)」を開発している[1]。鼎では，各種のアプリケーションで共通に使われる編集対象の種類(メディアタイプ)を6種類選定し，その編集機能(ファイル入出力，画面表示，マウス操作，キー操作等)を部品化している:

テキスト 文字列(日本語)。

表 マトリックス上に配置された文字列(将来的にはメディア一般)。データベース検索やフォームシートによる入力など，表の形式でユーザに見せるとわかりやすいものに使える。

グラフ構造 ノードとアークの組み合わせによる図式。モジュール間の接続や，制御フロー，状態遷移図などの入出力インターフェースとして使う。

階層構造 木構造をなすメディア。

図形 基本図形の組み合わせによる図。表，階層，ネットワーク以外の構造をもつ図や，一般的な作画による図を入力する手段として使う。

イメージ スキャンデータなどのビットマップデータ。

これらのメディアを編集するための基本機能と，機能拡張のための**スクリプト言語** (Lispベースのインタープリタ言語) との組み合わせによってアプリケーションのUI部を構築する。メディアの選定にあたっては，われわれが過去に作成してきたソフト生産支援ツール(SDMS[4]等)での経験を基にした。

たとえば図形メディアとグラフ構造メディアで、まったく同じ見かけをもった図式を作成することができるが、グラフ構造では、ノードとアークの接続関係など、図式のトポロジカルな構造を内部で保持しており、その関係に着目した処理(あるノードに接続しているノードをすべて削除する、など)が行なえるようになっている。一方、図形メディアでは、より一般的な図式を作成できるかわりに、そのような処理はサポートされていない。このように、鼎の扱うメディアは、見かけの形状のみならず、内部の論理的な構造によって分類されているのが特徴となっている。CASEツールなど、図式を入力し、その構造から処理を導き出すような応用分野で特に有効である。

2 鼎のエディタアーキテクチャ

鼎ではMVCモデルをエディタ向きに改造したものをエディタのアーキテクチャとして利用している。MVCモデルはXerox社のSmalltalk-80のユーザインタフェース構築のために考案されたモデルで，ユーザが操作する画面上の対象物をモデル(内容)とビュー(表示方式)とコントローラ(イベント制御)の三つのオブジェクトの組で表す。モデルが他からメッセージを受け取って，自分自身の内容が変化したときには，その旨をビューに通知する。ビューはモデルの変化が起きた時，画面をその変化に合わせて再表示する。ユーザからの入力はすべてイベントの形式でコントローラが一括して受け取り，モデルとビューへ処理を振り分ける。

鼎のエディタもモデル・ビュー・コントローラの三つのオブジェクトで構成されている。コントローラをエディタ画面というオブジェクト(X-Toolkit[2],[3]のWidgetで実現している)の形で実装し，そこからモデル，ビューのインスタンスをぶら下げるようにしている。エディタ画面(コントローラ)は，すべてのメディアタイプで同じクラスのオブジェクトを用いている。ビューを操作するために必要な関数群は

ビュークラスというデータ構造の中に収められている。エディタ画面はこのビュークラスの関数を呼び出すことによって再表示，スクロール，マウストラックなどの処理を行なう。原則としてひとつのメディアタイプにひとつのビュークラスが存在するが，場合によってはひとつのメディアに複数のビュークラスがあってもよい。これは，同じモデルに対して複数の異なる方法での表示を実現できるように考慮したからである。

いわゆる「文書データ」に相当するのはモデルである。モデルの内容は、編集終了時にファイルにセーブされる。

3 異種メディアの混在

前の章で説明したように、鼎では6種類のメディアタイプを定め、それに対応した編集機能を組み込みで用意している。

ところで、一般の文書や図版では、上記のメディア群を混在させて使用するのがむしろ普通である。たとえばテキストの中に図を置き、その図の中にさらに表を入れる、といったことは日常的に行なわれている。鼎が主に想定しているアプリケーションの分野はCASEツールやOAツールであり、画面上でこういった文書を扱えることへの要望は強い。

一方、ある文書から別の文書へのつなぎ情報(ハイパーメディアリンク)によって、複数の異なる文書を関連させることができる。

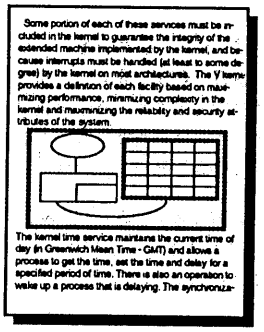
鼎では、複数のメディアを関連させるために、ネスト型とリンク型の2種類の方式を提供している(図1)。

