仕様記述変換に基づく対話型ユーザインタフェース設計システム

渡辺喜道[†*] 今宮淳美[†] 三宅一巧[††]

本論文では，Xウィンドウシステム上のグラフィカルユーザインタフェース環境 OSF/Motif でユーザインタフェースを設計するためのモデルの提案，およびそのモデルに基づくユーザインタフェース設計支援ツールの実現について述べる．ユーザインタフェース設計者は，この設計モデルに従って，ユーザインタフェースを対話の意味と構造の二つに分けて設計する．意味設計では，アプリケーションの持つ概念や機能を設計する．構造設計では，ユーザインタフェースの外観やインタラクション技法を設計する．本研究で作成した支援ツールは，これら二つの設計を支援する．意味設計支援ツールは，意味の抽象的な仕様記述を機能的に等価な複数の仕様記述に変換する機能を持つ．構造設計支援ツールは，意味設計支援ツールが出力した仕様記述を解析し，必要に応じて設計者と対話的な処理で情報を補いながら，OSF/Motif 環境に基づく実行可能コードを生成する．したがって，一つの仕様記述から異なるパラダイムに基づくさまざまなユーザインタフェースの実行可能コードを設計者は比較的容易に生成することができる．

An Interactive User Interface Design System Based on the Transformations of a Specification

Yoshimichi Watanabe,[†*] Atsumi Imamiya[†] and Kazuyoshi Miyake[††]

This paper provides an interactive user interface design model for the graphical user interface environment OSF/Motif on the X window system, and an implementation of tools based on the model. According to the design model, user interface designer can design a user interface in semantics and forms phases. The concept and functions of an application are specified in semantics phase, and views and interaction techniques are specified in forms phase. The tool for semantics allows designer to generate abstract semantic specifications of an application from one specification. The tool for forms allows designer to analyze the output of the semantics design subsystem and create the executable codes on the OSF/Motif. User interface designer, therefore, can easily generate several kinds of the user interface code which have same functions.

1. はじめに

最近のユーザインタフェースへの要求の複雑化にともないユーザインタフェース設計者（以下では，設計者と略す）の負担を軽減させるために，さまざまなインタフェースビルダやXツールキットなどが開発されている．しかし，ユーザインタフェースを操作対象とその対象に関する機能である対話の意味，およびスクリーンレイアウトや対話の流れである対話の構造に分けた場合，多くのユーザインタフェース設計ツールやシステムは構造設計を支援するが，意味設計支援を直接提供していない．

また，OPEN LOOK や OSF/Motif などのウィンドウベースのグラフィカルインタフェース作成ツールの標準化が提案されているが，それら標準化ツールに基づく総合的なユーザインタフェース管理システムの実現[5),8)]はまだない．

そこで，本論文では，はじめにXウィンドウシステム[6)]上の OSF/Motif グラフィカルユーザインタフェース作成環境におけるユーザインタフェース設計モデルを提案する．さらに，このモデルに基づくユーザインタフェース設計支援ツール群の作成およびそれらに基づくユーザインタフェースの設計手順について述べる．この支援ツールは，ユーザインタフェースの二つの部

† 山梨大学工学部電子情報工学科
Department of Electrical Engineering and Computer Science, Faculty of Engineering, Yamanashi University
†† 日本電気株式会社 C&C システム事業グループ装置システム事業部第1システム部
NEC
* 現在，東京工業大学工学部情報工学科
Currently with Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology

分である「対話の意味」と「対話の構造」の設計を支援するツールから成る．対話の意味設計支援ツールでは，意味の抽象的な仕様記述を機能的に等価な複数の仕様記述に変換する機能を持つ．そのため，これらのツール群を用いることにより，一つの記述から異なるパラダイムに基づくさまざまなユーザインタフェースの実行可能なコードを生成できるという特徴を持つ．

本論文の構成は次の通りである．2章では，新しいユーザインタフェース設計モデルを提案する．3章では，このモデルに基づく処理系が認識できる意味設計用の仕様記述言語について述べる．4章では，パラダイムの異なる対話意味の仕様記述を得るための変換アルゴリズムについて述べる．5章，6章では，意味設計支援ツール，構造設計支援ツールについてそれぞれ述べる．最後に，7章では，まとめと今後の課題について述べる．

2. ユーザインタフェース設計モデル

ここでは，本研究の基礎となる4レベルユーザインタフェース設計モデルと新たに本論文で提案する意味・構造設計モデルについて述べる．

2.1 4レベルユーザインタフェース設計モデル

1987年に Foley はユーザインタフェースを効率良く，系統立てて，素早く構築する4レベルの設計モデルを提案した[2),3)]．この設計モデルでは，ユーザインタフェースの設計を概念・機能・順序化・結合の四つのレベルに分け，設計者は概念設計から順に設計し，目的とするユーザインタフェースを作成する．それらの各レベルの設計内容を以下に示す．

