

# インタフェースボードを使用するソフトウェアのための ビジュアルプログラミングシステム

岡田 栄治<sup>†</sup>, 山口 真悟<sup>††</sup>, 田中 稔<sup>††</sup>

インタフェースボードを使用するソフトウェアの作成では次の要件を満たすことが望まれる：1) 処理とその入出力データを記述できる，2) 時間制約を記述できる，3) 処理の並列制御を記述できる，4) ユーザや外部機器からのイベントを記述できる．本稿ではこれらの要件を満たしたビジュアルプログラミングシステムを提案する．提案システムでは，1) ユーザインタフェースの作成，2) 処理構造の記述，3) 動作の記述という3つの側面からソフトウェアを作成する．それぞれの側面では次の図式を使用している：1) ユーザインタフェースを画面レイアウト図でプログラミングする，2) 処理とそのデータ入出力構造をデータフロー図でプログラミングする，3) 処理の並列制御，時間制約を時間ペトリネット図でプログラミングする．最後に例題のプログラミングと評価実験を行い，本システムが要件を満たし有効であることを確かめている．

## A Visual Programming System for Software to Use Interface Boards

EIJI OKADA,<sup>†</sup> SHINGO YAMAGUCHI<sup>††</sup> and MINORU TANAKA<sup>††</sup>

For developing software to use interface boards, the following are to be considered: 1) processing structure with input/output data, 2) time constraint for execution of each processing module, 3) execution timing such as parallel execution of processing modules, and 4) events (signals) from the user or outside devices. This paper proposes a visual programming system for software to use interface boards. In the system, a program can be developed in graphical way by the use of three type of diagrams: 1) the screen layout diagram for the user screen description, 2) the data flow diagram for the processing structure description, and 3) the timed Petri net diagram for describing execution timing of processing modules. The visual programming system is evaluated through developing example programs.

### 1. はじめに

パソコンで外部機器を制御するには，インタフェースボードを使用してパソコンと外部機器を接続し，制御のソフトウェアを開発する必要がある．本研究では，インタフェースボードを使用するソフトウェアの開発に，3つの図式を用いるビジュアルプログラミングシステムを提案する．

インタフェースボードを使用するソフトウェアには以下のような実現すべき要件がある．

#### (1) 処理とそのデータ入出力

#### (2) 時間制約

#### (3) 並列制御

#### (4) イベント駆動

このようなソフトウェアはその利用の性質上，利用者自らがソフトウェアを開発することが多い<sup>1)</sup>．そのため，プログラミングを専門としない者でも開発が行えるプログラミング環境が求められている．

そこで，本研究では上記の要件を満足するために，ソフトウェアを側面ごとに図式で表し，それらを1つのプログラムとして構築するビジュアルプログラミングシステムを提案する．ソフトウェアのユーザインタフェース，処理構造，動作の記述に，それぞれ，画面レイアウト図，データフロー図，時間ペトリネット図を利用する．

以下，2章でインタフェースボードを使用するソフトウェアについて検討し，その開発に必要な要件をまとめた後，3章でその要件を満たすビジュアルプログラミングシステムを設計する．次に4章で設計をもと

<sup>†</sup> 山口大学大学院理工学研究科

Graduate School of Science and Engineering, Yamaguchi University

現在，松下電工ソフトウェア株式会社

Presently with Matsushita Electric Works Software Co., Ltd.

<sup>††</sup> 山口大学工学部

Faculty of Engineering, Yamaguchi University

に実装したビジュアルプログラミングシステムを述べる．そして，5章でプログラム例や評価実験の結果からシステムの有効性を評価し，6章で結論を述べる．

## 2. インタフェースボードを使用するソフトウェア

インタフェースボードにはデジタル入出力ボード，アナログ入出力ボード，GPIBボード，ビデオボード，SCSIボードなど様々な種類が存在する．本研究では，デジタル入出力ボード，アナログ入出力ボード，GPIBボードなどを対象とする．

インタフェースボードにはその制御のためのドライバが用意されており，外部機器を制御するためにこのドライバを使用したソフトウェアが必要である．このようなソフトウェアは機器間でデータの入出力を行い，そのデータを処理することを目的とするため，処理とそのデータの入出力を実現することが重要である．また，設定時間単位の計測，処理ごとの時間制御などが必要であり，実時間制約を満たさなければならない．また，このようなソフトウェアは主に制御対象からのセンサ情報を入力して制御対象へ制御命令を出力する反応型システムであり，複数の並行処理から構成される<sup>2)</sup>．そのため，複数の処理の並列制御を実現しなければならない．さらに，ユーザ，機器からの割込みなど，イベント駆動処理が求められる．

以下にこれらのソフトウェアで実現すべき要件をまとめる．

- (1) 処理とそのデータ入出力
- (2) 時間制約
- (3) 並列制御
- (4) イベント駆動

本研究でデジタル入出力ボード，アナログ入出力ボード，GPIBボードなどを対象とするのは，利用者がその利用目的に合ったソフトウェアを自ら開発することが多いためである．このようなインタフェースボードの利用者のほとんどはこのようなプログラミング自体を専門としているわけではなく，本来の目的とは別に多大な労力を費やすことになる．そのため，このようなソフトウェアの開発環境は要件を満たしたソフトウェアを開発でき，このようなプログラミングを専門としない者でも開発が行えるプログラミング環境であることが求められる．

## 3. ビジュアルプログラミングシステムの設計

### 3.1 構成

伊藤ら<sup>4)</sup>は種々の応用分野に適したネット表現を紹

介して，相補的に用いる有効性を示唆した．本稿では，インタフェースボードを使用するソフトウェアを開発するために，ソフトウェアを側面ごとに図式で示すビジュアルプログラミング言語を提案する．