ネスト型とは、親のメディアの一要素として、別種のメディアをネストさせる方式である。たとえば、テキストメディアに図形メディアをネストさせて、文章のなかに図版を挿入することができる。このとき、親(テキスト)は、図形メディアをひとまとまりの要素として扱う。

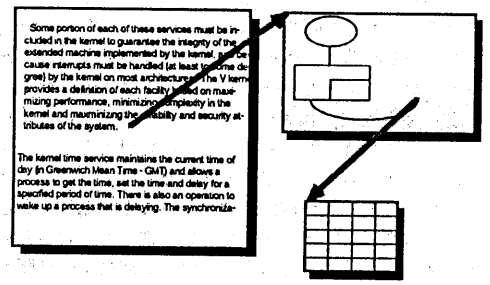
リンク型とは、親のメディアの要素に、別のメディアをたどるために必要な情報を添付しておく方式である。この情報は、鼎のスクリプトコマンドによって解釈され、たどった先のメディアが別の窓に、独立した文書として表示される。

これら2種類の方式を共に提供する理由は、以下の通りである。

ネスト型は、構成要素の一貫性が重要な場合に有効である。リンクでたどっていくのでは、一覧性にかける場合がある。ネストによって構成されているデータひとまとまりのものとして把握しやすい。また、印刷を考慮したWYSIWYG (What You See Is What You Get)な文書を構築するときには必要な機能である。



ネスト型



リンク型

図 1: ネスト型とリンク型

一方、リンク型は、複数の文書を共有するときに必要である。ネスト型はメディア間の包含関係を作りだすが、リンクでは、より一般的なネットワーク関係を作り出す。

4 ネスト型のメディア混在

以下で、親のメディアの中に子供のメディアをネストさせる場合の実現方式について述べる。多種類のメディアタイプ間の自由なネスティングをサポートすることを想定して、実現に際しては以下のような目標を立てた。

メディア独立 できるだけ、個々のメディアの機構と独立してネストを実現できるようにする。ネストの親側で、子供のデータ構造や性質を仮定しない。これは、新たにサポートするメディアのタイプが増えたときに、その影響が他のメディアに及ばないことを意味している。

実装の容易さ ネストを含まない、単一メディアの表示、編集、蓄積機能に、できるだけ容易にネスト機構を追加することが可能であるようにする。単一メディアの機構の自然な延長として、ネスト機構を組み込む。

この目標を達成するために、ネストする子供のメディアを、以下で述べるフラグメントという形式で保持することにした。

4.1 フラグメント

鼎の各メディアのモデルは、**フラグメント**という形式でファイルに格納されている。フラグメントは、モデルと同等な情報を一次元のバイト列に展開したもので、モデルをファイルにセーブするときの形式として用いる。各メディアのフラグメントは、共通のヘッダを持っており、ヘッダの情報から、フラグメントがどのメディアタイプに属するのか、フラグメントの中に何が格納されているのかが認識できるようになっている。モデルとフラグメントの相互変換操作は、以下で説明するメディア記述子内の関数によって定義されている。

図2に、表メディアに対応するフラグメントデータの例を示す。図中で、FORM BLOCK が表のデータに対応する部分、LINK BLOCK が、後述する文書属性のデータを保持している部分である。

4.2 メディア記述子

鼎でサポートしているメディアタイプには、そのメディアタイプに属しているモデルとフラグメントを操作するための基本となる関数群、データを格納した**メディア記述子**という構造体が対応している。これは、モデルやフラグメントをオブジェクトと見た時のクラス定義に対応している。

メディア記述子には、モデル生成・消去関数、ビュー生成・消去関数、選択生成・消去関数、カットバッファの管理関数、アンドゥー関数、デフォルトのキーマップ(キー入力をC関数やスクリプトコマンドに対応づける表)、デフォルトのイベントマップ(マウス操作によって発生するイベントをC関数やスクリプトコマンドに対応づける表)、そして、以下に挙げるフラグメント操作関数群が定義されている。

