ドラッグ＆ドロップを用いたビジュアルプログラミングシステム

小川 徹[†] 田中二郎[††]

アイコンを操作することでプログラムを編集し，プログラムを元にゴールを作成することでグラフィカルな実行を可能とするシステム CafePie を作成した．ここでドラッグ＆ドロップは 2 つの目的に使われる．1 つは静的なプログラムの編集であり，もう 1 つは実行時における視覚的な部分（ビュー）のカスタマイズである．

本論文では，代数的仕様記述言語 CafeOBJ のための視覚的プログラミング環境 CafePie について述べ，その中でドラッグ＆ドロップ操作がどのように使われているかを説明する．特に，ドラッグ＆ドロップを用いたビューのカスタマイズが可能であるかを示す．

Drag and Drop based Visual Programming System

Tohru Ogawa[†] and Jiro Tanaka[††]

We have developed a visual programming system CafePie in which a user edits a program by operating icons. The user can execute the program graphically by making a goal of the program. Drag and Drop is used for two purposes. One is for a static program editing, the other is for a view customization in an execution case.

In this paper, we describe the visual programming system CafePie for CafeOBJ, an algebraic specification language based on term rewriting. We explain how the drag and drop operation is used in the system. The operation can be applied to the view customization.

1. はじめに

ビジュアルプログラミング[1] は，テキストで記述していたプログラムを，グラフィカルな図形を用いて表現し，また，図形を用いてプログラムの編集や実行を行う手法である．このようなシステムとして，以前から Pict[2]， Hi-VISUAL[3]， PP[4] など，様々なシステムが試作されている．

一般に，テキストを用いたプログラミングでは，特別なツールが無くともテキストエディタを使用することで手軽にプログラムを記述することができる．しかし，図形を用いたプログラミングでは，ドローツールを使って図形を編集することも可能であるが，図形の編集には操作が複雑であり，テキストを用いたプログラミングに比べて手間が掛かる．操作が複雑になる理由は，決まった構文に従って図形を配置し，その後でレイアウト調整を行う必要があるためだと考えられる．操作が複雑でなく，図形を用いてプログラミングを行うための手法が必要である．

我々は，ドラッグ＆ドロップ（DnD）操作[5] を用いることにより手軽に編集可能な視覚的プログラミング環境 CafePie[6),7)] を作成した．本システムでは， DnD 操作を 2 つの目的に使用する． 1 つは静的なプログラミングの編集であり，もう 1 つは実行時における視覚的な部分 (ビュー) のカスタマイズである． DnD 操作を用いることで，キーボードを使わなくても手軽にプログラムを編集できる．

本システムは，マウス操作には慣れているがキーボードに慣れていない人に有効であると考えられる．図形で表現されたプログラムは自由にそのビューを変更することができ， CafeOBJ 言語や項書換え系の処理をこれから学習するための教材として役立つ．また，テキストで記述したプログラムをビジュアルに変換するので，グラフィカルな編集や実行することが手軽にでき，簡単なデバッガとしても利用できる．

本論文の新規性は，「CafePie における全てのプログラム編集をドラッグ＆ドロップで可能にしたこと」と「ビューのカスタマイズをドラッグ＆ドロップで実現したこと」にある．

以下では，我々が作成した CafePie を紹介し，その中

† 筑波大学 工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

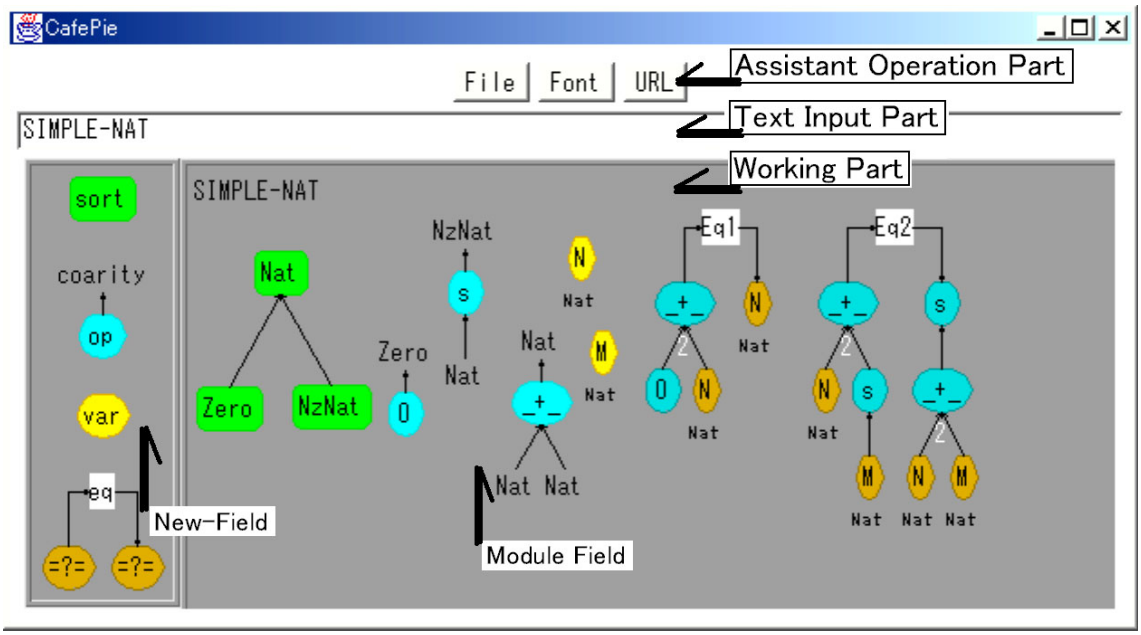


図 1 CafePie の画面
Fig. 1 A snapshot of CafePie.

