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

日本電気株式会社　ソフトウェア生産技術開発本部
暦本　純一、菅井　勝、森　岳志、内山　厚子
Junichi Rekimoto、Masaru Sugai、Takeshi Mori、Atsuko Uchiyama
垂水　浩幸、杉山　高弘、秋口　忠三
Hiroyuki Tarumi、Takahiro Sugiyama、Chuzo Akiguchi
日本電気マイコンテクノロジー株式会社
山崎　剛
Go Yamazaki

概要

ウィンドゥシステムの普及にともない各種のメディア[1]を活用した視覚的ユーザインタフェースを持つシステムへの要求が高まっているが，その作成，特にユーザインタフェース部の構築は容易ではない．われわれが現在開発中のシステム，鼎(かなえ)は，各種のメディア(テキスト，図形，ネットワーク，表，階層，イメージ)を扱うアプリケーションのユーザインタフェース部を容易に構築するための基盤となることを目指している．このシステムは，各種のメディアを編集するための基本機能を部品として持ち，エディタの機能を目的に応じて変更・拡張するための拡張言語を提供している．本稿では，鼎システムの構成と実現方式について述べる．

1　はじめに

最近の著しいハードウェアの機能向上と低価格化，X-Windowを代表とする共通基盤としてのウィンドゥシステムの普及によって，マルチウィンドゥ環境下で，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の要求が非常に高まってきている．また，ワークステーションの普及により，OSやプログラミングにあまり深い知識をもたないユーザが増大し，視覚的でわかりやすいユーザインタフェースをもつツールが求められるようになった．

ところが，これらのアプリケーションでは，ユーザとの対話処理部分(以下UI部と略す)の実現に高度なプログラミングが要求され，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，アプリケーションの中心部(たとえばデータベースツールならデータベース自体の制御部分)ではないが，場合によっては中心部を上回る作成コストがかかることになる．

また，UI部は，アプリケーションが一応完成した後でも，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．

こういった問題を解決するために，ウィンドゥシステムには，ツールキットあるいはツールボックス等と呼ばれるライブラリが提供されているのが常である．ツールキットは画面上でマウスで操作するために基本的な部品群(ボタン，メニュー，スクロールバーなど)をアプリケーションから扱えるようにするためのライブラリである．アプリケーションプログラマをウィンドゥを扱うための低レベルの作業から解放し，アプリケーション本来の開発に集中できるようにすることが，ツールキットのねらいである．

しかし，UI部の構築でもっとも手間のかかるのは，画面上で扱う対象物の編集作業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る部分，DTPツールでは紙面レイアウトを設計させる部分などの構築に手間がかかる．

一方，Emacs[4]やHyperCard[10]，Excel[7]のように，システムの核となる機能を，拡張言語を用いて改造，拡張させる，というアプローチがある．たとえばEmacsはテキストエディタの機能をLispによって機能拡張できるようになって

[1]本稿では，**メディア**という用語を「画面上で編集操作が可能な視覚的対象物」という意味で用いることにする．

おり，単にテキストエディタとしての使われ方を越えて，電子ニュースリーダ等のアプリケーションを構築するための基盤システムとして広く使われている．これらのシステムを使ってアプリケーションを構築する際の手間は，単にツールキットを用いた場合と比較して，はるかに少なくてすむ．

しかし，ウィンドゥシステムの機能を十分に活用したアプリケーションでは，取り扱う編集対象(メディア)の種類が多岐にわたる場合が多い．例えばソフトのモジュール管理システムでは，モジュールの関係を操作する部分では，ネットワーク図式(ノードとアークからなる図)を使い，モジュール間のインターフェースを入力する部分ではテーブル形式を使う，といったことが考えられる．ところが，複数のメディアを扱えて，かつ拡張言語を有するシステムは現状では見られない．

このように考えていくと，「もし，必要なメディアについて，その編集機能が部品として提供され，さらに(同一の)拡張言語によって改造，拡張できれば，アプリケーション構築の手間は大幅に削減されるのではないか」という発想が生まれる．われわれのグループで現在開発中のシステム「鼎(かなえ)」は，この発想を実現し，各種アプリケーション構築のための基盤となることを目標としている[2]．

2 鼎システムの構成

図1に，鼎のシステム構成を示す．鼎は，NEC EWS4800 シリーズ(OS は System V+4.3BSD のネットワーク機能)，SUN3 シリーズなどの UNIX ワークステーション上で，X-Window Version 11 Release 2 のクライアントプロセスとして動作する．すべてのアプリケーションの UI 部は，ひとつの UNIX プロセスの中に同居することになる．以下で，図1の鼎を構成するシステム要素について述べる．

