解 説

1. ユーザインタフェース管理システムの基礎

1.3 ユーザインタフェース管理システムと応用プログラムとの通信†

今 宮 淳 美††

1. はじめに

現在まで，UIMS の大部分の機能に対する基本的仮定は，ユーザインタフェース（対話部，以下では UI と略記する）とアプリケーション（計算部，または意味部，以下では AP と略記する）とが分離可能ということである[1),3)]．しかし，新しい対話技法や直接操作対話形式の出現によって，どのようにして UI と AP 間に境界線を引くのか，およびどのような機構で両者間の通信を行うのかという問題が生じた．さらに，実行と通信を連動させるにはどのような制御が必要であるのか．“分離”の意味が明確ではない．論理的分離といえるのは UI と AP ルーチンがデータを共有しないで，限定した相互間の通信機能をもつ場合である．物理的分離といえるのは UI の記述が AP ルーチンと異なるファイルにある場合である．実行時構造としての最良の組み合わせは，論理的分離がなく物理的分離のあるシステムである．なぜなら，UI，AP ともにおのおの最適の言語で記述でき，かつ対話システムの各要素間で必要に応じて柔軟に通信ができるからである．よく知られている UIMS アーキテクチャモデルである Seeheim モデルが，これら論理的および物理的分離の区別を扱っているのかどうかは明らかでない．

Seeheim モデル[4)]における UIMS の第三要素 Application Interface model (APIM) は，UI からみた AP 表現であり，UI に関係するデータ（オブジェクト）と UI が起動できるプロセスで AP の意味を定義する役割，および AP が要求する形で UI と AP 間の通信を処理する役割をもつ．さらに，第一要素であるプレゼンテーション要素と APIM との通信も明らかではない．対話型の図形処理システム，特に直接操作対話形式を用いるシステムでは，プレゼンテーション要素が AP データを通して動作することが AP にとって必要である．本稿では，これらの観点から現在までの UI と AP との通信に関する研究をまとめるとともに今後の課題について述べる．

2. UI と AP の分離

対象とするオブジェクトとの“直接通信感覚”[5)]をユーザに提供する方法のひとつは，その AP オブジェクトの振る舞いを反映する画面上のオブジェクト表現をフィードバックを通してユーザに見せることである．従来，フィードバックは三つのレベルに分けられている．すなわち字句，構文，意味[6)]である．しかし，直接通信感覚を強化するために高いレベルのフィードバックを提供しようとするとそれらのレベルの統合が必要になる．特に，普通，字句的レベルと考えられるところで意味的フィードバックを必要とする．ドラッギングはこの良い例である．普通，画面上のオブジェクトのドラッグは字句的オペレーションである．しかし，オブジェクトを動かすことが意味的結果をもつので，ドラッギングの結果についてのフィードバックをユーザに与えることによって，システムはユーザの直接通信感覚をおおいに増すことができる．たとえば，デスクトップインタフェースにおいて，ファイル削除のためにファイルアイコンをトラッシュ（ごみ箱）アイコン上にドラッグで移動させる．削除の意味的フィードバックのためにごみ箱のハイライト表示がなされる．もしこの作業中，ファイルアイコンが他のファイルアイコン上を通ってもそのアイコンをハイライト表示はしない．しかし，ファイルアイコンをフォルダアイコン上にもってくるとフォルダアイコンをハ

† The Communication between UIMS and Application Programs by Atsumi IMAMIYA (Department of Electrical Engineering and Computer Science, Yamanashi University).

†† 山梨大学電子情報工学科
e-mail: imamiya @esi.yamanashi.ac.jp

イライト表示してファイルとフォルダ間に意味的関係があることをユーザに思い出させる．もしその場所でマウスボタンを放すとファイルがフォルダに投入される．したがってUIで入力動作に対するエコーが必要ならば，UIがオブジェクト間の意味的関係の情報をもつ必要がある．

2.1 実行時アーキテクチャ

意味的フィードバックを高速にさせようとすると，UIの対話部とAPの意味（計算）部をなるべく“密接な結合”にしておく必要がある．しかし，意味処理を対話部に移すとシステム全体が複雑で分離の“度合い”が小さくなる．対話部の機能が小さければ分離は明確になるが，UIとAP間の通信のオーバヘッドを増大させる．

構成要素間の制御と通信の観点からシステムの分離を考えるために図-1に示すシステムアーキテクチャについて説明する[7]．対話部は自身のプロンプト，エラーメッセージ，および構文のエラーチェックと処理を扱う必要がある．したがって，入力要素はある程度の計算CIと表示DIの機能をもたねばならない．同様の理由から，表示要素はある程度の計算機能 C_D をもつ．

このアーキテクチャでは新しい問題が生じる．どれだけの計算と表示能力が入力要素に必要なのか．要素間の分離線はどうなるのか．これらの問題に対しては，個々のシステム設計それぞれについてしか答えられない．IにDとCの機能をたくさんもたせるとIの複雑さは増大するが，Iとその他との間での通信要件（帯域幅と通信制御）は減少する．幾つかのオペレーションでは，IからC，Dにデータを移動する時間(応答時間)を短くすることができる．分離線をもっと遠くに動かしてたくさんのCとDの機能をIにもたせることもできる．もちろん，そうするともともとの問題—UIとAPの独立性の問題—がでてくる．最良の分離は“対話部に適量の機能”をもたせることができるが，しかし，これは容易でなく対象となる個々のAPについてそれぞれ適切なシステム設計を考える必要がある．

