社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－SE－151（16）
2006／3／24

GUI抽象化規則を用いたモデル生成手法

水野 佑基 † 　金子 伸幸 † 　中元 秀明 ‡† 　小川 義明 ¶† 　山本 晋一郎 § 　阿草 清滋 †

† 名古屋大学大学院情報科学研究科 ‡ 野村総合研究所
¶中電シーティーアイ § 愛知県立大学情報科学部情報システム学科

{yuuki,nkaneko}@agusa.i.is.nagoya-u.ac.jp h-nakamoto@nri.co.jp
Ogawa.Yoshiaki@cti.co.jp yamamoto@ist.aichi-pu.ac.jp agusa@is.nagoya-u.ac.jp

概要

本稿では，GUI抽象化規則を用いて実装言語とGUIツールキットに対して柔軟に抽象GUI記述を生成するする手法を提案する．ウィジットと直接操作を表現するGUIプログラミングモデルと，共通GUIツールキットを定義する．抽象GUI記述はGUIプログラミングモデルに従い，共通GUIツールキットを用いて記述される．実装言語やGUIツールキットごとに異なるGUIコードと抽象GUI記述の対応付けをGUI抽象化規則として定義する．GUI抽象化規則に基づきGUIコードから抽象GUI記述を生成するシステムを提案し，異なる実装言語とGUIツールキットで実装された同一のGUIアプリケーションを同じ抽象GUI記述へと変換できることを確認した．

キーワード:　抽象GUI記述　マイグレーション　GUI抽象化規則

A Model Translation Using GUI Abstraction Rules

Yuuki Mizuno† , Nobuyuki Kaneko† , Hideaki Nakamoto‡† , Yoshiaki Ogawa‡¶ ,
Shinichiro Yamamoto§ , Kiyoshi Agusa†

†Graduate School of Information Science , Nagoya University
‡Nomura Research Institute, Ltd. ¶Chuden CTI
§Faculty of Information Science and Technology, Aichi Prefectural University

{yuuki,nkaneko}@agusa.i.is.nagoya-u.ac.jp h-nakamoto@nri.co.jp
Ogawa.Yoshiaki@cti.co.jp yamamoto@ist.aichi-pu.ac.jp agusa@is.nagoya-u.ac.jp

abstract

We propose a rule-based GUI abstraction for various GUI implementations. Abstract GUI description and GUI abstraction rules are the result and the rule of our GUI abstraction, respectively. We define common GUI toolkit and GUI programming model, which represents widgets and direct manipulations. Abstract GUI description is written by common GUI toolkit following GUI programming model. GUI abstraction rules relate GUI codes to GUI programming model and common GUI toolkit. We propose a GUI abstraction system implementing our method and show that the same GUI abstraction is generated from GUI applications written by C Motif and Java Swing.

Keywords:　Abstract GUI Description　Migration　GUI Abstraction Rule

1 はじめに

携帯電話やPDAの普及に伴い既存のアプリケーションをこれらの環境に移行する作業が頻繁に行われている．一般に移行作業は手作業で行われており，移行作業コストの削減は近年のソフトウェア開発における課題のひとつである．

一般にアプリケーションに含まれるGUIコードの量は半数以上を占める[5]．移行対象をGUIコードに限定した場合，移行作業はGUIコードの抽象化と具象化に大別できる．複数の実装技術に対応した具象化は多くの研究が行われている[1, 3, 4]．一方で抽象化に関してはHTMLなど特定の実装技術に対する研究しか行われていない[2, 6]．GUIコードの移行コストの削減には，特定の実装技術に依存しない抽象化技術が必要である．そこで，我々は実装技術に柔軟なGUIコード移行手法の確立を目指し，複数の実装技術に対応した具象化に共通で利用できる抽象GUI記述へのGUIコードの抽象化手法を提案する．

GUIの実装には様々なGUIツールキットが用いられるため，抽象化にはGUIツールキットに対する知識が必要である．また，これらの知識を基に行われる操作は実装言語に柔軟でなければならない．本研究では，様々な実装技術で実現されたGUIの移行作業を自動化するために，実装言語とGUIツールキットに柔軟なGUI抽象化システムを提案する．抽象化システムでは，GUIツールキットに対する知識を実装言語に非依存な形式で記述した規則を利用する．実装言語に柔軟な操作を実現するためにCTDS(Case Tool Data Schema)フレームワーク[7]を用いる．抽象化システムの適用例として，計算機アプリケーションをC MotifとJava Swingで実装し，2つのGUIコードを同一の抽象GUI記述に抽象化できることを確認した．

