ソフトウェア工学 88－2
(1992. 11. 10)

GUIを対象にしたリエンジニアリング方式

小泉 昌紀, 中島 震, 大竹 和雄

NEC C&Cシステム研究所
〒216 川崎市宮前区宮崎 4-1-1

あらまし 近年，既存プログラムから設計情報を復元するリエンジニアリング技術が注目されている．リエンジニアリング技術へのニーズが高い応用領域として，頻繁に改造要求が生ずるグラフィックユーザインタフェース(GUI)がある．本稿では，GUIライブラリ関数に関するプログラムスライスを対話的に抽出／評価することにより，画面レイアウト情報を再構築する方式を提案する．この方式を，GUIライブラリ「鼎」を用いて記述したCプログラムから，対話的GUI構築ツール「ゆず」の内部形式へ変換する問題に適用し，GUIリエンジニアリングシステム**Renée**の試作を行なった．

和文キーワード リエンジニアリング，リバースエンジニアリング，GUI，データフロー解析，プログラムスライス

Reengineering for GUI Programs

KOIZUMI, Masaki NAKAJIMA, Shin OTAKE, Kazuo

C&C Systems Research Laboratories, NEC Corporation
1-1, MIYAZAKI 4-CHOME, MIYAMAE-KU, KAWASAKI, KANAGAWA 216 JAPAN

Abstract Recently, demands for re-engineering, which recovers design information from existing programs, are rapidly growing. Especially, GUI(Graphical User Interface) is one of the most demanding domain for re-engineering since GUI programs are frequently changed. We present a re-engineering method that recovers layout information of GUI objects from GUI programs. Our method consists of extracting and evaluating *program slice* on GUI library functions. By using this method, we implemented GUI re-engineering system called **Renée**, which translates programs in C language with "CANAE" GUI libraries into a data format of GUI builder "YUZU".

英文 Key words Re-engineering, Reverse Engineering, GUI, Dataflow Analysis, Program Slicing.

1 はじめに

近年，ソフトウェアの保守改造を支援する機能を持つCASEツールへの要求が急増している．特に，既存プログラムから設計情報を復元するリエンジニアリング技術が注目され，手続き呼出関係などの構造情報を抽出するリバースシステム [2, 3] が開発されてきた．しかし，設計情報は対象とするソフトウェア分野に大きく依存するため，構造情報を抽出するリバースシステムだけでは不十分である．さらに上位の設計情報を抽出するためには，応用領域を決める必要がある．

リエンジニアリング技術へのニーズが高い応用領域として，頻繁に改造要求が生ずるグラフィックユーザインタフェース (GUI) がある．GUIプログラムにおける設計情報は，ボタン・メニュなどのGUIオブジェクトに関する画面レイアウトが主である．従って，GUIプログラムから，画面レイアウトを抽出できれば，GUIプログラム保守改造の工数が大幅に削減できる．そのため，画面レイアウトを復元する方式を考案する必要があった [5]．

本稿では，GUIプログラムから画面レイアウトを対話的に復元するリエンジニアリング方式を提案する．本方式は，GUIライブラリ関数に関するプログラムスライス [9] を対話的に抽出／評価することにより，画面レイアウト情報を再構築するものである．

この方式を，GUIライブラリ「鼎 [7]」を用いて記述したCプログラムから，画面レイアウトを直接操作できる対話的GUI構築ツール「ゆず [8]」の内部形式へ変換する問題に適用し，GUIリエンジニアリングシステム **Renée** の試作を行なった．また，**Renée** を，鼎サンプルプログラムに適用することにより，本方式の妥当性を示した．

本稿の構成は，以下のようである．まず，2章で，GUIプログラムのリエンジニアリングについて，例題を用いて問題点を明らかにする．3章で，GUIプログラムのリエンジニアリング方式の概要とその実現方式について報告する．4章で，**Renée** の適用例と応用分野について検討する．

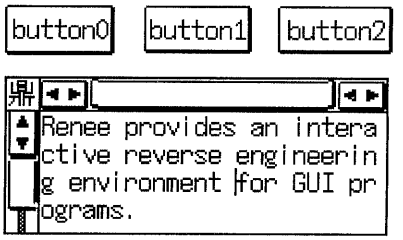

図 1: 画面レイアウト情報例

2 GUIのリエンジニアリング

リエンジニアリングは，仕様からプログラムを作成するフォワードエンジニアリングと，プログラムから逆に仕様を抽出するリバースエンジニアリングから成る [3]．GUIプログラムにとっての仕様は画面レイアウト情報であり，本稿で対象とするリエンジニアリング問題を次のように定義する．