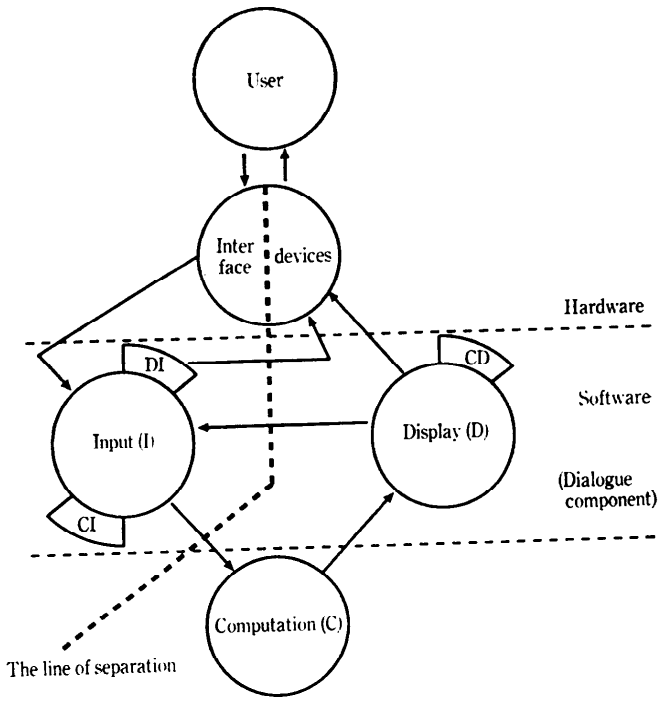


図-1 UIMSの実行時アーキテクチャ

2.2 要素分離の問題点

要素分離を決めるときに三つの重要な問題点がある．第一は，通信のオーバヘッドである．入力要素Iが処理能力を十分にもてば，ユーザからの生入力を解釈してそれらコマンドとパラメータを他の要素に送ることができる．このレベルの通信を“マクロ通信”と呼ぶことにする．もし入力要素Iがほとんど処理能力をもたなければ，Iはすべての入力イベントをCまたはDに送らなければならない．このレベルの通信を“ミクロ通信”と呼ぶことにする．ミクロ通信は明確な分離をもたらすが，通信のオーバヘッドは大きい．

対話の独立性が第二の問題である．もし生の入力イベントがCに送られると，本来Iにあるべき処理（たとえば，字句および構文的入力イベントを解釈してコマンドとパラメータに変換する機能）をCがしなければならない．もし対話部に対して字句および構文的変更を加えると計算部Cも変更しなければならないかもしれない．

第三の問題点は，特に直接操作対話形式[8),9)]で重要となる意味的フィードバックである．前述のように，この対話形式では意味をUIにより近づけるので明確な分離に反することになる．意味を

入力要素に近づける方法はマクロ通信とミクロ通信に対応して二つある．UIにおける意味的フィードバックを特定の比較的簡単なオブジェクトの振る舞いに限定すれば，マクロ通信で実行できる．選択図形オブジェクトのカットやコピーのような簡単なアクションを実行するには，Cはコマンドとそれが対象とするオブジェクトだけを知っていればよい．すなわち，Cは入力イベントに関する字句および構文上の特別の情報を知っている必要がない．しかし，選択オブジェクトをハイライトさせるときはどうであろうか．マクロ通信では，Iだけが未解釈の選択イベントを感知するので，Iが選択オブジェクトをハイライトさせなければならない．このことは，APオブジェクトのある意味（この例ではハイライトが意味すること）をIが知らねばならない．アンドゥコマンドは対話型システムが必ず提供すべきコマンドと言われている．直接操作対話形式においてこのコマンドは重要であるが，アンドゥコマンド自身は対話的でも直接的でもない．マクロ通信では，このコマンドに対してUIは意味的フィードバックを要求しないし，APも入力イベント情報を必要としない．すなわち，アンドゥコマンドがIからCに送られ，そこではオブジェクトが（バッファ中にある）前の状態に戻されてDがそのオブジェクトを表示する．

直接操作インタフェースでよく使われるその他のオペレーションには，オブジェクトのドラッギング，ドラッギングやラバーバンドによる大きさや形，位置の変更がある．もしこのオペレーションをミクロ通信で行うとすると，通信帯域幅を広くしなければならない．なぜならば，マウスが新しい座標値を検出するたびに再計算と再表示を引き起こすからである．しかし，マクロ通信でも問題がある．ドラッギングの場合に，オブジェクトのサイズと色をオブジェクトの座標の関数として計算する例では，要素Iが複雑な処理機能を要求される．各要素の機能分散からみたこのシステム分離の問題は，UIMS設計方法論の最も重要な研究課題のひとつである．