表示 (DrawFragment) フラグメントの内容を画面に表示する。

フラグメントからモデルへの変換 (DecodeFragment) フラグメントの内容から、それと等価な情報を持つモデルを生成する。

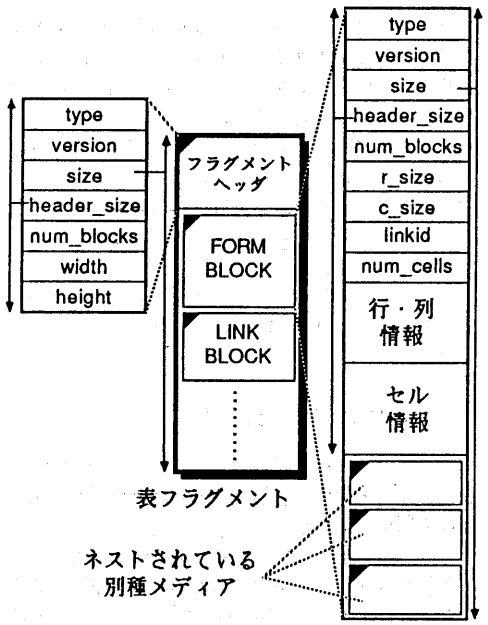


図 2: フラグメントデータの例

モデルからフラグメントへの変換 (EncodeFragment) モデルから、それと等価な情報をもつフラグメントを生成する。

また、フラグメントのヘッダ部から、フラグメントを画面に表示したときの大きさ(幅と高さ)を調べることができる。

4.3 ネストの実現方法

あるモデルが、自分の子供として別種メディアのモデルをネストさせて保持する場合の方法について説明する。子供のメディアは、モデル形式ではなく、フラグメント形式で持つことにする。

実際にネストされた子供メディアを配置・表示・編集・蓄積するには、以下の方法をとる。

配置 子供フラグメントの大きさ(幅と高さ)を求め、その大きさを持つ矩形として、親メディア中での表示位置を決定する。例えば親がテキストメディアの場合、フラグメントはその大きさを持つ「文字」として扱われ、文書中での位置を決定される。したがって、親は、子供のメディアタイプが何であるかにかかわらず、そのレイアウトを行なうことができる。

表示 子供フラグメントの DrawFragment メソッドに、フラグメントの位置情報を渡して表示させる。

このとき、親側では表示の内容に関与していないことに注意。

編集 フラグメントをモデルに変換 (DecodeFragment メソッドを呼び出す) し、そのモデルを編集するためのビューとエディタ画面を、親メディアのエディタ画面中の子供フラグメントを表示していた場所にポップアップさせる。ユーザからは、親のメディアの中に埋め込まれたままの状態でそのまま編集が可能になるように見える。実際の編集作業は、ポップアップした、子供に対応しているエディタ画面が管理することになる。

編集が終了した後は、EncodeFragment メソッドによりモデルをフラグメントに変換しなおして、親メディア中の元の子フラグメントと交換する。

蓄積 子供のフラグメントは、バイト列データなので、親のモデルをファイルにセーブするときは、単にそのバイト列を、(親モデルの方式で) 親のフラグメント形式の中に埋め込めみ、ファイルに書き出される。

ネストしたモデルが、さらに子供のメディアを編集する場合は、上記の処理が再帰的に呼び出される (図 3)。図中で、M はモデルを、V はビューを、E はエディタ画面をそれぞれ示す。まず、テキストの中に埋め込まれた図形メディアが表示されている (a) とき、図形に対して編集の開始を指示すると、図形フラグメントからモデルが生成され、編集環境が設定される (b)。さらに、図形の中の表メディアを編集しようとすると、(c) のようになる。

このように、親側では、子供のフラグメントタイプに依存せずに、配置・表示・編集・蓄積が可能になっている。

5 リンク型のメディア混在

この章では、もうひとつのメディア混在の方式である、リンク型について説明する。

5.1 文書属性

鼎があつかうメディア (モデル) の各要素には、属性情報を添付することができる。属性情報は、タグつきのバイト列データで、その使用方法はアプリケーションプログラムに委ねられている。典型的には、

- スクリプトプログラム
- ファイル名 (パスネーム)
- データベースの検索キー
- アプリケーション固有の情報

などを格納するために用いられる。属性データは、フラグメントの一部としてファイルに保存される (図 2、図 4の LINK BLOCK がこれに相当する)。

5.2 リンクの実現

リンク型は、この属性データを利用して実現している。すなわち、ユーザが文書のある部分を選択して、そこにリンクしている別の文書をオープンしようとするとき、鼎の内部では、次のような処理が起きる (ただし、鼎ではユーザ操作に対する処理が、スクリプトプログラムによって自由に再定義可能なので、ここで示した処理は、その一例にすぎない)。

