『維摩』システムにおける文書処理

諸橋正幸　穂積元一　吉永秀志

日本アイ・ビー・エム株式会社　サイエンス・インスティチュート

１．はじめに

オフィスシステムのためのワークステーション制御システム『維摩』を使用した複合オブジェクト・エディターを試作中であるが、このエディターの設計、構築について『維摩』特有の性質の利用のし方について述べる。

従来マルチウィンドウ・システムでは、既存のアプリケーションをそのまま使用することが、なかなか困難であった。これは次の理由によるところが大きい。

ａ．まったく新しい言語又はアーキテクチャを採用したために従来のプログラムが使えない。

ｂ．言語又はアーキテクチャは同じでも、マルチウィンドウ・システムという枠の中で従来のものが動作しないために、結局全体の作りかえになってしまう。

これらの弊害を除いて、従来の言語で書かれた既存のアプリケーションをも柔軟にマルチウィンドウ・システムの中に取り込めるようにしたものが『維摩』である。更に『維摩』システムでは独自の機能を提供しているため、これらを使用して統合的なマルチウィンドウ・システムが構築し易くなっている。本稿では、まず『維摩』システムの目的と特徴を述べ、次にこれを使用した複合オブジェクト・エディターの設計、構築について述べる。

２．『維摩』システムの目的と特徴

２．１　『維摩』システムの目的

『維摩』システムの目的は、従来のプログラミング言語とマルチウィンドウ環境との間にインターフェースを提供することにある。

『維摩』ではマルチウィンドウの各ウィンドウと従来のタスクを結び付けることで、Window Packageという概念を提供しており、１つのアプリケーションがＰＣで動作するように、このWindow Packageとして動作できる。このWindow Packageは仮想ＰＣ（Virtual personal computer)の集合として定義されている。各仮想ＰＣどうしは仮想キーボード、仮想マウスを使うことによって相互に通信できる。これを用いてアプリケーションはいくつかの仕事の単位を仮想ＰＣとして動作させることで、自分はWindow Packageとして動作する。

Window Package間にはメッセージ形式の通信手段がある。これによって複数のアプリケーションが『維摩』で複合的なシステムを構築して、動作することも可能である。更に、『維摩』では自然言語処理アプリケーションのための基礎的機能（例えば形態素解析）をサポートしている。これは自然言語処理を行う標準的な仮想ＰＣを『維摩』がもっているためで、これら仮想ＰＣを自分のアプリケーションに取り込む形で『維摩』下の各アプリケーションは日本語処理の標準機能のサポートが受けられる。（カナ漢字変換についてはWindow Packageとしてサポートされている。）

要約すると『維摩』システムの目的は以下のようになる。

ａ．マルチウィンドウシステムのための標準モデルとしてWindow Packageと仮想ＰＣを提供する。

ｂ．アプリケーションの構造をレイヤー化することでmoduleの共通化を行う。これは仮想ＰＣというモデルを使用していることから可能となる。

ｃ．自然言語処理アプリケーションのための基礎的環境を標準的にサポートする。つまりＰＣのＯＳ中に組み込み機能として自然言語処理の基礎的部分を持たせることで、各アプリケーションの負担を軽減すると同時に、複数の自然言語（例えば日本語と英語）を統一的手法で扱うことを可能にする。

２．２　『維摩』システムの特徴

２．２．１　仮想ＰＣの構造

通常のＰＣではアプリケーションはキーボード、ディスプレイ画面、マウス、およびプリンターやディスクにＩ／Ｏインターフェースを介してアクセスできる。『維摩』システムでは実際のＰＣの替わりに前述した仮想ＰＣの集合としてWindow Packageというモデルを提供している。Window Packageは後述するので、ここでは仕事の基本的構成単位となる仮想ＰＣをもう少し詳しく眺める。