2.1 対話部品

対話部品は，対話処理をするための基本的な部品群で，第1章で述べたツールキットに相当する部分である．ボタン，メニュー，リスト(スクロールするメニュー)，ボリューム(スクロールバー)，パレット(アイコンからなるメニュー)，フィールド(1行入力)が用意されている．対話部品は X-Toolkit[1] の Widget[3] として実装されている．

2.2 エディタ部品

エディタ部品は，6種類のメディアを編集するための基本的機能を提供する部分である．鼎では，メディアとしての一般性(特定のアプリケーションに依存しない)を失わない範囲で，アプリケーション構築のために必要であろうと思われる以下の6種のメディアタイプ選定している．

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．

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[3] 等)での経験を基にした．

エディタ共通部分，たとえばスクロール，各種編集イベントのディスパッチ等は，対話部品と同様に Widget として実装されている．

2.3 台紙

台紙は，アプリケーションが開くウインドゥに対応した概念で，台紙の上に置かれている対話部品，エディタ部品の位置やサイズ，対話部品やエ

[2]鼎という名称は，鼎の字がウインドウを支える形に見えることに由来している(つまり象形文字である)．

[3]Widget は X-Toolkit に特徴的な用語で，マウスで操作する画面上の部品のことである．

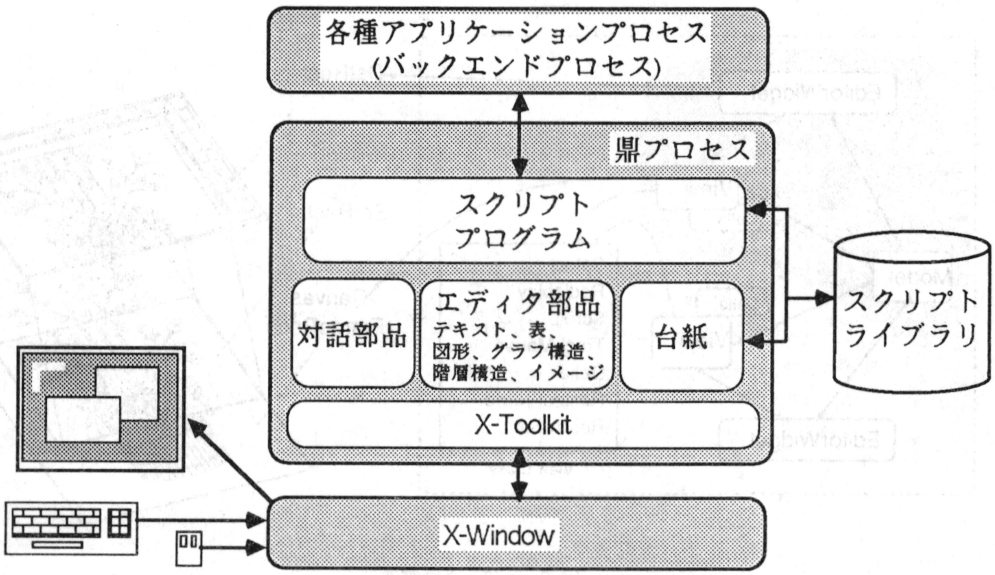

図 1: 鼎のシステム構成

ディタ部品から起動されるスクリプトを保持している.

2.4 バックエンドプロセス

アプリケーションのUI部以外の処理は，アプリケーションごとに作成される別プロセス (**バックエンドプロセス**と呼ぶ) が担当する．バックエンドプロセスは，鼎用に新規開発したものでもいいし，従来のコマンドラインインターフェースを持つツールでもいい．後者の場合，コマンドラインインターフェースを解釈する部分をスクリプトによって記述することになる．これは，既存のツールに対して，鼎を視覚的なUIを持つフロントエンドとして使う方法になっている．もちろん，簡単なアプリケーションの場合は，すべてをスクリプト言語で書き切ってしまうことも可能である.

鼎は全体がイベント駆動の構成になっている．外部から来るイベントはX-Window からのイベントとバックエンドプロセスからの送信がある．X-Window からのイベントはさらにキー入力，マウスによる対話部品操作，マウスによるエディタ部品の画面操作の3種類に大別できる．これらのイベントが発生したとき，対応するスクリプトプログラムが起動される.

3 エディタ部品の構成

次に，鼎に実装されているエディタ部品について説明する．鼎では，最初から複数エディタ機能を実装することがわかっていたので，以下の要求を設計の目標とした.