1. 概念設計レベル
   概念設計レベルでは，ユーザが理解すべきアプリケーションの概念を定義する．すなわち，アプリケーションのユーザモデルを定義する．典型的には，オブジェクトとそのクラス，オブジェクト間の関係，各オブジェクトに関するコマンドを定義する．
2. 機能設計レベル
   機能設計レベルでは，各コマンドに対して詳細な機能の設計と意味付けを行う．すなわち，各コマンドを実行するために必要な情報，起こり得るエラーとその処理方法，各コマンドの実行結果などを定義する．ここでは，意味付けだけを定義し，実際に入出力を処理するデバイスを定義しないことの注意されたい．この部分は，従来の意味設計[5)]に相当する．
3. 順序化設計レベル
   順序化設計レベルでは，入出力列を定義する．入力の定義ではインタラクションタスクを実行するための入力列，出力の定義では表示のための出力列をそれぞれ定義する．これらの列の概念には，空間的・時相的要因も含まれる．この部分は，従来の構文設計[5)]に相当する．
4. 結合設計レベル
   結合設計レベルでは，順序化設計レベルで定義した入出力の単位が，実際に使用するハードウェアプリミティブからどのように構成されるかを定義する．入力プリミティブは，利用可能な入力デバイスで，出力プリミティブはグラフィクスサブルーチンパッケージが提供する形状や属性である．すなわち，ここでの設計は，入力に対してはインタラクション技法の設計や選択であり，出力に対してはアイコンや他のシンボルを形成するための出力プリミティブと属性の組み合わせを定義する．この部分は，従来の字句設計[5)]に相当する．

2.2 意味・構造設計モデル

OSF/Motif[9)] では従来の X ツールキット[13),14)] と同様に，ユーザインタフェースをウィジェットと呼ぶ部品（ボタンやメニューなど）の組み合わせで定義する．そのため，4レベルユーザインタフェース設計モデルにおける機能設計レベルで定義する情報は少なくなる．たとえば，ユーザからのイベントに対するフィードバックはウィジェットのクラスの動作としてあらかじめ規定されているので，新たに定義する必要がない．また，インタラクションタスクに対して適切なウィジェットを選択することによりエラーを回避できる．したがって，起こり得るエラーの数は少なくなり，処理方法などの定義はほとんど不要となる．このように，4レベルユーザインタフェース設計モデルの機能設計レベルで設計する内容は非常に少ない．

また，本研究では OSF/Motif のスタイルガイドに準拠してユーザインタフェースを構築する方針であるので，インタフェースの構成およびレイアウトにいくつかの制約を設ける．このため，順序化設計レベルと結合設計レベルを明確に分離できなくなる．たとえば，「メニューバーの項目を選択するとプルダウンメニューが表示されなければならない．」という制約を適用すると，順序化設計レベルが結合設計レベルに依存するので，明確な分離ができない．

以上の考え方から本研究では，4レベルユーザインタフェース設計モデルの概念と機能の各設計レベル，および順序化と結合の各設計レベルをそれぞれ組み合わせて，前者を意味設計，後者を構造設計と呼ぶことにする．この2段階のユーザインタフェース構築モデル「意味・構造設計モデル」を**図1**に示す．このモデルに基づくシステム[10),12)]における典型的なユーザインタフェースの設計過程を以下に示す．ユーザインタフェースの設計・開発では，プロトタイプの作成とその評価を試行錯誤的に繰り返してユーザインタフェースを構築してゆくのが一般的である．本研究では，図1に示す二つのユーザインタフェース設計支援ツール（意味設計支援ツールと構造設計支援ツール）を提供して，この設計作業を軽減する．

設計者は，まず意味設計を行う．意味設計では，まず，後述の意味設計仕様記述言語を用いて，アプリケーションの基本的な仕様を記述する．そして，意味設計支援ツールにこの仕様記述を入力して，必要に応じて機能的に等価な別の仕様記述へ対話的に変換する．

次に，構造設計支援ツールを用いて，意味設計後の仕様記述を解析し，ユーザインタフェースの実行可能コードを生成する．この設計で，エンドユーザからみたユーザインタフェースの概念が構築される．構造設計支援ツールでは，意味設計後の仕様記述のみでは自動的に処理できない部分を設計者に対話的に問いかけながら必要な情報を補い，実行可能コードを生成する．

最後に，構造設計支援ツールが生成したコードをコンパイル・実行し，ユーザインタフェースを評価する．この結果，仕様を変更する場合は，意味設計支援ツールを再度用いて仕様記述を変換し，また同様な手順を繰り返す．設計者はこのような手順でユーザインタフェースを開発する．

3. 意味設計仕様記述言語

本論文で提案するモデルに基づいてユーザインタフェースを作成するためには，まず意味設計における仕様書を作成しなければならない．この章では，そのための仕様記述言語 IDL＋ について説明する．意味設計で定義する内容は簡単に前章で述べたが，その内容を言語 IDL＋ により形式的に記述する．言語 IDL＋ は Gibbs らが提案したインタフェース定義言語 IDL[4)] を，OSF/Motif のスタイルガイドに準拠したユーザインタフェースを作成するために必要な概念や機能を導入して著者らが拡張修正した言語である．主な拡張修正点を以下に示す．

- オブジェクトの属性の型に文字型を追加
- コマンドの階層構造の概念の導入
- 条件式に比較演算子を追加
- 文をC言語風に変更

IDL＋ による意味設計の仕様記述は，次の四つの部分からなる．以下にその主な記述内容を示す．

1. オブジェクト定義部
   - オブジェクトの親子関係の定義
   - オブジェクトの属性の定義
   - オブジェクトに対するコマンドの定義
2. 属性定義部
   - 各オブジェクトの属性とその値の範囲の指定
3. 変数初期化部
   - 変数の初期値の設定
4. コマンド定義部