仮想ＰＣとは図２．１のように、仮想ディスプレイ、仮想マウス及び仮想キーボードとその仮想ＰＣで動作するプログラム、その他のデバイスから成り立っている。

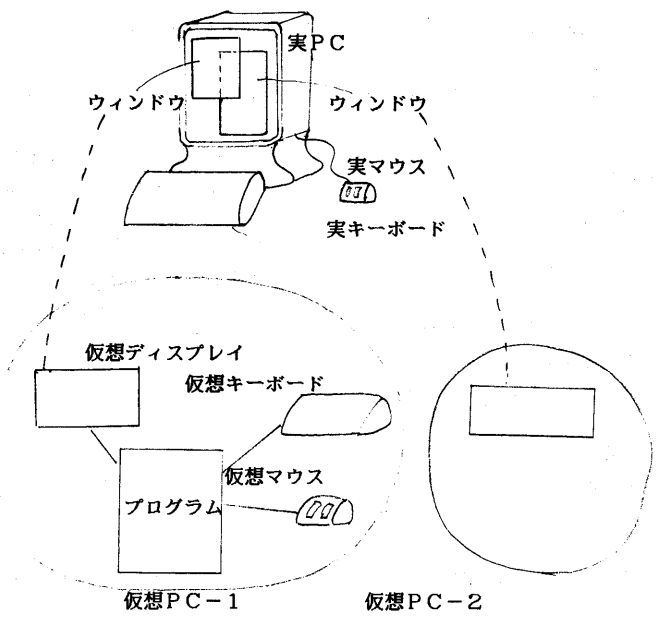

図２．１

・仮想ディスプレイ

いわば仮想ＰＣにおけるディスプレイで、簡単にいうとマルチウィンドウの１つのウィンドウが仮想ディスプレイと考えてよい。しかし仮想ディスプレイをもたない仮想ＰＣも『維摩』では許しているので、ウィンドウの数によってactiveな仮想ＰＣの個数をみることはできない。仮想ディスプレイは単なるウィンドウではなく『維摩』ではもっと積極的な意味を持っている。それは各仮想ディスプレイで扱うオブジェクトは単一のデータ型を持っているということである。つまりＴＥＸＴ，ＧＲＡＰＨＩＣ，ＩＭＡＧＥといった単一の型のみが１つの仮想ディスプレイで扱われる。これは１個の仮想ＰＣのみに着目すると、それがＴＥＸＴ処理ＰＣ、ＧＲＡＰＨＩＣ処理ＰＣといったものに見えることを意味している。

・仮想キーボード

仮想キーボードは『維摩』では本質的な役割をはたしている。それは仮想ＰＣと仮想ＰＣの間の通信手段が仮想キーボードであるからである。各仮想ＰＣは仮想キーボードよりメッセージを受取り、又他の仮想ＰＣと通信する場合には相手の仮想キーボードへメッセージを渡すことでこれを行う。仮想キーボードに乗ってやりとりされるデータはメッセージである。又実のキーボードは特別の仮想キーボードとみなすことができ、これは入力専用になる。

・仮想マウス

仮想マウスには２種類あって仮想ウィンドウマウスと、仮想入力マウスである。仮想ウィンドウマウスは仮想ディスプレイ上の指示デバイスとして機能する。ただし上述のように、１つの仮想ディスプレイは１つのデータオブジェクトを扱うので、この特定のオブジェクトへのポインターという形に表現される。仮想入力マウスの方は仮想ＰＣ間の通信メッセージへのポインターである。つまり仮想キーボード上のメッセージへのポインターとして表現される。これによって入力メッセージのどこまでが処理されたかといった情報も仮想ＰＣ間で受渡しできることになる。

２．２．２　仮想ＰＣの使い方と利点

次に上述の仮想ＰＣがどのように使われるか、及び何故仮想ＰＣを用いることが有利かについて述べる。ここでは『維摩』が標準的にサポートしているカナ漢字変換に例をとって、仮想ＰＣの使われ方をみてみる。

図２．２に於て人間が実キーボードで文字＂ｅ＂を打入したとすると、これが仮想ＰＣ＿Ａ内のprogramAによって認識されて、これをひらがなの＂え＂に変換して仮想ＰＣ＿ＢのprogramBに送る。programBは＂え＂を＂絵＂に変換した後に仮想ＰＣ＿CのprogramCに送る。programCは自分の仮想ディスプレイに＂絵＂を表示する。この結果、人間が見ると＂ｅ＂を入力してウィンドウに＂絵＂が表示されたと見える。