2 抽象GUI記述

抽象GUI記述とは，GUIの構成や振る舞いを実装言語とGUIツールキットに非依存な形式で記述したものである．抽象GUI記述はGUIが持つ特徴がモデルとなり，実装に非依存なGUIツールキットを語彙として持つ．

GUIはユーザが画面上に配置された“ウィジット”を“直接操作”することで対話を実現する．ウィジットと直接操作は，実装言語とGUIツールキットに関わらずGUIが持つ特徴である．この特徴を

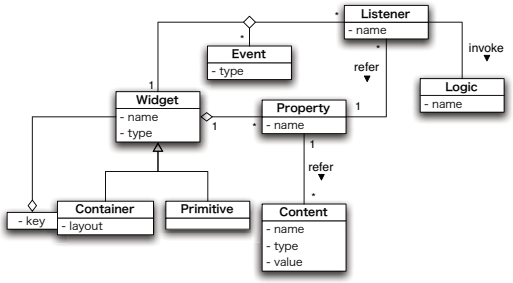

図 1: GUIプログラミングモデル

GUIプログラミングモデルとしてモデル化する．すべてのGUIの実装はGUIプログラミングモデルに基づき実現されている．

表 1: GUIプログラミングモデルの要素

要素名	意味
Widget	ユーザが操作する部品を表現する
Container	内部にWidgetを持つ部品
Primitive	内部にWidgetを持たない部品
Property	Widgetが持つ属性
Content	Propertyの値
Event	ユーザのWidgetに対する操作
Listener	ユーザがWidgetを操作したときに実行される処理
Logic	アプリケーションロジック

表 2: 共通GUIツールキットの一部

要素名	構造要素	説明
TextField	Primitive	一行の文字列の入出力を行う
Label	Primitive	文字列の出力を行う
Window	Container	ウィンドウ機能を提供する
Panel	Container	内部にウィジットを配置する
height	Property	ウィジットの高さ
width	Property	ウィジットの幅
text	Property	ウィジットの表示文字列
icon	Property	ウィジットのアイコン
onClick	Event	マウスでクリックされた
gainFocus	Event	フォーカスが当たった
loseFocus	Event	フォーカスが外れた
onSubmit	Event	完了動作が実行された

図1にGUIプログラミングモデルを示す．個々の要素の意味は表1に示す．GUIプログラミング

モデルは要素間の関連により GUI が持つ以下の特徴を表現する．

ウィジット　Widget - Property 間の関連で表現される．ウィジットは内部に見た目に関する属性を持つ．ユーザがウィジットを容易に操作できるように，属性が効果的に設定される．

ウィジットの階層構造　Container - Widget 間の関連で表現される．ウィジットは階層的に配置される．階層構造の途中に出現する要素はコンテナウィジットと呼ばれ，内部に配置されるウィジットの配置を決定する．

イベント通知機能　Widget - Event - Listener の 3 要素間の関連で表現される．個々のウィジットにユーザの操作に対する処理を対応付けすることで “直接操作” を定義する．

GUI の振る舞い　Listener - Property と Listener - Logic 間の関連で表現される．直接操作で実行される処理は，ウィジットの属性を参照・更新する処理とアプリケーション呼び出しの処理である．

GUI プログラミングモデルの語彙として共通 GUI ツールキットを定義する．共通 GUI ツールキットは一般的な GUI ツールキットが持つ機能を表現した実装に非依存な GUI ツールキットである．抽象 GUI 記述は共通 GUI ツールキットを用いて記述される．表 2 に共通 GUI ツールキットの一部を示す．