で DnD 操作がどのように使用されているかについて述べる．

2. システム CafePie

CafePie は代数的仕様記述言語 CafeOBJ[8),9)] のための視覚的プログラミング環境である． CafePie は，CafeOBJ における 1 つのモジュールを視覚化／編集し， CafeOBJ の一側面である項書換え系を視覚化することでプログラム実行を可能としたシステムである（**図 1**）．

2.1 CafeOBJ 言語

CafeOBJ は実行可能なサブセットを持つ仕様記述言語であり，意味論上の基盤として，順序ソート等号論理の拡張である順序ソート書き換え論理を持っている．モジュールをプログラムの基本単位として仕様を記述する．モジュール本体は，まずシグニチャを記述し，その後で公理系を記述する．シグニチャは問題表現のための言語の構文である演算を定義する．言語の意味は公理の集まりで記述する．公理系宣言の中には，変数と等式を定義する．構文要素の数は通常のプログラム言語に比べて非常に少なく，かつ強力なプログラミングを可能としている．項書換え系とみることでプログラムの実行を行うことが可能であり，また，実際に動作するインタプリタがある．簡約順序については，通常は先行評価（eager evaluation）を行う．また，必要に応じて演算毎に遅延評価（lazy evaluation）を行うなど評価順序を指定することも可能である．名前の有効範囲（スコープ）はモジュール全体である．ただし， 1 つの等式内のみ有効な局所変数を定義することができる．一度定義した変数はモジュール内部で有効である．重複宣言した場合には，一番最後に宣言した名前が有効になる．ブロック構造であるが， CafeOBJ ではモジュールは基本的にシグニチャを記述し，その後で公理系を記述しており，その中にネスト構造は現れない．

2.2 CafeOBJ の例

図 2 に CafeOBJ の仕様（プログラム）例を挙げる．これは自然数の上に定義した足し算のプログラムである． ソート宣言は

```
module SIMPLE-NAT
{
  [ Zero NzNat < Nat ]
  signature {
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _+_ : Nat Nat -> Nat { comm, assoc }
  }
  axioms {
    vars N M : Nat
    eq [0] : 0 + N = N .
    eq [1] : N + s(M) = s(N + M) .
  }
}
```

図 2 自然数の上に定義した足し算
Fig. 2 Natural numbers under addition.

［ 下位ソート < 上位ソート ］

という形で記述する．キーワード **“signature”**に続く { から } までがシグニチャ宣言である．キーワード **“op”** はオペレータ宣言の開始を示す．オペレータ宣言は新たな演算を導入し，同時にいくつかの引数の型（ソート）と 1 つの結果の型（ソート）を与える．また，必要に応じて { から } までの間に演算の諸属性を指定することができる．次に，キーワード **“axioms”**に続く { から } において公理系を記述する．キーワード **“var”** は変数の宣言であり，等式で用いる変数を定義する．ここでは，**“vars”** とすることで複数の変数を同時に定義している．キーワード **“eq”**で始まりピリオドで終わる記述が等式の宣言である．等式はキーワード **“=”**で区切られた 2 つの項から成る．項は演算名と変数名から構成される記号列であり、データやデータに対する演算の適用を表現する．

2.3 CafePie の画面説明

CafePie は図 2 のようなプログラムを読み込み，画面上に自動的に視覚化する（図 1）．画面中央（Module Field）がモジュールを作成する部分である． Module Field の左上には，作成中のモジュール名がある．CafePie は Module Field 上に読み込んだプログラムを表示する． Module Field の左側が，プログラムに新しい要素を追加するために用意した New-Field である．これらをまとめて Working Part と呼ぶ．また，ラベル変更を行うための Text Input Part やプログラムファイルの呼出しなどを行うために Assistant Operation Part が用意してある．

2.4 ビジュアルシンタックスの導入

テキストで書かれたプログラムを CafePie 上で扱うためには，何らかの方法を用いてテキストから CafePie 上の要素に変換する必要がある．CafePie では，CafeOBJ 言語の基本的な要素である ソート／演算／変数／等式に対してビジュアルシンタックスを定義している．ビジュアルシンタックスは，システム上での操作可能な要素（アイコン）によって表現される．

ソート： ソートの順序関係は半順序であるので，これを有向グラフで記述する．各ソート（Nat, NzNat, Zero）を緑色の矩形とし，順序関係を表すために “下位ソート” から “上位ソート” へと有向線で結ぶ．

演算： 演算は演算名と幾つかの引数と 1 つの返却値から成る．引数と返却値はソートによって与えられる．各演算（0, s, _+_）は演算名をラベルとして持つ水色の楕円で表現し，引数を演算の下方に，返却値を上方に配置する．引数から演算，演算から返却値へと有向線を引く．

変数： 各変数（N, M）は変数名とソートから成る．変数名をラベルとして持つ橙色の楕円で表され，その下方にソートを配置する．

等式： 各等式（Eq1, Eq2）は左辺と右辺の項とラベルから成る．ラベルを中央に置き，その左下には左辺の項を，その右辺には右辺の項を配置する．左辺から右辺に書換えるということを明確にするため左辺から（ラベルを通して）右辺へと有向エッジを引く．

このようなビジュアルシンタックスを用いることで，CafeOBJ の 1 つのモジュールにおける基本的な要素（ソート／演算／変数／等式）は， CafePie 上で表現可能である．

2.5 プログラム作成手順

CafePie 上でプログラムを作成する場合には，CafeOBJ プログラムのファイルを読み込む以外に，一から編集することも可能である．その場合の手順は，以下のようになる．

(a) モジュール名の編集： 図 1 の Module Field 上をクリックし，モジュール名を入力する． (b) ソートの作成（図 1 の Module Field 左端）： New-Field 上のソートアイコンを移動し Module Field 上にコピーする．各ソートの関連付けを作成する（**表 1** 参照）． (c) 演算の作成（図 1 の Module Field 左中）： New-Field 上の演算アイコンを移動し Module Field 上にコピーする．ソートアイコンを用いて 演算の型や引数を決める（表 1 参照）． (d) 変数の作成（図 1 の Module Field 右中）： New-Field 上の変数アイコンを移動し Module Field 上にコピーする．ソートアイコンを用いて 変数の型を決める（表 1 参照）． (e) 等式の作成（図 1 の Module Field 右端）： New-Field 上の等式アイコンを移動し Module Field 上にコピーする．右辺，左辺に当たる項を作成する． (f) 必要に応じて各アイコンをクリック選択しラベルを変更する．

3. ドラッグ＆ドロップによるプログラム編集

CafePie 上に表示されたアイコンは，マウスを用いて Module Field 内を自由に動かすことができる．

3.1 操作対象の限定

CafePie では，操作対象をアイコンに限定し，それ以外は操作しないようにすることで，簡潔な操作を実現している．アイコン同士の関係を示すエッジを操作対象に加えることも可能であるが，エッジを操作することは，ビジュアルシンタックスにはあり得ないような図形を編集できてしまう可能性を増大させ，構文エラーの原因と