ソフトウェアには一般にユーザインタフェース，処理構造，動作の3つの側面がある．ユーザインタフェースのプログラミングには画面を視覚的に直接構築できる画面レイアウト図を活用するのが最も有効である．インタフェースボードを使用するソフトウェアの処理構造という側面では2章であげた要件(1)が重要である．本研究では処理構造をプログラミングするために，データフロー図を活用する．それにより，処理とそのデータ入出力を視覚的な表現でプログラミングすることができる．インタフェースボードを使用するソフトウェアの動作という側面では2章であげた要件(2)～(4)が重要である．本研究では動作をプログラミングするために，時間ペトリネット図を利用する．並列制御の記述に優れた時間ペトリネット図を活用することで，並列制御プログラミングを図式の記述で行うことができる．それにより順序・並列などの動作の流れを視覚的に表現することができる．また，処理ごとに時間の制約を記述することができる．また，時間ペトリネット図は状態とイベントを表現することができ，イベント駆動が実現できる．

本システムでは画面レイアウト図によるユーザインタフェースのプログラムを「画面プログラム (screen program)」，データフロー図による処理構造のプログラムを「構造プログラム (structure program)」，時間ペトリネット図による動作のプログラムを「動作プログラム (behavior program)」と呼ぶ．また，それぞれで用いられる構成要素を「プログラム部品」または，単に「部品」と呼ぶ．

表1に3つのプログラムについて実現すべき要件，ソフトウェアの側面，使用する図式を示す．

### 3.2 画面プログラム

画面プログラムはソフトウェアのユーザインタフェースを画面レイアウト図 (screen layout diagram; SLD) を作成することでプログラミングする．

画面レイアウト図は表2に示すウィンドウ (window) と，画面プログラム部品である画面オブジェクト (screen object) からなる．画面オブジェクトにはデータ入力，データ表示，トリガ発生などの機能を持つエディットボックス，スタティックテキスト，ボタンなどがある．

### 3.3 構造プログラム

構造プログラムはソフトウェアの処理構造をデータ

表 1 プログラム構成  
Table 1 The constitution of program.

	requirement	view point	diagram
screen program	—	user interface	screen layout
structure program	data input/output	process structure	data flow
behavior program	time restriction parallel control event driven	behavior	timed Petri net

表 2 画面レイアウト図の構成要素とその表現  
Table 2 Components of screen layout diagram.







component	symbol
window	
screen object	

表 3 データフロー図の構成要素とその表現  
Table 3 Components of data flow diagram.

component	symbol
actor	
process	
data store	
data flow	

フロー図を作成することでプログラミングする。

データフロー図は構造プログラム部品である「アクタ (actor)」、「プロセス (process)」、「データストア (data store)」という 3 種類のノードを、データの流れを表す「データフロー (data flow)」で接続したネットである。表 3 にデータフロー図の構成要素の図形を示す。アクタ、プロセス、データストアの構造プログラム部品はそれぞれデータの入出力、処理、データ蓄積など固有の機能を備えている。各部品は入力端子と出力端子を決まった数だけ持ち、入力端子から入力したデータを処理して出力端子から出力する。各端子数は部品ごとに定まる。構造部品の機能を実行するとき、実行条件 (実行のときに満たすべき条件) が必要であれば、構造プログラミングのときにダイアログボックスを開き規定する。





処理構造の記述にデータフロー図を活用することで処理とそのデータ入出力を図式で視覚的にプログラミングすることが可能になる。

### 3.4 動作プログラム

動作プログラムはソフトウェアの動作を時間ペトリネット図を作成することでプログラミングする。

時間ペトリネット図は動作プログラム部品であるイベントを表す「トランジション (transition)」、状態を表す「プレース (place)」という 2 種類のノードを、実行の流れを表す「アーク (arc)」によって接続したネットである。発火時間関数はトランジションに対応付ける。プレースは内部にトークンを持ち、そのトークンの有無により動作の状態を示す。

表 4 時間ペトリネット図の構成要素とその表現  
Table 4 Components of timed Petri net diagram.

component	symbol
active transition	
passive transition	
place	
arc	

トランジションはすべての入力プレースにトークンがあるときに発火可能となり、発火するとすべての入力プレースからトークンを取り除き、すべての出力プレースにトークンを送り出す。これを繰り返すことで状態が変化していく<sup>3)</sup>。トランジションの発火は対応する構造部品の機能が実行されることを意味する。

本研究ではトランジションを能動的トランジション (active transition)、受動的トランジション (passive transition) に分ける。能動的トランジションは発火可能になるとすぐに発火 (処理) を実行する。受動的トランジションはボタンを押すなどの外部トリガにより発火する。受動的トランジションは能動的トランジションよりも高い優先度で実行される。同じ優先度のトランジションでは、対応する構造部品で実行条件が真であるものが実行される。

表 4 に時間ペトリネット図の構成要素の図形を示

表5 プログラム部品間の対応

Table 5 Correspondence among program components.

screen program	structure program	behavior program
screen object	actor	transition
—	process	transition
—	data store	—
—	data flow	—
—	—	place
—	—	arc

す。発火時間関数  $t$  はトランジションの下部に表示する。プレースの持つトークン数  $n$  はプレース内部に数値で表示する。アークの多重度  $w$  は矢印上に表示する。ただし、多重度が 1 のときは表示しない。

動作の記述に時間ペトリネット図を活用することで動作の並列制御プログラミングを図式で視覚的に行うことが可能になる。また、処理ごとに時間制約が設定できる。また、トランジションとプレースがそれぞれイベントと状態を表すことができるため、イベント駆動型のプログラミングが可能になる。さらに、プレース中のトークンを移動させていくシミュレーションにより、実行の流れ、状態の変化の理解を容易にすることができる。可達性、デッドロックの検出<sup>5)</sup>、実行時間の解析<sup>6)</sup> などペトリネットの解析技術を利用することができる。

### 3.5 プログラム間の対応

各図式プログラムをソフトウェアの 1 つの側面としたビジュアルプログラミングシステムを構築するために、各プログラムの部品間に対応を持たせる。表 5 に部品間の対応関係を示す。プログラム作成時にある部品を配置すると、それぞれのプログラムウインドウ内に対応する部品が配置される。