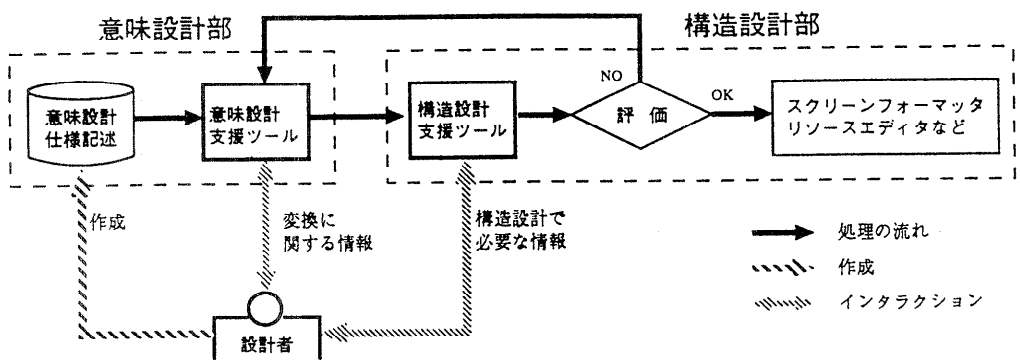


図1　ユーザインタフェース設計システム
Fig. 1　User interface design system.

- コマンドのグループの定義
- コマンド名とそのパラメータの定義
- 前条件（コマンド実行直前の条件）の指定
- 後条件（コマンド実行直後の条件）の指定

指定色で色塗りした三角形と正方形の生成，削除，および回転のできるアプリケーションの意味設計のIDL＋ による仕様記述例を**図2**に示す．

オブジェクト定義部では，shape, triangle, square の三つのオブジェクトを定義する．オブジェクト shape は二つのサブクラス triangle と square を持つ．さらに，図形の生成，削除，および回転の三つのコマンド，属性として位置，色，および角度を持つ．これらのコマンドや属性をサブクラスに継承することができる．また，オブジェクト triangle と square はスーパークラスとして shape を持ち，スーパークラス shape から三つのコマンドおよび三つの属性を継承する．

属性定義部では，三つの属性 position, color, angle を宣言する．位置を表す属性 position の値として，2次元配列で [0, 0] から [10, 10] までの値をとることができる．色を表す属性 color の値として，赤，緑，青，白，黒，シアン，マゼンタ，黄のうちの1色を指定できる．角度を表す属性 angle の値として，0から360の間の値を指定できる．

変数初期化部では，オブジェクト図形数を表す変数 num_shape の値がアプリケーション起動時に0であることを示す．

コマンド定義部では，図形の生成，回転，および削除の各コマンドを定義し，すべてを編集コマンドのグループ名である edit のメンバーであることを定義する．たとえば，図形を削除するコマンド delete は図形名をパラメータに持つ．コマンド実行の直前に真であるべき前条件では図形数が0でないこと，コマンド実行直後に成立すべき後条件では図形数が一つ少なくなることを示している．また，/* と */ で囲まれた部分は注釈である．

4. 仕様記述の変換

同じ機能を持つアプリケーションに対して，前節で述べたようなユーザインタフェースの意味設計の仕様記述は複数存在する．たとえば，図2の変数初期化部とコマンド定義部は**図3**に示す仕様でも表すことができる．この場合，図2と図3の記述によって定義されるユーザインタフェースは機能的には等しいが仕様は異なる．

```
object section
  shape {
    superclass ()
    subclasses (triangle, square)
    actions (create, delete, rotate)
          :originates, heritable
    attributes (position, color, angle)
          :originates, heritable
  }
  triangle, square {
    superclass (shape)
    subclasses ()
    actions (create, delete, rotate)
          :inherits from shape, not heritable
    attributes (position, color, angle)
          :inherits from shape, not heritable
  }
attribute section
  position:range[x[0..10], y[0..10]]
  color:set[red, blue, green, white, black,
          cyan, magenta, yellow]
  angle:range[0..360]
initialization section
  num_shape = 0
command section
  edit {
    create(p:position, c:color, a:angle, t:type_of_shape)
    postcondition(num_shape += 1)

    precondition(num_shape != 0)
    rotate(obj:shape, a:angle)

    precondition(num_shape != 0)
    delete(obj:shape)
    postcondition(num_shape -= 1)
  }
```

図 2 IDL+ による仕様記述の例
Fig. 2 An example of the specification in IDL+.

```
initialization section
  color_set = false /* 新しく追加された初期値の設定 */
  num_shape = 0
command section
  edit {
    set_color (c:color) /* 新しく追加されたコマンド */
    postcondition(color_set = true)
                        /* 新しく追加された後条件 */
    precondition(color_set == true)
                        /* 新しく追加された前条件 */
    create(p:position, c:color implicit, a:angle,
          t:type_of_shape) /*implicit が追加された */
    postcondition(num_shape += 1)

    precondition(num_shape != 0)
    rotate(obj:shape, a:angle)

    precondition(num_shape != 0)
    delete(obj:shape)
    postcondition(num_shape -= 1)
```

図 3 属性 color を因子化した後の仕様記述
Fig. 3 A specification after factoring color.

ユーザインタフェース作成環境において，一つのアプリケーションのための一つの仕様記述を作成しユーザインタフェースを作成した後に仕様を変更し再度ユーザインタフェースを作成することは，ユーザインタフェースの評価法が確立されていない現在ではよくあることである．この作業は，パラダイムや構造，機能が変化するたびごとにはじめから作成しなければならなく，非常に手間のかかる作業である．そこで，一つの仕様記述から複数のパラダイムの異なる仕様記述を自動的に変換できれば，この作業の軽減化が図れる．この章では，そのための変換技法の一つである因子化について述べる[11),15)]．