表 1 ドラッグ&ドロップによるプログラム編集
Table 1 Program Editing using Drag and Drop

ACTION NAME	SOURCE	TARGET	ACTION
ソート関係の作成	ソート	ソート	DnD 先を元の上位ソートとする
ソート関係の削除	ソート	ソート	ソート間の関係削除
引数の追加	ソート	演算	演算に引数を追加する
引数の変更	ソート	引数	引数を別のソートに変更する
引数の交換	引数	引数	引数の順番を入替える
項の生成	演算	変数	演算から項を生成し，変数を上書きする
項の追加	項	変数	項で変数を上書きする
変数名の変更	変数	変数	DnD 先のラベルを DnD 元と同じにする

成りかねない．マウスによりエッジを操作しなくても，エッジの追加や削除はアイコン操作で実現可能である．

3.2 ラベルの扱い

キーボードから入力を入力しなくても，変数名やソート名などの全てのラベルは，アイコン生成と同時に自動的に付加される．例えばソートのラベルは，生成した順に Sort1, Sort2, Sort3, … となる．また，システムが既に定義しているラベル情報を全て把握しているため，アイコンの作成時に既存のラベルを調べて，新しく生成するラベルが重複しないようにすることができる．このためキーボードを用いなくてもプログラミングが可能である．

キーボードからの入力は補助的に用いる．ユーザは必要に応じてキーボードを用いてラベルを変更することが可能である．ユーザが故意にラベルを変更しない限りはラベルの重複は起こらない．キーボードから直接入力できるラベルは，モジュール上に定義されたソート，演算，変数，等式のみに限定した．他に演算の引数や返却値や変数のソート，等式の両辺の項に現れる演算や変数のラベルを変更することも考えられるが，スペルミスを防止するためにキーボードからの直接編集はできないようにしている．演算の引数や変数のソートは，ソートをその上に DnD することで変更する．項に現れる変数は，モジュール上の変数や項上の変数を使い，変更したい変数に DnD することでラベルの変更を行う．項上の演算を直接変更することはできない．この点については，後ほど「変数の具体化による項の編集」において述べる．また，ユーザがラベルを変更すると，関係のある全てのラベルも同様に変更される．例えばソートのラベルを変更すると，演算の引数や返却値や変数のソートも同じように変更される．演算のラベルや変数のラベルを変更すると，項の中に現れる演算も同様に変更される．このようにラベル編集におけるユーザの負荷を軽減している．

3.3 ドラッグ&ドロップ操作

プログラムの編集は， DnD 操作を用いて表現することができる． DnD 操作は，従来からデスクトップのアイコン操作に用いられてきた手法であり，ファイルの移動などに用いられている． DnD 操作は，**選択／移動／重ね合わせ**という 3 つの操作から成る．そして，重ね合わせたと同時に何らかの動作（ACTION）が発生すると見なされる．この ACTION は，選択／移動しているアイコン（SOURCE）と重ね合わせるアイコン（TARGET）との種類によって区別することが可能である．

CafePie 上で DnD を用いてプログラム編集を実現するために， SOURCE アイコンと TARGET アイコンの種類によって表 1 のような ACTION を割り当てた．自動的にラベル付けされたアイコンを用いることで，CafePie における全てのプログラム編集操作は DnD 操作で実現可能となった．また，アイコンをドロップした後には，ビジュアルシンタックスで定めたレイアウト方式に従って，自動的にレイアウトされる．これにより図形のレイアウトを気にすることなくプログラミングの編集を行うことができる．

3.4 変数の具体化による項の編集

項の編集は，プログラムの動作を決定する等式の作成や実行時におけるゴールの作成など， CafePie 上では非常に重要な作業である．ここでは， DnD 操作を用いた項の編集方法を述べる．

我々は，項の編集を簡潔に表現するために，「変数の具体化」という概念を用いてモデル化した．この概念を利用し，項の追加を変数に対する具体化操作とみなす．従って，項の追加／削除は変数に対する操作として扱うことができる． CafePie では，項を木構造で表現し，演算をノード，演算の引数に対する具体化の関係をエッジで表している．

DnD 操作を用いた項の編集は，**初期化／追加／削除**という 3 操作の繰返しで定義できる．

- 変数を作成する（項の初期化）．
- 変数の上に演算をドラッグして重ね合わせる（項の追加）．このとき変数は演算によって置き換わる．

もし，演算に引数があるならば，その引数はそれぞれ変数に置き換わる．

- 項にある演算のラベルをドラッグして編集空間の枠外に移動させる（項の削除）．この演算より下の項（部分項）は全て削除され，変数に置き換わる．

項は演算と変数の組合わせによって表現されるが，この具体化操作は，変数に対してのみ行われる．それ以外の部分（演算）にはいっさい変更の操作をしないようにした．このように具体化によるモデル化を行うことで，項の編集を簡潔に表現できる．

ここで具体的な例を挙げる．等式（Eq1）の左辺（N:Nat + s(M:Nat) ）を作成してみる（**図 3**）．

[1] まず， New Field を利用して 変数（V）を作成する．

[2] 次に，変数（V）の上に演算（_+_）を DnD する．これにより変数（V）は演算（_+_）に上書きされる．また，演算に引数がある場合は，演算のビジュアルシンタックスにしたがって，引数をソートとする変数が追加される．変数のラベルはシステムにより自動的に付加される．ここでは V1, V2 としている．

[3]（必要に応じて変数名 V1 を N に変更する．既に変数 N が定義されている場合には，その変数を用いて V1 に DnD することでラベルをコピーする．そうでない場合にのみキーボードを利用する．特に変数名がそのままでもプログラムの実行は可能である．しかし，例えば図 1 の等式 Eq1 のように，右辺と左辺の変数 N は同じものを表すが，ラベルは自動生成されるため別のラベルになる．この場合には，一方をもう一方に DnD し，ラベルのコピーを行うことで同じラベルすることができる．）

[4] さらに変数（V2）の上に演算（s）を重ね合せる．これと同時に引数が変数（V3）に置き換わる．

[5]（必要に応じて変数名 V3 を M に変更する．）

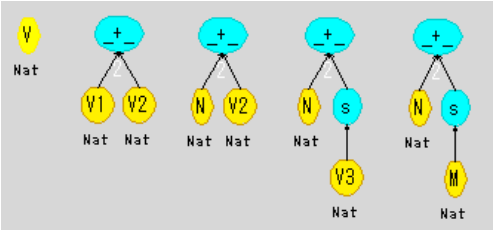


図 3 項の作成手順
Fig. 3 A process of making term.