### 3.6 プログラムの動作

以上に述べた 3 つの図式は「実行エンジン (exec. engine)」によって解釈・実行される。解釈・実行のアルゴリズムを以下に示す。

構造プログラム部品であるデータフロー図のノードが処理の実体を持っており、トランジションの発火がその実行タイミングを示す。生成されるソフトウェアは並列制御を実現するためにトークンの管理を行う 1 つのメインスレッドと、実際に処理を行う処理スレッドをトランジション数だけ持つ。

データフロー図のノードを  $n_i$ 、時間ペトリネット図のトランジションを  $t_i$ 、処理スレッドを  $th_i$ 、処理関数を  $f_i$  で表す。添字が同じものは対応している。

#### ● メインスレッド

メインスレッドは時間ペトリネット図の解釈・実行を行う。メインスレッドの動作アルゴリズムを以下に

示す。

《《メインスレッドの動作アルゴリズム》》

以下を繰り返す。

- 1) 能動的トランジション  $t_i$  が発火可能かつ実行条件式を満たすとき
  - 1-1)  $t_i$  の入力プレースからトークンを取り除く。
  - 1-2) 処理スレッド  $th_i$  を起動する。
- 2) 処理スレッド  $th_i$  から終了信号が送られたとき
  - 2-1) トランジション  $t_i$  の出力プレースにトークンを置く。
- 3) 受動的トランジション  $t_i$  に対応する処理スレッド  $th_i$  から実行信号が送られたとき
  - 3-1) トランジション  $t_i$  の入力プレースからトークンを取り除く。
  - 3-2) トランジション  $t_i$  の出力プレースにトークンを置く。

#### ● 処理スレッド

処理スレッドの動作アルゴリズムを以下に示す。

《《処理スレッド  $th_i$  の動作アルゴリズム》》

// トランジション  $t_i$  が能動的トランジションに分類されるとき

- 1) ノード  $n_i$  の入力データを取得する。
- 2) 処理関数  $f_i$  を実行する。
- 3) ノード  $n_i$  の出力データを更新する。
- 4) 設定時間の経過を待つ。
- 5) メインスレッドに終了信号を送る。
- 6) 停止する。

// トランジション  $t_i$  が受動的トランジションに分類されるとき

以下を繰り返す。

- 1) ノード  $n_i$  の入力データを取得する。
- 2) 処理関数  $f_i$  が実行されるのを待つ。
- 3) ノード  $n_i$  の出力データを更新する。
- 4) メインスレッドに実行信号を送る。

## 4. ビジュアルプログラミングシステムの実現

### 4.1 システム構成

システム構成図を図 1 に示す。システムはビジュアルエディタ ( Visual Editor ), トランスレータ ( Translator ), ライブラリ ( Library ) で構成される。

ビジュアルエディタで 3 つの図式を作成し、作成した図式をトランスレータが C++ のソースコードに変換する。ソースコードはコンパイラで実行コードに変換される。現行のシステムで利用可能な部品の一覧を付録 A.1 に示す。部品は汎用性の高い基本部品と利用目的に合った応用部品に大別できる。応用部品の例

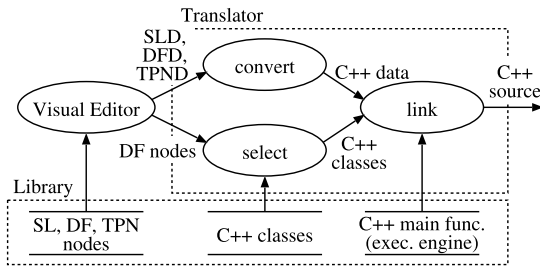


図 1 システム構成図

Fig. 1 System configuration.

として、5章で述べるロボットアームの制御に使われる部品を示す。なお、ソースコードへの変換に必要なビジュアルプログラムデータの構造を A.2 に、C++ クラスライブラリの形式を A.3 に示す。部品の拡張は、C++ クラスライブラリを修正・作成することで行う。

#### 4.2 ビジュアルエディタ

ビジュアルエディタにおいて画面プログラム、構造プログラム、動作プログラムをプログラミングし、後述する各図式のプログラムデータ (SLD, DFD, TPN D) を出力する。ビジュアルエディタの画面を図 2 に示す。ビジュアルエディタは画面プログラムウィンドウ (図 2(A)), 構造プログラムウィンドウ (図 2(B)), 動作プログラムウィンドウ (図 2(C)), 部品ウィンドウ (図 2(D)) を持つ。

ビジュアルエディタでのプログラム作成手順を以下に示す (図 3 参照)。

- (a) プログラム部品を選ぶ。図では部品ウィンドウのコンボボックスから部品「ボタン」を選んでいる。
- (b) 部品の配置をする。部品ウィンドウの「配置」ボタンを押すことによって、選んだ部品が各プログラムウィンドウにハイライト表示される。これらの部品はドラッグ・アンド・ドロップ操作でウィンドウ内の任意の位置に移動できる。
- (c) 構造プログラムの結線を行う。結線の始点、終点となる部品を指定すると結線される。
- (d) 動作プログラムの結線を行う。まず、(a), (b) の操作でプレースを配置する。次にプレース (トランジション) とトランジション (プレース) を指定すると結線される。

なお、(a)–(d) は順次操作に限らず、適宜に操作してもよい。

本システムは 1 つのソフトウェアを 3 つの図式を用いてプログラミングするため、対応するオブジェクトのハイライト表示により、対応部品の把握を容易にしている。また、プログラムのサポートのため、文字と

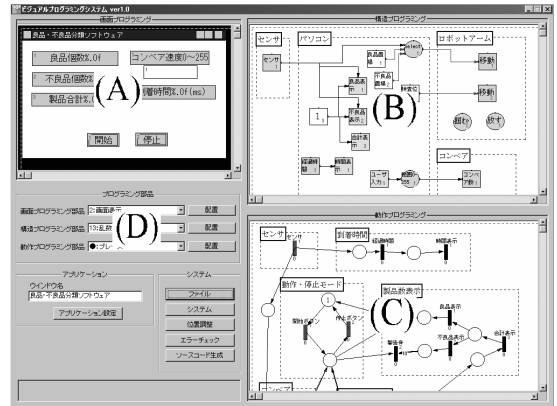


図 2 ビジュアルエディタの画面

Fig. 2 Visual Editor window.

