プログラミング 12－5
（1996．11．1）

クライアントGUIアプリケーション向けのビジュアルプログラミング環境

今村 大輔　小山 徳章　植木 克彦　池本 浩幸

(株)東芝　研究開発センター

システム・ソフトウェア生産技術研究所

抄録

ビジュアルプログラミング(以下VP)環境を用いることはソフトウエアの開発効率向上に有効である．VP環境を構築するためには，アプリケーションの特徴を十分にかつ簡単な方法で記述可能なVP言語が必須である．本稿ではクライアント/サーバシステムにおけるGUIアプリケーションを例にとり，独自の分析手法の結果を用いて設計したVP言語と，プログラム作成手順を支援するビジュアルな構成管理機能を報告する．我々はVP言語と構成管理機能を統合してVP環境を作成した．このVP環境を用いて監視用のGUIアプリケーションを作成したところ，VP言語の仕様記述能力が十分であることと，生産性の向上に寄与できることを示すことができた．

Visual Programming System for GUI Application Development

Daisuke Imamura, Noriaki Koyama, Katsuhiko Ueki, and Hiroyuki Ikemoto

Systems and Software Engineering Laboratory
Research and Development Center
TOSHIBA Corporation

Abstract

The use of visual programming (VP) systems is effective in improving software development productivity. In designing a VP system, it is necessary to define a VP language which makes it possible to describe the application's specifications precisely and briefly. In this study, we developed such a VP system. This VP system comprises VP editors and a visual configuration management system. We analyzed a specific industrial application structure using APF, our original framework method, and designed the VP language. Moreover, we analyzed the overall process of software development and visualized it using the configuration management system. In this report, we present several typical specifications of sample applications with the VP system. We then demonstrate that the VP language is sufficient for representing the specifications and that the configuration management system is useful for application development.

1.はじめに

現在，コンピュータを用いた情報システムの多くは次のような特徴を持っている．

- クライアント/サーバシステムのアーキテクチャを採用している．
- ユーザとのインタラクションにGUI (Graphical User Interface)を用いている．

クライアントのGUIアプリケーションは仕様の変更が頻繁に発生し，コスト高になる傾向にある．よって，クライアントのGUIアプリケーションの開発効率を向上させることは，ソフトウェア開発の生産性向上のために有効である．

GUIアプリケーションの開発効率を向上させる一方法として，開発者の認知活動を支援できるという理由でビジュアルプログラミング(以下 VP)環境が提案されてきた[1][2]．

GUIのLook & Feelの作成にVP環境を提供する試みとしては，InterViews[3]のibuildなどがあるが，アプリケーションの動作までは記述不可能であった．一方アプリケーションの動作を視覚的に記述する方法にはDFD(Data Flow Diagram)，CFD(Control Flow Diagram)，STD(State Transition Diagram)といった仕様記述法[4]，STDの拡張であるStateCharts[5]，DFDとCFDの記述力を併せ持つESML[6]があるが，GUIのレイアウトをアプリケーションの動作と関連させて視覚的に記述する能力には欠けている．GUIのLook & Feelとアプリケーションの処理の両方にVPを採用した例としてはLabVIEW[7]があり，計測器材のデータ処理に適用している．LabVIEWは構造化したデータフロー図を用いて制御の記述を行っており，かつGUIを編集するエディタを備えている．しかし，あくまでも処理の記述とGUIのLook & Feelの作成は別のエディタに分かれており，どのGUI部品に対する操作が処理を起こす事象となるか，また，処理結果がGUIのどの部品に反映されるかを一覧できない．

本稿では，GUIのLook & Feelとアプリケーションの処理を同時に作成可能なVP環境を開発し，監視用GUIアプリケーションに適用した事例について報告する．VP環境はビルダ，ジェネレータ，部品ライブラリ，エンジンという構成からなり，ビルダを用いてVP言語の編集とプログラム作成手順の支援ができる．

VP言語は，我々が提案するAPF(Adaptive Plug and play Framework)手法というアプリケーションフレームワークによる分析結果を用いて設計した．

開発効率を向上させるためにVP言語の設計では，頻繁に変更が行われる箇所をプログラムソースより抽象化して図的に表現することが要求される．APF手法は，対象ソフトウエアの構造を明確化し構成要素の一部を抽象的に切り出すことが可能である．