フォワード 画面レイアウト情報からGUIプログラムを生成すること

リバース GUIプログラムから画面レイアウト情報を抽出すること

このうち，GUIビルダの機能を用いることにより，フォワードの部分を実現できる．以下，リバースに焦点を絞って議論する．なお，GUIライブラリは「鼎 [7]」，画面レイアウト情報の表示は「ゆず [8]」を仮定する[1]．

2.1 GUIプログラムのリバースの定義

入力：GUIプログラム GUIプログラムは，ボタン，メニュなどのGUIオブジェクトを表示する画面レイアウト部分と，それ以外のアプリケーション部分から構成される．画面レイアウト部分では，画面上に生成するGUIオブジェクトの種類や，GUIオブジェクト間にあるサブメニュなどの関係を記述する．「鼎」では，鼎ライブラリ関数群[2]を用いてGUIオブジェクト生成などを表現する．

図1に，フォーム上に水平方向に連続した3つのボタンとテキストエディタを表示した画面

[1] 本稿で提案する方式は，鼎に限定するものでない．GUIライブラリと対応する画面レイアウト表示手段があるような他のGUIに対しても適用可能である．

[2] 本稿で対象にするのは，画面表示に関係する関数だけである．

```
 1  Widget Form, Editor, Button, Menu;
 2  void main()
 3  {
 4    int i, x, y;
 5    mkForm();
 6    mkEditor();
 7    mkMenu();
 8    y = 0;
 9    i = 0;
10    while (i < 3) {
11      mkButton(100 * i, y);
12      CxItemSetSub(Button, 0,
                     Menu->core.parent);
13      i++;
14    }
15  }
16  void mkButton(a, b)
17    int a, b;
18  {
19    Button = CxMakeManagedWidget(
        "button", buttonWidgetClass, form,
        XtNx, a, XtNy, b,
        XtNwidth, 80, XtNheight, 20,
        XtNbackgrand, "blue",
        NULL);
20  }
```

図 2: 鼎プログラム例

レイアウト例を示す．この画面レイアウトを出力する鼎プログラムを図2に示す．ここでは，鼎ライブラリ関数 *CxMakeManagedWidget* を呼び出して GUI オブジェクトを生成している．本関数は，引数としてオブジェクト名，クラス名，親オブジェクト名，x,y 座標，幅，高さ，背景色などのリソース[3]を指定する．

出力：画面レイアウト情報　GUI オブジェクトの属性と GUI オブジェクト間の関係を表したデータ形式を画面レイアウト情報と呼ぶ．「ゆず」では，このデータ形式は Lisp で用いられる S 式で表わされている．ゆず関数 *load-widget-part* は引数としてリソースを指定する．図2に対応する画面レイアウト記述を図3に示す．画面レイアウト記述は，以下の条件をみたす必要がある；

条件 1 制御構造を含まない，

条件 2 ゆず関数の引数は定数．

GUI のリバース　GUI のリバースとは，鼎ライブラリ関数の引数を解析することにより，条件 1,2 をみたす同等なゆず関数に変換することである．

[3]X-Toolkit のリソース．

```
1  (load-widget-part "button" '(0 0 100 20)
2    'backgrand "blue")
3  (load-widget-part "button" '(100 0 100 20)
4    'backgrand "blue")
5  (load-widget-part "button" '(200 0 100 20)
6    'backgrand "blue")
7  ...
```

図 3: 画面レイアウト情報例（部分）

2.2 リバースにおける問題点

*main*関数中から直接，鼎ライブラリ関数が呼ばれ，その引数がすべて定数であるようなプログラムを対象にした場合は，単純に鼎ライブラリ関数をゆず関数に変換するだけでリバースが実現できる．しかし，制御構造，変数などを含む実際のプログラムを対象にした場合，GUI ライブラリ関数の呼び出し，引数などの情報は，プログラムを詳細に解析しなければ決めることができない．

従って，リバースを実現するためには，GUI ライブラリ関数に関して以下の静的解析をすることにより，条件 1,2 をみたすような情報を抽出することが必要となる．これを図2のプログラムを例に説明する．