- それぞれのエディタ部品ごとに記述する部分をできるだけ少なくする．すべてのエディタで共通する部分をできるだけくくりだし，ライブラリ化する.
- ユーザからみた見かけ上の速度を確保する．いかにワークステーションの速度が上がっていても，複雑なメディアでは再表示に要する時間はユーザにとって無視できないほどになってしまうことが予想される．したがってユーザの入力中に無駄な再表示を起動させない，再表示中でも入力があった場合にはできるだけ迅速にその処理にとりかかれるようにする.
- それぞれのメディアを表示編集する機能の組合せで，複数のメディアの混在する文書も扱えるようにする．いわゆる「マルチメディア文書」を扱うために，ひとつのメディアの中に別のメディアが貼り込まれることをサポートしたいが，これをそれぞれのエディタ部品の独立性を保ちつつ実現する.

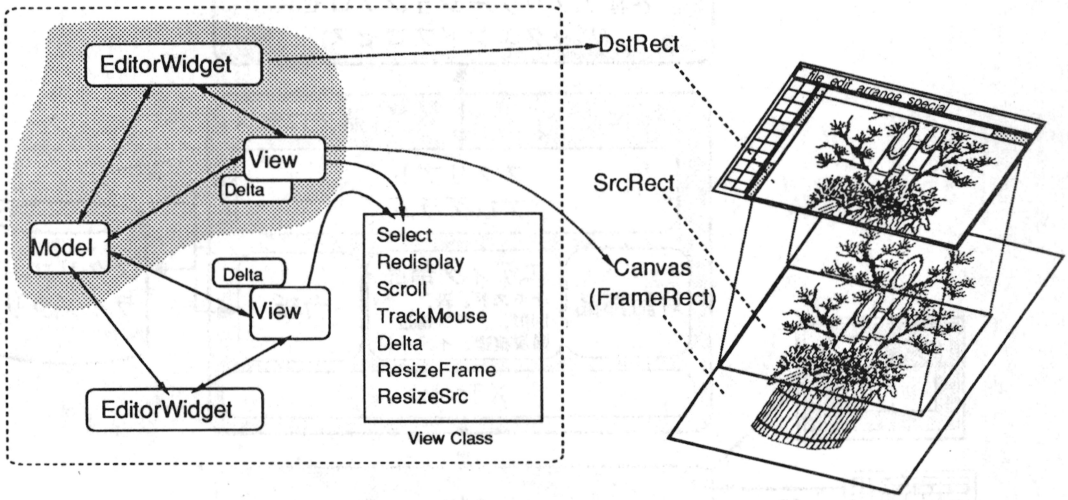

図 2: エディタの構造

- 将来，新しいメディアタイプのエディタ部品を追加することが容易であるようにする．

3.1 モデルとビュー

これらの目標を達成するために，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共通アーキテクチャとして採用している．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[4]．コントローラを X-Toolkit の EditorWidget というクラスの形で実装し，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．EditorWidget(コントローラ) は，すべてのメディアタイプで同じクラスを用いる．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．EditorWidget 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．

ビューはキャンバスと呼ばれる仮想平面上にモデルの内容を投影することを担当している．キャンバス上のある矩形領域が EditorWidget のウィンドゥに対応している (図 2)．

鼎のエディタアーキテクチャにおける MVC モデルの改良点は，モデルが変更したときに直ちにビューに画面を更新させずに，変更履歴だけを蓄積するようにしたことである．ユーザが連続的に入力しているときに，ひとつのキー入力ごとに画面を更新していてはむだな更新が頻繁に起き，対話性をそこなってしまう．鼎のエディタでは，モデルが変更したときに変化情報をビューに通知し，ビューはそれを蓄積するだけで，すぐには画面を更新しない．ユーザの入力がとだえたときに，蓄積してあった変化情報から，最適な画面更新を行う．

[4]ただし，鼎は C 言語で実現されているので，オブジェクト指向の内部の仕組みが表に出てきたような実装になってしまっている．このあたりの状況は，X-Toolkit でも同様である．

変化情報は，画面の効率的な更新に必要な最低限の情報であり，モデルの変化量そのものに対して一般に非常に少なくてすむ．例えば，鼎のテキストエディタでは，モデル (文字列) の一部が別の文字列で置換されたときにビューに送る情報は，置換開始位置，置換終了位置，置換後の終了位置，の三つだけで，置換された文字列自身は送らなくても効率的な画面更新ができるようになっている (図3)．