また，プログラム作成手順の支援は，開発効率を向上させるために製品分野の開発工程に適した開発環境を与えることを目的とする．開発者は各製品分野のプログラム作成手順に沿って開発をしており，プログラム作成手順を支援することは意義がある．

大規模で多くの機能に分かれている監視用GUIアプリケーションの作成手順を支援するために，ソフトウェアのアーキテクチャと開発工程を分析し，その結果を用いて構成管理機能を設計，開発した．構成管理機能はプログラム作成手順に沿って開発者をナビゲートすると伴に，成果物を視覚化することにより開発工程を支援するものである．

2.章でVP環境の構成を述べた後，3.章では，APF手法の紹介と分析手順，分析結果を用いたVP言語の設計，実装を述べる．4.章では，開発環境で実現すべき手順の分析と構成管理機能の設計，実装を述べる．5.章では，VP言語と構成管理機能を統合したGUIビルダの適用実験について述べる．

2.VP環境の構成

VP環境は以下の構成要素からなる．

- GUIビルダ
  VP言語の編集とプログラム言語に依存しない定義ファイルを生成するためのツール
- ジェネレータ
  定義ファイルからMotifのプログラムソースを合成するプログラム
- 部品ライブラリ
  GUIの部品やプリミティブな処理の実装ライブラリ
- エンジンライブラリ
  ジェネレータが合成したプログラムとリンクされて画面の表示制御やプロセス間通信を行うライブラリ．

本稿ではGUIビルダに焦点を当て，VP言語と構成管理機能について報告する．

3.VP言語

3.1.APF手法の概要

APF手法はアプリケーションフレームワークと呼ばれるオブジェクト指向技術の設計手法を具体化させたものである．

APF手法は以下の二つのフレームワークを持つ．

1.アプリケーション層，抽象モジュール層，基本部品層の3層からなるフレームワーク(3階層フレームワーク)
2.コントロールフローに基づくSubject, Bind, Viewからなる Smalltalk[8]のMVCに似たフレームワーク(SBVフレームワーク)

以下に上記二つのフレームワークの概要を説明する．

3階層フレームワーク

- アプリケーション層

コントロールフローに基づいてアプリケーションの動作を表す．この層に更にSBVフレームワークを適用する．

- 抽象モジュール層

アプリケーション層で記述されるコントロールの結果生じるデータの流れを抽象化したデータ型に変換して表現し，基本部品層へ伝える．

- 基本部品層

プリミティブな処理，GUIの部品，抽象モジュール層で抽象化したデータ型を実装するライブラリである．

SBVフレームワーク

- Subject

処理を起こす事象を表す．

- Bind

コントロールフローを表す．

- View

Look & Feel及びbehaviorを司るオブジェクトを抽象化したものを表す．

図1にAPF手法の構造を示す．

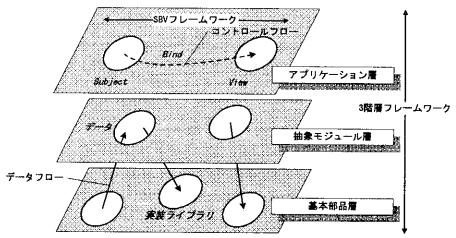


図1 APFの構造

従来のアプリケーションフレームワークとしてはSmalltalkのMVCが良く知られているが，MVCではM(Model)とV(View)，VとC(Controller)の関係に厳密な規定が無く，各カテゴリの定義が曖昧になり易い[9]．対してAPF手法は，対象ドメインを限定することでクラス関連を簡単にかつ明確にすることができる．

3.2. アプリケーションの分析

APF手法を用いて，GUIアプリケーションを分析する．まずGUIアプリケーションの構成要素を列挙すると，プロセス，プロセスのコントロール，GUI部品，ユーザイベント，システムイベント，データ，データのフローがある．これらをフレームワークに当てはめることにより，抽象化する部分を切り出す．

1.アプリケーション層

コントロールフローに基づくので，CFDを用いて記述可能な以下の要素が当てはまる．

- プロセスの抽象クラス
- コントロールフローとその意味

GUIアプリケーションを作成するためには上記に示した各項目に加え，以下に示すGUI及びイベントに関する要素が必要である．

- GUI部品
- GUI部品にて発生するユーザイベント
- サーバからのデータ着信イベント
- タイマーからのタイムアウトイベント

イベントを記述するためには状態遷移図(STD)を用いることが可能であるが，これは実用的ではない．なぜなら，大規模なアプリケーションを記述すると状態数の爆発を招く可能性が高く，また，GUI部品などのイベント発生元との図形的な結合に欠け，直感的でないからである．