(1) ライブラリ関数の呼び出しの解析

(1a) 手続き呼び出しの解析
*main*関数から，手続き呼び出し関係をたどり，実際に呼ばれるライブラリ関数を解析する必要がある．例では，*main*中から *mkButton*を経てライブラリ関数 *CxMakeManagedWidget*を 19 行目で呼び出すことを解析する．

(1b) 条件／繰り返し文の解析
ライブラリ関数の呼び出し回数を決定する必要がある．例では，ライブラリ関数を呼び出す関数が while 文の中の 11 行目で呼ばれているので，ライブラリ関数が 3 回呼び出されることを解析する．

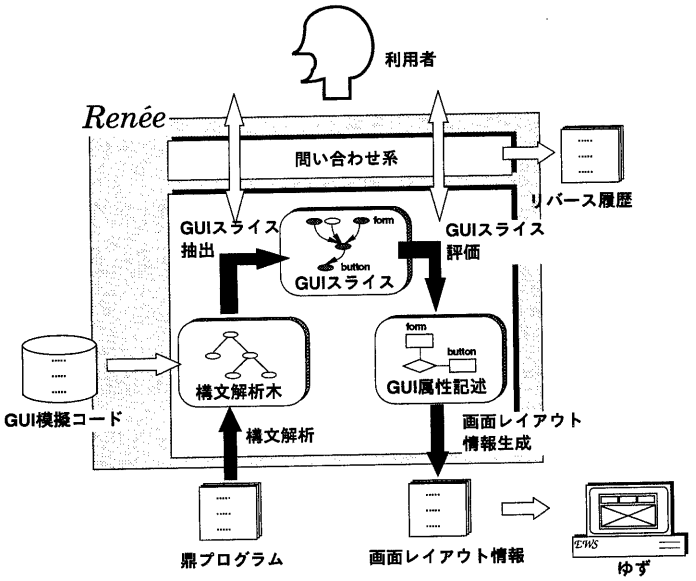

図 4: Renée の枠組み

(2) ライブラリ関数の引数の解析

(2a) 中間変数の解析
プログラム中で設定される中間変数の解析を行なうデータフロー解析機構が必要である．例では，19 行目で参照される変数 b の値は，8 行目で定義されているということを解析する．

(2b) ライブラリ関数が設定する変数の解析
プログラム中に明示されない関数の内部動作の解析機構が必要となる．12 行目で参照される `Menu->core.parent` の値は，ライブラリ関数 *CxMakeManagedWidget* の内で設定されることを解析する．

3 Renée のリエンジニアリング方式

3.1 概要

提案するプログラムリバース方式 Renée の枠組みを図 4に示す．Renée は以下の 4 段階を経ることにより，GUI プログラムから画面レイアウト情報を復元する．Renée がリバース途中で決定した情報はリバース履歴として保存し，説明機能として用いる．

(1) 構文解析 鼎ライブラリを用いて記述された C プログラムと予め用意された GUI 模擬コードを入力し，構文解析を行ない，構文解析木と記号表を出力する．ここで，GUI 模擬コードとは，ライブラリ関数の内部で行なわれている大域変数の設定を模擬したコードであり，例を図 5に示す[4]．GUI 模擬コードを用いることにより，2 章の (2b) で指摘した GUI ライブラリ関数内で設定される大域変数の解析を行なうことができる．

```
Widget
CxMakeManagedWidget(a1, a2, a3, a4, a5)
  void *a1, *a2, *a3, *a4, *a5;
{
  Widget w;
  CxGen(a1, a2, a3, a4, a5);
  w->core.name = a1;
  w->core.class = a2;
  w->core.parent = a3;
  w->core.x = a4;
  w->core.y = a5;
  return w;
}
```

図 5: GUI 模擬コードの例

[4]実際には，約 60 の引数を持つが，簡単のため 5 引数としている．

```
1    int x, y;
2    int *ip, *iq;
3    x = 10;
4    ip = &x;
5    y = *ip;
6    iq = ip;
7    y = *iq;
8    *iq = 20;
9    y = x;
```

図 6: ポインタを含むプログラム

(2) GUI スライス抽出 利用者がプログラム中で GUI ライブラリ関数の呼び出しや引数に関連した情報のみに注目できるように，GUI スライスを抽出する．ここで，GUI スライスとは，GUI ライブラリ関数の呼び出しに影響を与える全ての文の集合である．静的解析を行なうことにより，プログラムから GUI スライスを対話的に抽出する．