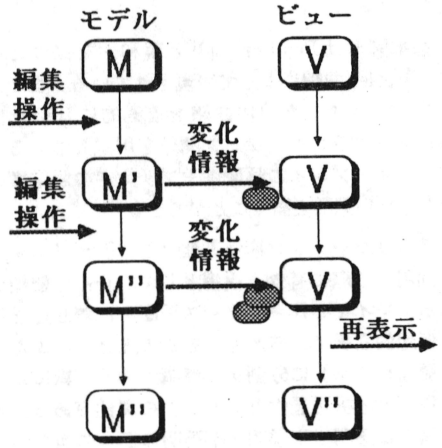

図 3: 再表示の過程

モデルとビューを明確に分離したため，ひとつのモデルに複数のビューを設定すること，ひとつのモデルに異なったタイプのビューを設定すること，が極めて容易に実現できた．前者は編集対象の複数の箇所を別々の画面から操作するときに，後者は精密表示と概略表示のように，編集対象の表示方式がいくつも考えられるときに必要となる機能である．

3.2 属性の添付

エディタをアプリケーション構築のための基盤として使うために，アプリケーション固有の情報をモデルに添付できると便利である．たとえばグラフ構造図をモジュール関連情報の表現として用いるツールでは，グラフ構造のノードにモジュール名や，それに付随する情報を格納したくなる．

そこで，鼎のエディタでは，扱っているモデルの基本要素 (選択できる最小単位) ごとに，属性情報を格納する枠組を用意している．属性情報はタグつきの文字列であり，その使用方法はアプリケーションに委ねられている．属性として格納するアプリケーション依存情報としては，

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

などがある．2. はひとつの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) を実現するための基本的なメカニズムになっている．3. は，つなぎ情報がデータベースに格納されているような場合のリンクの実現方法として使える．

3.3 フラグメント

あるメディアの中への別のメディアの貼り込み，異なるメディア間でのデータ交換を可能にするために，メディアの**フラグメント**という概念を導入している．フラグメントはバイトストリーム形式のデータで，**殻**と**中身**からなっている．中身は，各メディア (モデル) をバイトストリーム形式に変換したものであり，殻に変換したメディアのタイプと表示関数，フラグメントを表示したときの画面上での大きさが記述されている．

たとえばテキストの中に図形を貼り込む場合は，図形をフラグメント形式に変換し，それをテキスト中に挿入する．テキスト中では，このフラグメント全体が一つの文字として扱われる．フラグメントの大きさは殻の記述から知ることができるので，テキスト側ではフラグメントの内容に立ち入ることなく，どの位置にフラグメントを表示するかを決定できる．フラグメントの表示も，殻による記述から使うべき関数がわかるので，それを用いる．

貼り込まれているフラグメントを再び編集するときには，いったんフラグメントの中身をモデルに戻し，それにビューと EditorWidget を与えて編集環境を再構成する．

このように，フラグメントを他のメディアに貼り込んでいるときは，殻に記述されている情報のみを利用し，中身のデータには立ち入らないようになっている．したがって，任意のメディアを別のメディアに自由に貼り込むことができる．あるメディアに対するフラグメントを実現するのに，メディアごとに必要となる作業は，

- フラグメントとモデルの相互変換関数
- フラグメントの表示関数

を実装することだけである．

3.4 各エディタの特徴

以下で，各エディタ部品の特徴的な機能について簡単に紹介する．

3.4.1 テキスト

テキストは文字列を扱うためのもっとも基本的なメディアである．

鼎のテキストエディタ自体は仮名漢字変換機能を持っていないが，鼎と並行して開発している仮名漢字変換サーバ (**いろは**) を利用して変換機能を組み込んでいる．変換まわりのインターフェースルーチンは現在 C で記述され，約 700 行ほどであるが，将来は基本部分を除いてすべてスクリプトで書き直す予定である．

3.4.2 表

表はマトリックス状に配置された文字列を操作するためのメディアである（将来的には，前述のフラグメント機構によって任意のメディアを配置することが可能にする予定である）．

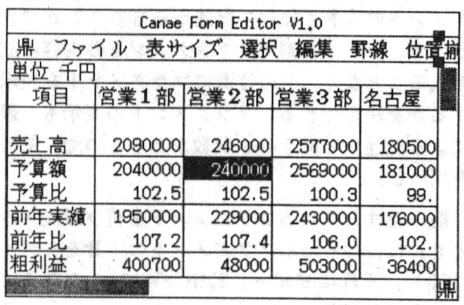

図 4: 表エディタの画面イメージ

表自体はいわゆるスプレッドシートのような値の自動計算能力を持っていない．これは，セルの属性として添付したスクリプトをそのセルに入力があったときに起動させることによって容易にその機能を実現することができるからである．