そこで，第二のフレームワークであるSBVフレームワークを用い，以下のよう当てはめる．

Subject

- ✓ ユーザイベント
- ✓ サーバからのデータ着信イベント
- ✓ タイマーからのタイムアウトイベント

Bind

- ✓ コントロールフローとその意味

View

- ✓ GUI部品の抽象クラス
- ✓ プロセスの抽象クラス

2.抽象モジュール層

データフローに基づくので，DFDを用いて記述可能な以下の要素が当てはまる．

- データ(DFDではストア)
- データフロー

DFDで記述されるプロセスはアプリケーション層で出現するプロセスと同一のものを指すので抽象モジュール層には割り当てない．

3.基本部品層

以下の実装ライブラリが当てはまる．これらの抽象クラスがアプリケーション層に存在する．

- プロセスのライブラリ
- GUI部品ライブラリ
- サーバトランザクションライブラリ
- タイマーライブラリ

以上に示した通り，GUIアプリケーションの構成要素をAPF手法によるフレームワークに当てはめることより，

アプリケーション層，抽象モジュール層として抽象化する部分を切り出すことができた．

3.3.VP 言語の設計

3.2.節での分析結果を用いVP言語を設計する．

図的に表記されるのは二つの抽象化された階層，即ち，アプリケーション層と抽象モジュール層である．以下に各要素の表記法を説明する．

1.アプリケーション層

View

- GUI 部品

画面に表示されるイメージ(図 2)を用いて表記する．ユーザイベントとデータからのデータフローが接続可能である．

- プロセス

クライアントで行うプリミティブな処理を表記し(図 3 a)，本稿では ***Calc*** という名称で扱う．また，サーバに対するトランザクション要求を表記し(図 3 b)，本稿ではサーバリクエストという名称で扱う．これらはコントロールフローに接続される．

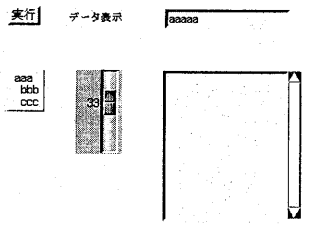
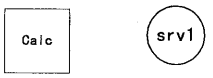


図2 GUI 部品のシンボルの例



a. Calc　b. サーバリクエスト

図3 プロセスの各シンボル

Subject

- ユーザイベント

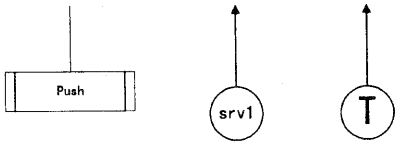
ボックスの中にボタン押下などのイベントの種類を記述する(図 4 a)．GUI 部品とコントロールフローに接続可能である．

- サーバデータ着信イベント

シンボルの中に要求と着信を対応づける ID を記述する(図 4 b)．コントロールフローに接続可能である．

- タイマイベント

コントロールフローに接続可能である(図 4 c)．



a. ユーザイベント　b. サーバデータ着信　c. タイマ

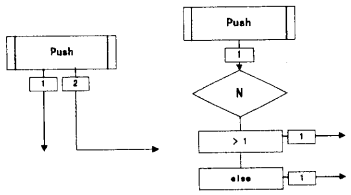
図4 Subject の各シンボル

Bind

- コントロールフロー

有向線分で表記し，発火順序を番号で示す(図 5 a)．イベント，データフロー，プロセスに接続可能で，データフロー，プロセスに接続された時は，各処理内容の制御となることを表す．

また，特殊なコントロールフローとして，if 分岐をサポートする(図 5 b)．



a. コントロールフロー　b. if文

図5 Bind の各シンボル

2. 抽象モジュール層

- データ

DFD のストアと同一に表記する(図 6)．データフローに接続可能である．本稿では ***BP***(Binder Plate)という名称で扱い，データ抽象型を用いている．

- データフロー

破線による有向線分で表記する(図 6)．データ，GUI 部品に接続可能で，GUI 部品に接続された時は，GUI 部品の表示属性へのアップデートとなる．

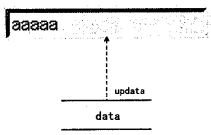


図6 データとデータフローのシンボル

3.4.VP 言語の実装

3.3.で設計した VP 言語を実装するにあたって設計の詳細化を以下のように行った.