1. 選択された部分に添付されている属性データを取り出す。
2. それが、スクリプトプログラムであれば、それを実行する。
3. それが、パスネームであれば、そのパスネームで示されるファイルをオープンする。
4. それが、データベースの検索キーであれば、データベースを検索し、その結果から得られるファイルをオープンする。

ハイパーメディアを構築するためのリンク情報の実装方法は、その応用によって様々である。文書に直接相手のファイル名をリンク情報として埋め込むのが適している場合もあれば、文書には検索キーワードを添付しておき、実際にリンクをたどるときに、そのキーワードでデータベースを引き、目的の文書を得るのが適している場合もある。例えば、Unix の tags は後者の典型的な例である[1]。

以上を考慮すると、全ての分野の要求に耐えるリンクの実装方式を実現するのではなく、むしろさまざまな実装方式をゆるし、しかもユーザから見た (リンクをたどるときのユーザインタフェースなどの) 振舞いが統一されているのが望ましくかつ現実的な設計であると考えられる。

鼎のリンクの実装方式では、文書属性と、そこに添付されるスクリプトプログラムの実行機能によって、上記の目的を達成している。

[1] tags は、ソースコード中の関数定義の位置情報を取り出したファイルである。vi や Emacs など、tags ファイルをサポートしているフロントエンドを利用して、複数のソースコード間を自由に渡り歩くことができる。

6 考察

上記の2方式のうち、ネスト処理部を実際に試作してみた(画面例を図5に示す)。以下で、実装経験を通して得た知見について述べる。

ネスト機構を実現する上で、メディアタイプごとにあらたに実現しなければならなかったのは、フラグメントを画面に表示するDrawFragmentメソッド、フラグメントを親メディアの中で配置し、その位置を決定する部分のみであり、その他は、単一のメディアを処理する機構がほとんどそのまま利用できた。

ネストされた編集が開始するとき、終了するときに、毎回フラグメントとモデルの相互変換が起きているが、これに関するオーバヘッドは特に問題とならなかった。これは、ネストしているデータは、一般にそれほど大きくならないからであると思われる。

Macintosh[5]上のアプリケーションでは、図形の標準データ形式(PICT形式)が定められており、テキストの中に図をいれるときなどは、この形式を用いるのが普通である。たとえば、スプレッドシートのようなメディアも、「絵」としてPICT形式に変換すれば、文書の中に張り込むことができるが、スプレッドシートとしての意味(計算能力)は失われてしまう。一方、鼎ではそれぞれのメディアを、そのまま文書の中に張り込めるので、文書中に埋め込まれたスプレッドシートで、セルの値を書き換え、再計算を行なわせることも可能である。このように、Macintoshの方法は、印刷物としてのネスティングであるのと比較して、鼎の方式はメディアとしての論理的な意味を保ったままのネスティングだと言える。

今回提案した方式では、ネスト型もリンク型も、ひとつのメディアタイプのオブジェクトに対してモデル、ビュー、エディタ画面を与えて編集環境を構成しているという点では共通している。したがって、ネスト型とリンク型を蓄積方式の相違ととらえ、ユーザの好みやアプリケーションの種類によって、プレゼンテーションとしての両方式を動的に切替えることも可能であると考えられる(リンク型で蓄積されている文書を、ネスト型のように親メディアの中に埋め込んだ形でユーザに見せる、あるいはその逆)。これについては今後の検討課題としたい。

また、今回の試作では、メディア記述子で定義されている関数は、鼎のプログラムにあらかじめリンクしておくという方式をとっている。すなわち、新たなメディアタイプを鼎に登録する場合には、システム全体の再リンクが必要である。動的リンクや共有ライブラリ機能を有効に利用することによって、必要なメディアタイプに関連する関数群を動的にロードしてくることも可能であると思われる。これも今後の課題としたい。

7 まとめ

メディアを混在させるためのふたつの方式、ネスト型とリンク型について、それぞれ実現方法を提案し、試作を通じて、性能面での問題がないことを確認した。今後は、この機構を活かした具体的な応用例を構築していきたい。

謝辞

研究の機会を与えて下さった佐谷本部長,本プロジェクトを発足させ,終始暖かい助言をいただいた紫合部長に感謝いたします。フラグメントの設計・実装に際しては、垂水浩幸氏、森 岳志氏をはじめとする日本電気(株)ソフトウェア生産技術開発本部 鼎グループのメンバから多大の協力と助言をいただきました。ここに深く感謝いたします。