3.4.3 図形

図形メディアは，線，矩形，円，ポリゴン，自由曲線などの基本図形を組み合せて図形を構成するメディアである (図 5)．それぞれの基本図形は，輪郭の太さや塗りつぶしパターンなどの表示属性を持っている．また，複数の基本図形をグループ化してひとつの基本図形として取り扱うことも可能である．図形メディアは，これらの基本図形をキャンバス平面上に積み重ねていったものになっている．

基本図形は PostScript[8] と親和性を保つために，Bézier 曲線によって表現されている．また，マウスで入力した自由曲線を自動的に Bézier 曲線に変換するプリミティブ関数を用意しているので，ディスプレイの解像度やプリンタの解像度に依存しない図形を描くことができる．

アプリケーション構築基盤として図形メディアを利用する際に有効な機構として，**レイヤ機能**がある．レイヤとはキャンバス平面上に差し込まれた仮想的な紙で，基本図形を紙の上と下のふたつの層（レイヤ）に分割する機構である．紙には表示属性と選択透過性という二つの属性がある．表示属性は透明・半透明・不透明の三つの値のいずれかをとり，値が半透明のときは，紙の下のレイヤに属する図形は灰色に表示される．値が不透明なときは表示されなくなる．紙の選択透過性はオンかオフの値を取り，値がオフのときは紙の下のレイヤに所属する図形に対するマウスによる選択が不能になる．

この機構を利用すると，たとえば帳票形式による入力インターフェースを構築することができる．帳票の枠線と見出し，テキスト（ユーザに入力させる部分）を通常の図形として設計し，枠線と見出しだけを下のレイヤに落す．こうすると，ユーザがデータを入力しているときに操作できるのはテキストフィールドのみになる．帳票のデザインを修正するときは，特別なツールを用意しなくても，単にレイヤを仕切る紙の選択透過属性を変えるだけでよい (図 6)．

3.4.4 グラフ構造

グラフ構造メディアはノードとアークからなる図式を表現するためのメディアである．この図式は図形メディアによって表現できないこともないが，ソフト開発ツール等のアプリケーションに非

