入選

エディタを部品としたユーザインタフェース構築基盤：鼎†

暦本純一††　垂水浩幸††　菅井勝††
山崎剛†††　猪狩錦光††††　森岳志††
杉山高弘††　内山厚子††　秋口忠三††

ウィンドウシステムの普及にともない各種のメディア（画面上で編集操作が可能な視覚的対象物）を活用した視覚的ユーザインタフェースをも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，6種のメディア（テキスト，イメージ，図形，グラフ構造，表，階層構造）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．鼎システムは，6種のメディアを編集するための基本機能をエディタ部品としてもち，エディタの機能を目的に応じて変更・拡張するためのカスタマイズ言語を提供している．MVC モデルに基づくオブジェクト指向的なエディタ実装方式を採用し，各メディアの混在が容易に行えるようになっている．エディタのユーザインタフェースを変更するときは，スタック構造をもつイベントマップ（マウス入力に対応）やキーマップ（キー入力に対応）にアプリケーションが定義したマップをプッシュして，標準の編集操作を上書きすることができる．編集対象物とアプリケーションのデータを関連づけるための機構も提供している．以上の方式に基づいて鼎システムを実装した．鼎を利用して作成した実際の CASE アプリケーション3例を調査した結果，システムの開発規模が，鼎の利用によって半分以下に削減されることが分かった．

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ウシステムの普及によって，マルチウィンドウ環境下で，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，視覚的で分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，使いやす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，ウィンドウシステムには，ツールキットあるいはツールボックスなど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ウ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をダイレクトマニピュレーションで操作，編集する部分である．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[1] や HyperCard[2] 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

† Canae: A High-Level Software Platform for User-Interface Development Using Editor Parts by JUNICHI REKIMOTO, HIROYUKI TARUMI, MASARU SUGAI (Software Architecture Department, Software Engineering Development Laboratory, NEC Corporation), Go YAMAZAKI (NEC Microcomputer Technology, Ltd.), KANEMITSU IGARI (NEC Scientific Information System Development Corporation), TAKESHI MORI, TAKAHIRO SUGIYAMA, ATSUKO UCHIYAMA and CHUZO AKIGUCHI (Software Architecture Department, Software Engineering Development Laboratory, NEC Corporation).

†† 日本電気(株)ソフトウェア生産技術開発本部基本方式開発部
††† 日本電気マイコンテクノロジー(株)
†††† 日本電気技術情報システム開発(株)

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おり，単にテキストエディタとしての使われ方を超えて，電子ニュースリーダなど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，ウィンドウ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．たと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．われわれが現在開発中のユーザインタフェース構築システム「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．鼎は，CASE システムを第一の応用分野と考えており，一般のユーザインタフェース構築システムで扱うテキスト・図形・イメージに加えて，CASE システム構築で必要となるグラフ構造・階層構造・表などの論理的な構造をもつメディアの編集機能を部品として提供している点に特徴がある．

以下，2.～4. で鼎システムの構成について述べ，5. で鼎を利用してアプリケーションを構築した場合の UI 作成効率の向上について評価する．6. では鼎の方式についての議論を行う．

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，UNIX ワークステーション上で，X-Window[3] のクライアントプロセスとして動作する．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，1. 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（アイコンをマトリックス状に配置したメニュー），フィールド（1 行入力），パネル（ダイアローグボックスのように，複数の対話部品をまとめるための部品）が用意されている．対話部品は X-Toolkit[4),5)] の Widget* 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を選定している．