3　抽象 GUI 記述への変換

様々な実装言語と GUI ツールキットに対応した変換を行うためには，以下に示す作業が実装言語と GUI ツールキットに柔軟である必要がある．

1. GUI コードの API 呼び出しから抽象 GUI 記述に必要な要素を検出する．
2. 検出した要素を共通 GUI ツールキットに対応付けする．

図 2 は GUI コードから抽象 GUI 記述への変換プロセスを示したものである．図中の “Mapping Rule” が GUI ツールキットの要素を共通 GUI ツールキットの要素に対応付ける規則である．“Analyzing Rule” が GUI コードに出現する GUI ツールキットの API 呼び出し記述から抽象 GUI 記述の要素や関連に対応付ける規則である．これらの規則によって様々な実装言語と GUI ツールキットに対応した抽象 GUI 記述への変換を実現する．

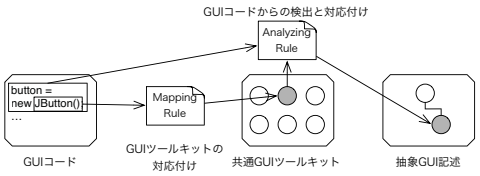

図 2: 抽象 GUI 記述への変換プロセス

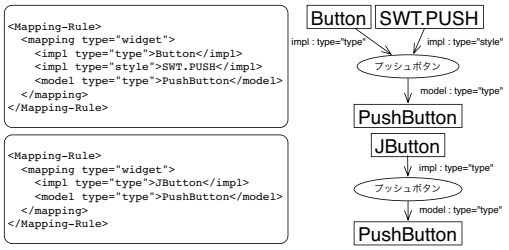

図 3: PushButton への Mapping Rule

3.1　GUI ツールキットの対応付け

GUI ツールキットの部品を共通 GUI ツールキットの要素に対応付けるために Mapping Rule を定義する．Mapping Rule の例を図 3 に示す．図 3 では，Swing の JButton と SWT の Button は同じ機能を持つ GUI 部品として共通 GUI ツールキットの PushButton ウィジットに対応付けされている．GUI ツールキットの他の部品も同様にして，同じ機能を持つ部品を共通 GUI ツールキットの同一の要素に対応付ける．

3.2　GUI ツールキットの API 呼び出しの対応付け

Analyzing Rule は GUI コードに出現する GUI ツールキットの API 呼び出しから抽象 GUI 記述に必要な要素や関連を検出し抽象 GUI 記述へと対

応付ける規則である．Analyzing Rule には 2 種類の規則が対となって定義される．

detection

detection は GUI ツールキットの API 呼び出しを分析して抽象 GUI 記述に必要な要素を検出するための規則である．GUI ツールキットの API 呼び出しは以下の 3 種類のステートメントとして GUI コードに出現する．

- 代入文
- 関数呼び出し (メソッド呼び出し含む)
- コンストラクタ呼び出し

また，GUI ツールキットの API は以下の特徴を持つ．

- GUI ツールキットのオブジェクトのコンストラクタやメソッド，または GUI ツールキットのオブジェクトを引数に取った関数として実装されている
- 上記のメソッド，関数は GUI ツールキットの部品の名称を用いて命名されている

detection はステートメントの GUI ツールキット API 呼び出しを検出するために，ステートメントの構造と変数の型，ステートメントの字句に対して比較を行う．

図 4 は detection の構造を表している．実線の関連は GUI ツールキットの API 呼び出しとして検出できるステートメントの構造を表している．破線の関連は変数，リテラルの型情報を示している．ident 要素がステートメントに出現する識別子を表しており，識別子の字句に対する比較を記述する var 要素，match 要素，affix 要素と関連している．

detection は検出したいステートメントの構造を実線の関連でつながった要素で記述する．さらに識別子の字句に対しても比較を行いたい場合に，ident 要素の下に match 要素と affix 要素を記述する．affix 要素は ident の部分文字列を指定する要素である．match 要素は識別子の字句に対して，Mapping Rule や抽象 GUI 記述の指定した値の集合と比較を行う要素である．var 要素は字句に名前を付けて後に述べる representation で参照できるようにする要素である．var 要素で参照される字句が検出された値である．match 要素は指定した