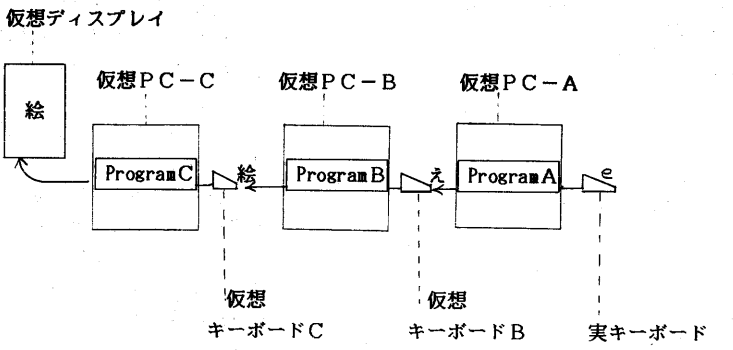

図２．２

このとき仮想キーボードＢはひらがな文字列ばかりのキーボードとにることができ、仮想キーボードＣは日本語の文節がキーとなっているキーボードとみることができる。これによって仮想ＰＣ＿Ｂはひらがな文字列の入力に対して文節によるカナ漢字変換を行っているとみてよく、仮想ＰＣ＿Ｃは単語に区切られた日本語文字列入力に対して、これの表示を行うものとみてよい。

こうすることで各仮想ＰＣはデータの構造のレイヤーに対応してそのレイヤーのみを扱うprogramを作ることができる利点がある。例えば前述の例では単なる文字のレイヤー、単語のレイヤー、文節のレイヤーといった構造に対応してprogramを作ることができる。しかも仮想ＰＣはその特徴からいって、それぞれの構造のみを扱う文字処理ＰＣ、単語処理ＰＣ、文節処理ＰＣといった具合に解釈できる。

２．２．３　Window Packageについて

『維摩』の特徴は仮想ＰＣだけではなく、この仮想ＰＣの集合を有機的単位に於て扱いうる点にもある。これは例えば、個々の仮想ＰＣ内のprogramはせいぜい100~1000ステップ程度のものであるが、この有機的な単位とはエディターとか辞書引きプログラムとかいった一般のアプリケーションにあたるレベルのものである。このことを『維摩』ではWindow Packageと呼んで、data abstractionの立場のpackageとマルチウィンドウシステムのウィンドウを結び付けた概念として提供している。

これは２．１に述べた仮想ディスプレイという概念が単なるウィンドウではなく、これに付随した環境（ここでは単一のデータ型のデータ集合）を意味しているため、仮想ＰＣ間の通信が仮想キーボードによるメッセージの他に、仮想ディスプレイを渡すことで環境の引継ができることを意味している。そこでこの環境の引継ができる範囲（つまり一種のclosure）のことを『維摩』ではWindow Packageと呼んでいる。

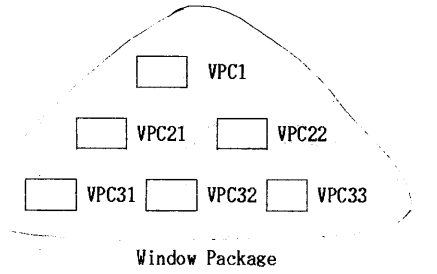

図２．３

Window Package間のコミュニケーションは厳密にメッセージのみに限られており、この意味で環境の引継はできない。つまりエディターと辞書引きプログラムの間ではメッセージをやりとりするだけで、それ以上には互いに干渉できないことになっている。これが『維摩』がマルチウィンドウシステムを作る上で、いくつかのアプリケーションを柔軟に取り込める鍵になっている。

図２．４に『維摩』プログラムの一例を示す。

３．エディターの特徴

上述した『維摩』の下に試作した複合オブジェクトエディターの特徴を次の各点に従って述べる。

ａ．操作性の特徴
ｂ．ホストシステムとの通信の特徴
ｃ．拡張性、変更性の特徴
ｄ．データストリームの特徴

エディターには各種の側面が存在する。これはエディターが各種のシステムに於けるマン・マシンインタフェースとなり得るためである。もちろん狭義には文書エディターは文書データの編集機能のためのマン・マシンインタフェースとしての意味であるが、すこし拡張解釈をすれば文書データベースとのインター

```
?WindowPackage Edit;
   {declarations of conditions}
   var Text: WordP;
        :
   begin

     :
   ?Call InputTxt::Text(Ctl);
   ... := Text; {ref. input text}
     :
   end;
```

```
?Task InputTxt(Ctl:char);

  ?Entry Condition
     WindowProp:   Txt;
     WindowMouse:  none;
     Input:        none;
     InputMouse:   none;
    ?end;

  ?Return Condition
     WindowProp:   Txt;
     WindowMouse:  none;
     Output:       Msg;
     OutputMouse:  WordP;
    ?end;

  var ...;   {local variables}
  begin

      :
    ... := ? WindowProp;

      :
    ? Output := ...;
    ?cease (Ctl);
  end;
```

