ヒューマンインタフェース 40－6
（1992. 1. 16）

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

三宅一巧　渡辺喜道　今宮淳美
山梨大学　電子情報工学科

本稿では，OSF/Motif上のユーザインタフェース設計のためのモデルを提案し，このモデルに基づくユーザインタフェース設計システムについて述べる．このシステムでは，設計者はユーザインタフェースのプロトタイプ作成に変換ツールおよび構造設計支援ツールを用いる．変換ツールは，インタフェース定義言語IDL+で記述されたユーザインタフェースの抽象記述を機能的に等価な別の記述に変換する．構造設計支援ツールは，変換ツールが出力する記述を解析して実行可能なコードを生成する．これらのツールを用いることにより，設計者は簡単に素早くプロトタイプを作成できる．

THE INTERACTIVE USER INTERFACE DESIGN BASED ON TRANSFORMATIONS OF FORMAL SPECIFICATION

Kazuyoshi Miyake　Yoshimichi Watanabe　Atsumi Imamiya
Department of Electrical Engineering and Computer Science, Yamanashi University

This paper presents a model for the user interface design on the OSF/Motif, and the user interface design system based on the model. This system allows the designer to create prototypes of the user interface with two tools: One is a "transformation tool" which transforms the abstract specification into a series of functionally equivalent specifications, but each of which is a slightly different. Another is a "structure design support tool" which generates executable code by analysing the specification of the transformation tool. These tools allow the designer to create the prototype quickly and easily.

1 はじめに

最近のユーザインタフェースではメニューやアイコンを使用し，理解しやすく対話性のよい多くのGUI（グラフィカルユーザインタフェース）が開発され，ユーザは直観的な操作で対話ができるようになりつつある．しかしその反面，ソフトウェア全体に占めるユーザインタフェース部分の割合が増大し，ユーザインタフェース設計者の負担が非常に大きくなっている．

この問題を解決するために，オブジェクト指向の概念をとり入れユーザインタフェースの構成要素（メニューやボタンなど）を「部品」として提供するツールキットや，ユーザインタフェース部分をアプリケーションから分離し，宣言的言語やスクリーンフォーマッタなどを用いてユーザインタフェースを設計可能とするユーザインタフェース管理システムについて多くの研究がなされている [3, 5]．

しかしユーザインタフェースを「対話意味」（操作対象や機能など）と「対話構造」（スクリーンレイアウトや対話の流れなど）の2要素に分けた場合，「対話構造」と比較して「対話意味」の設計支援に関しての研究は少ない．

本稿では，Xウィンドウシステム [4] 上のOSF/Motif GUI環境 [10] において，「対話意味」と「対話構造」の設計を統合するユーザインタフェース設計支援システムについて述べる．以下では，まず2節で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，本研究で提案する意味・構造設計モデルについて述べ，このモデルに基づくユーザインタフェース設計システムを提案する．3節では「対話意味」の設計の仕様を記述する言語を定義し，4節でこの言語によって記述された仕様記述を機能的に等価な別の仕様記述に変換する技法を述べる．5，6節では，その変換技法を用いて「対話意味」の設計を支援するツールと，仕様記述の解析により「対話構造」の設計を支援するツールについて述べる．

2 ユーザインタフェース設計モデル

2.1 4レベルユーザインタフェース設計モデル

ここでは，Foleyが提案した4レベルユーザインタフェース設計モデルについて述べる [1]．このモデルではユーザインタフェース設計を概念・機能・順序化・結合の4つのレベルに分けておこなう．以下に各レベルでの設計内容を示す．

1. 概念設計

   概念設計では，エンドユーザが理解しなければならない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．典型的な概念レベルの設計では，オブジェクトやオブジェクトのクラス，オブジェクト間の関係，オブジェクトに対するコマンドを定義する．

2. 機能設計

   機能設計では，各コマンドに対して詳細な機能の設計と意味付けをおこなう．すなわち，各コマンドを実行するために必要な情報，おこりう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に注意されたい．

3. 順序化設計

   順序化設計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ている．したがって，出力列には2Dおよび3D表示機構やアニメーションなどの時間的に変化する表示形式などが含まれる．

4. 結合設計

   結合設計では，順序化設計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である．

2.2 意味・構造設計モデル

OSF/Motifでは従来のXツールキット [8, 9] と同様，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）の組み合わせとして定義する [10]．したがって，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で定義すべき情報は非常に少なくなる．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クラスの動作としてあらかじめ規定されているため定義する必要がない．エラーに関してもイン