4.1 因 子 化

ある環境で一つのゴールへ到達する場合，その環境やパラメータを操作することにより，複数の手段（記述）でゴールを達成できることが多い．たとえば，前例で青色の正方形を座標 [5, 5] に角度 45 度で生成する場合，“生成（[5, 5]，青，45，正方形）”で生成できると仮定する．このとき，この目標を達成するための一つの方法は，このコマンドを直接実行することである．また，別の方法として，まず正方形オブジェクトに焦点を当て（選択し），その上で“生成（[5, 5]，青，45）”を実行することである．この時，正方形オブジェクトをコマンドから因子として取り出したという．一般に，パラメータをコマンドから取り出し，その情報（パラメータ）に関する環境の設定・解除のコマンドを定義し，その環境の設定，本質的なコマンドの実行，環境の解除の順に評価するような仕様記述に変換することを因子化と呼ぶ[3)]．あるパラメータを因子化して値が設定されると，設定または解除コマンドがこれを変更しない限り，もとのコマンド実行時にパラメータ値を設定する必要はない．コマンドから因子化されて取り出された情報を陰的と呼び，そうでない情報を陽的と呼ぶ．仕様記述中では陰的な情報を予約語 implicit で後置修飾して表す（たとえば，図 3 におけるコマンド create の属性 color*）．この因子化は，一見複雑な変換に見えるが，同じ環境で少しだけ異なるオブジェクトを生成する場合には便利な変換である．

本研究で現在までに作成したシステムで使用可能な因子化であるオブジェクト属性およびオブジェクト自身，コマンドの 3 種類の因子化を以下に説明する．この 3 種類の変換の組み合わせでほとんどのパラダイムに基づく仕様記述が生成可能である．

4.2 オブジェクト属性の因子化

因子化されるパラメータが属性の場合，この因子化をオブジェクト属性の因子化と呼ぶ．図 3 は，属性 color を因子化した例である．因子化によって変更される部分は，明らかに変数初期化部とコマンド定義部だけであるので，図 3 では他の定義部を省略してある．色が設定されているかどうかを表す変数 color_set が変数初期化部に，色を設定するためのコマンド set_color がコマンド定義部にそれぞれ加えられている．また，コマンド create に前条件が加えられたこと，および属性 color が陰的になっていることに注意されたい．

一般に，次の 1.〜3. の手順でオブジェクト属性を因子化することができる．因子化する属性を fa とする．

1. 変数の初期値設定を加える．
   fa_set＝false
2. コマンド定義を加える．
   set_fa(att : fa)
   ただし，att は適当な変数である．
   postcondition(fa_set＝true)
3. 2. で加えたコマンド以外のパラメータとして fa を持つすべてのコマンド定義に対して，
   （a） fa を陰的にする．
   （b） 前条件を加える．
   precondition(fa_set＝＝true)

4.3 オブジェクトの因子化

因子化されるパラメータがコマンドの操作対象のオブジェクトであるとき，この因子化をオブジェクトの因子化と呼ぶ．この因子化により，現選択オブジェクト (Currently Selected Object) パラダイムに基づくユーザインタフェースを記述できる．オブジェクトを因子化することにより，選択されたオブジェクトに対して，様々な操作を連続的に実行することができる．

オブジェクトの属性の因子化と同様に，次の 1.〜7. の手順でオブジェクトを因子化することができる．因子化するオブジェクトを fo とする．

1. 前条件の初期値設定を加える．
   CSO_set＝false
2. コマンド定義を加える．
   precondition(num_fo !＝0)
   select_fo(obj : fo)

---

* 正確にはその属性の実質的な変数が陰的になると表現すべきだが，以下では省略し上記のような表現を用いる．

postcondition(CSO_set＝true)

ただし，obj は適当な変数である．

3. 前条件 precondition(num_fo！＝0) を持つすべてのコマンド定義に対して，その前条件を次のように変更する．
   precondition(CSO_set＝＝true)
4. 前条件 precondition(num_fo＝＝0) を持つすべてのコマンド定義に対して，その前条件を次のように変更する．
   precondition(CSO_set＝＝false)
5. 後条件 postcondition(num_fo－＝1) を持つすべてのコマンド定義に対して，その後条件を次のように変更する．
   postcondition(CSO_set＝false)
6. 後条件 postcondition(num_fo＋＝1) を持つすべてのコマンド定義に対して，後条件を加える．
   postcondition(CSO_set＝true)
7. 2. で加えたコマンド以外のパラメータとして fo を持つすべてのコマンド定義に対して，fo を陰的にする．

4.4 コマンドの因子化

パラメータばかりでなくコマンド自身も因子化することができる．この因子化により，コマンドモードの概念に基づくユーザインタフェースを記述できる．コマンドを因子化することにより，選択されたコマンドを多くのオブジェクトに対して連続的に適用させることができる．

次の 1.～4. の手順でコマンドを因子化することができる．因子化するコマンドを co とする．

1. 前条件で用いる変数の初期値設定を加える．
   co_mode_set＝false
2. co 以外のすべてのコマンド定義に対して，前条件を加える．
   precondition(co_mode_set＝＝false)
3. 2つのコマンド定義を加える．
   （a） 選択状態にするためのコマンド
   precondition(co_mode_set＝＝false)
   begin_co_mode
   postcondition(co_mode_set＝true)
   （b） 選択状態から解放するためのコマンド
   precondition(co_mode_set＝＝true)
   end_co_mode
   postcondition(co_mode_set＝false)
4. co に対して，
   （a） 前条件を加える．
   precondition(co_mode_set＝＝true)
   （b） co を陰的にする．