3. 対話部と計算部の通信機構

3.1 完全分離

既存の大部分のUIMSでは，UIはAPのルーチンを呼び出し，APルーチンはUIを呼び出すことができるという実行時構造をもっている．問題は，UIとAP間の通信が手続き呼び出しだけとすると，この通信は呼ばれる側が呼ぶ側のルーチンに情報を返すだけの一方通行になりがちということである．たとえば，グラフィカルユーザインタフェース(GUI)では入力として表示変数をもつAPルーチンを呼び出す機構（APへの脱出口）をUIMSがもっている．しかし，UIとAPを完全に分離するとUIからAPのデータ構造を覗いたり，編集することはできない．この場合，完全な分離では図形表示更新の負荷をAPに負わせることになり，UIとAPの仕事のバランスからみてよいアプローチではない．ディスプレイマネージャ（表示要素）はAPのデータ構造について何も知らないので，表示を更新するためにAPはそのデータ構造に対して行った変更をすべてディスプレイマネージャに詳しく伝える必要がある．このことは，APのプログラマにこの図形問題を意識させることになるのでUIとAPとの完全な分離を避けるべき理由のひとつである．すなわち，UIはAPルーチンにコマンドと引数を送り，APルーチン状態が変化するにつれて変化する領域の範囲やデフォルトなどの情報を，APルーチンがUIに送るようなバランスあるアプローチが必要である．

3.2 active value

別のアプローチとして，表示用変数とAPのデータ構造の重要な変数とを連結するactive valuesの通信方法がある[10]．GUIにおいて，制約を使って図形オブジェクトの属性値で他の図形オブジェクト属性値を定義することができる．たとえば，グラフ構造を表示する場合，ノード間の結合を制約条件で指定しておけば，画面上のあるノードの位置を移動させてもノード間の結合制約をみたすように関係するノードの位置をソルバが再計算してグラフ全体の表示更新をする．

active valueは，データ値とその値に依存するオブジェクトリストと手続きからなる．データ値が設定されると，オブジェクトを再表示できるこ

とをそのオブジェクトに知らせ，関連する手続きを呼び出して AP に知らせる．active values は図形制約に似ているが，違いは，図形オブジェクト間でなくデータ値から図形オブジェクトに対する制約である点にある．すなわち，“データ制約”と呼ぶことができる．active value が設定されるとき呼ばれるその active value を引数にもつ手続きを AP は登録しておいて，その手続きが active value の変更を AP に知らせる．図形オブジェクトと active values でのデータ制約が自動的に表示更新を処理するので，AP は更新処理に関わる必要がない．ユーザインタフェースツールキット Coral では，各図形オブジェクトがマウスやキーボード入力を扱うのでなく，特定の入力処理用オブジェクト interactors がそれら入力を処理する（**図-2** 参照）．interactors は，active values を介して図形オブジェクトと通信する．データ制約を使って図形オブジェクトと active values を結合するので，interactors は表示を更新するために active values を設定するだけでよい．active values は UI の入力と出力を分離しているので，異なる interactors や AP 自身が出力機能を動かせたり，繰り返しの開発を容易にさせている．

この方法では，active values を AP 手続きと結合することができるので，ユーザが active values を変更するとそれにともなって適当なルーチンを呼ぶことで AP のデータ構造を更新することができる．また，UI が AP のデータ構造をモニタ可能で，データ構造の変更に従って表示を変更することができる．この方法の利点は，AP が UI について，たとえば，active values が図形としてどう表現されているかなどを知らなくてよい．また，UI が AP について，たとえば，木，リスト，集合などのデータ構造をどう実現しているかなどを知らなくてよい．この active values の方法では，UI と AP を物理的に分離している．この方法の欠点としては，図形表示更新の負荷を AP に移していることである．active values を使って，UI は，適当な表示変数を変更することによって AP のデータ構造の比較的簡単な編集が可能であるが，複雑な構造変更はできない．

3.3 共有データ構造

第三のアプローチは，UI と AP を統合してそれらに共通のデータ構造を共有させる方法である．この方法では，UI が AP のデータ構造を覗くことができる．したがって，ユーザは，入力コマンドを使って UI からそのデータ構造を編集することができる．さらに AP プログラマは，表示更新のことを考える必要がない．しかし，この共有データ構造に対して，UI と AP がそれぞれどれくらい制御するかを注意深く検討して決める必要がある．この方法の実現の仕方には，手続き型システム，および宣言型システムの二つがある．どちらのシステムでも，AP のオブジェクト間でどのような型の関係を保持すべきかを指定する．手続き型システムでは，オブジェクト間の関係をいかに満たすかを記述する命令集合をシステム設計者がユーザに明示的に提供する．宣言型システムでは，計算機構（たとえば，制約ソルバ）を用意してこの機構が自動的にオブジェクト間の関係を満たしてくれる．

3.3.1 手続き型システム

STUF システム[11]は，ブラウジング，編集，および AP オブジェクトの図形表示を制御するのに手続き型アプローチを採っている（**図-3** 参照）．STUF では，リンクリストや木のような一般的データ構造に関するオペレーションを実現するのに編集テンプレート（editing templates）を使う．ひとつのデータ構造の特定のインスタンスの生成に使うパラメータ集合，および UI がそのデータ構造を覗いたり，ユーザが発生させるコマンドに応答してそのデータ構造を編集したり，そのデータ構造内容に基づく表示の変更をするのに使う手続きの集合を編集テンプレートが用意している．