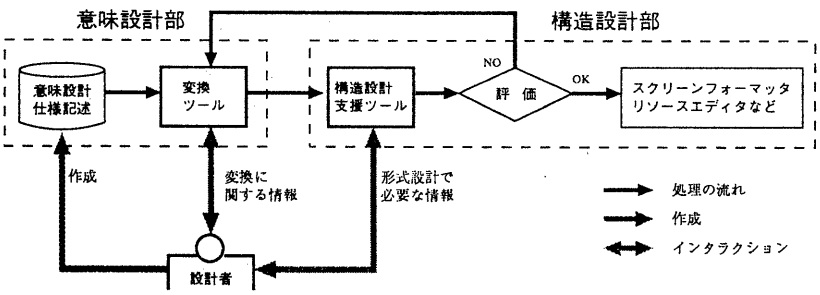

図 1: ユーザインタフェース設計システム

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回避できる．

また，OSF/Motif ではユーザインタフェース設計のガイドラインを示すスタイルガイドを提供している [10]．本研究では，このスタイルガイドを参考にして，ユーザ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限を加える．これにより順序化設計と結合設計は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」という制限を適用すると，順序化設計が結合設計に依存することになる．

したがって本稿では 4 レベルユーザインタフェース設計モデルにおける概念設計と機能設計，順序化設計と結合設計をそれぞれ組み合わせ，前者を意味設計，後者を構造設計と呼ぶことにする．

一般的にユーザインタフェースの開発はプロトタイプの作成とその評価の繰り返しによっておこなう必要があるので，本研究では図 1に示す 2 つのツールを統合しての設計システムを提案する．

設計者は，まず意味設計をおこなう．意味設計では，はじめにユーザインタフェース設計者が意味設計の仕様記述を作成する．そして機能的に等価な別の仕様記述に変換する機能をもつ変換ツールにこの記述を入力して，必要に応じて対話的に様々な変換をする．

次に，変換ツールの出力を構造設計支援ツールに入力して構造設計をおこなう．このツールは意味設計の仕様記述を解析し，ユーザインタフェースの実行可能なコードを生成する．このとき設計者の判断が必要ならば，このツールは設計者に質問を提示して設計者がそれに答えるという形で情報を補う．

最後に，構造設計支援ツールが生成したコードを実行して，ユーザインタフェースの評価をおこなう．その結果，仕様を変更する場合には変換ツールによって仕様記述を変換し，それ以降の手順を繰り返す．また，デフォルトのフォントやラベルを変更する場合には，既存のスクリーンフォーマッタやリソースエディタを用いてこれらを変更する．

3 意味設計仕様記述用言語

この節では，意味設計のための仕様記述言語 IDL+ について説明する．この言語は Gibbs らの提案したインタフェース定義言語 IDL[2] に基づいている．IDL+ による意味設計の仕様記述は，次の 4 つの部分からなる．

1. object section
   - 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. attribute section
   - 各オブジェクトの属性値の範囲の指定
3. initialization section
   - 変数の初期値の設定
4. command section
   - コマンドのグルーピング
   - コマンド名とそのパラメータの定義
   - 前条件（コマンド実行前の条件）の指定
   - 後条件（コマンド実行後の条件）の指定

例として，様々な色をもつ三角形と四角形の生成および削除，回転のできるアプリケーションを考える．この場合の意味設計の仕様記述を図 2に示す．

object section では，shape, triangle, square の 3 つのオブジェクトを定義している．オブジェクト shape は

```
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

2つのサブクラス triangle と square をもつ．さらに，形状の生成，削除，回転の3つのコマンドをもち，属性として位置，色，角度をもつ．また，オブジェクト triangle と square はスーパクラスとして shape をもつ．これらのオブジェクトはサブクラスをもたないので，スーパクラス shape から継承する生成，削除，回転の3つのコマンド，および位置，色，角度の3つの属性をサブクラスに継承できない．

attribute section では，3つの属性 position, color, angle を宣言している．位置を表す属性 position の値は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 は，赤，緑，青，白，黒，シアン，マゼンタ，黄の内の1色を指定できる．角度を表す属性 angle は 0 ～ 360 の範囲の値を指定できる．

initialization section では，初期状態の形状数が 0 であることを示している．

command section では，形状の生成，回転，削除の3つのコマンドを定義している．たとえば，形状を削除するコマンド delete は，形状を意味する1つのパラメータをもち，前条件により，このコマンド実行前では形状数が 0 ではなく，実行後は形状数が1つ減ることを示している．

4 意味設計の仕様記述の変換

同じ機能をも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 initialization section と command section は，図 3に示す仕様でも書ける．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．