図2．4

フェースでもあり、文書通信システムとのインタフェースともなり得る。更に複合オブジェクトを扱うことになれば、複合的な対象に対する操作性や統一的な処理の問題も発生する。従ってこれらの側面の特徴を取り上げた訳である。

３．１　操作性の特徴

３．１．１　テキスト操作の特徴

本エディターでは各テキストウィンドウに属性がついており、この属性に従った操作が可能になっている。行間やマージンはテキストの表示に関するものであり、又fontの種類もダイナミックに変更することが可能である。更に操作単位を指定することができ、例えばChar、Word、Pragraphといった文書構造の単位に従って操作ができる。Charを指定すれば、１単語内の文字の削除、挿入、置換等ができ、Wordになれば単語単位で削除、挿入、置換等を行うことができる。特に文書処理では、この単語単位操作が有効であり、カナ漢字変換、辞書引きアプリケーションとのインタフェースがこれによって自然にとれる。つまり辞書引きをエディターの中から行うときに、例えば英和辞書であれば、receivedというテキスト中の単語が指定されたときこの原形receiveを形態素解析によって求め、これで辞書を引くことができるからである。

次に、テキスト中にその整形出力の為に組み込まれているメタシンボル（センタリング記号、下線始め終り記号等）もこのエディターで扱うことができる。通常メタシンボルは表示されないが、ウィンドウの属性を変更することでメタシンボルが表示され、これを操作することができる。

属性の一覧
- 行間、マージン
- Char,Word,Paragraph
- メタシンボル
- fontの種類
- 枠

図３．１

このエディターの操作性の大きな特徴は操作単位による操作ができることと、これがテキストの言語（例えば日本語か英語かといった意味）によらずに統一的に同じ操作を行える点である。従って日本文中に英語が出現したり、又その逆の場合でも、まったく操作性に影響を受けない。これは例えば英文のエディティング中に日本文のカナ漢字変換を使用することができ、同時に英和辞書を引く際には英単語の形態素サービスをユーザは受けられることを意味している。

３．１．２　複合オブジェクト操作の特徴

本エディターはテキストだけでなくＧＲＡＰＨＩＣ，ＩＭＡＧＥといった複合オブジェクトを扱うことができる。これは前述の『維摩』の特徴を生かしたものである。ここではその操作性の特徴をみるが、これは大きくウィンドウの操作性に依存している。何故ならば『維摩』の特徴から１つのウィンドウでは１つのオブジェクトしか扱わないので、同一のウィンドウ内にテキストとＩＭＡＧＥが混在することはないからである。しかし現実の文書には、１ページ中にテキストもＩＭＡＧＥも混在しうるわけで、これに対処するために『維摩』では親子ウィンドウという概念を導入している。これは１つのウィンドウをいくつかの区画に分割して、その１区画は従来通り１つのオブジェクトしか扱わないようにしたものである。