4. プログラム実行と視覚化

ここでいうプログラムの実行とは，項書換え（TRS）を意味する． TRS は，項の書換えのみで計算が進む計算モデルであり，その形式の単純さと直観に優れた代数的意味論が特徴である． TRS は書換え規則（等式）の集合であり，書換え対象の項は計算の静的な状態を表す．

4.1 プログラムの実行処理

CafeOBJ の処理系は CafeMaster と呼ばれるネットワークサーバを介して利用することができる． CafePie は，ソケット通信を用いてサーバにアクセスし，既存の CafeOBJ の処理系を利用することで，項書換えの処理を実現している．

プログラム実行においてユーザが行う作業は，書換え対象となる項（ゴール）を作成し，ゴールを CafeOBJ の処理系に渡すことである．

ゴールの作成： ゴールの作成は，プログラム編集における項の作成と同じようにして行う．ユーザは，Module Field 上に変数を作成し， DnD 操作を用いて変数の上に演算を重ね合せることによって作成していく．

処理系への受渡し： 処理系に必要な情報は，書換え対象のゴールと，書換えの処理に必要なプログラムである．ユーザは， DnD 操作を用いて作成したゴールを Module Field の左上にあるモジュールラベルの上に重ね合せることで，処理系への受渡しを行う．

CafePie が行う処理は， (1) CafeOBJ の起動要求，(2) データのテキスト化処理， (3) サーバへの実行要求， (4) 結果の視覚化処理と表示，である．ゴールが正しく解釈されれば，インタプリタは項書換えた結果を返す．この結果は，書換え前と書換え後などの情報からなる簡約の組から構成される． CafePie はサーバから実行結果を受け取ると，その都度，その簡約手順に従って視覚化処理を施し，ゴールを書き換える．これによりユーザには，あたかもゴールが動的に変化したかのように見える（**図 4**）．書換えの終了は，ゴールの変化がなくなったことによって確認することができる．書換えが終了したゴールは，また編集し直すことができる．この切替えは，処理系への受渡しと同じように，項をモジュールラベルの上に重ね合せることで行う．

このようにビジュアルで実行過程が見えることは，デバッグの補助に役立ち，また，言語の初心者にとっては理解の助けとなる．

4.2 プログラムにおけるモデルとビュー

ここで，テキストで書かれた内容をプログラムの**モデル**，そしてプログラムモデルに対応した視覚的な部分を**ビュー**と呼ぶことにする．例えば，スタックについてみ

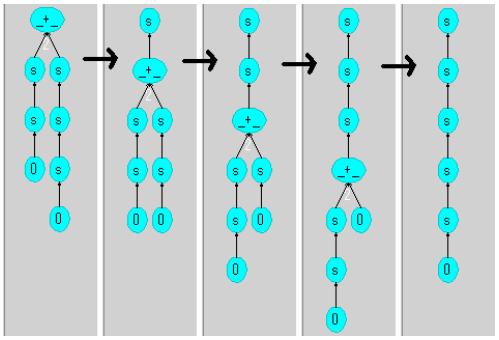


図 4 動的な実行の視覚化
Fig. 4 A dynamic visualization of program execution.

ると，項のモデルは，

```
push(E3,push(E2,push(E1,empty)))
```

のように表現できる． CafePie において，このモデルに対応するビューは，**図 5** のような木構造で表現される．

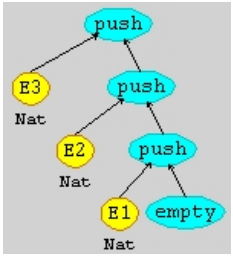


図 5 木構造
Fig. 5 Tree structure.

4.3 実行におけるビューの変更

スタックにおける実行を考えてみる．例えば，スタックにおけるゴールのモデルを，

```
pop(push(E1,push(E3,push(E2,pop(pop(
  push(E3,push(E4,push(E1,empty))))))))
```

のように与える．このモデルを木構造で表現した場合，スタックにおける基本的な構造部分と動作とが入り交じっているために視認性が損なわれる恐れがある．

我々は，ビジュアルプログラミングの醍醐味は，プログラムの視覚化にあると考えている．プログラムの実行において，ゴールを動的に変化させることでその動作を視覚的に捉えることができるが，スタックのように，プログラムの種類によっては動作を理解するのが困難な場合もある． CafePie では，項におけるビューを変更することによって，プログラムの視認性を向上することができる．

ビューを変更すると等式は**図 6** のようになる．図 6 左が変更前，図 6 右が変更後である．ここでは，スタックの静的な構造を与える演算 push に対して積木構造のようなビューの変更が行われている．

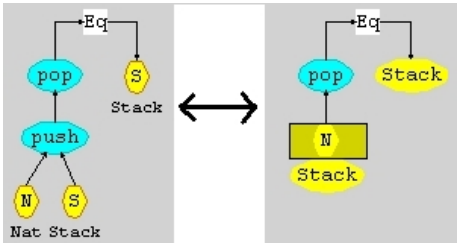


図 6 ビューの切替えと制御構造の表現
Fig. 6 Changing a view and expression of control structure.

実際に，図 6 右の表現を用いてゴールの書換えを行うと，**図 7** のような視覚化が行われる．

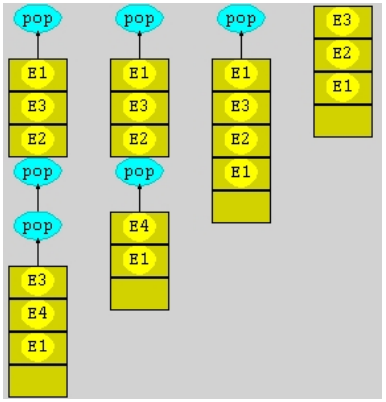


図 7 カスタマイズ表現を用いた実行表示
Fig. 7 Visualization of an execution with customized view.