5. 意味設計支援ツール

前に述べたように，同じ機能を持つアプリケーションに対して，意味設計の仕様記述は一般に複数種類存在する．ユーザインタフェースの仕様を一部変更するたびに，最初から意味仕様を書き換えることは，設計者にとって，非常に手間のかかる作業である．また，設計者は一つのアプリケーションの意味を構成する複数の表現を検討しなければならない．しかし，一般に複数の表現を考え出すことは難しい．そこで，本論文では，もととなる考え方である一つの仕様記述から半機械的に別のユーザインタフェース設計を支援するための仕様記述生成を支援するツールを開発した．ここでは，前に述べた各因子化を支援するツールについて述べる．設計者はこのツールを用いて意味設計の仕様記述を対話的に変換することができる．

このツールの実現のために，ユーザインタフェース作成にXツールキット，構文解析作成に yacc と lex[1]，その他の部分の作成に C 言語[7]を用いた．このツールの主な機能を以下に示す．

1. 意味仕様記述の構文エラー検出
   仕様記述の構文に合わない記述に対して，記述の順序などの基本的な文法的エラーを検出して，設計者にエラーメッセージを表示する．しかし，構文は単純であるので，一般には設計者にとってあまり負担とはならない．
2. 意味仕様記述の静的意味エラー検出
   コマンド定義部で用いられているオブジェクトや属性が定義されていない場合，または多重定義されている場合にエラーメッセージを表示する．
3. 後条件の正当性検査
   あるコマンド定義の後条件で用いられている変数が他のコマンド定義の前条件で使用されていない場合に警告する．この機能により，意味仕様記述の意味的誤りを減らすことができる．
4. 3種類の因子化の支援
   前に述べた属性，オブジェクト，およびコマンドの三つの因子化を各々，複数同時に，または連続的に繰り返して実行することができる．
5. 変換時の動的な意味検査
   変換の種類と対象とするパラメータの型の不一致

などを検出し，エラーメッセージを表示する．この機能により不適切な変換を防止でき，適切なユーザインタフェースのための意味仕様記述を得ることができる．

6. 意味仕様記述ファイルに対する操作
このツールでは，意味仕様記述をファイルとして扱う．そのため，そのファイル操作のための基本的なコマンドがツール中に組み込まれている．もちろん，この組込みコマンドを用いずに，単なるテキストファイルとみなして操作することも可能である．

7. ツールの対話的・視覚的な実行
図4に示すように，マウスによるメニュー選択やパラメータ指定を用いて対話的に実行ができる．

6. 構造設計支援ツール

この章では，ユーザインタフェースの意味仕様記述を解析し，設計者と対話しながら，ユーザインタフェースの外観部である構造を定義し，さらに実行可能コード生成を支援するツールについて述べる．

6.1 ユーザインタフェースの生成

一般に，OSF/Motif 上のツールキットを用いて様様なスタイルのユーザインタフェースを生成することができる．これは生成可能なユーザインタフェースの自由度が大きいという利点を持つ一方，ユーザインタフェースの一貫性の観点からは必ずしも好まれるとは限らない．そこで本研究では，OSF/Motif のスタイルガイドに準拠したユーザインタフェースを作成するために，ユーザインタフェースに制約を加える．ただし，ここで加える制約は，ほとんどのユーザインタフェースにとって，支障をきたさないことに注意されたい．アプリケーションは次のようなユーザインタフェースを持つとする．

アプリケーション起動時には，メインウィンドウが表示される（図4奥のウィンドウ）．メインウィンドウはメニューバーと情報表示領域からなる．情報表示領域には必要に応じてスクロールバーを付加することができる．

メニューバーの各項目をマウスでクリックするとプルダウンメニューを表示する．すべてのコマンドはプルダウンメニューから選択される．プルダウンメニューの各項目をクリックしてコマンドを選択する．パラメータを必要とする場合にはダイアログボックスがポップアップされる（図4手前のウィンドウ）．ダイアログボックスはボタンやテキストの入力領域などのコマンドのパラメータ入力のための複数ウィジェットからなる．このツールで提供するパラメータの入力のために使用するウィジェットを**図5**に示す．また，ダイアログボックスの中にはパラメータ入力用のウィジェットの他にOKとCancelの二つのボタンがある．OKボタンをクリックすると，ダイアログボックスでの入力をパラメータ値としてコマンドを実行し，Cancelボタンをクリックすると各入力値をキャンセルし，コマンドは実行されない．

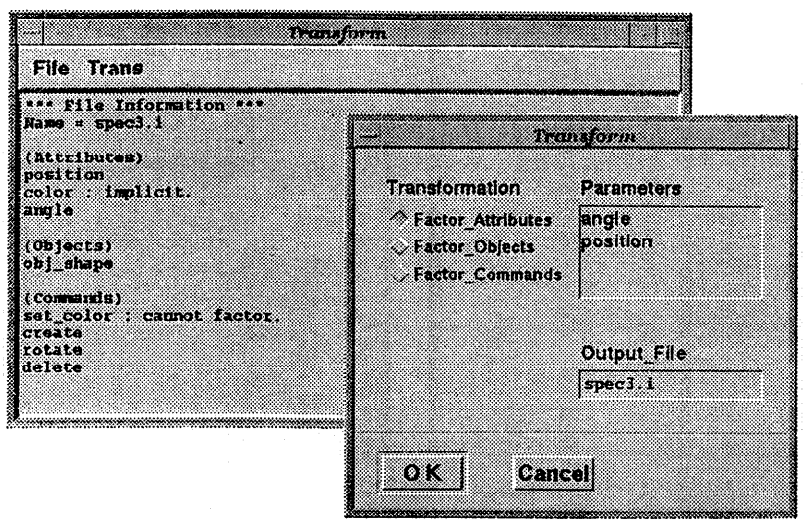


図4 アプリケーションのユーザインタフェース
Fig. 4 User interface of an application.