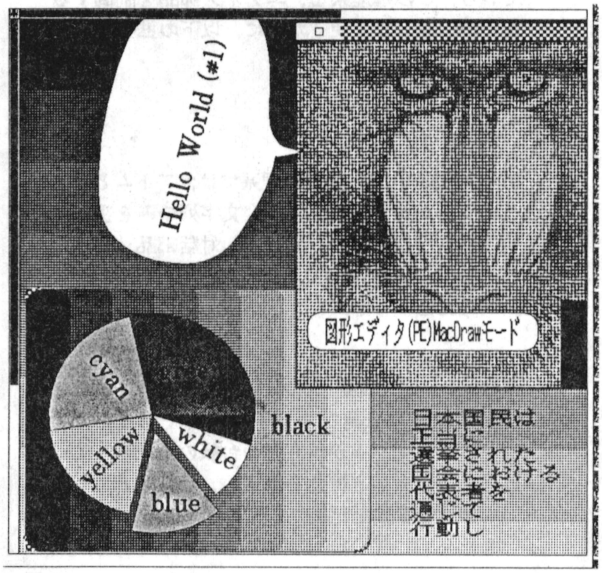

図 5: 図形エディタの画面イメージ

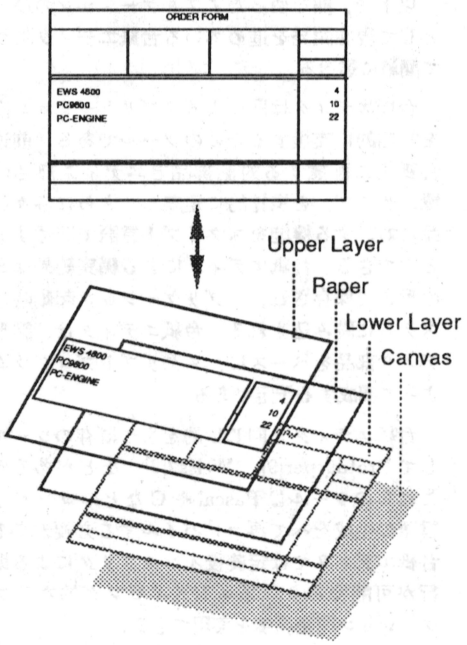

図 6: 図形エディタのレイヤ機能

常によく使われているタイプの図式であること，ノードとアークの接続関係を維持したままでの編集を容易にするために，独立のメディアとして用意した．

グラフ構造図は各種のフロー図，状態遷移図，データ構造図，モジュール関連図，データベースのER図，などを表現するのに利用できる(図7)．

グラフ構造図のノードの形状には図形メディアの基本図形がそのまま利用できるので，用途に応じた形状を容易に作り出すことができる．

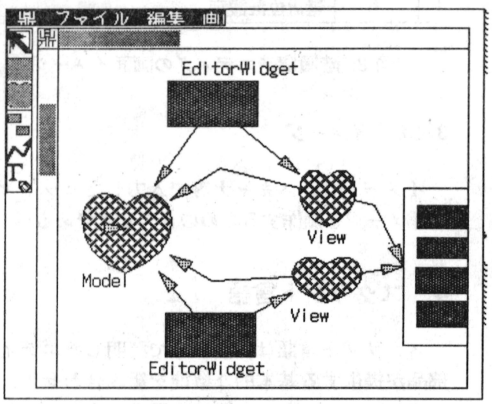

図 7: グラフ構造エディタの画面イメージ

3.4.5 階層構造

階層構造メディアは，いわゆるアウトラインプロセッサやアイデアプロセッサなどと呼ばれているシステムに対応したメディアで，木構造をなし，木のノードにはタイトル文字列と，自分の下に所属している部分木のリスト，そしてノード自体の情報をあらわす他メディアを格納している．他メディアの格納には，前述のフラグメント機構を利用している．

階層構造メディアは，論文やマニュアルのように章立ての明確な文章，プログラムの手順（アルゴリズム），などを表現するのに利用できる(図8)．

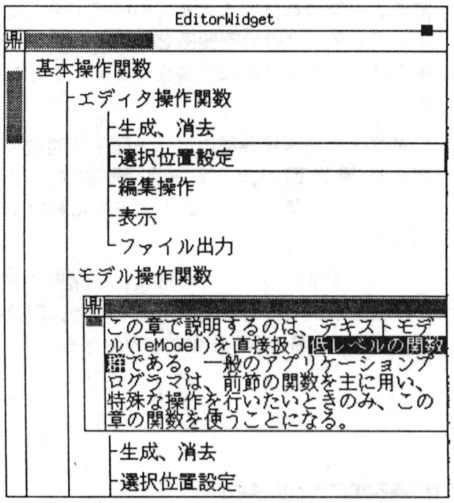

図 8: 階層構造エディタの画面イメージ

3.4.6 イメージ

イメージは，スキャナ等で入力したビットマップイメージを操作するためのメディアである．

4 スクリプト言語

スクリプト言語は，前の章で説明したエディタ部品が提供する基本的な機能を組み合わせて，アプリケーション向けの機能を提供するための言語である．Lisp に準じた言語仕様を持っているインタープリタ言語である．

スクリプト言語によるプログラムは，鼎の環境下で，次の用途に用いられている．

- キーボード入力に対するハンドラルーチン．
- 対話部品のコールバックルーチン．
- モデルに添付される属性．
- EditorWidget 上でのマウスイベントに対応するコールバックルーチン．
- バックエンドプロセスの応答に対応するコールバックルーチン．

スクリプト言語の処理系は，もともと別の目的のために作られたものを流用している．この処理系は PC-UX (PC98 上の UNIX システム．8086 スモールモデルのプロセスのみ動作可能) 上で動作するように作られており，Lisp のひと通りの機能を持ち，かつ非常にコンパクトに実装されている．この処理系を，エディタ部品を制御するための言語として使うために，以下の拡張を施している．

データ構造の拡張 元の処理系では，アトムとしてシンボル，整数，文字，文字列のみを扱っていたが，モデル，ビュー，対話部品，台紙，EditorWidget 等のオブジェクトを扱えるように拡張した．

変数スコープの変更 元の処理系はダイナミックスコープの単純なスコープ制御であったが，「スクリプトが現在どの台紙の上を走っているか」という概念を導入し，台紙ごとにローカルな変数を定義できるようにした．

5 台紙エディタ

以下で，鼎を使ったアプリケーションのひとつとして現在開発を進めている**台紙エディタ**について簡単に述べる．

台紙エディタは鼎によるアプリケーション作成を対話的に支援するためのツールである．前述の台紙上に配置する対話部品とエディタ部品の配置，サイズ，を視覚的に定義し，さらに各々の部品に対応する機能をスクリプト言語で記述することができる．台紙エディタによる編集結果は S 式の形式で保存され，アプリケーション起動時にシステムに読み込まれる．台紙エディタは，図形エディタ部品をベースに，スクリプトプログラムによって構成する予定である．

台紙エディタと同じ目的をもつ既存のツールとして Prototyper[9]，Weiss2[6]，などがあるが，これらのツールは Pascal や C などのコンパイラ言語の生成をへて再コンパイルする必要がある．台紙エディタでは定義後インタプリタによる即実行が可能であり，プロトタイピング的なアプリケーション開発環境を実現できる．