- ***BP*** は C 言語の union を用いて型の抽象化を実現する.
- データフローを実現するための ***BP*** のメソッドは以下の通りである.

  CopyBP

  BP から他の BP へデータの内容を複写する.
- コントロールフローを実現するために View に属する抽象クラスが持つメソッドは以下の通りである.

  Update

  GUI 部品のリソースへBPに格納されている内容を文字列型として反映する.

  Open / Close

  画面の表示, 非表示を行う.

  Request

  サーバへのトランザクションを要求する.

  Calc

  プリミティブな計算処理を行う.

VP を行うための GUI ビルダを X Window System† 上に tcl/tk‡ を用いて開発した. GUI ビルダは VP 環境の一部をなすものである. 画面毎のレイアウトと動作の定義は図 7に示す「画面製作エディタ」にて行う.

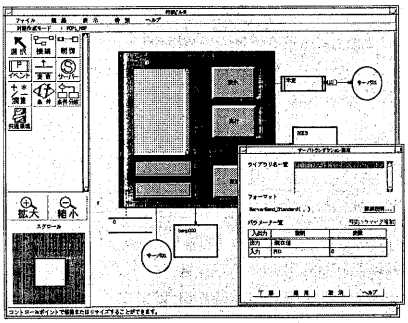


図7ビルダ画面(画面製作エディタ)

画面製作エディタは, 画面のレイアウトを行うモードと VP を行うモードの 2 つを持ち, モードの切り替えにより双方を行き来可能である. また, VP を行うモードで表示されている GUI のレイアウトはレイアウトモードと同一のものであるので, ユーザは二つのモードの存在に違和感を持つことは無い.

4.構成管理機能

4.1.開発工程の分析

開発効率を向上させるためには, 開発工程におけるプログラム作成手順をサポートすることも重要である. 本稿では, 事例として採り上げている製品分野で従来から行われているプログラム作成手順の変更をせずに支援する方法を採用した. この方法は従来の手順が含んでいる欠点を踏襲してしまうが, 新規の手順に移行するコストが発生しない点で有効である. 分析対象とした GUI アプリケーションの開発工程を調査しまとめたものを図 8に示す.

今回の適用対象となっている GUI アプリケーションは, メインメニューから最終的なデータ表示画面に最少の操作数でたどり着くことが重視されるという特質を持っている. よって, プログラマは自分が担当する機能を, 最少の画面遷移パスと各画面にて表示すべき項目数という制約条件の下, 以下の手順に従い作成する.

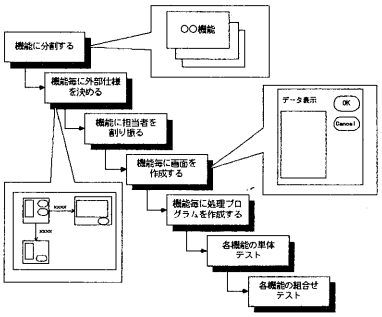


図8 開発工程

1. 自分の分担する機能の定義
2. 機能毎の画面遷移パターンの定義
3. 各画面のレイアウトの定義
4. 画面毎の動作の定義

上記手順を視覚化しナビゲートする開発環境を提供することで, 開発効率の向上が期待できる.

4.2.構成管理機能の設計, 実装

4.1.の分析結果に基づき, プログラマをプログラム作成手順に沿ってナビゲートするための構成管理機能の設計を行った. ナビゲートする手順と, 各々の手順にて視覚化される成果物を以下に示す.

1.プロジェクトの定義または選択
  成果物 : プロジェクト全体の格納場所
2.タスクの選択
  成果物 : 予約されたタスクの格納場所
3.機能の定義または選択
  成果物 : タスクごとの機能の格納場所
4.画面遷移の定義または編集画面の選択
  成果物 : 画面遷移パターンの定義ファイル
5.画面レイアウト / 画面毎の VP の定義

†X Window System は米国 MIT の登録商標
‡ tcl/tk は John K. Ousterhout 氏によって作成されたスクリプト言語及びツールキット

成果物：画面レイアウト及び動作の定義ファイル

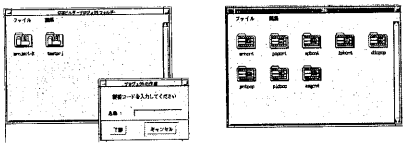
各手順における編集方法は以下の通りとした.

プロジェクト

フォルダメタファを用いたデスクトップ環境を提供し新たなプロジェクトフォルダを作成することでプロジェクトの定義を行う(図9a).