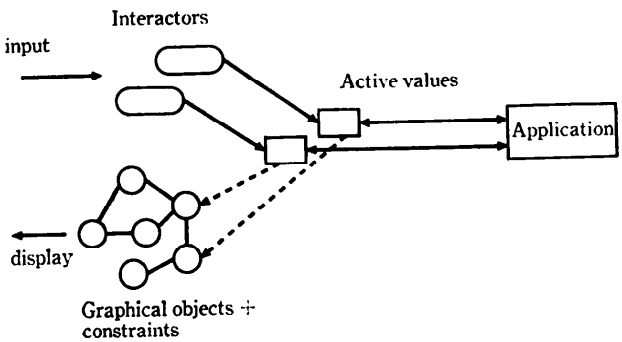


図-2 Coral システムのアーキテクチャ

リスト要素の生成とかリスト要素の表示方法の指定のようなAP自身にもたせたいオペレーションは，テンプレートから除かれている．STUFは，初めにユーザが発生するコマンドとテンプレート中のオペレーションとのマッチングを調べて，マッチしなければ，それらコマンドをAPに渡す．APプログラマがテンプレートのインスタンスを作成するには，インスタンスクラスエディタからテンプレートのインスタンスを要求して，そのパラメータ欄に適当なデータを記入する．このデータ構造についての要素の追加，削除，およびデータ構造の変更には，もしあればそのテンプレートが供給する手続きをAPプログラムは呼ぶことができる．テンプレートがその編集をAPプログラマの判断に任せるならば，プログラムは特定の編集ルーチンを書くことができる．その後で，テンプレートが用意する表示とブラウジングコマンドを使って画面表示を更新する．

3.3.2 宣言型システム

対話型グラフィクスシステムは，対象とするAP分野の特性に強く依存する構造と意味をもつ複雑な実体を扱う．このシステムを実現するのに最も重要な仕事のひとつは，APオブジェクトの構造をユーザがよく理解できてそれらを直接操作できるようにUIを作成することである．

［A］ Higgens：Higgensシステムは，高レベル仕様からGUIを生成するツールをもつUIMSである[12]．このシステムは，ユーザ入力に応答するためにAPの根底にある意味を利用できるUIを生成する．さらにUIがアクセスできるAPオブジェクトの構造と意味を記述するための強力なデータモデルをこのシステムはユーザに提供する．図-4はHiggens UIMSとSeeheimモデルの論理要素との対応を示す．Higgens UIMSは，三つの要素からなる．すなわちAPデータモデル，ビュー要素，およびプレゼンテーション要素である．この構造の特徴は，従来の研究で強調されている“対話管理”でなく，ユーザが扱える“オブジェクトのモデル化”に焦点をあてていることにある．APデータモデルは，APIMとしての役割をもつ．ここでは，UIがアクセスできるAPオブジェクト（データ）をモデル化してそれらを保持する．APもこれらオブジェクトをアクセスすることができる．ユーザが選択してオブジェクトの属性値を変更すると，UIはその変更をAPに知らせる．ビュー要素は，オブジェクトのビュー(オブジェクト表示の表現)を生成する．ビューには，オブジェクトそれ自身にはないがユーザに表示するのに必要な情報が存在する．たとえば，画

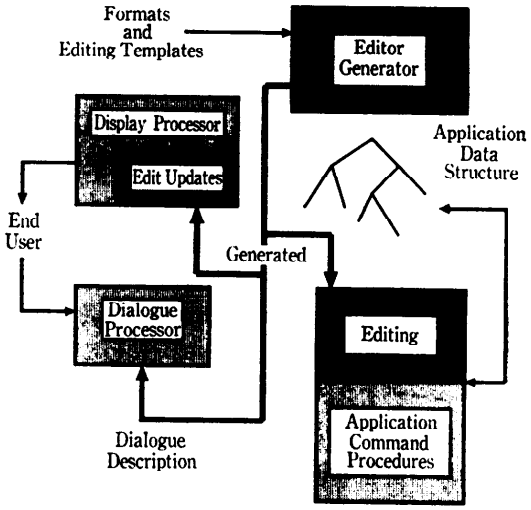


図-3 STUFシステムのテンプレート編集アーキテクチャ

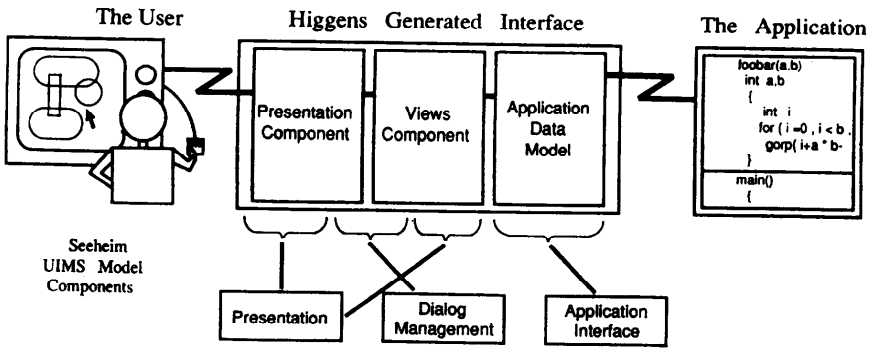


図-4 Higgens UIMSとSeeheimモデルとの比較

面のスペースや配置に関する情報などである．このビュー要素は，Seeheim UIMS モデルにおけるプレゼンテーション要素と対話管理要素両方の(すべてではない)役割をもつ．プレゼンテーション要素はユーザとの対話に使う図形処理用入出力デバイスを管理する．入力デバイスを管理するときは，この要素は表示の仕事と対話マネージャの役割を果たす．