台紙エディタ自体も鼎のアプリケーションであるから，そのユーザインタフェースも台紙エディタを用いて修正できる．すなわち，自分自身をインクリメンタルに改良していくことができる．

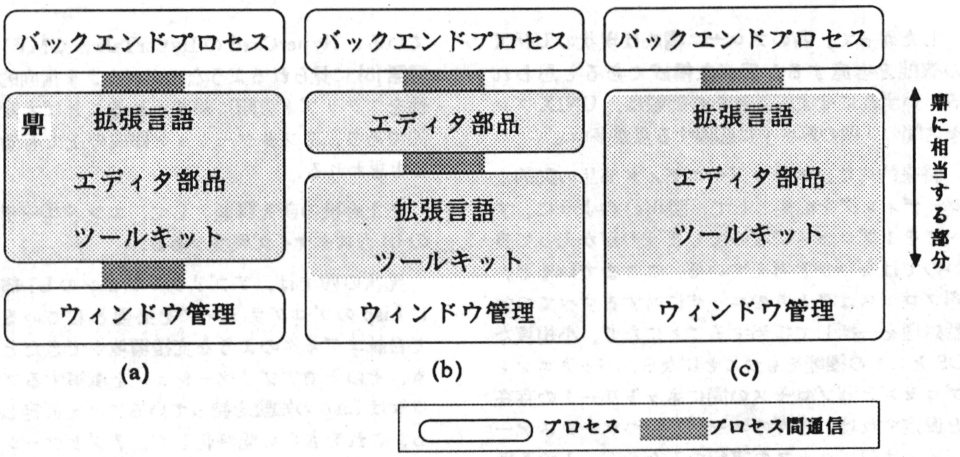

図 9: プロセス構成の比較

6 実現方式に関する考察と今後の課題

6.1 プロセス構成について

鼎では対話部品，エディタ部品，スクリプトインタプリタをすべてリンクして，ひとつのUNIXプロセスとして動作させる方式をとっている(図9(a))．ひとつの鼎プロセスが，同時に複数のアプリケーションのユーザインタフェースを一括して担当するのである．

このような構成をとった理由は以下の通りである．

- 開発に用いたUNIXではライブラリの共有ができないので，個々のアプリケーションごとに個別にライブラリをリンクする方式はプロセスイメージの総量を増大させるので適当でない．
- スクリプトプログラムやデータを複数のアプリケーションで共有させるには同じプロセスイメージ中にすべてのスクリプトが同居する必要があった．
- 実装が単純である．

一方で，この構成をとったために以下のような問題が発生した．

- アプリケーションがすべてのメディアを使わない場合には，その部分が無駄にリンクされていることになる．新しいメディアタイプを扱うためのライブラリを追加する場合には全体を再リンクしなければならない．
- スクリプト言語に並列性がないので，ひとつのイベントに対する処理が実行されているあいだ，他の処理が待たされることになる．

これらの問題は，最近のUNIXで実現されている，共有ライブラリ，動的リンク，スレッド(ライトウエイトプロセス)などの機能を利用して解決可能と思われるので，さらに検討を重ねていきたい．

一方で，GMWやNeWSのようにウィンドゥシステム自体がスクリプトに相当する言語のインタープリタを持ち，その言語によって自由にウィンドゥシステムの機能を拡張していくというアプローチがある(図9(b))．これらのシステムに鼎を実装することを考えてみると，ウィンドゥシステムの持つ言語を鼎の拡張言語として使用するのが妥当であると思われる．実際，これらのシステムではツールキットを拡張言語によって実現している．

しかし，拡張言語がウィンドゥサーバ側にあると，エディタ部品を操作するための言語として用いるのはいろいろと不都合が多い．例えば，文書に属性としてスクリプトを添付している場合，このスクリプトを実行するためには，インタプリタをもつサーバにスクリプトを送らなければならない．といって，拡張言語のインタプリタをサーバとクライアントの両方に持たせるのはいかにも無駄である．

したがって，鼎のプロセス構成は現状のUNIXの機能を考慮すると妥当な構成であると思われる．いずれにせよ，これらの問題は，UNIXプロセス間の「壁の厚さ」に起因する点が多い．

将来的には，スレッドやエディタ部品の動的なローディングを前提として，図9(c)のように，すべてを1プロセスに収めてしまうのがかえって良いのではないかと考えている．ここまでいくと，鼎プロセスはひとりのユーザに対するすべての対話処理を一括して担当することになり，小規模なOSとしての機能をもつことになる．バックエンドプロセスと鼎プロセスの間にネットワークの存在を仮定すれば，各々のユーザが持つワークステーションは鼎プロセスの実行のみをサポートする機能があればよく，OSの大幅な簡略化がはかれることになる．