（1） テキスト 文字列（日本語）を編集するため

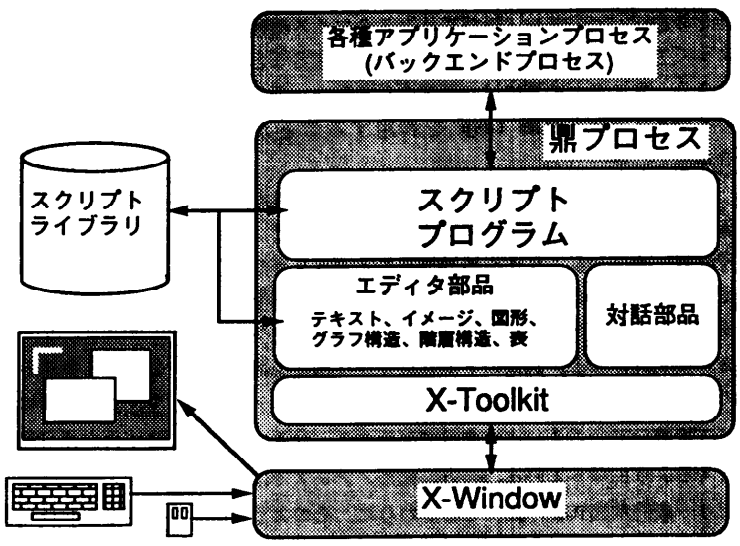

図-1 鼎のシステム構成
Fig. 1 The architecture of the Canae system.

* 鼎という名称は，鼎の字がウィ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）．

* 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．

の部品．かな漢字変換機能を含んでいる．他メディア上の文字列（たとえば表のセル中の文字列）を編集するためにも用いることができる．

（2） **図形** 基本図形の組合せによる図．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．

（3） **イメージ** スキャンデータなどのビットマップ．

（4） **グラフ構造** ノードとアークの組合せによる図式を編集するための部品．ノードの形状は，スキーマファイルの形で，アプリケーションの分野に合わせて定義することができる．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フェースを構築するために用いる．

（5） **階層構造** 木構造のデータを編集するための部品．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．構造化フローチャートや，操作マニュアルのように章立ての明確な文章を作成するために用いる．

（6） **表** マトリクス上に配置された文字列や他メディアを編集するための部品．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分かりやすいものに使える．

鼎では，これらのメディアを編集するための基本機能を部品として用い，後述するカスタマイズ機構によってアプリケーション向けのUIを構成することができる．カスタマイズには，通常のC言語と，鼎専用のインタプリタ言語**鼎 Lisp** を利用することができる．

6種の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CASEツール（SDMS[6]など）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こと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．

2.3 バックエンドプロセス

鼎でアプリケーションを構成する方式には，すべてをひとつのプロセスに組み込んでしまう場合と，UI以外の処理を別プロセスに分担させる場合とがある．後者の場合の別プロセスのことを**バックエンドプロセス**と呼んでいる．バックエンドプロセスは，(1)既存のコマンドラインインタフェースをもつツールに視覚的なインタフェースを付加するとき，(2)鼎Lispを利用している場合に，複雑な処理（インタプリタでは性能に問題が出るような処理）をバックエンドに分担させるとき，に有効な方式である．

鼎は全体がイベント駆動の構成になっている．外部からくるイベントは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，後述するキーマップ，イベントマップに登録されているアプリケーション定義関数（Cか鼎Lispで定義されている）に処理が渡される．アプリケーション定義関数は，必要ならバックエンドプロセスにリクエストを発行することができる．

3. エディタ部品の構成

ここでは，鼎に実装されているエディタ部品について説明する．

3.1 設計目標

鼎で実装されているエディタ部品は，通常の閉じたエディタとは異なり，UI構築の基盤部品として使われることが特徴である．また，複数のメディアをサポートしているので，メディア間での統一性を保つことも重要である．具体的には，以下の要求を設計目標として開発した．

- アプリケーションの必要に応じて柔軟に機能を拡張できる枠組みを用意する．たとえばキーボード入力やマウス処理は，すべてアプリケーションの必要に応じて再定義可能であるようにする．また，アプリケーションに依存するデータとエディタが扱う編集対象との関連を取りやすくする機構を用意する．
- それぞれのエディタ部品ごとにインプリメンタが記述しなければならない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．

●エンド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．

●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．

3.2 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**[7] 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[8] 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とき，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．コントローラは 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．モデルに対する操作のうち，メディア共通なもの（たとえばカット・コピー・ペーストなど）は，メディアごとに定義されているメディア記述子を経由して呼び出すようにしている（モデルオブジェクトのクラス定義に相当する）．

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．キャンバス上の矩形領域が EditorWidget のウィンドウに対応している(図-2)．

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，がきわ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．

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ただ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の列と変更後の