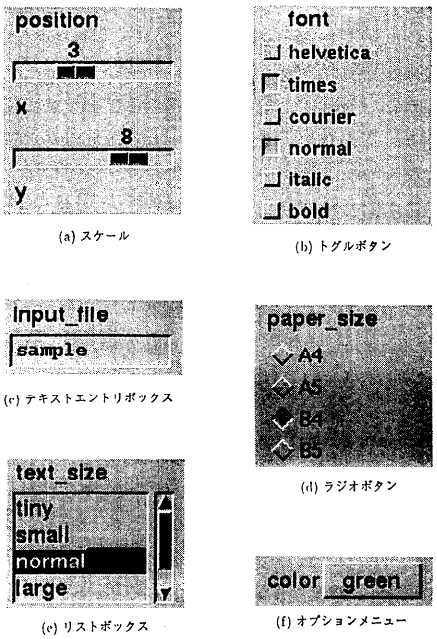
6.2 支援ツールに基づく構造設計手順

構造設計支援ツールを用いてユーザインタフェースの構造を設計するための手順を以下に示す.

1. 意味設計後の仕様記述ファイルの入力
   ファイル選択用のダイアログボックスから，構造設計に必要な仕様記述ファイルを選択すると，そのファイルは構造設計支援ツールに読み込まれる．それと同時にツールは仕様記述の構文等の解析を行い，このツールで必要となる情報を収集する．得られた情報に基づいて，生成するユーザインタフェースのメニュー階層やラベルなどが決定される．たとえば，コマンドのグループ名はメニューバーのラベル，コマンド名はプルダウンメニ

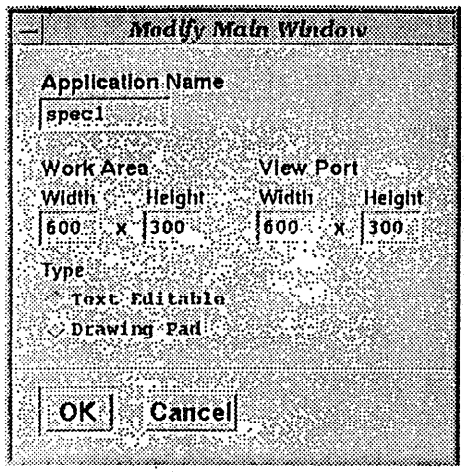
表 1 パラメータの型とウィジェットの対応
Table 1 Parameter types and widgets.

パラメータの型	ウィジェット
数　値	スケール テキストエントリボックス
文字列	テキストエントリボックス
集　合 (1 from N)	ラジオボタン リストボックス オプションメニュー
集　合 (M from N)	チェックボタン リストボックス
オブジェクト インスタンス	テキストエントリボックス リストボックス
オブジェクト クラス	テキストエントリボックス ラジオボタン リストボックス オプションメニュー

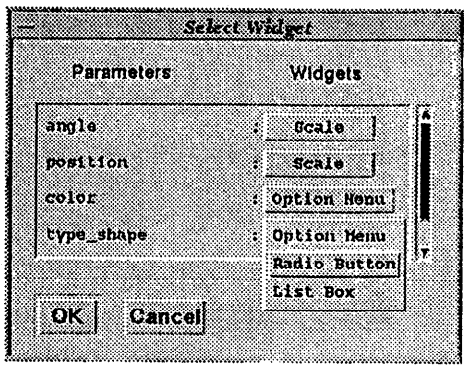


(a) スケール　(b) トグルボタン　(c) テキストエントリボックス　(d) ラジオボタン　(e) リストボックス　(f) オプションメニュー

図 5 パラメータ入力用ウィジェット
Fig. 5 Widgets for inputting parameters.
(a) Scale.
(b) Toggle button.
(c) Text entry box.
(d) Radio button.
(e) List box.
(f) Option menu.



(a) メインウィンドウの設定
(a) Main window setting.