```
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

以下では1つの仕様記述から機能的に等価な複数の仕様記述を生成するための変換技法の1つである因子化について述べる．

あるコマンドを実行するときに必要な1つ以上の情報の単位（パラメータやコマンド自身）を因子としてとりだし，その値を設定できるようなコマンドを追加することにより，一度その値が設定されると，以後値の変更のためのコマンドが施されるまでその値が使用されるような変換を因子化と呼ぶ [1]．たとえば，パラメータが因子化されると，コマンド実行時にそのパラメータを指定する必要がなくなる．またこのとき，因子化されてとりだされたパラメータを陰的と呼び，そうでないパラメータを陽的と呼ぶことにする．陰的なパラメータは予約語 implicit で後置修飾して表す．

因子化する情報単位の種類によって次の3つの因子化がある．

(1) オブジェクト属性の因子化

因子化されたパラメータが属性であるとき，この因子化をオブジェクト属性の因子化と呼ぶ．図3は図2の仕様記述中の属性 color を因子化した例である．この因子

化によって変更される部分は，initialization section と command section のみなので，他の部分は省略してある．色が設定されているかどうかを表す変数 color_set が initialization section に，色を設定するためのコマンド set_color が command section に，それぞれ追加されている．また，create コマンドに前条件が増えてパラメータ color が陰的になることに注意されたい．

一般に，属性を因子化するためには，次の 1 ～ 3 の操作が必要である．ここでは因子化する属性を fa とする．

1. 次の変数の初期値の設定を加える．
   fa_set = false
2. 次のコマンドを追加する．ただし att は任意の変数名である．
   set_fa(att:fa)
   postcondition(fa_set = true)
3. 2.で加えた以外の属性 fa を含むすべてのコマンドに対して，
   (a) fa を陰的にする．
   (b) 次の前条件を加える．
   precondition(fa_set == true)

(2) オブジェクトの因子化

因子化されたパラメータがオブジェクトであるとき，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になる．この因子化をオブジェクトの因子化と呼ぶ．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，様々な操作が連続的に実行できる．

変換の手順は属性の因子化とほぼ同様である．

(3) コマンドの因子化

オブジェクトの因子化と同様に，コマンドも因子化することができる．コマンドが因子化された場合，コマンドモードの概念に基づくユーザインタフェースになる．このような因子化をコマンドの因子化と呼ぶ．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連続的に適用できる．

変換の手順は前述の 2 つと比較して多少複雑になる．

5 変換ツール

前節で述べたようにユーザインタフェースの意味設計の仕様記述は同じ機能をもつアプリケーションに対して複数存在する．ユーザインタフェースの仕様を変更するたびに仕様記述を書き換えるのは設計者にとって非常に手間のかかる作業である．この節では，意味設計の仕様記述を対話的に変換するツール [6, 7] について述べる．

このツールは，ユーザインタフェース作成に X ツールキット，構文解析部作成に lex と yacc，その他の部分の作成に C 言語を用いている．このツールの主な機能を次に挙げる．

- 仕様記述の構文エラーの検出
  仕様記述の文法的な誤りを検出して，エラーメッセージを表示する．
- 仕様記述の静的意味エラーの検出
  command section 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や，多重定義されている場合などにエラーメッセージを表示する．
- 後条件の正当性の検査
  あるコマンドの後条件の変数が他のコマンドの前条件で使用されていなかった場合に警告する．この機能により，仕様記述の意味的誤りを少なくすることができる．
- 3 種類の因子化の支援
  属性やオブジェクト，コマンドの各因子化を複数同時に，また繰り返し連続的に実行できる．
- 変換時の動的意味検査
  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．
- 仕様記述ファイルに対する操作
  仕様記述の保存や読み込み，編集などの基本的なファイル操作をおこなえる．
- ツールの対話的・視覚的な実行
  マウスによるメニュー選択やパラメータの指定を用いた，ユーザフレンドリな処理ができる．

変換ツールの画面例を図 4に示す．

6 構造設計支援ツール

6.1 生成可能なユーザインタフェース

OSF/Motif 上のツールキットを用いた任意のユーザインタフェースを生成可能とするツールを作成するのは容易ではない．また，ユーザインタフェースの一貫性の問題からも有効とは言えない．すなわち，ツールキットでは作成できるユーザインタフェースの自由度が大きいため，作成されたユーザインタフェース間の一貫性が保