4.4 アニメーションの設定と表示

実行を動的に表示した場合，ユーザが与えたゴール（図 7 左）はその場で書換えられる．そして，スナップショットを一定の間隔で次々に表示させていき，最後（図 7 右）まで書換えられるとアニメーションが終了する．

しかし，スナップショット形式で表示しているため，同じような演算が重なっていたりすると，どの部分に対して書換えられたのか判断にしくい．例えば，図 7 左端には pop を重ねて表示している箇所がある．これから図 7 左中に書き換わる場合，どちらの pop を書き換えたのか判別しにくい．図形の書換えをスムーズに見せる仕組みが必要である．

我々は，図形書換えルールである等式の間に，**スナップショット間の状態を補間できるようにする**ことでアニメーションを定義できるようにした（**図 8**）．

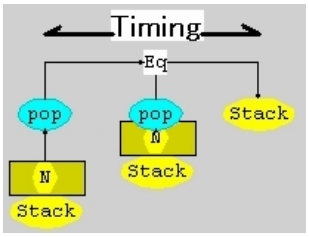


図 8 アニメーションの補間設定
Fig. 8 Setting up an interpolation of animation.

図 8 の左側と右側の間に，スナップショットをスムーズに見せるために補間した図（図 8 の真ん中）が挿入してある．これは，左側を評価して右側に書き換える途中に，全体を 1 とすると 0.5 のタイミングで表示することを意味している．これにより，プログラムの実行は**図 9** のようにスナップショット間が補完されてよりスムーズなアニメーションとして表示される．

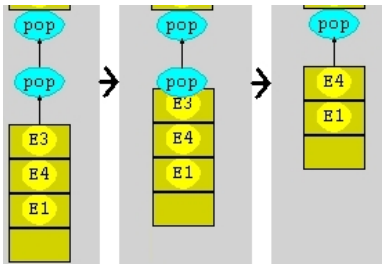


図 9 補間されたアニメーションの表示
Fig. 9 An expression of an interpolated animation.

図 8 の横軸の（ラベル Eq の左右に伸びる）線は，補間する図を表示するタイミングを表す．真ん中の図をドラッグすることで左側に近づければより速いタイミングで，右側に近づければより遅いタイミングで，真ん中の図を表示することになる．等式以外の所にドロップすることで補間する図形を削除することができる．また，ユーザが等式の左（または右の図形）を真ん中の方に DnD することでそのコピーが作成される．ユーザはこの図形の一部を移動させたりして間の図形を編集する．スナップショット間に補間する数を，2 個，3 個というように増やすことによって，よりスムーズなアニメーションを作成することができる（補間数を増やし過ぎると処理しきれなくなり全体として遅くなる）．

5. ドラッグ＆ドロップを用いたビューのカスタマイズ

ここでは，プログラムにおけるビューの変更について述べる．

我々は，プログラムをより視覚的に捉えてそのモデルを推測しやすくすことが必要であると考え，ノードとエッジではなく，より具体的な表示を支援することが重要であると考えた．プログラムの中でも特にデータ構造を表す項は，プログラムの動作を与える等式やプログラム実行の表示に頻繁に使用される．我々は，ユーザが**項におけるビューをよりリアリスティックな表現に変更可能**にすることでこの問題を解決することを考え，研究を行っている[10),11)]．この項におけるビューの変更を，ビューのカスタマイズと呼ぶ．

ビジュアルプログラミング上でビューのカスタマイズを実装するに当たって，次の点を重要視した．

- 簡単にカスタマイズが可能である
- 元のビューへの切替えが容易である
- プログラムのモデルは変更しない

5.1 項におけるレイアウト

項のレイアウトは，項の構成要素である演算に左右される．演算における引数は，項における部分項に対応する．したがって，各演算とそれ以下の引数との位置関係を決めることで，項全体のレイアウトが定まる．

スタックにおける演算 push は，2 引数演算である．ある要素（ここでは Natural Number） `Nat` を，他のスタック `Stack` に積むという意味を与える．このモデルのデフォルトのビューは，**図 10** 左となる．スタック

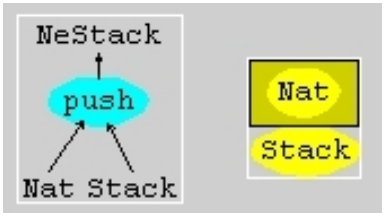


図 10 push のカスタマイズ
Fig. 10 Customizing an operator “push.”

におけるビューを積木構造に変更する場合には，この演算 push に対してレイアウトを（例えば図 10 右のように）定義する．

演算と引数とのレイアウトは，演算と各引数との位置関係を定義することによって行う．2 つの図形間の位置関係を定義することに他ならない．2 つの図形間の関係だけをみれば，DnD 操作を用いて定義することが可能である．

5.2 ドラッグ＆ドロップ操作

我々は，ビューカスタマイズにおける図形編集手法に対しても DnD 操作を採用している．ユーザは基本的に，**図形の移動**と**拡大／縮小**を使って編集を行う．図形は，マウスカーソルで選択し，それをドラッグすること

で自由に移動することができる．また，図形を拡大／縮小する場合は，図形の端をマウスカーソルで摘み DnD 操作することで行う．

カスタマイズで扱える図形オブジェクトの種類は，変数以外に，矩形，丸矩形，楕円とユーザ定義によるものがある．各図形には，色／大きさ／位置などの属性を持つが，カスタマイズによって決めるのは大きさと位置関係である．

ユーザのドラッグ操作に応じて，システムは現在の図形間の関係を例示する．ユーザは，その例示に応じて現在の状況を認識することができ，スムーズに図形間の関係を定義できる．

5.3 ビューのカスタマイズにおける操作例

スタックの push を積み上げるとき，中心がズレたり，幅の長さが違う場合には，うまく揃わない場合がある．これを解決するために，**配置の調整機能**を持たせてある．通常の図形オブジェクトはビューの構成にすぎないので，この場合，配置の調整が行われるだけである．変数オブジェクトは，実際の項の編集において代入操作によって何に置き換わるか特定できないため，サイズが変化することを考慮する必要がある．変数オブジェクトを操作する場合，配置調整以外に項の編集時に有効になる図形的な制約が付く．

矩形の内側に引数 Nat を収める場合を考える（**図 11**）．矩形の中ほどに Nat をドラッグする．ドラッグ中に矩形と Nat の（高さの）中心が揃うと，システムはそれを示すために横線「−」を提示する（図 11 左端）．ユーザが Nat をそのまま横にスライドさせ，（横幅の）中心が揃うと，中心が揃ったことを示すために今度は十字線「＋」を表示する（図 11 左中）．このときに Nat をドロップすると，システムは中心を揃えるという調整を行う．