(3) GUI スライス評価 GUI スライスに含まれる文の右辺式を順に評価し，変数値を定数化することにより，GUI ライブラリ関数の呼び出し回数と引数を確定する．確定の結果，GUI オブジェクトのリソース値，GUI オブジェクト間の位置関係などが求まる．

(4) 画面レイアウト生成 GUI オブジェクトのリソース値，GUI オブジェクト間の位置関係などから，データ変換を行なうことにより，画面レイアウト情報（ゆず構文）を自動生成する．

以下，本方式にとって中心部分となる (2) に関して説明を行なう．

3.2 GUI スライス

プログラムスライス $S(v,n)$ とは，プログラム中のある文 n における変数 v に影響を与える文をすべて含んでいて，かつ実行可能であるようなプログラムの部分集合である [9]．ここでは，プログラムスライスの特殊形として，GUI スライスを以下のように定義する[5]．

定義 1 フローグラフは $\langle N, E, s\rangle$ の 3 つ組である．N はノードの集合，E はエッジの集合で

[5]プログラムスライスに関する詳細な定義は [9] を参照されたい．ここでは，紙面の都合上，概略のみ与える．

```
 1  Widget Button;
 2  void main()
 3  {
 4    int i, x, y;
 8    y = 0;
 9    i = 0;
10    while (i < 3) {
11      mkButton(100 * i, y);
13      i++;
14    }
16  void mkButton(a, b)
17    int a, b;
18  {
19    Button = CxMakeManagedWidget(
        "button", buttonWidgetClass, form,
        XtNx, a, XtNy, b,
        XtNwidth, 80, XtNheight, 20,
        XtNbackgrand, "blue",
        NULL);
20  }
```

図 7: GUI スライス $G(19)$

ある．s は初期ノードであり，s から N 中の他の全てのノードへのパスが存在する．n' が n の後続ノードの時，$n \leq n'$ なる半順序関係をノード間に定義する．特に，n のすぐ後続のノードの集合を $\Gamma^{+}(n)$ と書く．

定義 2 プログラム P 中に現われる変数名の集合を V とする．P 中の各文 n(フローグラフのノードと対応）について，以下のような V の部分集合が定義できる．n において値が参照される変数の集合を $REF(n)$，n において値が上書きされる変数を $DEF(n)$ とする．また，変数 v に対して，$v = DEF(n')$ をみたす文 n' $(n' \leq n)$ の集合を検索する手続きを $search(v,n)$ とする[6]．

例として，図6において，$DEF(6)$ =iq, $REF(6)$ = {ip}, $search(\mathtt{ip},6)$ = {4}, $search(\mathtt{x},9) = \{3\}$ である．

GUI スライスは，GUI ライブラリ関数の呼び出しに影響を与える全ての文の集合であり，以下に定義する．

定義 3 プログラム P 中における *GUI* ライブラリ関数呼び出しを含む文の集合を $N_G = CALL_G(P)$

[6]以下で，$search(v,n)$ の v が集合の場合は，v のすべての要素 v' について $search(v',n)$ を求めた和集合を意味するとする．

入力 フローグラフ $\langle N, E, s\rangle$，変数集合 v，ノード n
出力 述語 $p(n) =$ true をみたす節 n の集合
使用するデータ構造 リスト L
方法
for $n \in N$ do $p(n) \leftarrow$ false;
L に n を登録
while L が空でない do begin
$m \leftarrow L$ のひとつの要素を取り出す;
for $n' \in search(REF(m), m)$ do begin
$p(n') \leftarrow$ true;
L に n' を加える
end
end

図 8: スライス $S(v, n)$ を求める手続き

とすると，$n \in N_G$ における GUI スライス，プログラム P の GUI スライスはそれぞれ，

$$S_G(n) = \bigcup_{v \in REF(n)} S(v, n)$$

$$S_G = \bigcup_{n \in N_G} S_G(n)$$

で与えられる．

例として，図2のプログラムにおける GUI スライス $G(19)$ を，図7に示す．

3.3 GUI スライスの抽出

一般に，プログラムスライスを求めることは，不動点を求めることに相当し [9, 1]，計算量が大きい．しかし，GUI スライスを求める本来の目的は，利用者に画面レイアウトに必要な情報を与えることであり，近似的な GUI スライスを求めることができれば良い．