タスク

タスク毎のフォルダを提供する(図9b).



a. プロジェクトフォルダ　　b. タスク毎のフォルダ

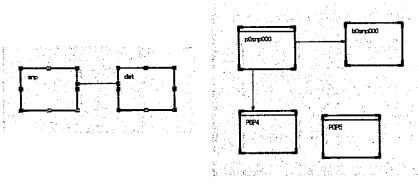
図9フォルダ

機能

機能の構成を視覚的に捉えるために,「機能構成定義エディタ」を与える. 各機能を矩形としてエディタ上に配置し, 矩形同士を結線することで, 機能の階層構造を記述する(図10a).

画面遷移

機能構成定義エディタにおいて編集したい機能を示す矩形をダブルクリックすることで, 機能毎の画面遷移を編集する「画面構成定義エディタ」が表示される. 機能構成定義エディタと同様に各画面を矩形として画面上に配置し, 矩形同士を結線することで画面の遷移関係を記述する(図10b).



a. 機能構成エディタで配置されるシンボル　　b. 画面構成エディタで配置されるシンボル

図10 構成管理機能の各エディタ

画面レイアウト/VP

画面構成定義エディタにおいて編集したい画面を示す矩形をダブルクリックすることで, 図7に示した画面製作エディタが表示される. 画面製作エディタでは, 編集対象となる画面と画面遷移上で隣接する画面をアイコン化して表示する. このアイコンをダブルクリックすることで, 隣接する画面の編集に移行することが可能である. また, VPを行うモードでは複数画面間でデータを共有するための領域を設け, ここにデータのシンボルをドロップすることで他の画面とデータの共有を行う.

5.適用実験

VP環境を監視用GUIアプリケーションの開発に適用し評価を行った. 適用対象とした製品分野のGUIアプリケーションは以下の特徴を持つ.

- 1つの機能が特定のデータ項目に関して「ユーザによる条件入力, 条件に合うデータの表示, ユーザによる任意のデータ設定」という処理手順で完結している.
- 1機能あたりの画面数が2〜4画面である.
- サーバへの要求が1機能につき1〜2程度である.

本VP環境を用いて上記アプリケーションの試作を行ったところ, 完全に仕様を記述でき, 1機能あたりのプログラム作成工数が従来の1/5程度になった.

6.おわりに

本稿では, クライアント/サーバシステムにおけるGUIアプリケーションの開発効率を向上させることを目的とし, 以下の事例について報告した.

- APF手法による分析結果を用いてVP言語を設計, 実装した.
- 開発工程を分析し, プログラム作成手順をナビゲートする構成管理機能を設計, 実装した.

試作したVP環境を監視用のGUIアプリケーションに適用したところ, プログラム作成工数が従来の1/5程度になり, VP言語の仕様記述能力が十分であることと生産性の向上に寄与できることを示すことができた.

参考文献

1 田原, 吉見, 平川, 田中, 市川：“視覚的プログラミング空間に関する一考察,” 情報処理学会 ソフトウエア工学研究会64-16, 1989年, pp121-128

2 “視覚的プログラミング環境,” 情報処理学会 学会誌 Vol.29, 1988年5月, pp485-504

3 Linton,M.A., Vlissides, J.M., and Calder, P.R. : “Computing User Interfaces with InterViews, ” IEEE Computer 22,2(February 1989), pp8-22

4 技術士ソフトウエア研究会編：“ソフトウエア生産工学ハンドブック,” フジ・テクノシステム, 1991年, pp169-178

5 J.ランボー他著, 羽生田 訳：“オブジェクト指向方法論 OMT,” トッパン, 1992年

6 Gernot Richter, Bruno Maffeo : “Toward a Rigorous Interpretation of ESML - Extended Systems Modeling Language, ” Transaction On Software Engineering Vol.19 No.2, 1993 Feb, pp165-180

7 J. Kodosky, J. MacCrisken, G. Rymar : “Visual Programming Using Structured Data Flow, ” Proceeding of the 1991 IEEE Workshop on Visual Languages, 1991, pp34-39

8 Deutch, L. P. :”Design Reuse and Framework in the Smalltalk-80 Systems, ” ACM Press, 1989, ch.A.3.

9 G. C. Murphy, and D. Notkin : “Difficulties with Object-Oriented Frameworks, ” Studies of Software Design. ICSE '93 Workshop Selected Papers, pp77-87