参考文献

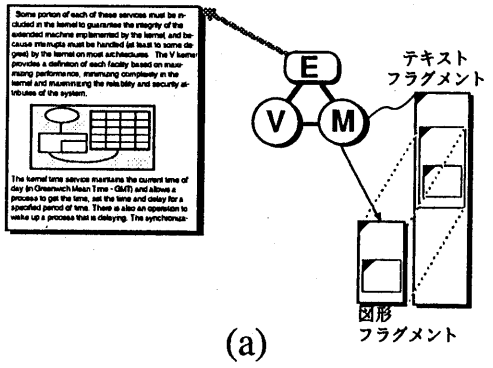
[1] 暦本,菅井,他「X ウィンドゥ上のマルチメディアユーザインタフェース構築環境: 鼎」情報処理学会第30回プログラミングシンポジウム予稿集, 1989.

[2] Joel McCormack *et al.*, "*X Toolkit Intrinsics — C Language X Interface*", X Window System, Version 11, Release 2.

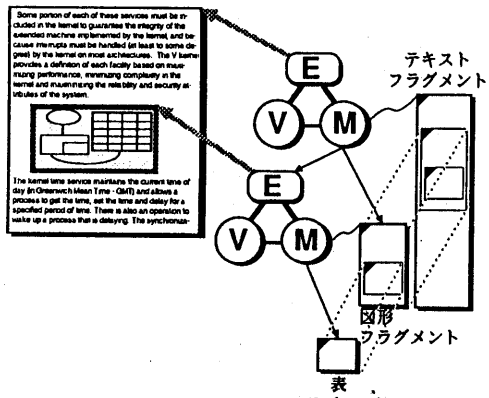
[3] Terry Weissman *et al.*, "*X Toolkit Widgets — C Language X Interface*", X Window System, Version 11, Release 2.

[4] 紫合 治,則房雅也,他「通信・制御システム系ソフトウェア生産システム」NEC技法, Vol.40, No.1, 1987. pp.10–18.

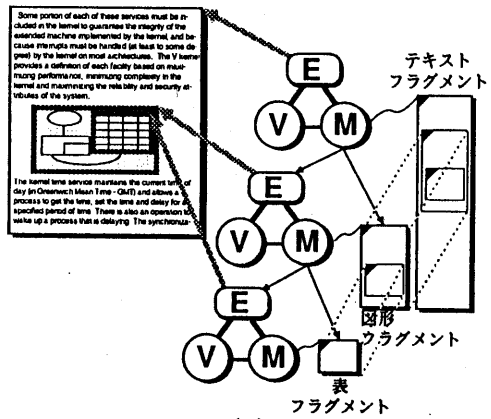
[5] APPLE COMPUTER, INC., "*Inside Macintosh Volume I-V*", Addison-Wesley, 1988.



(a)



(b)



(c)

図 3: ネストされた編集

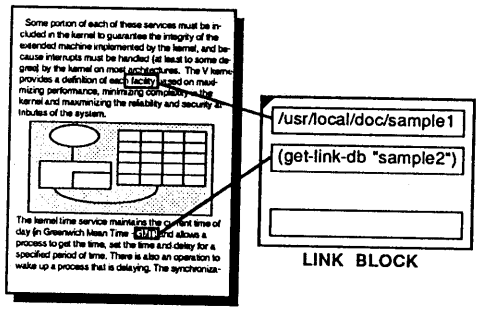


図 4: 文書属性と LINK BLOCK

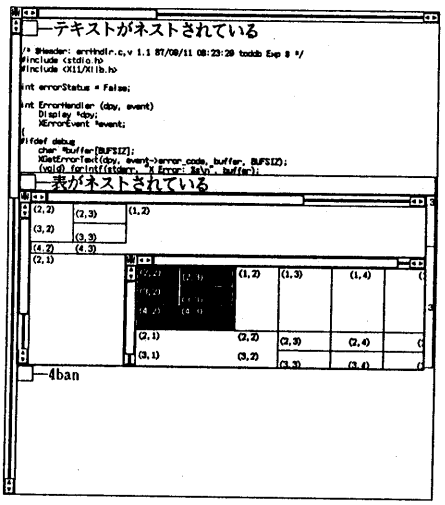


図 5: ネスト編集の画面例