AP オブジェクト処理：次に，UI が AP オブジェクトをどう扱うかについて述べる．意味的フィードバックをとおして UI が AP の状態を反映させるために，UI は AP オブジェクトをアクセスできる．UI が使えるこれら共有オブジェクトは，特別な意味データモデルを使って記憶されている[13]．このデータモデルは，オブジェクトを表現するだけでなく，これらオブジェクトに関係する AP の意味を表現するとともに直接実現する．このモデルは意味的制約と新しいデータを得てオブジェクトの属性値の変更に対して自動的に応答する．したがって，このデータモデルは意味的フィードバックを生成するのに必要な入力と出力を強く結合させる．表現されるオブジェクトは，型と方向つきの関係で結合され全体として有向グラフの構造をもつので，データモデルは部分的にはネットワークデータベース[14]や意味ネットワーク知識表現[15]と似ている．共有オブジェクトに対応する各ノードは属性をもつので UI が使うそれらオブジェクトを属性グラフと考えることができる．一般に，AP の意味では属性値間に関係または命題が成り立っている．したがって，たとえば属性値をユーザ入力によって変更すると意味的一貫性を維持するために制約や命題を調べて関連する属性値を再計算することが必要である．この計算を自動的に処理するために，Higgens システムでは属性評価技法を使う．長い間，属性評価はコンパイラ作成におけるプログラミング言語の静的意味記述に使われている[16]．Higgens システムは属性グラフに対する増分的属性評価アルゴリズムによるソルバをもっていて，評価の結果は意味的フィードバックによってプレゼンテーション要素に送られ，対応するオブジェクト表示が変更される．

Higgens システムの属性評価ソルバの特徴は，その時点の画面表示に必要なオブジェクトの属性値だけを再評価する点である（“怠け者の評価”）．UI と AP の通信はこの共有オブジェクトをとおして行われる．プレゼンテーション要素はユーザ入力をメッセージの形に再構成してオブジェクトに送る．それらのメッセージ送信によって必要なときに UI から AP に制御を渡すことができる．しかし Higgens システムでは AP をオブジェクト指向言語で記述してはいない．標準のプログラミング言語の手続き呼び出しにメッセージ渡しの動作を変換している（メッセージを受けるオブジェクトを手続きのパラメータのひとつとしている）．

［B］ Synthesizer Generator と子孫たち

Synthesizer Generator (SG) はプログラムや定理のようなテキストオブジェクトを扱うプログラミング環境を提供する非図形 UIMS である[17),18)]．属性文法は，これら環境を記述する強力な機構を提供する．属性文法は，型情報のような文脈依存情報を計算するために各生成規則に属性に関する式（意味規則）をもつ自由文脈文法である[19)]．式中の変数を属性と呼び，各式は，ひとつの属性の値を計算する．生成規則中の非終端記号ひとつにひとつの属性がある．属性式と文法の各生成規則と関係する帰納関数の集合として AP が実現されているので，AP と UI は完全に統合されている．SG のおもしろいところは，提供する編集モデルである．編集トランザクションは，任意に複雑なオブジェクトに作用できる変換集合である．変換は 3 要素からなる．

* 選択パターン：画面上で選択されたオブジェクトとマッチするパータン，
* コマンド：変換の呼び出し，
* 置き換え動作：選択オブジェクトを修正．

図-5 は，UI 記述言語用構造エディタの画面を示している[20)]．

SG に基づく複雑なオブジェクトを扱う GUI システムについては十分に研究が進んでいない．

以下に，SG から派生した三つの子孫システムを紹介する．

B-1 microCOSM：microCOSM[21)] は，制約ベースのグラフィカルシステムである．ユーザは提供されるエディタを使って，たとえばテキストおよび図的記述で立体図形を生成したり制約を付けることができる．microCOSM エディタは属性評価にオフラインのアルゴリズムを使っている“怠け者の評価”法を使う．もう一つの特徴は，micro-

COSM の UI はオブジェクト指向パラダイムを反映しているがその実現は属性文法に基づいている．属性文法によって制約の表現や制約方程式の変数に関する型情報などの重要な意味情報を抽出することができる．

B-2 IDEAL Synthesizer: IDEAL Synthesizer[21]は，ユーザのテキスト記述入力に基づく図形オブジェクトの計算と表示をする対話型言語ベースの(テキストエディタ)システムである．テキストによる記述の適切な個所をマウスで指すことでユーザは図形表示を対話的に変えることができる．その変更で影響されるオブジェクト数に比例する時間を必要とする増分制約アルゴリズムを使って図形表示が更新される．Higgens と異なり，現在画面に表示されているかどうかにかかわらず変更される属性をもつオブジェクトに関係するすべてのオブジェクトの属性を再評価する（"勤勉者の評価"）．ユーザが IDEAL Synthesizer を呼び出すと，IDEAL 記述を入力するウィンドウと生成図形を表示するウィンドウが現れる．テンプレートを選択した後で，再帰的に他のテンプレートでスロットを置き換えるか正しいテキストで置き換えてユーザは IDEAL 記述を入力する．

B-3 CONSTRAINT system: Constraint system[22),23)] は X window system 上で動作し，グラフィクス AP に制約文法記述[23)]を使いその AP に対するマウスおよびメニュベースの UI を生成する．制約文法記述は，オブジェクト記述の集合とそれらオブジェクトをいかに操作できるかを記述する変換の集合からなる．