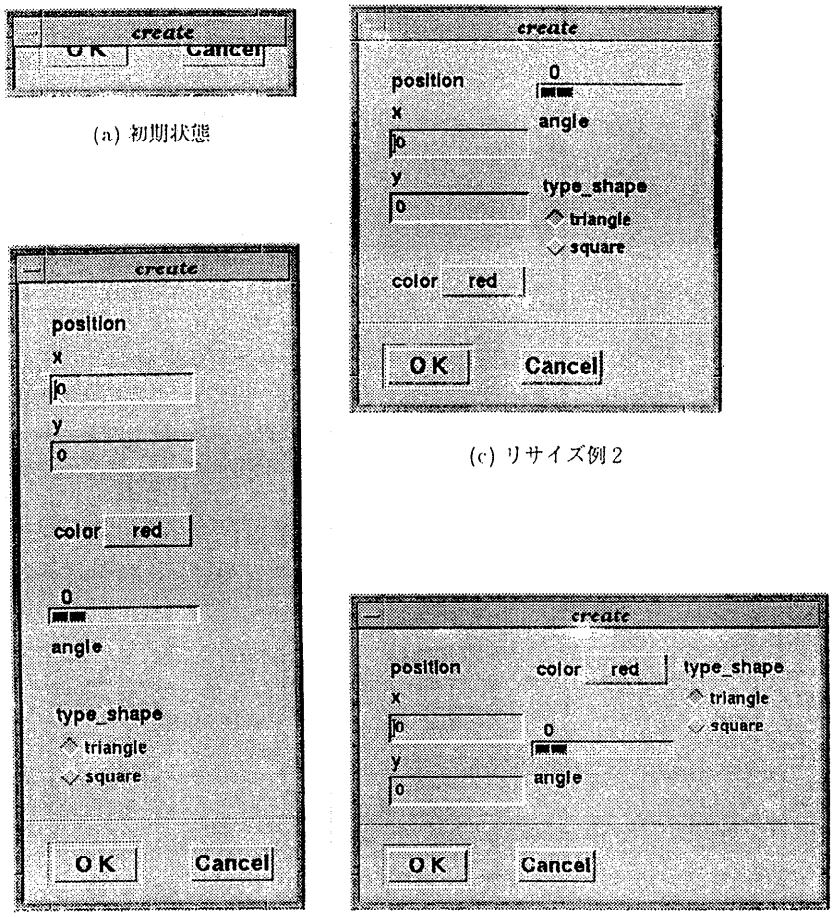


(b) パラメータ入力用ウィジェットの選択
(b) Widget selection for parameters inputs.

図 6 メニューによるパラメータの入力
Fig. 6 Parameters inputs by menu.

ューの選択項目のラベルである．また，コマンドの各パラメータの型解析により，そのパラメータ入力で使用可能なウィジェットを決定する．各パラメータの型に対応するウィジェットを**表1**に示す．

2. メインウィンドウの設定
アプリケーションの名前と情報表示領域の起動時の大きさとその型を設定する（**図6**(a)）．情報表示領域の型には，テキスト編集を可能にする型（Text Editable 型）とテキスト編集はできないが表示は可能で，ビックスマップが表示できる型（Drawing Pad 型）の2種類がある．設計者は，この2種類のうちいずれか一方を選択しなければならない．
3. パラメータ入力用ウィジェットの選択
各パラメータ入力用のウィジェットをオプションメニューから選択する（図6(b)）．これらダイアログボックス中のオプションメニュー群は，仕様記述ファイルを読み込んだときの解析により自動的に生成される．
4. ダイアログボックスの大きさの指定
生成するユーザインタフェースの各コマンドに対



(a) 初期状態　(c) リサイズ例2

(b) リサイズ例1　(d) リサイズ例3

図 7 ダイアログボックスのリサイズ
Fig. 7 Resize of a dialogue box.
(a) Iitial state.
(b) Resized dialogue box 1.
(c) Resized dialogue box 2.
(d) Resized dialogue box 3.

応するダイアログボックスの大きさを直接操作で設定する．ダイアログボックスの外枠をドラッグして適当な大きさに変更すると，それに伴い内部のウィジェットのレイアウトも変化する（**図7**）．これは，ウィジェット間に表示用の制約が働いているためである．

5. ソースファイルの生成
入力した仕様記述ファイルと設計者の各種の設定に基づき，ユーザインタフェースのソースファイルを生成する．生成するファイルの詳細については6.3節で述べる．
6. ユーザインタフェースのテスト
このツールは生成したインタフェースのソースファイルをコンパイル・実行する機能を持っているので，生成したユーザインタフェースをただちにテストできる．

6.3 ツールの出力

構造設計支援ツールは次の5種類のファイルを出力する．

1. UIL ファイル
使用したウィジェットの親子関係，リソース，コールバック（そのウィジェットから呼び出すアプリケーションの関数）を定義したファイルである．このファイルは OSF/Motif のユーザインタフェース定義言語 UIL で記述される．
2. メインファイル
ツールキットの初期化，UIL ファイルのオープン，コールバック関数の登録，メインウィンドウの表示を行うためのファイルである．このファイルはC言語で記述される．
3. コールバックファイル
C言語によるコールバック関数の集合を記述したファイルである．ここでは，ダイアログボックスの表示の制御やパラメータの値の獲得と検査，アプリケーションルーチンの呼び出し，コマンドの選択可能・不可能の切替え，および情報表示領域のバッキングストア（ウィンドウが他のウィンドウに覆われたときに，後で再表示できるようにウィンドウの内容を保存しておくこと）処理を行う．
4. リソースファイル
使用するウィジェットのフォントやラベル，色などのリソースの外部設定ファイルである．このファイルにより，再コンパイルしなくてもフォントやラベルを変更することができる．
5. make ファイル
ツールが出力する各種ソースファイルをコンパイルおよびリンクするための make ファイルである．

一つのアプリケーションは，支援ツールが出力する UIL ファイル，メインファイル，コールバックファイル，およびリソースファイルにアプリケーション用のファイルを加えた5種類のファイルからなる．アプリケーション用ファイルは意味仕様記述で定義したパラメータを持つアプリケーションルーチンの集合である．メインファイル，コールバックファイル，およびアプリケーション用ファイルはコンパイル・リンクされ，実行形式になる．UIL ファイルは UIL コンパイラによりコンパイルされ UID ファイルになる．UID

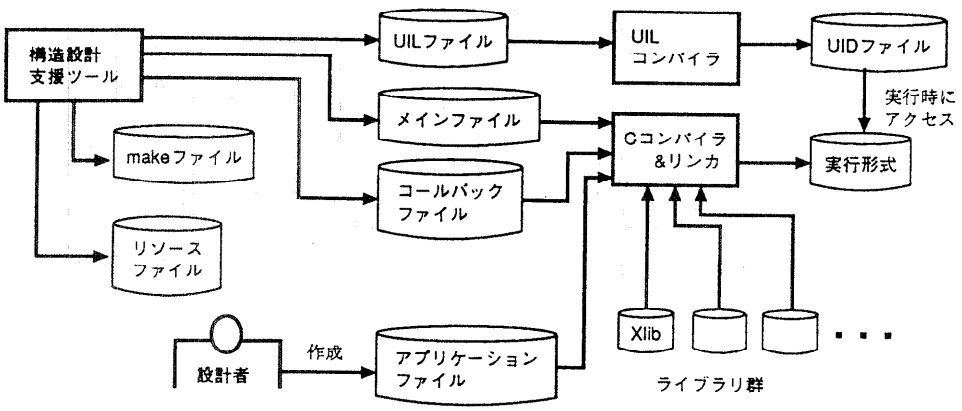


図 8 アプリケーションの生成過程
Fig. 8 Generating process of an application.