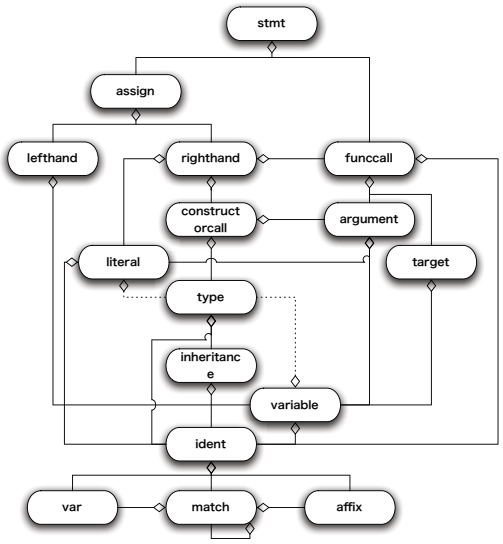

図 4: detection の構造

値に一致した場合に，その子要素を評価する．子要素に var 要素を記述することで，一致した字句を representation で利用することが可能になる．

reprentation

representation は detection で検出した値を抽象 GUI 記述の要素に対応付ける規則である．representation では対応付けたい要素を抽象 GUI 記述の構造に従って記述する．抽象 GUI 記述は全て共通 GUI ツールキットで記述されるので，detection した要素を共通 GUI ツールキットに変換する必要がある．そのために，mapto 要素と ref 要素を定義する．ref 要素は detection で定義した var 要素を参照するための要素である．mapto 要素は指定した字句を Mapping Rule に従って共通 GUI ツールキットに置き換える要素である．

4 GUI 抽象化システム

3 章で定義した規則を用いることで実装言語や GUI ツールキットの差異を吸収して抽象 GUI 記述へ変換する GUI 抽象化システムを提案する．

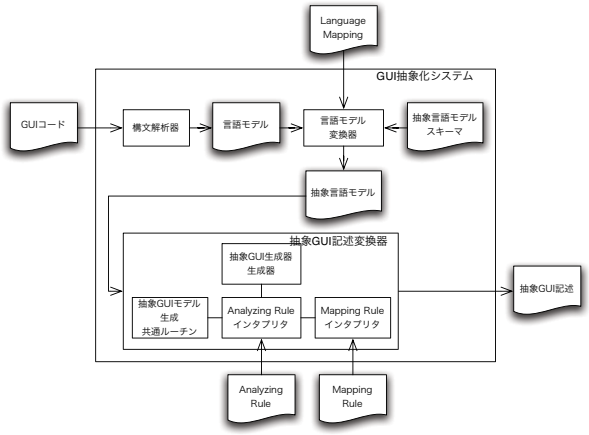

図 5: 抽象 GUI 構築システムの構成

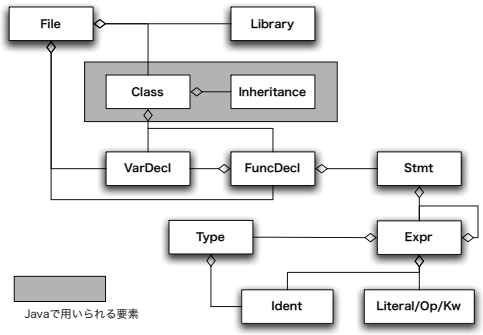

図 6: 抽象言語モデル

4.1 システム構成

GUI 抽象化システムの構成は図 5 に示す．抽象化する GUI コードと抽象言語モデルのマッピングルール (図 5 の Language Mapping)，Mapping Rule，Analyzing Rule を入力して，抽象 GUI 記述を出力する．マッピングルールと Mapping Rule，Analyzing Rule は GUI コードに用いられている実装言語と GUI ツールキットに対応したものを入力する．

4.2 抽象言語モデル変換器

抽象言語モデル変換器はソースコードの解析結果を抽象言語モデルへデータバインドする．抽象言語モデル変換器では CTDS フレームワーク [7] を利用する．CTDS フレームワークでは，開発したい CASE 技術に必要な言語モデルである CTDS を抽象ソフトウェアエレメントを用いて定義する．定義した CTDS に基づいてソースコードがデータバインドされたオブジェクト群が提供される．CTDS は実装言語に依存せずに定義されるため，開発者は提供されたオブジェクトに対する操作を設計することで実装言語に非依存な CASE 技術を開発できる．本システムでは既存アプリケーションで多く用いられている C と Java に対応した CTDS を抽象言語モデルとして定義した．