Nat の下に引数 Stack を配置する場合を考える．矩形の下側の方に Stack をドラッグする．（横幅の）中心が揃ったときにドロップすることで（横幅の）中心が自動的に揃う（図 11 右中）．（横幅の）大きさを揃えたい場合には，Stack の大きさを拡大／縮小して調整する．左右の端が揃ったことを示すために，システムは図 11 右端のように図形の両脇に縦線「｜」を表示する．

5.4 カスタマイズされた配置の調整

項の編集における我々のアプローチは，まず，各部品に対してそのビューを定義し，それらを組み合わせることで全体のビューを作成するという，ボトムアップによる方法である．このボトムアップ方式の問題点として，**図形を組み合わせて編集してみないと全体像がわかりにくい**ことである．これを解決するために，図形の編集中

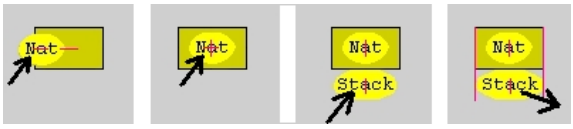


図 11 ドラッグ&ドロップによるプログラム編集（ビュー変更後）
Fig. 11 Program editing using drag and drop (after view changed).

において，対応するビュー対応画面を呼び出すことで，いつでも項のビューを修正をすることができるようにした．例えば，**図 12** 左上のようにスタック間の隙間があり，これを除去したいとする．この場合には，ユーザがスタックの一部（push）にあたる部分を指定すると，システムはそのビューをカスタマイズしている push のビュー対応画面（図 12 右上）を表示する．図 12 左下のように DnD で Stack を移動させ矩形下と Stack の上を隣接することで，図 12 右下のように隙間のないように配置することができる．

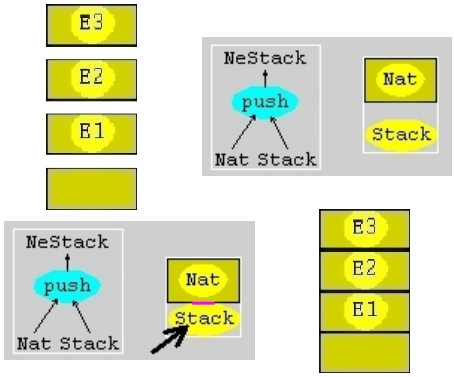


図 12 配置の調整
Fig. 12 Adjustment of arrangement.

5.5 ビューの切替え

カスタマイズ前のスタックを図 5 のように木構造である．この場合， push には図 10 左のビューが定義されている．このビューからポップアップメニューを呼び出し，カスタマイズ後のビューである図 10 右に切り替えることによって，木構造というビュー全体を積木構造（**図 13**）に変更することができる（この場合は，演算 empty に対するビューも同時に切り替えている）．また，逆に積木構造から木構造にビューを戻すことも可能である．

5.6 カスタマイズ機能の適用例

このカスタマイズ機能を適用することによって，一つのプログラムモデルに対し異なるビューを定義することが可能となる．例えば，積木構造（図 13）の代わりにスタックを人の並びと見なし，演算 empty を行き止ま

図 13 積木構造
Fig. 13 Building blocks.

り，演算 push を列の最後に人が並ぶこと考える．スタックの要素を人の顔の表情として他に定義することで，図 13 の代わりに，**図 14** のような視覚化を行うことも可能である（ただし，ここでは 7 個のスタック要素から成る）．

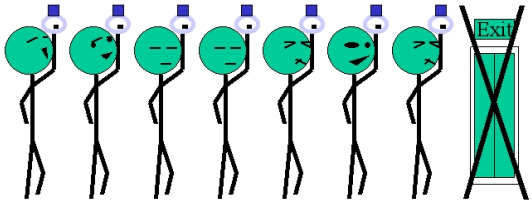


図 14 視覚化（人の並び）
Fig. 14 A visualization as a row of people.

現在のドラッグ＆ドロップを用いて定義できるレイアウトには制限がある．レイアウトを行う場合には，2 つの図形間の重なりを利用し，図形を重ねることで接触部分等の判定を行っている．2 つの図形が重ならない場合には，判定することができない仕様になっている．また，ある一点に固定して配置することが難しい．例えば，図 14 の人の顔の部分をスタック要素としているが，現在のドラッグ＆ドロップ手法では，レイアウトの調整ができない．このために，図形に近づいたら距離を表す線などを表示し，距離を一定に離して配置できるようにするなどの工夫が必要であると思われる．

スタック以外にも似たようなリストやキューなどのデータ構造もビューのカスタマイズを使って定義できる．例えばリストは，基本図形として矢印を与えることで，**図 15** に示すように容易に実現できる．

図 15 リストの視覚化
Fig. 15 A visualization of a list.

6. 評価実験

我々は，本システムの評価のために，筑波大学の大学生・大学院生 11 人を対象に評価実験を行った．事前にアンケートを取った結果，全員マウスやキーボードを十分に使用したことがあることがわかった．また，被験者は，本システムや CafeOBJ 言語をあまり使用したことがない人を中心に選んだ．

実験環境：ハードウェアは， CPU が PentiumII 400MHz で 128MB の主記憶を搭載した DOS/V 機を用い， OS は WindowsMe を使用した． SXGA の液晶ディスプレイに画面を表示し，被験者は全て同じマウスとキーボードを用いて実験を行った． CafePie は Java で記述されており，実験には JDK1.3.0 でコンパイルしたものを用いた．評価実験 1 で用いるテキストエディタは， emacs 互換の Meadow1.14 を用いた．

6.1 評価実験 1

評価実験 1 は， CafePie のプログラム編集時における有効性を示すことを目的とする． CafeOBJ のプログラム SIMPLE-NAT （図 2）を与え， CafePie を用いた場合とテキストエディタを用いた場合とでその編集に要した作業時間を測定した．

各実験を行う前に，本システムの使い方を説明してから少し使用してもらい，各被験者に基本的な作業を理解してもらった上で実験を行った．また，各プログラムの作成手順を予め指示しておき，同じような手順でプログラムを作成するという手順をとった．こうすることでプログラムの考察にかかる時間をなるべく排除し，プログラム編集にかかる時間のみを測定するように努めた．

測定は，まず， CafePie 上で DnD のみを用いてプログラムを記述し，最後にラベルを入力してもらった．その後，テキストエディタを用いて同等の CafeOBJ プログラムを編集するという手順をとった．