枠によるコメントの記入が可能である。さらに部品ごとに実行内容のヘルプ情報を表示する。プログラムの誤り検出機能として、エッジの未結線、パラメータの未設定、無駄なオブジェクト、複数の選択可能性などを検出することができる。

#### 4.3 ライブラリ

ライブラリはノードデータ (SLnode, DFnode, TPNnode), C++ クラス (C++ classes), C++ メイン関数 (C++ main func.) に分けられる。

##### (1) ノードデータ

ビジュアルエディタで利用するノードの情報を持つ。すべてのノードについて、処理名、処理 ID、入出力端子数、パラメータ数、ヘルプなどを持つ。

##### (2) C++ クラス

データフロー図のすべての部品に対し、C++ 形式のクラスソースコードを持つ。詳細は A.3 を参照。

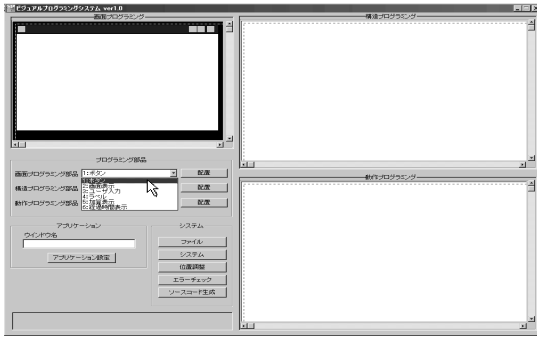
##### (3) C++ メイン関数

C++ ソースコードのメイン関数を持つ。3.6 節で述べた実行エンジンの C++ ソースコードを含む。

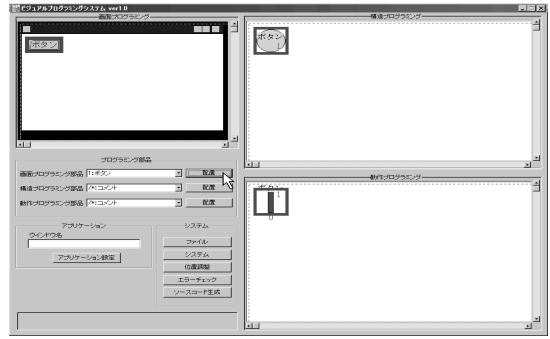
#### 4.4 トランスレータ

トランスレータはビジュアルプログラムデータを C++ のソースコードに変換する。変換の手順を以下に示す。

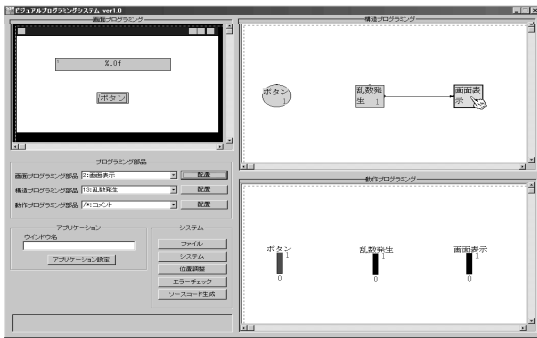
- 1) ビジュアルエディタで使用されたノードの処理 ID を基に部品クラスライブラリから C++ クラスのソースコードを選択する (select)。
- 2) ビジュアルプログラムデータを C++ ソースコードのデータ宣言形式に変換する (convert)。
- 3) convert, select の出力とメイン関数を結合して C++ ソースコードを完成する (link)。



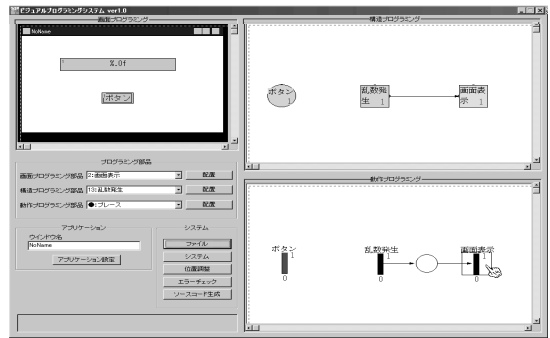
(a) プログラム部品の選択



(b) 部品の配置



(c) 構造プログラムの結線



(d) 動作プログラムの結線

図 3 ビジュアルエディタの操作例

Fig. 3 Example of programming by Visual Editor.

### 5. プログラム例と検討

本システムを用いて開発した 3 つのソフトウェアの事例を紹介し、これらを用いた評価結果を述べる．

S1：良品・不良品分類システムの制御用ソフトウェア（データフロー図のノード数 23）

S2：デジタル入出力ボードを使用してデータ入出力を行うソフトウェア（データフロー図のノード数 12）

S3：別のパソコンからの入力でロボットアームを制御し、状態を監視するソフトウェア（データフロー図のノード数 19）

#### 5.1 プログラム例

ソフトウェア S1 を例にとり、説明を行う．

##### (1) 仕様

良品・不良品分類システム（図 4）の制御用ソフトウェア S1 の仕様を以下に示す．

- (a) 実行ボタンでシステムの動作を開始する．停止ボタンを押すと初期状態で停止する．初期状態はシステム動作停止、コンベア停止、ロボットアームは検査位置で待機する．

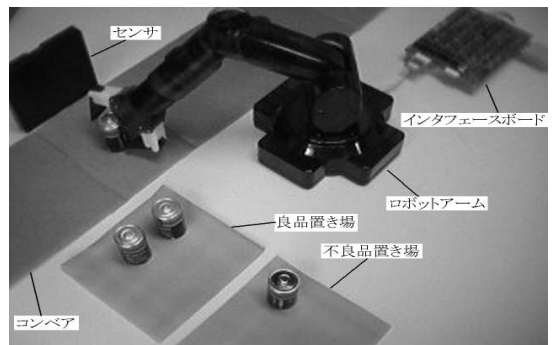


図 4 良品・不良品分類システム

Fig. 4 Selection system to be programmed.