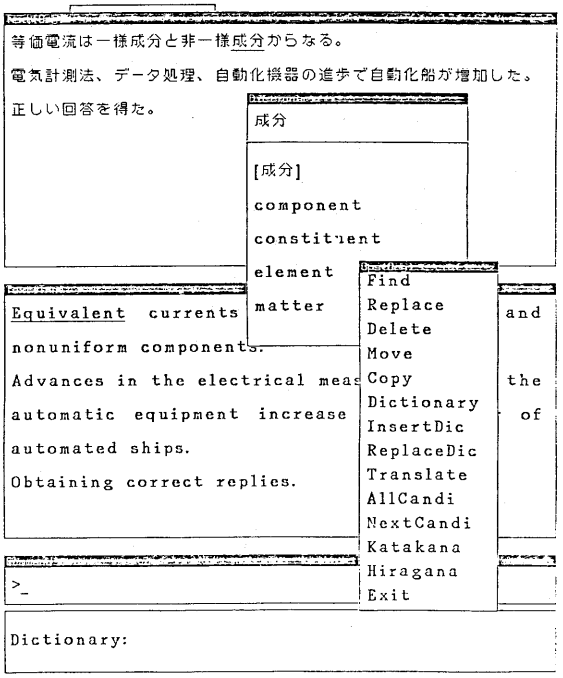

図3．2

こうすることによって図3．3のように、テキストのなかに子ウィンドウを発生させることでＩＭＡＧＥを埋めこめる。本当にテキスト内にＩＭＡＧＥを入れてしまうとテキストエディターがＩＭＡＧＥの面倒をみるか、逆にＩＭＡＧＥエディターがテキストの面倒をみることになってしまう。ところがこの方式だと、テキストエディターとＩＭＡＧＥエディターが別々に動いて見かけ上複合オブジェクトを扱っていることになる。従って従来のテキストエディターとＩＭＡＧＥエディターの操作性は１つも損なわれることがない。

更に別々のテキストとＩＭＡＧＥを２つ入力して、画面上で図３．３のように組み込むことも可能である。つまりテキストを作成するときに同時にＩＭＡＧＥを作成する必要はなく、後からＩＭＡＧＥを入力してそのテキスト内に調整するということが可能になっている。

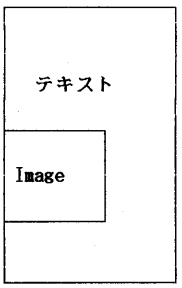

図3．3

３．２　ホストシステムとの通信の特徴

本エディターではホストをbackground machineとして使用できる。つまりホストの存在をユーザが意識しないでエディターの中からホストが使える。これはワークステーション上のエディターが自分では担えない仕事（翻訳、ＤＢ検索等）はホストに自動的に依頼できる仕組になっているためである。

このようなことは、現在のエディターでは必要不可欠の要素ではないが、将来エディターの機能を拡張したり、より大きなシステムの中でエディターを考える場合に有効である。

この機能は『維摩』のマルチタスクの特徴を利用したものであるが、ホストに仕事を依頼している間は、エディターは自由に自分の仕事ができる。ホストから返却された結果を、再びエディターが扱っているテキスト中に取り込むことも容易である。これはホストへの要求キーをエディターが内部的に管理していることから可能になっている。

通信そのものは通信専用のWindow Packageが存在するので、エディターとこのWindow Packageとのメッセージ交換という形でホストとの通信が行われる。

３．３　拡張性、変更性の特徴

ユーザが本エディターの機能を拡張したり、自分向きのマン・マシンインタフェースを導入して、これは変更したりできる。これは『維摩』が仮想ＰＣという概念で構築されているため、比較的容易にこれらのことが実現できる。

つまりユーザは仮想ＰＣに自分向きの外部仕様を定義してやれば、このように実行してくれるエディターを作ることができる。もう少し具体的にいうと次のようになる。例えばあるCommandを１つのキーに割当てたいとする。するとユーザはSET Commandでこの旨をエディターに通知する。するとエディターは以後その指定されたキーを特別のCommandと解釈する。

これにはマンマシインタフェースのみを司どる仮想ＰＣがあってこの仮想ＰＣがその特定のキーに対して、エディター本体向けに、仮想キーボード上に約束されたCommandの文字列をのせてやることで可能になっている。

更にユーザは簡単にProgramでエディター自身を拡張し、自分向きのエディターを構成することができる。これもエディターの核部分の仮想ＰＣ群を自分向きに組み合わせ、その上にいくつか自分が作った仮想ＰＣをつければ簡単に作成することができる。基本的な表示用の仮想ＰＣ、ファイル読みこみ仮想ＰＣ等はそのまま使用できる。又ある特定のアプリケーションを『維摩』下で動かしてエディターと組み合わせることも簡単である。

３．４　データストリームの特徴

本エディターが扱うデータストリーム（つまりファイル上や回線上のデータ形式）は文書データストリームである。この文書データストリームの中にはテキスト、ＧＲＡＰＨＩＣ，ＩＭＡＧＥといった複合オブジェクトを当然含んでいる。３．１．２節にも説明したが、基本的に１ページが１つの親ウィンドウに表示されて、子ウィンドウたちに各オブジェクトが入っている。この２次元の形式を１次元に表現し直したものがデータストリームであるといえる。従って文書交換によって別ワークステーションに移っても、この文書の画面上の再表示には混乱はない。