図 6 は抽象言語モデルの構成関係を示している．C と Java のソースコードに対応するために，両者の抽象構文木を統合した構造となっている．C は手続き型言語であり Java はオブジェクト指向言語である．言語間の違いが以下の構造に現れている．

- File - { FuncDecl , VarDecl } の階層構造：C の “関数宣言” と “変数宣言” に対応する
- File - Class - { FuncDecl , VarDecl } の階層構造：Java の “メンバ関数宣言，コンストラクタ宣言” と “メンバ変数，クラス変数宣言” に対応する
- Inheritance：クラスの継承関係に対応する
- Expr - Type：コンストラクタ呼び出しで呼び出す型に対応する

CTDS の各要素には定義/参照関係を表現する id/defid 属性と式の種類を表す sort 属性などを定義した．

4.3 抽象 GUI 記述変換器

抽象 GUI 記述変換器は抽象言語モデルを入力して抽象 GUI 記述に変換する．抽象 GUI 記述変換器は抽象 GUI 記述生成共通ルーチンと，抽象 GUI 記述生成器，Analyzing Rule インタプリタ，Mapping Rule インタプリタからなる．

抽象 GUI 記述生成共通ルーチン

抽象 GUI 記述生成共通ルーチンは抽象 GUI 記述生成器のメインルーチンである．処理フローを図 7 に示す．

この処理フローは，メイン関数から始まるステートメント呼び出しを関数やメソッド，コンストラ

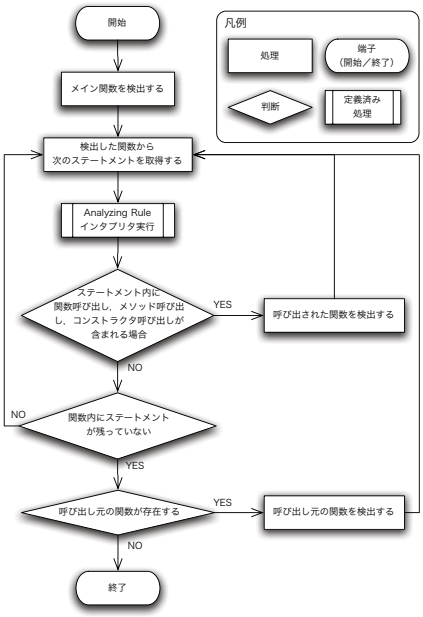

図 7: 抽象 GUI 記述生成共通ルーチンの処理フロー

クタまで遡って順番に取得する処理を表現している．ライブラリ関数は遡る対象としない．取得したステートメントは Analyzing Rule インタプリタで処理される．

Analyzing Rule インタプリタ

抽象 GUI 記述共通ルーチンから渡されたステートメントに対して，入力された Analyzing Rule を適用して抽象 GUI 記述へと変換する．ステートメントは，Expr 要素の繰り返しと末端に Ident 要素か Op 要素，Kw 要素，Literal 要素が出現する構造を取る．Expr 要素は属性 sort で種別を表現する．

detection では variable 要素の子要素の type 要素や inheritance 要素によって変数の型や継承関係を構造として表現しているので，これらの情報は抽象言語モデルの定義/参照関係である id/defid を用いて対応付けを行う．GUI コードの検出は detection の stmt 要素と入力されたステートメントの Stmt 要素との木構造を比較することで行う．また木構造に影響しない要素として，var 要素，match 要素，affix 要素がある．これらの要素と上記の type 要素が出現したときは処理を分岐する．detection のアルゴリズムを図 8 に示す．

detection で取得した要素を representation の構造から抽象 GUI 記述の要素に変換する．representation は，抽象 GUI 記述の部分木となっており，representation を抽象 GUI 記述に統合する．