- (b) コンベアを流れてくる製品が検査位置になるとコンベアが止まり、センサが良品か不良品かを判別する．
- (c) 製品到着間隔を表示する．
- (d) ロボットアームが製品を掴むと、コンベアが動く．コンベア速度は 0～255 の範囲をパソコンで設定できる．
- (e) ロボットアームがそれぞれの置場に運ぶ．
- (f) 良品・不良品の個数と総数を表示する．良品

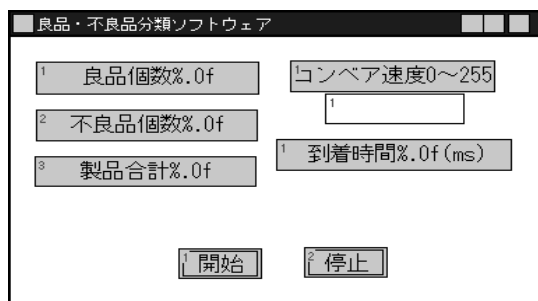


図 5 画面プログラム  
Fig.5 Screen program.

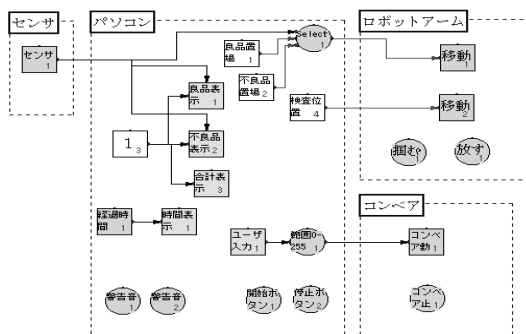


図 6 構造プログラム  
Fig.6 Structure program.

数が 100 になると警告音を鳴らし停止する。不良品数が 10 になると警告音を鳴らし停止し、動作再開を不可能にする。

(g) ロボットアームが検査位置へ移動する。

(2) 画面プログラム

画面部品を配置してユーザインタフェースを作成する。ソフトウェア S1 の画面プログラムを図 5 に示す。製品個数を表示するスタティックテキスト、コンベアの速度を設定するエディットボックス、システム状態を切り替えるボタンを配置して図のようにレイアウトを行う。

(3) 構造プログラム

機能を持った構造部品を配置し、データの流れを規定する。ソフトウェア S1 の構造プログラムを図 6 に示す。

処理構造は、機器ごとにグループ化するのが効果的である。このプログラムでは「センサ」、「パソコン」、「ロボットアーム」、「コンベア」という 4 つのシステム構成機器の処理をコメントでグループ化している。このプログラムでは「センサ」から「パソコン」へ判別データ、「パソコン」から「ロボットアーム」へ移動先データ、「パソコン」から「コンベア」へ速度データが流れていることが分かる。

「Select」(図 6 上中央)は「センサ」の判別値が真か偽かによって「良品置場」か「不良品置場」のデータを選択し、「移動(1)」の行き先を決定する。「ユーザ入力」は「範囲 0-255」によって 0 から 255 の範囲に切り詰めてコンベア速度とする。

この構造プログラムにより、要件 (1) を満たしたプログラムを開発できることが分かった。

(4) 動作プログラム

動作部品を配置し、時間制約と実行順序を規定する。ソフトウェア S1 の動作プログラムを図 7 に示す。

このプログラムでは図に示すように単位動作ごとにコメントでグループ化している。

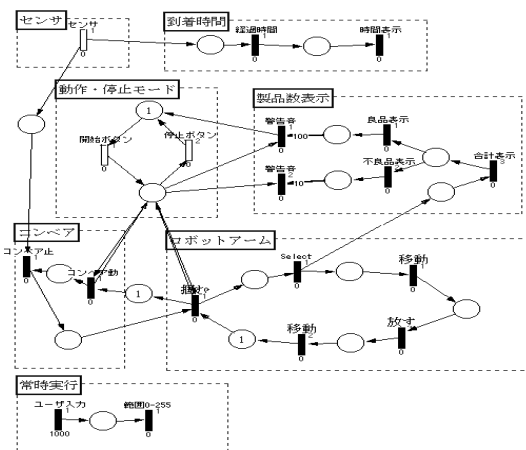


図 7 動作プログラム  
Fig.7 Behavior program.

「動作・停止モード」(図 7 左上)部分によって、動作モード・停止モードの状態切替を表している。下のプレースにトークンがあるときに動作モードであり、「コンベア動」、「掴む」を発火可能にする。並列に動作するロボットアームとコンベアは、ロボットアームの「掴む」で同期をとって動作する。「製品数表示」の(図 7 右上)部分の「良品表示」と「不良品表示」は構造プログラミングの「センサ」の値を利用して選択される。このように動作の選択は時間ベトリネット図によって記述することができる。また、コンベアの速度を 1000 ms ごとに更新できるように「ユーザ入力」(図 7 左下)の発火時間関数を 1000 にしている。

初期状態はトークンの初期配置によって、ロボットアームは検査位置で待機、コンベアを停止、システムを停止モードとしている。

この動作プログラムにより、要件 (2)~(4) を満たしたプログラムが開発できることが分かった。

5.2 評価実験

本システムでの学習・開発時間の調査のため、山口

表 6 評価実験結果  
Table 6 Experiment results.