１ページ全体の情報	各子どもウィンドウの情報	．．	各子どもウィンドウに入る１つのオブジェクトそのもの	次のオブジェクト	．．

図３．４

もう少し具体的にストリームを説明すると、テキストならテキストのオブジェクト自身が持っている情報は基本的には構成単位（文字、単語、文、段落）の情報と３．１．１にも述べたメタシンボルから成り立っている。ウィンドウ情報とは相対的な位置とウィンドウの属性に関するものから成り立っている。

４．エディターの設計

『維摩』システムを用いて、上述のエディターの特徴をもつものの設計を行った。ここではいくつかのアプリケーションの組み合わせには、Window Packageの組み合わせを用い、各Window Package内では機能レイヤー分けを重視し、各レイヤーに『維摩』の仮想ＰＣを割当てていくというやり方で設計した。

複合オブジェクト・エディターの構成を大きく３つの段階にわけて考えた。

ａ．複数のWindow Packageの組み合わせによる複合オブジェクト・エディターの構成。
ｂ．１つのWindow Packageとしてのテキスト・エディターの構成。
ｃ．複数のテキスト・エディター相互の構成。
（ａ．の特殊な場合）

以下これら３つの部分について、それぞれ設計の考え方を述べる。

４．１　複合オブジェクト・エディターの構成。

複合オブジェクト・エディターを構成するWindow Packageには次のものがある。

ａ．『維摩』が提供する標準Window Package
ｂ．テキスト・エディター
ｃ．ＩＭＡＧＥ・エディター
ｄ．ＧＲＡＰＨＩＣ・エディター
ｅ．通信Window Package
ｆ．辞書Window Package

このうちａ～ｄは必須のものであるが、ｅ，ｆは付加的存在で、ｅは３．２に述べたものであり、ｆは例えば日英、英日辞書引きプログラムである。ａは２種類あって、ウィンドウの操作にかんするもの（move reshapeといったマルチウィンドウの操作）と、カナ漢字変換に関するものがある。ｂ自身は４．２に記述される。

２，３章の説明で明らかと思われるが、Window Package相互は基本的に独立に動作し、必要なときだけメッセージを用いて通信できる。複合オブジェクト・エディターとして最終出力が複数のオブジェクト（テキスト，ＩＭＡＧＥ，ＧＲＡＰＨＩＣ）から構成され整形出力されればよいので、個々のオブジェクト編集中には１つのウィンドウに閉じた操作であると判定している。ウィンドウ相互にまたがる操作としては次のようなものがある。

（１）　ウィンドウの操作に関するもの
（２）　辞書またはカナ漢字変換とテキスト・エディター間のもの
（３）　テキスト・エディター相互のもの
（４）　ホストとの通信に伴うもの
（５）　ＩＭＡＧＥ，ＧＲＡＰＨＩＣとテキスト・エディター間のもの

１は例えばテキスト・ウィンドウのreshapeを行ったとき、ウィンドウ操作Window Packageはウィンドウ枠のreshapeはできても、その中にテキスト表示することはテキスト・エディターに依存しなければならない。

２は例えばカナ漢字変換の全候補リストがウィンドウに表示されていて、このうち１つをテキスト・ウィンドウに取り込みたいとき、カナ漢字変換Window Packageはテキスト・エディターWindow Packageに結果を返却しなければならない。

３はテキスト・ウィンドウの分割が発生して、いくつかの子ウィンドウができそれに伴って複数のテキスト・エディターが動きだしたとき、親ウィンドウ全体として１つのテキスト画面のようにふるまうには、テキスト・エディター相互に通信する必要がある。このことは特に４．３節に述べる。

４については３．２節に述べた。

５については例えば、ＩＭＡＧＥやＧＲＡＰＨＩＣの一部にテキスト・エディターからテキストをもらって、これを図や絵の一部に埋めこむ操作等がこれである。この場合もちろん埋めこまれたテキストはＩＭＡＧＥやＧＲＡＰＨＩＣのデータ型になる。これは『維摩』の特徴からあきらかである。

これらのWindow Package間の通信には、この通信を司どる特別の仮想ＰＣをそれぞれのWindow Packageに置いて、メッセージ・プロトコルの解釈、生成を行えばよい。従ってもともとのWindow PackageがWindow Package間の通信で影響される範囲はきわめて限定されている。