representation が抽象 GUI 記述の Listener 要素となる場合，コールバック関数内のステートメントの解析を Analyzing Rule で実施する．これはコールバック関数はメイン関数から直接呼び出されないためである．

```
for detection in [ Analyze Rule に含まれる detection
セクション ] :
  for stmt in [ detection の中に含まれる stmt 要素 ] :
    dnode = stmt
    snode = ( 入力されたステートメントの Stmt )
    (ノード評価開始地点) :
    if dnode != [ detection の var , match , affix ] :
      if dnode != [ detection の type ]
      or !( dnode の親要素が variable ) :
        if ( dnode に対応する抽象言語モデルの要
素 ) == snode :
          for dnode' in [ dnode の子要素 ] :
            for snode' in [ snode の子要素 ] :
              dnode = dnode'
              snode = snode'
              goto ( ノード評価開始地点 )
      else
        snode = ( id/defid を用いて宣言位置のノードを取
得する )
        goto ( ノード評価開始地点 )
    else :
      if dnode == affix :
        snode = snode - affix
      if dnode == match
      and ( snode の字句が match で指定した名称の集合に
含まれる ) :
        for dnode' in [ dnode の子要素 ] :
          for snode' in [ snode の子要素 ] :
            dnode = dnode'
            snode = snode'
            goto ( ノード評価開始地点 )
      else :
        return ( 失敗 )
      if dnode == var :
        ( ident の値を保存する )
        return ( 成功 )
```

図 8: detection のアルゴリズム

4.4 サンプルアプリケーションへの適用

提案したシステムをサンプルアプリケーションへ適用する．サンプルアプリケーションは，(1) テキストフィールドに数字を入力し，(2) コンボボックスから演算子を選択し，(3)“=” ボタンを押すと計算結果が右側のテキストフィールドに出力される．Java Swing と C Motif でサンプルアプリケーションを実装した．サンプルアプリケーションの実行画面を図 9 に示す．

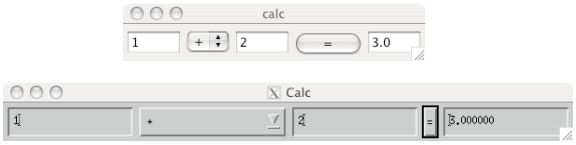

図 9: サンプルアプリケーション (上 Java Swing , 下 C Motif)

サンプルアプリケーションに出現する GUI ツールキットの語彙の差異を吸収するために Mapping Rule と Analyzing Rule を作成する．Mapping Rule で GUI ツールキットを以下のように共通 GUI ツールキットに対応付ける．

- Swing の JComboBox ウィジットと Motif の DropDownList ウィジットを共通 GUI ツールキットの PullDownList ウィジットに対応付ける．
- Swing の text プロパティと Motif の labelString プロパティ，string プロパティを共通 GUI ツールキットの text プロパティに対応付ける．
- Swing の Action イベントと Motif の Activate イベントを共通 GUI ツールキットの submit イベントに対応付ける．

次に Analyzing Rule を作成する．例えば，Swing のウィジットの定義を行うステートメントはウィジットのコンストラクタ呼び出しの代入文と JFrame を継承したクラスのオブジェクトの getContent メソッド呼び出しを検出すればよい．また，Motif のウィジットの定義を行うステートメントは “XmCreate + ウィジット名” の関数呼び出しの代入文と XtInitialize 関数呼び出しの代入文を検出すればよい．関数名のうち，“XmCreate” の部分を affix 要素を用いて判定し，残りの部分でマッチングを行っている．Analyzing Rule の例を図 10，11 に示す．

GUI 抽象化システムで実装言語と GUI ツールキットの異なる 2 種類のアプリケーションを Mapping Rule と Analyzing Rule を変更することで抽象 GUI 記述へ抽象化できることを確認できた．図 12 に Swing のサンプルアプリケーションの抽象 GUI 記述のウィジットの階層構造部分を示す．Motif のサンプルアプケーションの抽象 GUI 記述も同様の GUI モデルが生成できた．

```
<analyzing-rule>
  <analyze>
    <detection>
      <assign>
        <righthand><variable>
            <ident><var name="widget.name"/></ident>
          </variable></righthand>
        <lefthand><constructorcall>
            <type><ident>
              <match mapping="/mapping-rule/mapping
                [@type='widget']/from[@type='type']">
                <var name="widget.type"/>
              </match>
            </ident></type>
          </constructorcall></lefthand>
      </assign>
    </detection>
    <representation>
      <widgets><widget>
          <name><ref var="$widget.name"/></name>
          <type>
            <mapto section="widget" totype="type">
              <from type="type">
                <ref var="$widget.type"/>
              </from>
            </mapto>
          </type>
        </widget></widgets>
    </representation>
  </analyze>
  ...
</analyzing-rule>
```