制約文法： 制約文法は，属性文法の生成規則と属性を使って AP のデータ構造を表現するとともに，制約方程式を使って AP の動的で図的な動作を表現する．リスト，木，および集合のような複雑なデータ構造を処理できる強力な変換ベースの編集モデルをもっている．UI の設計者と AP プログラマからみると，それらデータ構造がひとつのパッケージに入っていて，変換はそれらデータ構造を操作するメッセージまたは手続きである．属性は制約で関係付けられるデータ構造の外側との接触部分である．制約文法は，図形オブジェクトを記述するのに有効と考えられているいくつかのアイデアを取り入れている．たとえば，そのなかには部分一全体階層，準階層構造，多方向制約がある．

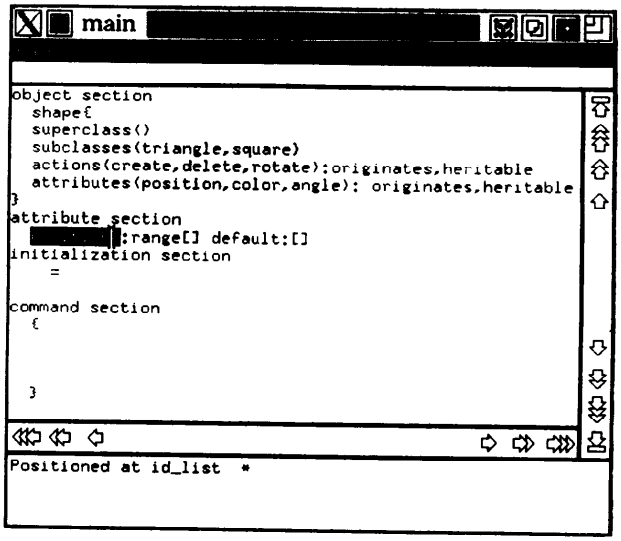


図-5 SynthesizerGenerator による構造エディタの例

部分一全体階層： この考えでは，オブジェクトは，既定義のオブジェクト，または点，テキスト，ビットマップなどのグラフィクスシステムがもつ基本オブジェクトの階層からなる．この階層構造を扱うために，制約文法は属性文法の考えを取り入れている．制約文法において，非終端記号はユーザ定義のオブジェクトに対応している．一方，終端記号は基本オブジェクトを表す．制約文法の各生成規則がオブジェクトをひとつ定義する．生成規則において，オブジェクトの名前を表す非終端記号が左辺にあり，構成要素を表す非終端と終端記号が右辺にある．

たとえば，生成規則

bintree ⟶ left_child : bintree
right_child : bintree

および

bintree ⟶ null_tree

は二分木が二つの表現をもつことを表している．すなわち表示される表現と空，すなわち表示されない表現である．

図-6 は，いかにこれらの生成規則がこの AP の内部表現を作るのに使われているかを表している．**図-6**(a)はユーザに対して外部的に提示されるときの二分木を表し，**図-6**(b)は内部的に表現

されるときの二分木を表している．

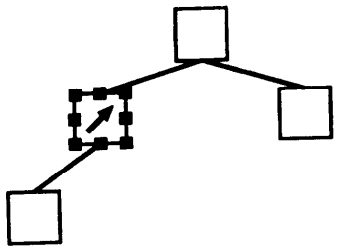
各非終端記号がもつ属性は，対応するオブジェクトのサイズ，次元などの配置に関する表示情報，およびたとえば，電気素子オブジェクトを流れる電流量などの意味情報を保持する．

準階層構造： 現実に発生する多くのAPを十分に表現するためには共有オブジェクトをもつ階層構造を使う．たとえば，電気回路における電気素子は共通の端子を必要とする．したがって，シミュレーションシステムでは“マージ”の考え方をよく持ち込む；同じ共通オブジェクトを表す二つのデータ構造を概念的にひとつの共通データ構造に結びつける．たとえば，抵抗と電池を接続するとこれら2素子を接続する2端子はマージされてひとつの端子になる．このマージの考えから準階層構造が発生した[24]．

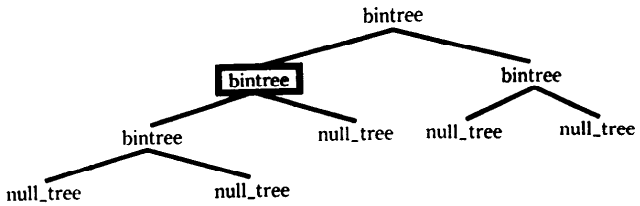
制約文法では，図形応用のオブジェクトの内部表現を木でなく有向グラフとすることができ準階層構造を扱える．言い換えると，制約文法の生成規則によるAPの記述から抽出される情報を有向グラフとして表すことができる．したがって，非終端のインスタンスは複数の親をもつことができる．このグラフの各ノードは生成規則の左辺にある非終端のインスタンスに対応し，その子孫は生成規則の右辺の非終端と終端のインスタンスに対応する．リーフノードは終端，内部ノードは非終端である．属性文法と同様に，非終端Xのインスタンスを表す各ノードはXの属性に対応する属性インスタンスの集合をもつ．

多方向制約： 多重制約を使って，準階層構造の属性，たとえば高さ，幅，画面上の位置などを関係づけることができる．それら関係には，二つのオブジェクトの並べ方のような構文的性質，回路を流れる電流量のような意味的性質の記述がある．電気回路に関する制約文法記述の例を図-7に示す[23]．

編集モデル： 制約文法のための編集モデルは“変換”に基づいている[17]．変換は，有向グラフのノードの集まりとコマンドをノードの変更集合へ写像する関数である．変換は，選択パターン，変換を呼び出すコマンド名，および置き換えアクションからなる．形式的には変換は次のように書かれる：



(a) A binary tree as it is externally presented to the user.