6.2 実験結果 1

プログラム編集における実験結果を**図 16** に載せる．各グラフは，左から，「DnD のみを用いてラベル入力を行わない場合の CafePie による編集」，「DnD に加えラベル入力を加えた場合の CafePie による編集」，「テキストエディタによる編集」である．縦軸はプログラム編集に要した時間を示しており，各々，ソート，演算，変数，等式にわけて測定し，その平均値の総和をグラフにした．

まず，プログラム編集に要した全体の時間を比較すると， CafePie （DnD のみ）の方がテキストエディタによる場合よりも若干速くなっている．しかし， t 検定を行ってみたが，有意な差は認められなかった．次に，

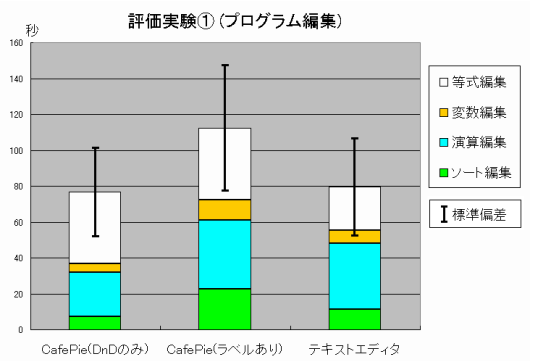


図 16　プログラム編集における評価
Fig. 16　The experimental result of program editing.

CafePie（ラベル入力あり）とテキストエディタによる場合とを比較すると，t 検定で 4.98(>3.17，危険率 0.5%) となり，テキストエディタの方が速く編集できることがわかった．しかし，平均値の割合でみると約 1.41 倍と思ったよりも差は開かなかった．

以上の結果により，テキストエディタを用いた場合と比較しても CafePie を用いてラベル入力をしない場合は同程度の時間で，ラベル入力をした場合でもさほど効率が悪くならないことがわかった．

実験のあとにアンケートをとってみた結果，半分以上がテキストと比べると視覚的に編集できるので分かりやすいという答だった．図形で表記した方が特殊記号を入力する必要がなく概観が捉えやすくて良いという意見もあった．一方で図形が増えすぎるとかえって見えづらくなるという意見もあった．また，操作に関して，DnD 中の図形の重なり判定が分かりにくい意見もあった．操作に応じてどのようなガイドをユーザに提供するか，フィードバック面での改善の余地が必要である．

6.3　評価実験 2

評価実験 2 は，CafePie のビューのカスタマイズ時における有効性を示すことを目的とする．ビューのカスタマイズを直接操作で可能にしたものは他になく，今回は，純粋に 1 つのビューのカスタマイズにかかる時間を測定した．

CafePie におけるスタックの演算 push における積木構造（図 13）を表すビューをカスタマイズし，その編集に要した時間を測定した．

各実験を行う前に，作成手順を示し，その後で約 1 分間 CafePie を使用してから実験を行った．ビューのカスタマイズについて同じ操作を 3 回行い，その時間を測定した．

6.4　実験結果 2

ビューカスタマイズにおける実験結果を**図 17** に載せる．各被験者がビューのカスタマイズに要した時間を

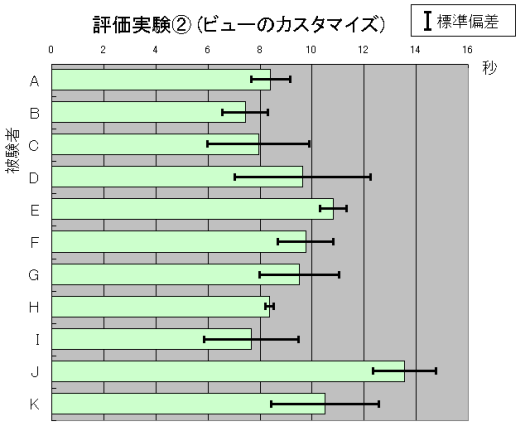


図 17　プログラム編集における評価
Fig. 17　The experimental result of view-customizing.

グラフに表示した．横軸がビューのカスタマイズに要した作業時間である．各グラフは，測定した作業時間の平均値を表している．

グラフを見てもわかるように，作業時間は平均で 9.42 秒（標準偏差 1.78）であり，これは，評価実験 1 で行った簡単なプログラム編集にかかる時間と比較してもほんの 1 割程度にすぎない．CafePie を使い始めてまもないにも関わらず，ビューのカスタマイズに要する時間は非常に少ないことがわかる．アンケートでも，操作中に現在の状態がガイドで表示されるため非常にわかりやすいという意見が得られた．このことから，ビューのカスタマイズはそれほど時間をかけずに手軽に行えると言える．

11 人中 6 人は失敗せずに一度で編集できたが，その一方で，残りの人達は一度では編集できず修正をする必要があった．その人たちのアンケートには，操作する対象が小さすぎて微調整が難しい などの意見があった．対象物を大きくするなどの工夫が必要である．

7. 関 連 研 究

BITPICT[12] や Visulan[13] などは，ビットマップの書換え規則の集合でプログラムを表現する．我々のシステムではベース言語を持たせているため，基本的に異なったシステムであるが，等式を書換え規則として捉えた場合，類似性がある．また，子供用の例示プログラミング環境である KIDSIM[14] では，格子状に区切られた画面と格子上の物体の書換え規則の集合によってプログラムを表現する．格子上の物体を DnD 操作を用いて動かすことにより，物体の移動（書換え）規則を例示で示

すことができる．DnD を書換え規則の定義ではなく，ビューのカスタマイズに用いている点で異なる．さらに編集操作の際にフィードバックを与えるなどの工夫している．

GUI シェル dish[15] は，機能を持たせたアイコンをドラッグ＆ドロップ操作することによってシェルプログラムなどを作成可能なシステムである．基本的なオブジェクトとは別に，clone 等の操作用のアイコンがある．CafePie におけるプログラム編集では，操作対象は基本的なアイコンに限定している．さらに項の編集操作のようにプログラミング言語の意味を持たせるなど工夫している．

GELO[16] は，データ構造の視覚化においてユーザがカスタマイズ可能なシステムである．これは直接操作によるビューの編集方法は述べていない．北村ら[17] は，GhostHouse を用いてカスタマイズ方式による GUI 構築を提案している．GUI 構築の際にドラッグ＆ドロップを用いて GUI 部品間の関係付けを行うことができる．この際に，置換／吸収などの概念を用いるなど工夫している．編集を対話的に行うことが可能であるが，操作に応じたフィードバックについては述べられていない．