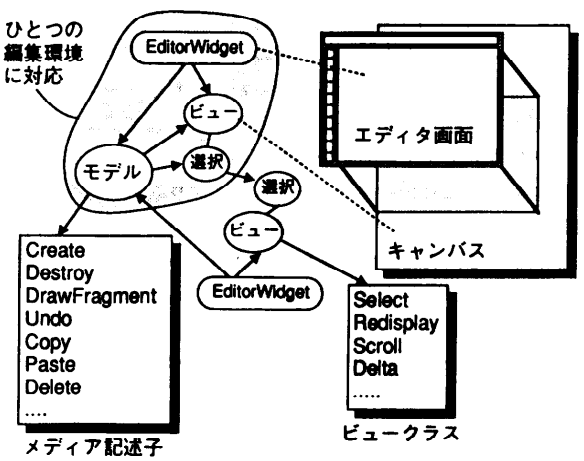

図-2 エディタ部品の構造
Fig. 2 The architecture of the Canae editor parts.

モデルから，画面を最新状態に更新する．ユーザからの入力の有無は，X-Windowから受け取るイベントを溜めておくイベントキューの内容から判断することができる．

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．たとえば，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，これを記録しておけば，再表示が起動されたときには，変換後のモデル，変換前のビューデータ（行の折り返し情報など），変化情報の三つからビューデータと画面内容を更新するようになっている．

この機構は，モデルへの操作のように内部から発生する変化だけではなく，ウィンドウの順位変更で画面の隠されていた部分が露出する場合やスクロール処理など，モデル内容は変化しないが画面に表示すべき領域が新たに発生する場合にも利用できる．たとえば画面をスクロールバーでスクロールさせる場合を考えてみる．マウスでスクロールバーの「つまみ」をつかんで画面をスクロールさせるとき，マウスをすばやく動かし続けているとイベントキューに「マウス移動」のイベントが溜っているので画面の再表示は抑制される．マウスを静止させる（マウスボタンを放す必要はない）と，その時点で再表示が起動される．

3.3 カスタマイズ機能

各メディアを編集するための基本的な機能は鼎のエディタ部品として組み込まれているが，それをベースとしてアプリケーションを構築するために，鼎ではエディタにカスタマイズ機能を与えている．

各エディタに対するキーボード入力，マウス入力に対応して，それを入力の処理関数（C か Lisp によって記述される）に対応づけるための表（キーマップ，イベントマップ）が定義されている．これらの表は，スタック構造をもち，新しい定義表を上からプッシュすると，下の表で定義された対応はオーバライドされる．たとえば下の表にマウスのダブルクリックとリリースに関する処理が定義されているときに，ダブルクリックに関する別の処理を定義した表をプッシュすると，ダブルクリックは上の表の定義が，リリースは下の表の定義が有効となる（図-3）．

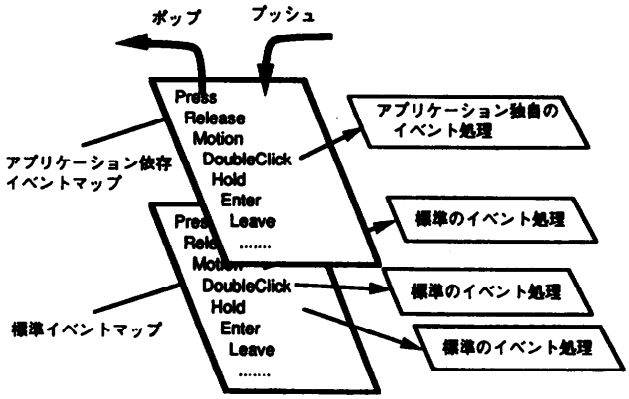

図-3 イベントマップの構造
Fig. 3 The structure of the Event-Map.