ファイルはユーザインタフェースの階層などを定義したファイルで，このファイルは実行時にロードされ使用される（**図8**）．また，アプリケーション用のファイルが完成していない場合にも，ユーザインタフェースのみの実行が可能である．

7. おわりに

本論文では，OSF/Motif 上のユーザインタフェースを意味設計と構造設計の2段階に分けて設計する意味・構造設計モデルの提案，およびそのモデルに基づくユーザインタフェース設計システムでの支援ツールについて述べた．また，Foley らが用いているユーザインタフェース定義言語を OSF/Motif 用に拡張・修正して言語 IDL+ を定義実現した．

本研究で開発した2種類のツールを用いたユーザインタフェースの設計では次のような利点がある．

1. 複数の同じ機能を持ちパラダイムの異なるユーザインタフェースを素早く生成できる．
2. 異なるパラダイムに基づくユーザインタフェースの意味仕様記述が半機械的に生成できる．
3. ユーザインタフェースの完全性や一貫性の評価が設計段階においてある程度可能になる．
4. ユーザインタフェースの設計誤りを抑えることができる．

今後の課題としては以下に示す項目がある．

1. 意味設計の段階における変換は現在3種類の因子化であるので，それ以外の変換技法を考案し，支援ツールに追加，拡張すること．
2. 構造設計支援ツールを直接操作型のユーザインタフェースに対応させること．そのためには，直接操作型の典型的なパターンを調査し，コマンドのパラメータ入力の技法として提供する．
3. ユーザインタフェースの質的評価のための基礎的研究とそのシステム化．

参考文献

1) Computer System Research Group: UNIX Programmer's Supplementary Documents, Vol. 1, PS1: 15, PS1: 16, Dept. of EE&CS, University of California, (1986).

2) Foley, J.: Models and Tools for the Designers of User-Computer Interface, Report GWU-IIST-87-03, Dept. of EE&CS, George Washington University, (1987).

3) Foley, J., Van Dam, A.: *Computer Graphics 2nd ed.*, Addison-Wesley (1990).

4) Gibbs, C., Kim, W.C. and Foley, J.: Case Studies in the Use of IDL: Interface Definition Language, Report GWU-IIST-86-30, Dept. of EE&CS, George Washington University (1986).

5) 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，第13編，第10章，オーム社，pp. 1194-1201 (1989).

6) Jones, O.: *Introduction to the X Window System*, Prentice-Hall (1989).

7) Kernighan, W.B. and Ritchie, M.D.: *The C Programming Language*, Prentice-Hall (1978).

8) Löwgren, J.: History, State and Future of User Interface. Management Systems, *ACM SIGCHI Bulletin*, Vol. 20, No. 1, pp. 32-44 (1988).

9) MIPS Computer Systems: RISCwindows Motif Reference, MIPS Computer Systems, Vol. I, II, III (1989).

10) 三宅一巧：仕様記述の変換に基づく対話型ユーザインタフェースの設計，山梨大学工学部修士論文 (1992).

11) 三宅一巧，渡辺喜道，今宮淳美：ユーザインタフェース設計における概念記述の変換，第41回情報処理学会全国大会論文集，4R-3 (1990).

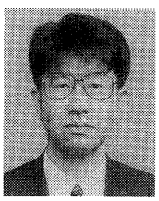
12) 三宅一巧，渡辺喜道，今宮淳美：仕様記述の変換に基づく対話型ユーザインタフェースの設計，情報処理学会ヒューマンインタフェース研究会，40-6 (1992).

13) Nye, A. and O'Reily, T.: *X Toolkit Intrinsics Programming Manual*, O'Reilly & Associates (1990).

14) O'Reilly, T. and Langlay, M.: *X Toolkit Intrinsics Reference Manual*, O'Reilly & Associates (1990).

15) 渡辺喜道，三宅一巧，今宮淳美：ユーザインタフェース設計における概念仕様記述の変換—因子化—，山梨大学工学部研究報告，pp. 31-37 (1991).

（平成4年9月22日受付）
（平成5年5月12日採録）

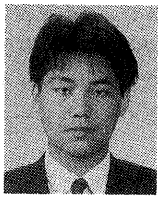


渡辺 喜道（正会員）

1964年生．1986年山梨大学工学部計算機科学科卒業．1988年同大学院修士課程修了．同年同計算機科学科助手，1989年同電子情報工学科助手，1993年東京工業大学工学部情報工学科助手，現在に至る．ソフトウェア開発環境，対話システム等に興味を持つ．ACM, IEEE, 電子情報通信学会，ソフトウェア科学会，人工知能学会会員．

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年山梨大学工学部計算機科学科助教授．現在，山梨大学工学部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．



三宅　一巧

1966 年生．1990 年山梨大学工学部計算機科学科卒業．1992年同大学院修士課程修了．同年日本電気(株)入社，現在に至る．対話型システム，コンピュータグラフィクス等に興味を持つ．