(b) A binary tree as it is internally presented to the application and interface.

図-6　二分木の内部表現と外部表現

```
electrical_component
  parts
    terminal 1 : node
    terminal 2 : node
    device : electrical_device;
  constraints
    device. terminal 1_voltage=terminal 1. voltage;
    device. terminal 2_voltage=terminal 2. voltage;
    device. current=terminal 1. current;
    /*Kirchhoff's Law*/
    terminal 1. current+terminal 2. current=0;
electrical_device
  attributes
      current : REAL; /*current flowing through the registor*/
      terminal 1_voltage : REAL; /*voltage at the two terminals*/
      terminal 2_voltage : REAL/*of the devices*/
  case : registor
    parts
      registance : INTEGER;
    constraints
      current*registance=terminal 1_voltage-terminal 2_voltage;
  case : battery
    parts
      voltage : REAL;
    constraints
      terminal 1_voltage=terminal 2_voltage+voltage;
  case : wire
    constraints
      terminal 1_voltage=terminal 2_voltage;
```

図-7　制約文法による電気回路の記述

transformation selection_pattern on
command_name {replacement rules}

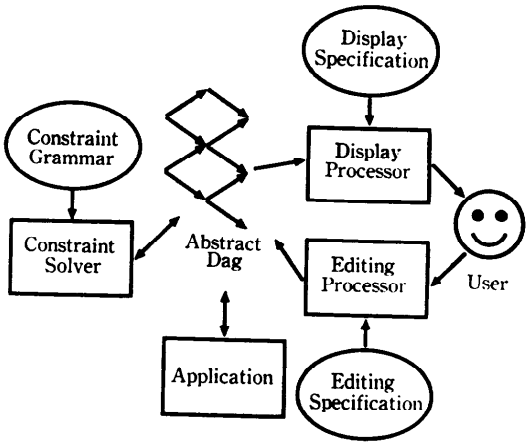
選択パターンは，ユーザが画面上で選んだオブジェクトにマッチする幾つかの階層的パターンである．パターンは，ルートにオブジェクト名すなわち非終端，子供として部分要素名(終端，非終端)をもつ木構造である．選択パターンは，変換を使う場合に選択オブジェクトが満たすべき条件の集合と考えることができる．あるコマンド名を選ぶと，有向グラフの選択された部分とマッチする選択パターンをもつ変換が呼び出される．そうすると，その変換に付随する置き換えアクションが選択オブジェクトを変更する．CONSTRAINT システムでは，三つの基本タイプのコマンドがある．すなわち delete, create, replace である．delete は再帰的にオブジェクトの部分要素を削除する．create と replace コマンドは，選択パターンと同じように構成されている階層的置き換えパータンに従う．たとえば，

create(bintree(bintree(null_tree, null_tree), null_tree))

は左の子をもつ二分木を生成する．

選択パターンが "bintree (left_child: bintree, right child: bintree)" の場合，replace (bintree (right_child, left_child)) は選択された木の左右部分木の順序を入れ替える．

CONSTRAINT システムの構造を図-8 に示す．記述は三つの要素からなる．すなわち制約文法，図形のイメージとオブジェクトを結びつける図形記述，および UI 上で実行可能な動作を記述する編集記述である．このシステムで生成される GUI の動作は次のとおりである：実働の UI を生成する CONSTRAINT のインタフェースジェネレータをとおしてこれら三つの記述が入力される．生成規則が抽象有向グラフを導出して，その有向グラフが実行中の AP のオブジェクトの図形構造を表す．ユーザは，画面上のオブジェクトを編集して AP と対話する．それらの動作により編集プロセッサが有向グラフを変更し，そのことで制約ネットワークのある部分が変更される．この変更で CONSTRAINT システムは，制約ソルバを呼び出し，ソルバは制約方程式を再充足させ有向グラフのリーフノードを変更する．そして，ディスプレイプロセッサが更新された属性値を使って画面を書き換える．



Organization of the CONSTRAINT System.

図-8 Constraint system のアーキテクチャ

3.3.3 手続き型と宣言型のアプローチの比較

手続き型アプローチの利点としては，システムが柔軟で広範囲の編集オペレーションを表現できることにある．欠点としては，各編集オペレーションは別々の手続きを使うことと，この手続きが AP のデータ構造のすべてを移動して適切な更新と表示変更をしなければならないことがある．さらに，これら手続きがそれ自身および他の手続きと協同して正しく動作することを確かめるためにテストが必要である．宣言型のアプローチの利点としては，設計者が AP のオブジェクト間の関係を記述するだけでよいことである．宣言型システムには各編集やアンドゥオペレーションのあとでこれら関係を自動的に再充足させるアルゴリズムがある．したがって，設計者がオブジェクト間の関係を正しく記述したかを調べるだけでよいので編集オペレーションの正しさを検証する時間量は結果として少なくなる．増分属性評価アルゴリズムを使うことにより各編集やアンドゥオペレーションのあとでオンラインで画面を更新することができる．欠点としては，グラフィクス環境において制約文法を除いて階層構造をもつオブジェクトの編集モデルがかなり弱い．制約文法についても，オブジェクト階層構造での継承・制約階層や構造的制約の強化が必要である．