エディタとしての基本的な操作は，標準キーマップ，標準イベントマップの形で定義されている．これらのマップは起動時に自動的にエディタに設定される．アプリケーション固有の操作は，この表の上に別の表をプッシュしていくことによって実現できる．標準イベントマップで定義されている操作は，オブジェクトの選択や移動処理など，編集操作の基本となるもので，オブジェクトを直接マウスでつかみ，操作できるような感覚を与える（ダイレクトマニピュレーション）UI を提供している．

たとえば，グラフ構造の各ノードをアプリケーション内部で管理しているモジュールに対応づけているアプリケーションでは，グラフ構造上でのダブルクリックに対して，

（1） ダブルクリック時のマウス位置から，対応するノードを求める．

（2） そのノードに対応するモジュールを求める．

（3） そのモジュールに関するデータを，別のウィンドウに表示する．

といった処理を定義することができる．ノードとモジュールを対応づけるためには，次の節で説明する属性情報を利用することができる．

3.4 属性情報の添付

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できると便利である．

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みを用意している．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の用途としては，

（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用いることができる．

3.5 メディアの混在

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**というデータ構造を導入している．フラグメントはモデルをファイルに格納したり，ネットワーク経由で送信するための共通フォーマット（バイトストリーム形式）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，いくつかのブロックの列からなる．殻にはメディアタイプとフラグメントを表示したときの画面上での大きさが書きこまれている．フラグメントタイプからは，メディア記述子（**3.2** 参照）をたどることができる．

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ひと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分かるので，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．編集後は，再びモデルをフラグメントに変換して親のモデル内に格納する．

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．また，将来新しいメディアタイプが追加されたときにも，その影響が他のメディアの処理に及ぶことがない．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と，フラグメント表示関数を実装することだけである．なお，フラグメント形式をそのままファイルに書き出したものが，鼎のファイルフォーマットになっている．

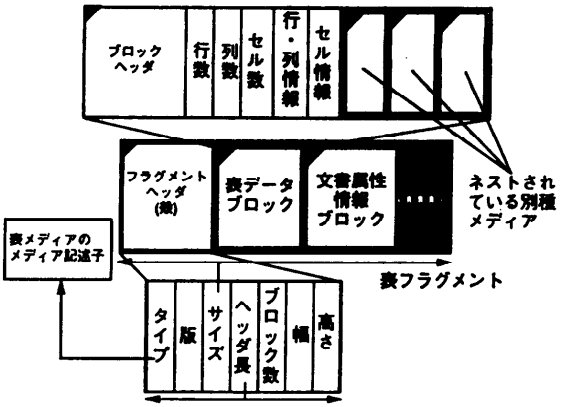

図-4 フラグメントデータの例
Fig. 4 An example of fragment data.

図-4 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．

4. スクリプト言語（鼎 Lisp）

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，Lisp ベース のインタプリタ言語である．スクリプト言語によるプログラムは，鼎の環境下で，（1）イベントマップ，キーマップに設定するコールバック関数，（2）対話部品のコールバック関数（画面上のボタンを押したときの処理など），（3）バックエンドプロセスの応答に対応するコールバック関数，などの用途に用いられる．

エディタの拡張言語として，Lisp を採用したのは，（1）Lisp を拡張言語として用いることは，Emacs での経験から有効であることが実証されている，（2）画面の構成記述など，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，（3）鼎用に新規の言

```
;;; グラフ構造エディタの要素にラベルを付加
(defun edit-label (editor arg event-type x y event)
  (let* ((s x (CeDst X ToSrc editor x))
         (s y (CeDst Y ToSrc editor y))
         (element (GeSearchElement editor s x s y))
         (label (GeCreateLabel
                  editor "label" element s x s y 100 50)))
    (GeEditLabel editor label nil)))
;;; グラフ構造エディタに，マウスプレスによって
;;; ラベル文字列を付加するためのイベントマップ
;;; をプッシュする．
(CxOverrideEventMap
 editor
 ;; プレスに上の関数をバインドしたイベントマップ
 (CxCreateEventMap CX_PRESS 'edit-label nil)
 NoEventMask)
```

図-5 鼎 Lisp による記述例
Fig. 5 An example of Canae-Lisp program.