４．２　テキスト・エディターの構成

テキスト・エディターWindow Packageは論理的な構想のレイヤーに対応するようにプログラムを分割す、それぞれを仮想ＰＣに割当てる方法が取られる。

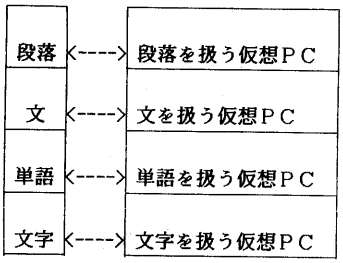

図４．１

図４．１のように段落、文、単語、文字といった文書構造に対応する形で仮想ＰＣに割当てる。これは複雑な操作や複雑な構造を、基本的な操作と基本的な構造の組合せで、いかに表現するかということになる。このための手段として２．２節にも述べたが、仮想ＰＣは大変有効である。

４．３　テキスト・エディター相互の構成

２つ以上のテキスト・エディターが協調して動かなければならない場合は、例えば図４．２のようなウィンドウの分割が発生して、２つの子供ウィンドウが同一の文章の一部とその続きを表示している場合等である。

ｴﾃﾞｨﾀｰの	ｲﾝﾀｰﾌｪｰｽ
核の設計	どこまで
に関して	考慮する
ﾏﾝ･ﾏｼﾝ･	か?

図４．２

このとき左側の子ウィンドウでスクロールが発生したとき、自動的に右側の子ウィンドウにもスクロールが発生しなければ不自然である。このような場合左側のテキスト・エディターWindow Packageと右側のテキスト・エディターWindow Package間で通信することが必要になる。

これには２つのテキスト・エディターが今オブジェクトとして同一のテキストを持っているわけであるから、Window Package間のプロトコルとして

ａ．この位置から表示する。

ｂ．マウスの指した位置を通知する。

といった約束事を決めておく。これにはWindow Package間のメッセージに要求と位置の情報があればよいことになる。テキスト上の位置をメッセージ形式にするには仮想ウィンドウ・マウスを通知する方法で解決される。

５．インプリメントの手法

エディターのインプリメント手法全体をここに解説することは不可能であるから、いくつかのトピックに限って説明する。

５．１　ウィンドウの組み込み機能のインプリメント

３．１．２に解説した複合オブジェクトを１つの親ウィンドウ内で扱うためのウィンドウの組み込み機能のインプリメントについて述べる。ウィンドウは内部的には図５．１のように、Window-idとそのウィンドウがもつオブジェクト及びウィンドウの属性（３．１．１参照）とから成り立っていると考えてよい。これらウィンドウは『維摩』によって管理され、双方向チェインで繋がれている。

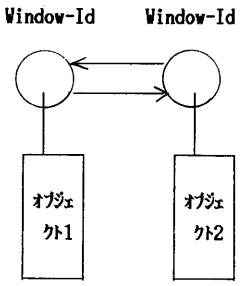

図５．１

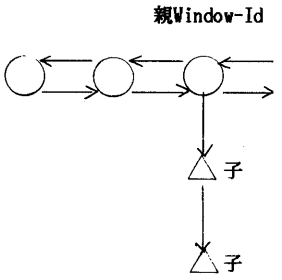

図５．２

更に、親子ウィンドウとなると図５．２のようなチェインとなる。これによって例えばマルチウィンドウ・システム内でウィンドウの移動が発生すると、親ウィンドウのみが移動すれば自動的にそれに伴って子ウィンドウも移動することになる。

さて、この道具立てでウィンドウの組み込みをインプリメントするわけであるが、これは１つのウィンドウを他のウィンドウの子供とみなすような構造にすることである。つまり組み込まれたウィンドウが組み込み側の親ウィンドウの子供と同一視される構造にすることである。このとき組み込み側はもともとあったオブジェクトをウィンドウへ表示するときに、図５．３のような自動的な分割が発生する。

子供１，３，４は子供２が割り込んできたために発生したものである。こうすることで、一般のウィンドウへテキストを表示する仮想ＰＣをそのまま用いて、子供１，３，４へテキストを表示することができる。（分割されたウィンドウ相互の通信については４．３を参照）このように基本的な仮想ＰＣの組み合わせで、より複雑な機能が実現できるところに『雑摩』の大きな特徴がある。