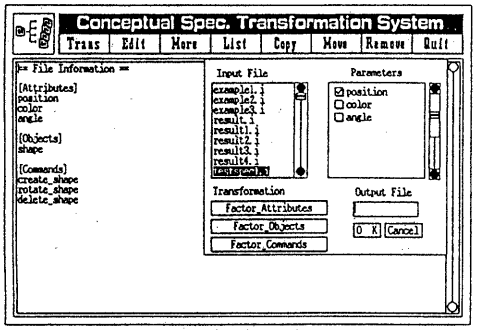

図 4: 変換ツール

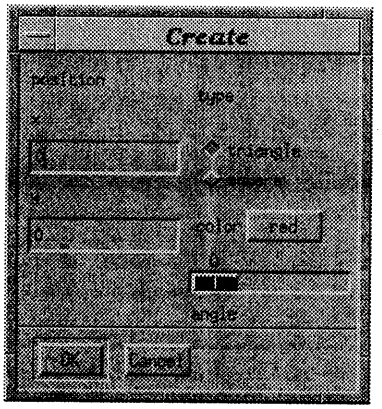

図 6: ダイアログボックス

たれない危険性がある．そこで，本研究では生成されるユーザインタフェースにいくつかの制限を加える．

アプリケーション起動時には図 5に示すメインウィンドウが表示される．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できる．

メニューバーの各項目をクリックするとプルダウンメニューが表示される．すべてのコマンドはプルダウンメニューから選択する．プルダウンメニューの各項目をクリック（コマンドを選択）すると，図 6に示すダイアログボックスがポップアップする．ダイアログボックスはボタンやテキスト入力領域など，パラメータ入力のための複数のウィジェットからなる．また，ダイアログボックス内にはパラメータ入力用ウィジェットの他に Ok, Cancel の 2つのボタンがある．Ok ボタンをクリックするとコマンドが実行され，Cancel ボタンをクリックすると各パラメータ値がデフォルト値に設定され，コマンドは実行されない．

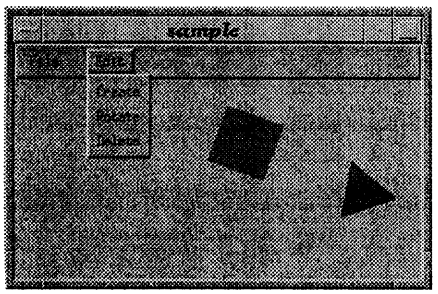

図 5: メインウィンドウ

6.2 構造設計の手順

構造設計支援ツールを用いた構造設計の手順を以下に示す．（○はツール，◎は設計者がおこなう作業）

1. メインウィンドウの定義
   (a) アプリケーション名の入力（◎）
   (b) 情報表示領域の大きさの指定（◎）
   (c) スクロールバーの有無の指定（◎）
2. メニューの定義
   (a) コマンド階層の解析（○）
   メニュー階層およびメニューバー，プルダウンメニューの項目数，ラベルなどを決定する．
3. ダイアログボックスの定義
   (a) コマンドのパラメータの解析（○）
   各コマンドのパラメータを抽出する．
   (b) パラメータの型の解析（○）
   上で得られた各パラメータの型を調べ，得られた型に基づいて使用可能なウィジェット群を決定する．各パラメータ型に対応するウィジェット群を表 1に示す．
   (c) 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ウィジェットをツールが示すウィジェット群から選択する．
4. レイアウト計算
   (a) メインウィンドウのレイアウト（○）
   (1) で指定した情報表示領域の大きさやスク

表 1: パラメータ型とウィジェット群の対応

パラメータ型	ウィジェット群
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

ロールバーの有無からメインウィンドウの大きさや内部のレイアウトを決定する.

(b) ダイアログボックス内のレイアウト (○)
ダイアログボックス内のボタン, スケール, テキスト入力領域など複数のウィジェットのレイアウトを決定する. この機能はいくつかの制約に基づいて子供の位置や大きさを管理する機能をもつ RowColumn ウィジェットと Form ウィジェットを用いて実現する.

6.3 ツールの出力

このツールは次の 3 つのファイルを出力する.

1. UIL ファイル
使用ウィジェットの親子関係, リソース, コールバック等の定義ファイル. OSF/Motif のユーザインタフェース階層定義用の高水準言語 UIL で記述される. このファイルはコンパイルされ, 実行時にアクセス可能な UID ファイルとなる.