Chimera[18] は，図形エディタなどにおける編集操作の履歴をマクロ定義するための例示システムである．GUI 操作をグラフィカルに表示したまま編集可能である．我々の実装ではシステムドラッグ＆ドロップ操作に応じてレイアウト状態を動的に例示しており，インタラクティブな操作を重視しているという点で異なる．

8. ま と め

ドラッグ＆ドロップ操作を用いることにより，静的なプログラムの編集と実行時におけるビューのカスタマイズを可能とする視覚的なプログラミング環境を作成した．図形的な編集はテキストに比べると面倒だという問題があったが，図形的な編集を全てドラッグ＆ドロップ操作で実現し，ユーザの操作に応じた結果のフィードバックを例示的に与えることで簡単に編集できるようになった．また，操作したあとに自動的にレイアウトされるため，位置や大きさを気にせず手軽に編集が行えるようになった．

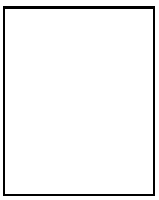
CafePie では，１つもモジュールにおける基本的な要素である，ソート／演算／変数／等式を視覚化することができる．しかし，これでは CafeOBJ の全てを視覚化したことにはならない．CafeOBJ の機能にある演算の属性の表現方法や複数モジュールへの対応が今後の課題である．

参 考 文 献

1) Myers, B. A.: Taxonomies of Visual Programming and Programming Visualization, *Journal of Visual Languages and Computing*, Vol. 1, No. 1, pp. 97–123 (1990).
2) Glinert, E. and Tanimoto, S.: PICT: An Interactive Graphical Programming Environment, *IEEE Computer*, Vol. 17, No. 11, pp. 7–25 (1984).
3) Hirakawa, M., Tanaka, M. and Ichikawa, T.: An Iconic Programming System, HI-VISUAL, *IEEE Transaction on Software Engineering*, Vol. 16, No. 10, pp. 1178–1184 (1990).
4) Tanaka, J.: PP : Visual Programming System For Parallel Logic Programming Language GHC, *Parallel and Distributied Computing and Networks '97*, pp. 188–193 (1997). Singapore.
5) Wagner, A., P.Curran and O'Brien, R.: Drag Me, Drop Me, Treat Me Like an Object, *Proceedings of CHI'95: Human Factors in Computing Systems*, pp. 525–530 (1995).
6) Ogawa, T. and Tanaka, J.: Double-Click and Drag-and-Drop in Visual Programming Environment for CafeOBJ, *Proceedings of International Symposium on Future Software Technology (ISFST'98)*, Hangzhou, pp.155–160 (1998).
7) Ogawa, T. and Tanaka, J.: CafePie: A Visual Programming System for CafeOBJ, *Cafe: An Approach to Industrial Strength Algebraic Formal Methods*, Elsevier Science, pp. 145–160 (2000).
8) Diaconescu, R. and Futatsugi, K.: *CafeOBJ Report*, World Scientific (1998).
9) Nakagawa, A. T., Sawada, T. and Futatsugi, K.: *CafeOBJ User's Manual*, IPA (1997).
10) Ogawa, T. and Tanaka, J.: Realistic Program Visualization in CafePie, *Proceedings of the fifth World Conference on Integrated Design and Process Technology (IDPT'99)*, Dallas, Texas (2000). (CD-ROM), Copyright (c) 2000 by the Society for Design and Process Science, (SDPS), 8 pages.
11) 小川徹, 田中二郎: データ構造の視覚的カスタマイズとプログラム実行の視覚化, インタラクティブシステムとソフトウェア VIII, 日本ソ フトウェア科学会 WISS2000, 近代科学社, pp. 85–90 (2000).
12) Furnas, G. W.: New Graphical Reasoning Models for Understanding Graphical Interfaces, *Proc. of CHI-91*, New Orleans, LA, pp. 71–78 (1991).
13) Yamamoto, K.: Visulan: A Visual programming Language for Self-Changing Bitmap, *Proceedings of International Conference on Visual*

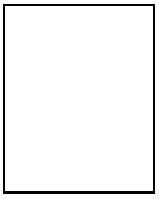
Information Systems, Melborune, pp. 88–96 (1996).
14) Cypher, A. and Smith, D.: KidSim: End User Programming of Simulations, *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, pp. 27–34 (1995). Denver, CO.
15) 早野浩生: ドラッグ&ドロップでスクリプトの記述が可能なシェル, 日本ソフトウェア科学会 第 15 回大会論文集, pp. 169–172 (1998).
16) Reiss, S. P., Meyers, S. and Duby, C.: Using GELO to Visualize Software Systems, *Proc. of the 2nd Annual Symposium on User Interface Software and Technology (UIST'89)*, Williamsburg, VA, pp. 149–157 (1989).
17) 北村操代, 杉本明: 生成・カスタマイズ方式による GUI 構築方法の提案とクラスライブラリ GhostHouse による表現, 情報処理学会論文誌, Vol. 36, pp. 944–958 (1995).
18) Kurlander, D. and Feiner, S.: Inferring Constraints from Multiple Snapshots, *ACM Transactions on Graphics (TOG'93)*, Vol. 12, No. 4, pp. 277–304 (1993).

(平成0年0月0日受付)
(平成0年0月0日採録)



小川　　徹（学生会員）

1975 年生. 1998 年筑波大学第三学群情報学類卒. 2000 年筑波大学博士課程工学研究科在学中に修士号取得. 修士（工学）. 現在, 同大学在学中. ビジュアルプログラミング, ヒューマンインタフェース, 視覚化に興味を持つ. 情報処理学会, 日本ソフトウェア科学会各会員.



田中　二郎（正会員）

1951 年生. 1975 年 東京大学理学部卒. 1977 年同大学院修士課程修了. 1984 年米国ユタ大学計算機科学科博士課程修了、 Ph.D. in Computer Science. 現在、筑波大学、電子・情報工学系教授. プログラミング一般やヒューマンインタフェースに関する研究を行っている. 最近では、 3 次元インタフェースの操作や制約に関心を深めている. ACM、 IEEE Computer Society、 電子情報通信学会、人工知能学会、日本ソフトウェア科学会各会員.