図 10: Java Swing のウィジットの定義を検出するための Analyzing Rule(コンストラクタ呼び出し)

GUI 抽象化システムは，実装言語や GUI ツールキットによる差異を Mapping Rule と Analyzing Rule に外部ファイル化して複数の実装言語や GUI ツールキットで実現された GUI コードに対応している．このためシステムを変更することなく，複数の実装言語や GUI ツールキットに対応することができる．

5 おわりに

本稿では，多数の実装言語と GUI ツールキットに対応した GUI コードの抽象化手法を提案した．実装言語や GUI ツールキットに依存しない形式で GUI を記述するために GUI の特徴を GUI プログラミングモデルとして定義し，GUI プログラミングモデルを実現するための共通 GUI ツールキットを定義した．多数の実装言語や GUI ツールキットに対応して GUI コードを抽象 GUI 記述へ変換するために，GUI ツールキットと共通 GUI ツールキットを対応付ける規則である Mapping Rule と GUI

```
<analyzing-rule>
  <funccall>
    <ident>
      <affix value="XmCreate"/>
      <match mapping="/mapping-rule/
      mapping[@type='widget']/from[@type='type']">
        <var name="widget.type"/>
      </match>
    </ident>
  </funccall>
  ...
</analyzing-rule>
```

図 11: C Motif のウィジットの定義を検出するための Analyzing Rule(関数呼び出し)

コードに出現する GUI ツールキットの API 呼び出しから抽象 GUI 記述へ対応付けを行う Analyzing Rule を定義した．これらの規則を変更することで複数の実装言語や GUI ツールキットに対応した GUI 抽象化システムを提案した．

謝辞

本研究の一部は文部科学省リーディングプロジェクト基盤ソフトウェアの統合開発 e-Society 高信頼 WebWare の生成技術および文部科学省科学技術研究費若手研究 (B) 課題番号 17700030．文部科学省科学技術研究費基盤研究 (B) 課題番号 17300006 の助成による．

参考文献

[1] User Interface Markup Language (UIML) Specification Draft Language Version 3.0. OASIS User Interface Markup Language (UIML) TC, 2002.

[2] L. Bouillon, J. Vanderdonckt, and J. Eisenstein. Model-Based Approach to Reengineering Web Pages. In *TAMODIA 2002 : Task Models and Diagrams for User Interface Design*, pp. 86–95, 2002.

[3] P. P. da Silva. User Interface Declarative Models and Development Environments: A survey. *Lecture Notes in Computer Science*, 1946:207–226, 2000.

```
<gui-model>
  <widgets>
    <widget name="calc" type="Window"/>
    <widget name="container"
            type="PlainContainer"/>
    <widget name="operand1TextField"
            type="TextField"/>
    <widget name="operatorComboBox"
            type="DropDownList"/>
    <widget name="operand2TextField"
            type="TextField"/>
    <widget name="calcButton"
            type="PushButton"/>
    <widget name="resultTextField"
            type="TextField"/>
  </widgets>
  <widget-composition>
    <composite widget="calcFrame">
      <composite widget="container"
                 layout="horizontal">
        <composite widget="operand1TextField"/>
        <composite widget="operatorComboBox"/>
        <composite widget="operand2TextField"/>
        <composite widget="calcButton"/>
        <composite widget="resultText"/>
      </composite>
    </composite>
  </widget-composition>
  ...
</gui-model>
```

図 12: Swing のサンプルアプリケーションの抽象 GUI 記述 (ウィジットの階層構造部分)

[4] Mozilla.org. XML User Interface Language (XUL) 1.0. Mozilla.org, 2001. http://www.mozilla.org/projects/xul/xul.html.

[5] B. A. Myers and M. B. Rosson. Survey on user interface programming. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 195–202. ACM Press, 1992.

[6] J. Vanderdonckt, L. Bouillon, and N. Souchon. Flexible Reverse Engineering of Web Pages with VAQUISTA. In *Working Conference on Reverse Engineering*, pp. 241–248, 2001.

[7] 新美健一, 山本晋一郎, 阿草清滋. 抽象ソフトウェアエレメントによる CASE ツール開発のためのフレームワーク．電子情報通信学会ソフトウェアサイエンス研究会, 第 103 巻, pp. 19–24, 2004.