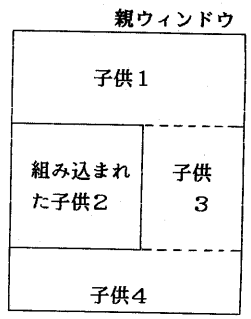

図５．３

５．２　表示プログラムのインプリメント

１つのウィンドウにテキストを表示する仮想ＰＣのインプリメントについて述べる。この仮想ＰＣは内部的論理構造からウィンドウにこれを表示するための構造化バッファ（以下物理構造という）を作成し、これに基づいてウィンドウ上にテキストを表示するものである。論理構造とは図５．４のような文、単語、文字といった文書の論理的構造をリスト構造に表現したものであり、他方物理構造とは単語と行のマトリックス表現といえる。

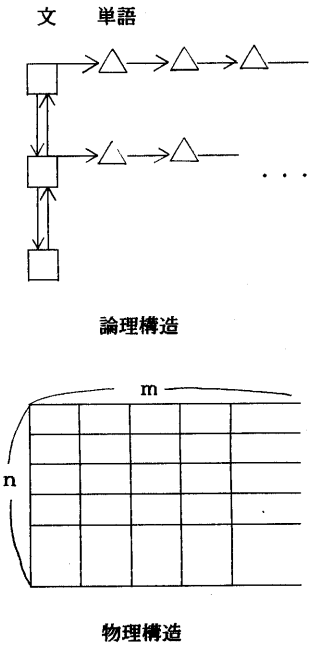

図５．４

この物理構造中に、ｘ，ｙ座標、font長等の表示に必要な情報が格納されている。ｍ，ｎはそれぞれ１つのウィンドウに入る最大の１行中の単語数、及び行数である。こうすることによってテキストに変更があった場合に論理構造、物理構造とも影響を小さく抑えるようになっている。又、表示速度、検索速度の点でも、fontの切り換えに関するハード的なことを除いてある程度満足のいく構造になっているといえる。ただし新しい行の追加に対して、この構造（物理構造のこと）は弱いが、それを補強すると全体の効率が下がるという二者択一であって、今後実際的な性能測定に結果を待つしかない。この２つの構造を用いることでApa Displayに於てダイナミックなフォーマッチングを可能としている。

６．おわりに

以上がワークステーション制御システム『維摩』の概要と『維摩』を使用した複合オブジェクトエディターの特徴である。『維摩』に於ける最大の特徴はなんといっても仮想ＰＣの概念であるが、これはメッセージと環境の両方をひきずることができる点でユニークである。

オブジェクト・オリエンテッドの考え方に立てばメッセージだけが必要で環境はオブジェクトの中に記述されることになる。これだと従来のProgramming styleと大きく離れる点、及び環境に対する記述が必ずしも十分でないため、Programもそれほど容易でないこともある。（例えば、文章を段落、文、単語、文字という構成要素からなるものとして表現しようとするとき、これらは直接にはオブジェクト・オリエンテッドのクラス・ハイアラキーでは表現できない。何故なら潜水艦は船のサブクラスであるが単語は文のサブクラスでないことを考えればよい。）

又ファンクショナル・プログラムの立場に立てば、メッセージは不要となるが、side effectの表現が容易でなくなる。しかし、現実には本質的にside effectを伴う処理がある。つまり以前の処理の文脈で現在の処理の意味が変化するようなもの（エディターにはこのような操作はいくらでもある）をファンクショナルに記述することは容易でない。

これらの理論的欠点を補い、現実的に柔軟かつ明快なモデルとして仮想ＰＣを導入したものが『維摩』である。従って『維摩』を用いたエディターは従来のProgram言語を用いて自然に機能のレイヤーによって仮想ＰＣを積み重ねる形で実現できたといえる。これは『維摩』の仮想ＰＣの概念が，各データ構造の各レイヤーを扱うことに自然に対応づけられるからに他ならない。

７．参考文献

[1] 諸橋他「ワークステーション制御システム『維摩』とテキスト・エディタ」日本ソフトウェア科学会第１回大会3C-3,1984

[2] 諸橋他「オフィスシステムのためのワークステーション制御システム『維摩』」情処全国大会59秋3E-4,1984