そこで，まず，分岐文や繰り返し文などを除去することにより，プログラムの実行経路を確定した後，GUI スライスの抽出を行なう簡易法を採用した．分岐文の分岐条件や繰り返し文の終了条件を利用者に提示し，条件分岐のパス，繰り返し文／再帰呼出の実行回数などの値を対話的に確定することにより，プログラムの実行経路を確定する．本仮定に基づくアルゴリズムを図8に示す．

3.4 ポインタ変数の取り扱い

Renée が解析の対象とするプログラムは，C 言語で記述されており，GUI ライブラリ関数が

入力 フローグラフ $\langle N, E, s\rangle$
出力 ポインタ変数への代入が影響を及ぼす変数を明示したフローグラフ $\langle N', E', s\rangle$
使用するデータ構造 リスト L
方法
L に s を登録
while L が空でない do begin
$n \leftarrow L$ のひとつの要素を取り出す;
for 各 $m \in \Gamma^+(n)$ につき do begin
if （ $v = DEF(m)$ が * 付き変数） then
for $v' \in search_p(v, n)$ do
"$v' = n$ の右辺 "を m の直前に挿入
L に m を加える
end
end

図 9: フローグラフを変換する $trans_p$ 手続き

操作するデータ構造は，ポインタ変数が多用されている．従って，ポインタ変数を解析することが本質的となる．そこで以下では，ポインタ変数を扱えるようにプログラムを変換する手続きについて述べる．

まず，プログラム上でのポインタ変数の検索手続き $search_p$ を定義し，この手続きを用いて，ポインタ変数を取り扱えるようにプログラムを変換する手続き $trans_p$ を定義する．以後，ポインタに関する演算として，& 演算子と * 演算子と代入のみを許し，加減算などは許さないことにする．

定義 4 プログラム P 中の文 n において参照される & 演算子付きの変数の集合を $ADR(n)$ とすると，文 n におけるポインタ変数 v が指す変数の集合を検索する $search_p(v, n)$ 手続きは，

$$search_p(v, n) = \bigcup_{m \in search(v, n)} search_{p'}(m)$$

$$search_{p'}(n) = \begin{cases} \bigcup_{m \in search(REF(n), n)} search_{p'}(m) & \\ \qquad\qquad (\text{if } ADR(n) = \emptyset) & \\ ADR(n) \qquad (\text{otherwise}) & \end{cases}$$

で与えられる．

例として，図6において，$search_p(\texttt{*iq}, 7) = search_p(\texttt{*iq}, 8) = \{\texttt{x}\}$ である．

定義 5 プログラム P において，ポインタ変数への代入が影響を及ぼす範囲を明示したプログ

```
1    int x, y;
2    int *ip, *iq;
3    x = 10;
4    ip = &x;
5    y = *ip;
6    iq = ip;
7    y = *iq;
*    x = 20;
8    *iq = 20;
9    y = x;
```

図 10: 図 6 に $trans_p$ を施したプログラム

ラムは，$trans_p(P)$ で与えられる．$trans_p(P)$ を求めるアルゴリズムを図 9に示す．

例として，図 6のプログラムを $trans_p$ により変換した結果を図 10に示す．8 行目で定義されるポインタ変数 `*iq` への代入文 `*iq = 20;` に関して，$search_p(\texttt{*iq}, 8) = \{\texttt{x}\}$ なので，`x` を左辺に持つ代入文 `x = 20;` を生成し，8 行目の直前に挿入する．

もとのプログラムでは正しく解析できなかったポインタ変数が影響を及ぼす変数の定義 - 参照関係が，変換 $trans_p$ を施すことにより，正しく解析できるようになる．例として，図 6のプログラムでは，$search(\texttt{x}, 9) = \{3\}$ であり，ポインタ変数 `*iq` が影響を及ぼす変数 `x` の定義 - 参照関係が正しく解析できないのに対し，変換を施した図 10のプログラムでは，$search(\texttt{x}, 9) = \{*\}$ であり，正しい解析ができる．

4 検討

4.1 適用例

Renée は，Yacc/Lex と C 言語を用いて Sun4 上で実現した．**Renée** は，C プログラムに前処理 cpp をかけた形式を入力とし，ゆず内部形式を出力する．**Renée** の受理する文法は ANSI-C に準拠[7]しており，C 言語のフルセットを扱うことができる．

Renée の実装に関して，解析の対象とした鼎ライブラリ関数とゆず関数の対応関係を表 1に示す[8]．また，利用者が全ての値を与えるのは現実的でないので，分岐条件や繰り返し文の終了条件はデフォルト値（if 文なら true，while 文ならループの回数 2 回など）を用意した．例として，図 2で while 文の実行回数が不定の場合，**Renée** はボタンを自動的に 2 個生成し，不都合な場合は利用者がゆず上で編集する．