語シンタックスを考案するよりも，既存の言語の枠組みを利用したほうがアプリケーションプログラマにとって馴染みやすい，などの理由による．

鼎では，Lisp をエディタ部品を制御するための拡張言語として使うために，C言語インタフェース（鼎の提供するプリミティブ関数を Lisp から呼び出す），データタイプの拡張（モデル，ビューなど，鼎を構成しているオブジェクトタイプを扱うため）などの機能拡張を行っている．

たとえば図-5 は，マウスのボタンを押すことによってノードやアークにラベル文字列を付加するイベントマップを生成してグラフ構造エディタに設定しているスクリプトプログラムの例である（大文字で始まる関数が，鼎特有の Lisp 関数である）．

5. 評 価

鼎システムは，1989年9月現在で日本電気社内の30以上の部門に配布され，ワークステーション上の各種アプリケーション構築に利用されている[9)~12)]．鼎を利用したときのソフト開発への効果を評価するために，その一部である CASE ツールに関する利用実績を3例について調査した．これらは，いずれも新規開発分（鼎を含まない部分）が数 10K ステップの中規模アプリケーションである．ただし，3例とも，鼎 Lisp や バックエンドプロセスは利用しておらず，鼎のCライブラリ層の上に直接アプリケーションプログラムをCで記述する開発方法をとっている．

例1はプロジェクト管理者のためのツールである．これはターゲットシステムの階層的構造，プロジェクト構成の階層的構造，プロジェクトスケジュール，要員の人員構成，各要員の進捗状況などを管理者が容易に把握できるようにしたものである．調査時点において，この例は階層構造エディタを含まないバージョンである鼎 V0R0 を使用しているため，グラフ構造エディタによってシステム階層図，プロジェクト階層図を表示するような実装になっている．要員名簿は表エディタを利用して表示される．スケジュールを示す水平バーチャート（いわゆる線表）の UI は，表エディタと図形エディタを利用して実装されている．図-6 に，例1のツールを実行中の画面例を示す．

例2は上流工程用の CASE ツールである．システム機能・データ構造の階層を階層構造エディタで，機能間の関係と業務フローをグラフ構造エディタで取り扱っている．

例3は中流工程用の CASE ツールである．グラフ構造エディタを使ってプロセスフロー図，ジョブフロー図を取り扱う．

表-1 に各ツールにおける鼎の使用実績を示す．(A)は各ツールで独自に開発した部分，(B)から(H)までは鼎のライブラリから各ツールにリンクされた部分，(I)から(K)まではXのライブラリから各ツールにリンクされた部分である．数値は(A)～(K)の総和を100% とした場合の，各部分の割合であり，ソースコード（C言語プログラム，ヘッダファイルとコメン

表-1 実アプリケーションにおける鼎の使用実績
Table 1 Amount of Canae's use in the real applications.

	例1	例2	例3
鼎のバージョン	V0R0	V0R1	V1R0
Xのバージョン	X11R2	X11R2	X11R2
(A) 新規開発部分	29.8%	18.9%	23.7%
(B) 鼎基本ライブラリ	2.6%	5.2%	8.2%
(C) 対話部品	5.8%	7.7%	6.5%
(D) かな漢字変換	5.4%	7.3%	5.7%
(E) テキストエディタ	0.0%	3.2%	3.2%
(F) グラフ構造エディタ＋図形エディタ	11.0%	17.7%	21.6%
(G) 表エディタ	7.5%	0.0%	0.0%
(H) 階層構造エディタ	0.0%	4.0%	0.0%
(I) Athena Widgets	10.4%	4.3%	6.3%
(J) X Toolkit	13.2%	15.1%	11.6%
(K) Xlib	14.3%	16.6%	13.3%
(L) $\frac{B+C+\cdots+H}{A+B+\cdots+H}$	52%	70%	66%
(M) $\frac{E+F+G+H}{B+C+\cdots+J}$	33%	38%	39%