Group	study time (min)	ex.1 (S2)		ex.2 (S3)	
		development time (min)	right answer (%)	development time (min)	right answer (%)
Group A	56.5	18.0	70.0	18.4	90.0
Group B	59.6	19.4	66.7	19.6	86.7
All	58.1	18.8	68.0	19.0	88.0

大学工学部の卒研究生・大学院生 25 人を対象に評価実験を行った。実験内容を以下に示す。

- (1) 例題をプログラミングしてプログラミング手法とシステム使用方法を学習する。
- (2) 前述の実験問題 S2, S3 をプログラミングして開発時間を調査する。
- (3) 被験者の調査のためのアンケートを行う。

データフロー図は一般的に知られているが、ペトリネット図はそうではない。そこで、どの程度のペトリネットを学習すれば、どの程度のソフトウェアを開発できるかを調べるため、ペトリネット既知のグループ (Group A) 10 人とペトリネット未知のグループ (Group B) 15 人に分けて結果を評価する。評価実験の結果を表 6 に示す。示す項目は、平均学習時間、2 つの実験問題 (S2, S3) それぞれの平均開発時間、正答率 (誤り訂正前の正答人数/人数  $\times 100(\%)$ ) である。

実験問題 S2, S3 とともに要件 (1)~(4) を含むソフトウェアである。結果、1 時間程度の学習を行えば、実験問題程度の規模のプログラムを開発できることが分かった。開発に要した時間は 20 分未満であった。また、プログラムの誤りは平均 2 分で訂正できた。

また、アンケートによる結果を示す。

- 文字型言語の知識がなくとも同様に開発できた 24 人
- データフロー図が理解しにくい 3 人
- ペトリネット図が理解しにくい 3 人
- 図式間の対応が理解しにくい 5 人

データフロー図が理解しにくいという点は、システムで異なる種類の端子接続を行えなくするという点で解決できる問題であった。図式間の対応に関してはさらにサポート手法を考慮する必要がある。

総合的に分かりやすいという意見が多く得られ、インタフェースボードを使用するソフトウェアの開発を専門としないプログラマでも短時間でこのようなソフトウェアの開発を行えることが分かった。

### 5.3 関連研究

まずインタフェースボードを使用するソフトウェアの開発に利用できるビジュアルプログラミングシステ

ムを紹介し、本システムと比較する。National Instruments 社の LabVIEW<sup>1)</sup> と DASYTEC 社の DASY-Lab<sup>7)</sup> はデータフローをベースとしたシステムである。その特徴は機能を表すアイコンを配置し、それらを互いに結線するという方法によって、プログラマが直観的にソフトウェアを開発できる点にある。また Auguston<sup>8)</sup> は分岐や繰返しを表現できるようにデータフローを拡張する研究を行っている。一方、Uchihira らの MENDELS ZONE<sup>9)</sup> や Miyagi らの MFG/PFS<sup>10)</sup>、村田らの C-net<sup>11)</sup>、長尾らの K-net<sup>12)</sup> はペトリネットをベースとしたシステムである。その特徴はソフトウェアの挙動を詳細に記述できる点と、それを数学的に解析できる点にある。以上に紹介したシステムはデータフローかペトリネットのいずれか一方をベースとしている。これらのシステムとは対照的に、本システムはデータフローとペトリネットの両方を併用するシステムである。この併用が、ソフトウェアの構造を直観的に記述し、その動作を厳密に記述できることに加え、ソフトウェアの構造に基づいて動作の実行条件を定義できるといった相乗的な記述を可能にしている。

次にソフトウェアの開発に複数のビューを用いるビジュアルプログラミングシステムを紹介し、本システムと比較する。Koike らの HOLON/VP<sup>13)</sup> はデータフローをベースとしたシステムであり、複雑な図のプログラムを複数のビューを用いて簡単に見せる機能 (MVO; Multiple View Object Representation) を持つ。MVO はプログラマがプログラミングする場合だけでなく、第三者が出来上がったプログラムを読む場合にも役立つ。Giese らの OCoNs<sup>14)</sup> はペトリネットをベースとしたシステムであり、システムの動作を設計するために 3 つのビューを提供している。3 つのビューはシームレスにつながっており、分散システムの動作を総合的に理解するのに役立つ。これらのシステムとは異なり、本システムは複数の図式をベースとしたシステムであり、ソフトウェアの 3 つの側面であるユーザインタフェース、処理構造、動作をプログラミングするために複数のビューを提供している。その特徴は各ビューごとに適切な図式を採用することに



よって、各ビューの表現能力と可読性を高めている点にある。さらに各図式間のノードの関係は、ある図式のノードをクリックすると、他の図式の対応するノードがハイライト表示される機能によって容易に知ることができる。

## 6. おわりに

インタフェースボードを使用するソフトウェアのためにソフトウェアの側面ごとに図式を活用してプログラミングするビジュアルプログラミング言語を提案した。本言語では、ユーザインタフェースを画面レイアウト図、処理構造をデータフロー図、動作を時間ベトリネット図でプログラミングする。提案した言語を実装し、ビジュアルプログラミングシステムを開発した。例題のプログラミングと評価実験より本システムでインタフェースボードを使用するソフトウェアの要件を満たしたソフトウェアの開発が行え、このようなソフトウェアの開発を専門としないプログラマでも短時間で開発が行え、システムが有効であることが分かった。

今後の課題としては、プログラミングサポートのため、動作シミュレーション機能の実現、また、可達性、デッドロックの検出、時間解析などベトリネットの解析技術を用いた評価機能の実現などがある。

謝辞 有益なご助言をくださいました(株)インタフェースの三輪信弘氏、森弘樹氏に感謝します。また、実験の実施にご協力くださいました山口大学工学部の石井孝典氏、白石憲一氏に感謝します。

## 参 考 文 献

- 1) 井上泰典：LabVIEW グラフィカルプログラミング，森北出版 (1998)。
- 2) 内平直志，川田秀司：制御用プログラムの試験，情報処理，Vol.39, No.1, pp.19-25 (1998)。
- 3) Murata, T.: Petri Nets: Properties, Analysis and Applications, *Proc. IEEE*, Vol.77, No.4, pp.541-580 (1989)。
- 4) 伊藤 潔，杵嶋修三：リアルタイムシステムにおけるネット指向開発技術の適用，情報処理，Vol.34, No.6, pp.747-760 (1993)。
- 5) 白石憲一，山口真悟，田中 稔，葛 崎偉：インタフェースボードの割込みを処理するためのシステムのモデル化とデッドロックの検出，2001年IEEE広島学生シンポジウム論文集，pp.75-78 (2001)。
- 6) Kavi, K.M., Buckles, B.P. and Bhat, U.N.: Isomorphisms Between Petri Nets and Dataflow Graphs, *IEEE Trans. Softw. Eng.*, Vol.13, No.10, pp.1127-1134 (1987)。
- 7) DASYTEC Inc.: DASYLab.