[7]ただし，通常の C コンパイラが扱えるいくつかの拡張部分を扱えるように文法を拡張してある．

[8]ただし，実際には一対一に対応しない場合もある．

表 1: 鼎ライブラリ関数とゆず関数の対応関係

鼎ライブラリ関数名	ゆず構文名
CxMakeManagedWidget	*load-widget-part*
CxPopupManagedWidget	*load-widget-part*
CxItemSetSub	*YzItemSetSub*
setBitmap	*YzItemSetBitmap*

表 2: 適用例

内容	行数	cpp 後の行数	GUI 関数数
ゆず生成プログラム	152	15,987	13
サンプルプログラム	668	15,459	50

Renée を，ゆずが自動生成した鼎プログラムと，鼎ライブラリに添付されているサンプルプログラム *dpdemo*[9]に適用した結果を表 2に示す．いずれの場合も，プログラムの実行結果に近い画面レイアウトを得られた．

4.2 応用分野

Renée の応用分野として，まず，ライブラリ変換が挙げられる．例えば，**Renée** と，ゆずから Motif の GUI プログラムを得るフォワードエンジニアリング [6] を組み合わせることにより，鼎 GUI から Motif GUI へ自動的に GUI ライブラリの変換をすることができる．

次に，安全な部分的改造が挙げられる．**Renée** をスライスを用いたプログラム改造方式 [4] と組み合わせれば，GUI プログラムを，改造対象部分（画面表示部分）と非改造対象部分（コールバック関数本体）に分解することで，非改造対象部分を保存しながら改造対象部分の修正を支援することが可能となる．

[9]対話部品の種々の利用方法，プログラム上の技法を示すためのサンプルで，特に何かの用途を持ったアプリケーションではない．

5 おわりに

GUIプログラム中から，GUIライブラリ関数を起点としたプログラムスライスを対話的に抽出し，抽出結果を評価することにより，画面レイアウト情報（GUIオブジェクトのリソース，GUIオブジェクト間の位置関係）を復元する方式を提案した．本方式を，GUIライブラリ「鼎」を用いて記述したプログラムから，対話的GUI構築ツール「ゆず」の内部形式へ変換する問題に適用し，試作システム**Renée**の開発を行なった．**Renée**を，鼎ライブラリに添付されているサンプルプログラムに適用することにより，本方式の妥当性を示した．

今後の課題は，**Renée**をサンプルプログラムだけでなく，実際に稼働している大規模プログラムに適用し，評価することである．

謝辞

本研究を行なう機会を与えて下さった吉村猛部長 および，「ゆず」について技術的支援をして下さったソフトウェア生産技術開発研究所の杉山高弘 主任に深謝いたします．

参考文献

[1] Aho,A.V., Sethi,R, and Ullman,J.D., *Compilers - principles, Techiniques, and Tools*, Addison-Wesley Publishers Limited (1986).

[2] Y.Chen, M.Y.Nishimoto, and C.V.Ramamoorthy, The C Information Abstraction System, *IEEE Trans. on Soft.Eng.*, Vol.16, No.3, pp.325-334 (1990).

[3] Chikofsky,E.J., Reverse Engineering and Design Recovery, *IEEE Software*, (1990).

[4] Gallagher,K.B. and Lyle,J.R., Using Program Slicing in Software Maintenance, *IEEE Trans. on Soft.Eng.*, Vol.17, No.8, pp.751-761 (1991).

[5] 小泉他, GUIプログラムを対象としたリエンジニアリング, 情報処理学会第*45*回全国大会, pp.5.227-228 (1992).

[6] 宮内他, 鼎（かなえ）インタフェースビルダ「ゆず」におけるGUIプログラムを対象としたリエンジニアリング, 情報処理学会第*45*回全国大会, pp.5.103-104 (1992).

[7] 暦本他, 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, 情報処理, Vol.31, No.5, pp.602-611 (1990).

[8] 杉山他, 「鼎（かなえ）インタフェースビルダ「ゆず」の構築, 情報処理学会第*45*回全国大会, pp.5.99-100 (1992).

[9] Weiser,M., Program Slicing, *IEEE Trans. on Soft.Eng.*, Vol. SE-10, No.4, pp.352-357 (1984).