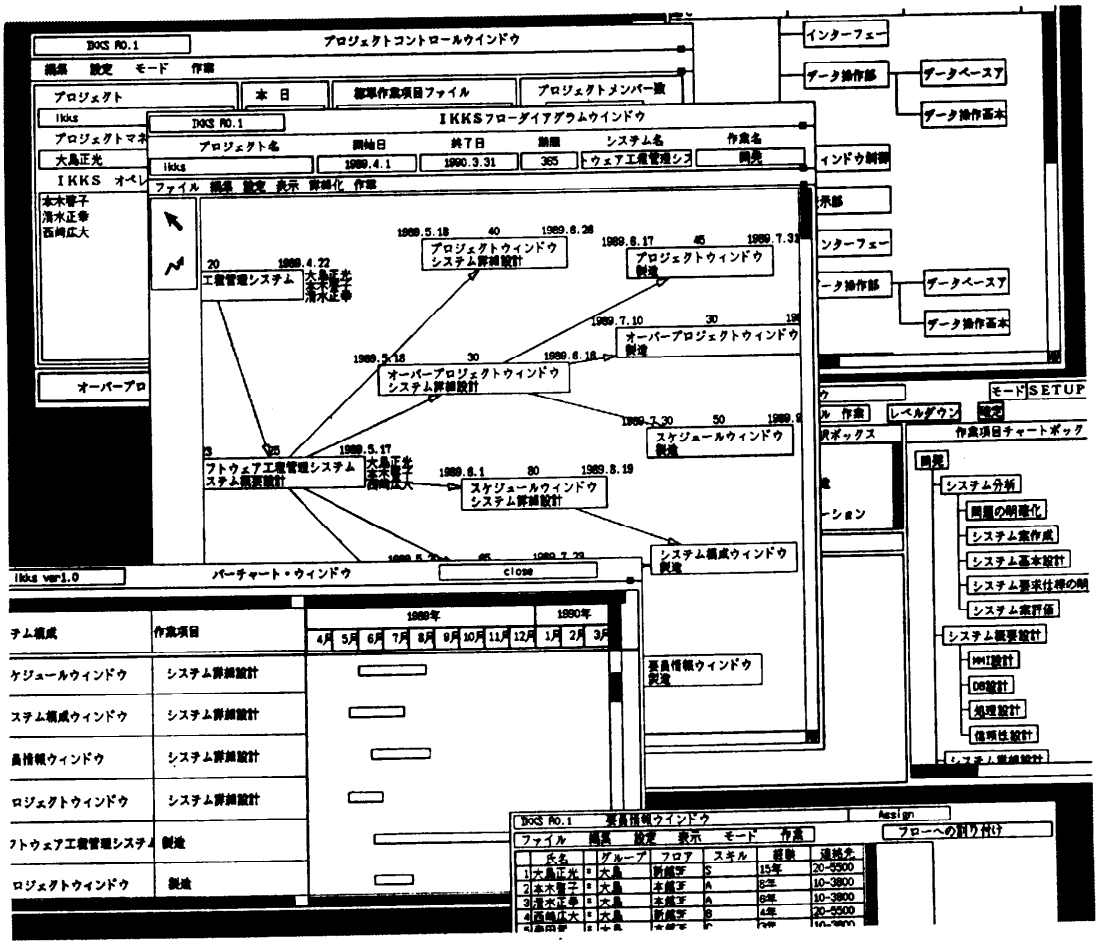

図-6 鼎を利用して構築したシステムの画面例
Fig. 6 A snapshot of Canae-based application.

トを含む）の行数をベースにして計算してある．なお，図形エディタを単独で用いた例はないので，グラフ構造エディタ（常に図形エディタとともに用いられる）との合算で示した．

例2では(A)の比率が小さくなっているが，これは例2の新規開発部分のサイズそのものが小さいためである．例1，2，3の(A)の絶対値比は 1.8：1.0：1.6 程度である．一方，鼎自身は後の例ほどバージョンアップにより大きくなっている．(B)から(H)までの総合計（リンクされなかったモジュールも含む）の比率は 1.0：1.3：1.9 程度である．

(L)は新規開発部分と鼎の利用部分の合計に対する後者の割合を示している．すなわち，ツール全体の開発規模に対する鼎担当部分の割合を示している．この数字が大きいほど，鼎利用による工数削減効果が大きいといえる．リンクされていても実際には利用されない関数や機能が勘定されているため，この数字がそのまま規模削減率だとはいえない．しかしこれらの例に関するかぎり，開発効率が少なくとも2倍程度に向上したといっても過言ではないであろう．なお，例1においては鼎V0R0を用いているため(L)の値が低くなっており，V1R0を用いればさらに向上するはずである．