<http://www.dasytec.com/>

- 8) Auguston, M. and Delgado, A.: Iterative Constructs in the Visual Data Flow Language, *IEEE Symposium on Visual Languages (VL'97)*, pp.154-161 (1997)。
- 9) Uchihira, N., Arami, M. and Honiden, S.: A Petri-net-based Programming Environment and its Design Methodology for Cooperating Discrete Event Systems, *IEICE Trans. Fundamentals*, Vol.E75-A, No.10, pp.1335-1347 (1992)。
- 10) Miyagi, P.E., Hasegawa, K. and Takahashi, K.: A Programming Language for Discrete Event Production Systems Based on Production Flow Schema and Mark Flow Graph, 計測自動制御学会論文誌，Vol.24, No.2, pp.183-190 (1988)。
- 11) 村田智洋，薦田憲久，解良和郎：色付きベトリネットに基づくリアルタイム情報処理用ソフトウェア，情報処理学会論文誌，Vol.29, No.12, pp.1129-1140 (1988)。
- 12) 長尾陽一，熊谷貞俊：ベトリネットのFA制御への応用，情報処理，Vol.34, No.6, pp.761-769 (1993)。
- 13) Koike, Y., Maeda, Y. and Koseki, Y.: Improving Readability of Iconic Programs with Multiple View Object Representation, *11th Intl. IEEE Symposium on Visual Languages (VL'95)*, pp.37-44 (1995)。
- 14) Giese, H., Graf, J. and Wirtz, G.: Seamless Visual Object-Oriented Behavior Modeling for Distributed Software Systems, *IEEE Symposium on Visual Languages (VL'99)*, pp.156-163 (1999)。

## 付 録

### A.1 部品一覧

現行のシステムで実装している部品の一覧を表7に示す。ただし、 $InX$  は  $X$  番目の入力端子へのデータ、 $PX$  は  $X$  番目のパラメータ設定値である。パラメータの値は部品を使用するときにダイアログボックスを介して設定される。

ノードデータライブラリと C++クラスライブラリを作成することで部品を追加することができる。

### A.2 プログラムデータ

ソースコードへの変換に必要なビジュアルプログラムデータを以下に示す。画面オブジェクト数を  $|O|$ 、データフロー図のノード数を  $|N|$ 、トランジション数を  $|T|$ 、プレース数を  $|P|$  とする。

// プログラムデータ (SLD,DFD,TPND) の定義

表 7 プログラム部品の一覧  
Table 7 List of programming components.

(a) 基本プログラム部品

処理名	入力	出力	パラメータ数	説明
数値出力	0	1	1-256	数値を出力する。配列の要素数はパラメータ数で決まる。
加算	2	1	0	$In1+In2$ を出力する。
減算	2	1	0	$In1-In2$ を出力する。
乗算	2	1	0	$In1 \times In2$ を出力する。
除算	2	1	0	$In1/In2$ を出力する。 $In2$ が 0 のときは 0 を出力する。
Select	3	1	0	$In1$ が真 (非 0) なら $In2$ を出力, $In1$ が偽 (0) なら $In3$ を出力する。
比較 (=)	2	1	0	$In1=In2$ なら 1 を出力。それ以外なら 0 を出力する。
比較 ( $\neq$ )	2	1	0	$In1 \neq In2$ なら 1 を出力。それ以外なら 0 を出力する。
比較 (>)	2	1	0	$In1 > In2$ なら 1 を出力。それ以外なら 0 を出力する。
比較 (<)	2	1	0	$In1 < In2$ なら 1 を出力。それ以外なら 0 を出力する。
比較 ( $\geq$ )	2	1	0	$In1 \geq In2$ なら 1 を出力。それ以外なら 0 を出力する。
比較 ( $\leq$ )	2	1	0	$In1 \leq In2$ なら 1 を出力。それ以外なら 0 を出力する。
ボタン	0	0	0	ボタンの入力待。
画面表示	1	1	0	対応するスタティックテキストに $In1$ を表示する。
ユーザ入力	0	1	0	対応するエディットボックスへの入力値を出力する。
乱数発生	0	1	2	$P1-P2$ の乱数発生する。
インクリメント	1	1	0	$In1$ に 1 加算して出力する。
デクリメント	1	1	0	$In1$ を 1 減算して出力する。
汎用演算	1-3	1	1	演算式の結果を出力する。演算式はパラメータとして設定する。
時間出力	0	1	0	時間を出力する。
データストア	0-3	0-3	1-256	データを保持する。初期値はパラメータで設定する。
データストア書き込み	1	1	0	出力先のデータストアにデータを書き込む。
変換 (2 進 → 10 進)	1	1	0	入力配列を 10 進数に変換して出力する。
変換 (10 進 → 2 進)	1	1	0	入力値を 2 進数に変換して配列として出力する。
終了	0	0	0	アプリケーションを終了する。
ビープ音	0	0	0	ビープ音を鳴らす。
論理和	2	1	0	$In1$ と $In2$ の論理和を出力する。非 0 を真とする。
論理積	2	1	0	$In1$ と $In2$ の論理積を出力する。非 0 を真とする。
論理否定	1	1	0	$In1$ の論理否定を出力する。非 0 を真とする。
排他的論理和	2	1	0	$In1$ と $In2$ の排他的論理和を出力する。非 0 を真とする。
スリーブ	1	0	0	$In1$ ミリ秒間待機する。
上下フィルタ	1	1	2	$In1 < P1$ なら $P1$ , $In1 > P2$ なら $P2$ を出力する。それ以外は $In1$ をそのまま出力する。