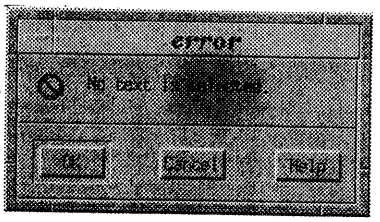

図 7: エラーメッセージの表示

2. メインファイル
ツールキットの初期化, UID 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おこなう. 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で, 以下の処理をおこなう.

- ダイアログボックスの表示の制御
ユーザの入力に従ったダイアログボックスのポップアップ&ダウンを制御する.
- パラメータ値の取得
ダイアログボックス内の各ウィジェットからコマンド実行に必要なパラメータ値を得る.
- パラメータ値の検査
入力されたパラメータ値が, 仕様記述の attribute section で定義された範囲内かどうかを検査する. 範囲外ならばエラーメッセージを表示する.
- アプリケーションからのメッセージの表示
アプリケーションからのエラーメッセージなどを図 7の形式で表示する.
- コマンド選択可能・不可能の切り替え
仕様記述内の前条件, 後条件に基づいて, コマンド選択可能, 不可能の切り替えをおこなう.
- 情報表示領域のバッキングストア
情報表示領域の一部または全部が他のウィンドウによって隠された場合のために, ウィンドウの内容を保存しておく.

1 つのアプリケーションは, 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にアプリケーションファイルを加えた 4 つ以上のファイルからなる. アプリケーションファイルは IDL ファイルで定義したパラメータをもつアプリケーションルーチンの集合で, アプリケーション設計者が C 言語で記述する. メインファイル, コールバックファイル, アプリケーションファイルはコンパイル後リンクされ実行形式となる. UIL ファイルはコンパイルされ UID ファイルとなる. UID ファイルに定義されたユーザインタフェース階層は実行時にロードされ使用される (図 8) . また, アプリケーションファイルが完成していない場合はユーザインタフェースのみの実行が可能である.

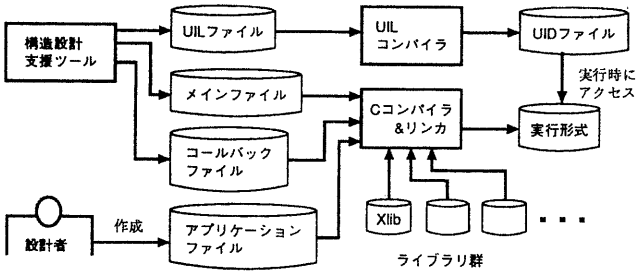

図 8: アプリケーションの生成

7 おわりに

本稿ではユーザインタフェース設計を意味設計と構造設計の 2 段階に分け，意味設計の仕様を記述するための言語を定義した．そしてこの言語による仕様記述の変換に基づくユーザインタフェース設計システムを新たに提案し，その設計を支援するための 2 つのツールについて述べた．

設計者はこれらのツールを用いることにより，簡単に素早くユーザインタフェースのプロトタイプを作成できる．また，意味設計の仕様を形式的に記述することにより，意味設計の段階での評価も考えられる．

現在までに 2 つのツールのうち変換ツールはほぼ完成しており，新しい変換技法の追加により拡張する予定である．また，構造設計支援ツールは現在開発中である．さらに，意味設計の仕様記述作成を支援する IDL+ 用構造エディタの開発も同時におこなっている．

参考文献

[1] Foley, J.:Models and Tools for the Designers of User-Computer Interfaces, Report GWU-IIST-87-03, Dept. of EE&CS,George Washington University, 1987.

[2] 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.

[3] 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.

[4] Johnes, O.:Introduction to the X Window System, Prentice-Hall, 1989.

[5] Löwgren, J.:History, State and Future of User Interface Management Systems, ACM SIGCHI Bulletin, Vol. 20, No. 1, pp.32-44, 1988.

[6] 三宅一巧：ユーザインタフェース設計における概念記述の変換，山梨大学工学部計算機科学科卒業論文，1990.

[7] 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，情報処理学会，第 41 回全国大会，4R-3，1990.

[8] Nye, A. and O'Reilly, T.:X Toolkit Intrinsics Programming Manual, O'Reilly & Assosiates, 1990.

[9] O'Reilly, T. and Langlay, M.:X Toolkit Intrinsics Reference Manual, O'Reilly & Assosiates, 1990.

[10] RISCwindows Motif Reference, MIPS Computer Systems, Vol. I , II, III, 1989.