また，今回の調査では定量的なデータは得られなかったが，鼎を利用することでアプリケーション設計時の負担が軽減しているのではないかと予測している．従来のアプリケーション設計では，一般的な機能（たとえばノードをマウスで移動させるときの操作）もすべて設計者が考えなければならなかった．鼎を利用する設計者はアプリケーション固有の機能，UI の設計

に注意を集中すればよく，一般的な編集操作は鼎の機能に委ねることができる．

(M)はUI部全体に対するエディタ関係部分の割合である．ここでは，(I)と(J)もUIの一部であると考えている．(K)はXウィンドウのクライアント・サーバ間の通信を実現する部分であるので除外した．これらのCASEツールの場合，エディタ部品担当部の規模が全UIの3割～4割程度であり，この部分を部品化した効果は十分にあったといえる．

6. 議 論

UIMSとの比較

従来のUIMSでは，ユーザからの入力の管理，アプリケーションの計算結果の表示，が主な担当範囲であり，編集操作（編集対象のデータ管理を含む）は，アプリケーションの担当範囲であった．たとえばHartsonらの調査報告[13]を見ても，編集対象の管理をも担当するUIMSは見当たらない．この点で，鼎は従来型のUIMSとは異なっている．しかし，編集対象物の管理はUIと密接な関係をもっているので，これをすべてアプリケーションの担当とするのは実際的でない．たとえば図形オブジェクトをマウスで選択し，マウスボタンを押しながら引きずるという操作（ドラギング）は，図形オブジェクト自体の管理と密接な関係をもっているので，UIMSが図形オブジェクトを管理する必要がある．

一方，UIMS側が編集対象を管理してしまうと，アプリケーションのもつデータと編集対象との間の整合性をどうやって保つかが問題となる．現状の鼎では，3.3で述べたイベントマップ・キーマップの機構によって，編集対象を変更しようとする入力操作のレベルでアプリケーションに制御を移せるので，ここで編集対象に施される変更をアプリケーション側のデータに波及させることができる．しかし，この方式では編集対象を内部のスクリプトプログラムなどが（ユーザの操作を経由しないで）直接変更する場合には対処できない．そこで，編集対象を変更するプリミティブメソッドに，アプリケーションが定義するフック関数を差し込めるような機構の導入を検討中である．この機構を使うと，モデルに対して及ぼされるあらゆる変更をアプリケーションが監視し，場合によってはその変更を禁止することも可能になる．

適用範囲

鼎の方式は，アプリケーションがユーザに見せるオブジェクトは6種のメディアの組合せで表現できることを仮定している．もちろん，これはすべてのアプリケーションに当てはまるわけではない．たとえば音声を扱うアプリケーションを構築する場合，鼎を使うことにより得られるメリットはあまりないかも知れない．このように，鼎は汎用型のUIMSというよりは，ある領域のアプリケーションの作成を簡便化するためのシステムであるといえる．逆に，その領域内のアプリケーションであれば，従来のUIMSよりも効果が大きい．鼎は，主にCASEツールやOAツールの作成効率を改善することを目標として作成した．この範囲のシステムは，表示内容を鼎の提供するメディアの組合せで表現できるものが多く，操作インタフェースも鼎の提供するカスタマイズ機構で対応可能である．したがって，この目標は十分に達成できたと考えている．

一方，たとえばDTP（卓上出版）システムを鼎で構築することを考えてみる．鼎のテキストエディタは文字の割り付けをDTPレベルの品質で行う機能は提供していないので，割り付けイメージで直接編集する，いわゆるWYSIWYGなDTPシステムを構成することはできない．しかし，章立てのある文章を階層構造エディタで入力し，用紙上のどこに文章を流し込むかの指定を図形エディタ（またはグラフ構造エディタ）で指示し，整形処理自身はバックエンドプロセスに委ね，割り付けのプレビューイメージをイメージエディタで表示するようなDTPシステムを構築することは可能である．

適用できるアプリケーションの領域を拡大していくためのアプローチのひとつとして，編集操作のカスタマイズ機構のみならず，メディアそのものの拡張機構，カスタマイズ機構を提供することが考えられる．しかし，スクリプト言語でメディアの表示メソッドを定義する際の性能など，現状ではまだ不明な点も多い．これらの機構の導入は今後の研究課題である．