(b) 応用プログラム部品

処理名	入力	出力	パラメータ数	説明
加算表示	1	0	0	$In1$ に入力された値を現在値に加算して表示する。
経過後時間表示	0	0	0	アプリケーションが実行されてからの経過時間をスタティックテキストに表示する。
処理なし	1-3	1-3	0	処理をしない。テスト用に開発。
常時実行	0	0	0	発火すると処理を実行し続ける。テスト用に開発。
デジタルボード初期化	0	0	0	デジタルボードを 0 で初期化する。
デジタル入力 2 進 OUT	0	1	2	デジタルボードから端子 $P1-P2$ のデータを入力し, 2 進数で出力する。
デジタル出力 2 進 IN	1	0	2	2 進数で入力したデータをデジタルボードの端子 $P1-P2$ に出力する。
デジタルトリガ *	0	0-1	2	デジタルボードの端子 $P1-P2$ のどれかが変更されるのを待つ。変更された端子番号を出力する。
デジタルトリガ 0	0	0-1	2	デジタルボードの端子 $P1-P2$ のどれかが 0 になるのを待つ。変更された端子番号を出力する。
デジタルトリガ 1	0	0-1	2	デジタルボードの端子 $P1-P2$ のどれかが 1 になるのを待つ。変更された端子番号を出力する。
デジタル入力 10 進 OUT	0	1	2	デジタルボードから端子 $P1-P2$ のデータを入力し, 10 進数で出力する。
デジタル出力 10 進 IN	1	0	2	10 進数で入力したデータをデジタルボードの端子 $P1-P2$ に出力する。
ロボットアーム移動	1	0	0	ロボットアームを位置 ( $In1$ の要素 1, $In1$ の要素 2) に移動する。
ロボットアーム移動 P	0	0	2	ロボットアームを位置 ( $P1, P2$ ) に移動する。
ロボットアーム移動 2IN	2	0	0	ロボットアームを位置 ( $In1, In2$ ) に移動する。
ロボットアーム掴む	0	0	0	ロボットアームのハンド部分を閉じる。
ロボットアーム放す	0	0	0	ロボットアームのハンド部分を開く。
ロボットアーム上げる	2	0	0	ロボットアームのハンド部分を上げる。
ロボットアーム下げる	2	0	0	ロボットアームのハンド部分を下げる。
ロボットアーム回転	0	0	1	ロボットアームを角度 $P1$ だけ回転させる。
判別	0	1	0	センサの値を読む。

```

struct VisualProgramming
{
    ScreenProgram prog1; //画面プログラム
    StructureProgram prog2; //構造プログラム
    BehaviorProgram prog3; //動作プログラム
    int corON[|O|][|N|]; //画面オブジェクト × ノード対応
    int corNT[|N|][|T|]; //ノード × トランジション対応
};

// 画面プログラムデータの定義
struct ScreenProgram
{
    ScreenObject obj[|O|]; //画面オブジェクト
    int winsizeW, winsizeH; //ウィンドウサイズ
};

// 画面オブジェクトデータの定義
struct ScreenObject
{
    int objpointX, objpointY; //画面オブジェクト座標
    int objsizeW, objsizeH; //画面オブジェクトサイズ
    char objlabel[ ]; //画面オブジェクトラベル
};

}; // 構造プログラムデータの定義
struct StructureProgram
{
    StructureNode node[|N|]; //ノード
    int dataflow[|N|][|N|]; //データフロー接続行列
};

// ノードデータ ( DF nodes ) の定義
struct StructureNode
{
    int proID; //処理 ID
    int par[|N|][ ]; //処理パラメータ
    char exe[ ]; //実行可能条件式
};

// 動作プログラムデータの定義
struct BehaviorProgram
{
    Transition tra[|T|]; //トランジション
    Place pl[|P|]; //プレース
};
    
```

```

int arcTP[[T]][[P]]; //T × P 接続行列
int arcPT[[P]][[T]]; //P × T 接続行列
};
// トランジションデータの定義
struct Transition
{
    int time; //発火時間関数
};
// プレースデータの定義
struct Place
{
    int token; //初期トークン数
};

```

### A.3 C++クラスライブラリ

C++クラスライブラリの構成を示す。クラス定義やスレッド制御関数はすべての部品に共通のものがトランスレータによって自動的に組み込まれる。クラスの処理関数は部品によって異なり以下の形式で記述される。

```

void クラス : : 処理関数 ()
{
    //初期化
    { 変数宣言 }
    スレッド停止関数 ();

    //メインループ
    while(1)
    {
        { 処理本体 }
        スレッド停止関数 ();
    }
}

```

新しい部品の作成は、{ 変数宣言 }、{ 処理本体 } の部分を記述することで行う。

(平成 13 年 3 月 12 日受付)

(平成 13 年 6 月 21 日採録)



岡田 栄治 (正会員)

1999 年 3 月山口大学工学部電気電子工学科卒業。2001 年 3 月同大学大学院理工学研究科知能情報システム工学専攻博士前期課程修了。同年松下電工ソフトウェア株式会社に入社。



山口 真悟 (正会員)

1992 年 3 月山口大学工学部電子工学科卒業。1994 年 3 月同大学大学院工学研究科博士前期課程修了。1997 年 5 月同大学大学院工学研究科博士後期課程単位取得退学。同年山口大学工学部助手となり、現在に至る。ビジュアルプログラミング、グループウェア、ペトリネット等の研究に従事。電子情報通信学会、IEEE 各会員。



田中 稔 (正会員)

1969 年 3 月大阪大学基礎工学部電気工学科卒業。同年 4 月(株)北辰電機製作所勤務。1976 年 3 月大阪大学基礎工学研究科物理系専攻博士課程修了。日本学術振興会奨励研究員、大阪大学基礎工学部助手、広島大学工学部助教授を経て 1991 年 3 月より山口大学教授。この間 1 年間ピッツバーグ大学訪問準教授。工学博士。ビジュアルワークスペース、ユーザ支援システム、CAI システム等の研究に従事。電子情報通信学会、人工知能学会、IEEE、ACM 各会員。