プロセス構成に関連する今後の研究課題として，**分散環境への対応**がある．鼎プロセスとユーザは1対1に対応していると考えられるが，その間をプロセス間通信で結んで，お互いに相手のプロセス中の文書(モデル)をあたかも自分のプロセス内の文書であるかのようにして扱うことができないか，検討中である．この機構の実現により，たとえばマルチメディア分散会議システムのようなアプリケーションの構築基盤として，鼎を利用することができるようになる．

6.2 スクリプト言語について

鼎のスクリプト言語は，Lispをベースとしている．これは，

- Lisp を拡張言語に用いることの有効性はEmacs や AutoCAD によって実証されている．
- Lispプログラマならば，短期間の内に鼎のスクリプト言語を習得し，システムを構築することができる．
- 台紙の構成記述など，実行環境をS式の形で保存できるので，開発環境が構築しやすい．
- 手元に自由に手を入れられるコンパクトなLispの処理系があった．

等の理由による．

しかし，鼎のように扱う対象物が多岐にわたる場合，用意するプリミティブ関数の個数がどうしても多くなってしまうという問題が発生している．HyperCardのHyperTalk，GMWのG言語[5]に見られるようなオブジェクト指向的な特性をスクリプト言語に採り入れることは大変魅力的であり，アプリケーション作成の上で有効であると思われる．

また今後の研究課題として，**エンドユーザ向けのUIカスタマイズ環境**がある．

現状の鼎では，アプリケーションのUI部定義にLispのプログラミングを必要としている．また台紙エディタのような支援環境ができたとしても，その上でアプリケーションを構築するプログラマはLispの知識を持っていることを仮定している．これをさらに簡略化して，アプリケーションを利用するエンドユーザが，(Lispによるプログラミングを介さずに)コマンドを自分でカスタマイズし登録できる環境について検討中である．具体的には，アプリケーションが提供するコマンドの最小単位 — それは，ひとつのキー入力，対話部品のボタンやメニューの項目に対応している — を視覚化し，グラフ構造エディタ部品をベースとしたコマンドフロー定義ツールによって，コマンド実行のシーケンスや，条件分岐によるコマンド実行の切りわけを定義していく．定義したコマンドはメニューやボタンに自由に貼り込むことができる．

7 むすび

鼎の開発状況は，1988年11月現在で，対話部品，各エディタ部品およびスクリプト言語インタプリタの第1版が完成し，全体をひとつにまとめるための組み込み作業中である．システムのおおよその規模は，対話部品が6K 行，エディタ部品が33K 行，スクリプトインタプリタが8K 行程度である(いずれもCのソース行数)．

また，中間段階として，対話部品とエディタ部品を単なるCライブラリとして使用する(スクリプトを使わない)方法で，いくつかのアプリケーション(工程管理システム，ジョブフロー設計支援システム等)の開発が別グループによって進行中である．

今後は機能の充実をはかると共に，実際のアプリケーションによって，その実用性を評価していきたい．

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，プロジェクト発足時のメンバであり，鼎の基本設計に関する議論に熱心に参加していただいいた竹内章平氏，小沼千絵氏（日本電気C&C共通ソフトウェア開発本部），スクリプト言語の元となったコンパクトで高品質なLisp処理系の開発者である佐治信之氏（日本電気C&C共通ソフトウェア開発本部），および社内関連部門各位に深く感謝致します．

参考文献

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

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

[3] 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.

[4] Richard Stallman, "*GNU Emacs Manual, Sixth Edition, Emacs Version 18*", Free Software Foundation, 1987.

[5] 大谷浩司，児島彰，他「ウィンドウ・システム上のアプリケーション構築について」，日本ソフトウェア科学会第5回大会予稿集，1988.

[6] 松浦敏雄，杉本直樹，「X Window 上のユーザインタフェースプログラムの自動生成」，第12回 UNIX シンポジウム予稿集，1988.

[7] マイクロソフト コーポレーション，「Microsoft Excel ユーザーズガイド」，1986.

[8] Adobe Systems Incorporated, "*PostScript Language Reference Manual*", Addison-Wesley, 1985.

[9] SmethersBarnes, "*PROTOTYPER manual*", 1988.

[10] Danny Goodman, "The Complete HyperCard Handbook", Bantam Books, 1987.

本 PDF ファイルは 1989 年発行の「第 30 回プログラミング・シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトに、下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載し、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

https://www.ipsj.or.jp/topics/Past_reports.html

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間： 2020 年 12 月 18 日 ～ 2021 年 3 月 19 日
掲載日： 2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html