4. おわりに

本稿では，UI と AP の分離の問題点，および UI と AP 間の通信機構からみた既存の UIMS の特徴について述べた．これからの課題の幾つかを以下に示す．

* 分離の尺度と評価法の確立，
* 機能分散 UIMS 設計法の確立，
* 実用的な UI と既存 AP との結合方法，
* テキストベースと図形の直接操作による編集システムの共存（混在），
* AP 領域の知識と図形階層構造が共存する UIMS における通信制御，設計環境．

おわりに本稿の図を作成してくれた土肥浩君に感謝する．

参考文献

1) Dance, J. R., Granor T. E., Hill, R. D., Hudson, S. E., Meads, J., Myets, B. A. and Schulert, A.: The run-time Structure of UIMS-Supported Applications, Computer Graphics, Vol. 21, No. 2, pp. 97-101 (1987).
2) 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).
3) Duce, D. A., Gomes, M. R., Hopgood, F. R. A. and Lee, J. R. eds: User Interface Management and Design, Springer-Verlag (1991).
4) Green, M.: A Survey of Three Dialog Models, ACM Trans. on Graphics, Vol. 5, No. 3, pp. 21-29 (1986).
5) Hutchins, E. L., Hollan, J. D. and Norman, D. A.: Direct Manipulation Interfaces, In User centered system design, Norman, D. A. and Draper, S. W. eds, Lawrence Erlbaum Associates, pp. 87-124 (1986).
6) 今宮淳美：ユーザインタフェース管理システム，情報処理ハンドブック，情報処理学会編，オーム社，pp. 1194-1201 (1989).
7) Hartson, R.: User-Interface Management Control and Communication, IEEE Software, Vol. 6, No. 1, pp. 62-70 (1989).
8) Shneiderman, B.: Direct Manipulation: A Step beyond Programming Languages, IEEE Computer, Vol. 16, No. 8, pp. 57-69 (1983).
9) Jacob, R.: A Specification Language for Direct-Manipulation User Interfaces, ACM Trans. on Graphics, Vol. 5, No. 4, pp. 283-317 (1986).
10) Szekely, P. and Myers, B. A.: A User Interface Toolkit Based on Graphical Objects and Constraints, OOPSLA '88 Proceeding, pp. 36-45 (1988).
11) Olsen, D. R.: Editing Templates: A User Interface Generation Tool, IEEE CG&A, Vol. 6, No. 11, pp. 40-45 (1986).
12) Hudson, S. E. and King, R.: Semantic Feedback in the Higgens UIMS, IEEE Trans. on Software Engineering, Vol. 14, No. 8, pp. 1188-1206 (1988).
13) Hull, R. and King, R.: Semantic Database Modeling: Survey, Application, and Research Issues, ACM Computing Surveys, Vol. 20, pp. 201-260 (1988).
14) Data, C. J.: An Introduction to Database Systems, Addison Wesley (1981).
15) Winston, P. H.: Artificial Intelligence, Addison Wesley (1977).
16) Knuth, D. E.: Semantics of Context-Free Languages, Math. Syst. Theory J., Vol. 2, No. 2, pp. 127-145 (1968).
17) Reps, T. and Teitelbaum, T.: The Synthesizer Generator, A System for Construction Language-Based Editors, Springer-Verlag (1988).
18) 渡辺喜道，今泉貴史，徳田雄洋：構造エディタ生成系，情報処理，Vol. 32, No. 3, pp. 282-294 (1991).
19) エイホ，A. V.，セシイ，R.，ウルマン，J. D.（原田賢一訳）：コンパイラ I，第5章構文主導翻訳，サイエンス社 (1990).
20) 土肥 浩：シンセサイザジェネレータによる IDL+ 用構造エディタの作成，山梨大学電子情報工学科卒業論文 (1992).
21) Barford, L. A. and Vander Zanden, B. T.: Attribute Grammars in Constraint-Based Graphics Systems, Software-Practice and Experience, Vol. 19, No. 4, pp. 309-328 (1989).
22) Vander Zanden, B. T.: Incremental Constraint Satisfaction and its Application to Graphical Interfaces, PhD Dissertation, Cornell University (1989).
23) Vander Zanden, B. T.: Constraint Grammars—A New Model for Specifying Graphical Applications, CHI '89 Proceedings (1989).
24) Sussman, G. J. and Steele, G. L.: CONSTRAINTS—A Language for Expressing Almost-Hierarchical Descriptions, Artificial Intelligence, Vol. 14, pp. 1-39 (1980).

（平成4年7月13日受付）

今宮 淳美（正会員）

1945年生．1968年東北大学通信工学科卒業．1973年東北大学博士課程修了．工学博士．1973年東北大学助手．1975年山梨大学計算機科学科助教授．現在，山梨大学電子情報工学科教授．研究：図形処理と対話システム，ヒューマンインタフェース，アルゴリズム，ACM, IEEE, 人工知能学会各会員．