アプリケーション間でのUIの統一

複数のアプリケーション間でのUIが統一されていることは，アプリケーションごとに新たに使用法を学ぶ労力が軽減されるという点で非常に大切である．坂村は，アプリケーション間でのUIを統一させるためには，メタファの活用，イントリンシック（基盤部品）の提供，標準アプリケーションの提供，の三つの

方法があるとしている[15]．鼎では，従来のツールキットよりも高いレベルのエディタ部品を用意し，基本となる編集操作は，標準キーマップ，標準イベントマップの形で与えられているので，各種のアプリケーションでの編集操作インタフェースは必然的に似通ったものとなる．これは編集操作レベルでのイントリンシックをアプリケーションに提供しているものだといえる．

7. おわりに

エディタを部品としてユーザインタフェースを構築するための機構を提案し，それに基づくシステム「鼎」を実現した．鼎を利用した実際のアプリケーションを調査して，本システムの方式によってユーザインタフェースの開発効率が向上することを確認した．

今後は機能の充実をはかるとともに，鼎 Lisp，プロセス間通信機構を利用した際の開発効率の向上についての検証をすすめていきたい．

現状の鼎では，アプリケーションの UI 部定義に C や Lisp でのプログラミングを必要としている．現在，鼎 Lisp を用いてアプリケーション開発者向けの視覚的な UI 定義環境を作成中であるが，これに関する報告は今後行っていきたい．UI 定義をさらに簡略化し，エンドユーザが（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の構築は今後の研究課題と考えている．

謝辞 鼎の初期の基本設計に関する議論に熱心に参加していただいた竹内章平氏，小沼千絵氏（日本電気 C & C 共通ソフトウェア開発本部），スクリプト言語のベースとなったコンパクトで高品質な CI-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いたします．

参考文献

1) Stallman, R. M.: Emacs: The Extensible, Customizable, Self-documenting Display Editor, in Proc. ACM SIGPLAN/SIGOA Conference of Text Manipulation (1981).
2) Goodman, D.: The Complete HyperCard Handbook, Bantam Books (1987).
3) Scheifler, R. W. and Gettys, J.: The X Window System, ACM Trans. Graph. Vol. 5, No. 3, pp. 79-109 (1986).
4) McCormack, J. et al.: X Toolkit Intrinsics—C Language X Interface, X Window System, Version 11, Release 3 (1988).
5) Weissman, T. et al.: X Toolkit Widgets—C Language X Interface, X Window System, Version 11, Release 3 (1988).
6) 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 技報，Vol. 40, No. 1, pp. 10-18 (1987).
7) Cunningham, W.: Smalltalk-80 によるアプリケーション・プログラムの作り方，bit, Vol. 18, No. 4, pp. 379-396, 共立出版 (1986).
8) Goldberg A. and Robson, D.: Smalltalk-80—The Language and its Implementation, Addison-Wesley (1983).
9) 櫨山淳雄，原田勝利他：ソフトウェア工程管理システム IKKS (2)—概要と特徴—情報処理学会第 39 回全国大会，pp. 1411-1412 (1989).
10) 杉山高弘，吉田宗弘他：例文からの文法獲得に基づく日本文インタフェース構築ツール「ゆい」，情報処理学会自然言語研究会，89-NL-73, pp. 9-16 (1989).
11) 平野文康：マルチメディア文書 DB の利用者インタフェース，情報処理学会データベースシステム研究会，70-5 (1989).
12) 渡部和雄，阪田史郎他：マルチメディア分散在席会議システム MERMAID，電子情報通信学会オフィスシステム技報，OS 89-27 (1989).
13) Hartson, H. R. and Hix, D.: Human-Computer Interface Development: Concepts and Systems for Its Management, ACM Computing Surveys, Vol. 21, No. 1, pp. 5-92 (1989).
14) Young, R. et al.: Software Environment Architectures and User Interface Facilities, IEEE Trans. on Softw. Eng., Vol. 14, No. 6, pp. 697-708 (1988).
15) 坂村 健：BTRON におけるヒューマン・インタフェース設計手法，電子情報通信学会論文誌，Vol. J 70-D, No. 11 (1987).

（平成元年 8 月 31 日受付）